

The SELinux Notebook



The Foundations

0. Notebook Information

0.1 Copyright Information

Copyright © 2009 [Richard Haines](#).

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled “[GNUFree Documentation License](#)”.

The scripts and source code in this Notebook are covered by the GNU General Public License. The scripts and code are free source: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or any later version.

These are distributed in the hope that they will be useful in researching SELinux, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with scripts and source code. If not, see <<http://www.gnu.org/licenses/>>.

0.2 Revision History

Version	Date	Description of Change	Changed By
1.0	20 th Nov '09	First release.	Richard Haines

0.3 Acknowledgements

Logo designed by [Máirín Duffy](#)

0.4 Abbreviations

Term	Definition
apol	Policy analysis tool
AV	Access Vector
AVC	Access Vector Cache
BLP	Bell-La Padula
CC	Common Criteria
CMW	Compartmented Mode Workstation
DAC	Discretionary Access Control

Term	Definition
F-10	Fedora 10
FLASK	Flux Advanced Security Kernel - A security-enhanced version of the Fluke kernel and OS developed by the Utah Flux team and the US Department of Defence.
Fluke	Flux μ -kernel Environment - A specification and implementation of a micro kernel and Operating System architecture.
Flux	The Flux Research Group (http://www.cs.utah.edu/flux/)
ID	Identification
LSM	Linux Security Module
LSP	Labeled Security Protection Profile
MAC	Mandatory Access Control
MCS	Multi-Category Security
MLS	Multi-Level Security
NSA	National Security Agency
OM	Object Manager
PAM	Pluggable Authentication Module
RBAC	Role-based Access Control
rpm	Red Hat Package Manager
SELinux	Security-Enhanced Linux
SID	Security Identifier
SL	Security Level
SLIDE	SELinux Integrated Development Environment
SMACK	Simplified Mandatory Access Control Kernel
SUID	Super-user Identifier
TE	Type Enforcement
UID	User Identifier

0.5 Index

0. NOTEBOOK INFORMATION.....	2
0.1 COPYRIGHT INFORMATION.....	2
0.2 REVISION HISTORY.....	2
0.3 ACKNOWLEDGEMENTS.....	2
0.4 ABBREVIATIONS.....	2
0.5 INDEX.....	3
0.5.1 Tables.....	11
0.5.2 Figures.....	12
1. THE SELINUX NOTEBOOK.....	16
1.1 INTRODUCTION.....	16
1.1.1 Sample Policy Source.....	16

1.2 HEALTH WARNING.....	16
1.3 ROOT LOGIN REQUIREMENTS.....	17
1.4 RELEVANT F-10 PACKAGES.....	17
1.5 NOTEBOOK SECTIONS.....	18
2. SELINUX OVERVIEW.....	20
2.1 INTRODUCTION.....	20
2.2 CORE SELINUX COMPONENTS.....	20
2.3 MANDATORY ACCESS CONTROL (MAC).....	23
2.4 TYPE ENFORCEMENT (TE).....	24
2.4.1 Constraints.....	25
2.5 ROLE-BASED ACCESS CONTROL (RBAC).....	26
2.6 SECURITY CONTEXT.....	26
2.7 SUBJECTS.....	28
2.8 OBJECTS.....	29
2.8.1 Object Classes and Permissions.....	29
2.8.2 Allowing a Process Access to an Object.....	30
2.8.3 Labeling Objects.....	31
2.8.3.1 Labeling Extended Attribute Filesystems.....	32
2.8.3.1.1 Copying and Moving Files.....	32
2.8.3.2 Labeling Subjects.....	34
2.8.4 Object Reuse.....	34
2.9 DOMAIN AND OBJECT TRANSITIONS.....	34
2.9.1 Domain Transition.....	35
2.9.1.1 Type Enforcement Rules.....	36
2.9.2 Object Transition.....	38
2.10 MULTI-LEVEL SECURITY AND MULTI-CATEGORY SECURITY.....	39
2.10.1 Security Levels.....	41
2.10.1.1 MLS / MCS Range Format.....	41
2.10.1.2 Translating Levels.....	42
2.10.2 Managing Security Levels via Dominance Rules.....	42
2.10.3 MLS Labeled Network and Database Support.....	44
2.10.4 Common Criteria Certification.....	44
2.11 TYPES OF SELINUX POLICY.....	45
2.11.1 Example Policy.....	45
2.11.2 Reference Policy.....	45
2.11.2.1 Policy Functionality Type.....	46
2.11.3 Custom Policy.....	46
2.11.4 Monolithic Policy.....	47
2.11.5 Loadable Module Policy.....	47
2.11.5.1 Optional Policy.....	47
2.11.6 Conditional Policy.....	47
2.11.7 Binary Policy.....	48
2.11.8 Policy Versions.....	48
2.12 SELINUX PERMISSIVE AND ENFORCING MODES.....	49
2.13 AUDIT LOGS.....	50
2.13.1 SELinux-aware Application Events.....	50
2.13.2 AVC Audit Events.....	52
2.14 POLYINSTANTIATION.....	54
2.14.1 Polyinstantiation support in PAM.....	54

2.14.1.1 namespace.conf Configuration File.....	55
2.14.1.2 Example Configurations.....	56
2.14.2 Polyinstantiated Objects.....	57
2.14.3 Polyinstantiation support in the Reference Policy.....	57
2.15 PAM LOGIN PROCESS.....	57
2.16 LINUX SECURITY MODULE AND SELINUX.....	59
2.16.1 The LSM Module.....	60
2.16.2 The SELinux Module.....	61
2.16.2.1 Fork System Call Walk-through.....	62
2.16.2.2 Process Transition Walk-through.....	65
2.16.2.3 SELinux Filesystem.....	70
2.16.2.4 SELinux Boot Process.....	72
2.17 SELINUX NETWORKING SUPPORT.....	73
2.17.1 compat_net Controls.....	74
2.17.2 SECMARK.....	74
2.17.3 NetLabel - Fallback Peer Labeling.....	76
2.17.4 Labeled IPSec.....	77
2.17.5 NetLabel - CIPSO.....	79
3. SELINUX CONFIGURATION FILES.....	80
3.1 INTRODUCTION.....	80
3.2 GLOBAL CONFIGURATION FILES.....	81
3.2.1 /etc/selinux/config File.....	81
3.2.2 /etc/selinux/semanage.conf File.....	82
3.2.3 /etc/selinux/restorecond.conf File.....	84
3.2.4 /etc/sestatus.conf File.....	84
3.2.5 /etc/security/sepermit.conf File.....	85
3.3 POLICY STORE CONFIGURATION FILES.....	86
3.3.1 modules/ Files.....	86
3.3.2 modules/active/base.pp File.....	86
3.3.3 modules/active/base.linked File.....	87
3.3.4 modules/active/commit_num File.....	87
3.3.5 modules/active/file_contexts.template File.....	87
3.3.6 modules/active/file_contexts File.....	90
3.3.7 modules/active/homedir_template File.....	91
3.3.8 modules/active/file_contexts.homedirs File.....	92
3.3.9 modules/active/netfilter_contexts & netfilter.local File.....	92
3.3.10 modules/active/policy.kern File.....	93
3.3.11 modules/active/seusers.final and seusers Files.....	93
3.3.12 modules/active/users_extra, users_extra.local and users.local Files.....	95
3.3.13 modules/active/booleans.local File.....	97
3.3.14 modules/active/file_contexts.local File.....	97
3.3.15 modules/active/interfaces.local File.....	98
3.3.16 modules/active/nodes.local File.....	98
3.3.17 modules/active/ports.local File.....	98
3.3.18 modules/active/modules Directory Contents.....	98
3.4 POLICY CONFIGURATION FILES.....	99
3.4.1 seusers File.....	99
3.4.2 setrans.conf File.....	100
3.4.3 policy/policy.23 File.....	101

3.4.4 contexts/customizable_types File.....	102
3.4.5 contexts/default_contexts File.....	102
3.4.6 contexts/debus_contexts File.....	104
3.4.7 contexts/default_type File.....	105
3.4.8 contexts/failsafe_context File.....	105
3.4.9 contexts/initrc_context File.....	106
3.4.10 contexts/netfilter_contexts File.....	107
3.4.11 contexts/removable_contexts File.....	107
3.4.12 contexts/securetty_types File.....	108
3.4.13 contexts/userhelper_context File.....	108
3.4.14 contexts/x_contexts File.....	109
3.4.15 contexts/files/file_contexts File.....	110
3.4.16 contexts/files/file_contexts.local File.....	110
3.4.17 contexts/files/file_contexts.homedirs File.....	110
3.4.18 contexts/files/media File.....	111
3.4.19 contexts/users/[seuser_id] File.....	111
4. SELINUX POLICY LANGUAGE.....	113
4.1 INTRODUCTION.....	113
4.2 POLICY STATEMENTS AND RULES.....	113
4.2.1 Policy Source Files.....	113
4.2.2 Conditional, Optional and Require Statement Rules.....	115
4.2.3 MLS Statements and Optional MLS Components.....	115
4.2.4 General Statement Information.....	115
4.2.5 SELinux Identifier Naming Conventions.....	118
4.2.6 Section Contents.....	119
4.3 TYPE ENFORCEMENT AND ATTRIBUTE STATEMENTS.....	120
4.3.1 type Statement.....	120
4.3.2 attribute Statement.....	122
4.3.3 typeattribute Statement.....	122
4.3.4 typealias Statement.....	123
4.4 TYPE ENFORCEMENT RULES.....	124
4.4.1 type_transition Statement.....	125
4.4.2 type_change Statement.....	126
4.4.3 type_member Statement.....	126
4.5 ACCESS VECTOR RULES.....	127
4.5.1 allow Rule.....	128
4.5.2 dontaudit Rule.....	129
4.5.3 auditallow Rule.....	129
4.5.4 neverallow Rule.....	129
4.6 USER STATEMENT.....	130
4.6.1 user Statement.....	130
4.7 ROLE STATEMENT.....	132
4.7.1 role Statement.....	132
4.8 ROLE RULES.....	133
4.8.1 Role allow Rule.....	133
4.8.2 role_transition Rule.....	134
4.8.3 Role dominance Rule.....	135
4.9 CONDITIONAL POLICY STATEMENTS.....	136
4.9.1 bool Statement.....	137

4.9.2 <i>if</i> Statement.....	138
4.10 CONSTRAINT STATEMENTS.....	140
4.10.1 <i>constrain</i> Statement.....	140
4.10.2 <i>validatetrans</i> Statement.....	142
4.11 FILE SYSTEM LABELING STATEMENTS.....	144
4.11.1 <i>fs_use_xattr</i> Statements.....	144
4.11.2 <i>fs_use_task</i> Statement.....	145
4.11.3 <i>fs_use_trans</i> Statement.....	146
4.11.4 <i>genfscon</i> Statements.....	147
4.12 NETWORK LABELING STATEMENTS.....	148
4.12.1 <i>IP Address Formats</i>	149
4.12.1.1 IPv4 Address Format.....	149
4.12.1.2 IPv6 Address Formats.....	149
4.12.2 <i>netifcon</i> Statement.....	149
4.12.3 <i>nodecon</i> Statement.....	150
4.12.4 <i>portcon</i> Statement.....	152
4.13 MLS STATEMENTS.....	153
4.13.1 <i>sensitivity</i> Statement.....	154
4.13.2 <i>MLS dominance</i> Statement.....	154
4.13.3 <i>category</i> Statement.....	155
4.13.4 <i>level</i> Statement.....	156
4.13.5 <i>range_transition</i> Statement.....	157
4.13.5.1 <i>MLS range Definition</i>	158
4.13.6 <i>mlsconstrain</i> Statement.....	159
4.13.7 <i>mlsvalidatetrans</i> Statement.....	160
4.14 POLICY SUPPORT STATEMENTS.....	162
4.14.1 <i>module</i> Statement.....	162
4.14.2 <i>require</i> Statement.....	163
4.14.3 <i>optional</i> Statement.....	164
4.14.4 <i>polycycap</i> Statement.....	165
4.14.5 <i>permissive</i> Statement.....	166
4.15 OBJECT CLASS AND PERMISSION STATEMENTS.....	168
4.15.1 <i>Object Classes</i>	168
4.15.2 <i>Permissions</i>	168
4.15.2.1 <i>Defining common Permissions</i>	169
4.16 SECURITY ID (SID) STATEMENT.....	170
4.16.1 <i>sid</i> Statement.....	170
4.16.2 <i>sid context</i> Statement.....	171
5. BUILDING A BASIC POLICY.....	173
5.1 INTRODUCTION.....	173
5.1.1 <i>Overall Objectives</i>	173
5.1.2 <i>Build Requirements</i>	173
5.1.3 <i>The Test Policies</i>	173
5.2 BUILDING THE POLICY SOURCE FILES.....	174
5.2.1 <i>Policy Source Files</i>	176
5.2.1.1 <i>Problem Resolution</i>	178
5.2.1.2 <i>Monolithic and Base Policy Source File</i>	178
5.2.1.3 <i>file_contexts</i> File.....	183
5.2.1.4 <i>default_contexts</i> File.....	183

5.2.1.5 seusers File	184
5.2.1.6 dbus_contexts File	184
5.3 BUILDING THE MONOLITHIC POLICY.....	184
5.3.1 <i>Checking the Build</i>	186
5.4 BUILDING THE BASE POLICY MODULE.....	186
5.4.1 <i>Checking the Base Policy Build</i>	188
5.5 BUILDING THE LOADABLE MODULES.....	189
5.5.1 <i>Overview of modules</i>	189
5.6 BUILDING THE SECMARK TEST LOADABLE MODULE.....	190
5.6.1 <i>Testing the Module</i>	204
5.6.1.1 <i>Running the Tests</i>	204
5.6.2 <i>Points to Note</i>	207
5.6.2.1 <i>Importance of Loading the iptables</i>	207
5.6.2.2 <i>Running tests out of sequence</i>	208
5.7 BUILDING THE NETLABEL LOADABLE MODULE.....	208
5.8 BUILDING THE MESSAGE FILTER SERVICE.....	211
5.8.1 <i>Internal Gateway Loadable Policy Module</i>	211
5.8.2 <i>File Move Application</i>	214
5.8.3 <i>File Mover Loadable Policy Module</i>	216
5.8.4 <i>Testing the Message Filter Build</i>	220
6. POLICY INVESTIGATION TOOLS.....	222
6.1 INTRODUCTION.....	222
6.2 USING AUDIT2ALLOW AND AUDIT2WHY	222
6.3 USING SEAUDIT AND SETROUBLESHOOT	223
6.4 USING SEDIFFX	223
6.5 USING SECHECKER	224
6.5.1 <i>Testing the Policy</i>	225
6.5.2 <i>The Results</i>	226
6.6 USING APOL	235
6.6.1 <i>General Information</i>	235
6.6.2 <i>Type Enforcement Rules</i>	237
6.6.3 <i>Direct Relabel</i>	238
6.6.3.1 <i>apol Direct Relabel Analysis</i>	238
6.6.4 <i>Transitive Information Flows</i>	239
6.6.4.1 <i>apol Transitive Information Flows Analysis</i>	240
7. THE REFERENCE POLICY.....	242
7.1 INTRODUCTION.....	242
7.1.1 <i>Notebook Reference Policy Information</i>	242
7.2 REFERENCE POLICY OVERVIEW.....	243
7.2.1 <i>Distributing Policies</i>	243
7.2.2 <i>Policy Type Functionality</i>	244
7.2.3 <i>Reference Policy Module Files</i>	244
7.2.4 <i>Reference Policy Documentation</i>	247
7.3 REFERENCE POLICY SOURCE.....	248
7.3.1 <i>Source Layout</i>	248
7.3.2 <i>Reference Policy Files and Directories</i>	251
7.3.3 <i>Source Configuration Files</i>	253
7.3.3.1 <i>Reference Policy Build Options - build.conf</i>	253

7.3.3.2 Reference Policy Build Options – policy/modules.conf.....	255
7.3.3.2.1 Building the modules.conf File.....	257
7.3.4 Source Installation and Build Make Options.....	258
7.3.5 Booleans, Global Booleans and Tunable Booleans.....	259
7.3.6 Modular Policy Build Structure.....	260
7.3.7 Creating Additional Layers.....	262
7.4 INSTALLING AND BUILDING THE REFERENCE POLICY SOURCE.....	262
7.4.1 Installation and Configuration.....	263
7.4.2 Building the targeted Policy Type.....	264
7.4.3 Checking the Build.....	265
7.4.4 Running with the new Policy.....	266
7.5 REFERENCE POLICY HEADERS.....	266
7.5.1 Building and Installing the Header Files.....	267
7.5.2 Using the Header Files.....	268
7.5.3 Using F-10 Supplied Headers.....	269
7.6 REFERENCE POLICY MACROS.....	269
7.6.1 Loadable Policy Macros.....	271
7.6.1.1 policy_module Macro.....	271
7.6.1.2 gen_require Macro.....	272
7.6.1.3 optional_policy Macro.....	273
7.6.1.4 gen_tunable Macro.....	274
7.6.1.5 tunable_policy Macro.....	275
7.6.1.6 interface Macro.....	276
7.6.1.7 template Macro.....	278
7.6.2 Miscellaneous Macros.....	280
7.6.2.1 gen_context Macro.....	280
7.6.2.2 gen_user Macro.....	282
7.6.2.3 gen_bool Macro.....	283
7.6.3 MLS and MCS Macros.....	284
7.6.3.1 gen_cats Macro.....	284
7.6.3.2 gen_sens Macro.....	285
7.6.3.3 gen_levels Macro.....	286
7.6.3.4 System High/Low Parameters.....	287
7.6.4 <i>ifdef</i> / <i>ifndef</i> Parameters.....	287
7.6.4.1 <i>hide_broken_symptoms</i>	287
7.6.4.2 <i>enable_mls</i> and <i>enable_mcs</i>	288
7.6.4.3 <i>direct_sysadm_daemon</i>	288
7.7 MODULE EXPANSION PROCESS.....	289
7.7.1 Module Expansion.....	290
7.7.2 File Context Expansion.....	298
8. APPENDIX A - OBJECT CLASSES AND PERMISSIONS.....	299
8.1 INTRODUCTION.....	299
8.2 DEFINING OBJECT CLASSES AND PERMISSIONS.....	299
8.3 COMMON PERMISSIONS.....	300
8.3.1 Common File Permissions.....	300
8.3.2 Common Socket Permissions.....	300
8.3.3 Common IPC Permissions.....	301
8.3.4 Common Database Permissions.....	302
8.4 FILE OBJECT CLASSES.....	302

8.5 NETWORK OBJECT CLASSES.....	304
8.5.1 IPSec Network Object Classes.....	306
8.5.2 Netlink Object Classes.....	307
8.5.3 Miscellaneous Network Object Classes.....	309
8.6 IPC OBJECT CLASSES.....	310
8.7 PROCESS OBJECT CLASS.....	311
8.8 SECURITY OBJECT CLASS.....	312
8.9 SYSTEM OPERATION OBJECT CLASS.....	312
8.10 CAPABILITY OBJECT CLASSES.....	312
8.11 X WINDOWS OBJECT CLASSES.....	314
8.12 DATABASE OBJECT CLASSES.....	318
8.13 MISCELLANEOUS OBJECT CLASSES.....	320
9. APPENDIX B – SELINUX COMMANDS.....	322
10. APPENDIX C – API SUMMARY FOR LIBSELINUX.....	323
11. APPENDIX D - REFERENCE POLICY MACROS.....	337
12. APPENDIX E – NETLABEL MODULE SUPPORT FOR NETWORK_PEER_CONTROLS	341
12.1 INTRODUCTION.....	341
12.2 CONFIGURATION.....	341
13. APPENDIX F – LABELED IPSEC MODULE EXAMPLE.....	344
13.1 INTRODUCTION.....	344
13.2 MANUAL IPSEC CONFIGURATION.....	344
13.3 KEY EXCHANGE IPSEC CONFIGURATION.....	348
14. APPENDIX G – IMPLEMENTING A CONSTRAINT	351
14.1 INTRODUCTION.....	351
14.2 CONFIGURATION.....	351
14.3 REFERENCE POLICY CONSTRAINTS INFORMATION.....	352
15. APPENDIX H - BUGS AND FEATURES.....	354
15.1 OPENOFFICE.ORG - WRITER.....	354
15.2 SEMANAGE – NODE CONFIGURATION.....	354
15.3 SEMANAGE - ROLES GET DELETED.....	354
15.4 CHECKMODULE - NEVERALLOW NOT PICKED UP IN MODULES.....	354
15.5 APOL NOT SHOWING ALL SCREEN IN WINDOW.....	354
15.6 RACoon COREDUMPS.....	355
15.7 RED HAT SELINUX CONFIG UTILITY.....	356
16. APPENDIX I - USEFUL INFORMATION.....	357
16.1 BUILDING THE SOURCE RPM.....	357
17. APPENDIX J – REFERENCES.....	359
17.1 DOCUMENT REFERENCES.....	359
17.2 USEFUL WEBSITES.....	359
18. APPENDIX K - GNU FREE DOCUMENTATION LICENSE.....	360

0.5.1 Tables

TABLE 1-1: THE SELINUX NOTEBOOK SECTIONS.....	19
TABLE 2-1: LEVEL, LABEL, CATEGORY OR COMPARTMENT – THIS TABLE SHOWS THE MEANINGS DEPENDING ON THE CONTEXT BEING DISCUSSED.....	41
TABLE 2-2: MLS SECURITY LEVELS – SHOWING THE SCOPE OF A PROCESS RUNNING AT A SECURITY RANGE OF S0 – S3:C1.C5.....	43
TABLE 2-3: POLICY VERSION DESCRIPTIONS	49
TABLE 2-4: AVC AUDIT MESSAGE DESCRIPTION – THE KEYWORDS IN BOLD ARE IN ALL AVC AUDIT MESSAGES, THE OTHERS DEPEND ON THE TYPE OF EVENT BEING AUDITED.....	53
TABLE 2-5: LSM HOOKS - THESE ARE THE KERNEL SERVICES THAT LSM HAS INSERTED SECURITY HOOKS AND STRUCTURES TO ALLOW ACCESS CONTROL TO BE MANAGED BY 3RD PARTY MODULES (SEE ./KERNEL-2.6.27/INCLUDE/LINUX/SECURITY.H).....	60
TABLE 2-6: THE CORE LSM SOURCE MODULES.....	61
TABLE 2-7: THE CORE SELINUX SOURCE MODULES - THE .H FILES AND THOSE IN THE INCLUDE DIRECTORY HAVE A NUMBER OF USEFUL COMMENTS.....	62
TABLE 2-8: THE LSM / SELINUX PROGRAM LOADING HOOKS.....	65
TABLE 2-9: /SELINUX FILE AND DIRECTORY INFORMATION.....	71
TABLE 4-1: BASE AND MODULE POLICY STATEMENTS – A MONOLITHIC SOURCE FILE WOULD CONTAIN THE SAME STATEMENTS AS THE BASE MODULE. THE MANDATORY POLICY ENTRIES ARE NOTED (THE TYPE, ROLE AND USER REQUIRE AT LEAST ONE ENTRY EACH).....	114
TABLE 4-2: POLICY LANGUAGE RESERVED WORDS.....	117
TABLE 4-3: THE POLICY LANGUAGE STATEMENTS AND RULES THAT ARE ALLOWED WITHIN EACH TYPE OF POLICY SOURCE FILE - THE LEFT HAND SIDE OF THE TABLE SHOWS WHAT POLICY LANGUAGE STATEMENTS AND RULES ARE ALLOWED WITHIN EACH TYPE OF POLICY SOURCE FILE. THE RIGHT HAND SIDE OF THE TABLE SHOWS WHETHER THE STATEMENT IS VALID WITHIN THE IF / ELSE CONSTRUCT, OPTIONAL {RULE_LIST}, OR REQUIRE {RULE_LIST} STATEMENT.....	118
TABLE 4-4: SELINUX IDENTIFIER NAMING CONVENTIONS.....	119
TABLE 4-5: SENSITIVITY AND CATEGORY = SECURITY LEVEL – THIS TABLE SHOWS THE MEANINGS DEPENDING ON THE CONTEXT BEING DISCUSSED.....	153
TABLE 5-1: POLICY COMPONENTS - FOR THE POLICY.CONF AND BASE.CONF SOURCE FILE.	177
TABLE 6-1: SECHECKER PROFILES – THE PROFILES USED TO CHECK THE MODULAR-TEST PACKAGES AND THE BINARY POLICY FILES..	226

TABLE 6-2: MODULES IN VERSION 1.1 OF SECHECKER(8) – THE COMMENTS COLUMN COVERS THE AUTHORS INTERPRETATION OF THE TEST RESULTS ON THE MODULAR-TEST POLICY BASE AND LOADABLE MODULES USING THE SECHECKER MODULES AND PROFILES.....231

TABLE 6-3: UTILITY MODULES IN VERSION 1.1 OF SECHECKER(8) – THE COMMENTS COLUMN COVERS THE AUTHORS INTERPRETATION OF THE TEST RESULTS ON THE MODULAR-TEST POLICY BASE AND LOADABLE MODULES USING THE SECHECKER MODULES AND PROFILES.....234

TABLE 7-1: THE REFERENCE POLICY FILES AND DIRECTORIES.....253

TABLE 7-2: BUILD.CONF ENTRIES.....254

TABLE 7-3: M4 PARAMETERS SET AT BUILD TIME.....255

TABLE 7-4: REQUIRED MODULES.CONF ENTRIES.....257

TABLE 7-5: GENERAL BUILD MAKE TARGETS.....258

TABLE 7-6: MODULAR POLICY BUILD MAKE TARGETS.....258

TABLE 7-7: MONOLITHIC POLICY BUILD MAKE TARGETS.....259

TABLE 7-8: RED HAT SPECIFIC POLICY CONFIGURATION FILES – THIS EXAMPLE BUILDS A ‘TARGETED’ POLICY.....263

TABLE 7-9: HEADER POLICY BUILD MAKE TARGETS.....269

TABLE 7-10: MACROS LISTED IN THIS SECTION.....270

TABLE 8-1: COMMON FILE PERMISSIONS.....300

TABLE 8-2: COMMON SOCKET PERMISSIONS.....301

TABLE 8-3: COMMON IPC PERMISSIONS.....301

TABLE 8-4: COMMON POSTGRESQL DATABASE PERMISSIONS.....302

0.5.2 Figures

FIGURE 2-1: HIGH LEVEL CORE SELINUX COMPONENTS - DECISIONS BY THE SECURITY SERVER ARE CACHED IN THE AVC TO ENHANCE PERFORMANCE OF FUTURE REQUESTS.....20

FIGURE 2-2: HIGH LEVEL SELINUX ARCHITECTURE – SHOWING THE MAJOR SUPPORTING SERVICES.....21

FIGURE 2-3: PROCESSING A SYSTEM CALL – THE DAC CHECKS ARE CARRIED OUT FIRST, IF THEY PASS THEN THE SECURITY SERVER IS CONSULTED FOR A DECISION.....24

FIGURE 2-4: ROLE BASED ACCESS CONTROL – SHOWING HOW SELINUX CONTROLS ACCESS VIA USER, ROLE AND DOMAIN TYPE ASSOCIATION.....26

FIGURE 2-5: OBJECT CLASS = ‘FILE’ AND PERMISSIONS – THE POLICY RULES WOULD DEFINE THOSE PERMISSIONS ALLOWED FOR EACH

PROCESS THAT NEEDS ACCESS TO THE /ETC/SELINUX/CONFIG FILE.30

FIGURE 2-6: THE ALLOW RULE – SHOWING THAT THE SUBJECT (THE PROCESSES RUNNING IN THE UNCONFINED_T DOMAIN) HAS BEEN GIVEN THE TRANSITION PERMISSION ON THE EXT_GATEWAY_T ‘PROCESS’ OBJECT.....31

FIGURE 2-7: DOMAIN TRANSITION – WHERE THE SECURE_SERVER IS EXECUTED WITHIN THE UNCONFINED_T DOMAIN AND THEN TRANSITIONED TO THE EXT_GATEWAY_T DOMAIN.....36

FIGURE 2-8: SECURITY LEVELS AND DATA FLOWS – THIS SHOWS HOW THE PROCESS CAN ONLY ‘READ DOWN’ AND ‘WRITE UP’ WITHIN AN MLS ENABLED SYSTEM.....40

FIGURE 2-9: SHOWING THE MLSCONSTRAIN STATEMENTS CONTROLLING READ DOWN & WRITE UP – THIS TIES IN WITH TABLE 2-2 THAT SHOWS A PROCESS RUNNING WITH A SECURITY RANGE OF S0 – S3:C1.C5.....44

FIGURE 2-10: HOOKS FOR THE FORK SYSTEM CALL - THIS DESCRIBES THE STEPS REQUIRED TO CHECK ACCESS PERMISSIONS FOR OBJECT CLASS ‘PROCESS’ AND PERMISSION ‘FORK’.....64

FIGURE 2-11: PROCESS TRANSITION - THIS SHOWS THE MAJOR STEPS REQUIRED TO CHECK IF TRANSITION IS ALLOWED FROM THE UNCONFINED_T DOMAIN TO THE EXT_GATEWAY_T DOMAIN.68

FIGURE 2-12: THE MAIN LSM / SELINUX MODULES – THE FORK AND EXEC FUNCTIONS LINK TO FIGURE 2-7 WHERE THE TRANSITION PROCESS IS DESCRIBED.....69

FIGURE 2-13: THE BOOT SEQUENCE – THIS SHOWS HOW SELINUX IS INITIALISED AND THE POLICY LOADED DURING THE BOOT PROCESS.....73

FIGURE 2-14: COMPAT_NET CONTROLS – SHOWING THE POLICY STATEMENTS AND RULES REQUIRED TO ALLOW COMMUNICATIONS.74

FIGURE 2-15: SECMARK PROCESSING – RECEIVED PACKETS ARE PROCESSED BY THE INPUT CHAIN WHERE LABELS ARE ADDED TO THE APPROPRIATE PACKETS THAT WILL EITHER BE ACCEPTED OR DROPPED BY THE SECMARK PROCESS. PACKETS BEING SENT ARE TREATED THE SAME WAY.....75

FIGURE 2-16: FALLBACK LABELING – SHOWING THE DIFFERENCES BETWEEN THE POLICY CAPABILITY NETWORK_PEER_CONTROLS SET TO 0 AND 1.....76

FIGURE 2-17: IPSEC COMMUNICATIONS – THE SPD CONTAINS INFORMATION REGARDING THE SECURITY CONTEXTS TO BE USED. THESE ARE EXCHANGED BETWEEN THE TWO SYSTEMS AS PART OF THE CHANNEL SET-UP.....77

FIGURE 2-18: MLS SYSTEMS ON THE SAME NETWORK.....79

FIGURE 2-19: MLS SYSTEMS ON DIFFERENT NETWORKS COMMUNICATING VIA A GATEWAY.....79

FIGURE 3-20: FILE CONTEXT CONFIGURATION FILES - THE TWO FILES COPIED TO THE POLICY AREA WILL BE USED BY THE FILE LABELING UTILITIES TO RELABEL FILES.....88

FIGURE 5-21: MESSAGE FILTER COMPONENTS.....189

FIGURE 5-22: SECMARK TESTING – THE SCENARIOS FOR TESTING THE ACCESS ALLOWED FOR SECMARK PACKETS. NOTE THAT NOT ALL OF THESE TESTS WILL BE DESCRIBED.....191

FIGURE 5-23: TESTING USING THREE VIRTUAL TERMINAL SESSIONS204

FIGURE 5-24: RUNNING THE SECURE CLIENT / SERVER WITH NETLABEL ENABLED.....210

FIGURE 5-25: TESTING THE MESSAGE FILTER SERVICE.....220

FIGURE 6-26: OPENING THE TWO POLICIES IN SEDIFFX.....223

FIGURE 6-27: SEDIFFX SHOWING THE DIFFERENCES IN THE TWO POLICIES.....224

FIGURE 6-28: OPENING THE PACKAGE POLICY FILES FOR ANALYSIS236

FIGURE 6-29: THE MODULAR, BINARY AND BASE SOURCE POLICY SUMMARIES.....237

FIGURE 6-30: TYPE ENFORCEMENT RULES (1) - FINDING TE RULES USING A REGULAR EXPRESSION.....237

FIGURE 6-31: TYPE ENFORCEMENT RULES (2) - FINDING TE RULES USING A REGULAR EXPRESSION WITH CLASS/PERMISSIONS TAB SET TO SHOW ONLY THE PROCESS CLASS.....238

FIGURE 6-32: DIRECT RELABEL - SUBJECT: UNCONFINED_T.....239

FIGURE 6-33: TRANSITIVE INFORMATION FLOW - STARTING TYPE : IN_FILE_T SHOWING THE ‘TO’ DIRECTION.....240

FIGURE 7-34: EXAMPLE DOCUMENTATION SCREEN SHOT.....248

FIGURE 7-35: THE REFERENCE POLICY SOURCE TREE – WHEN BUILDING A MODULAR POLICY FILES ARE ADDED TO THE POLICY STORE, FOR MONOLITHIC BUILDS THE POLICY STORE IS NOT USED.250

FIGURE 7-36: BASE MODULE BUILD – THIS SHOWS THE TEMPORARY BUILD FILES USED TO BUILD THE BASE MODULE ‘BASE.CONF’ AS A PART OF THE ‘MAKE’ PROCESS. NOTE THAT THE MODULES MARKED AS BASE IN MODULES.CONF ARE BUILT HERE.261

FIGURE 7-37: MODULE BUILD – THIS SHOWS THE MODULE FILES AND THE TEMPORARY BUILD FILES USED TO BUILD EACH MODULE AS A PART OF THE ‘MAKE’ PROCESS (I.E. THOSE MODULES MARKED AS MODULE IN MODULES.CONF).....262

FIGURE 7-38: THE TWO ‘TARGETED’ POLICIES SHOULD BE THE SAME USING SEDIFFX.....266

FIGURE 7-39: THE MAKE ADA SEQUENCE OF EVENTS.....289

FIGURE 7-40: THE RESULTING CODE - THE EXPANDED CODE IN THE MODULE EXPANSION SECTION.....290

FIGURE 11-41: RUNNING THE SECURE CLIENT / SERVER – ONCE WITH NO NETLABEL AND ONCE WITH NETLABEL ENABLED.....343

FIGURE 12-42: RUNNING THE SECURE CLIENT / SERVER – ONCE WITH NO NETLABEL, ONCE WITH NETLABEL ENABLED AND ONCE WITH LABELED IPSEC ENABLED (NOTE THAT THIS LABEL TAKES PRECEDENCE OVER THE FALLBACK NETLABEL).....348

FIGURE 15-43: APOL BEFORE AND AFTER THE FONT SIZE CHANGE.355

1. The SELinux Notebook

1.1 Introduction

This Notebook has been assembled from information that is available within the public domain and where necessary, updated to reflect the Linux Security Module (LSM) and Security-Enhanced Linux (SELinux) services as built into the Fedora 10 release¹ of GNU / Linux.

It should help with explaining:

1. SELinux and its purpose in life.
2. The LSM / SELinux architecture, its supporting services and how they are implemented within GNU / Linux.
3. The core SELinux policy language and how basic policy modules can be constructed for instructional purposes.
4. The core SELinux policy management tools with examples of usage.
5. The [Reference Policy](#) architecture, its supporting services and how it is implemented.

However, this Notebook will **not** explain how the SELinux policies are managed within each GNU / Linux distribution as they have their own supporting information (e.g. Fedora has the [Fedora 10 SELinux User Guide](#) [Ref. 1] and Gentoo has the [Gentoo SELinux Handbook](#) [Ref. 2]).

1.1.1 Sample Policy Source

This Notebook contains a number of sample policy source files that have been written by the author to better understand SELinux (see the [Building a Basic Policy](#) section). These do not use the Reference Policy but are built using SELinux policy language statements to form a [very simple message filter](#) that is then investigated using various SELinux tools.

The software is available as an rpm, however it is possible to copy and paste the code from the relevant sections of this Notebook into an editor such as `gedit`.

Note that the naming convention used for the sample policy source files in this Notebook are `<module_name>.conf` (e.g. `ext_gateway.conf`) as they are not Reference Policy modules.

1.2 Health Warning

This Notebook has been written as a set of notes on how SELinux is implemented within GNU / Linux (using Fedora 10 as the base). The author is not an SELinux expert (as this was written to capture information that seems helpful with learning about SELinux), therefore readers need to be aware that the text should be treated with caution (however all reasonable care has been taken to ensure accuracy – the sample code has even been tested!!). All corrections are welcome.

¹ This Notebook uses Fedora 10 simply because that is what is installed on the authors test system.

1.3 Root Login Requirements

To be able to build and test the policies described in the [Building a Basic Policy](#) section users need to be logged in as 'root', however by default F-10 does not allow root login, therefore to fix this:

1. Boot from the F-10 build CD / DVD and select the rescue option. The system will then be mounted under `/mnt/sysimage`.

2. Login as root by:

```
su -
```

3. Edit the `/mnt/sysimage/etc/pam.d/gdm` file with `vi` and comment out the line:

```
auth required pam_succeed_if.so user != root quiet
```

```
# auth required pam_succeed_if.so user != root quiet
```

4. Save the file and exit the rescue process. When F-10 is re-booted, it will be possible to login as root (the root password was chosen when the system was initially installed).

1.4 Relevant F-10 Packages

The following are the relevant rpm packages installed on the test machine and used for all code listings, testing and research:

```
checkpolicy-2.0.16-3.fc10.i386
checkpolicy-2.0.16-3.fc10.src.rpm

coreutils-6.12-20.f10.src.rpm

ipsec_tools-0.7.2-1.fc10.i386

kernel-2.6.27.30-170.2.82.fc10.i686
kernel-2.6.27.30-170.2.82.fc10.src.rpm

libselinux-2.0.78-1.fc10.i386
libselinux-devel-2.0.78-1.fc10.i386
libselinux-python-2.0.78-1.fc10.i386
libselinux-utils-2.0.78-1.fc10.i386

libsemanage-2.0.28-1.fc10.i386
libsemanage-devel-2.0.28-1.fc10.i386
libsemanage-python-2.0.28-1.fc10.i386

libsepol-2.0.33-1.fc10.i386
libsepol-devel-2.0.33-1.fc10.i386
libsepol-static-2.0.33-1.fc10.i386
libsepol-2.0.33-1.fc10.src.rpm

mcstrans-0.2.11-1.fc10.i386

mkinitrd-6.0.71-6.fc10.src.rpm

netlabel_tools-0.18-1.fc10.i386.rpm

policycoreutils-2.0.57-22.fc10.i386
policycoreutils-gui-2.0.57-22.fc10.i386
policycoreutils-newrole-2.0.57-22.fc10.i386
```

```

selinux-policy-3.5.13-70.fc10.src.rpm
selinux-policy-3.5.13-70.fc10.noarch
selinux-policy-doc-3.5.13-70.fc10.noarch
selinux-policy-minimum-3.5.13-70.fc10.noarch
selinux-policy-mls-3.5.13-70.fc10.noarch
selinux-policy-targeted-3.5.13-70.fc10.noarch

setools-3.3.5-1.fc10.i386
setools-console-3.3.5-1.fc10.i386
setools-gui-3.3.5-1.fc10.i386
setools-libs-3.3.5-1.fc10.i386
setools-libs-java-3.3.5-1.fc10.i386
setools-libs-tcl-3.3.5-1.fc10.i386

upstart-0.3.9-22.fc10.src.rpm
    
```

The gcc tools will be required to compile and link the test ‘C’ applications used in some of the [Building a Basic Policy](#) loadable module scenarios (gcc-4.3.2-7.i386 and libgcc-4.3.2-7.i386 rpms are installed on the test machine that is using the kernel-2.6.27.30-170.2.82.fc10.i686 rpm).

1.5 Notebook Sections

[Table 1-1](#) describes the contents of the main sections within this Notebook.

Section	Description
SELinux Overview	<p>Gives a high level description of SELinux and its major components to provide Mandatory Access Control services for GNU / Linux.</p> <p>Hopefully it will show how all the SELinux components link together.</p>
SELinux Configuration Files	<p>Describes all the known SELinux configuration files in F-10 with samples. Also lists any specific commands or libselinux APIs used to manage them. (may have missed some !!).</p>
SELinux Policy Language	<p>Gives a brief description of each policy language statement, with supporting examples taken from the F-10 policy source.</p>
Building a Basic Policy	<p>Describes how to build monolithic, base and loadable policy modules using core policy language statements and SELinux commands. Note that these policies should not to be used in a live environment, they are examples to show simple policy construction.</p>
Policy Investigation Tools	<p>Describes the applications available to investigate policy. These are used to investigate the loadable modules developed in the Building a Basic Policy section.</p>
The Reference Policy	<p>Describes the Reference Policy and its supporting macros.</p> <p>Note that most GNU / Linux distributions now ship</p>

Section	Description
	with the Reference Policy and have their own supporting information.
Appendix A - Object Classes and Permissions	Describes the SELinux object classes and permissions. These have been updated to reflect those in F-10 (X-Windows needs work).
Appendix B – SELinux Commands	Describes each of the core SELinux commands.
Appendix C – API Summary for libselinux	Contains a sorted alphabetical list of <code>libselinux</code> library functions with comments extracted from the header files.
Appendix D - Reference Policy Macros	Reference Policy Macros used in module source files.
Appendix E – NetLabel Module Support for network_peer_controls	This builds on the modules developed in the Building a Basic Policy section to implement an enhanced module to support the new network peer controls.
Appendix F – Labeled IPSec Module Example	This builds on the modules developed in the Building a Basic Policy section to implement Labeled IPSec.
Appendix G – Implementing a constraint	This builds on the modules developed in the Building a Basic Policy section to show a simple constraint statement and its impact on the policy.
Appendix H - Bugs and Features	This section contains a list of possible bugs and features found while using SELinux (including OpenOffice.org - Writer).
Appendix I - Useful Information	This section contains information that could be useful.
Appendix J – References	List of references used within this Notebook and useful websites.
Appendix K - GNU Free Documentation License	A license to allow anyone to copy this Notebook.

Table 1-1: The SELinux Notebook Sections

2. SELinux Overview

2.1 Introduction

SELinux is the primary Mandatory Access Control (MAC) mechanism built into a number of GNU / Linux distributions. SELinux originally started as the Flux Advanced Security Kernel (FLASK) development by the Utah university Flux team and the US Department of Defence. The development was enhanced by the NSA and released as open source software. The history of SELinux can be found at the [Flux](#) and [NSA](#) websites.

This Notebook will concentrate on describing SELinux as delivered in the Fedora F-10 release.

Each of the sections that follow will describe a component of SELinux, and hopefully they are in some form of logical order.

2.2 Core SELinux Components

[Figure 2-1](#) shows a high level diagram of the SELinux core components that manage enforcement of the policy and comprise of the following:

1. A [subject](#) that must be present to cause an action to be taken by an [object](#) (such as read a file as information only flows when a subject is involved).
2. An Object Manager that knows the actions required of the particular resource (such as a file) and can enforce those actions (i.e. allow it to write to a file if permitted by the policy).
3. A Security Server that makes decisions regarding the subjects rights to perform the requested action on the object, based on the security policy rules.
4. A Security Policy that describes the rules using the SELinux [policy language](#).
5. An Access Vector Cache (AVC) that improves system performance by caching security server decisions.

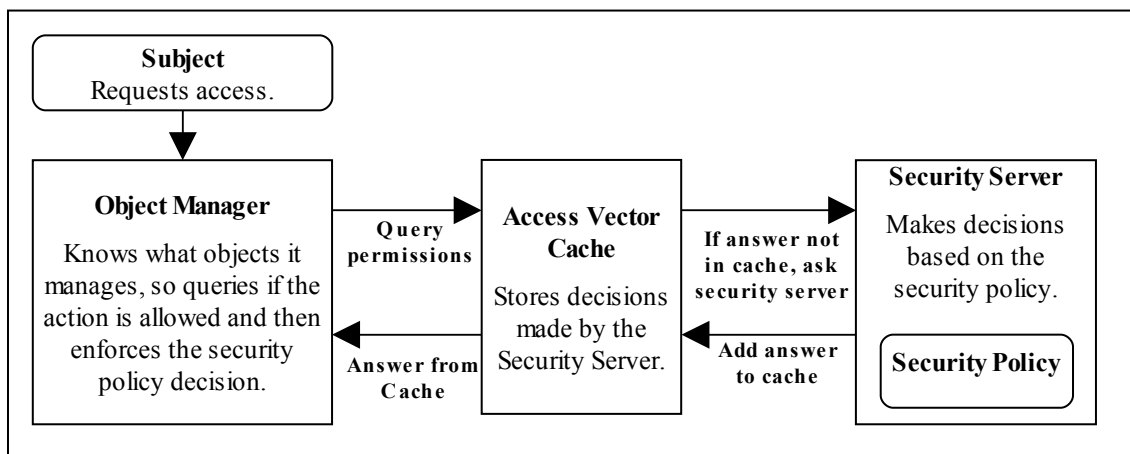


Figure 2-1: High Level Core SELinux Components - Decisions by the Security Server are cached in the AVC to enhance performance of future requests.

The SELinux Notebook - The Foundations

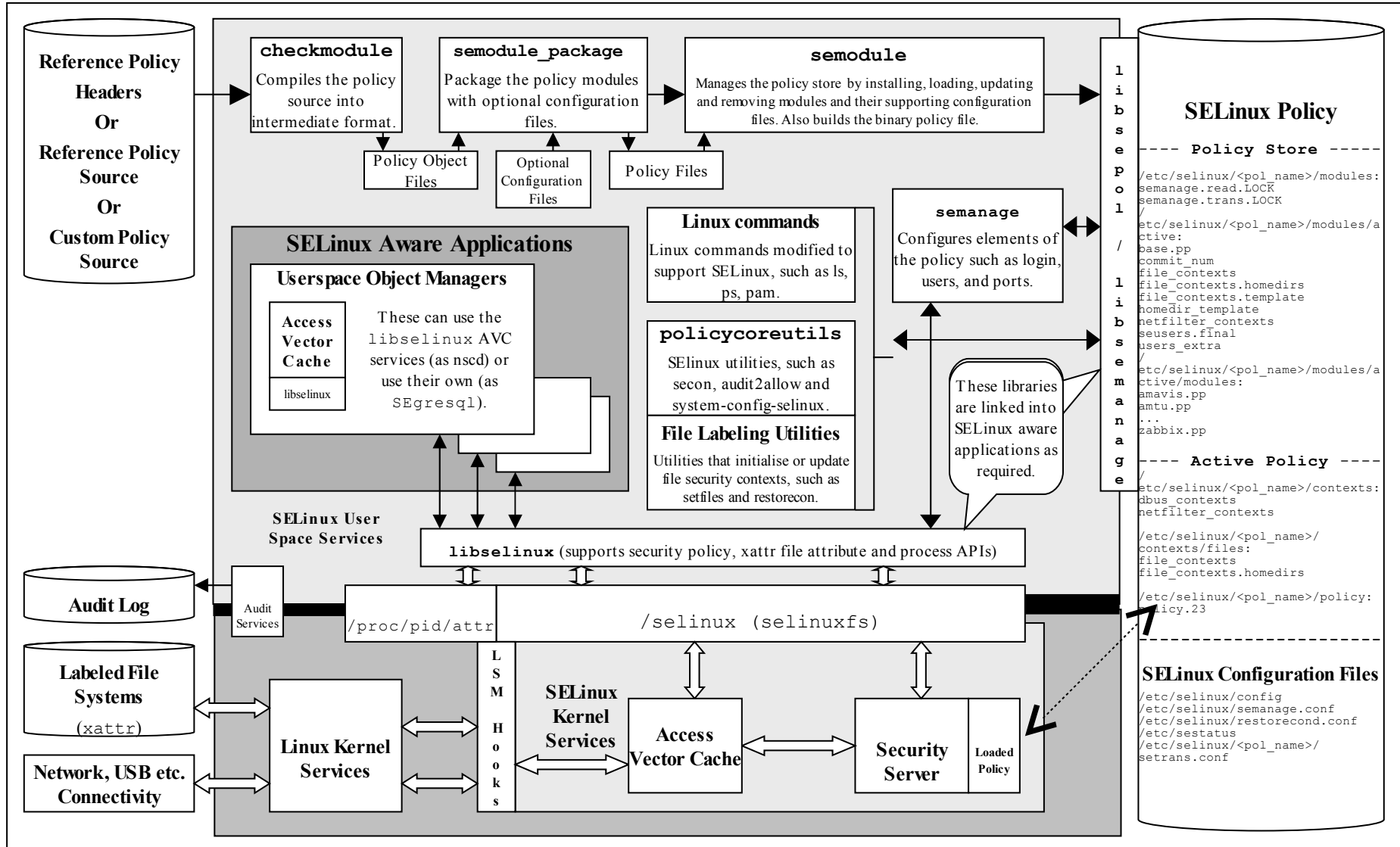


Figure 2-2: High Level SELinux Architecture – Showing the major supporting services

[Figure 2-2](#) shows a more complex diagram of kernel and userspace with a number of supporting services which are used to manage the SELinux environment. This diagram will be referenced a number of times to explain areas of SELinux, therefore using [Figure 2-2](#) as the reference and starting from the bottom:

- a) In the current implementation of SELinux, the security server is embedded in the kernel² with the policy being loaded from userspace via a series of functions contained in a library ([libselinux](#)).

However the object managers (OM) and access vector cache (AVC) can reside in:

kernel space – These object managers are for the kernel services such as files, directory, socket, IPC etc. and are provided by hooks into the SELinux sub-system via the Linux Security Module (LSM) framework (shown as LSM Hooks in [Figure 2-2](#)) that is discussed in the [LSM](#) section. The SELinux kernel AVC service is used to cache their requests and the security servers response.

userspace – These object managers are provided with the application / service that requires support for MAC and are known as ‘SELinux-aware³’ applications or services. Examples of these are: X-Windows, D-bus messaging (used by the Gnome desktop), PostgreSQL database, Name Service Cache Daemon (nscd), and the GNU / Linux `passwd` command. Generally, these OMs use the AVC services built into the SELinux library ([libselinux](#)), however they could if required supply their own AVC or not use an AVC at all.

Note that all userspace OMs use the SELinux library functions to communicate access decisions to and from the kernel security server.

- b) The loadable policy (right hand side of [Figure 2-2](#)) and its supporting configuration files are contained in the `/etc/selinux` directory. This directory contains the main SELinux configuration file ([config](#)) that names the policy to be loaded and the initial status of SELinux at boot time (enforcing⁴ the policy or not). The area also holds all policies that can be activated in their respective directories `/etc/selinux/<policy_name>` (e.g. `/etc/selinux/targeted` would hold the ‘targeted’ F-10 policy and all its configuration files). All know configuration files for F-10 SELinux are shown in the [SELinux Configuration Files](#) section.
- c) SELinux supports a ‘modular policy’, this means that a policy does not have to be one large policy, but can be built from modules. A modular policy consists of a base policy that contains the mandatory information (such as object classes, permissions etc.), and zero or more policy modules that generally support a particular application or service. These modules are compiled, linked, and held in a ‘policy store’ where they can be built into a binary format that is then

² There is a project developing a Policy Management Server (PMS) that will utilise a userspace security server, however it is beyond the scope of this Notebook.

³ Generally this means that they use the services provided by the `libselinux` library as a minimum.

⁴ When SELinux is enabled the policy can be running in ‘permissive mode’, where all accesses are allowed and logged in the audit log, even if they are not permitted by the policy (useful for debugging policy). The policy can also be run in ‘enforcing mode’, where any access that is not defined in the policy is denied and an entry placed in the audit log.

loaded into the security server. The types of policy and their construction are covered in the [Types of SELinux Policy](#) section.

- d) To be able to build the policy in the first place, policy source is required (top left hand side of [Figure 2-2](#)). This can be supplied in two basic ways:
 - i) as source code written using the [SELinux Policy Language](#). This is how the simple policies have been written to support the examples in this [Notebook](#), however it is not recommended for real-world policy development.
 - ii) using the Reference Policy that uses high level macros to define policy rules as well as the policy language. This is the standard way policies are now built for SELinux distributions such as F-10 and is discussed in the [Reference Policy](#) section.
- e) To be able to compile and link the source code and load it into the security server requires a number of tools (top of [Figure 2-2](#)). These are used to build the [sample policy modules](#) where their use is described.
- f) To enable system administrators to manage the policy, the SELinux environment and label file systems also requires tools and modified GNU / Linux commands. These are mentioned throughout the Notebook as needed and summarised in [Appendix B – SELinux Commands](#). Note that there are many other applications to manage policy, however this Notebook only concentrates on the core services.
- g) To ensure security events are logged, GNU / Linux has an audit service that captures policy violations. The [Audit Logs](#) section describes the format of these AVC security events.
- h) SELinux supports network services that are described in the [SELinux Networking Support](#) section, and the [message filter](#) is built using some of these features.

The [Linux Security Module and SELinux](#) section goes into greater detail of the LSM / SELinux modules with a walk through of a fork and exec process.

2.3 Mandatory Access Control (MAC)

Mandatory Access Control (MAC) is a type of access control in which the operating system is used to constrain a user or process (the subject) from accessing or performing an operation on an object (such as a file, disk, memory etc.).

Each of the subjects and objects have a set of security attributes that can be interrogated by the operating system to check if the requested operation can be performed or not. For SELinux the:

- [subjects](#) are processes.
- [objects](#) are system resources such as files, sockets, etc. (there are 73 objects defined in F-10).
- security attributes are the [security context](#).
- Security Server within the Linux kernel authorizes access (or not) using the:
 - security policy (or policy) that describes rules that must be obeyed.

Note that the subject (and therefore the user) cannot decide to bypass the policy rules being enforced by the MAC policy with SELinux enabled. Contrast this to standard Linux Discretionary Access Control (DAC), which also governs the ability of subjects to access objects, however it allows users to make policy decisions.

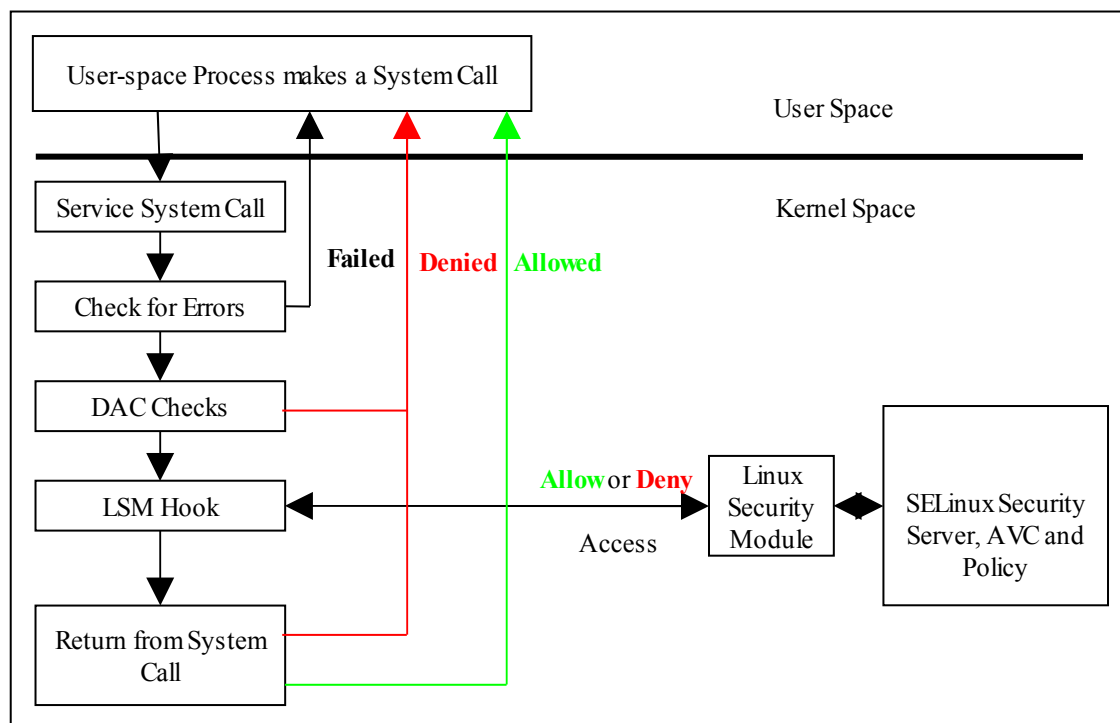


Figure 2-3: Processing a System Call – The DAC checks are carried out first, if they pass then the Security Server is consulted for a decision.

SELinux supports two forms of MAC:

Type Enforcement – Where processes run in domains and the actions on objects are controlled by the policy. This is the implementation used for general purpose MAC within SELinux. The [Type Enforcement](#) section covers this in more detail.

Multi-Level Security – This is an implementation based on the Bell-La Padula (BLP) model, and used by organizations where different levels of access are required so that (for example in some defence / Government systems) restricted information is separated from classified information (i.e. maintaining confidentiality). This allows enforcement rules such as ‘no write down’ and ‘no read up’ to be implemented in a policy by extending the security context to include security levels. The [MLS](#) section covers this in more detail along with a variant of MLS called Multi-Category Security (MCS).

2.4 Type Enforcement (TE)

SELinux makes use of a specific style of type enforcement⁵ (TE) to enforce mandatory access control. For SELinux it means that all [subjects](#) and [objects](#) have a type identifier associated to them that can then be used to enforce rules laid down in a policy.

The SELinux type identifier is a simple variable-length string that is defined in the policy and then associated to a [security context](#). It is also used in the majority of

⁵ There are various ‘type enforcement’ technologies.

[SELinux language statements and rules](#) used to [build a policy](#) that will, when loaded into the security server, enforce the policy.

Because the type identifier (or just ‘type’) is associated to all subjects and objects, it can sometimes be difficult to distinguish what the type is actually associated with (it’s not helped by the fact that by convention, type identifiers all end in ‘_t’). In the end it comes down to understanding how they are allocated in the policy itself and how they are used by SELinux services.

Basically if the type identifier is used to reference a subject it is referring to a GNU / Linux process or domain (i.e. domain type). If the type identifier is used to reference an object then it is specifying its object type (i.e. file type).

While SELinux refers to a subject as being an active process that is associated to a domain type, the scope of an SELinux type enforcement domain can vary widely. For example in the simple policy built in the [Building a Basic Policy](#) section, all the processes on the system run in the `unconfined_t` domain, therefore every process is ‘of type `unconfined_t`’ (that means it can do whatever it likes within the limits of the standard Linux DAC policy).

It is only when additional policies are implemented in the simple policy (via [loadable modules](#)), that areas start to be confined, for example an external gateway is run in its own isolated domain (`ext_gateway_t`) that cannot be ‘interfered’ with by any of the `unconfined_t` processes (except to run or transition the gateway process into its own domain). This scenario is similar to the ‘targeted’ policy delivered as standard in F-10 where the majority of user space processes run under the `unconfined_t` domain (although don’t think they are equivalent as the policies supplied with F-10 have areas isolated by various domains and has evolved over years of work).

2.4.1 Constraints

Within a TE environment the way that subjects are allowed to access an object is via an [allow rule](#), for example:

```
allow unconfined_t ext_gateway_t : process transition;
```

This is explained in more detail later, however it states that a process running in the `unconfined_t` domain has permission to transition a process to the `ext_gateway_t` domain. However it could be that the policy writer wants to constrain this further and state that this can only happen if the role of the source domain is the same as the role of the target domain. To achieve this a constraint can be imposed using a [constrain](#) statement:

```
constrain process transition ( r1 == r2 );
```

This states that a process transition can only occur if the source role is the same as the target role, therefore a constraint is a condition that must be satisfied in order for one or more permissions to be granted (i.e. a constraint imposes additional restrictions on TE rules). An example of this can be found in [Appendix G – Implementing a Constraint](#).

There are a number of different constraint statements within the policy language to support areas such as MLS (see the [Constraint Statements](#) and [MLS Statements](#) sections).

2.5 Role-Based Access Control (RBAC)

To further control access to TE domains SELinux makes use of role-based access control (RBAC). This feature allows SELinux users to be associated to one or more roles, where each role is then associated to one or more domain types as shown in [Figure 2-4](#). Note that GNU / Linux users are not a direct part of the RBAC feature, they are associated to SELinux users via SELinux specific commands⁶ such as:

semanage login - That manages the association of GNU / Linux users to SELinux users.

semanage user - That manages the association of SELinux users to roles.

[Figure 2-4](#) shows how the SELinux user and roles are associated within the basic loadable modules that form the [simple message filter exercise](#).

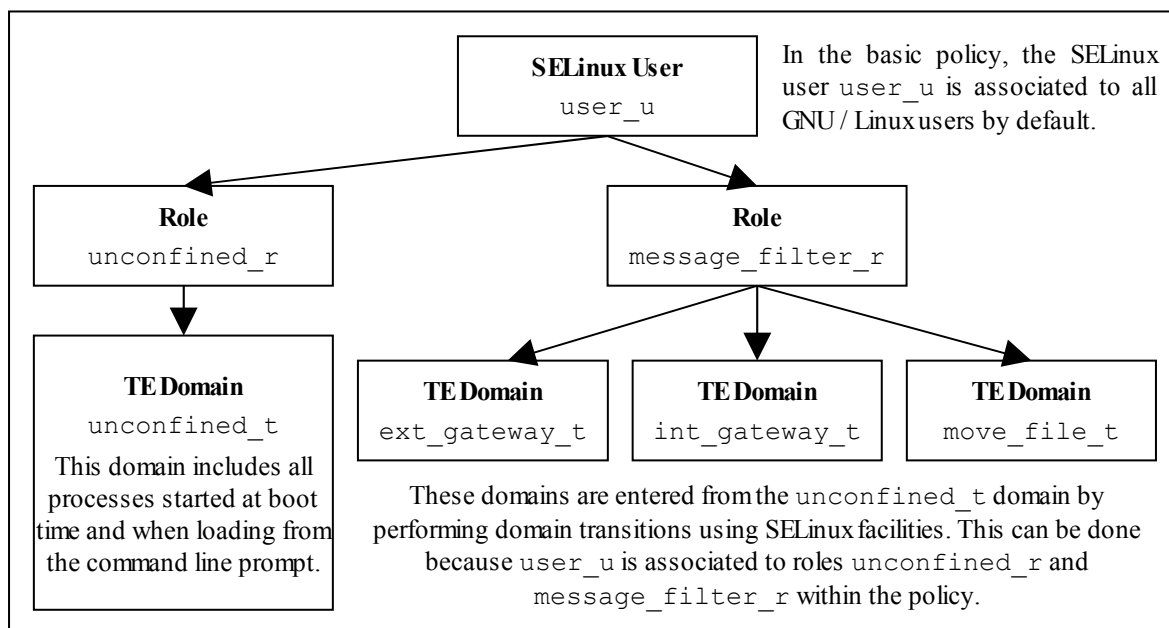


Figure 2-4: Role Based Access Control – Showing how SELinux controls access via user, role and domain type association.

SELinux users can be equated to groups or classes of user, for example in the Reference Policy there is `user_u` for general users with `staff_u` and `sysadm_u` for more specialised users. There is also a `system_u` defined that must never be associated to a GNU / Linux user as it a special identity for system processes and objects.

2.6 Security Context

SELinux requires a security context to be associated with every process (or subject) and object that are used by the security server to decide whether access is allowed or not as defined by the policy.

The security context is also known as a ‘security label’ or just label that can cause confusion as there are many types of label depending on the context (another context!!).

⁶ There are other SELinux utilities that can manage users etc., however this Notebook will only use the core utilities.

Within SELinux, a security context is represented as variable-length strings that define the SELinux user⁷, their role, a type identifier and an optional MCS / MLS range as follows:

```
user:role:type[:range]
```

Where:

<code>user</code>	The SELinux user identity. This can be associated to one or more roles that the SELinux user is allowed to use.
<code>role</code>	The SELinux role. This can be associated to one or more types the SELinux user is allowed to access.
<code>type</code>	When a type is associated with a process, it defines what processes (or domains) the SELinux user (the subject) can access. When a type is associated with an object, it defines what access permissions the SELinux user has to that object.
<code>range</code>	This field is only present if the policy supports MCS / MLS and is discussed in the MLS / MCS Range section.

However note that:

1. Access decisions regarding a subject make use of all the components of the [security context](#).
2. Access decisions regarding an object make use of all the components of the security context, **however**:
 - a) the `user` is either set to a special user called `system_u` or it is set to the SELinux user id of the creating process (as it serves no real purpose other than it can be used for audit purposes within logs).
 - b) the `role` is not relevant to security decisions and is always set to a special SELinux internal role of `object_r`.

Therefore for an object, the `type` (and `range` for MLS) are the only relevant security fields that are used in access decisions.

Examples of using `system_u` and `object_r` can be seen in the file system after relabeling and running the `ls -Z` command on various directories.

The examples below show security contexts for processes, directories and files (note that the policy did not support MCS or MLS, therefore no `range` fields):

⁷ An SELinux user id is not the same as the GNU / Linux user id. The GNU / Linux user id is mapped to the SELinux user id by configuration files (via the `semanage (8)` command).

Example Process Security Context:

```
# These are process security contexts taken from a ps -Z command
# (edited for clarity) that show four processes:

LABEL                                PID  TTY  CMD
user_u:unconfined_r:unconfined_t    2539 pts/0 bash
user_u:message_filter_r:ext_gateway_t 3134 pts/0 secure_server
user_u:message_filter_r:int_gateway_t 3138 pts/0 secure_server
user_u:unconfined_r:unconfined_t    3146 pts/0 ps

# Note the bash and ps processes are running under the
# unconfined_t domain, however the secure_server has two instances
# running under two different domains (ext_gateway_t and
# int_gateway_t). Also note that they are using the
# message_filter_r role whereas bash and ps use unconfined_r.
#
# These results were obtained by running the system in permissive
# mode (as in enforcing mode the gateway processes would not
# be shown).
```

Example Object Security Context:

```
# These are the message queue directory object security contexts
# taken from an ls -Zd command (edited for clarity):

system_u:object_r:in_queue_t    /user/message_queue/in_queue
system_u:object_r:out_queue_t   /user/message_queue/out_queue

# Note that they are instantiated with system_u and object_r
```

```
# These are the message queue file object security contexts
# taken from an ls -Z command (edited for clarity):

/user/message_queue/in_queue:
user_u:object_r:in_file_t      Message-1
user_u:object_r:in_file_t      Message-2

/user/message_queue/out_queue:
user_u:object_r:out_file_t     Message-10
user_u:object_r:out_file_t     Message-11

# Note that they are instantiated with user_u as that was the
# SELinux user id of the process that created the files (see the
# process example above). The role remained as object_r.
```

2.7 Subjects

A subject is an active entity generally in the form of a person, process, or device that causes information to flow among objects or changes the system state.

Within SELinux a subject is generally an active process and has a [security context](#) associated with it, however a process can also be referred to as an object depending on the context in which it is being taken, for example:

1. A running process (i.e. an active entity) is a subject because it causes information to flow among objects or can change the system state.
2. The process can also be referred to as an object because each process has an associated object class⁸ called '[process](#)'. This process 'object', defines what permissions the policy is allowed to grant or deny on the active process.

An example is given of the above scenarios in the [Allowing a Process Access to an Object](#) section.

In SELinux subjects can be:

Trusted – Generally these are commands, applications etc. that have been written or modified to support specific SELinux functionality to enforce the security policy (e.g. the kernel, init, pam, xinetd and login). However, it can also cover any application that the organisation is willing to trust as a part of the overall system. Although (depending on your paranoia level), the best policy is to trust nothing until it has been verified that it conforms to the security policy. Generally these trusted applications would run in either their own domain (e.g. the audit daemon could run under `auditd_t`) or grouped together (e.g. the `semanage` and `semodule` commands could be grouped under `semanage_t`).

Untrusted – Everything else.

2.8 Objects

Within SELinux an object is a resource such as files, sockets, pipes or network interfaces that are accessed via processes (also known as subjects). These objects are classified according to the resource they provide with access permissions relevant to their purpose (e.g. read, receive and write), and assigned a [security context](#) as described in the following sections.

2.8.1 Object Classes and Permissions

Each object consists of a class identifier that defines its purpose (e.g. `file`, `socket`) along with a set of permissions⁹ that describe what services the object can handle (read, write, send etc.). When an object is instantiated it will be allocated a name (e.g. a file could be called `config` or a socket `my_connection`) and a security context (e.g. `system_u:object_r:selinux_config_t`) as shown in [Figure 2-5](#).

⁸ The object class and its associated permissions are explained in the [Process Object Class](#) section.

⁹ Also known in SELinux as Access Vectors (AV).

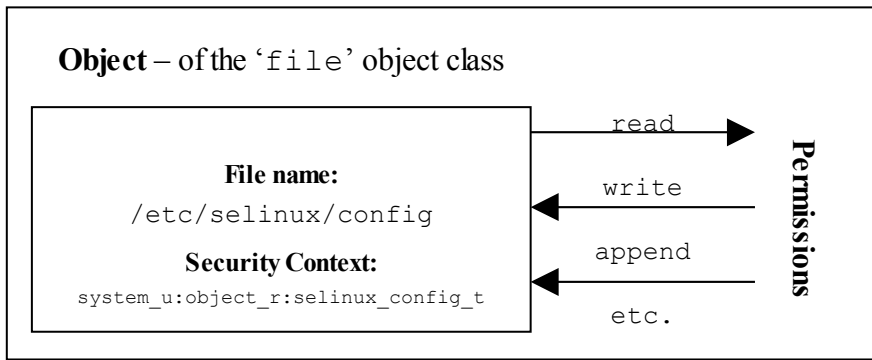


Figure 2-5: Object Class = 'file' and permissions – the policy rules would define those permissions allowed for each process that needs access to the /etc/selinux/configfile.

The objective of the policy is to enable the user of the object (the subject) access to the minimum permissions needed to complete the task (i.e. do not allow write permission if only reading information).

These object classes and their associated permissions are built into the GNU / Linux kernel and user space object managers by developers and are therefore not generally updated by policy writers.

The object classes consist of kernel object classes (for handling files, sockets etc.) plus user space object classes used by user space object managers (for services such as the name service cache daemon (nscd), X-Windows and debus). The number of object classes and their permissions can vary depending on the features configured in the GNU / Linux release. For example, F-10 defines 73 different object classes and 1,025 associated permissions which are described in [Appendix A - Object Classes and Permissions](#).

2.8.2 Allowing a Process Access to an Object

This is a simple example that attempts to explain two points:

1. How a process is given permission to use an objects resource.
2. By using the 'process' object class, show that a process can be described as a process or object.

The SELinux policy language has a number of rules and statements, however the majority of policies are built from [allow rules](#) as this (simply) allows processes to be given access permissions to an objects resources.

The following allow rule and [Figure 2-6](#) illustrates 'a process can also be an object' as it allows processes running in the unconfined_t domain, permission to 'transition' the external gateway application to the ext_gateway_t domain once it has been executed:

```

allow Rule | source_domain | target_type : class | permission
-----▼-----▼-----▼-----
allow      unconfined_t   ext_gateway_t : process transition;
    
```

Where:

- allow The SELinux language allow rule.
- unconfined_t The source domain (or subject) identifier – in this case the

shell that wants to exec the gateway application.

ext_gateway_t The target object identifier – the object instance of the gateway application process.

process The target object class - the ‘process’ object class.

transition The permission granted to the source domain on the targets object – in this case the unconfined_t domain has transition permission on the ext_gateway_t ‘process’ object.

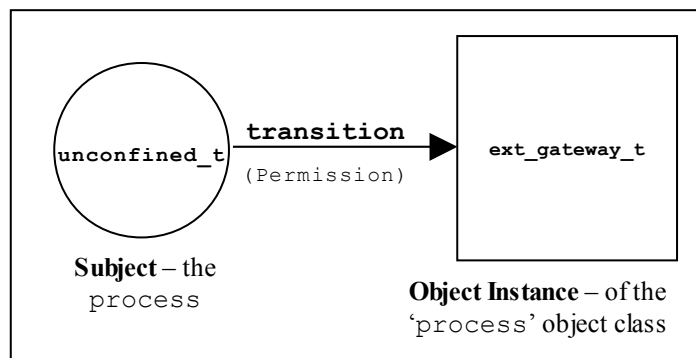


Figure 2-6: The allow rule – Showing that the subject (the processes running in the unconfined_t domain) has been given the transition permission on the ext_gateway_t ‘process’ object.

It should be noted that there is more to a domain transition than described above, for a more detailed explanation, see the [Domain Transition](#) section.

2.8.3 Labeling Objects

Within a running SELinux enabled GNU / Linux system the labeling of objects is managed by the system and generally unseen by the users (until labeling goes wrong !!). As processes and objects are created and destroyed, they either:

1. Inherit their labels from the parent process.
2. The policy type_transition statements allow a different label to be assigned as discussed in the [Domain and Object Transitions](#) section.
3. SELinux-aware applications can enforce a new label (with the policies approval of course) using the [libselinux API](#) functions.
4. The object manager (OM) can enforce a default label that can either be built into the OM or obtained via a configuration file (such as [X-Windows](#), [NetLabel](#) Labeled IPsec, and [SECMARK \(iptables\)](#)).
5. Use an ‘[initial security identifier](#)’ (or initial SID). These are defined in all [base and monolithic policies](#) and are used to either set an initial context during the boot process, or if an object requires a default (i.e. the object does not already have a valid context).

While the majority of objects are managed via the system automatically using either the inherited label or an initial SID as required, there are objects that need to have labels

defined for them either by their OM (bullet 4 above) using configuration files¹⁰ or by using policy language statements.

The SELinux policy language supports object labeling statements for file and network services that are defined in the [File System Labeling Statements](#) and [Network Labeling Statements](#) sections.

An overview of the process required for labeling files systems that use extended attributes (such as `ext3`) is discussed in the [Labeling Extended Attribute Filesystems](#) section.

2.8.3.1 Labeling Extended Attribute Filesystems

The labeling of files systems that implement extended attributes¹¹ (such as `ext3`), is supported by SELinux by:

1. The `fs_use_xattr` statement within the policy to identify what file systems use extended attributes. This statement will be used to inform the security server how the file system is labeled.
2. A 'file contexts' file that defines what the initial contexts should be for each file and directory within the file system. The format of this file is described in the [./modules/active/file_contexts.template file](#)¹² section.
3. A method to initialise the filesystem with these extended attributes. This is achieved by SELinux utilities such as `fixfiles(8)` and `setfiles(8)` that are described in [Appendix B – SELinux Commands](#). There are also commands such as `chcon(1)`, `restorecon(8)` and `restorecond(8)` that can be used to relabel files

Extended attributes containing the SELinux context of a file can be viewed by the `ls -Z` or `getfattr(1)` commands as follows:

```
ls -Z myfile
-rw-r--r-- root root unconfined_u:object_r:admin_home_t:s0 myfile
```

```
getfattr -n security.selinux myfile
#file: rpmbuild
security.selinux="unconfined_u:object_r:admin_home_t:s0\000"

# Where -n security.selinux is the name of the attribute and
# rpmbuild is the file name.
# The security context (or label) for the file is:
# system_u:object_r:admin_home_t:s0
```

2.8.3.1.1 Copying and Moving Files

¹⁰ The advantage of defining labels in an OM configuration file and not in the policy language is that the OM can then be used by other security mechanisms (for example NetLabel can be used by the [Simplified Mandatory Access Control Kernel](#) (SMACK) as this MAC system also hooks into the LSM).

¹¹ These file systems store the security context in an attribute associated with the file.

¹² Note that this file contains the contexts of all files in all extended attribute filesystems for the policy. However within a modular policy each module describes its own file context information, that is then used to build this file.

Assuming that the correct permissions have been granted by the policy, the effects on the security context of a file when copied or moved differ as follows:

- copy a file – takes on label of new directory unless the `-Z` option is used.
- move a file – retains the label of the file.

However, if the `restorecond` daemon is running and the [restorecond.conf](#) file is correctly configured, then other security contexts can be associated to the file as it is moved or copied (provided it is a valid context and specified in the [file_contexts](#) file).

The examples below show the effects of copying and moving files:

```
# These are the test files in the /root directory and their current security
# context:
#
-rw-r--r--  root root user_u:object_r:unconfined_t    copied-file
-rw-r--r--  root root user_u:object_r:unconfined_t    moved-file

# These are the commands used to copy / move the files:
#
# Standard copy file:
cp copied-file /usr/message_queue/in_queue

# Copy using -Z to set the files context:
cp -Z user_u:object_r:unconfined_t copied-file /usr/message_queue/in_queue/copied-
file-with-Z

# Standard move file:
mv moved-file /usr/message_queue/in_queue

# The target directory (/usr/message_queue/in_queue) is label "in_queue_t".
# The results of "ls -Z" on target the directory are:
#
-rw-r--r--  root root user_u:object_r:in_queue_t      copied-file
-rw-r--r--  root root user_u:object_r:unconfined_t    copied-file-with-Z
-rw-r--r--  root root user_u:object_r:unconfined_t    moved-file
```

However, if the `restorecond` daemon is running:

```
# If the restorecond daemon is running with a restorecond.conf file entry of:
#
/usr/message_queue/in_queue/*

# AND the file_context file has an entry of:
#
/usr/message_queue/in_queue(/.*)? -- system_u:object_r:in_file_t

# Then all the entries would be set as follows when the daemon detects the files
# creation:
#
-rw-r--r--  root root user_u:object_r:in_file_t      copied-file
-rw-r--r--  root root user_u:object_r:in_file_t      copied-file-with-Z
-rw-r--r--  root root user_u:object_r:in_file_t      moved-file

# This is because the restorecond process will set the contexts defined in
# the file_contexts file to the context specified as it is created in the
# new directory.
```

This is because the `restorecond` process will set the contexts defined in the `file_contexts` file to the context specified as it is created in the new directory.

2.8.3.2 Labeling Subjects

On a running GNU / Linux system, processes inherit the security context of the parent process. If the new process being spawned has permission to change its context, then a ‘type transition’ is allowed that is discussed in the [Domain Transition](#) section.

The [Initial Boot - Loading the Policy](#) section discusses how GNU / Linux is initialised and the processes labeled for the login process.

The policy language supports a number of statements to either assign labels to processes such as:

[user](#), [role](#) and [type](#) statements.

and manage their scope:

[role_allow](#) and [constrain](#)

and manage their transition:

[type_transition](#), [role_transition](#) and [range_transition](#)

One point to note is that the current Reference Policy does not support role transitions / changes as these are ‘constrained’ by the policy. To change to a different role, the `newrole(1)` command needs to be used (although there are ways around this as described in [Appendix G – Implementing a Constraint](#)).

2.8.4 Object Reuse

As GNU / Linux runs, it creates instances of objects and manages the information they contain (read, write, modify etc.) under the control of processes, and at some stage these objects may be deleted or released allowing the resource (such as memory blocks and disk space) to be available for reuse.

GNU / Linux handles object reuse by ensuring that when a resource is re-allocated, it is cleared. This means that when a process releases an object instance (e.g. release allocated memory back to the pool, delete a directory entry or file), there may be information left behind that could prove useful if harvested. If this should be an issue, then the process itself should clear or shred the information before releasing the object (which can be difficult in some cases unless the source code is available).

2.9 Domain and Object Transitions

This section discusses the [type_transition statement](#) that is used for:

1. Transition a process from one domain to another (a domain transition).
2. Transition an object from one type to another (an object transition or relabel).

These transitions can also be achieved using the [libselinux API](#) functions, however they are beyond the scope of this Notebook as is dynamically changing a processes security context using the `dyntransition` permission.

2.9.1 Domain Transition

A domain transition is where a process in one domain, transitions to another domain (i.e. runs under a different security context). There are two ways a process can request a domain transition in a policy:

1. Using a `type_transition` statement to perform a domain transition for programs that are not themselves SELinux-aware. This is the most common method and would be in the form of the following statement:

```
type_transition unconfined_t secure_services_exec_t : process ext_gateway_t;
```

2. SELinux-aware applications can specify the domain of the new process using the [libselinux API](#) call `setexeccon`. To achieve this the SELinux-aware application must also have the `setexec` permission by:

```
allow crond_t self : process setexec;
```

However, before any domain transition can take place the policy must specify that:

1. The source *domain* has permission to *transition* into the target domain.
2. The application binary file needs to be *executable* in the source domain.
3. The application binary file needs an *entry point* into the target domain.

The following is a `type_transition` statement taken from the example [message filter ext_gateway.conf](#) loadable module that will be used to explain the transition process¹³:

```
type_transition | source_domain | target_type          : class | target_domain;
-----▼-----▼-----▼-----
type_transition unconfined_t secure_services_exec_t : process ext_gateway_t;
```

This `type_transition` statement states that when a *process* running in the *unconfined_t* domain (the source domain) executes a file labeled *secure_services_exec_t*, the *process* should be changed to *ext_gateway_t* (the target domain) if allowed by the policy (i.e. transition from the *unconfined_t* domain to the *ext_gateway_t* domain).

However, as stated above to be able to *transition* to the *ext_gateway_t* domain, the following minimum permissions must be granted in the policy using [allow rules](#), where (note that the bullet numbers correspond to the numbers shown in [Figure 2-7](#)):

1. The *domain* needs permission to *transition* into the *ext_gateway_t* (target) domain:

```
allow unconfined_t ext_gateway_t : process transition;
```

2. The executable file needs to be *executable* in the *unconfined_t* (source) domain, and therefore also requires that the file is readable:

```
allow unconfined_t secure_services_exec_t : file { execute read getattr };
```

¹³ For reference, the external gateway uses a server application called `secure_server` that is transitioned into the `ext_gateway_t` domain from the `unconfined_t` domain. The `secure_server` executable is labeled `secure_services_exec_t`.

3. The executable file needs an *entry point* into the *ext_gateway_t* (target) domain:

```
allow ext_gateway_t secure_services_exec_t : file entrypoint;
```

These are shown in [Figure 2-7](#) where *unconfined_t* forks a child process, that then *exec's* the new program into a new domain called *ext_gateway_t*. Note that because the *type_transition* statement is being used, the transition is automatically carried out by the SELinux enabled kernel.

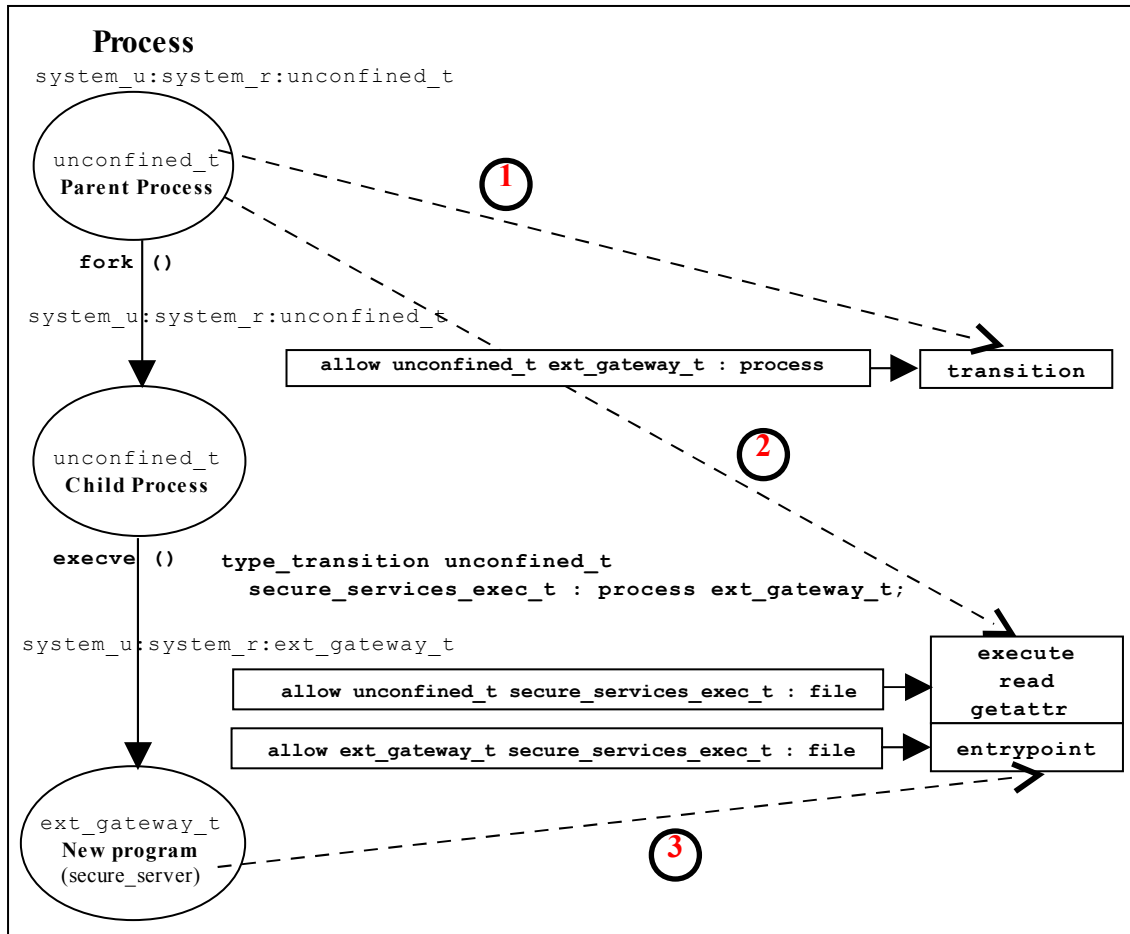


Figure 2-7: Domain Transition – Where the *secure_server* is executed within the *unconfined_t* domain and then transitioned to the *ext_gateway_t* domain.

2.9.1.1 Type Enforcement Rules

When building the `ext_gateway.conf` and `int_gateway.conf` modules the intention was to have both of these transition to their respective domains via *type_transition* statements. The `ext_gateway_t` statement would be:

```
type_transition unconfined_t secure_services_exec_t : process ext_gateway_t;
```

and the `int_gateway_t` statement would be:

```
type_transition unconfined_t secure_services_exec_t : process int_gateway_t;
```

However, when linking these two loadable modules into the policy, the following error was given:

```
semodule -v -s modular-test -i int_gateway.pp -i ext_gateway.pp
Attempting to install module 'int_gateway.pp':
Ok: return value of 0.
Attempting to install module 'ext_gateway.pp':
Ok: return value of 0.
Committing changes:
libsepol.expand_terule_helper: conflicting TE rule for (unconfined_t,
secure_services_exec_t:process): old was ext_gateway_t, new is int_gateway_t
libsepol.expand_module: Error during expand
libsemanage.semanage_expand_sandbox: Expand module failed
semodule: Failed!
```

This happened because the type enforcement rules will only handle a single ‘default’ type for a given source and target (see the [Type Enforcement Rules](#) section). In the above case there were two type_transition statements with the same source and target. The ext_gateway.conf module had the following statements:

```
# Allow the client/server to transition for the gateways:
allow unconfined_t ext_gateway_t : process { transition };
allow unconfined_t secure_services_exec_t : file { read execute getattr };
allow ext_gateway_t secure_services_exec_t : file { entrypoint };
type_transition unconfined_t secure_services_exec_t : process ext_gateway_t;
```

And the int_gateway.conf module had the following statements:

```
# Allow the client/server to transition for the gateways:
allow unconfined_t int_gateway_t : process { transition };
allow unconfined_t secure_services_exec_t : file { read execute getattr };
allow int_gateway_t secure_services_exec_t : file { entrypoint };
type_transition unconfined_t secure_services_exec_t : process int_gateway_t;
```

While the allow rules are valid to enable the transitions to proceed, the two type_transition statements had different ‘default’ types, that break the type enforcement rule.

It was decided to resolve this by:

1. Keeping the type_transition rule for the ‘default’ type of ext_gateway_t and allow the secure server process to be exec’ed from unconfined_t as shown in [Figure 2-7](#), by simply running the command from the prompt as follows:

```
# Run the external gateway 'secure server' application on port 9999 and
# let the policy transition the process to the ext_gateway_t domain:

secure_server 99999
```

2. Use the SELinux runcon(1) command to ensure that the internal gateway runs in the correct domain by running runcon from the prompt as follows:

```
# Run the internal gateway 'secure server' application on port 1111 and
# use runcon to transition the process to the int_gateway_t domain:

runcon -t int_gateway_t -r message_filter_r secure_server 1111

# Note - The role is required as a role transition is also defined in the
# policy.
```

The `runcon` command makes use of a number of [libselinux API](#) functions to check the current context and set up the new context (for example `getfilecon` is used to check the executable files context and `setexeccon` is used to ensure the `setexec` permission is allowed). If the all contexts are correct, then the `execvp(2)` system call is executed that `exec`'s the `secure_server` application with the argument of '1111' in the `int_gateway_t` domain with the `message_filter_r` role. The `runcon` source can be found in the `coreutils-6.12-20.f10.src.rpm` package.

Other ways to resolve this issue are:

1. Use the `runcon` command for both gateways to transition to their respective domains. The `type_transition` statements are therefore not required.
2. Use different names for the secure server executable files and ensure they have a different type (i.e. instead of `secure_service_exec_t` label the external gateway `ext_gateway_exec_t` and the internal gateway `int_gateway_exec_t`. This would involve making a copy of the application binary (which has already been done as part of the [module testing](#) by calling the server 'server' and labeling it `unconfined_t` and then making a copy called `secure_server` and labeling it `secure_services_exec_t`).
3. Implement the policy using the Reference Policy utilising the template interface principles discussed in the [template Macro](#) section.

It was decided to use `runcon` as it demonstrates the command usage better than reading the man pages, also did not want to use the reference policy.

2.9.2 Object Transition

An object transition is where an object needs to be relabeled, for example changing a files label from one type to another. There are two ways this can be achieved within policy:

1. Using a [type transition statement](#) to perform an object transition (relabel) for programs that are not SELinux-aware. This is the most common method and would be in the form of the following statement:

```
type_transition ext_gateway_t in_queue_t:file in_file_t;
```

2. Using a [type change statement](#) to perform an object transition for programs that are SELinux-aware.

```
type_change sysadm_t server_ptynode : chr_file sysadm_devpts_t;
```

The [libselinux API](#) call `security_compute_relabel` would be used to compute the new context.

The following details an object transition used in the [ext_gateway.conf](#) loadable module where by default, files would be labeled `in_queue_t` when created by the gateway application as this is the label attached to the parent directory as shown:

```
ls -Za /usr/message_queue/in_queue
drwxr-xr-x root root user_u:object_r:in_queue_t .
drwxr-xr-x root root system_u:object_r:unconfined_t ..
```

However the requirement is that files in the `in_queue` directory must be labeled `in_file_t`. To achieve this the files created must be relabeled to `in_file_t` by using a `type_transition` statement as follows:

```
# type_transition | source_domain | target_type : object | default_type;
-----▼-----▼-----▼-----
type_transition  ext_gateway_t  in_queue_t  : file      in_file_t;
```

This `type_transition` statement states that when a *process* running in the `ext_gateway_t` domain (the source domain) wants to create a *file* object in the directory that is labeled `in_queue_t`, the file should be relabeled `in_file_t` if allowed by the policy (i.e. label the file `in_file_t`).

However, as stated above to be able to relabel the file, the following minimum permissions need to be granted in the policy using [allow rules](#), where:

1. The source domain needs permission to *add file entries into the directory*:

```
allow ext_gateway_t in_queue_t : dir { write search add_name };
```

2. The source domain needs permission to *create file entries*:

```
allow ext_gateway_t in_file_t : file { write create getattr };
```

3. The policy can then ensure (via the SELinux kernel services) that files created in the `in_queue` are relabeled:

```
type_transition ext_gateway_t in_queue_t:file in_file_t;
```

An example output from a directory listing shows the resulting file labels:

```
ls -Za /usr/message_queue/in_queue
drwxr-xr-x root root user_u:object_r:in_queue_t .
drwxr-xr-x root root system_u:object_r:unconfined_t ..
-rw-r--r-- root root user_u:object_r:in_file_t Message-1
-rw-r--r-- root root user_u:object_r:in_file_t Message-2
```

2.10 Multi-Level Security and Multi-Category Security

As stated in the [Mandatory Access Control \(MAC\)](#) section as well as supporting Type Enforcement (TE), SELinux also supports MLS and MCS by adding an optional range entry to the security context. This section gives a brief introduction to MLS and MCS.

[Figure 2-8](#) shows a simple diagram where security levels represent the classification of files within a file server. The security levels are strictly hierarchical and conform to the [Bell-La Padula model](#) (BLP) in that (in the case of SELinux) a process (running at the ‘Confidential’ level) can read / write at their current level but only read down levels or write up levels (the assumption here is that the process is authorised).

This ensures confidentiality as the process can copy a file up to the secret level, but can never re-read that content unless the process ‘steps up to that level’, also the process cannot write files to the lower levels as confidential information would then drift downwards.

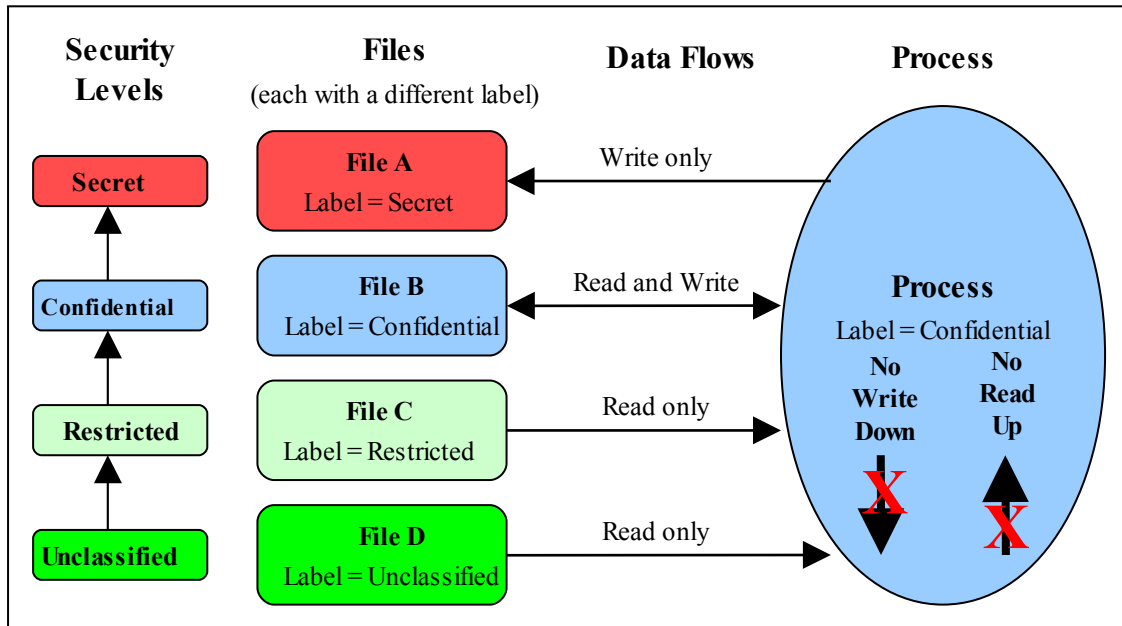


Figure 2-8: Security Levels and Data Flows – This shows how the process can only ‘Read Down’ and ‘Write Up’ within an MLS enabled system.

To achieve this level of control, the MLS extensions to SELinux make use of constraints similar to those described in the type enforcement [Constraints](#) section, except that the statement is called [mlsconstrain](#).

However, as always life is not so simple as:

1. Processes and objects can be given a range that represents the low and high security levels.
2. The security level can be more complex, in that it is a hierarchical sensitivity and zero or more non-hierarchical categories.
3. Allowing a process access to an object is managed by ‘dominance’ rules applied to the security levels.
4. Trusted processes can be given privileges that will allow them to bypass the BLP rules and basically do anything (that the security policy allowed of course).
5. Some objects do not support separate read / write functions as they need to read / respond in cases such as networks.

The sections that follow discuss the format of a security level and range, and how these are managed by the constraints mechanism within SELinux using the ‘dominance’ rules.

2.10.1 Security Levels

[Table 2-1](#) shows the components that make up a security level and how two security levels form a range that form the fourth and optional [: range] of the [security context](#) within an MLS / MCS environment.

The table also adds terminology in general use as other terms can be used that have the same meanings.

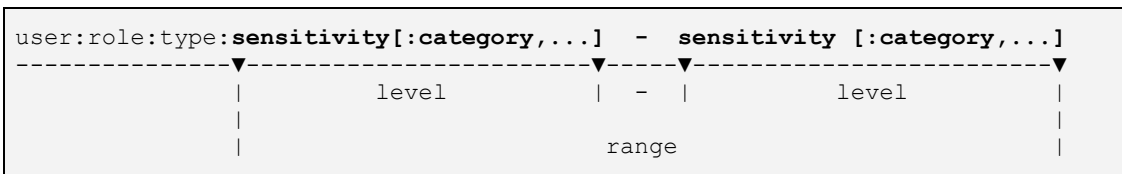
Security Level (or Level) Consisting of a sensitivity and zero or more category entries:	Note that SELinux uses <code>level</code> , <code>sensitivity</code> and <code>category</code> in the language statements (see the MLS Language Statements section), however when discussing these the following terms can also be used: labels, classifications, and compartments.	
<code>sensitivity [: category, ...]</code> also known as: Sensitivity Label Consisting of a classification and compartment.		
← Range →		
Low	-	High
<code>sensitivity [: category, ...]</code>		<code>sensitivity [: category, ...]</code>
For a process or subject this is the current level or sensitivity		For a process or subject this is the Clearance
For an object this is the current level or sensitivity		For an object this is the maximum range (for SELinux polyinstantiated directories)
SystemLow		SystemHigh
This is the lowest level or classification for the system (for SELinux this is generally 's0', note that there are no categories).		This is the highest level or classification for the system (for SELinux this is generally 's15:c0,c255', although note that they will be the highest set by the policy).

Table 2-1: Level, Label, Category or Compartment – *this table shows the meanings depending on the context being discussed.*

The format used in the policy language statements is fully described in the [MLS Statements](#) section, however a brief overview follows.

2.10.1.1 MLS / MCS Range Format

The following components (shown in bold) are used to define the MLS / MCS security levels within the security context:



Where:

`sensitivity` Sensitivity levels are hierarchical with (traditionally) `s0` being the lowest. These values are defined using the [sensitivity](#) language statement. To define their hierarchy, the [dominance](#) statement is used.

For MLS systems the highest sensitivity is the last one defined in the dominance statement (low to high). Traditionally the maximum for MLS systems is `s15` (although the maximum value for the [Reference Policy](#) is a compile time option).

For MCS systems there is only one sensitivity defined, and that is `s0`.

`category` Categories are optional (i.e. there can be zero or more categories) and they form unordered and unrelated lists of ‘compartments’. These values are defined using the [category](#) statement, where for example `c0.c3` represents a range (`c0 c1 c3`) and `c0, c3, c7` represent an unordered list. Traditionally the values are between `c0` and `c255` (although the maximum value for the [Reference Policy](#) is a compile time option).

`level` The level is a combination of the sensitivity and category values that form the actual security level. These values are defined using the [level](#) statement.

2.10.1.2 Translating Levels

When writing policy for MLS / MCS security level components it is usual to use an abbreviated form such as `s0, s1` etc. to represent sensitivities and `c0, c1` etc. to represent categories. This is done simply to conserve space as they are held on files as extended attributes and also in memory. So that these labels can be represented in human readable form a translation service is provided via the [setrans.conf](#) configuration file that uses the `mcstransd` daemon. For example `s0` = Unclassified, `s15` = TopSecret and `c0` = Finance, `c100` = SpyStories (unfortunately the translation does not support spaces in words). The `semanage(8)` command can be used to set up this translation and is shown in the [setrans.conf](#) configuration file section.

2.10.2 Managing Security Levels via Dominance Rules

As stated earlier, allowing a process access to an object is managed by ‘dominance’ rules applied to the security levels. These rules are as follows:

Security Level 1 dominates Security Level 2 - If the sensitivity of Security Level 1 is equal to or higher than the sensitivity of Security Level 2 and the categories of Security Level 1 are the same or a superset of the categories of Security Level 2.

Security Level 1 is dominated by Security Level 2 - If the sensitivity of Security Level 1 is equal to or lower than the sensitivity of Security Level 2 and the categories of Security Level 1 are a subset of the categories of Security Level 2.

Security Level 1 equals Security Level 2 - If the sensitivity of Security Level 1 is equal to Security Level 2 and the categories of Security Level 1 and Security Level 2 are the same set (sometimes expressed as: both Security Levels dominate each other).

Security Level 1 is incomparable to Security Level 2 - If the categories of Security Level 1 and Security Level 2 cannot be compared (i.e. neither Security Level dominates the other).

To illustrate the usage of these rules, [Table 2-2](#) lists the security level attributes in a table to show example files (or documents) that have been allocated labels such as `s3:c0`. The process that accesses these files (e.g. an editor) is running with a range of `s0 - s3:c1.c5` and has access to the files highlighted within the grey box area.

As the [MLS dominance statement](#) is used to enforce the sensitivity hierarchy, the security levels now follow that sequence (lowest = `s0` to highest = `s3`) with the categories being unordered lists of ‘compartments’. To allow the process access to files within its scope and within the dominance rules, the process will be constrained by using the [mlsconstrain statement](#) as illustrated in [Figure 2-9](#).

	Category →	c0	c1	c2	c3	c4	c5	c6	c7
s3	Secret	s3:c0					s3:c5	s3:c6	
s2	Confidential		s2:c1	s2:c2	s2:c3	s2:c4			s2:c7
s1	Restricted	s1:c0	s1:c1						s1:c7
s0	Unclassified	s0:c0			s0:c3				s0:c7
↑ Sensitivity	↑ Security Level (sensitivity:category) aka: classification	↑ File Labels ↑ A process running with a range of <code>s0 - s3:c1.c5</code> has access to the files within the grey boxed area.							

Table 2-2: MLS Security Levels – Showing the scope of a process running at a security range of `s0 - s3:c1.c5`.

Using [Figure 2-9](#):

- To allow write-up, the source level (l1) must be **dominated by** the target level (l2):

Source level = `s0:c3` or `s1:c1`

Target level = `s2:c1.c4`

As can be seen, either of the source levels are **dominated by** the target level.

- To allow read-down, the source level (l1) must **dominate** the target level (l2):

Source level = `s2:c1.c4`

Target level = s0:c3

As can be seen, the source level does **dominate** the target level.

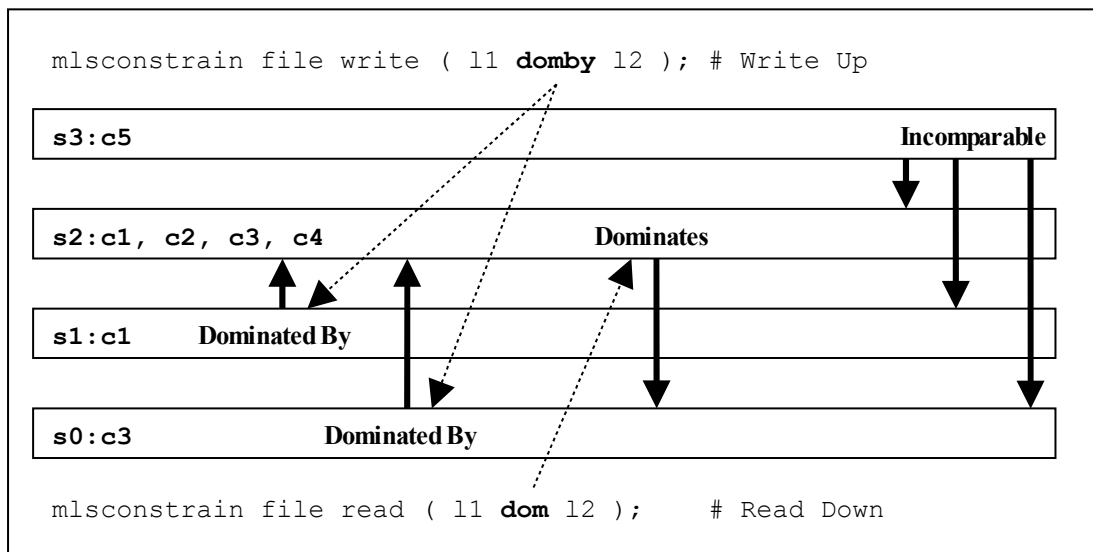


Figure 2-9: Showing the mlsconstrain Statements controlling Read Down & Write Up – This ties in with [Table 2-2](#) that shows a process running with a security range of s0 - s3:c1.c5.

However in the real world the SELinux MLS Reference Policy does not allow the write-up unless the process has a special privilege (by having the domain type added to an attribute), although it does allow the read-down. The default is to use `l1 eq l2` (i.e. the levels are equal). The reference policy MLS source file (`policy/mls`) shows these `mlsconstrain` statements.

2.10.3 MLS Labeled Network and Database Support

Networking for MLS is supported via the NetLabel CIPSO (commercial IP security option) service as discussed in the [SELinux Networking Support](#) section.

PostgreSQL supports labeling for MLS database services as discussed in the “Security-Enhanced PostgreSQL Security Guide” [Ref.3].

2.10.4 Common Criteria Certification

While the [Common Criteria](#) certification process is beyond the scope of this Notebook, it is worth highlighting that specific Red Hat GNU / Linux versions of software, running on specific hardware platforms with SELinux / MLS policy enabled, have passed the Common Criteria evaluation process. Note, for the evaluation (and deployment) the software and hardware are tied together, therefore whenever an update is carried out, an updated certificate should be obtained.

The Red Hat evaluation process cover the:

- Labeled Security Protection Profile ([LSPP](#)) – This describes how systems that implement security labels (i.e. MLS) should function.
- Controlled Access Protection Profile ([CAPP](#)) – This describes how systems that implement DAC should function.

An interesting point:

- Both Red Hat Linux 5.1 and Microsoft Server 2003 (with XP) have both been certified to EAL4+ , however while the evaluation levels may be the same the Protection Profiles that they were evaluated under were: Microsoft CAPP only, Red Hat CAPP and LSPP. Therefore always look at the protection profiles as they define what was actually evaluated.

2.11 Types of SELinux Policy

This section describes the different type of policy descriptions and versions that can be found within SELinux.

The types of SELinux policy can described in a number of ways:

1. Source code – These can be described as: [Example](#), [Reference Policy](#) or [Custom](#)
2. The source code descriptions or builds can also be sub-classified as: [Monolithic](#), [Base Module](#) or [Loadable Module](#).
3. Policies can also be described by the [type of policy functionality](#) they provide such as: `targeted`, `mls`, `mcs`, `standard`, `strict` or `minimum`.
4. Classified using language statements – These can be described as [Modular](#), [Optional](#) or [Conditional](#).
5. Binary policy (or kernel policy) – These can be described as [Monolithic](#), [Kernel Policy](#) or [Binary file](#).
6. Classification can also be on the ‘[policy version](#)’ used (examples are version 21, 22 and 23).

As can be seen the description of a policy can vary depending on the context.

2.11.1 Example Policy

The Example policy is the name used to describe the original SELinux policy source used to build a [monolithic](#)¹⁴ policy produced by the NSA and is now superseded by the [Reference Policy](#).

2.11.2 Reference Policy

Note that this section only gives an introduction to the reference policy, the installation, configuration and building of a policy using the source code is contained in Chapter 7 - [The Reference Policy](#).

The Reference Policy is now the standard policy source used to build SELinux policies, and its main aim is to provide a single source tree with supporting documentation that can be used to build policies for different purposes such as: confining important daemons, supporting MLS / MCS and locking down systems so that all processes are under SELinux control.

The Reference Policy is now used by all major distributions of SELinux, however each distribution makes its own specific changes to support their ‘version of the Reference

¹⁴ The term ‘monolithic’ generally means a single policy source is used to create the binary policy file that is then loaded as the ‘policy’ using the `checkpolicy(8)` command. However the term is sometimes used to refer to the binary policy file (as it is one file that describes the policy).

Policy'. For example, the F-10 distribution is based on a specific build of the standard Reference Policy that is then modified and distributed by Red Hat as an RPM. The release numbers will vary, however this Notebook uses:

```
selinux-policy-3.5.13-70.fc10.src.rpm
```

For information, the policy RPMs installed on the authors test machine for F-10 are as follows (the contents are explained in the main [Reference Policy](#) section):

```
selinux-policy-3.5.13-70.fc10.noarch
```

```
selinux-policy-doc-3.5.13-70.fc10.noarch
```

```
selinux-policy-minimum-3.5.13-70.fc10.noarch
```

```
selinux-policy-mls-3.5.13-70.fc10.noarch
```

```
selinux-policy-targeted-3.5.13-70.fc10.noarch
```

2.11.2.1 Policy Functionality Type

As can be seen from the reference policies distributed with F-10 above, they can also be classified by the functionality they support, for example the Red Hat policies:

`minimum` – supports a minimal set of confined daemons within their own domains. The remainder run in the `unconfined_t` space. Red Hat pre-configure MCS support within this policy.

`mls` – supports server based MLS systems.

`targeted` – supports a greater number of confined daemons and can also confine other areas and users (this targeted version also supports the older 'strict' version). Red Hat pre-configure MCS support within this policy.

For information, the Reference Policy from the Tresys repository supports the following types:

`standard` – supports confined daemons and can also confine other areas and users (this is an amalgamated version of the older 'targeted' and 'strict' versions).

`mcs` – As `standard` but supports MCS labels.

`mls` – supports MLS labels and confines server processes (as currently MLS does not support desktop applications (but should in F-12)).

2.11.3 Custom Policy

This generally refers to a policy source that is either:

1. A customised version of the Example policy.
2. A customised version of the Reference Policy (i.e. not the standard distribution version).
3. A policy that has been built using the language statements to build a specific policy such as those shown in the [Building a Basic Policy](#) section.

2.11.4 Monolithic Policy

A Monolithic policy is an SELinux policy that is compiled from one source file called `policy.conf` (i.e. it does not use the [Loadable Module Policy](#) statements and infrastructure which therefore makes it suitable for embedded systems as there is no policy store overhead).

An example monolithic policy is the NSAs original [Example Policy](#). A simple monolithic policy is shown in the [Building the Monolithic Policy](#) section and [Table 4-1](#) shows the order of language statements that can be in a source file.

Monolithic policies are compiled using the `checkpolicy` (8) SELinux command.

The Reference Policy supports the building of monolithic policies.

In some cases the policy binary file (see the [Binary Policy](#) section) is also called a monolithic policy.

2.11.5 Loadable Module Policy

The loadable module infrastructure allows policy to be managed on a modular basis, in that there is a base policy module that contains all the core components of the policy (i.e. the policy that should always be present), and zero or more modules that can be loaded and unloaded as required (for example if there is a module to enforce policy for ftp, but ftp is not used, then that module could be unloaded).

There are number of parts that form the infrastructure:

1. Policy source code that is constructed for a modular policy with a base module and optional loadable modules.
2. Utilities to compile and link modules and place them into a ‘policy store’.
3. Utilities to manage the modules and associated configuration files within the ‘policy store’.

[Figure 2-2](#) shows these components along the top of the diagram. The files contained in the policy store are detailed in the [Policy Store Configuration Files](#) section.

The policy language was extended to handle loadable modules as detailed in the [Policy Support Statements](#) section. For a detailed overview on how the modular policy is built into the final [binary policy](#) for loading into the kernel, see “[SELinux Policy Module Primer](#)” [Ref. 4].

2.11.5.1 Optional Policy

The loadable module policy infrastructure supports an [optional policy statement](#) that allows policy rules to be defined but only enabled in the binary policy once the conditions have been satisfied. The example loadable modules shown in the [Building a Basic Policy](#) section use this feature.

2.11.6 Conditional Policy

Conditional policies can be implemented in monolithic or loadable module policies and allow policy to be enabled or not depending on the state of a boolean flag. This is often used to enable or disable features within the policy (i.e. change the policy enforcement rules).

The boolean flag status is held in kernel and can be changed using the `setsebool(8)` command either persistently across system re-boots or temporarily (i.e. only valid until a re-boot). The following example shows a persistent conditional policy change:

```
setsebool -P ext_gateway_audit=false
```

The conditional policy language statements are the [bool Statement](#) that defines the boolean flag identifier and its initial status, and the [if Statement](#) that allows certain rules to be executed depending on the state of the boolean value or values.

2.11.7 Binary Policy

The binary policy is the policy file that is loaded into the kernel and is always located at (on F-10) `/etc/selinux/<policy_name>/policy` and is called `policy.<version>`. Where `<policy_name>` is the policy name specified in the SELinux configuration file `/etc/selinux/config` and `<version>` is the SELinux [policy version](#) supported by the kernel and SELinux tools.

The binary policy can be built from source files supplied by the [Example Policy](#), the [Reference Policy](#) or custom built source files as described in the [Building a Basic Policy](#) section.

An example `/etc/selinux/config` file is shown below where the **SELINUXTYPE=targeted** entry identifies the `<policy_name>` that will be used to locate and load the active policy:

```
SELINUX=permissive  
  
SELINUXTYPE=targeted
```

From the above example, the actual binary policy file would be located at `/etc/selinux/targeted/policy` and be called `policy.23` (as version 23 is supported by F-10):

```
/etc/selinux/targeted/policy/policy.23
```

2.11.8 Policy Versions

SELinux has a policy database (built by the `libsepol` library) that describes the format of data held within a [binary policy](#), however, if any new features are added to SELinux (generally language extensions) this can result in a change to the policy database. Whenever the policy database is updated, the policy version is incremented.

The `sestatus(8)` command will show the current policy version number in its output as follows:

```
SELinux status:          enabled  
SELinuxfs mount:        /selinux  
Current mode:           enforcing  
Mode from config file:  permissive  
Policy version:        23
```



```
Policy from config file:    modular-test
```

The F-10 policy version is '23' with [Table 2-3](#) describing the different versions. There is also another version that applies to the modular policy, however the main policy database version is the one that is generally quoted (some SELinux utilities (e.g. apol) give both version numbers).

<i>policy db Version</i>	<i>modular db Version</i>	<i>Description</i>
15	4	The base version (when SELinux was merged into the kernel).
16	-	Added Conditional Policy support (the <code>bool</code> feature).
17	-	Added support for IPv6.
18	-	Added netlink support.
19	5	Added MLS support, plus the validate_trans_Statement .
20	-	Reduced the size of the access vector table.
21	6	Added support for the MLS range_transition_Statement .
22	7	Added policy capabilities .
23	8	Added support for the permissive Statement .

Table 2-3: Policy version descriptions

2.12 SELinux Permissive and Enforcing Modes

SELinux has three major modes of operation:

Enforcing – SELinux is enforcing the loaded policy.

Permissive – SELinux has loaded the policy, however it is not enforcing the policy. This is generally used for testing as the audit log will contain the AVC denied messages as defined in the [Audit Logs](#) section. The SELinux utilities such as `audit2allow(1)` and `audit2why(8)` can then be used to determine the cause and possible resolution by generating the appropriate allow rules.

Disabled – The SELinux infrastructure (in the kernel) is not loaded.

These flags are set in the `/etc/selinux/config` file as described in the [Global Configuration Files](#) section.

There is another method for running specific domains in permissive mode using the [permissive statement](#). This can be used directly in a user written loadable module or `semanage(8)` will generate the appropriate module and load it using the following example command:

```
# This example will add a new module in
# /etc/selinux/<policy_name> # /modules/active/modules/permissive_unconfined_t.pp
# and then reload the policy:

semanage permissive -a unconfined_t
```

The `sestatus` (8) command will show the current policy mode in its output as follows:

```
SELinux status:                enabled
SELinuxfs mount:              /selinux
Current mode:                permissive
Mode from config file:        enforcing
Policy version:                23
Policy from config file:      modular-test
```

2.13 Audit Logs

For SELinux there are two main types of audit event:

1. [SELinux-aware Application Events](#) – These are generated by the SELinux kernel services and SELinux-aware applications for events such as system errors, initialisation, policy load, changing boolean states, setting of enforcing / permissive mode and relabeling.
2. [AVC Audit Events](#) – These are generated by the AVC subsystem as the result of access denials, or where specific events have requested an audit message (i.e. where an [auditallow rule](#) has been used in the policy).

The audit and event messages can be stored in one of two places (in F-10 anyway):

1. The system log located at `/var/log/messages` that contains the `syslog` boot and runtime events. The AVC messages logged here are those generated by SELinux before the audit daemon has been loaded, as F-10 uses the audit framework (`auditd`) as standard. However, some SELinux-aware audit messages are logged here as well¹⁵. Note that the detailed SELinux kernel boot events are logged in the `/var/log/dmesg` file.
2. The audit log located at `/var/log/audit/audit.log`. Audit events that take place after the audit daemon has been loaded are in this log file as are some SELinux system messages. The AVC audit messages of interest are those starting with `type=AVC` and are described in the [AVC Audit Events](#) section.

2.13.1 SELinux-aware Application Events

For SELinux and SELinux-aware applications (excluding the AVC entries that are discussed in the [AVC Audit Events](#) section) the following are possible `type=` entries within the `audit.log` file that have been seen:

```
# Example audit.log entries for adding a new SELinux user with:
#
#     semanage user -a -R staff_r -P STAFF test_u
#
# Note that the audit messages appear in the following sequence:
#
# 1) That USER_AVC has received a policy reload message.
```

¹⁵ For example if the iptables are loaded and there are SECMARK security contexts present, BUT the contexts are invalid (i.e. not in the policy), then the event is logged in the `messages` log and nothing will appear in the audit log. Interestingly the NetLabel code allows any context to be added provided it is correctly formatted as it is checked when used.

```
# 2) A notice that the policy is changing by MAC_POLICY_LOAD.
# Note that this event is followed by another audit message
# with type=SYSCALL with further information. These two
# events are tied by having the same serial_number in the
# msg=audit(time:serial_number) field.
# 3) That semanage carried out a USER_ROLE_CHANGE.

type=USER_AVC msg=audit(1243855640.568:29): user pid=1543 uid=81 auid=4294967295
ses=4294967295 subj=system_u:system_r:system_dbusd_t:s0-s0:c0.c1023 msg='avc:
received policyload notice (seqno=3) : exe="?" (saudit=81, hostname=?, addr=?,
terminal=?)'

type=MAC_POLICY_LOAD msg=audit(1243855634.660:30): policy loaded auid=0 ses=1

type=SYSCALL msg=audit(1243855634.660:30): arch=40000003 syscall=4 success=yes
exit=3480819 a0=4 a1=b7c74000 a2=351cf3 a3=bfbd9dc8 items=0 ppid=2731 pid=2732
auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=1
comm="load_policy" exe="/usr/sbin/load_policy"
subj=unconfined_u:unconfined_r:load_policy_t:s0-s0:c0.c1023 key=(null)

type=USER_ROLE_CHANGE msg=audit(1243855646.618:31): user pid=2731 uid=0 auid=0
ses=1 subj=unconfined_u:unconfined_r:semanage_t:s0-s0:c0.c1023 msg='op=add SELinux
user record acct="test_u" old-seuser=? old-role=? old-range=? new-seuser=test_u
new-role=staff_r new-range=? exe=/usr/sbin/semanage (hostname=?, addr=?,
terminal=pts/0 res=success)'
```

```
# Example audit.log entries for setting enforcing mode by:
#
#       setenforce 1
#
# Note that the audit messages appear in the following sequence:
#
type=MAC_STATUS msg=audit(1243856597.296:32): enforcing=1 old_enforcing=0 auid=0
ses=1

type=USER_AVC msg=audit(1243856597.312:33): user pid=1543 uid=81 auid=4294967295
ses=4294967295 subj=system_u:system_r:system_dbusd_t:s0-s0:c0.c1023 msg='avc:
received setenforce notice (enforcing=1) : exe="?" (saudit=81, hostname=?, addr=?,
terminal=?)'

type=SYSCALL msg=audit(1243856597.296:32): arch=40000003 syscall=4 success=yes
exit=1 a0=3 a1=bf95b554 a2=1 a3=bf95b554 items=0 ppid=2643 pid=2761 auid=0 uid=0
gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=1 comm="setenforce"
exe="/usr/sbin/setenforce" subj=unconfined_u:unconfined_r:unconfined_t:s0-
s0:c0.c1023 key=(null)
```

Other entries are can of course appear in the `audit.log` file, however they are not covered in this Notebook.

It should be noted that entries placed in the `/var/log/messages` file before the audit daemon is loaded are in a different format, for example:

```
# This is an example entry of an AVC denial and corresponding
# SYSCALL from the /var/log/messages file that were captured
# before the audit daemon was loaded:

May 26 12:34:16 localhost kernel: type=1400 audit(1243337656.638:1358): avc:
denied { read } for pid=3033 comm="rsyslogd" path="/proc/kmsg" dev=proc
ino=4026531848 scontext=system_u:system_r:syslogd_t:s0
tcontext=system_u:object_r:unlabeled_t:s0 tclass=file

May 26 12:34:16 localhost kernel: type=1300 audit(1243337656.638:1358):
arch=40000003 syscall=3 success=yes exit=230 a0=7 a1=11f7e0 a2=fff a3=11f7e0
items=0 ppid=1 pid=3033 auid=4294967295 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0
sgid=0 fsgid=0 tty=(none) ses=4294967295 comm="rsyslogd" exe="/sbin/rsyslogd"
subj=system_u:system_r:syslogd_t:s0 key=(null)
```

The more detailed SELinux boot time messages are placed in the `/var/log/dmesg` file where selected SELinux entries are as follows:

```

Security Framework initialized
SELinux: Initializing.
SELinux: Starting in permissive mode
...
SELinux: Registering netfilter hooks
....
SELinux: 8192 avtab hash slots, 113530 rules.
SELinux: 8192 avtab hash slots, 113530 rules.
SELinux: 9 users, 11 roles, 2608 types, 122 bools, 1 sens, 1024 cats
SELinux: 73 classes, 113530 rules
SELinux: Completing initialization.
SELinux: Setting up existing superblocks.
SELinux: initialized (dev dm-0, type ext3), uses xattr
SELinux: initialized (dev tmpfs, type tmpfs), uses transition SIDs
SELinux: initialized (dev usbfs, type usbfs), uses genfs_contexts
SELinux: initialized (dev selinuxfs, type selinuxfs), uses genfs_contexts
SELinux: initialized (dev mqueue, type mqueue), uses transition SIDs
....
SELinux: initialized (dev sockfs, type sockfs), uses task SIDs
...
SELinux: initialized (dev rootfs, type rootfs), uses genfs_contexts
SELinux: initialized (dev sysfs, type sysfs), uses genfs_contexts
type=1403 audit(1243839417.933:2): policy loaded auid=4294967295 ses=4294967295
...
...
SELinux: initialized (dev sdal, type ext3), uses xattr
SELinux: initialized (dev tmpfs, type tmpfs), uses transition SIDs
SELinux: Context user_u:unconfined_r:unconfined_t is not valid (left unmapped).
...
...
SELinux: initialized (dev binfmt_misc, type binfmt_misc), uses genfs_contexts
    
```

2.13.2 AVC Audit Events

[Table 2-4](#) shows the general format of AVC audit message within the `/var/log/audit/audit.log` file. These appear when access has been denied or an audit event has been specifically requested.

<i>Keyword</i>	<i>Description</i>
type	<p>For SELinux AVC events this will always be:</p> <pre>type=AVC</pre> <p>Note that once the AVC event has been logged, another event with <code>type=SYSCALL</code> will always follow that contains further information regarding the event.</p> <p>The AVC event can always be tied to the relevant SYSCALL event as they have the same <code>serial_number</code> in the <code>msg=audit(time:serial_number)</code> field as shown in the following example:</p> <pre> type=AVC msg=audit(1243332701.744:101): avc: denied { getattr } for pid=2714 comm="ls" path="/usr/lib/locale/locale-archive" dev=dm-0 ino=353593 scontext=system_u:object_r:unlabeled_t:s0 tcontext=system_u:object_r:locale_t:s0 tclass=file type=SYSCALL msg=audit(1243332701.744:101): arch=40000003 syscall=197 success=yes exit=0 a0=3 a1=553ac0 a2=552ff4 a3=bfc5eab0 items=0 ppid=2671 pid=2714 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts1 ses=1 comm="ls" </pre>

<i>Keyword</i>	<i>Description</i>
	<code>exe="/bin/ls" subj=system_u:object_r:unlabeled_t:s0 key=(null)</code>
msg	This will contain the audit keyword with a reference number (e.g. <code>msg=audit(1243332701.744:101)</code>)
avc	<p>This will be either denied when access has been denied or granted when the auditallow rule has been executed by the AVC system.</p> <p>The entries that follow the <code>avc=</code> field depend on what type of event is being audited. Those shown below are generated by the kernel AVC audit function, however the user space AVC audit function will return fields relevant to the application being managed by their Object Manager.</p>
pid	If a task, then log the process id (<code>pid</code>) and the name of the executable file (<code>comm</code>).
comm	
key	If an IPC event then log the identifier.
capability	If a Capability event then log the identifier.
path	If a File System event then log the relevant information. Note that the name field may not always be present.
name	
dev	
ino	
laddr	If a Socket event then log the Source / Destination addresses and ports for IP4 or IP6 sockets (<code>AF_INET</code>).
lport	
faddr	
fport	
path	If a File Socket event then log the path (<code>AF_UNIX</code>).
saddr	If a Network event then log the Source / Destination addresses and ports with the network interface for IP4 or IP6 networks (<code>AF_INET</code>).
src	
daddr	
dest	
netif	
scontext	The security context of the source or subject.
tcontext	The security context of the target or object.
tclass	The object class of the target or object.

Table 2-4: AVC Audit Message Description – *The keywords in bold are in all AVC audit messages, the others depend on the type of event being audited.*

Example `audit.log` denied and granted events are shown in the following examples:

```
# This is an example denied message note that there are two
# type=AVC calls, but only one corresponding type=SYSCALL entry.

type=AVC msg=audit(1242575005.122:101): avc: denied { rename } for pid=2508
comm="canberra-gtk-pl" name="c73a516004b572d8c845c74c49b2511d:runtime.tmp" dev=dm-
0 ino=188999 scontext=test_u:staff_r:oddjjob_mkhomedir_t:s0
tcontext=test_u:object_r:gnome_home_t:s0 tclass=lnk_file

type=AVC msg=audit(1242575005.122:101): avc: denied { unlink } for pid=2508
comm="canberra-gtk-pl" name="c73a516004b572d8c845c74c49b2511d:runtime" dev=dm-0
ino=188578 scontext=test_u:staff_r:oddjjob_mkhomedir_t:s0
tcontext=system_u:object_r:gnome_home_t:s0 tclass=lnk_file

type=SYSCALL msg=audit(1242575005.122:101): arch=40000003 syscall=38 success=yes
exit=0 a0=82d2760 a1=82d2850 a2=da6660 a3=82cb550 items=0 ppid=2179 pid=2508
auid=500 uid=500 gid=500 euid=500 suid=500 fsuid=500 egid=500 sgid=500 fsgid=500
tty=(none) ses=1 comm="canberra-gtk-pl" exe="/usr/bin/canberra-gtk-play"
subj=test_u:staff_r:oddjjob_mkhomedir_t:s0 key=(null)
```

```
# This is an example granted audit message:

type=AVC msg=audit(1239116352.727:311): avc: granted { transition } for
pid=7687 comm="bash" path="/usr/move_file/move_file_c" dev=dm-0 ino=402139
scontext=user_u:unconfined_r:unconfined_t tcontext=user_u:unconfined_r:move_file_t
tclass=process

type=SYSCALL msg=audit(1239116352.727:311): arch=40000003 syscall=11 success=yes
exit=0 a0=8a6ea98 a1=8a56fa8 a2=8a578e8 a3=0 items=0 ppid=2660 pid=7687 auid=0
uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=(none) ses=1
comm="move_file_c" exe="/usr/move_file/move_file_c"
subj=user_u:unconfined_r:move_file_t key=(null)
```

2.14 Polyinstantiation

GNU / Linux supports the polyinstantiation of directories (sockets are not yet supported in F-10) that can be utilised by SELinux via the Pluggable Authentication Module (PAM) that is explained in the next section. The “[Polyinstantiation of directories in an SELinux system](#)” [Ref. 5] also gives a more detailed overview of the subject.

2.14.1 Polyinstantiation support in PAM

PAM supports polyinstantiation of directories at login time using the Shared Subtree / Namespace services available within GNU / Linux (the `namespace.conf(5)` man page is also a good reference). Note that PAM and Namespace services are SELinux-aware.

The default installation of F-10 does not enable polyinstantiated directories, therefore this section will show the configuration required to enable the feature and some [examples](#).

To implement polyinstantiated directories PAM requires the following files to be configured:

1. A `pam_namespace` module entry added to the appropriate `/etc/pam.d/` login configuration file (e.g. `login`, `sshd`, `gdm` etc.). F-10 already has these entries configured, with an example `/etc/pam.d/gdm` file being:

```
##%PAM-1.0
auth [success=done ignore=ignore default=bad] pam_selinux_permit.so
# auth required pam_succeed_if.so user != root quiet
```

auth	required	pam_env.so
auth	substack	system-auth
auth	optional	pam_gnome_keyring.so
account	required	pam_nologin.so
account	include	system-auth
password	include	system-auth
session	required	pam_selinux.so close
session	required	pam_loginuid.so
session	optional	pam_console.so
session	required	pam_selinux.so open
session	optional	pam_keyinit.so force revoke
session	required	pam_namespace.so
session	optional	pam_gnome_keyring.so auto_start
session	include	system-auth

2. Entries added to the `/etc/security/namespace.conf` file that defines the directories to be polyinstantiated by PAM (and other services that may need to use the namespace service). The entries are explained in the [namespace.conf Configuration File](#) section, with the default entries in F-10 being (note that the entries are commented out in the distribution):

#polydir	instance-prefix	method	list_of_uids
/tmp	/tmp-inst/	level	root,adm
/var/tmp	/var/tmp/tmp-inst/	level	root,adm
\$HOME	\$HOME/\$USER.inst/	level	

Once these files have been configured and a user logs in (although not `root` or `adm` in the above example), the PAM `pam_namespace` module would unshare the current namespace from the parent and mount namespaces according to the rules defined in the `namespace.conf` file. The F-10 configuration also includes an `/etc/security/namespace.init` script that is used to initialise the namespace every time a new directory instance is set-up. This script receives four parameters: the polyinstantiated directory path, the instance directory path, a flag to indicate if new instance, and the user name. If a new instance is being set up, the directory permissions are set and the `restorecon(8)` command is run to set the correct file contexts.

2.14.1.1 namespace.conf Configuration File

Each line in the `namespace.conf` file is formatted as follows:

```
polydir instance_prefix method list_of_uids
```

Where:

- `polydir` The absolute path name of the directory to polyinstantiate. The optional strings `$USER` and `$HOME` will be replaced by the user name and home directory respectively.
- `instance_prefix` A string prefix used to build the pathname for the polyinstantiated directory. The optional strings `$USER` and `$HOME` will be replaced by the user name and home directory respectively.
- `method` This is used to determine the method of polyinstantiation with valid entries being:

`user` - Polyinstantiation is based on user name.

`level` - Polyinstantiation is based on the user name and MLS level.

`context` - Polyinstantiation is based on the user name and security context.

Note that `level` and `context` are only valid for SELinux enabled systems.

`list_of_uids`

A comma separated list of user names that will not have polyinstantiated directories. If blank, then all users are polyinstantiated. If the list is preceded with an '~' character, then only the users in the list will have polyinstantiated directories.

There are a number of optional flags available that are described in the `namespace.conf(5)` man page.

2.14.1.2 Example Configurations

This section shows two sample `namespace.conf` configurations, the first uses the `method=user` and the second `method=context`.

Example 1 - `method=user`:

1. Set the `/etc/security/namespace.conf` entries as follows:

#polydir	instance-prefix	method	list_of_uids
/tmp	/tmp-inst/	user	root,adm
/var/tmp	/var/tmp/tmp-inst/	user	root,adm
\$HOME	\$HOME/\$USER.inst/	user	

2. Login as a normal user (`rch` in this example) and the PAM / Namespace process will build the following polyinstantiated directories:

```
# The directories will contain the user name as a part of
# the polyinstantiated directory name as follows:

# /tmp
/tmp/tmp-inst/rch

# /var/tmp:
/var/tmp/tmp-inst/rch

# $HOME
/home/rch/rch-inst/rch
```

Example 2 - `method=context`:

1. Set the `/etc/security/namespace.conf` entries as follows:

#polydir	instance-prefix	method	list_of_uids
/tmp	/tmp-inst/	context	root,adm


```
/var/tmp      /var/tmp/tmp-inst/    context    root,adm
$HOME        $HOME/$USER.inst/    context
```

2. Login as a normal user (rch in this example) and the PAM / Namespace process will build the following polyinstantiated directories:

```
# The directories will contain the security context and
# user name as a part of the polyinstantiated directory
# name as follows:

# /tmp
/tmp/tmp-inst/user_u:unconfined_r:unconfined_t_rch

# /var/tmp:
/var/tmp/tmp-inst/user_u:unconfined_r:unconfined_t_rch

# $HOME
/home/rch/rch-inst/user_u:unconfined_r:unconfined_t_rch
```

2.14.2 Polyinstantiated Objects

Polyinstantiation is supported by SELinux using the [type_member Statement](#). This statement is not limited to specific object classes, however GNU / Linux currently only supports dir type objects.

The following libselinux API functions support polyinstantiation as detailed in [Appendix C – API Summary for libselinux](#):

```
avc_compute_member
security_compute_member
security_compute_member_raw
```

2.14.3 Polyinstantiation support in the Reference Policy

The reference policy `files.te` and `files.if` modules (in the kernel layer) support polyinstantiated directories. There is also a global tunable (a boolean called `allow_polyinstantiation`) that can be used to set this functionality on or off during login. By default this boolean is set `false` (off).

2.15 PAM Login Process

Applications used to provide login services (such as `gdm` and `ssh`) in F-10 use the PAM (Pluggable Authentication Modules) infrastructure to provide the following services:

Account Management – This manages services such as password expiry, service entitlement (i.e. what services the login process is allowed to access).

Authentication Management – Authenticate the user or subject and set up the credentials. PAM can handle a variety of devices including smart-cards and biometric devices.

Password Management – Manages password updates as needed by the specific authentication mechanism being used and the password policy.

Session Management – Manages any services that must be invoked before the login process completes and / or when the login process terminates. For SELinux this is where hooks are used to manage the domains the subject may enter.

The `pam` and `pam.conf` man pages describe the services and configuration in detail and only a summary is provided here covering the SELinux services.

The PAM configuration for F-10 is managed by a number of files located in the `/etc/pam.d` directory which has configuration files for login services such as: `gdm`, `gdm-autologin`, `login`, `remote` and `sshd`, and at various points in this Notebook the `gdm` configuration file has been modified to allow root login and the `pam_namespace.so` module used to manage polyinstantiated directories for users.

There are also a number of PAM related configuration files in `/etc/security`, although only one is directly related to SELinux that is described in the [/etc/security/sepermit.conf file](#) section.

The main login service related PAM configuration files (e.g. `gdm`) consist of multiple lines of information that are formatted as follows:

```
service type control module-path arguments
```

Where:

<code>service</code>	The service name such as <code>gdm</code> and <code>login</code> reflecting the login application. If there is a <code>/etc/pam.d</code> directory, then this is the name of a configuration file name under this directory. Alternatively, a configuration file called <code>/etc/pam.conf</code> can be used. F-10 uses the <code>/etc/pam.d</code> configuration.
<code>type</code>	These are the management groups used by PAM with valid entries being: <code>account</code> , <code>auth</code> , <code>password</code> and <code>session</code> that correspond to the descriptions given above. Where there are multiple entries of the same ‘type’, the order they appear could be significant.
<code>control</code>	This entry states how the module should behave when the requested task fails. There can be two formats: a single keyword such as <code>required</code> , <code>optional</code> , and <code>include</code> ; or multiple space separated entries enclosed in square brackets consisting of: <pre>[value1=action1 value2=action2 ..]</pre> Both formats are shown in the example file below, however see the <code>pam.conf</code> man pages for the gory details.
<code>module-path</code>	Either the full path name of the module or its location relative to <code>/lib/security</code> (but does depend on the system architecture).
<code>arguments</code>	A space separated list of the arguments that are defined for the

module.

An example PAM configuration file is as follows, although note that the ‘service’ parameter is actually the file name because F-10 uses the /etc/pam.d directory configuration (in this case gdm for the Gnome login service).

```
# /etc/pam.d/gdm configuration rule entry.
# SERVICE = file name (gdm)
# TYPE      CONTROL    PATH      ARGUMENTS

#%PAM-1.0
auth        [success=done ignore=ignore default=bad] pam_selinux_permit.so
# auth      required    pam_succeed_if.so user != root quiet
auth        required    pam_env.so
auth        substack    system-auth
auth        optional    pam_gnome_keyring.so
account     required    pam_nologin.so
account     include     system-auth
password    include     system-auth
session    required    pam_selinux.so close
session     required    pam_loginuid.so
session     optional    pam_console.so
session    required    pam_selinux.so open
session     optional    pam_keyinit.so force revoke
session     required    pam_namespace.so
session     optional    pam_gnome_keyring.so auto_start
session     include     system-auth
```

The core services are provided by PAM, however other library modules can be written to manage specific services such as support for SELinux. The SELinux PAM modules use the `libselinux` API to obtain its configuration information and the three SELinux PAM entries highlighted in the above configuration file perform the following functions:

pam_selinux_permit.so - Allows pre-defined users the ability to logon without a password provided that SELinux is in enforcing mode (see the [/etc/security/sepermit.conf](#) file section).

pam_selinux.so open - Allows a security context to be set up for the user at initial logon (as all programs exec'ed from here will use this context). How the context is retrieved is described in the [seusers configuration file](#) section.

pam_selinux.so close - This will reset the login programs context to the context defined in the policy.

2.16 Linux Security Module and SELinux

This section gives a high level overview of the LSM and SELinux internal structure and workings. A more detailed view can be found in the “[Implementing SELinux as a Linux Security Module](#)” [Ref. 6] that was used extensively to develop this section (with the SELinux kernel source code). The major areas covered are:

1. How the LSM and SELinux modules work together.
2. The major SELinux internal services.
3. The fork system call and exec are followed through as an example to tie in with the transition process covered in the [Domain Transition](#) section.
4. The SELinux filesystem `/selinux`.

5. The boot sequences that are relevant to SELinux.

2.16.1 The LSM Module

The LSM is the Linux security framework that allows 3rd party access control mechanisms to be linked into the GNU / Linux kernel. Currently there are two 3rd party services that utilise the LSM: SELinux and SMACK (Simplified Mandatory Access Control Kernel) that both provide mandatory access control services. Details regarding SMACK can be found at: <http://www.schaufler-ca.com/>.

The basic idea behind the LSM is to:

1. Insert security function calls (or hooks) and security data structures in the various kernel services to allow access control to be applied over and above that already implemented via DAC. The type of service that have hooks inserted are shown in [Table 2-5](#) with an example task and program execution shown in the [Fork Walk-thorough](#) and [Process Transition Walk-thorough](#) sections.
2. Allow registration and initialisation services for the 3rd party security modules.
3. Allow process security attributes to be available to userspace services by extending the /proc filesystem with a security namespace.
4. Support filesystems that use extended attributes (SELinux uses `security.selinux` as explained in the [Labeling Extended Attribute Filesystems](#) section).
5. Consolidate the Linux capabilities into an optional module.

It should be noted that the LSM does not provide any security services itself, only the hooks and structures for supporting 3rd party modules. If no 3rd party module is loaded, the capabilities module becomes the default module thus allowing standard DAC access control.

Program execution	Filesystem operations	Inode operations
File operations	Task operations	Netlink messaging
Unix domain networking	Socket operations	XFRM operations
Key Management operations	IPC operations	Memory Segments
Semaphores	Capability	Sysctl
Syslog	Audit	

Table 2-5: LSM Hooks - *These are the kernel services that LSM has inserted security hooks and structures to allow access control to be managed by 3rd party modules (see `./kernel-2.6.27/include/linux/security.h`).*

The major kernel source files (relative to `./kernel-2.6.27/security`) that form the LSM are shown in [Table 2-6](#). However there is one major header file (`include/linux/security.h`) that describes all the security hooks and structures defined by the LSM.

Name	Function
<code>capability.c</code>	Some capability functions were in various kernel modules have been consolidated into these source files. These are now (from Kernel 2.6.27)
<code>commoncap.c</code>	

Name	Function
device_cgroup.c	always linked into the kernel. This means the dummy.c security module (mentioned in [Ref. 6]) is no longer required.
inode.c	This allows the 3 rd party security module to initialise a security filesystem. In the case of SELinux this would be /selinux that is defined in the selinux/selinuxfs.c source file.
root_plug.c	This is a sample 3 rd party module and therefore not used.
security.c	Contains the LSM framework initialisation services that will set up the hooks described in security.h and those in the capability source files. It also provides functions to initialise 3 rd party modules.

Table 2-6: The core LSM source modules.

2.16.2 The SELinux Module

This section does not go into detail of all the SELinux module functionality as [Ref 6] does this, however it attempts to highlight the way some areas work by using the [fork and transition process example](#) described in the [Domain Transition](#) section and also by describing the [boot process](#).

The major kernel SELinux source files (relative to ./kernel-2.6.27/security/selinux) that form the SELinux security module are shown in [Table 2-7](#). The diagrams shown in [Figure 2-2](#) and [Figure 2-12](#) can be used to see how some of these kernel source modules fit together.

Name	Function
avc.c	Access Vector Cache functions and structures. The function calls are for the kernel services, however they have been ported to form the libselinux userspace library detailed in Appendix C – API Summary for libselinux .
exports.c	Exported SELinux services for SECMARK (as there is SELinux specific code in the netfilter source tree).
hooks.c	Contains all the SELinux functions that are called by the kernel resources via the security_ops function table (they form the kernel resource object managers). There are also support functions for managing process exec's, managing SID allocation and removal, interfacing into the AVC and Security Server.
netif.c	These manage the mapping between labels and SIDs for the net* language statements when they are declared in the active policy.
netnode.c	
netport.c	
netlabel.c	The interface between NetLabel services and SELinux.
netlink.c	Manages the notification of policy updates to resources including userspace applications via libselinux.
nlsmsgtab.c	
selinuxfs.c	The selinuxfs pseudo filesystem (/selinux) that exports the security policy to userspace services via libselinux. The services exported are shown in the SELinux Filesystem section.
xfrm.c	Contains the IPsec XFRM hooks for SELinux.

Name	Function
include/flask.h	This contains all the kernel security classes and initial SIDs. Note that the Reference Policy source (<code>policy/flask</code> directory) contains a list of all the kernel and userspace security classes and permissions.
ss/avtab.c	AVC table functions for inserting / deleting entries.
ss/conditional.c	Support boolean statement functions and implements a conditional AV table to hold entries.
ss/avtab.c	AVC table functions for inserting / deleting entries.
ss/ebitmap.c	Bitmaps to represent sets of values, such as types, roles, categories, and classes. (not sure how it works)
ss/hashtab.c	Hash table (not sure how it works).
ss/mls.c	Functions to support MLS.
ss/policydb.c	Defines the structure of the policy database. See the “ SELinux Policy Module Primer ” [Ref. 4] article for details on the structure.
ss/services.c	This contains the supporting services for kernel hooks defined in <code>hooks.c</code> , the AVC and the Security Server. For example the <code>security_transition_sid</code> that computes the SID for a new subject / object shown in Figure 2-12 .
ss/sidtab.c	The SID table contains the security context indexed by its SID value.
ss/symtab.c	Maintains associations between symbol strings and their values. (not sure how it works).

Table 2-7: The core SELinux source modules - The .h files and those in the include directory have a number of useful comments.

2.16.2.1 Fork System Call Walk-through

This section walks through the the `fork` system call shown in [Figure 2-7](#) starting at the kernel hooks that link to the SELinux services. The way the SELinux hooks are initialised into the LSM `security_ops` and `secondary_ops` function tables are also described.

Using [Figure 2-10](#), the major steps to check whether the `unconfined_t` process has permission to use the fork permission are:

1. The `kernel/fork.c` has a hook that links it to the LSM function `security_task_create()` that is called to check access permissions.
2. Because the SELinux module has been initialised as the security module, the `security_ops` table has been set to point to the SELinux `selinux_task_create()` function in `hooks.c`.
3. The `selinux_task_create()` function will first call the capabilities code in `capability.c` via the `secondary_ops` function table to check the DAC permission.
4. This is simply a `return 0;`, therefore no error would be generated.

5. The `selinux_task_create()` function will then check whether the task has permission via the `task_has_perm(current_process, current_process, PROCESS__FORK)` function.
6. This will result in a call to the AVC via the `avc_has_perm()` function in `avc.c` that checks whether the permission has been granted or not. First (via `avc_has_perm_noaudit()`) the cache is checked to for an entry. Assuming that there is no entry in the AVC, then the `security_compute_av()` function in `services.c` is called.
7. The `security_compute_av()` function will search the SID table for source and target entries, and if found will then call the `context_struct_compute_av()` function.

The `context_struct_compute_av()` function carries out many check to validate whether access is allowed. The steps are (assuming the access is valid):

- a) Initialise the AV structure so that it is clear.
 - b) Check the object class and permissions are correct. It also checks the status of the `allow_unknown` flag (see the [SELinux Filesystem, /etc/selinux/semanage.conf file](#) and [Reference Policy Build Options - build.conf - UNK_PERMS](#) sections).
 - c) Checks if there are any type enforcement rules (`ALLOW`, `AUDIT_ALLOW`, `AUDIT_DENY`).
 - d) Check whether any conditional statements are involved via the `cond_compute_av()` function in `conditional.c`.
 - e) Remove permissions that are defined in any constraint via the `constraint_expr_eval()` function call (in `services.c`). This function will also check any MLS constraints.
 - f) Finally `context_struct_compute_av()` checks if a process transition is being requested (it is not). If it were, then the `TRANSITION` and `DYNTRANSITION` permissions are checked and whether the role is changing.
8. Once the result has been computed it is returned to the `kernel/fork.c` system call via the initial `selinux_task_create()` function. In this case the fork call is allowed.
 9. The End.

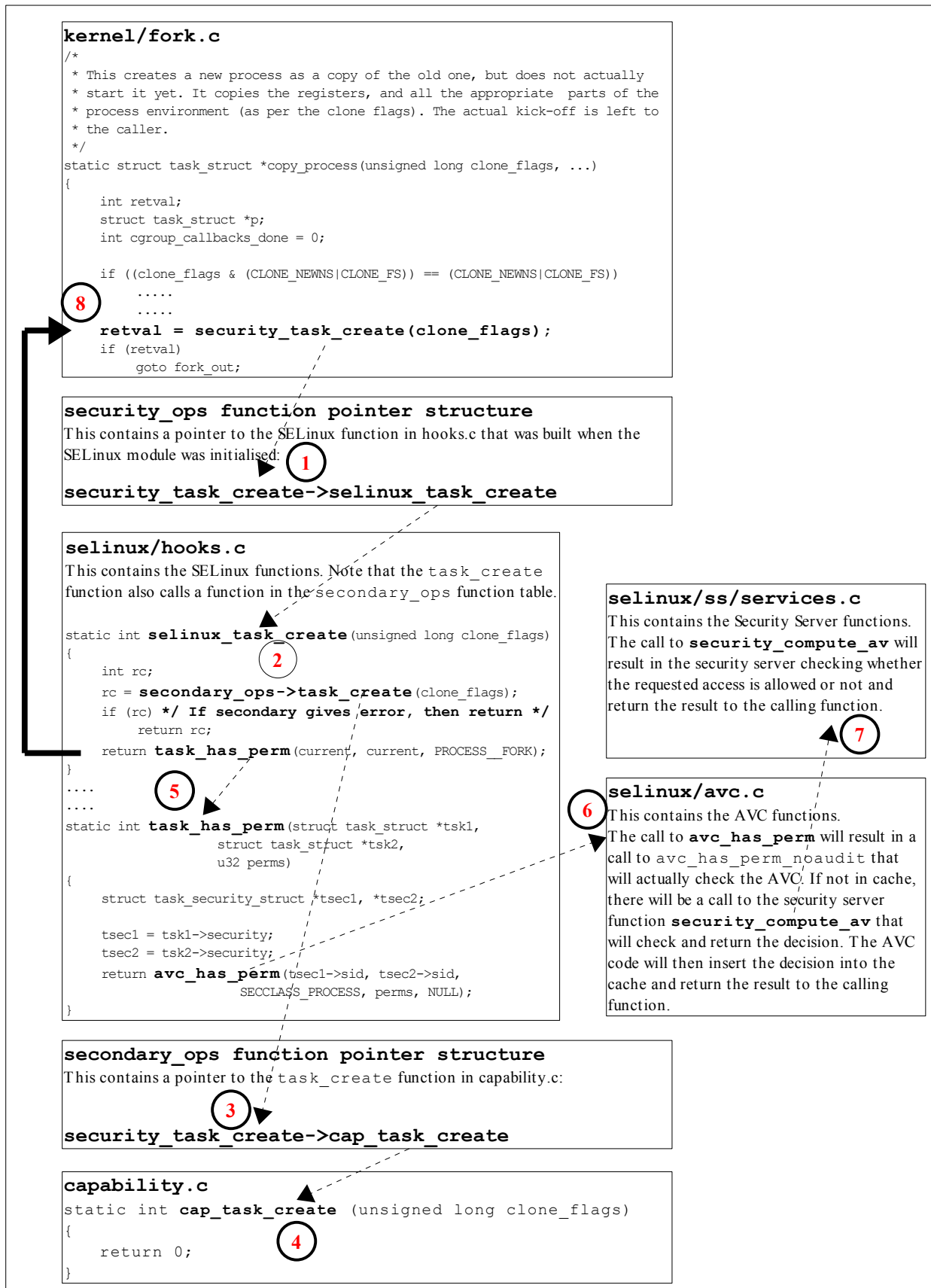


Figure 2-10: Hooks for the fork system call - This describes the steps required to check access permissions for Object Class 'process' and permission 'fork'.

2.16.2.2 Process Transition Walk-through

This section walks through the `execve()` and checking whether a process transition to the `ext_gateway_t` domain is allowed, and if so obtain a new SID for the context (`user_u:message_filter_r:ext_gateway_t`) as shown in [Figure 2-7](#).

The process starts with the Linux operating system issuing a `do_execve()`¹⁶ call from the CPU specific architecture code to execute a new program (for example, from `arch/ia64/kernel/process.c`). The `do_execve()` function is located in the `fs/exec.c` source code module and does the loading and final `exec` as described below.

`do_execve()` has a number of calls to `security_bprm_*` functions that are a part of the LSM (see `security.h`), and are hooked by SELinux during the initialisation process (in `hooks.c`). [Table 2-8](#) briefly describes these `security_bprm` functions that are hooks for validating program loading and execution (although see `security.h` or [Ref. 6] for greater detail).

LSM / SELinux Function Name	Description
<code>security_bprm_alloc-></code> <code>selinux_bprm_alloc_security</code>	Allocates memory for the <code>bprm</code> structure.
<code>security_bprm_free-></code> <code>selinux_bprm_free_security</code>	Frees memory from the <code>bprm</code> structure.
<code>security_bprm_apply_creds-></code> <code>selinux_bprm_apply_creds</code>	Sets task lock and new security attributes for a transformed process on <code>execve</code> . Seems to be used for libraries, scripts etc. Called from various Linux OS areas via <code>compute_creds()</code> located in <code>fs/exec.c</code> .
<code>security_bprm_post_apply_creds-></code> <code>selinux_bprm_post_apply_creds</code>	Supports the <code>security_bprm_apply_creds</code> function for areas that must not be locked.
<code>security_bprm_secureexec-></code> <code>selinux_bprm_secureexec</code>	Called after the <code>selinux_bprm_post_apply_creds</code> function to check <code>AT_SECURE</code> flag for glibc secure mode support.
<code>security_bprm_set-></code> <code>selinux_bprm_set_security</code>	Carries out the major checks to validate whether the process can transition to the target context, and obtain a new SID if required.
<code>security_bprm_check-></code> <code>selinux_bprm_check_security</code>	This hook is not used by SELinux.

Table 2-8: The LSM / SELinux Program Loading Hooks

Therefore starting at the `do_execve()` function and using [Figure 2-11](#), the following major steps will be carried out to check whether the `unconfined_t` process has permission to transition the `secure_server` executable to the `ext_gateway_t` domain:

1. The executable file is opened, a call issued to the `sched_exec()` function and the `bprm` structure is initialised with the file parameters (name, environment and arguments).

¹⁶ This function call will pass over the file name to be run and its environment + arguments. Note that for loading shared libraries the `exec_mmap` function is used.

The `security_bprm_alloc()->selinux_bprm_alloc_security()` function is then called (in `hooks.c`) where SELinux will allocate memory for the `bprm` security structure and set the `bsec->set` flag to 0 indicating this is the first time through this process for this `exec` request.

2. Via the `prepare_binprm()` function call the UID and GIDs are checked and a call issued to `security_bprm_set()` that will carry out the following:
 - a) The `selinux_bprm_set_security()` function will call the `secondary_ops->bprm_set_security` function in `capability.c`, that is effectively a no-op.
 - b) The `bsec->set` flag will be checked and if 1 will return as this function can be called multiple times during the `exec` process.
 - c) The target SID is checked to see whether a transition is required (in this case it is), therefore a call will be made to the `security_transition_sid()` function in `services.c`. This function will compute the SID for a new subject or object (subject in this case) via the `security_compute_sid()` function that will (assuming there are no errors):
 - i. Search the SID table for the source and target SIDs.
 - ii. Sets the SELinux user identity.
 - iii. Set the source role and type.
 - iv. Checks that a `type_transition` rule exists in the AV table and / or the conditional AV table (see [Figure 2-12](#)).
 - v. If a `type_transition`, then also check for a `role_transition` (there is a role change in the `ext_gateway.conf` policy module), set the role.
 - vi. Check if any MLS attributes by calling `mls_compute_sid()` in `mls.c`. It also checks whether MLS is enabled or not, if so sets up MLS contexts.
 - vii. Check whether the contexts are valid by calling `compute_sid_handle_invalid_context()` that will also log an audit message if the context is invalid.
 - viii. Finally obtains a SID for the new context by calling `sidtab_context_to_sid()` in `sidtab.c` that will search the SID table (see [Figure 2-12](#)) and insert a new entry if okay or log a kernel event if invalid.
 - d) The `selinux_bprm_set_security()` function will then continue by checking via the `avc_has_perm()` function (in `avc.c`) whether the `file_execute_no_trans` is set (in this case it is not), therefore the `process_transition` and `file_entrypoint` permissions are checked (in this case they are), therefore the new SID is set, the `bsec->set` flag is set to 1 so that this part of the function is not executed again for this `exec`, finally control is passed back to the `do_execve` function:

3. Various strings are copied (args etc.) and a check is made to see if the exec succeeded or not (in this case it did), therefore the `security_bprm_free()` function is called to free the `bprm` security structure.
4. The End.

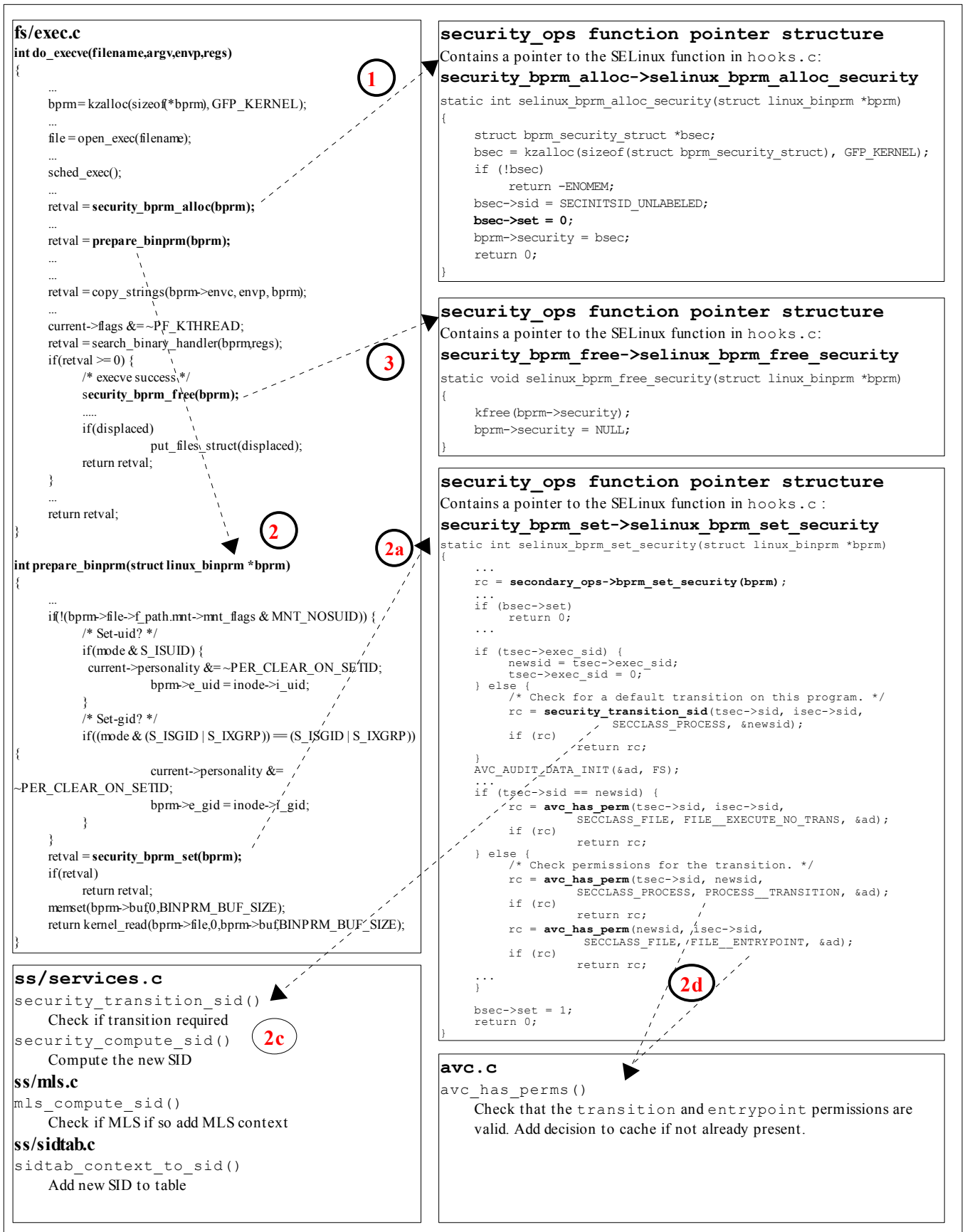


Figure 2-11: Process Transition - This shows the major steps required to check if transition is allowed from the unconfined_t domain to the ext_gateway_t domain.

The SELinux Notebook - The Foundations

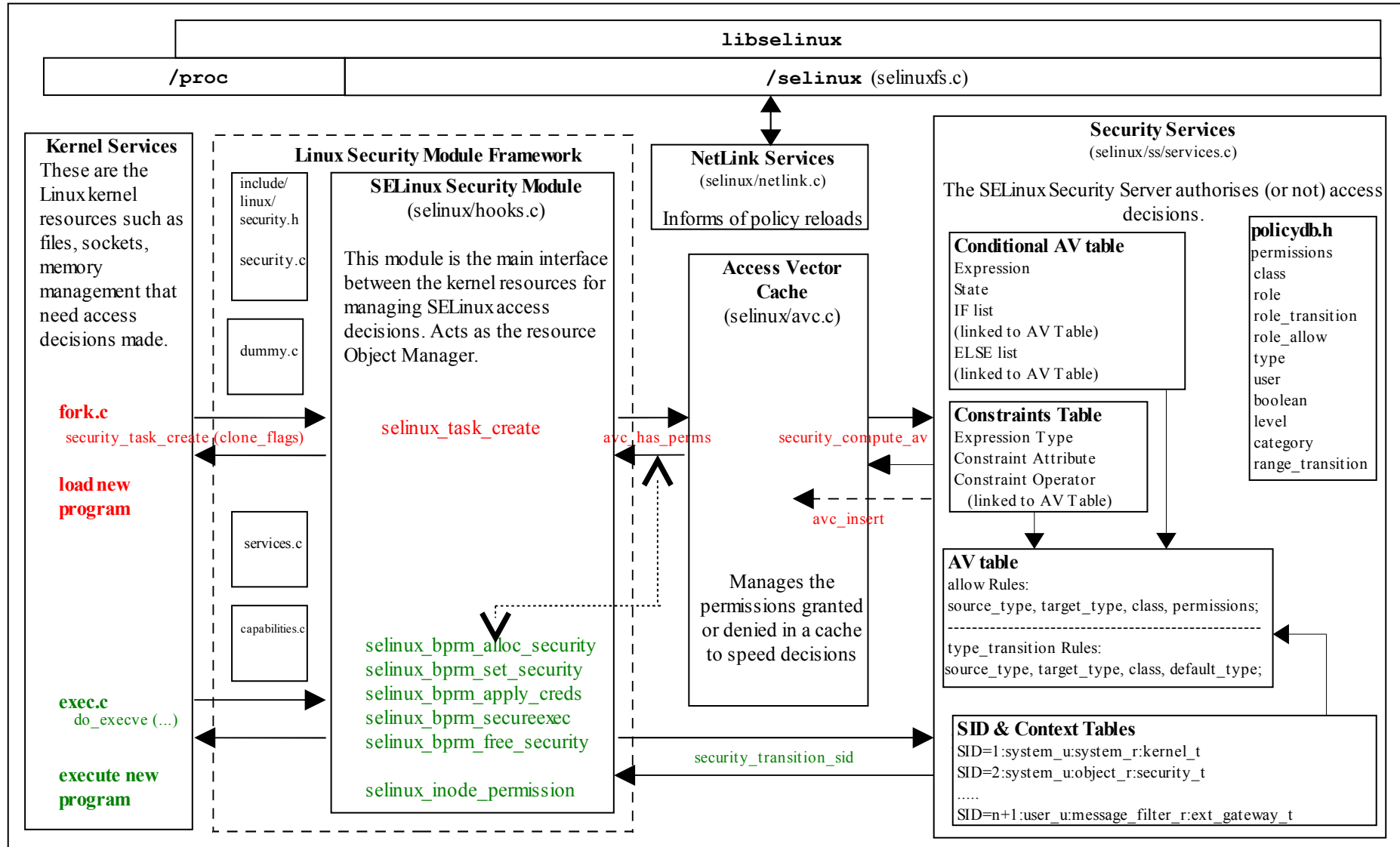


Figure 2-12: The Main LSM / SELinux Modules – The fork and exec functions link to [Figure 2-7](#) where the transition process is described.

2.16.2.3 SELinux Filesystem

[Table 2-9](#) shows the information contained in the pseudo file system `/selinux` where the SELinux kernel exports relevant information regarding its configuration and active policy for use by the `libselinux` library (that is used by user space object managers and SELinux-aware applications).

<i>Directory / File Name</i>	<i>Permissions</i>	<i>Comments</i>
/selinux	Directory	This is the root directory where the SELinux kernel exports relevant information regarding its configuration and active policy for use by the <code>libselinux</code> library (that is used by user space object managers and SELinux-aware applications).
access	-rw-rw-rw-	Compute access decision interface.
checkreqprot	-rw-r--r--	Check requested protection, not kernel-applied one.
commit_pending_bools	--w-----	Commit new boolean values to the kernel policy.
compat_net	-rw-r--r--	Whether SECMARK is enabled or not: 0 = SECMARK enabled (F-10 default). 1 = SECMARK disabled At some stage (kernel 2.6.30 and greater), the older <code>compat_net</code> network controls listed in the Network Labeling Statements section will be removed and replaced by SECMARK.
context	-rw-rw-rw-	Validate context interface.
create	-rw-rw-rw-	Compute create labeling decision interface.
deny_unknown	-r--r--r--	These two files export unknown deny and reject handling status to user space. This is taken from the <code>handle-unknown</code> parameter set ¹⁷ in the /etc/selinux/semanage.conf file and are set as follows: 0:0 = allow, 1:0 = deny and 1:1 = reject.
reject_unknown	-r--r--r--	
disable	--w-----	Disable SELinux until next reboot.
enforce	-rw-r--r--	Get or set enforcing status. Used by the <code>setenforce(8)</code> command.
load	-rw-----	Load policy interface.
member	-rw-rw-rw-	Compute polyinstantiation membership decision interface.
mls	-r--r--r--	Returns 1 if MLS policy is enabled or 0 if not.
null	crw-rw-rw-	
policyvers	-r--r--r--	Returns policy version for this kernel (F-10 = 23).
relabel	-rw-rw-rw-	Compute relabeling decision interface.
user	-rw-rw-rw-	Compute reachable user contexts interface.
/selinux/avc	Directory	This directory contains information regarding the kernel AVC. These can be read by the <code>avcstat(8)</code> command.
cache_stats	-r--r--r--	
cache_threshold	-rw-r--r--	

¹⁷ That is taken from the `UNK_PERMS` entry in the Reference Policy [build.conf](#) file.

The SELinux Notebook - The Foundations

<i>Directory / File Name</i>	<i>Permissions</i>	<i>Comments</i>
hash_stats	-r--r--r--	
/selinux/booleans	Directory	This directory contains one file for each boolean defined in the active policy.
secmark_audit	-rw-r--r--	Each file contains the current status of the boolean (0 = false or 1 = true). The <code>getsebool(8)</code> , <code>setsebool(8)</code> and <code>sestatus -b</code> commands use this information.
/selinux/initial_contexts	Directory	This directory contains one file for each initial SID defined in the active policy.
any_socket devnull	-r--r--r--	Each file contains the initial context of the initial SID as defined in the active policy (e.g. <code>any_socket</code> was assigned <code>system_u:object_r:unconfined_t</code>).
/selinux/policy_capabilities	Directory	This directory contains the policy capabilities that have been configured by default in the kernel. They are generally new features that can be enabled for testing by using the policycap Statement in a monolithic or base policy.
network_peer_controls	-r--r--r--	For the F-10 kernel, this file contains '0' (false) which means that the following <code>network_peer_controls</code> are not enabled by default: node: sendto recvfrom netif: ingress egress peer: recv See the NetLabel Loadable Module section for an example with this enabled.
open_perms	-r--r--r--	For the F-10 kernel, this file contains '0' (false) which means that open permissions are not enabled by default on the following objects: <code>dir</code> , <code>file</code> , <code>fifo_file</code> , <code>chr_file</code> , <code>blk_file</code> .
/selinux/class	Directory	This directory contains a list of classes and their permissions as defined within the policy.
/selinux/class/appletalk_socket	Directory	Each class has its own directory that contains the following:
index	-r--r--r--	This file contains the allocated class number (e.g. <code>appletalk_socket</code> is '56' in <code>flask.h</code> (<code>linux-2.6.27/security/selinux/include/flask.h</code>)).
/selinux/class/appletalk_socket/permissions	Directory	This directory contains one file for each permission defined in the policy.
accept append bind	-r--r--r--	Each file contains a number but not sure what it represents ???

Table 2-9: /selinux File and Directory Information

Notes:

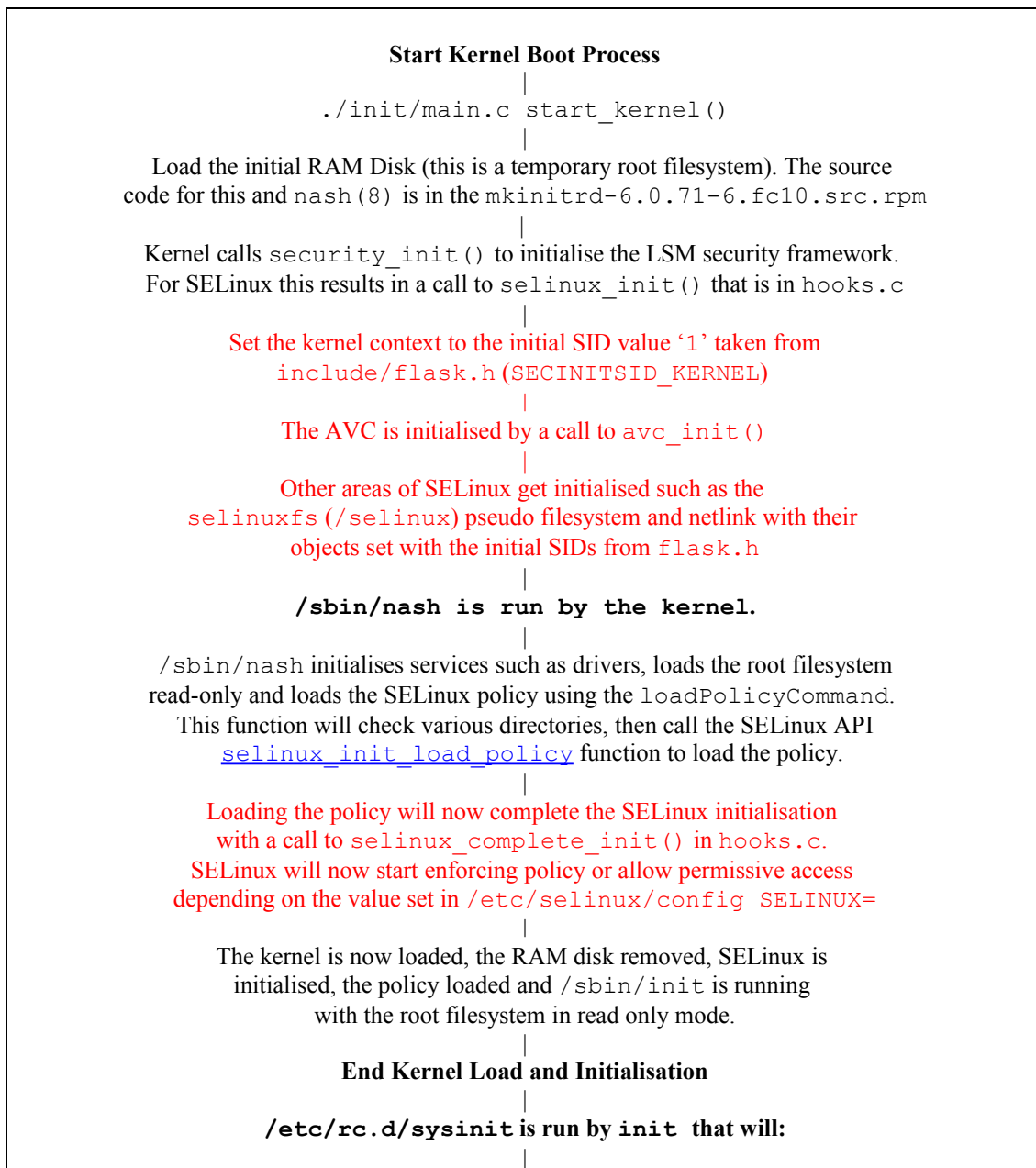
1. SIDs are not passed to userspace, only the security context string. The context can be read via the [libselinux API](#) where a userspace object manager

normally manages the relationship (see “SELinux Support for Userspace Object Managers” [Ref. 17]).

2. The `/proc` filesystem exports the process security context string to userspace via `/proc/<pid>/attr` interface (where `<pid>` is the process ID). These can also be managed via the [libselinux API](#).

2.16.2.4 SELinux Boot Process

[Figure 2-13](#) shows the boot process that has been limited to what is considered relevant for initialising SELinux¹⁸, loading the policy and checking whether re-labeling is required. The SELinux kernel initialisation areas are in red. The `kernel`, `mkinitrd` and `upstart` source code rpms were used to find the sequence of events (all corrections welcome - as going through the source was painful).



¹⁸ There is a Linux overview at: http://en.wikipedia.org/wiki/Linux_startup_process.

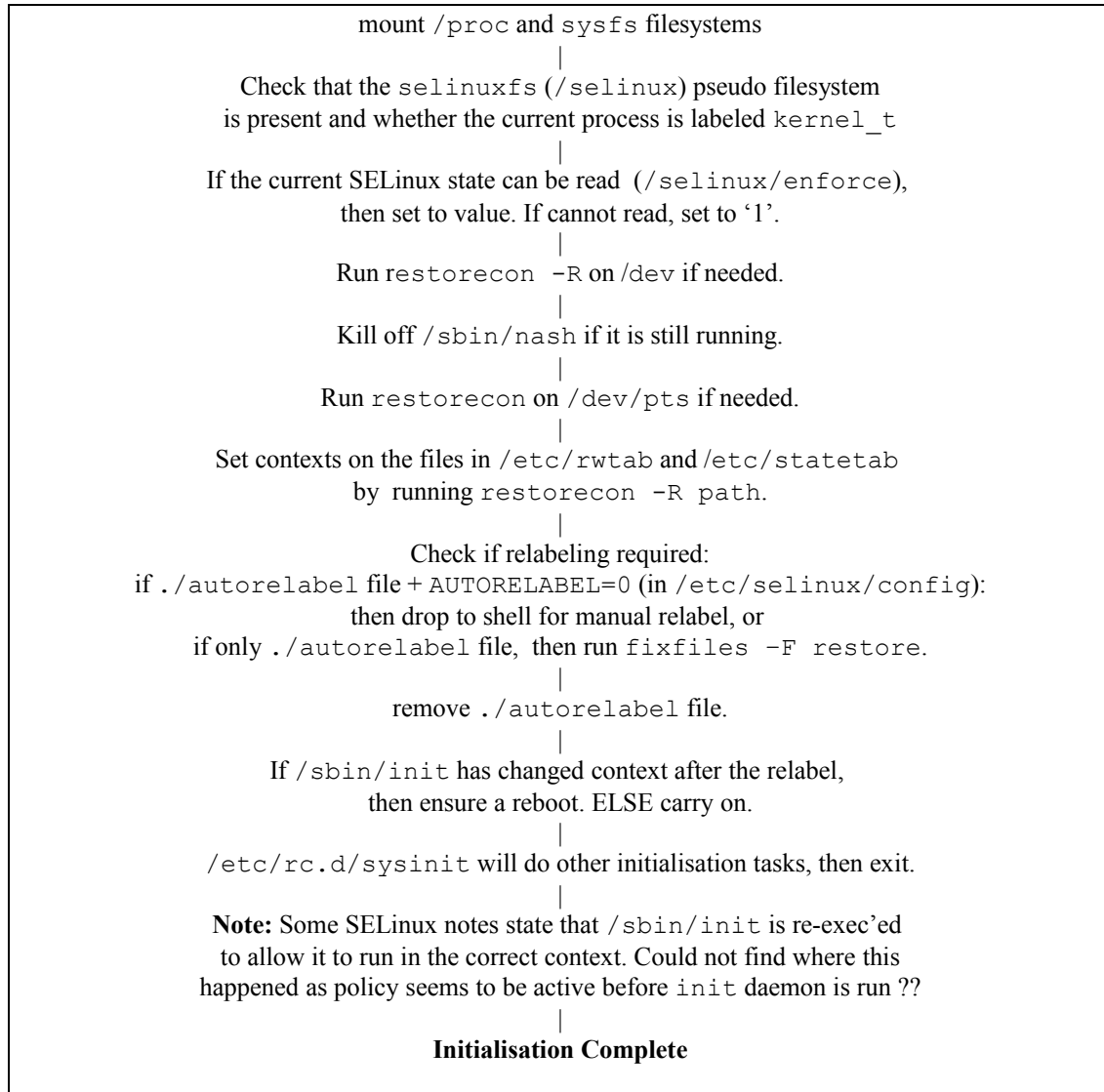


Figure 2-13: The Boot Sequence – This shows how SELinux is initialised and the policy loaded during the boot process.

2.17 SELinux Networking Support

SELinux supports the following types of network labeling:

Internal labeling – This is where network objects are labeled and managed internally within a single machine (i.e. their labels are not transmitted as part of the session with remote systems). There are three types supported: those known as 'compat_net' controls that label nodes, interfaces and ports; SECMARK that labels packets; and fallback peer labeling.

Labeled Networking – This is where labels are passed to/from remote systems where they can be interpreted and a MAC policy enforced on each system. This is also known as 'peer labeling'. There are two types supported: Labeled IPsec and CIPSO (commercial IP security option).

Note that F-10 does not have NetLabel or IPsec tools installed as standard, therefore yum can be used to install them as shown below:

```
yum install netlabel_tools

# yum will then install netlabel_tools-0.18-1.fc10.i386.rpm
# or a later version.
```

```
yum install ipsec_tools

# yum will then install ipsec_tools-0.7.2-1.fc10.i386.rpm
# or a later version.
```

2.17.1 compat_net Controls

These labeling services make use of the [Network Labeling Statements](#) to label network object nodes, interfaces and ports with a security context that are then used to enforce controls. The [Network Labeling Statements](#) section defines each of the statements with examples of their usage.

[Figure 2-14](#) shows how these network statements are used and the type of allow rules that would be required.

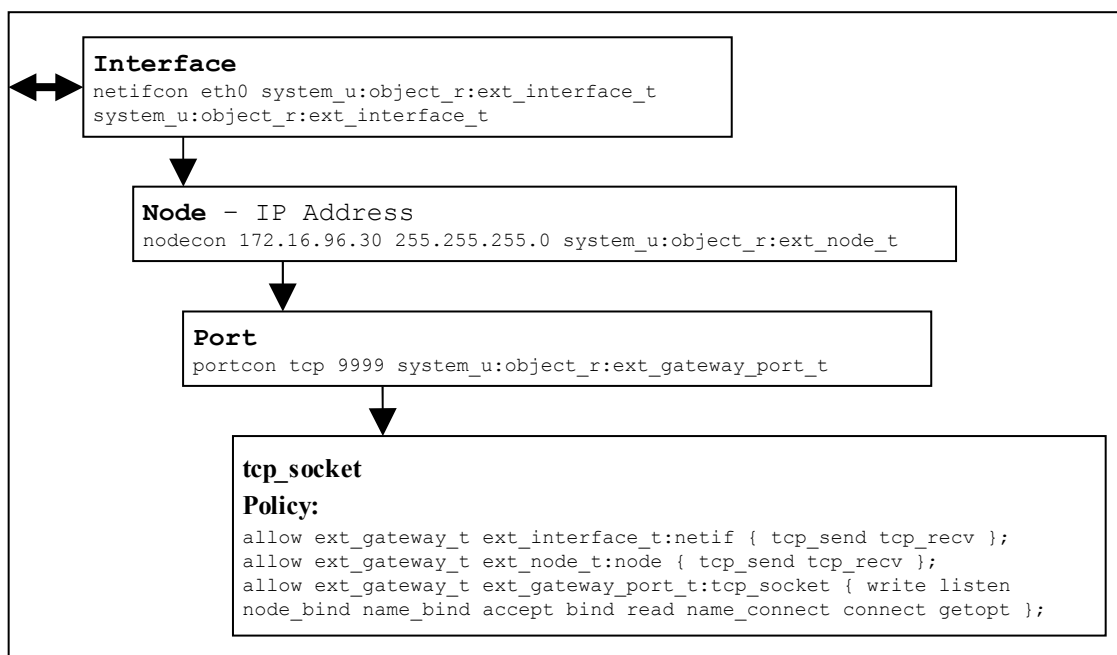


Figure 2-14: compat_net Controls – Showing the policy statements and rules required to allow communications.

In a future release of the Linux kernel these controls will be removed and replaced by the SECMARK services with the Reference Policy also being updated.

2.17.2 SECMARK

SECMARK makes use of standard the kernel NetFilter framework that underpins the GNU / Linux IP networking sub-system. NetFilter automatically inspects all incoming and outgoing packets and can place controls on interfaces, IP addresses (nodes) and ports with the added advantage of connection tracking. The SECMARK and CONNSECMARK are security extensions to the Netfilter iptables that allow security contexts to be added to packets (SECMARK) or sessions (CONNSECMARK)

such as those used by ftp (as some applications within a single session can use a number of different ports, some fixed and others dynamically allocated).

The NetFilter framework is used to inspect and tag packets with labels as defined within the `iptables` and then use the security framework (e.g. SELinux) to enforce the policy rules. Therefore SECMARK services are not SELinux specific as other security modules that use the LSM infrastructure could also implement the same services (e.g. SMACK).

While the implementation of `iptables` / NetFilter is beyond the scope of this Notebook, there are tutorials available¹⁹. [Figure 2-15](#) shows the basic structure and the process works as follows:

- A table called the ‘mangle table’ is used to define the parameters that identify and ‘mark’ packets that can then tracked as the packet travels through the networking sub-system. These ‘marks’ are called SECMARK and CONNSECMARK.
- A SECMARK is placed against a packet if it matches an entry in the mangle table. This marker is used to apply a security context (a label) that can then enforce policy on the packet.
- The CONNSECMARK ‘marks’ all packets within a session²⁰ with the appropriate label that can then be used to enforce policy.

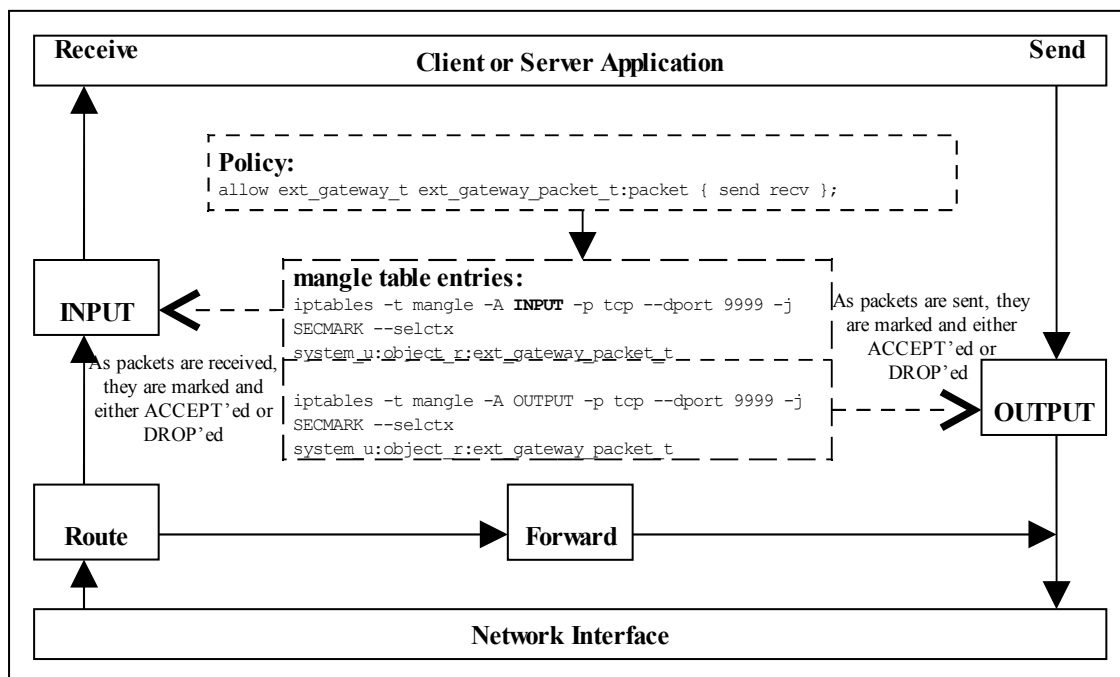


Figure 2-15: SECMARK Processing – Received packets are processed by the **INPUT** chain where labels are added to the appropriate packets that will either be accepted or dropped by the SECMARK process. Packets being sent are treated the same way.

¹⁹ There is a very good tutorial at <http://iptables-tutorial.frozentux.net/iptables-tutorial.html> [Ref.7].

²⁰ For example, an ftp session where the server is listening on a specific port (the destination port) but the client will be assigned a random source port. The CONNSECMARK will ensure that all packets for the ftp session are marked with the same label.

An example iptables²¹ entry is as follows:

```
# Flush the mangle table first:
iptables -t mangle -F

#----- INPUT IP Stream -----#
# This INPUT rule sets all packets to default_secmark_packet_t
iptables -t mangle -A INPUT -i lo -p tcp -d 127.0.0.0/8 -j SECMARK --selctx
system_u:object_r:default_secmark_packet_t

#----- OUTPUT IP Stream -----#
# This OUTPUT rule sets all packets to default_secmark_packet_t
iptables -t mangle -A OUTPUT -o lo -p tcp -d 127.0.0.0/8 -j SECMARK --selctx
system_u:object_r:default_secmark_packet_t
```

An example loadable module that makes use of SECMARK services is described in the [Building the SECMARK Test Loadable Module](#) section and there is an article “[New secmark-based network controls for SELinux](#)” [Ref. 8] that explains the services.

As stated in the [compat_net Controls](#) section, SECMARK will be replacing these and there is an article “[Transitioning to Secmark](#)” [Ref. 9] that explains the transition.

2.17.3 NetLabel - Fallback Peer Labeling

Fallback labeling can optionally be implemented on a system if the Labeled IPsec or CIPSO is not being used (hence ‘fallback labeling’). If either Labeled IPsec or CIPSO are being used, then these take priority. There is an article “[Fallback Label Configuration Example](#)” [Ref. 10] that explains the usage.

The example message filter has an optional module that makes use of fallback labels as explained in the [Overview of modules](#) section.

The network peer controls has been extended to support an additional object class of ‘peer’, although by default this is not enabled in F-10. To enable this functionality the policy capability needs to be set as explained in [Appendix E – NetLabel Module Support for network_peer_controls](#) where an example loadable module is given.

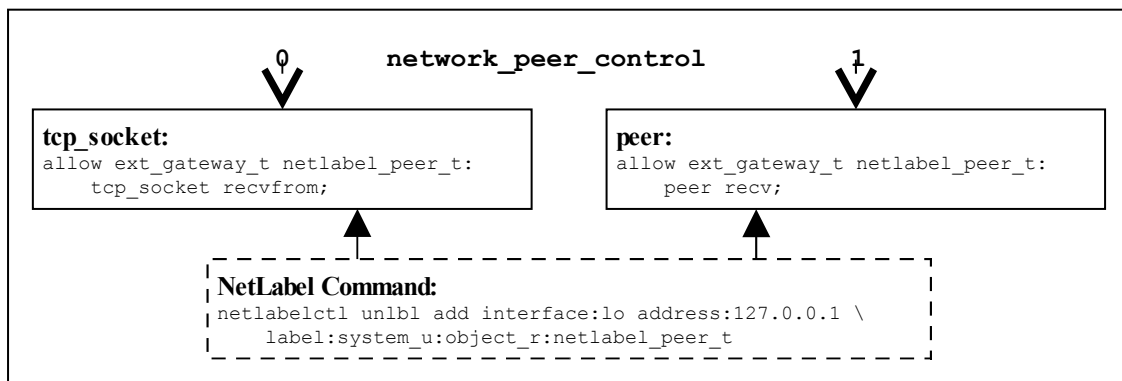


Figure 2-16: Fallback Labeling – Showing the differences between the policy capability network_peer_controls set to 0 and 1.

²¹ Note that the iptables will not load correctly if the policy does not allow the iptables domain to relabel the SECMARK labels (unless permissive mode is enabled).

2.17.4 Labeled IPsec

Labeled IPsec has been built into the standard GNU / Linux IPsec services as described in the “[Leveraging IPsec for Distributed Authorization](#)” [Ref. 11] document. [Figure 2-17](#) shows the basic components that form the IPsec service where it is generally used to set up either an encrypted tunnel between two machines²² or an encrypted transport session. The extensions defined in [Ref. 11] describe how the security context is used and negotiated between the two systems (called security associations (SAs) in IPsec terminology).

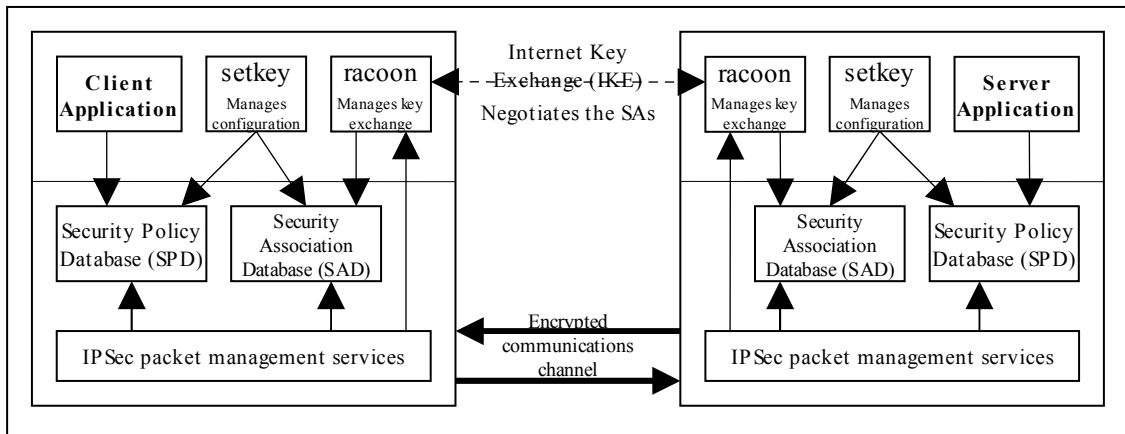


Figure 2-17: IPsec communications – The SPD contains information regarding the security contexts to be used. These are exchanged between the two systems as part of the channel set-up.

Basically what happens is as follows²³:

1. The security policy database (SPD) defines the security communications characteristics to be used between the two systems. This is populated using the `setkey(8)` utility and an example is shown below.
2. The SAs have their configuration parameters such as protocols used for securing packets, encryption algorithms and how long the keys are valid held in the Security Association database (SAD). For Labeled IPsec the security context (or labels) is also defined within the SAD. SAs can be negotiated between the two systems using either `racoon(8)`²⁴ that will automatically populate the SAD or manually by the `setkey` utility (see the example below).
3. Once the SAs have been negotiated and agreed, the link should be active.

A point to note is that SAs are one way only, therefore if two systems are communicating then (using the above example), one system will have an SA, `SAout` for processing outbound packets and another SA, `SAin`, for processing the inbound packets. The other system will also create two SAs for processing its packets.

²² Also known as a virtual private network (VPN).

²³ There is an “IPsec HOWTO” [Ref. 12] at <http://www.ipsec-howto.org> that gives the gory details, however it does not cover Labeled IPsec.

²⁴ This is the Internet Key Exchange (IKE) daemon that exchanges encryption keys securely and also supports Labeled IPsec parameter exchanges.

Each SA will share the same cryptographic parameters such as keys and protocol²⁵ such as AH (authentication header) and ESP (encapsulated security payload).

The object class used for the association of an SA is `association` and the permissions available are as follows:

<code>polmatch</code>	Match the SPD context (<code>-ctx</code>) entry to an SELinux domain (that is contained in the SAD <code>-ctx</code> entry)
<code>recvfrom</code>	Receive from an IPsec association.
<code>sendto</code>	Send to an IPsec association.
<code>setcontext</code>	Set the context of an IPsec association on creation (e.g. when running <code>setkey</code> the process will require this permission to set the context in the SAD and SPD, also <code>racoon</code> will need this permission to build the SAD).

A worked example of a Labeled IPsec session showing manual and `racoon`²⁶ to configure the SAD is described in [Appendix E – Labeled IPsec Module Example](#).

There is a further example is shown in the “[Secure Networking with SELinux](#)” [Ref. 13] article.

```
# setkey -f configuration file entries
#
# Flush the SAD and SPD
flush;
spdflush;

# Security Association Database entries.
# 1) There would be another SAD entry on the other system (the
#    client), where the IP addresses would be reversed.
# 2) The security context must be that of the running application.

add 172.16.96.30 172.16.96.31 esp 0x201
-ctx 1 1 "user_u:message_filter_r:ext_gateway_t"
-E 3des-cbc 0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831;

# Security Policy Database entries.
# 1) there would be another SPD entry on the other system (the
#    client), where the IP addresses would be reversed.
# 2) The security context must be valid (i.e. defined in the active policy as
#    it will be used by the polmatch permission process to find a matching
#    domain. (note only the 'type' field is used unlike the SAD, where
#    the context is the active process).

# SAin
spdadd 172.16.96.30 172.16.96.31 any
-ctx 1 1 "system_u:object_r:ext_gateway_t"
-P in ipsec esp/transport//require;
# SAout
spdadd 172.16.96.31 172.16.96.30 any
-ctx 1 1 "system_u:object_r:ext_gateway_t"
-P out ipsec esp/transport//require;
```

To manually load the above configuration file to populate the SPD and SAD²⁷ the following command would be used:

²⁵ The GNU / Linux version supports a number of secure protocols, see the `setkey` man page for details.

²⁶ Unfortunately `racoon` core dumps using the [example base module](#) but does work using the standard Red Hat targeted policy.

²⁷ If using `racoon`, the SAs would be negotiated using information from the SPD on each machine, with the SAD then being populated by `racoon` calling the `setkey` services.

```
setkey -f <SPD_configuration_file>
```

2.17.5 NetLabel - CIPSO

To allow [security levels](#) to be passed over a network between MLS systems²⁸, the CIPSO protocol is used that is defined in the [CIPSO Internet Draft](#) document (this is an obsolete document, however the protocol is still in use). The protocol defines how security levels are encoded in the IP packet header.

The protocol is implemented by the NetLabel service and can be used by other security modules that use the LSM infrastructure. The NetLabel implementation supports:

1. Tag Type 1 bit mapped format that allows a maximum of 256 sensitivity levels and 240 categories to be mapped.
2. A non-translation option where labels are passed to / from systems unchanged (for host to host communications as show in [Figure 2-18](#)).

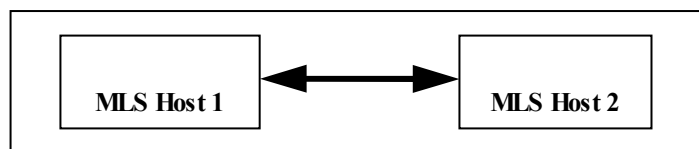


Figure 2-18: MLS Systems on the same network

3. A translation option where both the sensitivity and category components can be mapped for systems that have either different definitions for labels or information can be exchanged over different networks (for example using an SELinux enabled gateway as a guard as shown in [Figure 2-19](#)).

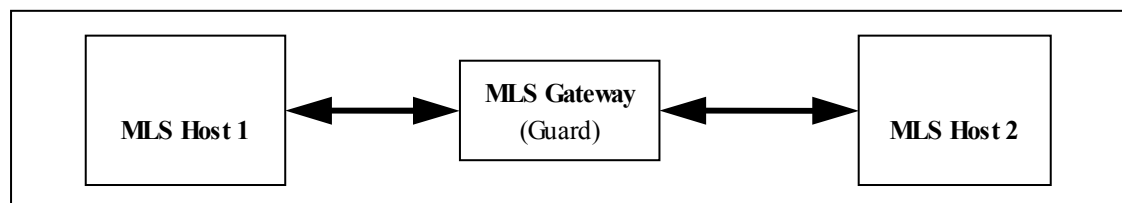


Figure 2-19: MLS Systems on different networks communicating via a gateway

²⁸ Note only the security levels are passed over as the SELinux security context is not part of a standard MLS system (as SELinux supports two MAC services (Type Enforcement and MLS)).

3. SELinux Configuration Files

3.1 Introduction

This section explains each SELinux configuration file with its format, example content and where applicable, any supporting SELinux command or library API function names (where [Appendix C – API Summary for libselinux](#) or the appropriate man(3) pages should be consulted regarding their use).

Note: Configuration file names and content have changed over the various releases of SELinux, this section defines those seen in the F-10 release when building custom and Reference Policy policies.

This Notebook classifies the types of configuration file used in SELinux as follows:

1. [Global Configuration files](#) that affect the active policy and their supporting SELinux-aware applications, utilities or commands. These can be located in `/etc/selinux` or other places depending on the application. This Notebook will only refer to the commonly used configuration files.
2. Files specific to a named policy configuration that are located at `/etc/selinux/<policy_name>`, where `<policy_name>` is the name given in the `SELINUXTYPE=` entry of the [/etc/selinux/config](#) file. The files in this area are split into two main sections:
 - a. The [Policy Store Configuration files](#) that are ‘private’²⁹ and managed by the `semanage(8)` and `semodule(8)` commands³⁰. These are located in the `/etc/selinux/<policy_name>/module` set of directories. These are used to build the majority of the [Policy Configuration files](#).
 - b. The [Policy Configuration files](#) that are used when the policy is activated³¹. The majority of these files are now managed via the Policy Store and should not be edited directly, however others are specific to SELinux-aware applications and have no configuration utilities (e.g. `debus` and X-Windows context files).
3. [SELinux Kernel Configuration files](#) that are located under the `/selinux` directory and reflect the current configuration of SELinux and the active policy. This area is used extensively by the `libselinux` library for user space object managers and other SELinux-aware applications. These files and directories should not be updated by users (the majority are read only anyway), however they can be read to check various configuration parameters.

Whenever possible the appropriate SELinux application should be used to manage all of these configuration files.

²⁹ They should NOT be edited as together they describe the ‘policy’.

³⁰ The `system-config-selinux` GUI (supplied in the `polycoreutils-gui` rpm) can also be used to manage users, booleans and the general configuration of SELinux as it calls `semanage`, however it does not manage all that the `semanage` command can (it also gets bitter & twisted if there are no MCS/MLS labels on some operations).

³¹ The ‘active policy’ is pointed to by an entry in the `/etc/selinux/config` file discussed in the [/etc/selinux/config](#) file section.

When these configuration files are used to configure a security context and the policy supports MCS / MLS, then the appropriate `level` or `range` should be added (generally an object like a file has a `level`, and a user or process (a subject) has a `level` and `range`, although directories can have a `range` if they support [polyinstantiation](#)).

3.2 Global Configuration Files

Listed in the sections that follow are the common configuration files used by SELinux and are therefore not policy specific.

3.2.1 `/etc/selinux/config` File

If this file is missing or corrupt no SELinux policy will be loaded (i.e. SELinux is disabled). The `config` file controls the state of SELinux using the following parameters:

```
SELINUX=enforcing|permissive|disabled
SELINUXTYPE=policy_name
SETLOCALDEFS=0|1
REQUIREUSERS=0|1
AUTORELABEL=0|1
```

Where:

SELINUX	This can contain one of three values: <code>enforcing</code> SELinux security policy is enforced. <code>permissive</code> SELinux logs warnings (see the Audit Logs section) instead of enforcing the policy (i.e. the action is allowed to proceed). <code>disabled</code> No SELinux policy is loaded.
SELINUXTYPE	Where <code>policy_name</code> is the policy type or name that will be loaded at system boot time. With F-10 there are three possible values (<code>targeted</code> , <code>mls</code> or <code>minimum</code>) as these are the policies available with the release. However it can be any custom policy with examples being given in the Building a Basic Policy section. The policy MUST be located at: <code>/etc/selinux/<policy_name>/</code>
SETLOCALDEFS	This optional field should be set to 0 (or the entry removed) as F-10 uses the policy store management infrastructure (<code>semanage</code> / <code>semodule</code>). If set to 1, then <code>init(8)</code> and <code>load_policy(8)</code> will read the local customisation for booleans and users.
REQUIRESEUSERS	This optional field can be used to fail the login when

there is no [seusers](#) file if it is set to 1.

The default action (if 0 or the entry is not present) the libselinux function `getseuserbyname` will use the GNU/Linux user name.

AUTORELABEL

This is an optional field. If set to '0' and there is a file called `.autorelabel` in the root directory, then on a reboot, the loader will drop to a shell where a root logon is required. An administrator can then manually relabel the file system.

If set to '1' or the parameter name is not used (the default) there is no login for manual relabeling, however should the `/.autorelabel` file exist, then the file system will be automatically relabeled using `fixfiles -F restore`.

In both cases the `/.autorelabel` file will be removed so the relabel is not done again. This option is also discussed in the [System Boot Process](#) section.

Example config file contents are:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#   enforcing - SELinux security policy is enforced.
#   permissive - SELinux prints warnings instead of enforcing.
#   disabled - No SELinux policy is loaded.
SELINUX=permissive
#
# SELINUXTYPE= can take one of these two values:
#   targeted - Targeted processes are protected,
#   mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

3.2.2 /etc/selinux/semanage.conf File

The `semanage.conf` file controls the configuration and actions of the `semanage` and `semodule` set of commands using the following parameters:

```
module-store = method
policy-version = policy_version
expand-check = 0|1
file-mode = mode
save-previous = true|false
save-linked = true|false
disable-genhomedircon = true|false
handle-unknown = allow|deny|reject
```

Where:

`module-store`

The method can be one of four options:

`direct`

`libsemanage` will write

	directly to a module store. This is the default value.
source	libsemanage manipulates a source SELinux policy.
/foo/bar	Write via a policy management server, whose named socket is at /foo/bar. The path must begin with a '/
foo.com:4242	Establish a TCP connection to a remote policy management server at foo.com. If there is a colon then the remainder is interpreted as a port number; otherwise default to port 4242.
policy-version	This optional entry can contain a policy version number , however it is normally commented out as it then defaults to that supported by the system (for F-10 this is policy version 23).
expand-check	This optional entry controls whether hierarchy checking on module expansion is enabled (1) or disabled (0). The default is 0.
file-mode	This optional entry allows the file permissions to be set on runtime policy files. The format is the same as the mode parameter of the chmod command and defaults to 0644 if not present.
save-previous	This optional entry controls whether the previous module directory is saved (TRUE) after a successful commit to the policy store. The default is to delete the previous version (FALSE).
save-linked	This optional entry controls whether the previously linked module is saved (TRUE) after a successful commit to the policy store. Note that this option will create a base.linked file in the module policy store. The default is to delete the previous module (FALSE).
disable-genhomedircon	This optional entry controls whether the embedded genhomedircon function is run when using the semanage command. The default is FALSE.
handle-unknown	This optional entry controls the kernel behaviour for handling permissions defined in the kernel but missing from the policy (that are declared at the start of the base.conf (loadable policy) or policy.conf (monolithic policy)). The options are: allow the permission, reject by

not loading the policy or deny the permission. The default is deny. See the [SELinux Filesystem](#) section for how these are reported in `/selinux`.

Note: to activate any change, the base policy needs to be reloaded with the `semodule -b` command (as `semodule -R` does not change them).

Example `semanage.conf` file contents are:

```
# /etc/selinux/semanage.conf

module-store = direct
expand-check = 0
```

3.2.3 `/etc/selinux/restorecond.conf` File

The `restorecond.conf` file contains a list of files that may be created by applications with an incorrect security context. The `restorecond` daemon will then watch for their creation or modification and automatically correct their security context to that specified by the active policy file context configuration files³² (located in the `/etc/selinux/<policy_name>/contexts/files` directory).

Each line of the file contains the full path of a file or directory. The only different entry is one that starts with a tilde (`~`) as that signifies that the entries will be expanded to logged in users home directories (e.g. `~/public_html` would cause the daemon to listen for changes to `public_html` in all logged on users home directories).

Example `restorecond.conf` file contents are:

```
# /etc/selinux/restorecond.conf

/etc/services
/etc/resolv.conf
/etc/samba/secrets.tdb
/etc/mtab
/var/run/utmp
/var/log/wtmp

# This entry expands to listen for all files created for all
# logged in users within their home directories:
~/*
```

3.2.4 `/etc/sestatus.conf` File

This file is used by the `sestatus(8)` command to list files and processes whose security context should be displayed when the `-v` flag is used (`sestatus -v`).

The `sestatus.conf` file has the following parameters:

³² The daemon uses functions in `libselinux` such as `matchpathcon(3)` to manage the context updates.

```
[files]
List of files to display context

[process]
List of processes to display context
```

Example `sestatus.conf` file contents are:

```
# /etc/sestatus.conf

[files]
/etc/passwd
/etc/shadow
/bin/bash
/bin/login
/bin/sh
/sbin/agetty
/sbin/init
/sbin/mingetty
/usr/sbin/sshd
/lib/libc.so.6
/lib/ld-linux.so.2
/lib/ld.so.1

[process]
/sbin/mingetty
/sbin/agetty
/usr/sbin/sshd
```

3.2.5 `/etc/security/sepermit.conf` File

This file is used by the `pam_sepermit.so` module to allow or deny a user login depending on whether SELinux is enforcing the policy or not. An example use of this facility is the Red Hat kiosk policy where a terminal can be set up with a guest user that does not require a password, but can only log in if SELinux is in enforcing mode.

The entry is added to the appropriate `/etc/pam.d` configuration file, with the example shown being the `/etc/pam.d/gdm` file (the [PAM Login Process](#) section describes PAM in more detail):

```
##PAM-1.0
auth [success=done ignore=ignore default=bad] pam_selinux_permit.so
# auth required pam_succeed_if.so user != root quiet
auth required pam_env.so
auth substack system-auth
auth optional pam_gnome_keyring.so
account required pam_nologin.so
account include system-auth
password include system-auth
session required pam_selinux.so close
session required pam_loginuid.so
session optional pam_console.so
session required pam_selinux.so open
session optional pam_keyinit.so force revoke
session required pam_namespace.so
session optional pam_gnome_keyring.so auto_start
session include system-auth
```

The usage is described in the `pam_sepermit` man page, but the example taken from F-10 describes the configuration:

```
# /etc/security/sepermit.conf
#
# Each line contains either:
#   - an user name
#   - a group name, with @group syntax
#   - a SELinux user name, with %seuser syntax
#
# Each line can contain optional arguments separated by :
# The possible arguments are:
#   - exclusive - only single login session will be allowed for
#     the user and the user's processes will be killed on logout
#
# An example entry for 'kiosk mode':
xguest:exclusive
```

3.3 Policy Store Configuration Files

Each file discussed in this section is relative to the policy name as follows:

```
/etc/selinux/<policy_name>
```

The Policy Store files in the `/etc/selinux/<policy_name>/modules` area are either installed, updated or built by the `semodule` and `semanage` commands, and as a part of their process, relevant files are then copied to the [Policy Configuration files](#) area.

The files present in each `<policy_name>` policy store will vary from policy to policy as different items could be configured for each one.

Generally if a file has the extension `.local`, then it has been generated by `semanage` and used to update the binary policy located at `/etc/selinux/<policy_name>/policy`.

All files can have comments inserted where each line must have the `#` symbol to indicate the start of a comment.

3.3.1 modules/ Files

The policy store has two lock files that are used by `libsemanage` for managing the store. Their format is not relevant to policy construction:

```
semanage.read.LOCK
semanage.trans.LOCK
```

3.3.2 modules/active/base.pp File

This is the packaged base policy that contains the mandatory modules and policy components such as object classes and permission declarations, initial SIDs as shown in the [Policy Source Files](#) section ([Table 5-1](#)).

The [Building a Loadable Module Policy](#) section shows how to build a simple base policy that would result in the `base.pp` module being placed here.

3.3.3 `modules/active/base.linked File`

This is only present if the `save-linked` is set to `TRUE` as described in the [/etc/selinux/semanage.conf](#) section. It contains the modules that have been linked using the `semodule_link(8)` command.

3.3.4 `modules/active/commit_num File`

This is a binary file used by `libsemanage` for managing updates to the store. The format is not relevant to policy construction.

3.3.5 `modules/active/file_contexts.template File`

This contains a copy all the modules 'Labeling Policy File' entries (e.g. the `<module_name>.fc` files) that have been extracted from the [base.pp](#) and the loadable modules in the [modules/active/modules](#) directory.

The entries in the `file_contexts.template` file are then used to build the following files:

1. [homedir_template](#) file that will be used to produce the [file_contexts.homedirs](#) file which will then become the policies `./contexts/files/file_contexts.homedirs` file.
2. [file_contexts](#) file that will become the policies `./contexts/files/file_contexts` file.

The way these two files are built is as follows (and shown in [Figure 3-20](#)):

homedir_template - Any line in the `file_contexts.template` file that has the keywords `HOME_ROOT` or `HOME_DIR` are extracted and added to the `homedir_template` file. This is because these keywords are used to identify entries that are associated to a users home directory area. These lines can also have the `ROLE` keyword declared.

The `homedir_template` file will then be used by `genhomedircon(8)`³³ to generate individual SELinux user entries in the `file_contexts.homedirs` file as discussed in the [./modules/active/file_contexts.homedirs](#) section.

file_contexts - All other lines are extracted and added to the `file_contexts` file as they are files not associated to a users home directory.

³³ The `genhomedircon` command has now been built into the `libsepol` library as a function to build the `file_contexts.homedirs` file via `semanage`.

The SELinux Notebook - The Foundations

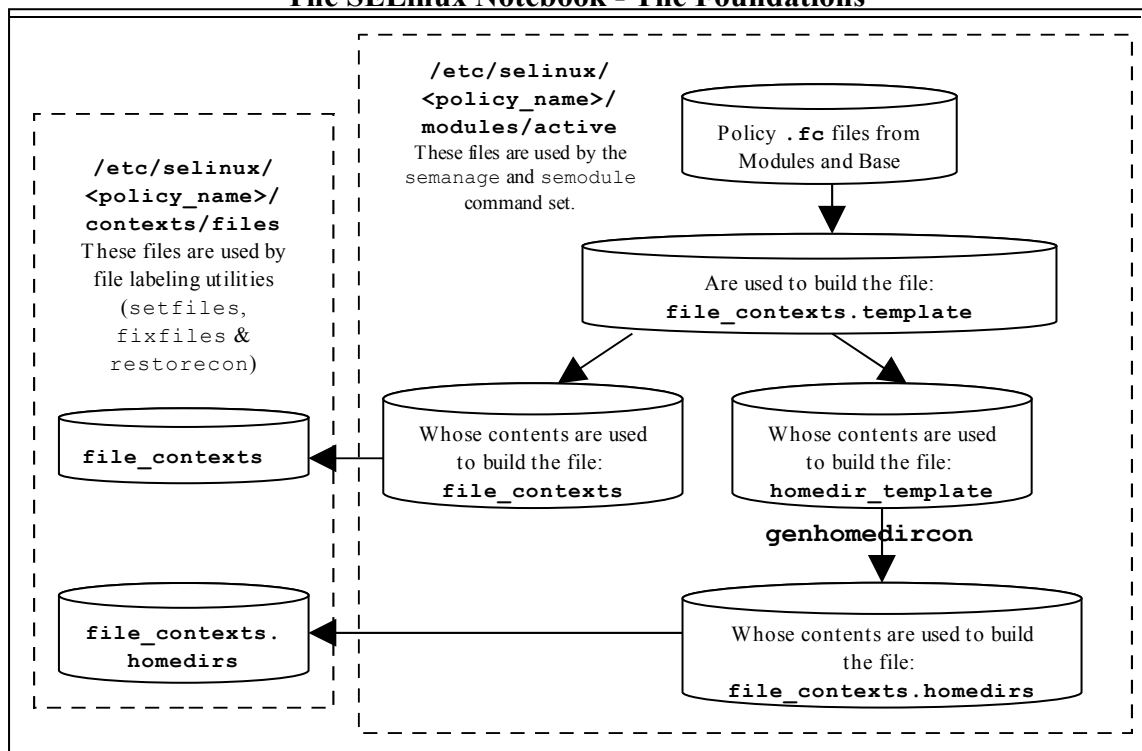


Figure 3-20: File Context Configuration Files - The two files copied to the policy area will be used by the file labeling utilities to relabel files.

The format of the `file_contexts.template` file is as follows:

Each line within the file consists of either type of entry:

```
pathname_regexp opt_security_context
```

Or

```
pathname_regexp file_type opt_security_context
```

Where:

`pathname_regexp`

An entry that defines the pathname in the form of a regular expression.

The metacharacters '^' (match beginning of line) and '\$' (match end of line) are automatically added to the expression by the routines that process this file, however they can be overridden by using '.' *' at either the beginning or end of the expression (see the example `file_contexts` files below).

There are also keywords of `HOME_ROOT`, `HOME_DIR`, `ROLE` and `USER` that are used by file labeling commands (see the [keyword definitions below](#) and the [./modules/active/homedir_template](#) file section for their usage).

`file_type`

The `file_type` options are:

'-b' - Block Device Device '-c' - Character Device

'-d' - Directory '-p' - Named Pipe

'-l' - Symbolic Link '-s' - Socket

'--' - Ordinary file

opt_security_context This entry can be either:

- a. The security context, including the MLS / MCS level or range if applicable that will be assigned to the file.
- b. A value of <<none>> can be used to indicate that the matching files should not be re-labeled.

Keywords that can be in the file_contexts.template file are:

HOME_ROOT This keyword is replaced by the GNU / Linux users root home directory, normally '/home' in a default F-10 configuration.

HOME_DIR This keyword is replaced by the GNU / Linux users home directory, normally '/home/' in a default F-10 configuration.

ROLE This keyword is replaced by the 'prefix' entry from the users_extra configuration file that corresponds to the SELinux users user id. Example users_extra configuration file entries are:

```
user user_u prefix user;
user staff_u prefix staff;
user group1_u prefix group1;
```

It is used for files and directories within the users home directory area when relabeling takes place to allow the domain context to be based on a specific role (or any identifier !!) to allow easier identification in log files.

It can be added by the semanage user command as follows:

```
# Add prefix for SELinux user:
semanage user -a -R staff_r -P group1 group1_u

# Add login user:
semanage login -a -s group1_u rch
```

The usage is similar to the Reference Policy 'per_role_template' (<param name="userdomain_prefix">) that is an optional component of the external interface file (see the ftp.if or ssh.if files in the Reference Policy source). This feature will probably be removed as the semanage

user -P option is more flexible !!!.

USER This keyword will be replaced by the users GNU / Linux user id.

Example file_contexts.template contents:

```
# ./modules/active/file_contexts.template - These sample entries
# have been taken from the F-10 Reference Policy and show the
# HOME_DIR, HOME_ROOT keywords whose lines will be extracted and
# added to the homedir_template file that is used to manage
# user home directory entries. The USER keyword will be replaced
# by the file labeling utilities with the corresponding GNU /
# Linux user id. The ROLE keyword will be replaced by the prefix
# assigned to the SELinux seuser_id taken from the users_extra
# file.

/*                system_u:object_r:default_t
/a?quota\.(user|group) -- system_u:object_r:quota_db_t
/xen(/.*)?        system_u:object_r:xen_image_t
/dev/mcdx?        -b system_u:object_r:removable_device_t
HOME_DIR/.+       system_u:object_r:user_home_t
/var/log/*        system_u:object_r:var_log_t
/tmp/gconfd-USER/* -- system_u:object_r:gconf_tmp_t
/var/log/sxid\.log.* -- system_u:object_r:sxid_log_t
/var/log/messages[^/]* system_u:object_r:var_log_t
/var/run/wnn-unix(/.*) system_u:object_r:canna_var_run_t
HOME_DIR/\.ircmotd -- system_u:object_r:ROLE_irc_home_t
HOME_ROOT/lost\+found/* <<none>>
HOME_DIR/\.config/gtk-* system_u:object_r:gnome_home_t
```

3.3.6 modules/active/file_contexts File

This file becomes the policies [./contexts/files/file_contexts](#) file and is built from entries in the [./modules/active/file_contexts.template](#) file as explained above and shown in [Figure 3-20](#). It is then used by the file labeling utilities to ensure that files and directories are labeled according to the policy.

The format of the file_contexts file is the same as the [./modules/active/file_contexts.template](#) file.

The USER keyword is replaced by the users GNU / Linux user id when the file labeling utilities are run.

Example file_contexts contents:

```
# ./modules/active/file_contexts - These sample entries have been
# taken from the F-10 Reference Policy and show the USER keyword
# that will be replaced by the users GNU / Linux user id when the
# file labeling utilities are run.
# The other keywords HOME_DIR, HOME_ROOT and ROLE have been
# extracted and put in the homedir_template file.

/*                system_u:object_r:default_t
/a?quota\.(user|group) -- system_u:object_r:quota_db_t
/xen(/.*)?        system_u:object_r:xen_image_t
```

```
/dev/mcdx?          -b system_u:object_r:removable_device_t
/var/log/.*         system_u:object_r:var_log_t
/tmp/gconfd-USER/. * -- system_u:object_r:gconf_tmp_t
/var/log/sxid\.log.* -- system_u:object_r:ssid_log_t
/var/log/messages[^/]* system_u:object_r:var_log_t
/var/run/wnn-unix(/.*) system_u:object_r:canna_var_run_t
```

```
# ./contexts/files/file_contexts - Sample entries taken from the
# MLS F-10 reference policy.
```

```
# Notes:
```

```
# 1) The fixed_disk_device_t is labeled SystemHigh (s15:c0.c255)
# as it needs to be trusted. Also some logs and configuration
# files are labeled SystemHigh as they contain sensitive
# information used by trusted applications.
#
# 2) Some directories (e.g. /tmp) are labeled
# SystemLow-SystemHigh (s0-s15:c0.c255) as they will
# support polyinstantiated directories.
```

```
/. *                system_u:object_r:default_t:s0
/a?quota\(user|group) -- system_u:object_r:quota_db_t:s0
/mnt(/[^/]*)       -l system_u:object_r:mnt_t:s0
/mnt(/[^/]*)/*.*   <<none>>
/dev/. *mouse.*    -c system_u:object_r:mouse_device_t:s0
/dev/. *tty[^/]*   -c system_u:object_r:tty_device_t:s0
/dev/[shmx]d[^/]* -b system_u:object_r:fixed_disk_device_t:s15:c0.c255
/var/[xgk]dm(/.*)? system_u:object_r:xsrvr_log_t:s0
/dev/(raw/)?rawctl -c system_u:object_r:fixed_disk_device_t:s15:c0.c255
/tmp               -d system_u:object_r:tmp_t:s0-s15:c0.c255
dev/pts           -d system_u:object_r:devpts_t:s0-s15:c0.c255
/var/log          -d system_u:object_r:var_log_t:s0-s15:c0.c255
/var/tmp          -d system_u:object_r:tmp_t:s0-s15:c0.c255
/var/run          -d system_u:object_r:var_run_t:s0-s15:c0.c255
/usr/tmp          -d system_u:object_r:tmp_t:s0-s15:c0.c255
```

3.3.7 modules/active/homedir_template File

This file is built from entries in the [file_contexts.template](#) file (as shown in [Figure 3-20](#)) and explained in the [./modules/active/file_contexts.template](#) section.

The file is used by `genhomedircon`, `semanage login` or `semanage user` to generate individual user entries in the [file_contexts.homedirs](#) file.

The `homedir_template` file has the same per line format as the [./modules/active/file_contexts.template](#) file.

Example file contents:

```
# ./modules/active/homedir_template - These sample entries have
# been taken from the F-10 Reference Policy and show the
# HOME_DIR, HOME_ROOT and ROLE keywords that are used to manage
# users home directories:

HOME_DIR/.+        system_u:object_r:user_home_t
HOME_DIR/\.ircmotd -- system_u:object_r:ROLE_irc_home_t
HOME_ROOT/lost\+found/. * <<none>>
```

```
HOME_DIR/\.config/gtk-.* system_u:object_r:gnome_home_t
```

3.3.8 modules/active/file_contexts.homedirs File

This file becomes the policies

[./contexts/files/file_contexts.homedirs](#) file when building policy as shown in [Figure 3-20](#). It is then used by the file labeling utilities to ensure that users home directory areas are labeled according to the policy.

The file can be built by the `genhomedircon` command (that in F-10 just calls `/usr/sbin/semodule -Bn`) or if using `semanage` with `user` or `login` options to manage users, where it is called automatically as it is now a `libsepol` library function.

The `file_contexts.homedirs` file has the same per line format as the [./modules/active/file_contexts.template](#) file, however the `HOME_DIR`, `ROOT_DIR` and `ROLE` keywords will be replaced as explained in the [keyword definitions](#) section above. Note that the `ROLE` keyword will only be replaced for those valid types within the policy (for example if `staff IRC home_t` cannot be found in the policy it will be silently dropped from the `file_context.homedirs` when being built **True?**

Example file_contexts.homedirs contents:

```
# ./modules/active/file_contexts.homedirs - These sample entries
# have been taken from the F-10 Reference Policy and show that
# the HOME_DIR, HOME_ROOT and ROLE keywords have been replaced
# by entries as explained above.
#
# User-specific file contexts, generated via libsemanage
# use semanage command to manage system users to change the file_context
#
# Home Context for user user_u
/home/.+ system_u:object_r:user_home_t
/home/\.ircmotd -- system_u:object_r:user IRC home_t
/home/lost\+found/.+ <<none>>
/home/\.config/gtk-.* system_u:object_r:gnome_home_t

# Home Context for user root
/root/.+ system_u:object_r:user_home_t
/root/\.ircmotd -- system_u:object_r:user IRC home_t
/root/lost\+found/.+ <<none>>
/root/\.config/gtk-.* system_u:object_r:gnome_home_t
```

3.3.9 modules/active/netfilter_contexts & netfilter.local File

These files do not seem to be used at present. There is code to produce a `netfilter_contexts` file for use by the GNU/Linux `iptables` service³⁴ in the Reference Policy that would generate a file similar to the example below, however there seems much debate on how they should be managed (see [bug 201573 – Secmark iptables integration](#) for details).

³⁴ This uses `SECMARK` labeling that has been utilised by SELinux as described in the [SELinux Networking Support](#) section.

Example netfilter_contexts contents:

```
# This is an example that would be generated by the Reference
# Policy, however seems on hold.

# This is the standard iptables header:
*mangle
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
:selinux_input - [0:0]
:selinux_output - [0:0]
:selinux_new_input - [0:0]
:selinux_new_output - [0:0]
-A INPUT -j selinux_input
-A OUTPUT -j selinux_output
-A selinux_input -m state --state NEW -j selinux_new_input
-A selinux_input -m state --state RELATED,ESTABLISHED -j CONNSECMARK --restore
-A selinux_output -m state --state NEW -j selinux_new_output
-A selinux_output -m state --state RELATED,ESTABLISHED -j CONNSECMARK --restore
-A selinux_new_input -j SECMARK --selctx system_u:object_r:server_packet_t:s0

# These entries are built from the ports defined in the policy:
-A selinux_new_input -p udp --dport 7007 -j SECMARK --selctx system_u:object_r:afs_bos_server_packet_t:s0
-A selinux_new_input -p tcp --dport 2040 -j SECMARK --selctx system_u:object_r:afs_fs_server_packet_t:s0
-A selinux_new_input -p udp --dport 7000 -j SECMARK --selctx system_u:object_r:afs_fs_server_packet_t:s0
-A selinux_new_input -p udp --dport 7005 -j SECMARK --selctx system_u:object_r:afs_fs_server_packet_t:s0
.....
.....
# This is the standard iptables trailer:
-A selinux_new_input -j CONNSECMARK --save
-A selinux_new_input -j RETURN
-A selinux_new_output -j CONNSECMARK --save
-A selinux_new_output -j RETURN
COMMIT
```

3.3.10 modules/active/policy.kern File

This is the binary policy file built by either the `semanage` or `semodule` process (depending on the configuration action), that is then copied as the [./policy/policy.23](#) binary policy that will be loaded into the kernel.

3.3.11 modules/active/seusers.final and seusers Files

The `seusers.final` file maps GNU / Linux users to SELinux users and becomes the policies `seusers`³⁵ file as discussed in the [./seusers](#) section. The `seusers.final` file is built or modified when:

1. Building a policy where an optional `seusers` file has been included in the base package via the `semodule_package(8)` command (signified by the `-s` flag) as follows³⁶:

```
semodule_package -o base.pp -m base.mod -s seusers ...
```

³⁵ Many `seusers` make confusion: The `./modules/active/seusers` file is used to hold initial `seusers` entries, the `./modules/active/seusers.final` file holds the complete entries that then becomes the policy `./seusers` file.

³⁶ The Reference Policy Makefile 'Rules.modular' script uses this method to install the initial `seusers` file.

The `seusers` file would be extracted by the subsequent `semodule` command when building the policy to produce the `seusers.final` file.

2. The `semanage login` command is used to map GNU / Linux users to SELinux users as follows:

```
semanage login -a -s staff_u rch
```

This action will update the `seusers` file that would then be used to produce the `seusers.final` file with both policy and locally defined user mapping.

The format of the `seusers.final` & `seusers` files are as follows:

```
user_id:seuser_id [:range]
```

Where:

<code>user_id</code>	The GNU / Linux user identity.
<code>seuser_id</code>	The SELinux user identity.
<code>range</code>	The optional range as defined in the MLS range definition section.

Example `seusers.final` file contents:

```
# ./modules/active/seusers.final
system_u:system_u
root:root
__default__:user_u
```

Example `semanage login` command to add a GNU / Linux user mapping:

```
# This command will add the rch:user_u entry in the seusers file:
semanage login -a -s user_u rch
```

The resulting `seusers` file would be:

```
# ./modules/active/seusers
rch:user_u
```

The `seusers.final` file that will become the `./<policy_name>/seusers` file is as follows:

```
# ./modules/active/seusers.final
system_u:system_u
root:root
__default__:user_u
rch:user_u
```

3.3.12 modules/active/users_extra, users_extra.local and users.local Files

These three files work together to describe SELinux user information as follows:

1. The `users_extra` and `users_extra.local` files are used to map a prefix to users home directories as discussed in the [./modules/active/file_contexts.template](#) file section, where it is used to replace the `ROLE` keyword. The prefix is linked to an SELinux user id and should reflect the users role. The `semanage user` command will allow a prefix to be added via the `-P` flag.

The `users_extra` file contains all the policy prefix entries, and the `users_extra.local` file contains those generated by the `semanage user` command.

The `users_extra` file can optionally be included in the base package via the `semodule_package(8)` command (signified by the `-u` flag) as follows³⁷:

```
semodule_package -o base.pp -m base.mod -u users_extra ...
```

The `users_extra` file would then be extracted by a subsequent `semodule` command when building the policy.

2. The `users.local` file is used to add new SELinux users to the policy without editing the policy source itself (with each line in the file following a policy language [user Statement](#)). This is useful when only the Reference Policy headers are installed and additional users need to added. The `semanage user` command will allow a new SELinux user to be added that would generate the `user.local` file and if a `-P` flag has been specified, then a `users_extra.local` file is also generated (note: if this is a new SELinux user and a prefix is not specified a default prefix of `user` is generated).

The sections that follow will:

- Define the format and show example `users_extra` and `users_extra.local` files.
- Execute an `semanage user` command that will add a new SELinux user and associated prefix, and show the resulting `users_extra`, `users_extra.local` and `users.local` files.

Note that each line of the `users.local` file contains a user statement that is defined in the policy language [user Statement](#) section, and will be built into the policy via the `semanage` command.

The format of the `users_extra` & `users_extra.local` files are as follows:

```
user seuser_id prefix prefix_id;
```

Where:

³⁷ The Reference Policy Makefile 'Rules.modular' script uses this method to install the initial `users_extra` file.

user	The user keyword.
seuser_id	The SELinux user identity.
prefix	The prefix keyword.
prefix_id	An identifier that will be used to replace the ROLE keyword within the <code>./modules/active/homedir_template</code> file when building the <code>./modules/active/file_contexts.homedirs</code> file for the relabeling utilities to set the security context on users home directories.

Example `users_extra` file contents:

```
# ./modules/active/users_extra entries, note that the
# users_extra.local file contents are similar and generated by
# the semanage user command.

user user_u prefix user;
user staff_u prefix user;
user sysadm_u prefix user;
user root prefix user;
```

Example `semanage user` command to add a new SELinux user:

```
# This command will add the user test_u prefix staff entry in
# the users_extra.local file:

semanage user -a -R staff_r -P staff test_u
```

The resulting `users_extra.local` file is as follows:

```
# ./modules/active/users_extra.local

user test_u prefix staff;
```

The resulting `users_extra` file is as follows:

```
# ./modules/active/users_extra

user user_u prefix user;
user staff_u prefix user;
user sysadm_u prefix user;
user root prefix user;
user test_u prefix staff;
```

The resulting `users.local` file is as follows:

```
# ./modules/active/users.local file entry:

user test_u roles { staff_r } level s0 range s0;
```


3.3.13 `modules/active/booleans.local` File

This file is created and updated by the `semanage boolean` command and holds boolean value as requested. It should be noted that instead of using this file, the command allows a different file to be specified (see the `semanage man` page).

Example `semanage boolean` command to modify a boolean value:

```
# This command will add an entry in the booleans.local
# file and set the boolean value to 'off':

semanage boolean -m -0 ext_gateway_audit
```

The resulting `booleans.local` file would be:

```
# ./modules/active/booleans.local

ext_gateway_audit=0
```

3.3.14 `modules/active/file_contexts.local` File

This file is created and updated by the `semanage fcontext` command. It is used to hold file context information on files and directories that were not delivered by the core policy (i.e. they are not defined in any of the `*.fc` files delivered in the base and loadable modules).

The `semanage` command will add the information to the policy stores `file_contexts.local` file and then copy this file to the `./contexts/files/file_contexts.local` file, where it will be used when the file context utilities are run.

The format of the `file_contexts.local` file is the same as the [./modules/active/file_contexts.template](#) file.

Example `semanage fcontext` command to add a new entry:

```
# This command will add an entry in the file_contexts.local
# file:

semanage fcontext -a -t user_t /usr/move_file

# Note that the type (-t flag) must exist in the policy
# otherwise the command will fail.
```

The resulting `file_contexts.local` file would be:

```
# ./modules/active/file_contexts.local

/usr/move_file    system_u:object_r:user_t
```

3.3.15 `modules/active/interfaces.local` File

This file is created and updated by the `semanage interface` command to hold network interface information that was not delivered by the core policy (i.e. they are not defined in `base.conf` file). The new interface information is then built into the policy by the `semanage` process.

Each line of the file contains a `netifcon` statement that is defined along with examples in the [netifcon Statement](#) section.

3.3.16 `modules/active/nodes.local` File

This file is created and updated by the `semanage node` command to hold network address information that was not delivered by the core policy (i.e. they are not defined in `base.conf` file). The new node information is then built into the policy by the `semanage` process.

Each line of the file contains a `nodecon` statement that is defined along with examples in the policy language [nodecon Statement](#) section.

3.3.17 `modules/active/ports.local` File

This file is created and updated by the `semanage port` command to hold network port information that was not delivered by the core policy (i.e. they are not defined in `base.conf` file). The new port information is then built into the policy by the `semanage` process.

Each line of the file contains a `portcon` statement that is defined along with examples in the policy language [portcon Statement](#) section.

3.3.18 `modules/active/modules` Directory Contents

This directory contains the loadable modules (`<module_name>.pp`) that have been packaged by the `semodule_package` command and placed in the store by the `semodule` command as shown in the following example:

```
# Package the module move_file_c:

semodule_package -o move_file_c.pp -m move_file_c.mod -f
  move_file.fc

# Then to install it in the store (at /etc/selinux/modular-test/
# modules/active/modules/move_file_c.pp) and build the binary
# policy file, run the semodule command:

semodule -v -s modular-test -i move_file_c.pp
```

3.4 Policy Configuration Files

Each file discussed in this section is relative to the policy name as follows:

```
/etc/selinux/<policy_name>
```

The majority of files are installed by the Reference Policy, `semanage` or `semodule` processes. It is possible to build custom monolithic policies that only use the files installed in this area (i.e. do not use `semanage` or `semodule`). For example the simple monolithic policy described in the [Building a Basic Policy](#) section could run at `init 3` (i.e. no X-Windows), and only require the following configuration files:

- `./policy/policy.23` – The binary policy loaded into the kernel.
- `./context/files/file_contexts` – To allow the filesystem to be relabeled.

If the simple policy is to run at `init 5`, (i.e. with X-Windows) then an additional file is required:

- `./context/dbus_contexts` – To allow the dbus messaging service to run under SELinux.

3.4.1 seusers File

This file is used by login programs (normally via the `libselinux` library) and maps GNU / Linux users (as defined in the `user/passwd` files) to SELinux users (defined in the policy). A typical login sequence would be:

- Using the GNU / Linux `user_id`, lookup the `seuser_id` from this file. If an entry cannot be found, then use the `__default__` entry.
- To determine the remaining context to be used as the security context, read the [./contexts/users/\[seuser_id\]](#) file. If this file is not present, then:
 - Check for a default context in the [./contexts/default_contexts](#) file. If no default context is found, then:
 - Read the [./contexts/failsafe_context](#) file to allow a fail safe context to be set.

Note: The `system_u` user is defined in this file, however there must be **no** `system_u` GNU / Linux user configured on the system.

The format of the `seusers` file is the same as the files described in the [./modules/active/seusers.final and seusers](#) section, where an example `semanage user` command is also shown.

Example `seusers` file contents:

```
# ./seusers file for non-MCS/MLS systems.

system_u:system_u
root:root
fred:user_u
__default__:user_u
```

```
# ./seusers file for an MLS system. Note that the system_u user
# has access to all security levels and therefore should not be
# configured as a valid GNU / Linux user.
```

```
system_u:system_u:s0-s15:c0.c255
root:root:s0-s15:c0.c255
fred:user_u:s0
__default__:user_u:s0
```

Supporting `libselinux` API functions are:

```
getseuser
getseuserbyname
```

3.4.2 `setrans.conf` File

This file is used by the `mcstransd(8)` daemon (available in the `mcstrans` rpm). The daemon enables SELinux-aware applications to translate the MCS / MLS internal policy levels into user friendly labels.

The daemon will not load unless a valid MCS or MLS policy is active.

The `semanage` command can be used to update this file.

The file format is as follows:

```
# Enable / disable translation service:
disable=1|0

# Each line consists of a level or range with user friendly
# label:
level|range=text_label
```

Where:

<code>disable</code>	To disable the translation service, set <code>disable=1</code> . To enable the service comment out the entry or set <code>disable=0</code> .
<code>range</code>	The optional level or range as defined in the MLS range definition section.
<code>text_label</code>	The user friendly label to be displayed by SELinux-aware applications that use the translation service API.

Example file contents:

```
# ./setrans.conf - Taken from the F-10 reference policy.
#
# Multi-Level Security translation table for SELinux
#
# Uncomment the following to disable translation library
# disable=1
#
# SystemLow and SystemHigh
s0=SystemLow
s15:c0.c1023=SystemHigh
```

```
s0-s15:c0.c1023=SystemLow-SystemHigh

# Unclassified level
s1=Unclassified

# Secret level with compartments
s2=Secret
s2:c0=A
s2:c1=B

# ranges for Unclassified
s0-s1=SystemLow-Unclassified
s1-s2=Unclassified-Secret
s1-s15:c0.c1023=Unclassified-SystemHigh

# ranges for Secret with compartments
s0-s2=SystemLow-Secret
s2:c1-s15:c0.c1023=Secret:B-SystemHigh
s2:c0,c1-s15:c0.c1023=Secret:AB-SystemHigh
```

Example semanage command:

```
# Add a new entry to the file. Note that the -T flag component
# (the user friendly name for the level) must not have spaces.

semanage translation -a -T Top-Level s15:c1023
```

```
# List the setrans.conf file contents

semanage translation -l

...
s15:c1023=Top-Level
```

Supporting libselinux API functions are:

```
selinux_translations_path
selinux_raw_to_trans_context
selinux_trans_to_raw_context
```

3.4.3 policy/policy.23 File

This is the binary policy file that is loaded into the kernel to enforce policy and is built by either `checkpolicy` or `semodule` (the [Building a Basic Policy](#) section shows these command in use). Life is too short to describe the format but the `libsepol` source could be used as a reference or for an overview the “[SELinux Policy Module Primer](#)” [Ref. 4] notes.

The file name extension (‘.23’ on F-10), is the policy database version supported by the GNU / Linux release and can be found by executing the following command:

```
cat /selinux/policyvers
23
```

The different versions are discussed in the [Policy Versions](#) section.

3.4.4 contexts/customizable_types File

This file contains a list of types that will not be relabeled by the `setfiles(8)` or `restorecon(8)` commands. The commands check this file before relabeling and excludes those in the list unless the `-F` flag is used (see the man pages).

The file format is as follows:

```
type
```

Where:

<code>type</code>	The <code>type</code> defined in the policy that needs to be excluded from relabeling. An example is when a file has been purposely relabeled with a different type to allow an application to work.
-------------------	--

Example file contents:

```
# ./contexts/customizable_types - Taken from the F-10 reference
# policy.

mount_loopback_t
public_content_rw_t
public_content_t
swapfile_t
sysadm_untrusted_content_t
sysadm_untrusted_content_tmp_t
```

Supporting libselinux API functions are:

```
is_context_customizable
selinux_customizable_types_path
selinux_context_path
```

3.4.5 contexts/default_contexts File

Used by SELinux-aware applications that need to set a security context for user processes (generally the login applications) where:

1. The GNU / Linux user identity should be known by the application.
2. If a login application, then the SELinux user (`seuser`), would have been determined as described in the [seusers](#) file section.
3. The login applications will check the `./contexts/users/[seuser_id]` file first and if no valid entry, will then look in the `[seuser_id]` file for a default context to use.

The file format is as follows:

```
role:type role:type ...
```

Or:

```
role:type:range role:type:range ...
```

Where:

`role:type` The file contains one or more lines that consist of `role:type` pairs.

The entry at the start of a new line corresponds to the partial `role:type` context of (generally) the login application.

The other `role:type` entries on that line represent an ordered list of valid contexts that could be used to set the users context.

`range` The range as defined in the [MLS range definition](#) section.

Example file contents:

```
# ./contexts/default_contexts - Taken from the F-10 reference
# policy. The highlighted entry at the start of each line
# corresponds to the login applications role:type context.

system_r:cron_d_t      user_r:user_cron_d_t staff_r:staff_cron_d_t
sysadm_r:sysadm_cron_d_t system_r:system_cron_d_t unconfined_r:unconfined_cron_d_t
#
system_r:local_login_t user_r:user_t staff_r:staff_t sysadm_r:sysadm_t
unconfined_r:unconfined_t
#
system_r:remote_login_t user_r:user_t staff_r:staff_t unconfined_r:unconfined_t
#
system_r:sshd_t        user_r:user_t staff_r:staff_t sysadm_r:sysadm_t
unconfined_r:unconfined_t
```

```
# ./contexts/default_contexts - Taken from the MLS F-10
# reference policy.

system_r:cron_d_t:s0      system_r:system_cron_d_t:s0
system_r:local_login_t:s0 user_r:user_t:s0
system_r:remote_login_t:s0 user_r:user_t:s0
system_r:sshd_t:s0       user_r:user_t:s0
system_r:sulogin_t:s0    sysadm_r:sysadm_t:s0
system_r:xdm_t:s0        user_r:user_t:s0
```

Supporting libselinux API functions are:

```
# Note that the ./contexts/users/[seuser_id] file is also read
# by some of these functions.

selinux_contexts_path
selinux_default_context_path
get_default_context
get_ordered_context_list
```

```
get_ordered_context_list_with_level
get_default_context_with_level
get_default_context_with_role
get_default_context_with_rolelevel
query_user_context
manual_user_enter_context
get_default_role
```

An example use in this Notebook (to get over a small feature) is that when the initial [basic policy](#) was built, no `default_contexts` file entries were required as only one `role:type` of `unconfined_r:unconfined_t` had been defined, therefore the login process did not need to decide anything (as the only user context was `user_u:unconfined_r:unconfined_t`).

However when adding the [loadable module](#) that used another type (`ext_gateway_t`) but with the same role and user (e.g. `user_u:unconfined_r:ext_gateway_t`), then it was found that the login process would always set the logged in user context to `user_u:unconfined_r:ext_gateway_t` (i.e. the login application now had a choice and choose the wrong one, probably because the types are sorted and 'e' comes before 'u').

The end result was that as soon as enforcing mode was set, the system got bitter and twisted. To resolve this the `default_contexts` file entries were set to:

```
unconfined_r:unconfined_t unconfined_r:unconfined_t
```

The login process could now set the context correctly to `unconfined_r:unconfined_t`. Note that adding the same entry to the `contexts/users/user_u` configuration file instead could also have achieved this.

3.4.6 `contexts/debus_contexts` File

This file is for the dbus messaging service daemon (a form of IPC) that is used by a number of GNU / Linux applications such as GNOME and KDE desktops. If SELinux is enabled, then this file needs to exist in order for these applications to work. The [dbus-daemon](#) man page details the contents, however it is not recommended that this file is changed. The Free Desktop web site has detailed information at:

<http://dbus.freedesktop.org>

Example file contents:

```
# ./contexts/debus_contexts - Taken from the F-10 reference
# policy.

<!DOCTYPE busconfig PUBLIC "-//freedesktop//DTD D-BUS Bus
Configuration 1.0//EN" "http://www.freedesktop.org/standards/dbus/
1.0/busconfig.dtd">
<busconfig>
  <selinux>
  </selinux>
</busconfig>
```


Supporting `libselinux` API function is:

```
selinux_context_path
```

3.4.7 `contexts/default_type` File

This file allows SELinux-aware applications such as `newrole(1)` to select a default type for a role if one is not supplied. An example use is by `newrole` when it is called to change a users role, with no type specified, this file would then be consulted to determine the default type to use for the requested role.

The file format is as follows:

```
role:type
```

Where:

`role:type` The file contains one or more lines that consist of `role:type` entries. There should be one line for each role defined within the policy.

Example file contents:

```
# ./contexts/default_type - Taken from the F-10 reference
# policy.

auditadm_r:auditadm_t
secadm_r:secadm_t
sysadm_r:sysadm_t
staff_r:staff_t
unconfined_r:unconfined_t
user_r:user_t
```

Supporting `libselinux` API functions are:

```
selinux_context_path
get_default_type
```

3.4.8 `contexts/failsafe_context` File

If the login process cannot determine a default context to use, then this can be set to allow an administrator access to the system by setting a known valid context.

The file format is as follows:

```
role:type
```

Or:

```
role:type:range
```

Where:

- `role:type` The file contains a single line that consist of a `role:type` for a known valid context to allow an administrator access to the system..
- `range` The range as defined in the [MLS range definition](#) section.

Example file contents:

```
# ./contexts/failsafe_context - Taken from the F-10 reference
# policy.

sysadm_r:sysadm_t
```

```
# ./contexts/failsafe_context - Taken from the MLS F-10
# reference policy.

sysadm_r:sysadm_t:s0
```

Supporting `libselinux` API functions are:

```
selinux_context_path
selinux_failsafe_context_path
```

3.4.9 `contexts/initrc_context` File

This is used by the `run_init(8)` command to allow system services to be started in the same security context as `init`. This file could also be used by other SELinux-aware applications for the same purpose.

The file format is as follows:

```
security_context
```

Where:

- `security_context` The file contains one line that consists of a full security context, including the MLS / MCS level or range if applicable.

Example file contents:

```
# ./contexts/initrc_context - Taken from the F-10 reference
# policy.

system_u:system_r:initrc_t
```

```
# ./contexts/initrc_context - Taken from the MLS F-10 reference
# policy. Note that the init process has full access via the
# range s0-s15:c0.c255.
```

```
system_u:system_r:initrc_t:s0-s15:c0.c255
```

Supporting `libselinux` API functions are:

```
selinux_context_path
```

3.4.10 `contexts/netfilter_contexts` File

This file will support the Secmark labeling for Netfilter / iptable rule matching of network packets, however it is currently unused (see the [./modules/active/netfilter_contexts & netfilter.local](#) file section for further information).

Supporting `libselinux` API functions are:

```
selinux_context_path  
selinux_netfilter_context_path
```

3.4.11 `contexts/removable_contexts` File

This file contains the default label that should be used for removable devices that are not defined in the [contexts/files/media](#) file.

The file format is as follows:

```
security_context
```

Where:

`security_context` The file contains one line that consists of a full security context, including the MLS / MCS level or range if applicable.

Example file contents:

```
# ./contexts/removable_contexts - Taken from the F-10 reference  
# policy.  
  
system_u:object_r:removable_t
```

```
# ./contexts/removable_contexts - Taken from the MLS F-10  
# reference policy.  
  
system_u:object_r:removable_t:s0
```

Supporting `libselinux` API functions are:

```
selinux_removable_context_path
```

3.4.12 contexts/securetty_types File

This file is used by the `newrole(1)` command to find the `type` to use with `tty` devices when changing roles or levels.

The file format is as follows:

```
type
```

Where:

`type` Zero or more `type` entries that are defined in the policy for `tty` devices.

Example file contents:

```
# ./contexts/securetty_types - Taken from the F-10 reference
# policy.

sysadm_tty_device_t
user_tty_device_t
staff_tty_device_t
```

```
# ./contexts/securetty_types - Taken from the MLS F-10 reference
# policy.

sysadm_tty_device_t
user_tty_device_t
staff_tty_device_t
auditadm_tty_device_t
secureadm_tty_device_t
```

Supporting `libselinux` API functions are:

```
selinux_securetty_types_path
```

3.4.13 contexts/userhelper_context File

This file contains the default security context used by the `system-config-*` applications when running from `root`.

The file format is as follows:

```
security_context
```

Where:

`security_context` The file contains one line that consists of a full security context, including the `MLS / MCS level` or `range` if applicable.

Example file contents:

```
# ./contexts/userhelper_context - Taken from the F-10 reference
# policy.

system_u:sysadm_r:sysadm_t
```

```
# ./contexts/userhelper_context - Taken from the MLS F-10
# reference policy.

system_u:sysadm_r:sysadm_t:s0
```

Supporting libselinux API functions are:

```
selinux_context_path
```

3.4.14 contexts/x_contexts File

This file provides the security contexts (and other configuration information) for the X-Windows SELinux security extensions provided via Xace (X access control extension). No idea how it works (yet anyway). The MCS / MLS version of the file has the appropriate level or range context added.

Example file contents:

```
# ./contexts/x_contexts - Taken from the F-10 reference policy.
#
# Config file for XSELinux extension
#
# The default client rule defines a context to be used for all clients
# connecting to the server from a remote host.
#
client *                system_u:object_r:remote_xclient_t

##
### Rules for X Properties
# Property rules map a property name to a context. A default property
# rule indicated by an asterisk should follow all other property rules.
#
# Properties that normal clients may only read
property XFree86_VT      system_u:object_r:info_xproperty_t
property XFree86_DDC_EDID1_RAWDATA system_u:object_r:info_xproperty_t
##
### Rules for X Extensions
##
# Extension rules map an extension name to a context. A default extension
# rule indicated by an asterisk should follow all other extension rules.
#
# Standard extensions
extension BIG-REQUESTS   system_u:object_r:std_xext_t
extension SHAPE          system_u:object_r:std_xext_t
##
### Rules for X Selections
##
# Selection rules map a selection name to a context. A default selection
# rule indicated by an asterisk should follow all other selection rules.
#
# Standard selections
selection XA_PRIMARY     system_u:object_r:clipboard_xselection_t
##
### Rules for X Events
##
# Event rules map an event protocol name to a context. A default event
```

```
# rule indicated by an asterisk should follow all other event rules.
#
# Input events
event X11:KeyPress                system_u:object_r:input_xevent_t
event X11:KeyRelease              system_u:object_r:input_xevent_t
```

Supporting `libselinux` API functions are:

```
selinux_x_context_path
```

3.4.15 contexts/files/file_contexts File

This file is managed by the `semodule` and `semanage` commands³⁸ as the policy is updated (adding or removing modules or updating the base), and therefore should not be edited.

The file is used by a number of SELinux-aware commands (`setfiles(8)`, `fixfiles(8)`, `matchpathcon(8)`, `restorecon(8)`) to relabel either part or all of the file system.

Note that users home directory file contexts are not present in this file as they are managed by the [file_contexts.homedirs](#) file as explained below.

The format of the `file_contexts` file is the same as the files described in the [./modules/active/file_contexts](#) file section.

Supporting `libselinux` API functions are:

```
selinux_file_context_path
selinux_file_context_verify
selinux_file_context_local_path
```

3.4.16 contexts/files/file_contexts.local File

This file is added by the `semanage fcontext` command as described in the [./modules/active/file_contexts.local](#) file section to allow locally defined files to be labeled correctly.

3.4.17 contexts/files/file_contexts.homedirs File

This file is managed by the `semodule` and `semanage` commands as the policy is updated (adding or removing users and modules or updating the base), and therefore should not be edited.

It is generated by the `genhomedircon(8)` command (in fact by `semodule -Bn` that rebuilds the policy) and used to set the correct contexts on the users home directory and files.

³⁸ As each module would have its own `file_contexts` component that is either added or removed from the policies overall `/etc/selinux/[policy_name]/contexts/files/file_contexts` file.

It is fully described in the [./modules/active/file_contexts.homedirs](#) file section.

Supporting `libselinux` API functions are:

```
selinux_file_context_homedir_path
selinux_homedir_context_path
```

3.4.18 contexts/files/media File

Used to map `media` types to a file context. If the `media_id` cannot be found in this file, then the default context in the [./contexts/removable_contexts](#) is used instead.

The file format is as follows:

```
media_id file_context
```

Where:

<code>media_id</code>	The media identifier (those known are: <code>cdrom</code> , <code>floppy</code> , <code>disk</code> and <code>usb</code>).
<code>file_context</code>	The context to be used for the device. Note that it does not seem to have the <code>MLS / MCS</code> level).

Example file contents:

```
# contexts/files/media - Taken from the F-10 reference policy
# (note that the same file is generated for all types of
# policy).

cdrom system_u:object_r:removable_device_t
floppy system_u:object_r:removable_device_t
disk system_u:object_r:fixed_disk_device_t
```

Supporting `libselinux` API functions are:

```
selinux_media_context_path
```

3.4.19 contexts/users/[seuser_id] File

These optional files are named after the SELinux user they represent (e.g. `seuser_id` = `user_u`). Each file has the same format as the [contexts/default_contexts](#) file and is used to assign the correct context to the SELinux user.

Example file contents:

```
# ./contexts/users/user_u - Taken from the F-10 reference policy.
```

```
system_r:local_login_t      user_r:user_t
system_r:remote_login_t    user_r:user_t
system_r:sshd_t            user_r:user_t
system_r:crond_t           user_r:user_t
```

```
# ./contexts/users/user_u - Taken from the MLS F-10
# reference policy.
```

```
system_r:local_login_t:s0  user_r:user_t:s0
system_r:remote_login_t:s0 user_r:user_t:s0
system_r:sshd_t:s0         user_r:user_t:s0
system_r:crond_t:s0        user_r:user_t:s0
system_r:xdm_t:s0          user_r:user_t:s0
user_r:user_su_t:s0        user_r:user_t:s0
user_r:user_sudo_t:s0      user_r:user_t:s0
```

Supporting libselinux API functions are:

```
selinux_user_contexts_path
selinux_users_path
selinux_usersconf_path
```


4. SELinux Policy Language

4.1 Introduction

This section is intended as a reference to give a basic understanding of the policy language statements and rules with supporting examples taken from the Fedora F-10 policy sources³⁹. Also all of the language updates to Policy DB version 23 should have been captured. For a more detailed explanation of the policy language the “SELinux by Example” [Ref. 14] book is recommended.

4.2 Policy Statements and Rules

4.2.1 Policy Source Files

There are three basic types of policy source file⁴⁰ that can contain language statements and rules (examples of these can be found in the [Building a Basic Policy](#) section). The three types of policy source file⁴¹ are:

Monolithic Policy – This is a single policy source file that contains all statements. By convention this file is called `policy.conf` and is compiled using the `checkpolicy` command that produces the binary policy file.

Base Policy – This is the mandatory base policy source file that supports the loadable module infrastructure. The whole system policy could be fully contained within this file, however it is more usual for the base policy to hold the mandatory components of a policy, with the optional components contained in loadable module source files. By convention this file is called `base.conf` and is compiled using the `checkpolicy` or `checkmodule` command.

Module (or Non-base) Policy – These are optional policy source files that when compiled, can be dynamically loaded or unloaded within the policy store. By convention these files are named after the module or application they represent, with the compiled binary having a `.pp` extension. These files are compiled using the `checkmodule` command.

[Table 4-1](#) shows the order in which the statements should appear in source files with the minimum (and therefore mandatory) statements that must be defined.

³⁹ From the resulting `base.conf` and loadable module sources in the `./tmp` directory after a `make load` had been executed for a reference policy ‘standard’ build and where applicable an ‘mls’ build.

⁴⁰ It is important to note that the [Reference Policy](#), builds policy using makefiles and support macros within its own source file structure. However, the end result of the `make` process is that there can be three possible types of source file built (depending on the `MONOLITHIC=Y/N` build option). These files contain the policy language statements and rules that are finally compiled into a binary policy.

⁴¹ This does not include the `file_contexts` file as it does not contain policy statements, only persistent security contexts (labels) that will be used by files and directories.

<i>Base Entries</i>	<i>M/O</i>	<i>Module Entries</i>	<i>M/O</i>
Security Classes (<code>class</code>)	m	module Statement	o
Initial SIDs	m		
Access Vectors (permissions)	m	require Statement	o
MLS sensitivity, category and level Statements	o		
MLS Constraints	o		
Policy Capability Statements	o		
Attributes	o	Attributes	o
Booleans	o	Booleans	o
Type / Type Alias	m	Type / Type Alias	o
Roles	m	Roles	o
Policy Rules	o	Policy Rules	o
Users	m	Users	o
Constraints	o		
Default SID labeling	m		
<code>fs_use_xattr</code> Statements	o		
<code>fs_use_task</code> and <code>fs_use_trans</code> Statements	o		
<code>genfscon</code> Statements	o		
<code>portcon</code> , <code>netifcon</code> and <code>nodecon</code> Statements	o		

Table 4-1: Base and Module Policy Statements – A Monolithic source file would contain the same statements as the Base Module. The Mandatory policy entries are noted (the type, role and user require at least one entry each).

The language grammar defines what statements and rules can be used within the different types of source file. To highlight these rules, the following table is included in each statement and rule section to show what circumstances each one is valid within a policy source file:

Monolithic Policy	Base Policy	Module Policy
Yes/No	Yes/No	Yes/No

Where:

- Monolithic Policy Whether the statement is allowed within a monolithic policy source file or not.
- Base Policy Whether the statement is allowed within a base (for loadable module support) policy source file or not.
- Module Policy Whether the statement is allowed within the optional loadable module policy source file or not.

[Table 4-3](#) shows a cross reference matrix of statements and rules allowed in each type of policy source file.

4.2.2 Conditional, Optional and Require Statement Rules

The language grammar specifies what statements and rules can be included within [Conditional Policy](#), [Optional Policy](#) statements and the [require statement](#). To highlight these rules the following table is included in each statement and rule section to show what circumstances each one is valid within a policy source file:

Conditional Policy (if) Statement	optional Statement	require Statement
Yes/No	Yes/No	Yes/No

Where:

Conditional Policy (if) Statement	Whether the statement is allowed within a conditional statement (IF / ELSE construct) as described in the if Statement section. Conditional statements can be in all types of policy source file.
optional Statement	Whether the statement is allowed within the optional { rule_list } construct as described in the optional Statement section.
require Statement	Whether the statement keyword is allowed within the require { rule_list } construct as described in the require Statement section.

[Table 4-3](#) shows a cross reference matrix of statements and rules allowed in each of the above policy statements.

4.2.3 MLS Statements and Optional MLS Components

The [MLS Statements](#) section defines statements specifically for MLS support. However when MLS is enabled, there are other statements that require the MLS [Security Context](#) component as an argument, therefore these statements show an example taken from the F-10 [Reference Policy](#) MLS build.

4.2.4 General Statement Information

1. Identifiers can generally be any length but should be restricted to the following characters: a-z, A-Z, 0-9 and _ (underscore).
2. A '#' indicates the start of a comment in policy source files.
3. Statements that were defined in the older NSA documentation have been updated to capture changes such as to prohibit the use of * and ~ in type and role sets (other than in the neverallow statement). Note that some of these changes are not captured by the language grammar, but are managed within the policy_parse.y source code).
4. When multiple source and target entries are shown in a single statement or rule, the compiler (checkpolicy or checkmodule) will expand these to individual statements or rules as shown in the following example:

```
# This allow rule has two target entries console_device_t and
```

```
# tty_device_t:
allow apm_t { console_device_t tty_device_t }:chr_file
    { getattr read write append ioctl lock };

# The compiler will expand this to become:
allow apm_t console_device_t:chr_file { getattr read write
    append ioctl lock };
# and:
allow apm_t tty_device_t:chr_file { getattr read write append
    ioctl lock };
```

Therefore when comparing the actual source code with a compiled binary using (for example) `apol`, `sedispol` or `sedismod`, the results will differ (however the resulting policy rules will be the same).

5. Some statements can be added to a policy (via the policy store) using the `semanage(8)` command. Examples of these are shown where applicable, however the `semanage` man page should be consulted for all the possible command line options.
6. [Table 4-2](#) lists words reserved for the SELinux policy language.

alias	allow	and
attribute	auditallow	auditdeny
bool	category	cfalse
class	clone	common
constrain	ctrue	dom
domby	dominance	dontaudit
else	eq	false
fs_use_task	fs_use_trans	fs_use_xattr
fscon	genfscon	h1
h2	if	incomp
inherits	ipv4_addr	ipv6_addr
l1	l2	level
mlsconstrain	mlsvalidatetrans	module
netifcon	neverallow	nodecon
not	object_r	optional
or	permissive	polycyap
portcon	r1	r2
r3	range	range_transition
require	role	role_transition
roles	sameuser	self
sensitivity	sid	source
t1	t2	t3
target	true	type
type_change	type_member	type_transition
typealias	typeattribute	types
u1	u2	u3
user	validatetrans	version

version_identifier	xor	
--------------------	-----	--

Table 4-2: Policy language reserved words.

7. [Table 4-3](#) shows what policy language statements and rules are allowed within each type of policy source file, and whether the statement is valid within an if / else construct, optional {rule_list}, or require {rule_list} statement.

Statement / Rule	Monolithic Policy	Base Policy	Module Policy	Conditional Statements	optional Statement	require Statement ⁴²
allow	Yes	Yes	Yes	Yes	Yes	No
allow - Role	Yes	Yes	Yes	No	Yes	No
attribute	Yes	Yes	Yes	No	Yes	Yes
auditallow	Yes	Yes	Yes	Yes	Yes	No
auditdeny (Deprecated)	Yes	Yes	Yes	Yes	Yes	No
bool	Yes	Yes	Yes	No	Yes	Yes
category	Yes	Yes	No	No	No	Yes
class	Yes	Yes	No	No	No	Yes
common	Yes	Yes	No	No	No	No
constrain	Yes	Yes	No	No	No	No
dominance - MLS	Yes	Yes	No	No	No	No
dominance - Role (Deprecated)	Yes	Yes	Yes	No	Yes	No
dontaudit	Yes	Yes	Yes	Yes	Yes	No
fs_use_task	Yes	Yes	No	No	No	No
fs_use_trans	Yes	Yes	No	No	No	No
fs_use_xattr	Yes	Yes	No	No	No	No
genfscon	Yes	Yes	No	No	No	No
if	Yes	Yes	Yes	No	Yes	No
level	Yes	Yes	No	No	No	No
mlsconstrain	Yes	Yes	No	No	No	No
mlsvalidatetrans	Yes	Yes	No	No	No	No
module	No	No	Yes	No	No	No
netifcon	Yes	Yes	No	No	No	No
neverallow	Yes	Yes	Yes ⁴³	No	Yes	No
nodecon	Yes	Yes	No	No	No	No
optional	No	Yes	Yes	Yes	Yes	Yes
permissive	Yes	Yes	Yes	Yes	Yes	No
polycap	Yes	Yes	No	No	No	No
portcon	Yes	Yes	No	No	No	No
range_transition	Yes	Yes	Yes	No	Yes	No
require	No	Yes ⁴⁴	Yes	Yes ⁴⁵	Yes	No

⁴² Only the statement keyword is allowed.

⁴³ This should be allowed, however the F-10 checkmodule command ignores the statement completely !!!.

Statement / Rule	Monolithic Policy	Base Policy	Module Policy	Conditional Statements	optional Statement	require Statement
role	Yes	Yes	Yes	No	Yes	Yes
role_transition	Yes	Yes	Yes	No	Yes	No
sensitivity	Yes	Yes	No	No	No	Yes
sid	Yes	Yes	No	No	No	No
type	Yes	Yes	Yes	No	No	Yes
type_change	Yes	Yes	Yes	Yes	Yes	No
type_member	Yes	Yes	Yes	Yes	Yes	No
type_transition	Yes	Yes	Yes	Yes	Yes	No
typealias	Yes	Yes	Yes	No	Yes	No
typeattribute	Yes	Yes	Yes	No	Yes	No
user	Yes	Yes	Yes	No	Yes	Yes
validatetrans	Yes	Yes	No	No	No	No

Table 4-3: The policy language statements and rules that are allowed within each type of policy source file - *The left hand side of the table shows what Policy Language Statements and Rules are allowed within each type of policy source file. The right hand side of the table shows whether the statement is valid within the if / else construct, optional {rule_list}, or require {rule_list} statement.*

4.2.5 SELinux Identifier Naming Conventions

[Table 4-4](#) shows some of the general naming conventions used when developing policy in SELinux, these need not be obeyed but does make reading policy much easier.

[Table 4-2](#) in the [SELinux Policy Language](#) section lists all the reserved words for the SELinux policy language.

type identifier naming conventions	
<code>_t</code>	type identifier (Note that attributes by convention do not end in <code>_t</code> by are normally <code>domain_types</code> , <code>file_types</code> etc.)
<code>bin_t</code>	Ordinary program (a binary file)
<code>sbin_t</code>	System administration program
<code>shbin_t</code>	System shared library
<code>ld_so_t</code>	Dynamic linked library
<code>etc_t</code>	Files in <code>/etc</code> directory
<code>var_log_t</code>	Files in <code>/var/log</code> directory
<code>exec_t</code>	Executable files (allowed the <code>file</code> object <code>execute</code> permission)
<code>wtmp_t</code>	System log
<code>boolean_t</code>	A boolean
Miscellaneous naming conventions	
<code>_r</code>	SELinux role identifier
<code>_u</code>	SELinux user identifier

⁴⁴ Only if preceded by the `optional` statement.

⁴⁵ Only if preceded by the `optional` statement.

File naming conventions	
.conf	General policy source files: policy.conf – monolithic policy base.conf – base policy for loadable policy infrastructure <module_name>.conf – name of the loadable module
.pp	Policy packages (built by semodule(8))
.te	Private policy files in the Reference Policy
.fc	File labeling (context) files in the Reference Policy
.if	Interface files in the Reference Policy

Table 4-4: SELinux identifier naming conventions

4.2.6 Section Contents

The policy language statement and rule sections are as follows:

- a) [Type Enforcement and Attribute Statements](#)
- b) [Type Enforcement Rules](#)
- c) [Access Vector Rules](#)
- d) [User Statement](#)
- e) [Role Statement](#)
- f) [Role Rules](#)
- g) [Conditional Policy Statements](#)
- h) [Constraint Statements](#)
- i) [File System Labeling Statements](#)
- j) [Network Labeling Statements](#)
- k) [MLS Statements](#)
- l) [Policy Support Statements](#)
- m) [Object Class and Permission Statements](#)
- n) [Security ID \(SID\) Statement](#)

4.3 Type Enforcement and Attribute Statements

These statements share the same namespace, therefore the general convention is to use ‘_t’ as the final two characters of a type identifier to differentiate it from an attribute identifier as shown in the following examples:

```
# StatementIdentifier Comment
#-----
type      bin_t;      # A type identifier ends with _t
attribute file_type; # An attribute identifier ends with
                    # anything else
```

4.3.1 type Statement

The type statement declares the type identifier and any optional associated alias or attribute identifiers. Type identifiers are the main component of a [Security Context](#).

The statement definition is:

```
type type_id;
```

Or

```
type type_id ,attribute_id;
```

Or

```
type type_id alias alias_id;
```

Or

```
type type_id alias alias_id ,attribute_id;
```

Where:

type	The type keyword.
type_id	The type identifier.
alias	Optional alias keyword that signifies alternate identifiers for the type_id that are declared in the alias_id list.
alias_id	One or more alias identifiers. Multiple entries consist of a space separated list enclosed in braces ({}).
attribute_id	One or more optional attribute identifiers that have been previously declared by the attribute Statement . Multiple entries consist of a comma (,) separated list, also note the lead comma.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
<hr/>		
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	Yes

Examples:

```
# Using the type statement to declare a type of shell_exec_t,
# where exec_t is used to identify a file as an executable type.

type shell_exec_t;
```

```
# Using the type statement to declare a type of bin_t, where
# bin_t is used to identify a file as an ordinary program type.

type bin_t;
```

```
# Using the type statement to declare a type of bin_t with two
# alias names. The sbin_t is used to identify the file as a
# system admin program type.

type bin_t alias { ls_exec_t sbin_t };
```

```
# Using the type statement to declare a type of boolean_t that
# also associates it to a previously declared attribute
# booleans_type (see the attribute Statement).

attribute booleans_type;      # declare the attribute

type boolean_t, booleans_type; # and associate with the type
```

```
# Using the type statement to declare a type of setfiles_t that
# also has an alias of restorecon_t and one previously declared
# attribute of can_relabelto_binary_policy associated with it.

attribute can_relabelto_binary_policy;

type setfiles_t alias restorecon_t, can_relabelto_binary_policy;
```

```
# Using the type statement to declare a type of
# ssh_server_packet_t that also associates it to two previously
# declared attributes packet_type and server_packet_type.

attribute packet_type;      # declare attribute 1
attribute server_packet_type; # declare attribute 2

# Associate the type identifier with the two attributes:

type ssh_server_packet_t, packet_type, server_packet_type;
```

4.3.2 attribute Statement

An attribute statement declares an identifier that can then be used to refer to a group of types.

The statement definition is:

```
attribute attribute_id;
```

Where:

attribute The attribute keyword.
attribute_id The attribute identifier.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	Yes

Examples:

```
# Using the attribute statement to declare attributes domain,  
# daemon, file_type and non_security_file_type:  
  
attribute domain;  
attribute daemon;  
attribute file_type;  
attribute non_security_file_type;
```

4.3.3 typeattribute Statement

The typeattribute statement allows the association of previously declared types to one or more previously declared attributes.

The statement definition is:

```
typeattribute type_id attribute_id [ ,attribute_id ];
```

Where:

typeattribute The typeattribute keyword.
type_id The identifier of a previously declared type.
attribute_id One or more previously declared attribute identifiers. Multiple entries consist of a comma (,) separated list.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	No

Examples:

```
# Using the typeattribute statement to associate a previously
# declared type of setroubleshootd_t to a previously declared
# domain attribute.

# The previously declared attribute:
attribute domain;

# The previously declared type:
type setroubleshootd_t;

# The association using the typeattribute statement:
typeattribute setroubleshootd_t domain;
```

```
# Using the typeattribute statement to associate a type of
# setroubleshootd_exec_t to two attributes file_type and
# non_security_file_type.

# These are the previously declared attributes:
attribute file_type;
attribute non_security_file_type;

# The previously declared type:
type setroubleshootd_exec_t;

# These are the associations using the typeattribute statement:
typeattribute setroubleshootd_exec_t file_type, non_security_file_type;
```

4.3.4 typealias Statement

The `typealias` statement allows the association of a previously declared type to one or more alias identifiers (an alternative way is to use the [type Statement](#)).

The statement definition is:

```
typealias type_id alias alias_id;
```

Where:

- `typealias` The `typealias` keyword.
- `type_id` The identifier of a previously declared type.
- `alias` The alias keyword.
- `alias_id` One or more alias identifiers. Multiple entries consist of a space separated list enclosed in braces (`{}`).

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	No

Examples:

```
# Using the typealias statement to associate the previously
# declared type mount_t with an alias of mount_ntfs_t.

# Declare the type:
type mount_t;

# Then alias the identifier:
typealias mount_t alias mount_ntfs_t;
```

```
# Using the typealias statement to associate the previously
# declared type netif_t with two alias, lo_netif_t and netif_lo_t.

# Declare the type:
type netif_t;

# Then assign two alias identifiers lo_netif_t and netif_lo_t:
typealias netif_t alias { lo_netif_t netif_lo_t };
```

4.4 Type Enforcement Rules

The TE rules define what access control privileges are allowed for processes. There are three types of enforcement rule: `type_transition`, `type_change`, and `type_member` that are explained below.

The common format of the Type Enforcement Rule is:

```
type_rule source_type target_type : class default_type;
```

Where:

<code>type_rule</code>	The applicable <code>type_transition</code> , <code>type_change</code> , or <code>type_member</code> rule keyword.
<code>source_type</code>	One or more source / target type or attribute identifiers. Multiple entries consist of a space separated list enclosed in braces ({}).
<code>target_type</code>	One or more target type or attribute identifiers. Multiple entries consist of a space separated list enclosed in braces ({}). Entries can be excluded from the list by using the negative operator (-).
<code>class</code>	One or more object classes. Multiple entries consist of a space separated list enclosed in braces ({}).
<code>default_type</code>	A single type identifier that will become the default process type for a domain transition or the

`type` for object transitions.

The statements are valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
Yes	Yes	No

4.4.1 `type_transition` Statement

The `type_transition` statement specifies the labeling and object creation allowed between the `source_type` and `target_type` when a [Domain Transition](#) is requested.

Example – Domain Transition:

```
# Using the type_transition statement to show a domain transition
# (as the statement has the process object class in the
# class).

# The rule states that when a process of type initrc_t executes
# a file of type acct_exec_t, the process type should be changed
# to acct_t if allowed by the policy (i.e. Transition from the
# initrc_t domain to the acct_t domain).

type_transition initrc_t acct_exec_t:process acct_t;

# Note that to be able to transition to the acct_t domain the
# following minimum permissions need to be granted in the policy
# using allow rules (as shown in the allow Rule section).

# File needs to be executable in the initrc_t domain:
allow initrc_t acct_exec_t:file execute;

# The executable file needs an entry point into the acct_t domain:
allow acct_t acct_exec_t:file entrypoint;

# Process needs permission to transition into the acct_t domain:
allow initrc_t acct_t:process transition;
```

Example – Object Transition:

```
# Using the type_transition statement to show an object
# transition (as it has other than process in the class).

# The rule states that when a process of type acct_t creates a
# file in the directory of type var_log_t, by default it should
# have the type wtmp_t if allowed by the policy.

type_transition acct_t var_log_t:file wtmp_t;
```

```
# Note that to be able to create the new file object with the
# wtmp_t type, the following minimum permissions need to be
# granted in the policy using allow rules (as shown in the
# allow Rule section).

# A minimum of: add_name, write and search on the var_log_t
# directory. The actual F-10 policy has:
#
allow acct_t var_log_t:dir { read getattr lock search ioctl
    add_name remove_name write };

# A minimum of: create and write on the wtmp_t file. The actual
# F-10 policy has:
#
allow acct_t wtmp_t:file { create open getattr setattr read
    write append rename link unlink ioctl lock };
```

4.4.2 `type_change` Statement

The `type_change` statement is used to determine any re-labeling of default types for user space SELinux-aware applications that would then manage any required re-labeling via the `libselinux` API.

Examples:

```
# Using the type_change statement to show that when relabeling a
# character file with type sysadm_devpts_t on behalf of
# auditadm_t, the type auditadm_devpts_t should be used:

type_change auditadm_t sysadm_devpts_t:chr_file auditadm_devpts_t;
```

```
# Using the type_change statement to show that when relabeling a
# character file with any type associated to the attribute
# server_ptynode on behalf of staff_t, the type staff_devpts_t
# should be used:

type_change staff_t server_ptynode:chr_file staff_devpts_t;
```

4.4.3 `type_member` Statement

The `type_member` statement determines whether an object can be polyinstantiated. It is used by SELinux-aware applications that would then manage any required polyinstantiation requirements via the `libselinux` API (see the [Polyinstantiation](#) section). Currently only directories are managed by SELinux-aware applications, although the actual statement is not limited to specific object classes.

Example:

```
# Using the type_member statement to show that if the source
# type is sysadm_t, and the target type is user_home_dir_t,
# then use user_home_dir_t as the type on the newly created
# directory object.

type_member sysadm_t user_home_dir_t:dir user_home_dir_t;
```

4.5 Access Vector Rules

The AV rules define what access control privileges are allowed for processes. There are four types of AV rule: `allow`, `dontaudit`, `auditallow`, and `neverallow` as explained in the sections that follow with a number of examples to cover all the scenarios. There is also an `auditdeny` rule, however it is no longer used in the [Reference Policy](#) and has been replaced by the `dontaudit` rule.

The general format of an AV rule is that the `source_type` is the identifier of a process that is attempting to access an object identifier `target_type`, that has an object class of `class`, and `perm_set` defines the access permissions `source_type` is allowed.

The common format of the Access Vector Rule is:

```
rule_name source_type target_type : class perm_set;
```

Where:

<code>rule_name</code>	The applicable <code>allow</code> , <code>dontaudit</code> , <code>auditallow</code> , and <code>neverallow</code> rule keyword.
<code>source_type</code>	One or more source / target type or attribute identifiers. Multiple entries consist of a space separated list enclosed in braces ({}). Entries can be excluded from the list by using the negative operator (-). The <code>target_type</code> can have the <code>self</code> keyword instead of type or attribute identifiers. This means that the <code>target_type</code> is the same as the <code>source_type</code> . The <code>neverallow</code> rule also supports the wildcard operator (*) to specify that all types are to be included and the complement operator (~) to specify all types are to be included except those explicitly listed.
<code>target_type</code>	
<code>class</code>	One or more object classes. Multiple entries consist of a space separated list enclosed in braces ({}).
<code>perm_set</code>	The access permissions the source is allowed to access for the target object (also known as the Access Vector). Multiple entries consist of a space separated list enclosed in braces ({}). The optional wildcard operator (*) specifies that all permissions for the object class can be used. The complement operator (~) is used to specify all permissions except those explicitly listed (although the compiler issues a warning if the <code>dontaudit</code> rule has '~').

The statements are valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
allow = Yes auditallow = Yes dontaudit = Yes neverallow = No	allow = Yes auditallow = Yes dontaudit = Yes neverallow = Yes	allow = No auditallow = No dontaudit = No neverallow = No

4.5.1 allow Rule

The allow rule checks whether the operations between the `source_type` and `target_type` are allowed. It is the most common statement that many of the [Reference Policy](#) helper macros and interface definitions expand into multiple allow rules.

Examples:

```
# Using the allow rule to show that initrc_t is allowed access
# to files of type acct_exec_t that have the getattr, read and
# execute file permissions:
```

```
allow initrc_t acct_exec_t:file { getattr read execute };
```

```
# This rule includes an attribute filesystem_type and states
# that kernel_t is allowed mount permissions on the filesystem
# object for all types associated to the filesystem_type
# attribute:
```

```
allow kernel_t filesystem_type:filesystem mount;
```

```
# This rule includes the self keyword in the target_type that
# states that staff_t is allowed setgid, chown and fowner
# permissions on the capability object:
```

```
allow staff_t self:capability { setgid chown fowner };
```

```
# This would be the same as the above:
```

```
allow staff_t staff_t:capability { setgid chown fowner };
```

```
# This rule includes the wildcard operator (*) on the perm_set
# and states that bootloader_t is allowed to use all permissions
# available on the dbus object that are type system_dbusd_t:
```

```
allow bootloader_t system_dbusd_t:dbus *;
```

```
# This would be the same as the above:
```

```
allow bootloader_t system_dbusd_t:dbus { acquire_svc send_msg };
```

```
# This rule includes the complement operator (~) on the perm_set
```



```
# and two class entries file and chr_file.
#
# The allow rule states that all types associated with the
# attribute files_unconfined_type are allowed to use all
# permissions available on the file and chr_file objects except
# the execmod permission when they are associated to the types
# listed within the attribute file_type:

allow files_unconfined_type file_type:{ file chr_file } ~execmod;
```

4.5.2 dontaudit Rule

The `dontaudit` rule stops the auditing of denial messages as it is known that this event always happens and does not cause any real issues. This also helps to manage the audit log by excluding known events.

Example:

```
# Using the dontaudit rule to stop auditing events that are
# known to happen. The rule states that when the traceroute_t
# process is denied access to the name_bind permission on a
# tcp_socket for all types associated to the port_type
# attribute (except port_t), then do not audit the event:

dontaudit traceroute_t { port_type -port_t }:tcp_socket name_bind;
```

4.5.3 auditallow Rule

Audit the event as a record as it is useful for auditing purposes. Note that this rule only audits the event, it still requires the `allow` rule to grant permission.

Example:

```
# Using the auditallow rule to force an audit event to be
# logged. The rule states that when the ada_t process has
# permission to execstack, then that event must be audited:

auditallow ada_t self:process execstack;
```

4.5.4 neverallow Rule

This rule specifies that an [allow Rule](#) must not be generated for the operation, even if it has been previously allowed. The `neverallow` statement is a compiler enforced action, where the `checkpolicy` or `checkmodule` compiler checks if any `allow` rules have been generated in the policy source, if so it will issue a warning and stop.

Examples:

```
# Using the neverallow rule to state that no allow rule may ever
# grant any file read access to type shadow_t except those
# associated with the can_read_shadow_passwords attribute:

neverallow ~can_read_shadow_passwords shadow_t:file read;
```

```
# Using the neverallow rule to state that no allow rule may ever
# grant mmap_zero permissions any type associated to the domain
# attribute except those associated to the mmap_low_domain_type
# attribute (as these have been excluded by the negative
# operator (-)):

neverallow { domain -mmap_low_domain_type } self:memprotect
mmap_zero;
```

4.6 User Statement

4.6.1 user Statement

The `user` statement is used to declare an SELinux user identifier within the policy and associate that to one or more roles. The statement also allows an optional MLS `level` and `range` to control a users security level. It is also possible to add SELinux user id's outside the policy using the 'semanage user' command that will associate the user with roles previously declared within the policy.

The statement definition is:

```
user seuser_id roles role_id;
```

Or for MCS/MLS Policy:

```
user seuser_id roles role_id level mls_level range mls_range;
```

Where:

<code>user</code>	The user keyword.
<code>seuser_id</code>	The SELinux user identifier.
<code>roles</code>	The roles keyword.
<code>role_id</code>	One or more previously declared <code>role</code> identifiers. Multiple role identifiers consist of a space separated list enclosed in braces (<code>{ }</code>).
<code>level</code>	If MLS is configured, the MLS <code>level</code> keyword.
<code>mls_level</code>	The users default MLS security <code>level</code> that has been previously declared with a level Statement . Note that the compiler only accepts the sensitivity component of the <code>level</code> (e.g. <code>s0</code>).
<code>range</code>	If MLS is configured, the MLS <code>range</code> keyword.
<code>mls_range</code>	The range of security levels that the user can run. The format is described in the MLS range Definition section.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
<hr/>		
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	Yes

Example:

```
# Using the user statement to define an SELinux user user_u that
# has been assigned the role of user_r. The SELinux user_u is a
# generic user identity for Linux users who have no specific
# SELinux user identity defined.
#
user user_u roles { user_r };
```

MLS Examples:

```
# Using the user statement to define an MLS SELinux user user_u
# that has been assigned the role of user_r and has a default
# login security level of s0 assigned, and is only allowed access
# to the s0 range of security levels (See the MLS Statements
# section for details):
user user_u roles { user_r } level s0 range s0;
```

```
# Using the user statement to define an MLS SELinux user
# sysadm_u that has been assigned the role of sysadm_r and has
# a default login security level of s0 assigned, and is
# allowed access to the range of security levels (low - high)
# between s0 and s15:c0.c255 (See the MLS Statements section
# for details):
user sysadm_u roles { sysadm_r } level s0 range s0 - s15:c0.c255;
```

semanage (8) Command example:

```
# Add user mque_u to SELinux and associate to the unconfined_r
# role:
semanage user -a -R unconfined_r mque_u
```

This command will produce the following files in the default <policy_name> policy store and then activate the policy:

/etc/selinux/<policy_name>/modules/active/users.local:

```
# This file is auto-generated by libsemanage
# Do not edit directly.

user mque_u roles { unconfined_r } ;
```

/etc/selinux/<policy_name>/modules/active/users_extra:

```
# This file is auto-generated by libsemanage
# Do not edit directly.

user mque_u prefix user;
```

/etc/selinux/<policy_name>/modules/active/users_extra.local:

```
# This file is auto-generated by libsemanage
# Do not edit directly.

user mque_u prefix user;
```

4.7 Role Statement

4.7.1 role Statement

The `role` statement associates a role identifier to one or more `types` (i.e. authorise the role to access the domain or domains). Where there are multiple `role` statements declaring the same role, the compiler will associate the additional `types` with the `role`.

The statement definition is:

```
role role_id;
```

Or

```
role role_id types type_id;
```

Where:

<code>role</code>	The <code>role</code> keyword.
<code>role_id</code>	The identifier of the role being declared. The same role identifier can be declared more than once in a policy, in which case the <code>type_id</code> entries will be amalgamated by the compiler.
<code>types</code>	The optional <code>types</code> keyword.
<code>type_id</code>	When used with the <code>types</code> keyword, one or more <code>type</code> or <code>attribute</code> identifiers associated with the <code>role_id</code> . Multiple entries consist of a space separated list enclosed in braces (<code>{ }</code>). Entries can be excluded from the list by using the negative operator (<code>-</code>). For <code>role</code> statements, only <code>type</code> or <code>attribute</code> identifiers associated to domains have any meaning within SELinux.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	Yes

Examples:

```
# Using the role statement to define standard roles in the
# Reference Policy. Note that there are no domains associated
# with them yet.

role system_r;
role sysadm_r;
role staff_r;
role user_r;
role secadm_r;
role auditadm_r;

# Within the policy the roles are then associated to the
# required domains with this example showing the user_r role
# being associated to two domains:

role user_r types user_t;
role user_r types chfn_t;
```

4.8 Role Rules

4.8.1 Role allow Rule

The role allow rule checks whether a request to change roles is allowed, if it is, then there may be a further request for a role_transition so that the process runs with the new role or role set.

Important Notes:

1. The role allow rule has the same keyword as the allow AV rule.
2. The role allow rule is used in the [Reference Policy](#) sources, however there are no corresponding role_transition rules. This is because the policy expects users to either keep the same role as when they logged onto the system, or use the newrole(1) command to change roles.
3. The [Reference Policy](#) uses the [constrain Statement](#) to manage role relationships.

The statement definition is:

```
allow from_role_id to_role_id;
```

Where:

allow The role allow rule keyword.
from_role_id One or more role identifiers that identify the

current role. Multiple entries consist of a space separated list enclosed in braces ({ }).

`to_role_id` One or more `role` identifiers that identify the new role to be granted on the transition. Multiple entries consist of a space separated list enclosed in braces ({ }).

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	No

Example:

```
# Using the role allow rule to define authorised role
# transitions in the Reference Policy. The current role
# sysadm_r is granted permission to transition to the secadm_r
# role in the MLS policy.

allow sysadm_r secadm_r;
```

4.8.2 role_transition Rule

The `role_transition` rule specifies that a role transition is required, and if allowed, the process will run under the new role.

Note that the `role_transition` rule is not used by the [Reference Policy](#) as the policy only allows roles to change at login or by executing the `newrole` command.

The [SECMARK Loadable Module](#) however does use a `role_transition` that is described as the example.

The statement definition is:

```
role_transition current_role_id type_id new_role_id;
```

Where:

`role_transition` The `role_transition` keyword.

`current_role_id` One or more `role` identifiers that identify the current role. Multiple entries consist of a space separated list enclosed in braces ({ }).

`type_id` One or more `type` or `attribute` identifiers. Multiple entries consist of a space separated list enclosed in braces ({ }). Entries can be excluded from the list by using the negative operator (-). Only 'domain' types make sense within the policy.

`new_role_id` The new role to be granted on transition.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	No

Example:

```
# This is a role_transition used in the ext_gateway.conf
# loadable module to allow the secure client / server process to
# run under the message_filter_r role. The role needs to be
# declared, allowed to transition from its current role of
# unconfined_r and it then transitions when the process
# transitions via the type_transition statement (not shown).
# Note that the role needs to be associated to a user by either:
# 1) An embedded user statement in the policy. This is not
#    recommended as it makes the policy fixed to either
#    standard, MCS or MLS.
# 2) Using the semanage(8) command to add the role. This will
#    allow the module to be used by MCS/MLS policies as well.
#
# The secure client / server will run in this domain:
type ext_gateway_t;
# The binaries will be labeled:
type secure_services_exec_t;
# Use message_filter_r role and then transition
role message_filter_r types ext_gateway_t;
allow unconfined_r message_filter_r;
role_transition unconfined_r secure_services_exec_t message_filter_r;
```

4.8.3 Role dominance Rule

This rule has been deprecated and therefore should not be used. The role dominance rule allows the `dom_role_id` to dominate the `role_id` (consisting of one or more roles). The dominant role will automatically inherit all the type associations of the other roles.

Notes:

1. There is another dominance rule for MLS (see the [MLS dominance Statement](#)).
2. The role dominance rule is not used by the [Reference Policy](#) as the policy manages role dominance using the [constrain Statement](#).
3. Note the usage of braces ‘{ }’ and the ‘;’ in the statement.

The statement definition is:

```
dominance { role dom_role_id { role role_id; } }
```

Where:

dominance The dominance keyword.
 role The role keyword.
 dom_role_id The dominant role identifier.
 role_id For the simple case each { role role_id; } pair defines the role_id that will be dominated by the dom_role_id. More complex rules can be defined but as the statement is depreciated !!!.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	No

Example:

```
# This shows the dominance role rule that is used in the
# constraint example shown in:
# Appendix G - Implementing a Constraint. Note however that it
# has been depreciated and should not be used.

dominance { role message_filter_r { role unconfined_r };}
```

4.9 Conditional Policy Statements

Conditional policies consist of a bool statement that defines a condition as true or false, with a supporting if / else construct that specifies what rules are valid under the condition as shown in the example below:

```
bool allow_daemons_use_tty true;

if (allow_daemons_use_tty) {
    # Rules if condition is true;
} else {
    # Rules if condition is false;
}
```

[Table 4-3](#) shows what policy statements or rules are valid within the if / else construct under the “Conditional Statements” column.

The bool statement default value can be changed when a policy is active by using the setsebool command as follows:


```
# This command will set the allow_daemons_use_tty bool to false,
# however it will only remain false until the next system
# re-boot where it will then revert back to its default state
# (in the above case, this would be true).

setsebool allow_daemons_use_tty false
```

```
# This command will set the allow_daemons_use_tty bool to false,
# and because the -P option is used (for persistent), the value
# will remain across system re-boots. Note however that all
# other pending bool values will become persistent across
# re-boots as well (see the setsebool (8) man page).

setsebool -P allow_daemons_use_tty false
```

The `getsebool` command can be used to query the current `bool` statement value as follows:

```
# This command will list all bool values in the active policy:

getsebool -a
```

```
# This command will show the current allow_daemons_use_tty bool
# value in the active policy:

getsebool allow_daemons_use_tty
```

4.9.1 `bool` Statement

The `bool` statement is used to specify a boolean identifier and its initial state (`true` or `false`) that can then be used with the [if Statement](#) to form a ‘conditional policy’ as described in the [Conditional Policy](#) section.

The statement definition is:

```
bool bool_id default_value;
```

Where:

<code>bool</code>	The <code>bool</code> keyword.
<code>bool_id</code>	The boolean identifier.
<code>default_value</code>	Either <code>true</code> or <code>false</code> .

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	Yes

Examples:

```
# Using the bool statement to allow unconfined executables to
# make their memory heap executable or not. As the value is
# false, then by default they cannot make their heap executable.

bool allow_execheap false;
```

```
# Using the bool statement to allow unconfined executables to
# make their stack executable or not. As the value is true,
# then by default their stacks are executable.

bool allow_execstack true;
```

4.9.2 if Statement

The `if` statement is used to form a ‘conditional block’ of statements and rules that are enforced depending on whether one or more boolean identifiers (defined by the [bool Statement](#)) evaluate to TRUE or FALSE. An `if / else` construct is also supported.

The only statements and rules allowed within the `if / else` construct are:

`allow, auditallow, auditdeny, dontaudit, type_member, type_transition, type_change` and `require`.

The statement definition is:

```
if (conditional_expression) { true_list } [ else { false_list } ]
```

Where:

- `if` The `if` keyword.
- `conditional_expression` One or more `bool_name` identifiers that have been previously defined by the [bool Statement](#). Multiple identifiers must be separated by the following logical operators: `&&, ||, ^, !, ==, !=`.
The `conditional_expression` is enclosed in brackets `()`.
- `true_list` A list of rules enclosed within braces `{ }` that will be executed when the `conditional_expression` is ‘true’.
Valid statements and rules are highlighted

else
false_list

within each language definition statement.
Optional else keyword.
A list of rules enclosed within braces ‘{ }’ that will be executed when the optional ‘else’ keyword is present and the conditional_expression is ‘false’.
Valid statements and rules are highlighted within each language definition statement.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No – As this is a Conditional Statement and cannot be nested.	Yes	No

Examples:

```
# An example showing a boolean and supporting if statement.
bool allow_execmem false;

# The bool allow_execmem is FALSE therefore the allow statement
# is not executed:
if (allow_execmem) {
    allow sysadm_t self:process execmem;
}
```

```
# An example showing two booleans and a supporting if statement.
bool allow_execmem false;
bool allow_execstack true;

# The bool allow_execmem is FALSE and allow_execstack is TRUE
# therefore the allow statement is not executed:
if (allow_execmem && allow_execstack) {
    allow sysadm_t self:process execstack;
}
```

```
# An example of an IF - ELSE statement where the bool statement
# is FALSE, therefore the ELSE statements will be executed.
#
bool read_untrusted_content false;

if (read_untrusted_content) {
    allow sysadm_t { sysadm_untrusted_content_t
        sysadm_untrusted_content_tmp_t }:dir { getattr search read
```

```
        lock ioctl };
        .....
} else {
    dontaudit sysadm_t { sysadm_untrusted_content_t
        sysadm_untrusted_content_tmp_t }:dir { getattr search read
        lock ioctl };
    ...
}
```

4.10 Constraint Statements

4.10.1 constrain Statement

The `constrain` statement allows further restriction on permissions for the specified object classes by using boolean expressions covering: source and target types, roles and users as described in the examples.

The statement definition is:

```
constrain class perm_set expression;
```

Where:

<code>constrain</code>	The <code>constrain</code> keyword.
<code>class</code>	One or more object classes. Multiple entries consist of a space separated list enclosed in braces (<code>{ }</code>).
<code>perm_set</code>	One or more permissions. Multiple entries consist of a space separated list enclosed in braces (<code>{ }</code>).
<code>expression</code>	The boolean expression of the constraint that is defined as follows:

```
( expression : expression )
| not expression
| expression and expression
| expression or expression
| u1 op u2
| r1 role_op r2
| t1 op t2
| u1 op names
| u2 op names
| r1 op names
| r2 op names
| t1 op names
| t2 op names
```

Where:

`u1, r1, t1` = Source user, role, type

`u2, r2, t2` = Target user, role, type

and:

`op` : `==` | `!=`

```
role_op : == | != | eq | dom | domby | incomp
names : name | { name_list }
name_list : name | name_list name
```

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Examples:

These examples have been taken from the [Reference Policy](#) source `./policy/constraints` file.

```
# This constrain statement is the "SELinux process identity
# change constraint" taken from the Reference Policy source and
# contains multiple expressions.
#
# The overall constraint is on the process object class with the
# transition permission, and is stating that a domain transition
# is being constrained by the rules listed (u1 == u2 etc.),
# however only the first two expressions are explained.
#
# The first expression u1 == u2 states that the source (u1) and
# target (u2) user identifiers must be equal for a process
# transition to be allowed.
#
# However note that there are a number of or operators that can
# override this first constraint.
#
# The second expression:
# ( t1 == can_change_process_identity and t2 == process_user_target )
#
# states that if the source type (t1) is equal to any type
# associated to the can_change_process_identity attribute, and
# the target type (t2) is equal to any type associated to the
# process_user_target attribute, then a process transition is
# allowed.
#
# What this expression means in the 'standard' build Reference
# Policy is that if the source domain is either cron_t,
# firstboot_t, local_login_t, su_login_t, sshd_t or xdm_t (as
# the can_change_process_identity attribute has these types
# associated to it) and the target domain is sysadm_t (as that is
# the only type associated to the can_change_process_identity
# attribute), then a domain transition is allowed.
#
# SELinux process identity change constraint:
constrain process transition (
    u1 == u2
or
```

```
( t1 == can_change_process_identity and t2 == process_user_target )
or
( t1 == cron_source_domain and ( t2 == cron_job_domain or u2 == system_u ) )
or
( t1 == can_system_change and u2 == system_u )
or
( t1 == process_uncond_exempt ) );
```

```
# This constrain statement is the "SELinux file related object
# identity change constraint" taken from the Reference Policy
# source and contains two expressions.
#
# The overall constraint is on the listed file related object
# classes (dir, file etc.), covering the create, relabelto, and
# relabelfrom permissions. It is stating that when any of the
# object class listed are being created or relabeled, then they
# are subject to the constraint rules listed (u1 == u2 etc.).
#
# The first expression u1 == u2 states that the source (u1) and
# target (u2) user identifiers (within the security context)
# must be equal when creating or relabeling any of the file
# related objects listed.
#
# The second expression:
# or t1 == can_change_object_identity
#
# states or if the source type (t1) is equal to any type
# associated to the can_change_object_identity attribute, then
# any of the object class listed can be created or relabeled.
#
# What this expression means in the 'standard' build
# Reference Policy is that if the source domain (t1) matches a
# type entry in the can_change_object_identity attribute, then
# any of the object class listed can be created or relabeled.
#
# SELinux file related object identity change constraint:
constrain { dir file lnk_file sock_file fifo_file chr_file
            blk_file } { create relabelto relabelfrom }
(
  u1 == u2
  or t1 == can_change_object_identity
);
```

4.10.2 **validatetrans Statement**

Only file related object classes are currently supported by this statement and it is used to control the ability to change the objects security context.

Note there are no *validatetrans* statements specified within the F-10 [Reference Policy](#) source.

The statement definition is:

```
validatetrans class expression;
```

Where:

validatetrans	The validatetrans keyword.
class	One or more file related object classes. Multiple entries consist of a space separated list enclosed in braces ({}).
expression	The boolean expression of the constraint that is defined as follows: <pre style="margin-left: 40px;">(expression : expression) not expression expression and expression expression or expression u1 op u2 r1 role_op r2 t1 op t2 u1 op names u2 op names r1 op names r2 op names t1 op names t2 op names u3 op names r3 op names t3 op names</pre>

Where:

u1, r1, t1 = Old user, role, type
u2, r2, t2 = New user, role, type
u3, r3, t3 = Process user, role, type

and:

op : == | !=
role_op : == | != | eq | dom | domby | incomp
names : name | { name_list }
name_list : name | name_list name

The statement is valid in:

<u>Monolithic Policy</u>	<u>Base Policy</u>	<u>Module Policy</u>
Yes	Yes	No
<u>Conditional Policy (if) Statement</u>	<u>optional Statement</u>	<u>require Statement</u>
No	No	No

Examples:

4.11 File System Labeling Statements

There are four types of file labeling statements: `fs_use_xattr`, `fs_use_task`, `fs_use_trans` and `genfscon` that are explained below.

The filesystem identifiers (`fs_name`) used by these statements are defined by the SELinux teams who are responsible for their development, the policy writer then uses those needed to be supported by the policy.

A security context is defined by these filesystem labeling statements, therefore if the policy supports MCS / MLS, then an `mls_range` is required as described in the [MLS range Definition](#) section.

4.11.1 fs_use_xattr Statements

The `fs_use_xattr` statement is used to allocate a security context to filesystems that support the extended attribute `security.selinux`. The labeling is persistent for filesystems that support these extended attributes, and the security context is added to these files (and directories) by the SELinux commands such as `setfiles` as explained in the [Labeling Extended Attribute Filesystems](#) section.

The statement definition is:

```
fs_use_xattr fs_name fs_context;
```

Where:

<code>fs_use_xattr</code>	The <code>fs_use_xattr</code> keyword.
<code>fs_name</code>	The filesystem name that supports extended attributes. The known valid names are: <code>encfs</code> , <code>ext2</code> , <code>ext3</code> , <code>ext4</code> , <code>ext4dev</code> , <code>gfs</code> , <code>gfs2</code> , <code>jffs2</code> , <code>jfs</code> , <code>lustre</code> and <code>xfs</code> .
<code>fs_context</code>	The security context allocated to the filesystem.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Example:

```
# These statements define file systems that support extended
# attributes (security.selinux).

fs_use_xattr encfs system_u:object_r:fs_t;
fs_use_xattr ext2 system_u:object_r:fs_t;
fs_use_xattr ext3 system_u:object_r:fs_t;
```


MLS Examples:

```
# These statements define file systems that support extended
# attributes (security.selinux).

fs_use_xattr encfs system_u:object_r:fs_t:s0;
fs_use_xattr ext2 system_u:object_r:fs_t:s0;
fs_use_xattr ext3 system_u:object_r:fs_t:s0;
```

4.11.2 fs_use_task Statement

The `fs_use_task` statement is used to allocate a security context to pseudo filesystems that support task related services such as pipes and sockets.

The statement definition is:

```
fs_use_task fs_name fs_context;
```

Where:

<code>fs_use_task</code>	The <code>fs_use_task</code> keyword.
<code>fs_name</code>	Filesystem name that supports task related services. The known valid names are: <code>eventpollfs</code> , <code>pipefs</code> and <code>sockfs</code> .
<code>fs_context</code>	The security context allocated to the task based filesystem.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Example:

```
# These statements define the file systems that support pseudo
# filesystems that represent objects like pipes and sockets, so
# that these objects are labeled with the same type as the
# creating task.
#
fs_use_task eventpollfs system_u:object_r:fs_t;
fs_use_task pipefs system_u:object_r:fs_t;
fs_use_task sockfs system_u:object_r:fs_t;
```

MLS Example:

```
# These statements define the file systems that support pseudo
# filesystems that represent objects like pipes and sockets, so
# that these objects are labeled with the same type as the
# creating task.
```

```
#
fs_use_task eventpollfs system_u:object_r:fs_t:s0;
fs_use_task pipefs system_u:object_r:fs_t:s0;
fs_use_task sockfs system_u:object_r:fs_t:s0;
```

4.11.3 fs_use_trans Statement

The `fs_use_trans` statement is used to allocate a security context to pseudo filesystems such as pseudo terminals and temporary objects. The assigned context is derived from the creating process and that of the filesystem type based on transition rules.

The statement definition is:

```
fs_use_trans fs_name fs_context;
```

Where:

<code>fs_use_trans</code>	The <code>fs_use_trans</code> keyword.
<code>fs_name</code>	Filesystem name that supports transition rules. The known valid names are: <code>mqueue</code> , <code>shm</code> , <code>tmpfs</code> and <code>devpts</code> .
<code>fs_context</code>	The security context allocated to the transition based on that of the filesystem.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Example:

```
# These statements define pseudo filesystems such as devpts
# and tmpfs where objects are labeled with a derived context.
#
fs_use_trans mqueue system_u:object_r:tmpfs_t;
fs_use_trans shm system_u:object_r:tmpfs_t;
fs_use_trans tmpfs system_u:object_r:tmpfs_t;
fs_use_trans devpts system_u:object_r:devpts_t;
```

MLS Example:

```
# These statements define pseudo filesystems such as devpts
# and tmpfs where objects are labeled with a derived context.
#
fs_use_trans mqueue system_u:object_r:tmpfs_t:s0;
fs_use_trans shm system_u:object_r:tmpfs_t:s0;
fs_use_trans tmpfs system_u:object_r:tmpfs_t:s0;
```

```
fs_use_trans devpts system_u:object_r:devpts_t:s0;
```

4.11.4 genfscon Statements

The `genfscon` statement is used to allocate a security context to filesystems that cannot support any of the other file labeling statements (`fs_use_xattr`, `fs_use_task` or `fs_use_trans`). Generally a filesystem would have a single default security context assigned by `genfscon` from the root (`/`) that would then be inherited by all files and directories on that filesystem. The exception to this is the `/proc` filesystem, where directories can be labeled with a specific security context (as shown in the examples). Note that there is no terminating semi-colon (`;`) on this statement.

The statement definition is:

```
genfscon fs_name partial_path fs_context
```

Where:

<code>genfscon</code>	The <code>genfscon</code> keyword.
<code>fs_name</code>	The filesystem name.
<code>partial_path</code>	If <code>fs_name</code> is <code>proc</code> , then the partial path (see the examples). For all other types, this must be <code>'/'</code> .
<code>fs_context</code>	The security context allocated to the filesystem

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Examples:

```
# The following examples show those filesystems that only
# support a single security context across the filesystem.

genfscon msdos / system_u:object_r:dosfs_t
genfscon iso9660 / system_u:object_r:iso9660_t
genfscon usbfs / system_u:object_r:usbfs_t
genfscon selinuxfs / system_u:object_r:security_t

# The following show some example /proc entries that can have
# directories added to the path.

genfscon proc / system_u:object_r:proc_t
genfscon proc /sysvipc system_u:object_r:proc_t
genfscon proc /fs/openafs system_u:object_r:proc_afs_t
genfscon proc /kmsg system_u:object_r:proc_kmsg_t
```

MLS Examples:

```
# The following examples show those filesystems that only
# support a single security context across the filesystem
# with the MLS levels added.

genfscon msdos / system_u:object_r:dosfs_t:s0
genfscon iso9660 / system_u:object_r:iso9660_t:s0
genfscon usbfs / system_u:object_r:usbfs_t:s0
genfscon selinuxfs / system_u:object_r:security_t:s0

# The following show some example /proc entries. Note that the
# /kmsg has the highest sensitivity level assigned (s15) because
# it is a trusted process.

genfscon proc / system_u:object_r:proc_t:s0
genfscon proc /sysvipc system_u:object_r:proc_t:s0
genfscon proc /fs/openafs system_u:object_r:proc_afs_t:s0
genfscon proc /kmsg system_u:object_r:proc_kmsg_t:s15:c0.c255
```

4.12 Network Labeling Statements

The network labeling statements are used to label the following objects:

Network interfaces – This covers those interfaces managed by the `ifconfig(8)` command.

Network nodes – These are generally used to specify host systems using either IPv4 or IPv6 addresses.

Network ports – These can be either `udp` or `tcp` port numbers.

A security context is defined by these network labeling statements, therefore if the policy supports MCS / MLS, then an `mls_range` is required as described in the [MLS range Definition](#) section. Note that there are no terminating semi-colons (;) on these statements.

If any of the network objects do not have a specific security context assigned by the policy, then the value given in the policies initial SID is used (`netif`, `node` or `port` respectively), as shown below:

```
# Network Initial SIDs from the Standard F-10 Reference Policy:
sid netif system_u:object_r:netif_t
sid node system_u:object_r:node_t
sid port system_u:object_r:port_t

# Network Initial SIDs from the MLS F-10 Reference Policy:
sid netif system_u:object_r:netif_t:s0 - s15:c0.c255
sid node system_u:object_r:node_t:s0 - s15:c0.c255
sid port system_u:object_r:port_t:s0
```

4.12.1 IP Address Formats

4.12.1.1 IPv4 Address Format

IPv4 addresses are represented in dotted-decimal notation (four numbers, each ranging from 0 to 255, separated by dots as shown:

```
192.77.188.166
```

4.12.1.2 IPv6 Address Formats

IPv6 addresses are written as eight groups of four hexadecimal digits, where each group is separated by a colon (:) as follows:

```
2001:0db8:85a3:0000:0000:8a2e:0370:7334
```

To shorten the writing and presentation of addresses, the following rules apply:

- a) Any leading zeros in a group may be replaced with a single '0' as shown:

```
2001:db8:85a3:0:0:8a2e:370:7334
```

- b) Any leading zeros in a group may be omitted and be replaced with two colons (::), however this is only allowed once in an address as follows:

```
2001:db8:85a3::8a2e:370:7334
```

- c) The localhost (loopback) address can be written as:

```
0000:0000:0000:0000:0000:0000:0000:0001
```

Or

```
::1
```

- d) An undetermined IPv6 address i.e. all bits are zero is written as:

```
::
```

4.12.2 netifcon Statement

The `netifcon` statement is used to label network interface objects (e.g. `eth0`).

It is also possible to add SELinux user id's outside the policy using the `'semanage interface'` command that will associate the interface to a security context.

The statement definition is:

```
netifcon netif_id netif_context packet_context
```

Where:

<code>netifcon</code>	The <code>netifcon</code> keyword.
<code>netif_id</code>	The network interface name (e.g. <code>eth0</code>).
<code>netif_context</code>	The security context allocated to the network interface.
<code>packet_context</code>	The security context allocated packets. Note that these are defined but currently unused.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Examples:

```
# The following netifcon statement has been taken from the F-10
# MLS policy that shows an interface name of lo with the same
# security context assigned to both the interface and packets.

netifcon lo system_u:object_r:lo_netif_t:s0 - s15:c0.c255
        system_u:object_r:unlabeled_t:s0 - s15:c0.c255
```

semanage (8) Command example:

```
semanage interface -a -t unconfined_t eth0
```

This command will produce the following file in the default `<policy_name>` policy store and then activate the policy:

```
/etc/selinux/<policy_name>/modules/active/interfaces.local:
```

```
# This file is auto-generated by libsemanage
# Do not edit directly.

netifcon eth0 system_u:object_r:unconfined_t system_u:object_r:unconfined_t
```

4.12.3 nodecon Statement

The `nodecon` statement is used to label network address objects that represent IPv4 or IPv6 IP addresses and network masks.

It is also possible to add SELinux these outside the policy using the ‘`semanage node`’ command that will associate the node to a security context.

The statement definition is:

```
nodecon subnet netmask node_context
```

Where:

nodecon	The <code>nodecon</code> keyword.
subnet	The subnet or specific IP address in IPv4 or IPv6 format. Note that the <code>subnet</code> and <code>netmask</code> values are used to ensure that the <code>node_context</code> is assigned to all IP addresses within the subnet range.
netmask	The subnet mask in IPv4 or IPv6 format.
node_context	The security context for the node.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Examples:

```
# The Standard Reference Policy nodecon statement for the IPv4
# Local Host:
nodecon 127.0.0.1 255.255.255.255 system_u:object_r:lo_node_t

# The equivalent MLS Reference Policy nodecon statement for the
# IPv4 Local Host:
nodecon 127.0.0.1 255.255.255.255 system_u:object_r:lo_node_t:
s0 - s15:c0.c255
```

```
# The Standard Reference Policy nodecon statement for the IPv4
# multicast address:
nodecon 127.0.0.1 255.255.255.255 system_u:object_r:lo_node_t:
s0 - s15:c0.c255

# The equivalent MLS Reference Policy nodecon statement for the
# multicast address, however using an IPv6 address:
nodecon ff00:: ff00:: system_u:object_r:multicast_node_t:
s0 - s15:c0.c255
```

semanage (8) Command example:

```
semanage node -a -t unconfined_t -p ipv4 -M 255.255.255.255 127.0.0.2
```

This command will produce the following file in the default `<policy_name>` policy store and then activate the policy:

`/etc/selinux/<policy_name>/modules/active/nodes.local:`

```
# This file is auto-generated by libsemanage
# Do not edit directly.
```

COMMAND FAILED TO WORK on F-10

4.12.4 portcon Statement

The portcon statement is used to label udp or tcp ports.

It is also possible to add a security context to ports outside the policy using the 'semanage port' command that will associate the port (or range of ports) to a security context.

The statement definition is:

```
portcon protocol port_number port_context
```

Where:

portcon	The portcon keyword.
protocol	The protocol type. Valid entries are udp or tcp.
port_number	The port number or range of ports. The ranges are separated by a hyphen (-).
port_context	The security context for the port or range of ports.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Examples:

```
# The Standard Reference Policy portcon statements:
portcon tcp 20 system_u:object_r:ftp_data_port_t
portcon tcp 21 system_u:object_r:ftp_port_t
portcon tcp 600-1023 system_u:object_r:hi_reserved_port_t
portcon udp 600-1023 system_u:object_r:hi_reserved_port_t
portcon tcp 1-599 system_u:object_r:reserved_port_t
portcon udp 1-599 system_u:object_r:reserved_port_t

# The equivalent MLS Reference Policy portcon statements:
portcon tcp 20 system_u:object_r:ftp_data_port_t:s0
portcon tcp 21 system_u:object_r:ftp_port_t:s0
portcon tcp 600-1023 system_u:object_r:hi_reserved_port_t:s0
portcon udp 600-1023 system_u:object_r:hi_reserved_port_t:s0
portcon tcp 1-599 system_u:object_r:reserved_port_t:s0
portcon udp 1-599 system_u:object_r:reserved_port_t:s0
```

semanage (8) Command example:


```
semanage port -a -t unconfined_t -p udp 1234
```

This command will produce the following file in the default <policy_name> policy store and then activate the policy:

```
/etc/selinux/<policy_name>/modules/active/ports.local:
```

```
# This file is auto-generated by libsemanage
# Do not edit directly.

portcon udp 1234 system_u:object_r:unconfined_t
```

4.13 MLS Statements

The optional MLS policy extension adds an additional security context component that consists of the following highlighted entries:

```
user:role:type:sensitivity[:category,...]- sensitivity [:category,...]
```

These consist of a mandatory hierarchical [sensitivity](#) and optional non-hierarchical [category](#)'s. The combination of the two comprise a [level](#) or security level as shown in [Table 4-5](#). Depending on the circumstances, there can be one level defined or a [range](#) as shown in [Table 4-5](#).

Security Level (or Level)	Note that SELinux uses level, sensitivity and category in the language statements, however when discussing these the following terms can also be used: labels, classifications, and compartments.	
Consisting of a sensitivity and zero or more category entries:		
sensitivity [: category, ...]		
← Range →		
Low	-	High
sensitivity [: category, ...]		sensitivity [: category, ...]
For a process or subject this is the current level or sensitivity		For a process or subject this is the Clearance
For an object this is the current level or sensitivity		For an object this is the maximum range (for SELinux polyinstantiated directories)
SystemLow		SystemHigh
This is the lowest level or classification for the system (for SELinux this is generally 's0', note that there are no categories).		This is the highest level or classification for the system (for SELinux this is generally 's15:c0,c255', although note that they will be the highest set by the policy).

Table 4-5: Sensitivity and Category = Security Level – this table shows the meanings depending on the context being discussed.

To make the security levels more meaningful, it is possible to use the `setransd` daemon to translate these to human readable formats. The `semanage` command will allow this mapping to be defined as discussed in the [./setrans.conf file](#) section.

4.13.1 sensitivity Statement

The `sensitivity` statement defines the MLS policy sensitivity identifies and optional alias identifiers.

The statement definition is:

```
sensitivity identifier;
```

Or

```
sensitivity sens_id alias alias_id [ alias_id ];
```

Where:

<code>sensitivity</code>	The <code>sensitivity</code> keyword.
<code>sens_id</code>	The sensitivity identifier.
<code>alias</code>	The optional <code>alias</code> keyword.
<code>alias_id</code>	One or more alias identifiers in a space separated list.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	Yes

Examples:

```
# The MLS Reference Policy default is to assign 16 sensitivity
# identifiers (s0 to s15):
sensitivity s0;
....
sensitivity s15;

# The policy does not specify any alias entries, however a valid
# example would be:
sensitivity s0 alias secret wellmaybe ornot;
```

4.13.2 MLS dominance Statement

When more than one [sensitivity Statement](#) is defined within a policy, then a dominance statement is required to define the actual hierarchy between all sensitivities.

The statement definition is:

```
dominance { sens_id ... }
```

Where:

dominance The dominance keyword.

sens_id A space separated list of previously declared sensitivity identifiers (or alias) in the order lowest to highest. They are enclosed in braces ({}), and note that there is no terminating semi-colon (;).

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Example:

```
# The MLS Reference Policy dominance statement defines s0 as the
# lowest and s15 as the highest sensitivity level:

dominance { s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 }
```

4.13.3 category Statement

The category statement defines the MLS policy category identifiers⁴⁶ and optional alias identifiers.

The statement definition is:

```
category cat_id;
```

Or

```
category cat_id alias alias_id;
```

Where:

category The category keyword.

cat_id The category identifier.

alias The optional alias keyword.

alias_id One or more alias identifiers in a space separated

⁴⁶ SELinux use the term ‘category’ or ‘categories’ while some MLS systems and documentation use the term ‘compartment’ or ‘compartments’, however they have the same meaning.

list.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	Yes

Examples:

```
# The MLS Reference Policy default is to assign 256 category
# identifiers (c0 to c255):
category c0;
...
category c255;

# The policy does not specify any alias entries, however a valid
# example would be:
category c0 alias planning development benefits;
```

4.13.4 level Statement

The `level` statement enables the previously declared sensitivity and category identifiers to be combined into a Security Level.

Note there must only be one `level` statement for each [sensitivity Statement](#).

The statement definition is:

```
level sens_id [ :category_id ];
```

Where:

- | | |
|--------------------------|---|
| <code>level</code> | The <code>level</code> keyword. |
| <code>sens_id</code> | A previously declared sensitivity identifier. |
| <code>category_id</code> | An optional set of zero or more previously declared category identifiers that are preceded by a colon (:), that can be written as follows: <ul style="list-style-type: none"> The stop sign (.) separating two category identifiers means an inclusive set (e.g. <code>c0.c16</code>). The comma (,) separating two category identifiers means a non-contiguous list (e.g. <code>c21,c36,c45</code>). Both separators may be used (e.g. <code>c0.c16, c21,c36,c45</code>). |

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Examples:

```
# The MLS Reference Policy default is to assign each Security
# Level with the complete set of categories (i.e. the inclusive
# set from c0 to c255):

level s0:c0.c255;
...
level s15:c0.c255;
```

4.13.5 range_transition Statement

The `range_transition` statement is primarily used by the `init` process or administration commands to ensure processes run with their correct MLS range (for example `init` would run at `SystemHigh` and needs to initialise / run other processes at their correct MLS range). The statement was enhanced in Policy version 21 to accept other object classes.

The statement definition is (for pre-policy version 21):

```
range_transition source_domain target_execetype new_mls_range;
```

or (for policy version 21 and greater):

```
range_transition source_domain target_execetype : class new_mls_range;
```

Where:

<code>range_transition</code>	The <code>range_transition</code> keyword.
<code>source_domain</code>	A source process domain (as only the process object class is supported).
<code>target_execetype</code>	A target executable type or attribute. (i.e. an identifier for a file that has the <code>execute</code> permission set).
<code>class</code>	The optional object <code>class</code> keyword (this allows policy versions 21 and greater to specify a class other than the default of <code>process</code>).
<code>new_mls_range</code>	The new MLS range for the object class. The format of this field is described in the MLS range Definition section.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	No

Examples:

```
# A range_transition statement from the MLS Reference Policy
# showing that a process anaconda_t can transition between
# systemLow and systemHigh depending on calling applications
# level.

range_transition anaconda_t init_script_file_type:process s0 -
s15:c0.c255;

# Two range_transition statements from the MLS Reference Policy
# showing that init will transition the audit and cups daemon
# to systemHigh (that is the lowest level they can run at).

range_transition initrc_t auditd_exec_t:process s15:c0.c255;
range_transition initrc_t cupsd_exec_t:process s15:c0.c255;
```

4.13.5.1 MLS range Definition

The MLS range is appended to a number of statements and defines the lowest and highest security levels. The range can also consist of a single level as discussed at the start of the [MLS section](#).

The definition is:

```
low_level
```

Or

```
low_level - high_level
```

Where:

low_level

The processes lowest level identifier that has been previously declared by a [level Statement](#).

If a high_level is not defined, then it is taken as the same as the low_level.

-

The optional hyphen (-) separator if a high_level is also being defined.

high_level

The processes highest level identifier that has been previously declared by a [level Statement](#).

4.13.6 mlsconstrain Statement

The `mlsconstrain` statement allows further restriction on permissions for the specified object classes by using boolean expressions covering: source and target types, roles, users and security levels as described in the examples.

The statement definition is:

```
mlsconstrain class perm_set expression;
```

Where:

<code>mlsconstrain</code>	The <code>mlsconstrain</code> keyword.
<code>class</code>	One or more object classes. Multiple entries consist of a space separated list enclosed in braces <code>{}</code> .
<code>perm_set</code>	One or more permissions. Multiple entries consist of a space separated list enclosed in braces <code>{}</code> .
<code>expression</code>	The boolean expression of the constraint that is defined as follows:

```
( expression : expression )
| not expression
| expression and expression
| expression or expression
| u1 op u2
| r1 role_mls_op r2
| t1 op t2
| l1 role_mls_op l2
| l1 role_mls_op h2
| h1 role_mls_op l2
| h1 role_mls_op h2
| l1 role_mls_op h1
| l2 role_mls_op h2
| u1 op names
| u2 op names
| r1 op names
| r2 op names
| t1 op names
| t2 op names
```

Where:

`u1, r1, t1, l1, h1` = Source user, role, type, low level, high level
`u2, r2, t2, l2, h2` = Target user, role, type, low level, high level

and:

```
op : == | !=
role_mls_op : == | != | eq | dom | domby | incomp
names : name | { name_list }
name_list : name | name_list name
```

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Examples:

These examples have been taken from the [Reference Policy](#) source `../policy/mls` constraints file (the `mcs` file supports the MCS constraints).

These are built into the policy at build time and add constraints to many of the object classes.

```
# The MLS Reference Policy mlsconstrain statement for searching
# directories that comprises of multiple expressions. Only the
# first two expressions are explained.
#
# Expression 1 ( l1 dom l2 ) reads as follows:
# The dir object class search permission is allowed if the
# source lowest security level is dominated by the targets
# lowest security level.
# OR
# Expression 2 ( ( t1 == mlsfilereadtoclr ) and ( h1 dom l2 ) )
# reads as follows:
# If the source type is equal to a type associated to the
# mlsfilereadtoclr attribute and the source highest security
# level is dominated by the targets lowest security level,
# then search permission is allowed on the dir object class.

mlsconstrain dir search
  ( ( l1 dom l2 ) or
    ( ( t1 == mlsfilereadtoclr ) and ( h1 dom l2 ) ) or
    ( t1 == mlsfileread ) or
    ( t2 == mlstrustedobject ) );
```

4.13.7 mlsvalidatetrans Statement

The `mlsvalidatetrans` is the MLS equivalent of the `validatetrans` statement and is only used for file related object classes where it is used to control the ability to change the objects security context.

The statement definition is:

```
mlsvalidatetrans class expression;
```

Where:

- `mlsvalidatetrans` The `mlsvalidatetrans` keyword.
- `class` One or more file type object classes. Multiple entries consist of a space separated list enclosed in braces `{ }`.
- `expression` The boolean expression of the constraint that

is defined as follows:

```
( expression : expression )
| not expression
| expression and expression
| expression or expression
| u1 op u2
| r1 role_mls_op r2
| t1 op t2
| l1 role_mls_op l2
| l1 role_mls_op h2
| h1 role_mls_op l2
| h1 role_mls_op h2
| l1 role_mls_op h1
| l2 role_mls_op h2
| u1 op names
| u2 op names
| r1 op names
| r2 op names
| t1 op names
| t2 op names
| u3 op names
| r3 op names
| t3 op names
```

Where:

u1, r1, t1, l1, h1 = Old user, role, type, low level, high level
 u2, r2, t2, l2, h2 = New user, role, type, low level, high level
 u3, r3, t3, l3, h3 = Process user, role, type, low level, high level

and:

```
op : == | !=
role_mls_op : == | != | eq | dom | domby | incomp
names : name | { name_list }
name_list : name | name_list name
```

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Examples:

This example has been taken from the [Reference Policy](#) source ./policy/mls file.

```
# The MLS Reference Policy mlsvalidatetrans statement for
# managing the file upgrade/downgrade rules that comprises of
# multiple expressions. Only the first two expressions are
# explained.
#
# Expression 1: ( l1 eq l2 ) reads as follows:
# For a file related object to change security context, its
# current (old) low security level must be equal to the new
```

```
# objects low security level.
#
# The second part of the expression:
# or (( t3 == mlsfileupgrade ) and ( l1 domby l2 )) reads as
# follows:
# or the process type must equal a type associated to the
# mlsfileupgrade attribute and its current (old) low security
# level must be dominated by the new objects low security level.
#
mlsvalidatetrans { dir file lnk_file chr_file blk_file sock_file
fifo_file }
(( l1 eq l2 ) or
( t3 == mlsfileupgrade ) and ( l1 domby l2 )) or
( t3 == mlsfiledowngrade ) and ( l1 dom l2 )) or
( t3 == mlsfiledowngrade ) and ( l1 incomp l2 ))) and (( h1 eq h2 ) or
( t3 == mlsfileupgrade ) and ( h1 domby h2 )) or
( t3 == mlsfiledowngrade ) and ( h1 dom h2 )) or
( t3 == mlsfiledowngrade ) and ( h1 incomp h2 )));
```

4.14 Policy Support Statements

This section contains language statements used to support policy.

4.14.1 module Statement

This statement is mandatory for loadable modules (non-base) and must be the first line of any module policy source file. The identifier should not conflict with other module names within the overall policy, otherwise it will over-write an existing module when loaded via the semodule command. The semodule -l command can be used to list all active modules within the policy.

The statement definition is:

```
module module_name version_number;
```

Where:

- module The module keyword.
- module_name The module name.
- version_number The module version number in M.m.m format (where M = major version number and m = minor version numbers).

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
No	No	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Example:

```
# Using the module statement to define a loadable module called
# bind with a version 1.0.0:

module bind 1.8.0;
```

4.14.2 **require Statement**

The `require` statement is used for two reasons:

1. Within loadable module policy source files to indicate what policy components are required from an external source file (i.e. they are not explicitly defined in this module but elsewhere). The examples below show the usage.
2. Within a base policy source file, but only if preceded by the [optional Statement](#) to indicate what policy components are required from an external source file (i.e. they are not explicitly defined in the base policy but elsewhere). The examples below show the usage.

The statement definition is:

```
require { rule_list }
```

Where:

<code>require</code>	The <code>require</code> keyword.
<code>require_list</code>	One or more specific statement keywords with their required identifiers in a semi-colon (;) separated list enclosed within braces ({}).

The valid statement keywords are:

- `role`, `type`, `attribute`, `user`, `bool`, `sensitivity` and `category`. The keyword is followed by one or more identifiers in a comma (,) separated list, with the last entry being terminated with a semi-colon (;).
- `class`. The `class` keyword is followed by a single object class identifier and one or more permissions. Multiple permissions consist of a space separated list enclosed within braces ({}). The list is then terminated with a semi-colon (;).

The examples below show these in detail.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
No	Yes – But only if preceded by the optional Statement.	Yes
Conditional Policy (if) Statement	optional Statement	require Statement

Yes – But only if proceeded by the optional Statement.	Yes	No
---	------------	-----------

Examples:

```
# A series of require statements showing various entries:

require {
    role system_r;
    class security { compute_av compute_create compute_member
        check_context load_policy compute_relabel compute_user
        setenforce setbool setseparam setcheckreqprot };
    class capability2 { mac_override mac_admin };
}

#
require {
    attribute direct_run_init, direct_init, direct_init_entry;
    type initrc_t;
    role system_r;
    attribute daemon;
}

#
require {
    type nscd_t, nscd_var_run_t;
    class nscd { getserv getpwd getgrp gethost shmempwd shmemgrp
        shmemhost shmemserv };
}
```

4.14.3 optional Statement

The `optional` statement is used to indicate what policy statements may or may not be present in the final compiled policy. The statements will be included in the policy only if all statements within the `optional { rule list }` can be expanded successfully, this is generally achieved by using a [require Statement](#) at the start of the list.

The statement definition is:

```
optional { rule_list }
```

Or

```
optional { rule_list } else { rule_list }
```

Where:

<code>optional</code>	The <code>optional</code> keyword.
<code>rule_list</code>	One or more statements enclosed within braces (<code>{ }</code>). The list of valid statements is given in Table 4-3 .
<code>else</code>	An optional <code>else</code> keyword.

rule_list

As the rule_list above.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
No	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
Yes	Yes	Yes

Examples:

```
# Use of optional block in a base policy source file.

optional {
    require {
        type unconfined_t;
    } # end require

    allow acct_t unconfined_t:fd use;
} # end optional
```

```
# Use of optional / else blocks in a base policy source file.

optional {
    require {
        type ping_t, ping_exec_t;
    } # end require

    allow dhcpc_t ping_exec_t:file { getattr read execute };
    .....
    require {
        type netutils_t, netutils_exec_t;
    } # end require
    allow dhcpc_t netutils_exec_t:file { getattr read execute };
    .....
    type_transition dhcpc_t netutils_exec_t:process netutils_t;
    ...
} else {
    allow dhcpc_t self:capability setuid;
    .....
} # end optional
```

4.14.4 polycap Statement

Policy database version 22 introduced the polycap statement to allow new capabilities to be enabled or disabled via the policy. In F-10 there are two policy capabilities configured as shown in the [SELinux Filesystem](#) section, and are network_peer_controls and open_perms. An example of its usage is shown in [Appendix E – NetLabel Module Support for network_peer_controls](#).

The statement definition is:

```
policycap capability;
```

Where:

`policycap` The `policycap` keyword.

`capability` The `capability` identifier that needs to be enabled for this policy.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Example:

```
# This statement enables the network_peer_controls to be enabled
# for use by the policy.
#
policycap network_peer_controls;
```

4.14.5 permissive Statement

Policy database version 23 introduced the `permissive` statement to allow the named domain to run in permissive mode instead of running all SELinux domains in permissive mode (that was the only option prior to version 23). Note that the `permissive` statement:

1. Only tests the source context for any policy denial.
2. Can be set by the `semanage` command as it supports a `permissive` option as follows:

```
# semanage supports enabling and disabling of permissive
# mode using the following command:
# semanage permissive -a|d type

# This example will add a new module in /etc/selinux/
# <policy_name>/modules/active/modules/ called
# permissive_unconfined_t.pp and then reload the policy:

semanage permissive -a unconfined_t
```

3. Can be built into a loadable policy module so that permissive mode can be easily enabled or disabled by adding or removing the module. An example module is as follows:

```
# This is an example loadable module that would allow the
# domain to be set to permissive mode.
#
```

```
module permissive_unconfined_t 1.0.0;
require {
    type unconfined_t;
}
permissive unconfined_t;
```

The statement definition is:

```
permissive type_id;
```

Where:

permissive The permissive keyword.

type_id The type identifier of the domain that will be run
in permissive mode.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	No

Example:

```
# This is the simple statement that would allow permissive mode
# to be set on the httpd_t domain, however this statement is
# generally built into a loadable policy module so that the
# permissive mode can be easily removed by removing the module.
#
permissive httpd_t;
```

semanage (8) Command example:

```
semanage permissive -a unconfined_t
```

This command will produce the following module in the default <policy_name> policy store and then activate the policy:

```
/etc/selinux/<policy_name>/modules/active/modules/permissive_unconfined_t.pp
```

4.15 Object Class and Permission Statements

For those who write or manager SELinux policy, there is no need to define new objects and their associated permissions as these would be done by those who actually design and/or write object managers.

4.15.1 Object Classes

A list of object classes used by Fedora can be found in the [Reference Policy](#) source in the `./policy/flask/security_classes` file.

Object classes are defined within a policy as follows:

The statement definition is:

```
class class_id
```

Where:

`class` The `class` keyword.
`class_id` The `class` identifier.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	Yes

Example:

```
# Define the PostgreSQL db_tuple object class
#
class db_tuple
```

4.15.2 Permissions

A list of permissions used by Fedora can be found in the [Reference Policy](#) source in the `./policy/flask/access_vectors` file.

Permissions can be defined within policy in two ways:

1. Define class specific permissions. This is where permissions are declared for a specific object class only (i.e. the permission is not inherited by any other object class).
2. Define a set of common permissions that can then be inherited by one or more object classes. The statement for creating a set of common permissions is shown in the [Defining common Permissions](#) section.

The permission (or AVC) statement definition is:


```
class class_id [ inherits common_set ] [ { perm_set } ]
```

Where:

<code>class</code>	The <code>class</code> keyword.
<code>class_id</code>	The previously declared <code>class</code> identifier.
<code>inherits</code>	The optional <code>inherits</code> keyword that allows a set of common permissions to be inherited.
<code>common_set</code>	A previously declared common identifier as described in the Defining common Permissions section.
<code>perm_set</code>	One or more optional permission identifiers in a space separated list enclosed within braces (<code>{ }</code>).

Note:

There must be at least one `common_set` or one `perm_set` defined within the statement.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	Yes

Examples:

```
# The following example shows the db_tuple object class being  
# allocated two permissions:
```

```
class db_tuple { relabelfrom relabelto }
```

```
# The following example shows the db_blob object class inheriting  
# permissions from the database set of common permissions (as  
# described in the Defining common Permissions section):
```

```
class db_blob inherits database
```

```
# The following example (from the access_vector file) shows the  
# db_blob object class inheriting permissions from the database  
# set of common permissions and adding a further four  
# permissions:
```

```
class db_blob inherits database { read write import export }
```

4.15.2.1 Defining common Permissions

A list of common permissions used by Fedora can be found in the [Reference Policy](#) source in the `./policy/flask/access_vectors` file.

New or updated `common` permissions would only be updated by those who produce kernel or user space object managers.

The statement definition is:

```
common common_id { perm_set }
```

Where:

<code>common</code>	The <code>common</code> keyword.
<code>common_id</code>	The <code>common</code> identifier.
<code>perm_set</code>	One or more permission identifiers in a space separated list enclosed within braces (<code>{ }</code>).

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Example:

```
# Define the common PostgreSQL permissions
#
common database { create drop getattr setattr relabelfrom
relabelto }
```

4.16 Security ID (SID) Statement

There are two SID statements, the first one declares the actual SID identifier and is defined at the start of a policy source file. The second statement is used to add an initial security context to the SID that is used when SELinux initialises or as a default if an object is not labeled correctly. The [Building a Basic Policy](#) section shows their usage.

4.16.1 `sid` Statement

The `sid` statement declares the actual SID identifier and is defined at the start of a policy source file.

The statement definition is:

```
sid sid_id
```

Where:

<code>sid</code>	The <code>sid</code> keyword.
<code>sid_id</code>	The <code>sid</code> identifier. Note that there is no terminating <code>' ; '</code> .

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Example:

This example has been taken from the [Reference Policy](#) source `../policy/flask/initial_sids` file.

```
# This example was taken from the
# ../policy/flask/initial_sids file and declares some
# of the initial SIDs:
#
sid kernel
sid security
sid unlabeled
sid fs
```

4.16.2 sid context Statement

The sid context statement is used to add an initial security context to the SID that is used when SELinux initialises, or as a default if an object is not labeled correctly.

```
sid sid_id context
```

Where:

- sid The sid keyword.
- sid_id The previously declared sid identifier.
- context The initial security context associated with the SID.
Note that there is no terminating ‘;’.

The statements are valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Examples:

```
# These statements add an initial security context to an object
# that is used when SELinux initialises or as a default if a
# context is not available or labeled incorrectly.
#
# This one is from a targeted policy:
```

```
sid unlabeled system_u:object_r:unlabeled_t

# This one is from an MLS policy. Note that the security level is
# set to SystemHigh as it may need to label any object in the
# system.

sid unlabeled system_u:object_r:unlabeled_t:s15:c0.c255
```

5. Building a Basic Policy

5.1 Introduction

The objective of this section is to show how policy files are constructed, compiled and loaded using the SELinux command line tools and editors such as `vi` or `gedit` to produce a usable policy for instructional use only.

A monolithic and modular (with loadable modules) policy are built without the use of any support macros or make files from the [Reference Policy](#) source.

5.1.1 Overall Objectives

The main objectives of this section are to:

1. Show how to construct and build a simple monolithic policy.
2. Show how to construct and build a simple base module.
3. Show how to construct and build a series of loadable modules using the base module. This builds into a very [simple message filter](#) using a network client / server application and file moving (filter) application.

To examine the message filter application policy, some very minor policy errors have been introduced into the modules that will then be investigated using the SETools package in the [Policy Investigation Tools](#) section.

5.1.2 Build Requirements

To be able to build the policy files only standard SELinux utilities are required. However to build the test 'C' programs, development tools will be required, therefore ensure that the following are installed:

- `gcc` tools to compile and link the test applications (`gcc-4.3.2-7.i386` and `libgcc-4.3.2-7.i386` rpms are installed on test machine).
- The `libselinux` library from the `libselinux-devel` package (`libselinux-devel-2.0.78-1.fc10.i386` is installed on the test machine).

If the NetLabel module is being built, the NetLabel tools will need to be installed as they are not part of the standard F-10 installation (`netlabel_tools-018-1.fc10.i386.rpm` was installed on test machine).

5.1.3 The Test Policies

Normally SELinux policies are built to deny everything by default, and then enable access as required, however the example policies in this section grant access to everything and then run the test applications in their own domains to isolate them.

The policies built in this section have been tested using the follow sequence:

1. Will the system load, allow users to logon and run applications in permissive mode – If yes then:
2. Set the system to enforcing mode by `setenforce 1`, if still okay then:

3. Log out users and log in again (as now in enforcing mode, the login may fail), if okay then:
4. Edit the `config` file and set `SELINUX=enforcing`, then reboot the system, if okay then:
5. Log in users and run applications, if okay then:
6. Test that the policy meets the security requirements.

If at any stage the load fails, then the repair CD/DVD may have to be used to investigate the cause. Setting the `config` file `SELINUX` entry to `permissive` and investigating the messages and audit logs can be helpful (but not always).

5.2 Building The Policy Source Files

There are at least three ways to build a monolithic or base policy source file to experiment with:

1. Use the samples shown in this section that are valid for F-10. However as SELinux gets updated, the object classes and their associated permissions do change, therefore these samples may not be correct for other versions.
2. Rebuild the source files using the `flask security_classes`, `initial_sids` and `access_vectors` files from the [Reference Policy](#) source appropriate to the GNU / Linux distribution being used. The policy statements must then be added as necessary.

The `buildpolicy` script shown below can be used to produce the complete policy automatically from the Reference Policy source⁴⁷ using the `flask security_classes`, `initial_sids` and `access_vectors` files.

```
#!/bin/sh
#
#####
#
# This script will build an SELinux monolithic or base policy file suitable
# for building test policy for use in the SELinux Notebook. The Reference
# Policy must be available for this script to build the policy.
#
# A full description of its use is in the SELinux Notebook - The Foundations
#
# Copyright (C) 2009 Richard Haines
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.
#
#####
#
usage() {
    echo "Command format is: ./buildpolicy <policy_name> <ref_policy_root_dir>"
}
```

⁴⁷ Instructions on how to install the Reference Policy source files for F-10 are given in the [Installing the Reference Policy Source](#) section.

```
echo "Examples:"
echo "buildpolicy base.conf ."
echo "Or:"
echo "./buildpolicy policy.conf $HOME/rpmbuild/SOURCES/serefpolicy-3.5.13"
exit 1
}

if test "$1" = ""
then echo "Need policy file name"
usage
fi

if test ! -f "$2/policy/flask/security_classes"
then echo "Not a valid Reference Policy source tree"
usage
fi

echo -e "#\n# ***** WARNING - THIS POLICY MUST NOT BE USED IN LIVE
*****\n# ***** IT IS FOR TESTING ONLY
*****\n#" > $1

echo -e "##### START OF POLICY BUILD
#####\n#" >> $1

echo -e "#\n##### Start of FLASK Entries
#####\n#" >> $1
echo -e "#\n# ./policy/flask/security_classes file entries\n#" >> $1
cat "$2/policy/flask/security_classes" >> $1

echo -e "#\n# ./policy/flask/initial_sids file entries\n#" >> $1
cat "$2/policy/flask/initial_sids" >> $1

echo -e "#\n# ./policy/flask/access_vectors file entries\n#" >> $1
cat "$2/policy/flask/access_vectors" >> $1

echo -e "\n#\n##### End of FLASK Entries
#####\n#" >> $1

echo -e "#\n# This polycypac statement will be used in a netlabel module
exercise\n# to show network_peer_controls. For now comment out:\n# polycypac
network_peer_controls;\n#" >> $1

echo -e "# The only type defined for this policy:" >> $1
echo -e "type unconfined_t;\n" >> $1

echo -e "# The only role defined for this policy:" >> $1
echo -e "role unconfined_r types { unconfined_t };;\n" >> $1

echo -e "#\n# These allow rules enable all of the objects to access all of
their\n# permissions. This effectively gives access to everything.\n#" >> $1
awk '$1 == "class" {print "allow unconfined_t self:"$2 " *;"}'
"$2/policy/flask/security_classes" >> $1

echo -e "\n# The only real SELinux user defined for this policy:" >> $1
echo -e "user user_u roles { unconfined_r };;\n" >> $1

echo -e "#\n# The system_u user is defined so that objects can be labeled
with" >> $1
echo -e "# system_u:object_r as in standard policies, also so that semanage
can add" >> $1
echo -e "# ports etc. as it requires a system_u user for adding these type of
objects." >> $1
echo -e "user system_u roles { unconfined_r };;\n" >> $1

echo -e "#\n# This role constraint statement will be used to show
limiting\n# a role transition in the external gateway. For now comment
out:\n# constrain process transition ( r1 == r2 );;\n" >> $1

echo -e "#\n# These are the default labeling operations for these
objects.\n# Note that the kernel entry is unconfined_r not object_r\n#" >>
$1
awk '$1 == "sid" {if ($2 == "kernel") print $1 " " $2 "
system_u:unconfined_r:unconfined_t"}' "$2/policy/flask/initial_sids" >> $1
awk '$1 == "sid" {if ($2 != "kernel") print $1 " " $2 "
system_u:object_r:unconfined_t"}' "$2/policy/flask/initial_sids" >> $1
```

```

echo -e "\n#\n# These are the default file labeling routines.\n#" >> $1
echo "fs_use_xattr ext3 system_u:object_r:unconfined_t;" >> $1

echo "fs_use_task eventpollfs system_u:object_r:unconfined_t;" >> $1
echo "fs_use_task pipefs system_u:object_r:unconfined_t;" >> $1
echo "fs_use_task sockfs system_u:object_r:unconfined_t;" >> $1

echo "fs_use_trans mqueue system_u:object_r:unconfined_t;" >> $1
echo "fs_use_trans shm system_u:object_r:unconfined_t;" >> $1
echo "fs_use_trans tmpfs system_u:object_r:unconfined_t;" >> $1
echo "fs_use_trans devpts system_u:object_r:unconfined_t;" >> $1

echo "genfscon proc / system_u:object_r:unconfined_t" >> $1
echo "genfscon sysfs / system_u:object_r:unconfined_t" >> $1
echo "genfscon selinuxfs / system_u:object_r:unconfined_t" >> $1
echo "genfscon securityfs / system_u:object_r:unconfined_t" >> $1

echo -e "\n#\n##### END OF POLICY BUILD
#####\n#\n" >> $1

```

- There is an article [“SELinux From Scratch”](#) [Ref. 15] that describes a process using a C program and some scripts for building a test policy from the GNU / Linux kernel source tree. This process has since been enhanced and built into the kernel source tree from version 2.6.28⁴⁸, where the following files can be found that describe and build the ‘make dummy policy’(mdp):

```

Documentation/SELinux.txt
scripts/Makefile
scripts/selinux/Makefile
scripts/selinux/README
scripts/selinux/install_policy.sh
scripts/selinux/mdp/Makefile
scripts/selinux/mdp/dbus_contexts
scripts/selinux/mdp/mdp.c

```

5.2.1 Policy Source Files

The policies built in this section make use of a common [policy.conf](#) source file to demonstrate a monolithic build and a base loadable policy build (traditionally called `base.conf`). The source is shown in the [Policy Source File](#) section with [Table 5-1](#) describing the core policy components.

<i>Entry</i>	<i>Comments</i>
Security Classes (class)	These are from the F-10 Reference Policy <code>./policy/flask/security_classes</code> file entries. Note that these entries must be kept in the order shown. The policy will compile, however it will not load into the kernel (<code>load_policy err 2</code>).
Initial SIDs	These are from the F-10 Reference Policy <code>./policy/flask/initial_sids</code> file entries.
Access Vectors	These are from the F-10 Reference Policy

⁴⁸ Unfortunately F-10 ships with the 2.6.27 kernel build. Also need to be aware that if mdp is used, the kernel `selinux flask.h` file does not contain the userspace classes, therefore the process will enter these in the policy as `'class user<number>'`. Note: this needs to be confirmed with a real kernel build at some stage.

<i>Entry</i>	<i>Comments</i>
(permissions)	<code>./policy/flask/access_vectors</code> file entries.
MLS Sensitivity, category and level Statements	There are no MLS security level information in the sample policy.
MLS Constraints	There are no MLS constraints in the sample policy.
Policy Capability Statements	There are no <code>policycap</code> statements in the sample policy, however one is added later for a NetLabel exercise using <code>network_peer_controls</code> .
Attributes	There are no <code>attributes</code> in the sample policy.
Booleans	There are no <code>bool</code> statements in the sample policy.
Type / Type Alias	There is only one <code>type:unconfined_t</code> . There are no <code>typealias</code> statements.
Roles	There is only one <code>role:unconfined_r</code> .
Policy Rules	There is one allow rule for each object class (taken from the <code>security_classes</code> file) in the policy that allows unrestricted access to all its permissions as follows: <pre>allow unconfined_t self : class_name *;</pre>
Users	There is one <code>user:user_u</code> , for logging on. The <code>system_u</code> user is there to allow objects to be labeled <code>system_u:object_r</code> as in the standard Reference Policy. The <code>system_u</code> user is also required by <code>semanage(8)</code> to add network objects.
Constraints	There are no constraints in the sample policy, however one is added later to show role constraints.
Default SID labeling	These have been taken from the standard Reference Policy build with the security contexts updated. Note that the kernel is labeled <code>unconfined_r</code> and not <code>object_r</code> .
<code>fs_use_xattr</code> Statement	Only the <code>ext3</code> filesystem has been added. If the system being built supports other filesystems then these will need to be added (e.g. <code>ext4</code>).
<code>fs_use_task</code> and <code>fs_use_trans</code> Statements	These have been taken from the standard Reference Policy build.
<code>genfscon</code> Statements	Only a selection have been taken from the standard Reference Policy build.
<code>portcon</code> , <code>netifcon</code> and <code>nodecon</code> Statements	There are none of these statements in the policy.

Table 5-1: Policy Components - for the `policy.conf` and `base.conf` source file.

5.2.1.1 Problem Resolution

The following may help with resolving issues when building the examples:

1. If the files are cut from this document and then pasted into a GNU / Linux editor (such as `vi` or `gedit`) as a text file, then there could be a `cr` at the end of each line. This can cause problems with some compilers such as `checkpolicy` and `checkmodule`. To remove the `cr` use the following command:

```
cat <file_name> | tr -d \\r >new_file_name
```

2. Once the policies etc. have been built and all goes well, the `init` process will relabel the filesystem and then load the new policy, however any errors encountered will probably result in either:
 - a. GNU / Linux hanging, in which case the repair disk will be required. To allow GNU / Linux to load, the `/etc/selinux/config` file should be edited to set either `SELINUX=disabled` or the `SELINUXTYPE=` to a known working policy. The reason for the hang can then be investigated (such as correcting the policy source files and/or re-running the build commands).
 - b. The policy will be rejected by the kernel and not loaded, GNU / Linux will then load with no policy enabled, giving another chance at fixing the problem (the screen messages will generally give the reason for the rejection).

5.2.1.2 Monolithic and Base Policy Source File

The policy source file for monolithic and base loadable module is as follows:

```
#
# ***** WARNING THIS POLICY MUST NOT BE USED IN LIVE *****
# ***** IT IS FOR TESTING ONLY *****
#
#
##### Start F-10 FLASK Entries #####
#
# ./policy/flask/security_classes file
# Define the security object classes *** THESE MUST BE KEPT IN THIS ORDER ***
class security
class process
class system
class capability
class filesystem
class file
class dir
class fd
class lnk_file
class chr_file
class blk_file
class sock_file
class fifo_file
class socket
class tcp_socket
class udp_socket
class rawip_socket
class node
class netif
class netlink_socket
class packet_socket
class key_socket
class unix_stream_socket
```

```
class unix_dgram_socket
class sem
class msg
class msgq
class shm
class ipc
class passwd
class x_drawable
class x_screen
class x_gc
class x_font
class x_colormap
class x_property
class x_selection
class x_cursor
class x_client
class x_device
class x_server
class x_extension
class netlink_route_socket
class netlink_firewall_socket
class netlink_tcpdiag_socket
class netlink_nflog_socket
class netlink_xfrm_socket
class netlink_selinux_socket
class netlink_audit_socket
class netlink_ip6fw_socket
class netlink_dnrt_socket
class dbus
class nscd
class association
class netlink_kobject_uevent_socket
class appletalk_socket
class packet
class key
class context
class dccp_socket
class memprotect
class db_database
class db_table
class db_procedure
class db_column
class db_tuple
class db_blob
class peer
class capability2
class x_resource
class x_event
class x_synthetic_event
class x_application_data
# --- End of class order -----
#
# ./policy/flask/initial_sids file
# Define initial security identifiers
#
sid kernel
sid security
sid unlabeled
sid fs
sid file
sid file_labels
sid init
sid any_socket
sid port
sid netif
sid netmsg
sid node
sid igmp_packet
sid icmp_socket
sid tcp_socket
sid sysctl_modprobe
sid sysctl
sid sysctl_fs
sid sysctl_kernel
sid sysctl_net
```

```
sid sysctl_net_unix
sid sysctl_vm
sid sysctl_dev
sid kmod
sid policy
sid sctp_packet
sid devnull
#
# ./policy/flask/access_vectors file
# Define common prefixes for access vectors
#
common file { ioctl read write create getattr setattr lock relabelfrom relabelto
append unlink link rename execute swapon quotaon mounton }
common socket { ioctl read write create getattr setattr lock relabelfrom relabelto
append bind connect listen accept getopt setopt shutdown recvfrom sendto recv_msg
send_msg name_bind }
common ipc { create destroy getattr setattr read write associate unix_read
unix_write }
common database { create drop getattr setattr relabelfrom relabelto }
#
# Define the access vectors.
#
class filesystem { mount remount unmount getattr relabelfrom relabelto transition
associate quotamod quotaget }
class dir inherits file { add_name remove_name reparent search rmdir open }
class file inherits file { execute_no_trans entrypoint execmod open }
class lnk_file inherits file
class chr_file inherits file { execute_no_trans entrypoint execmod open }
class blk_file inherits file { open }
class sock_file inherits file
class fifo_file inherits file { open }
class fd { use }
class socket inherits socket
class tcp_socket inherits socket { connectto newconn acceptfrom node_bind
name_connect }
class udp_socket inherits socket { node_bind }
class rawip_socket inherits socket { node_bind }
class node { tcp_recv tcp_send udp_recv udp_send rawip_recv rawip_send
enforce_dest dccp_recv dccp_send recvfrom sendto }
class netif { tcp_recv tcp_send udp_recv udp_send rawip_recv rawip_send dccp_recv
dccp_send ingress egress }
class netlink_socket inherits socket
class packet_socket inherits socket
class key_socket inherits socket
class unix_stream_socket inherits socket { connectto newconn acceptfrom }
class unix_dgram_socket inherits socket
class process { fork transition sigchld sigkill sigstop signull signal ptrace
getsched setsched getsession getpgid setpgid getcap setcap share getattr setexec
setfscreate noatsecure siginh setrlimit rlimitinh dyntransition setcurrent execmem
execstack execheap setkeycreate setsockcreate }
class ipc inherits ipc
class sem inherits ipc
class msgq inherits ipc { enqueue }
class msg { send receive }
class shm inherits ipc { lock }
class security { compute_av compute_create compute_member check_context
load_policy compute_relabel compute_user setenforce setbool setseccparam
setcheckreqprot }
class system { ipc_info syslog_read syslog_mod syslog_console }
class capability { chown dac_override dac_read_search fowner fsetid kill setgid
setuid setpcap linux_immutable net_bind_service net_broadcast net_admin net_raw
ipc_lock ipc_owner sys_module sys_rawio sys_chroot sys_ptrace sys_pacct sys_admin
sys_boot sys_nice sys_resource sys_time sys_tty_config mknod lease audit_write
audit_control setfcap }
class capability2 { mac_override mac_admin }
class passwd { passwd chfn chsh rootok crontab }
class x_drawable { create destroy read write blend getattr setattr list_child
add_child remove_child list_property get_property set_property manage override
show hide send receive }
class x_screen { getattr setattr hide_cursor show_cursor saver_getattr
saver_setattr saver_hide saver_show }
class x_gc { create destroy getattr setattr use }
class x_font { create destroy getattr add_glyph remove_glyph use }
class x_colormap { create destroy read write getattr add_color remove_color
install uninstall use }
```

```
class x_property { create destroy read write append getattr setattr }
class x_selection { read write getattr setattr }
class x_cursor { create destroy read write getattr setattr use }
class x_client { destroy getattr setattr manage }
class x_device { getattr setattr use read write getfocus setfocus bell
force_cursor freeze grab manage }
class x_server { getattr setattr record debug grab manage }
class x_extension { query use }
class x_resource { read write }
class x_event { send receive }
class x_synthetic_event { send receive }
class x_application_data { paste paste_after_confirm copy }
class netlink_route_socket inherits socket { nlmsg_read nlmsg_write }
class netlink_firewall_socket inherits socket { nlmsg_read nlmsg_write }
class netlink_tcpdiag_socket inherits socket { nlmsg_read nlmsg_write }
class netlink_nflog_socket inherits socket
class netlink_xfrm_socket inherits socket { nlmsg_read nlmsg_write }
class netlink_selinux_socket inherits socket
class netlink_audit_socket inherits socket { nlmsg_read nlmsg_write nlmsg_relay
nlmsg_readpriv nlmsg_tty_audit }
class netlink_ip6fw_socket inherits socket { nlmsg_read nlmsg_write }
class netlink_dnrt_socket inherits socket
class dbus { acquire_svc send_msg }
class nscd { getpwd getgrp gethost getstat admin shmempwd shmegrp shmemhost
getserv shmehost }
class association { sendto recvfrom setcontext polmatch }
class netlink_kobject_uevent_socket inherits socket
class appletalk_socket inherits socket
class packet { send rcv relabelto flow_in flow_out forward_in forward_out }
class key { view read write search link setattr create }
class context { translate contains }
class dccp_socket inherits socket { node_bind name_connect }
class memprotect { mmap_zero }
class db_database inherits database { access install_module load_module get_param
set_param }
class db_table inherits database { use select update insert delete lock }
class db_procedure inherits database { execute entrypoint }
class db_column inherits database { use select update insert }
class db_tuple { relabelfrom relabelto use select update insert delete }
class db_blob inherits database { read write import export }
class peer { rcv }

#
##### End of FLASK Entries #####
#

#
# This polycap statement will be used in a netlabel module exercise
# to show network_peer_controls. For now comment out:
# polycap network_peer_controls;

# The only type defined for this policy:
type unconfined_t;

# The only role defined for this policy:
role unconfined_r types { unconfined_t };

#
# These allow rules enable all of the objects to access all of their
# permissions. This effectively gives access to everything.
#
allow unconfined_t self:security *;
allow unconfined_t self:process *;
allow unconfined_t self:system *;
allow unconfined_t self:capability *;
allow unconfined_t self:filesystem *;
allow unconfined_t self:file *;
allow unconfined_t self:dir *;
allow unconfined_t self:fd *;
allow unconfined_t self:lnk_file *;
allow unconfined_t self:chr_file *;
allow unconfined_t self:blk_file *;
allow unconfined_t self:sock_file *;
allow unconfined_t self:fifo_file *;
allow unconfined_t self:socket *;
```

```
allow unconfined_t self:tcp_socket *;
allow unconfined_t self:udp_socket *;
allow unconfined_t self:rawip_socket *;
allow unconfined_t self:node *;
allow unconfined_t self:netif *;
allow unconfined_t self:netlink_socket *;
allow unconfined_t self:packet_socket *;
allow unconfined_t self:key_socket *;
allow unconfined_t self:unix_stream_socket *;
allow unconfined_t self:unix_dgram_socket *;
allow unconfined_t self:sem *;
allow unconfined_t self:msg *;
allow unconfined_t self:msgq *;
allow unconfined_t self:shm *;
allow unconfined_t self:ipc *;
allow unconfined_t self:passwd *;
allow unconfined_t self:x_drawable *;
allow unconfined_t self:x_screen *;
allow unconfined_t self:x_gc *;
allow unconfined_t self:x_font *;
allow unconfined_t self:x_colormap *;
allow unconfined_t self:x_property *;
allow unconfined_t self:x_selection *;
allow unconfined_t self:x_cursor *;
allow unconfined_t self:x_client *;
allow unconfined_t self:x_device *;
allow unconfined_t self:x_server *;
allow unconfined_t self:x_extension *;
allow unconfined_t self:netlink_route_socket *;
allow unconfined_t self:netlink_firewall_socket *;
allow unconfined_t self:netlink_tcpdiag_socket *;
allow unconfined_t self:netlink_nflog_socket *;
allow unconfined_t self:netlink_xfrm_socket *;
allow unconfined_t self:netlink_selinux_socket *;
allow unconfined_t self:netlink_audit_socket *;
allow unconfined_t self:netlink_ip6fw_socket *;
allow unconfined_t self:netlink_dnrt_socket *;
allow unconfined_t self:dbus *;
allow unconfined_t self:nscd *;
allow unconfined_t self:association *;
allow unconfined_t self:netlink_kobject_uevent_socket *;
allow unconfined_t self:appletalk_socket *;
allow unconfined_t self:packet *;
allow unconfined_t self:key *;
allow unconfined_t self:context *;
allow unconfined_t self:dccp_socket *;
allow unconfined_t self:memprotect *;
allow unconfined_t self:db_database *;
allow unconfined_t self:db_table *;
allow unconfined_t self:db_procedure *;
allow unconfined_t self:db_column *;
allow unconfined_t self:db_tuple *;
allow unconfined_t self:db_blob *;
allow unconfined_t self:peer *;
allow unconfined_t self:capability2 *;
allow unconfined_t self:x_resource *;
allow unconfined_t self:x_event *;
allow unconfined_t self:x_synthetic_event *;
allow unconfined_t self:x_application_data *;

# The only real SELinux user defined for this policy:
user user_u roles { unconfined_r };

#
# The system_u user is defined so that objects can be labeled with
# system_u:object_r as in standard policies, also so that semanage can add
# ports etc. as it requires a system_u user for adding these type of objects.
user system_u roles { unconfined_r };

#
# This role constraint statement will be used to show limiting
# a role transition in the external gateway. For now comment out:
# constrain process transition ( r1 == r2 );

#
```

```
# These are the default labeling operations for these objects.
# Note that the kernel entry is unconfined_r not object_r
#
sid kernel system_u:unconfined_r:unconfined_t
sid security system_u:object_r:unconfined_t
sid unlabeled system_u:object_r:unconfined_t
sid fs system_u:object_r:unconfined_t
sid file system_u:object_r:unconfined_t
sid file_labels system_u:object_r:unconfined_t
sid init system_u:object_r:unconfined_t
sid any_socket system_u:object_r:unconfined_t
sid port system_u:object_r:unconfined_t
sid netif system_u:object_r:unconfined_t
sid netmsg system_u:object_r:unconfined_t
sid node system_u:object_r:unconfined_t
sid igmp_packet system_u:object_r:unconfined_t
sid icmp_socket system_u:object_r:unconfined_t
sid tcp_socket system_u:object_r:unconfined_t
sid sysctl_modprobe system_u:object_r:unconfined_t
sid sysctl system_u:object_r:unconfined_t
sid sysctl_fs system_u:object_r:unconfined_t
sid sysctl_kernel system_u:object_r:unconfined_t
sid sysctl_net system_u:object_r:unconfined_t
sid sysctl_net_unix system_u:object_r:unconfined_t
sid sysctl_vm system_u:object_r:unconfined_t
sid sysctl_dev system_u:object_r:unconfined_t
sid kmod system_u:object_r:unconfined_t
sid policy system_u:object_r:unconfined_t
sid scmp_packet system_u:object_r:unconfined_t
sid devnull system_u:object_r:unconfined_t

#
# These are the default file labeling routines.
#
fs_use_xattr ext3 system_u:object_r:unconfined_t;
fs_use_task eventpollfs system_u:object_r:unconfined_t;
fs_use_task pipefs system_u:object_r:unconfined_t;
fs_use_task sockfs system_u:object_r:unconfined_t;
fs_use_trans mqueue system_u:object_r:unconfined_t;
fs_use_trans shm system_u:object_r:unconfined_t;
fs_use_trans tmpfs system_u:object_r:unconfined_t;
fs_use_trans devpts system_u:object_r:unconfined_t;
genfscon proc / system_u:object_r:unconfined_t
genfscon sysfs / system_u:object_r:unconfined_t
genfscon selinuxfs / system_u:object_r:unconfined_t
genfscon securityfs / system_u:object_r:unconfined_t

#
##### END OF POLICY BUILD #####
#
```

5.2.1.3 file_contexts File

The file_contexts file for the build is as follows:

```
/ system_u:object_r:unconfined_t
/* system_u:object_r:unconfined_t
```

5.2.1.4 default_contexts File

The default_contexts file is to ensure that the initial logon process uses the unconfined_r:unconfined_t role / type pair and is as follows:

```
unconfined_r:unconfined_t unconfined_r:unconfined_t
```

Note that this file will only be required when the additional loadable modules are built as they contain multiple types associated to a single role (therefore the logon process

needs to know which of the types to use for the users `user:role:type` security context).

5.2.1.5 seusers File

The `seusers` file is not mandatory, however one is added as all policies tend to have one, also when adding additional users via `semanage`, one will be required.

```
system_u:system_u
user_u:user_u
__default__:user_u
```

5.2.1.6 dbus_contexts File

The `dbus_contexts` file is required to allow X-Windows to run and is as follows:

```
<!DOCTYPE busconfig PUBLIC "-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
<busconfig>
  <selinux>
  </selinux>
</busconfig>
```

5.3 Building the Monolithic Policy

The basic steps to produce a simple monolithic test policy are:

1. Ensure you are logged on as ‘root’ and SELinux is running in permissive mode (`setenforce 0`) to perform the build process. It is assumed that the files are built in the `./notebook-source/monolithic-policy` directory.
2. Produce a `policy.conf` file with a text editor (such as `vi` or `edit`) containing the contents shown in the [Policy Source File](#) section.
3. Produce a `file_contexts` file with the contents shown in the [file_contexts file](#) section. This will be used to relabel the file system.
4. Produce a `dbus_contexts` file with the contents shown in the [dbus_contexts file](#) section. This is required for X-Windows to load as it uses the `dbus` messaging service that has a SELinux user space object manager.
5. Find the maximum policy version the SELinux kernel will support by executing the following command:

```
cat /selinux/policyvers
23
```

The output for the F-10 kernel should be ‘23’ depending on any package updates that have been added.

6. Compile the policy with `checkpolicy` to produce the binary policy file:

```
checkpolicy -c23 -o policy.23 policy.conf
```

The output from the compilation should be:


```
checkpolicy: loading policy configuration from policy.conf
checkpolicy: policy configuration loaded
checkpolicy: writing binary representation (version 23) to
policy.23
```

7. Make the following directories to store the policy:

```
mkdir /etc/selinux/monolithic-test/policy
mkdir -p /etc/selinux/monolithic-test/contexts/files
```

8. Copy the following files to SELinux policy area:

```
cp policy.23 /etc/selinux/monolithic-test/policy
cp seusers /etc/selinux/monolithic-test/seusers
cp dbus_contexts /etc/selinux/monolithic-test/contexts
cp file_contexts /etc/selinux/monolithic-test/contexts/files
```

9. The file and directory list in the /etc/selinux/monolithic-test directory area should now consist of the following:

```
monolithic-test:
drwxr-xr-x 3 root root 4096 2008-09-10 13:04 contexts
drwxr-xr-x 2 root root 4096 2008-09-10 13:59 policy
-rw-r--r-- 1 root root 70 2008-09-10 14:00 seusers

monolithic-test/contexts:
-rw-r--r-- 1 root root 195 2008-09-10 13:04 dbus_contexts
drwxr-xr-x 2 root root 4096 2008-09-10 14:00 files

monolithic-test/contexts/files:
-rw-r--r-- 1 root root 48 2008-09-10 14:00 file_contexts

monolithic-test/policy:
-rw-r--r-- 1 root root 11926 2008-09-10 13:59 policy.23
```

10. Edit the /etc/selinux/config file and change the entries shown. This will set permissive mode and the location of the policy that will be loaded on the next re-boot. Note - do not put any spaces after these entries.

```
SELINUX=permissive
SELINUXTYPE=monolithic-test
```

11. To allow file system relabeling to be actioned on reboot execute the following command:

```
touch /.autorelabel
```

12. Optionally clear the log files so that they are clear for easier reading after the reboot:

```
> /var/log/messages
> /var/log/audit/audit.log
```

13. Reboot the system. During the boot process, the file system should be re-labeled.

```
shutdown -r now
```

5.3.1 Checking the Build

Once the system has reloaded, SELinux will be running in ‘permissive’ mode. Logon as root and use either `seaudit`, `troubleshooter` or simply `tail` in a couple of ‘terminal windows’ to view the logs:

```
# In one terminal window run:
tail -f /var/log/messages

# In another terminal window run:
tail -f /var/log/audit/audit.log
```

There should be entries for the boot process in the `/var/log/messages` file, however the `/var/log/audit/audit.log` file should only contain entries for the audit daemon, user logon and role change for the logon process.

If the system is ‘working’ (i.e. it should be stable, load the desktop and allow utilities to be loaded from the menus), then SELinux can be set to enforcing mode by:

```
setenforce 1
```

The new policy will be enforced and the only entries in the logs should be about setting enforcing mode.

If the system is unstable when rebooted, then see the [Problem Resolution](#) section for a possible resolution.

5.4 Building the Base Policy Module

This exercise will build the mandatory base policy module that uses the same policy source file as the monolithic policy discussed above.

The basic steps to produce a simple base test policy are:

1. Ensure you are logged on as ‘root’ and SELinux is running in permissive mode (`setenforce 0`) to perform the build process. It is assumed that the files are built in the `./notebook-source/modular-policy-base` directory.
2. Produce a `base.conf` file with a text editor (such as `vi` or `gedit`) containing the contents shown in the [Policy Source File](#) section.
3. Produce a `base.fc` file with the contents shown in the [file_contexts](#) file section. This will be used to relabel the file system.
4. Produce a `default_contexts` file with the contents shown in the [default_contexts](#) file section. This will be used to ensure that the correct context is used for the logon process (only really needed when the additional loadable modules are built).

5. Produce an `seusers` file with the contents shown in the [seusers](#) file section.
6. Produce a `dbus_contexts` file with the contents shown in the [dbus_contexts](#) file section. This is required for X-Windows to load as it uses the dbus messaging service that has a SELinux user space object manager.
7. Compile the policy with `checkmodule` to produce an intermediate binary policy file:

```
checkmodule -o base.mod base.conf
```

The output from the compilation should be:

```
checkmodule: loading policy configuration from base.conf
checkmodule: policy configuration loaded
checkmodule: writing binary representation (version 8) to base.mod
```

8. Package the policy with `semodule_package`, this will produce a base policy module file (note – if successful there are no output messages):

```
semodule_package -o base.pp -m base.mod -f base.fc -s
seusers
```

9. Make the following directories to store the policy:

```
mkdir /etc/selinux/modular-test/policy
mkdir -p /etc/selinux/modular-test/contexts/files
mkdir -p /etc/selinux/modular-test/modules/active/modules
```

10. Copy the following files to SELinux policy area:

```
cp seusers /etc/selinux/modular-test
cp dbus_contexts /etc/selinux/modular-test/contexts
cp default_contexts /etc/selinux/modular-test/contexts
```

11. Install the base policy with `semodule`. This will produce a base policy and a number of files, some of which will be empty (note – if successful there are no output messages):

```
semodule -s modular-test -b base.pp
```

12. The file and directory list in the `/etc/selinux/modular-test` directory area should now consist of the following:

```
modular-test:
drwxr-xr-x 3 root root 4096 2008-09-10 20:12 contexts
drwx----- 3 root root 4096 2008-09-10 18:55 modules
drwxr-xr-x 2 root root 4096 2008-09-10 18:55 policy
-rw-r--r-- 1 root root 70 2008-09-10 18:55 seusers
modular-test/contexts:
-rw-r--r-- 1 root root 195 2008-09-10 20:12 dbus_contexts
-rw-r--r-- 1 root root 52 2008-09-10 20:12 default_contexts
drwxr-xr-x 2 root root 4096 2008-09-10 18:55 files
-rw-r--r-- 1 root root 0 2008-09-10 18:55 netfilter_contexts

modular-test/contexts/files:
-rw-r--r-- 1 root root 48 2008-09-10 18:55 file_contexts
-rw-r--r-- 1 root root 0 2008-09-10 18:55 file_contexts.homedirs
```

```
modular-test/modules:
drwx----- 3 root root 4096 2008-09-10 18:55 active
-rw----- 1 root root 0 2008-09-10 18:49 semanage.read.LOCK
-rw----- 1 root root 0 2008-09-10 18:49 semanage.trans.LOCK

modular-test/modules/active:
-rw----- 1 root root 19741 2008-09-10 18:55 base.pp
-rw----- 1 root root 32 2008-09-10 18:55 commit_num
-rw----- 1 root root 48 2008-09-10 18:55 file_contexts
-rw-r--r-- 1 root root 0 2008-09-10 18:55 file_contexts.homedirs
-rw----- 1 root root 48 2008-09-10 18:55 file_contexts.template
-rw----- 1 root root 0 2008-09-10 18:55 homedir_template
drwx----- 2 root root 4096 2008-09-10 18:55 modules
-rw----- 1 root root 0 2008-09-10 18:55 netfilter_contexts
-rw-r--r-- 1 root root 11028 2008-09-10 18:55 policy.kern
-rw-r--r-- 1 root root 70 2008-09-10 18:55 seusers.final
-rw-r--r-- 1 root root 70 2008-09-10 18:55 users_extra

modular-test/modules/active/modules:

modular-test/policy:
-rw-r--r-- 1 root root 11028 2008-09-10 18:55 policy.23
```

13. Edit the `/etc/selinux/config` file and change the entries shown. This will set permissive mode and the policy location that will be loaded on the next re-boot. Note - do not put any spaces after these entries.

```
SELINUX=permissive
SELINUXTYPE=modular-test
```

This will set permissive mode so if the policy is too restrictive it will still allow a login at least. The SELinux policy name/location is also added (`modular-test`). Note do not put any spaces after the entries.

14. To allow a file system relabeling to be actioned on reboot execute the following command:

```
touch /.autorelabel
```

15. Optionally clear the log files so that they are clear for easier reading:

```
> /var/log/messages
> /var/log/audit/audit.log
```

16. Reboot the system. During the boot process, the file system should be re-labeled.

```
shutdown -r now
```

5.4.1 Checking the Base Policy Build

Once the system has reloaded, SELinux will be running in 'permissive' mode. Logon as root and follow the same routine as defined in the [Checking The Build](#) section.

5.5 Building the Loadable Modules

5.5.1 Overview of modules

In the sections that follow there are a number of loadable modules built with supporting 'C' programs that form a very simple message filter service as shown in [Figure 5-21](#). The external and internal gateways are client / server applications making use of SEMARK services that are built into `iptables` as discussed in [SELinux Networking](#) section. The server component can also read and write files to / from a protected directory area (or message queues). The message filter itself is a simple file mover application that moves files from one queue to another.

The modules attempt to use as many SELinux statements and rules as possible that are then analysed in the [policy investigation](#) section.

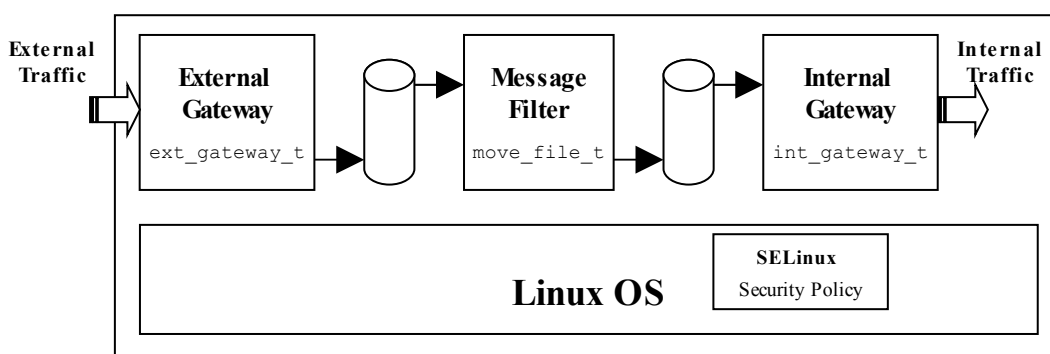


Figure 5-21: Message Filter Components

The components that form the message filter are:

External Gateway – This has a loadable module `ext_gateway.conf` that defines the policy for the external gateway, it also includes an `optional` section that is loaded when other message filter modules are loaded. The gateway requires client / server applications (`client.c` and `server.c`) to be compiled for testing and `iptables` (the `mangle` table) to be loaded for SEMARK testing.

NetLabel Service – This is a simple `netlabel.conf` module that just adds a label at peer level. As F-10 does not have NetLabel services installed as default, `yum` is used to install the service.

Internal Gateway – This is a version of the external gateway module that has been modified to handle internal processing permissions (`int_gateway.conf`). It requires additional entries in the `iptables` as it uses a different network port. The same client / server applications are used as for the external gateway.

File Mover - This has a loadable module `move_file.conf` that defines the policy for moving files between the external and internal gateways. There is also an application (`move_file.c`) that copies files from one message queue to another (but controlled by the policy).

The security policy for the message filter is simply:

1. No other application must use the secured ports configured in the `iptables` and allocated to the gateways. The secure ports are:

port 1111 and labeled `int_gateway_packet_t`

port 9999 and labeled `ext_gateway_packet_t`

All other ports are labeled: `default_secmark_packet_t`

2. The message queues and files must be protected from all possible access (read, write, delete etc.) by other domains.

The assumptions are:

1. The SELinux policy will always be in enforcing mode while the message filter is active.
2. The SELinux message filter modules may be in permissive mode for the initial file and directory configuration / initialisation via `restorecon` (this is so that permissions such as `relabelto / relabelfrom` are limited to the absolute minimum, in fact only the `iptables` need relabeling permissions as they are loaded under the `unconfined_t` domain).

The modules are built and tested in the following sequence:

1. The external gateway is built along with the client / server applications. This is used to demonstrate the basic `secmark` functionality using the `iptables`.
2. The `NetLabel` module is then built to demonstrate adding a `netlabel` to the peer network service.
3. The internal gateway and the file mover application and module are finally built to demonstrate the overall message filter as shown in [Figure 5-21](#).

Once these have been built and tested, the policy will be reviewed in the [Policy Investigation Tools](#) section.

Any comments or views on the modules, applications and their testing are welcome.

5.6 Building the SECMARK Test Loadable Module

The SECMARK tests make use of the external gateway loadable module. The objective of this module is to prove that SECMARK labels can be added to packets, and that depending on the label assigned, those packets can be granted access to the correct domain and denied access other domains using SELinux enforcement.

The tests will use various client / server configurations using the network loop back (`lo`) interface (see [Figure 5-22](#)) as follows:

1. Use a 'secure' client / server running in the `ext_gateway_t` domain that will show that packets labeled:

`system_u:object_r:ext_gateway_packet_t`

on ports 9999 will get through, while other ports will NOT get through, as they would be labeled:

`system_u:object_r:default_secmark_packet_t`

2. Use an 'insecure' client / server running in the `unconfined_t` domain that will show that packets labeled:

`system_u:object_r: ext_gateway_packet_t`

will NOT get through, while other ports will get through, as they would be labeled:

```
system_u:object_r:default_secmark_packet_t
```

3. A mixture of secure and insecure client / server configurations to show access is denied by SELinux unless both services are running in the `ext_gateway_t` domain using port 9999 on the `lo` network.

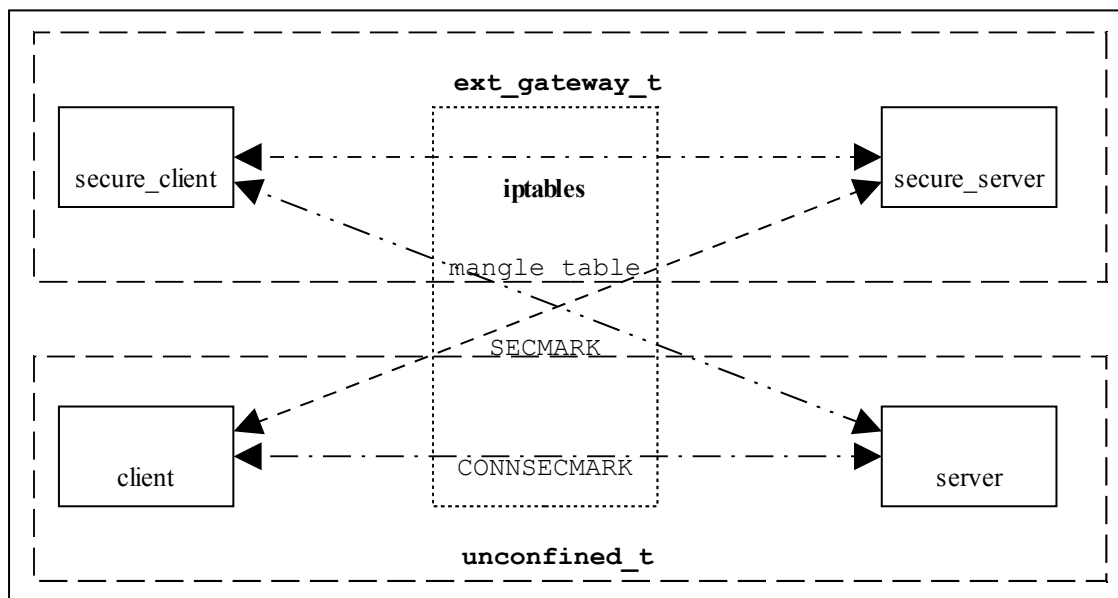


Figure 5-22: SECMARK Testing – *The scenarios for testing the access allowed for SECMARK packets. Note that not all of these tests will be described.*

The SECMARK test loadable module (`ext_gateway.conf`) has a boolean called `ext_gateway_audit` that by default enables the transition, send and receive audit events to be logged when successful, these events are shown in the test results below. The `auditallow` statements can be disabled by using the following command:

```
setsebool -P ext_gateway_audit=false
```

To test SECMARK functionality the following will need to be built and installed:

- The base loadable module built in the [Building a Base Loadable Module Policy](#) section is installed and active.
- A loadable policy module (`ext_gateway`) that will enforce the SECMARK policy configured via `iptables`. Note the following points:
 - a. The `ext_gateway` module requires a new role of `message_filter_r` to be added to SELinux. This has only been added to demonstrate a [role transition rule](#).
 - b. The `ext_gateway` module has an [optional](#) section that is only enabled when other modules are loaded as a further exercise for the message filter service.
- An `iptables` configuration file that will set-up the mangle table to mark packets with SECMARK and CONNSECMARK labels.

- Two executable clients (`secure_client` and `client`) and two executable servers (`secure_server` and `server`) that will be used to test the SECMARK functionality. Note that the clients and servers are built using common source code, and are only labeled differently to allow testing (the secure executables are labeled `secure_services_exec_t` while the others are labeled by default with `unconfined_t`).

The following steps need to be followed to build the test services. It is assumed that the services are installed in `./notebook-source/message_filter/gateway`:

1. Ensure you are logged on as 'root' and SELinux is running in permissive mode (`setenforce 0`) to perform the build process.
2. Produce a `ext_gateway.conf` loadable module file with a text editor (such as `vi` or `gedit`) containing the contents shown below:

```
module ext_gateway 1.0.0;

#####
#
# This Loadable Module will allow SECMARK, NetLabel and a simple
# Message Filter to be tested with the simple client / server 'C'
# programs (client.c and server.c).
#
# The module is used with the base.conf that sets up the unconfined_t
# space. This module can be built by:
#   checkmodule -m ext_gateway.conf -o ext_gateway.mod
#   semodule_package -o ext_gateway.pp -m ext_gateway.mod
#   -f gateway.fc
#   semodule -v -s modular-test -i ext_gateway.pp
#
# The gateway.fc file can be modified to reflect where the client /
# binaries will located (currently /usr/local/bin) and once they have
# been compiled the exe's need to be labeled by restorecon:
#   restorecon -f restorefiles_gateway
#
# The test requires the client.c and server.c programs compiled by:
#   gcc -o secure_server server.c -lselinux
#   gcc -o secure_client client.c -lselinux
# The executables will be labeled secure_services_exec_t
#
#   gcc -o server server.c -lselinux
#   gcc -o client client.c -lselinux
# The executables will be labeled the default of unconfined_t
#
# The secure port for this external gateway is 9999 and can only be
# read/write by the secure client / server. The internal gateway will
# use port 1111 and can only be read/write by the secure client / server.
# Any other port can be read / write by the standard client / server.
#
# The iptables_secmark script can be modified to other ports if required.
# WARNING - If the iptables are not loaded to label packets, ports etc.
# then the standard client / server can use the secure ports.
#
# Run setenforce 1, the policy can be tested using combinations of:
#   ./server <port>
#   ./secure_server <port>
#
#   ./client <host> <port>
#   ./secure_client <host> <port>
#
# There is a boolean that can turn off the auditallow statements:
#   setsebool [-P] ext_gateway_audit off
#
# The module transitions to a role of message_filter_r simply to show
# a role transition. To add the role to user_u the semanage command is
# used as follows:
#   semanage user -m -R "message_filter_r unconfined_r" user_u
# Note: Need to put in the unconfined_r role as semanage will remove it
```



```
# from the current policy otherwise, causing much grief. #
# #
# To allow the message filter to be tested with the move_file.conf #
# (and int_gateway.conf) modules, a set of optional statements are #
# added that will be enabled once the move_file.conf module is loaded #
# for testing. This allows the server application to either read or #
# write files to the message queues as described in the server.c #
# comments section (and the SELinux Notebook). #
# #
#####
require {
    type unconfined_t;
    role unconfined_r;
    class packet { send recv relabelto };
    class process { fork sigchld transition siginh rlimitinh noatsecure };
    class file { entrypoint read getattr execute relabelto unlink write
create };
    class filesystem { getattr associate };
    class chr_file { read write getattr };
    class dir { read search getattr write add_name remove_name };
    class fd use;
    class lnk_file read;
    class tcp_socket { write listen node_bind name_bind accept bind read
name_connect connect create getopt };
    class association recvfrom;
    class unix_stream_socket { create connect };
}

attribute message_filter_domains;

# All iptables SECMARK packets other than ports 9999 and 1111 are marked
# with this label:
type default_secmark_packet_t;

# The external gateway will have a SECMARK in the iptables of this label:
type ext_gateway_packet_t;

# The external gateway will run in this domain:
type ext_gateway_t;
# The binaries will be labeled:
type secure_services_exec_t;

# Add the gateway domain to the attribute:
typeattribute ext_gateway_t message_filter_domains;

# Use message_filter_r role and role transition for the gateway:
role message_filter_r types ext_gateway_t;
allow unconfined_r message_filter_r;
role_transition unconfined_r secure_services_exec_t message_filter_r;

# boolean to enable / disable audit events
bool ext_gateway_audit true;

if ( ext_gateway_audit ) {
    # Audit the send and recv events:
    auditallow unconfined_t default_secmark_packet_t : packet { send recv };
    # Audit the send and recv events:
    auditallow ext_gateway_t ext_gateway_packet_t : packet { send recv };
    # Audit the transition:
    auditallow unconfined_t ext_gateway_t : process { transition };
}
# End of conditional statements

# Allow all ports except 1111 & 9999 to be handled by unconfined_t:
allow unconfined_t default_secmark_packet_t : packet { send recv };

# Allow unconfined_t to relabel the secure ports. This is needed so that
# iptables can be updated easily. Note: Against security policy, however
# these need to be loaded at boot time when the policy is in enforcing mode
# so no choice !!
allow unconfined_t ext_gateway_packet_t : packet relabelto;
allow unconfined_t default_secmark_packet_t : packet relabelto;

# Allow gateway access only to secure ports:
```

```
allow ext_gateway_t ext_gateway_packet_t : packet { send recv };

# Allow the external gateway to transition to ext_gateway_t by using the
# type_transition statement (note that the internal gateway does not use
# this method but transitions via runcon instead):
allow unconfined_t ext_gateway_t : process { transition };
allow unconfined_t secure_services_exec_t : file { read execute getattr };
allow ext_gateway_t secure_services_exec_t : file { entrypoint };
type_transition unconfined_t secure_services_exec_t : process ext_gateway_t;
#

# Stop segmentation faults
allow ext_gateway_t unconfined_t : filesystem associate;
allow unconfined_t ext_gateway_t : process noatsecure;
dontaudit unconfined_t ext_gateway_t : process { siginh rlimitinh };
#

# Allow the ext_gateway_t access to areas under unconfined_t domain:
allow ext_gateway_t unconfined_t : packet { recv send };
allow ext_gateway_t unconfined_t : chr_file { read write getattr };
allow ext_gateway_t unconfined_t : dir search;
allow ext_gateway_t unconfined_t : fd use;
allow ext_gateway_t unconfined_t : filesystem getattr;
allow ext_gateway_t unconfined_t : tcp_socket name_connect;
allow ext_gateway_t unconfined_t : association recvfrom;
dontaudit ext_gateway_t unconfined_t : process sigchld;
allow ext_gateway_t self : dir search;
allow ext_gateway_t self : tcp_socket { read create connect };

# For client and server to access the shared libc:
allow ext_gateway_t unconfined_t : file { read getattr execute };
dontaudit ext_gateway_t unconfined_t : dir { getattr };
allow ext_gateway_t unconfined_t : lnk_file read;
#

# Required if use host name instead of the IP address (e.g. localhost
# instead of 127.0.0.1) in the client command line:
dontaudit ext_gateway_t self : unix_stream_socket { create connect };

# Required to get context information when using the libselinux api calls
# getcon() and getpeercon():
allow ext_gateway_t self : file read;
allow ext_gateway_t self : tcp_socket getopt;

# Required to allow setfiles to relabel the secure client/server binaries:
# Note: Against security policy so commented out as can do this at
# system build time with setenforce 0
# allow unconfined_t secure_services_exec_t : file { write relabelto };
# allow secure_services_exec_t unconfined_t : filesystem associate;

# These entries are for the server only:
allow ext_gateway_t self : tcp_socket { listen write accept bind };
allow ext_gateway_t unconfined_t : tcp_socket { name_bind node_bind };
#

#
##### START OPTIONAL SECTION #####
#
optional {
#
#####
#
# These entries are for the message filter part of the exercise #
# where files are moved from the in_queue to the out_queue by the #
# message filter (move_file.c) application. #
# #
# These rules allow ext_gateway_t to write files to the #
# in_queue. The int_gateway_t is allowed to read and remove #
# files from the out_queue. #
# #
#####
#
require {
# These are defined in the move_file.conf module:
type in_queue_t, out_queue_t, in_file_t, out_file_t;
```

```
# This is defined in the int_gateway.conf module:
type int_gateway_t;
}
# Allow the external gateway access to in_queue rules:
# The server application then writes the file to the in_queue:
type_transition ext_gateway_t in_queue_t : file in_file_t;
allow ext_gateway_t in_queue_t : dir { read getattr write search add_name };
allow ext_gateway_t in_file_t : file { write create getattr };
allow in_file_t unconfined_t : filesystem associate;
dontaudit ext_gateway_t unconfined_t : filesystem getattr;
dontaudit ext_gateway_t self : file getattr;

# Allow the internal gateway access to out_queue rules:
type_transition int_gateway_t out_queue_t : file out_file_t;
allow int_gateway_t out_queue_t : dir { read getattr write remove_name
search };
allow int_gateway_t out_file_t : file { read getattr unlink };
}
#
##### END OPTIONAL SECTION #####
#
```

3. Produce a `gateway.fc` file (a segment that will be added to the `file_contexts` file during the build) with the contents shown below. This will be used to relabel application files and directories.

```
/usr/local/bin/secure_client system_u:object_r:secure_services_exec_t
/usr/local/bin/secure_server system_u:object_r:secure_services_exec_t
/usr/local/bin/client system_u:object_r:unconfined_t
/usr/local/bin/server system_u:object_r:unconfined_t
```

4. Compile the policy with `checkmodule` to produce an intermediate binary policy file:

```
checkmodule -m ext_gateway.conf -o ext_gateway.mod
```

5. Package the policy with `semodule_package`, this will produce a policy module file:

```
semodule_package -o ext_gateway.pp -m ext_gateway.mod -f gateway.fc
```

6. Install the loadable module with `semodule` (note – if successful there are no output messages):

```
semodule -v -s modular-test -i ext_gateway.pp
```

7. If there are no errors reported, then the loadable module has been added to the policy store and loaded as a part of the policy. The policy module can be checked by:

```
semodule -s modular-test -l
```

8. Produce a ‘C’ application called `client.c` with the contents shown below:

```
/*
/*
/* This is the client component for the Notebook demo modular policy. It
/* will be used to demonstrate SECMARK, NetLabel and Message Filter
/* loadable modules.
/*
/*
```

```
/* Copyright (C) 2009 Richard Haines */
/* */
/* This program is free software: you can redistribute it and/or modify */
/* it under the terms of the GNU General Public License as published by */
/* the Free Software Foundation, either version 3 of the License, or */
/* (at your option) any later version. */
/* */
/* This program is distributed in the hope that it will be useful, */
/* but WITHOUT ANY WARRANTY; without even the implied warranty of */
/* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the */
/* GNU General Public License for more details. */
/* */
/* You should have received a copy of the GNU General Public License */
/* along with this program. If not, see <http://www.gnu.org/licenses/>. */
/* */
/*****
/* This client connects to the server that builds a buffer containing */
/* information on ports and contexts used by the server, returning this */
/* to the client. */
/* */
/* The client is compiled as follows: */
/* gcc client.c -o client -lselinux */
/* (This is labeled system_u:object:unconfined_t) */
/* gcc client.c -o secure_client -lselinux */
/* (This is labeled system_u:object:secure_services_exec_t) */
/* */
/* For the tests, the binaries should be installed in /usr/local/bin and */
/* then the restorecon -f restorefiles_gateway run once the */
/* external_gateway loadable module has been installed. */
/*****
/* For SECMARK, NetLabel and Message Filter demos the clients are called */
/* as follows: */
/* ./client <port> - Where the port is any you like (e.g. 1234) */
/* ./secure_client <port> - Where for the demo the ports are 1111 */
/* and 9999 as the iptables mangle table has been configured for these. */
/* */
/*****
/* */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <selinux/selinux.h>

#define MAXBUFFERSIZE 256
#define ENFORCING 1

#define ESC 0x1B

char red [] = "0;31";
char green [] = "0;32";
char reset [] = "0";

int main(int argc, char *argv [])
{
    int rc, sock_fd, bytes_received;
    char buffer [MAXBUFFERSIZE];
    struct hostent *server_info;
    struct sockaddr_in server_addr;
    short client_port;
    security_context_t context, peer_context;
    char *peer_context_str;

    if (argc != 3) {
        fprintf (stderr, "usage: %s <hostname> <port> (Use port 9999 for secure
port test)\n", argv[0]);
        exit (1);
    }

    client_port = atoi (argv [2]);

    if ((server_info = gethostbyname (argv [1])) == NULL) {
```

```
    perror ("Client gethostbyname");
    exit (1);
}

if (rc = security_getenforce () != ENFORCING)
    printf ("Should be in enforcing mode for valid testing\n");

if ((sock_fd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
    perror ("Client Socket");
    exit (1);
}

bzero((char *) &server_addr, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(client_port);
server_addr.sin_addr = *((struct in_addr *)server_info->h_addr);

if (connect (sock_fd, (struct sockaddr *)&server_addr, sizeof(struct
sockaddr)) == -1) {
    perror ("Client connect");
    exit (1);
}

// clear the buffer
memset (buffer, 0, sizeof(buffer));
if ((bytes_received = recv (sock_fd, buffer, MAXBUFFERSIZE-1, 0)) == -1) {
    perror ("Client recv");
    exit (1);
}

buffer [bytes_received] = '\0'; // Add null at end of line.
printf ("\033[%smServer Information in RED:\n", red);
printf ("%s \n", buffer);

// Print the Clients context information
if (rc = getcon (&context) < 0) {
    perror ("Client context");
    exit (1);
}

if (rc = getpeercon (sock_fd, &peer_context) < 0)
    peer_context_str = strdup ("No Peer Context Available");
else {
    peer_context_str = strdup (peer_context);
    freecon (peer_context);
}

printf ("\033[%smClient Information in GREEN:\n", green);
printf ("Client Context: %s \nClient Peer Context: %s \n", context,
peer_context_str);

freecon (context);
close (sock_fd);
printf ("\033[%sm\n", reset);
return 0;
}
```

9. Compile two versions of the client by running:

```
gcc -o secure_client client.c -lselinux
gcc -o client client.c -lselinux
```

10. Produce a 'C' application called server.c with the contents shown below:

```
/*
/*
/* *****
/* This is the server component for the Notebook demo modular policy. It
/* will be used to demonstrate SECMARK and NetLabel network functionality
/* and also creates and reads files for the Message Filter example.
/*
/*
/* Copyright (C) 2009 Richard Haines
/*
/*
```

The SELinux Notebook - The Foundations

```
/* This program is free software: you can redistribute it and/or modify */
/* it under the terms of the GNU General Public License as published by */
/* the Free Software Foundation, either version 3 of the License, or */
/* (at your option) any later version. */
/*
/* This program is distributed in the hope that it will be useful, */
/* but WITHOUT ANY WARRANTY; without even the implied warranty of */
/* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the */
/* GNU General Public License for more details. */
/*
/* You should have received a copy of the GNU General Public License */
/* along with this program. If not, see <http://www.gnu.org/licenses/>. */
/*
/*****
/*
/* The server is compiled as follows: */
/* gcc server.c -o server -lselinux */
/* (This is labeled system_u:object:unconfined_t) */
/* gcc server.c -o secure_server -lselinux */
/* (This is labeled system_u:object:secure_services_exec_t) */
/*
/* For the tests, the binaries should be installed in /usr/local/bin and */
/* then the restorecon -f restorefiles_gateway run once the */
/* external_gateway loadable module has been installed. */
/*
/*****
/* The server receives a connection from a client (but no data) and then */
/* builds a buffer containing information on ports and contexts used, */
/* returning this to the client (the default action). */
/*
/*****
/* For the SECMARK and NetLabel demo the servers are called as follows: */
/* server <port> - Where the port is any you like (e.g. 1234) */
/* secure_server <port> - Where for the demo are ports 9999 and 1111 */
/* as the test iptables mangle table has been configured for these. */
/* ports. */
/*****
/* For the Message Filter demo the servers are called as follows: */
/*-----*/
/* To queue messages to the Message Filter's IN queue: */
/* secure_server <port> in - Where the port is 1111 */
/* And then run the secure_client in another terminal session: */
/* secure_client 127.0.0.1 9999 */
/* Note - This is using the external_gateway module for controlling */
/* network access and the move_file module for controlling file access. */
/*
/* The [in] command line option writes the buffer to a file named */
/* Message-<message_number> in the in_path directory. If the server is */
/* restarted, then the <message_number> just starts from 1 again */
/* These files will be removed by the Message Filter move_file */
/* application and moved to the out_path directory. */
/*-----*/
/* To read messages from the Message Filter's OUT queue (after they have */
/* been move by the move_file application): */
/* runcon -t int_gateway_t -r message_filter_r secure_server 9999 */
/*
/* And then run the secure_client in another terminal session: */
/* runcon -t int_gateway_t -r message_filter_r secure_client \ */
/* 127.0.0.1 9999 */
/*
/* Note - This is using the internal_gateway module for controlling */
/* network access and the move_file module for controlling file access. */
/*
/* The [out] command line option reads files from the out_path directory */
/* and sends them to the client. The file is then unlinked. */
/*
/*****
/*
/* NOTE: unconfined_t is not allowed to read/write files in the [in] or */
/* [out] directories, if it tries, the context is displayed and the */
/* server will exit (e.g 'server 1234 in' and 'client 127.0.0.1 1234'). */
/*
/* Note when tested, the fopen function call on the [in] queue processing */
/* caused a segmentation fault (a feature ??). The only way found to stop */
/* this was to add an 'opendir' function call to the code, the server can */
```

```
/* then exit gracefully displaying the context. */
/* */
/*****/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/param.h>
#include <sys/types.h>
#include <sys/param.h>
#include <dirent.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <selinux/selinux.h>

#define MAXBUFFERSIZE 256

// variable to store current path
char in_path[] = "/usr/message_queue/in_queue";
char out_path[] = "/usr/message_queue/out_queue";

int main(int argc, char *argv [])
{
    short server_port;
    int count, i, rc, sock_fd, new_sock_fd, message_number, option;
    struct sockaddr_in server_addr;
    struct sockaddr_in client_addr;
    socklen_t sin_size;
    char buffer [MAXBUFFERSIZE];
    security_context_t context, peer_context, dir_context;
    char *peer_context_str;
    FILE *fp;
    char file_name [MAXPATHLEN];
    char in[] = "in";
    char out[] = "out";
    DIR *dp;
    struct dirent *ep;

    if (argc < 2) {
        fprintf (stderr, "Usage: %s <port>\n", argv[0]);
        exit (1);
    }

    if ((server_port = atoi (argv [1])) == 0) {
        fprintf (stderr, "Usage: %s <port>\n", argv[0]);
        exit (1);
    }

    option = 0; // Set to default (i.e. no in or out queue parameter)

    // Display default message about port, but alter if other options selected.
    sprintf (buffer, "Listening on port %d", server_port);
    if (argc == 3) {
        if (strcmp(argv [2], in) == 0) {
            option = 1;
            sprintf (buffer, "Listening on port %d. Information sent to client
will be written to files in %s", server_port, in_path);
        }
        else if (strcmp(argv [2], out) == 0) {
            option = 2;
            sprintf (buffer, "Listening on port %d. Files will be read from %s
and the contents sent to the client", server_port, out_path);
        }
        else {
            fprintf (stderr, "Usage: %s <port> [in | out]\n", argv[0]);
            exit (1);
        }
    }

    printf ("%s\n", buffer);

    if ((sock_fd = socket (PF_INET, SOCK_STREAM, 0)) == -1) {
        perror ("Server socket");
        exit (1);
    }
}
```

```
}

// Set the message number to 1 so a "Message[message_number] is generated.
message_number = 1;

bzero((char *) &server_addr, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(server_port);
server_addr.sin_addr.s_addr = INADDR_ANY;

if (bind (sock_fd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr))
== -1) {
    perror ("Server bind");
    exit (1);
}

if (listen (sock_fd, 5) == -1) {
    perror ("Server listen");
    exit (1);
}

while (1) {
    sin_size = sizeof(struct sockaddr_in);
    if ((new_sock_fd = accept (sock_fd, (struct sockaddr *)&client_addr,
&sin_size)) == -1) {
        perror ("Server accept");
        continue;
    }

    // Get Server context information
    if (rc = getcon (&context) < 0) {
        perror ("Server context");
        exit (1);
    }

    if (rc = getpeercon (new_sock_fd, &peer_context) < 0)
        peer_context_str = strdup ("No Peer Context Available");
    else {
        peer_context_str = strdup (peer_context);
        freecon (peer_context);
    }
    // Clear the buffer of rubbish
    memset (buffer, 0, sizeof(buffer));

    switch (option) {
    case 1:
        // This option sends the buffer to client,
        // and then writes it to a file in the in_queue
        // Make up a file name
        sprintf (file_name, "Message-%d", message_number);

        // Build buffer with Message Number at start.
        // The Message number will also be the file name
        sprintf (buffer, "This is %s from the server listening on port:
%d \nClient source port: %d \nServer Context: %s \nServer Peer Context:
%s \n", file_name, ntohs (server_addr.sin_port), ntohs
(client_addr.sin_port), context, peer_context_str);

        if (send (new_sock_fd, buffer, strlen (buffer), 0) == -1)
            perror ("Server send");
        // Now write buffer to file as well

        // This opendir has been put here as get Segmentation
        // fault if just do the fopen and its
        // unconfined_t trying to write a file here
        if ((dp = opendir (in_path)) == 0) {
            // Could be that unconfined_t is trying this, if so exit showing
context:
            getcon (&dir_context);
            printf ("Open Directory error %s Context is: %s\n", in_path,
dir_context);
            exit (1);
        }
        closedir (dp);
    }
}
```



```
// Make up full path + file name
sprintf (file_name,"%s/Message-%d", in_path, message_number);
if ((fp = fopen (file_name, "w")) == 0) {
    perror (fp);
    exit (0);
}

count = strlen (buffer);
if (fwrite (buffer, count, 1, fp) != 1) {
    perror (fp);
    exit (0);
}
fclose (fp);
break;

case 2:
// This option will read a file from the out_queue,
// send it to the client and then delete it.
if ((dp = opendir (out_path)) == 0) {
// Could be that unconfined_t is trying this on insecure
// channel ?? If so then exit showing context:
getcon (&dir_context);
printf ("Open Directory error %s Context is: %s\n", out_path,
dir_context);
exit (1);
}

do {
if ((ep = readdir (dp)) == 0) {
    sprintf (buffer, "Server has no files to send\n");
    break;
}
} while ((strcmp(ep->d_name, ".") == 0) || (strcmp(ep->d_name,
"..") == 0));

if (ep != 0) { // There is a file if ep != 0, otherwise send note to
client saying no more files
    sprintf (file_name,"%s/%s", out_path, ep->d_name);

    if ((fp = fopen (file_name, "r")) == 0) {
        perror (fp);
        exit (0);
    }

    // Read Contents of File
    if (fread (buffer, sizeof (buffer), 1, fp) != 0) {
        perror (fp);
        exit (0);
    }
    unlink (file_name);
    fclose (fp);
    closedir (dp);
}

// Now send the buffer to client
count = strlen (buffer);
if (send (new_sock_fd, buffer, count, 0) == -1)
    perror ("Server send");
break;

default: // There is no in_que or out_que parameter so just send the
buffer.
    // Make up a Message name
    sprintf (file_name,"Message-%d", message_number);

    // Print Server network information
    printf ("Server has connection from client: host = %s destination
port = %d source port = %d\n",
inet_ntoa(client_addr.sin_addr), ntohs (server_addr.sin_port), ntohs
(client_addr.sin_port));

    printf ("Server Context: %s \nServer Peer Context: %s \n", context,
peer_context_str);
```

```
        sprintf (buffer, "This is %s from the server listening on port:
%d \nClient source port: %d \nServer Context: %s \nServer Peer Context:
%s \n", file_name, ntohs (server_addr.sin_port), ntohs
(client_addr.sin_port), context, peer_context_str);

        if (send (new_sock_fd, buffer, strlen (buffer), 0) == -1)
            perror ("Server send");
    }
    message_number++;
    freecon (context);
    close (new_sock_fd);
}
return 0;
}
```

11. Compile two versions of the server by running:

```
gcc -o secure_server server.c -lselinux
gcc -o server server.c -lselinux
```

12. Move the binaries to /usr/local/bin:

```
cp client /usr/local/bin
cp secure_client /usr/local/bin
cp server /usr/local/bin
cp secure_server /usr/local/bin
```

13. Produce a script called iptables_secmark with the contents shown below. This will be used to load the iptables (notes: 1. that if the current mangle table has other entries, then they will be lost as this script flushes the table before loading the new contents. 2. The entries for the internal gateway are commented out. This is because the module has not been built yet and leaving these in would produce an error when loading the table with SELinux in enforcing mode).

```
# Flush the mangle table first:
iptables -t mangle -F

#----- INPUT IP Stream -----#

# This INPUT rule sets all packets to default_secmark_packet_t: as it is
# executed first:
iptables -t mangle -A INPUT -i lo -p tcp -d 127.0.0.0/8 -j SECMARK --selctx
system_u:object_r:default_secmark_packet_t

# These rules that will replace the above context with the internal or
# external gateway if port 9999 or 1111 is found in either the source or
# destination port of the packet:
iptables -t mangle -A INPUT -i lo -p tcp --dport 9999 -j SECMARK --selctx
system_u:object_r:ext_gateway_packet_t
iptables -t mangle -A INPUT -i lo -p tcp --sport 9999 -j SECMARK --selctx
system_u:object_r:ext_gateway_packet_t
#
# These are not required until using the internal gateway:
#iptables -t mangle -A INPUT -i lo -p tcp --dport 1111 -j SECMARK --selctx
system_u:object_r:int_gateway_packet_t
#iptables -t mangle -A INPUT -i lo -p tcp --sport 1111 -j SECMARK --selctx
system_u:object_r:int_gateway_packet_t

iptables -t mangle -A INPUT -m state --state ESTABLISHED,RELATED -j
CONNSECMARK --save

#----- OUTPUT IP Stream -----#
```

```
# This OUTPUT rule sets all packets to default_secmark_packet_t: as it is
# executed first:
iptables -t mangle -A OUTPUT -o lo -p tcp -d 127.0.0.0/8 -j SECMARK --selctx
system_u:object_r:default_secmark_packet_t

# These rules that will replace the above context with the internal or
# external gateway if port 9999 or 1111 is found in either the source or
# destination port of the packet:

iptables -t mangle -A OUTPUT -o lo -p tcp --dport 9999 -j SECMARK --selctx
system_u:object_r:ext_gateway_packet_t
iptables -t mangle -A OUTPUT -o lo -p tcp --sport 9999 -j SECMARK --selctx
system_u:object_r:ext_gateway_packet_t
#
# These are not required until using the internal gateway:
#iptables -t mangle -A OUTPUT -o lo -p tcp --dport 1111 -j SECMARK --selctx
system_u:object_r:int_gateway_packet_t
#iptables -t mangle -A OUTPUT -o lo -p tcp --sport 1111 -j SECMARK --selctx
system_u:object_r:int_gateway_packet_t

iptables -t mangle -A OUTPUT -m state --state ESTABLISHED,RELATED -j
CONNSECMARK --save

iptables -t mangle -L
```

14. Produce a `restorefiles_gateway` file with the contents shown below. This will be used by the `restorecon` command to relabel the SECMARK test client / server executables after compilation and if any updates are done later.

```
/usr/local/bin/secure_client
/usr/local/bin/secure_server
/usr/local/bin/client
/usr/local/bin/server
```

15. Run the `restorecon(8)` command to relabel the secure versions of the client / server as follows:

```
restorecon -f restorefiles_gateway
```

16. Check that the secure versions of the client / server are labeled correctly using `ls -Z /usr/local/bin`.

```
user_u:object_r:unconfined_t          client
user_u:object_r:secure_services_exec_t secure_client
user_u:object_r:secure_services_exec_t secure_server
user_u:object_r:unconfined_t          server
```

17. Add the `message_filter_r` role by running `semanage` as follows:

```
semanage user -m -R "message_filter_r unconfined_r" user_u
```

Note: Need to add the `unconfined_r` role as `semanage` will remove it from the current policy otherwise, causing much grief (a bug or feature ??). See the [Using sediffx](#) section for more information.

The installation process is now complete, the testing is discussed in the next section.

5.6.1 Testing the Module

To test the SECMARK functionality it is recommended that three virtual terminal sessions are opened (as shown in [Figure 5-23](#)) for:

1. Running clients as they will display status messages if successful.
2. Running the servers as they display messages when connections are made with the clients.
3. Viewing the audit log file. Note that the module has `auditallow` rules on `packet { send recv }` so that these events can be seen.

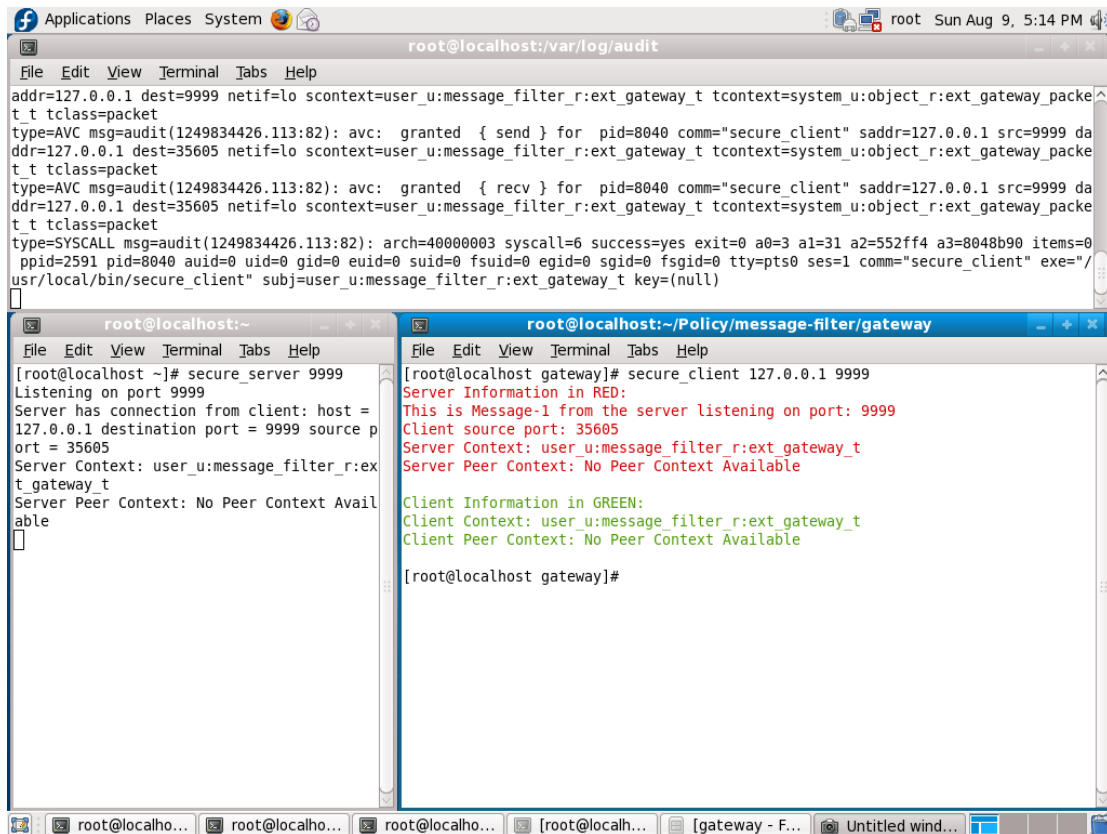


Figure 5-23: Testing using three virtual terminal sessions

5.6.1.1 Running the Tests

It is assumed that there are three terminal sessions logged in as root as shown in [Figure 5-23](#), with the client and server windows both at the directory with the executable `secmark` code and scripts, and the third window for tailing the `audit.log` file.

Before starting the tests:

1. In the window that will display the audit log, execute the following command:

```
tail -f /var/log/audit/audit.log.
```

2. In a window run the following command to load the `iptables`:

```
./iptables_secmark
```

Note that it is important to load the iptables as explained in the [Importance of Loading the iptables](#) section below.

3. In a window run the following command to start enforcing policy:

```
setenforce 1
```

Note that the server must be started before the client. To exit any of the server sessions press `ctrl/c`.

Test 1 – Running secure server and secure client sessions on port 9999 using the loopback interface (127.0.0.1):

1. In a window run the following command to start the secure server:

```
secure_server 9999
```

2. In a window run the following command to start the secure client:

```
secure_client 127.0.0.1 9999
```

The `audit.log` should contain only granted events on transition, send and recv (note that the transition also transitioned the role to `message_filter_r`):

```
type=AVC msg=audit(1249742538.972:23): avc: granted { transition } for pid=2905
comm="bash" path="/usr/local/bin/secure_server" dev=dm-0 ino=355514
scontext=user_u:unconfined_r:unconfined_t
tcontext=user_u:message_filter_r:ext_gateway_t tclass=process
type=SYSCALL msg=audit(1249742538.972:23): arch=40000003 syscall=11 success=yes
exit=0 a0=858a678 a1=85932c0 a2=858c8e8 a3=0 items=0 ppid=2520 pid=2905 auid=0
uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts1 ses=1
comm="secure_server" exe="/usr/local/bin/secure_server"
subj=user_u:message_filter_r:ext_gateway_t key=(null)

type=AVC msg=audit(1249742544.827:24): avc: granted { transition } for pid=2907
comm="bash" path="/usr/local/bin/secure_client" dev=dm-0 ino=354307
scontext=user_u:unconfined_r:unconfined_t
tcontext=user_u:message_filter_r:ext_gateway_t tclass=process
type=SYSCALL msg=audit(1249742544.827:24): arch=40000003 syscall=11 success=yes
exit=0 a0=87f92d8 a1=87e9ca8 a2=87ee8e8 a3=0 items=0 ppid=2496 pid=2907 auid=0
uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=1
comm="secure_client" exe="/usr/local/bin/secure_client"
subj=user_u:message_filter_r:ext_gateway_t key=(null)

type=AVC msg=audit(1249742544.833:25): avc: granted { send } for pid=2907
comm="secure_client" saddr=127.0.0.1 src=43592 daddr=127.0.0.1 dest=9999 netif=lo
scontext=user_u:message_filter_r:ext_gateway_t
tcontext=system_u:object_r:ext_gateway_packet_t tclass=packet

type=AVC msg=audit(1249742544.833:25): avc: granted { recv } for pid=2907
comm="secure_client" saddr=127.0.0.1 src=43592 daddr=127.0.0.1 dest=9999 netif=lo
scontext=user_u:message_filter_r:ext_gateway_t
tcontext=system_u:object_r:ext_gateway_packet_t tclass=packet
....
....
type=AVC msg=audit(1249742544.834:26): avc: granted { send } for pid=2905
comm="secure_server" saddr=127.0.0.1 src=9999 daddr=127.0.0.1 dest=43592 netif=lo
scontext=user_u:message_filter_r:ext_gateway_t
tcontext=system_u:object_r:ext_gateway_packet_t tclass=packet

type=AVC msg=audit(1249742544.834:26): avc: granted { recv } for pid=2905
comm="secure_server" saddr=127.0.0.1 src=9999 daddr=127.0.0.1 dest=43592 netif=lo
scontext=user_u:message_filter_r:ext_gateway_t
tcontext=system_u:object_r:ext_gateway_packet_t tclass=packet
```

Test 2 – Running the server on port 9999 and the secure client on port 1234 using the loopback interface:

1. In a window run the following command to start the server:

```
server 9999
```

2. In a window run the following command to start the secure client:

```
secure_client 127.0.0.1 1234  
Note: ctrl/c will exit the session
```

There should be an AVC audit message where the secure client is granted the transition but denied the send:

```
# Note that the client is allowed to transition:  
type=AVC msg=audit(1249742696.572:30): avc: granted { transition } for pid=2944  
comm="bash" path="/usr/local/bin/secure_client" dev=dm-0 ino=354307  
scontext=user_u:unconfined_r:unconfined_t  
tcontext=user_u:message_filter_r:ext_gateway_t tclass=process  
type=SYSCALL msg=audit(1249742696.572:30): arch=40000003 syscall=11 success=yes  
exit=0 a0=87f92d8 a1=87f5300 a2=87ee8e8 a3=0 items=0 ppid=2496 pid=2944 auid=0  
uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=1  
comm="secure_client" exe="/usr/local/bin/secure_client"  
subj=user_u:message_filter_r:ext_gateway_t key=(null)  
  
# But is not allowed to send message to the server as the packet  
# is marked default_secmark_packet_t:  
type=AVC msg=audit(1249742696.579:31): avc: denied { send } for pid=2944  
comm="secure_client" saddr=127.0.0.1 src=42942 daddr=127.0.0.1 dest=1234 netif=lo  
scontext=user_u:message_filter_r:ext_gateway_t  
tcontext=system_u:object_r:default_secmark_packet_t tclass=packet
```

Test 3 – Running both client and server sessions using port 1234 on the loopback interface (127.0.0.1):

1. In a window run the following command to start the server:

```
server 1234
```

2. In a window run the following command to start the client:

```
client 127.0.0.1 1234
```

The audit.log should contain only granted events on send and recv (note that there is NO transition and the role remains as unconfined_r):

```
type=AVC msg=audit(1249742778.361:34): avc: granted { send } for pid=2964  
comm="client" saddr=127.0.0.1 src=42943 daddr=127.0.0.1 dest=1234 netif=lo  
scontext=user_u:unconfined_r:unconfined_t  
tcontext=system_u:object_r:default_secmark_packet_t tclass=packet  
  
type=AVC msg=audit(1249742778.361:34): avc: granted { recv } for pid=2964  
comm="client" saddr=127.0.0.1 src=42943 daddr=127.0.0.1 dest=1234 netif=lo  
scontext=user_u:unconfined_r:unconfined_t  
tcontext=system_u:object_r:default_secmark_packet_t tclass=packet  
....  
....  
type=AVC msg=audit(1249742778.362:35): avc: granted { send } for pid=2961  
comm="server" saddr=127.0.0.1 src=1234 daddr=127.0.0.1 dest=42943 netif=lo
```

```
scontext=user_u:unconfined_r:unconfined_t
tcontext=system_u:object_r:default_secmark_packet_t tclass=packet

type=AVC msg=audit(1249742778.362:35): avc: granted { recv } for pid=2961
comm="server" saddr=127.0.0.1 src=1234 daddr=127.0.0.1 dest=42943 netif=lo
scontext=user_u:unconfined_r:unconfined_t
tcontext=system_u:object_r:default_secmark_packet_t tclass=packet
```

Test 4 – Running the server on port 9999 and the secure client on port 9999 using the loopback interface:

3. In a window run the following command to start the server:

```
server 9999
```

4. In a window run the following command to start the secure client:

```
secure_client 127.0.0.1 9999

Note: ctrl/c will exit the session
```

The AVC audit messages show that the secure client has been granted the transition and send but denied the `recv` from the standard server (but note that the server was allowed to accept the connection):

```
type=AVC msg=audit(1249742873.035:38): avc: granted { transition } for pid=2987
comm="bash" path="/usr/local/bin/secure_client" dev=dm-0 ino=354307
scontext=user_u:unconfined_r:unconfined_t
tcontext=user_u:message_filter_r:ext_gateway_t tclass=process
type=SYSCALL msg=audit(1249742873.035:38): arch=40000003 syscall=11 success=yes
exit=0 a0=8801cf0 a1=87e9ca8 a2=87ee8e8 a3=0 items=0 ppid=2496 pid=2987 auid=0
uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=1
comm="secure_client" exe="/usr/local/bin/secure_client"
subj=user_u:message_filter_r:ext_gateway_t key=(null)

type=AVC msg=audit(1249742873.041:39): avc: granted { send } for pid=2987
comm="secure_client" saddr=127.0.0.1 src=35900 daddr=127.0.0.1 dest=9999 netif=lo
scontext=user_u:message_filter_r:ext_gateway_t
tcontext=system_u:object_r:ext_gateway_packet_t tclass=packet

type=AVC msg=audit(1249742873.041:39): avc: denied { recv } for pid=2987
comm="secure_client" saddr=127.0.0.1 src=35900 daddr=127.0.0.1 dest=9999 netif=lo
scontext=user_u:unconfined_r:unconfined_t
tcontext=system_u:object_r:ext_gateway_packet_t tclass=packet
```

The reader can experiment with the remaining scenarios to find if there are any holes in the configuration.

5.6.2 Points to Note

5.6.2.1 Importance of Loading the `iptables`

The external gateway policy module relies on the fact that the `iptables` are loaded correctly to label the network packets. If they are not loaded, or (for example) the command:

```
iptables -t mangle -F
```

was allowed to be run that removes the mangle table entries, then the network packets would be labeled with the initial SID default of `unconfined_t`. The result is of

course that all packets would be allowed. For example, running the `secure_client` and `standard_server` on port 9999 with no `iptables` loaded would have the following `audit.log` entries (as all traffic on all ports would flow, as no policy is being enforced):

```
type=AVC msg=audit(1247241956.542:32): avc: granted { transition } for pid=2876
comm="bash" path="/usr/local/bin/secure_client" dev=dm-0 ino=354307
scontext=user_u:unconfined_r:unconfined_t
tcontext=user_u:message_filter_r:ext_gateway_t tclass=process
type=SYSCALL msg=audit(1247241956.542:32): arch=40000003 syscall=11 success=yes
exit=0 a0=9474a68 a1=947f460 a2=946d8e8 a3=0 items=0 ppid=2634 pid=2876 auid=0
uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=1
comm="secure_client" exe="/usr/local/bin/secure_client"
subj=user_u:message_filter_r:ext_gateway_t key=(null)
```

Compare this `audit.log` trail with those shown in [Test 4](#) that was run using the same scenario except that the `iptables` were loaded, thus denying the `recv`.

5.6.2.2 Running tests out of sequence

The server component allows files to be created in an ‘in queue’, and read / unlinked for the ‘out queue’ when running the message filter test. However should the [message filter](#) tests be run (see the [Testing the Message Filter Build](#) section) before the internal gateway and file mover loadable modules are loaded, the following will be noted:

1. When running the unconfined client / server, files can be written (`server 1234 in` with `client 127.0.0.1 1234`), moved (`move_file`) and then read / unlinked (`server 1234 out` with `client 127.0.0.1 1234`). This is because the base policy allows `unconfined_t` to do anything it likes.
2. When running the secure client / server, files cannot be written to the ‘in queue’ or read / unlinked from the ‘out queue’. This is because the `ext_gateway_t` process does not have the required `allow` permissions to do this.

5.7 Building the NetLabel Loadable Module

This simple module enables a NetLabel `netlabel_peer_t` label to be added to the network connection to show that additional information at the peer level (as `secmark` handles packet level labeling) can be added.

Because standard F-10 has the Policy Capabilities⁴⁹ `network_peer_controls` set to ‘0’, the full peer controls are not enabled, however the legacy implementation will use the `tcp_socket` object class with the `recvfrom` permission to manage peer labeling for this example.

For an example where the `network_peer_controls` is set to ‘1’, allowing the use of the new controls see [Appendix E – NetLabel Module Support for network_peer_controls](#).

The following steps need to be followed to build the test services (it is assumed that the files are built in the `./notebook-source/message_filter/netlabel` directory):

⁴⁹ See the [SELinux Filesystem](#) section.

1. Ensure you are logged on as 'root' and SELinux is running in permissive mode (`setenforce 0`) to perform the build process.
2. Download and install the NetLabel rpm as this is not included in the FC-10 build:

```
yum install netlabel_tools
# yum will then install netlabel_tools-0.18-1.fc10.i386.rpm
# or a later version.
```

3. Produce a `netlabel.conf` loadable module file with a text editor (such as `vi` or `gedit`) containing the contents shown below:

```
module netlabel 1.0.0;
#
#####
#
# This Loadable Module will allow the netlabels to be added and checked #
# within the client / server applications that form part of the SECMARK #
# test examples. #
#
# The following needs to happen to enable Netlabel to work as it is not #
# installed by default in F-10: #
#
# (1) Download and install netlabel_tools-018-1.fc10.i386.rpm (or equiv) #
#
# (2) Install this loadable module. #
#
# (3) Run the following netlabelctl command: #
#     netlabelctl unlbl add interface:lo address:127.0.0.1 \ #
#         label:system_u:object_r:netlabel_peer_t #
#
# (4) Run netlabelctl -p unlbl list command to check all is okay. #
#
# (5) Run the secure and standard client/server that should now display #
#     the netlabel_peer_t as the peer context. #
#
# Important note: F-10 does not support the latest netlabel services in #
# the kernel as: #
#     /selinux/policy_capabilities/network_peer_controls = 0 #
#
# The optional section is used when the internal gateway module is #
# loaded. #
#####
require {
    type ext_gateway_t, unconfined_t;
    class tcp_socket { recvfrom };
}
type netlabel_peer_t;
type socket_t;

# These are used when /selinux/policy_capabilities/network_peer_controls = 0
# which is the default for F-10
allow ext_gateway_t netlabel_peer_t : tcp_socket recvfrom;
allow unconfined_t netlabel_peer_t : tcp_socket recvfrom;

#
##### START OPTIONAL SECTION #####
#
optional {
    require {
        # This is defined in the int_gateway.conf module:
        type int_gateway_t;
    }
    allow int_gateway_t netlabel_peer_t : tcp_socket recvfrom;
}
#
##### END OPTIONAL SECTION #####
```

```
#
```

4. Compile and install the module as follows:

```
checkmodule -m netlabel.conf -o netlabel.mod
semodule_package -o netlabel.pp -m netlabel.mod
semodule -v -s modular-test -i netlabel.pp
```

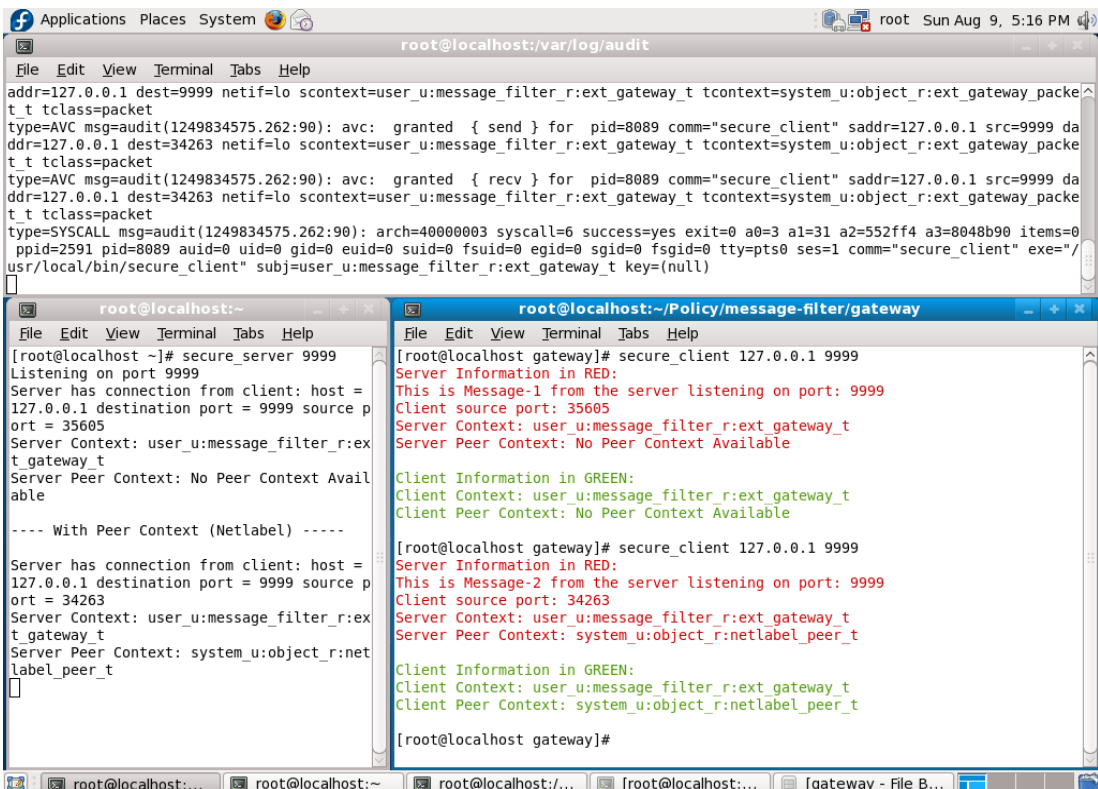
5. Run the following command to add the `netlabel_peer_t` label as follows:

```
netlabelctl unlbl add interface:lo address:127.0.0.1 \
    label:system_u:object_r:netlabel_peer_t
```

6. Run enforcing mode:

```
setenforce 1
```

7. Run either the client / server or `secure_client` / `secure_server` applications as shown in the [SECMARK tests](#). There should now be a peer context displayed as shown in [Figure 5-24](#).



The screenshot shows a terminal window with three panes. The top pane displays SELinux audit logs for network traffic and system calls. The bottom-left pane shows the output of the `secure_server` command, which is listening on port 9999 and has received two connections from clients at 127.0.0.1. The bottom-right pane shows the output of the `secure_client` command, which is connecting to the server. The output shows that the server context is `user_u:message_filter_r:ext_gateway_t` and the client context is `user_u:message_filter_r:ext_gateway_t`. The server peer context is `system_u:object_r:netlabel_peer_t` and the client peer context is `system_u:object_r:netlabel_peer_t`.

Figure 5-24: Running the secure client / server with NetLabel enabled

To remove the NetFilter label, the following command can be run:

```
netlabelctl unlbl del interface:lo address:127.0.0.1 \
    label:system_u:object_r:netlabel_peer_t
```

5.8 Building the Message Filter Service

To complete the overall message filter shown in [Figure 5-21](#), the internal gateway and file mover applications and policy modules need to be built. These are explained in this section plus how to test the modules via simple helper scripts. The source and scripts are included in the source code rpm package.

The following will be built in this section:

1. The internal gateway policy module.
2. The file mover application.
3. The file mover policy module.

5.8.1 Internal Gateway Loadable Policy Module

This loadable module will apply policy rules for the internal gateway. The policy applies `dontaudit` rules for those permissions known not to cause problems.

The following steps need to be followed to build the internal gateway module. It is assumed that the services are installed in `./notebook-source/message_filter/gateway`:

1. Ensure you are logged on as 'root' and SELinux is running in permissive mode (`setenforce 0`) to perform the build process.
2. Produce a `int_gateway.conf` file with a text editor (such as `vi` or `gedit`) containing the contents shown below:

```
module int_gateway 1.0.0;

#####
#
# This Loadable Module will allow a simple Message Filter to be tested.
#
# The module is used with the base.conf that sets up the unconfined_t
# space, the external_gateway.conf that manages the incoming data, and
# the move_file.conf module that copies files from the in queue to the
# out queue as explained in the SELinux Notebook.
# This module can be built by:
#   checkmodule -m int_gateway.conf -o int_gateway.mod
#   semodule_package -o int_gateway.pp -m int_gateway.mod
#   semodule -v -s modular-test -i int_gateway.pp
#
# The secure port for this internal gateway is 1111 and can only be
# read/write by the secure client / server. The external gateway will
# use port 9999 and can only be read/write by the secure client / server.
# Any other port can be read / write by the standard client / server.
#
# The iptables_secmark script can be modified to other ports if required.
# WARNING - If the iptables are not loaded to label packets, ports etc.
# then the standard client / server can use the secure ports.
#
# Run setenforce 1, the policy can be tested using combinations of:
#   ./server <port>
#   ./secure_server <port>
#
#   ./client <host> <port>
#   ./secure_client <host> <port>
#
# The module transitions to a role of message_filter_r simply to show
# a role transition. To add the role to user_u the semanage command is
# used as follows:
#   semanage user -m -R "message_filter_r unconfined_r" user_u
# Note: Need to put in the unconfined_r role as semanage will remove it
# from the current policy otherwise, causing much grief.
#
```

```
#
# Note: To run the internal gateway the runcon command must be used.
# This is because the external gateway has a type_transition statement:
#   type_transition unconfined_t secure_services_exec_t :
#     process ext_gateway_t;
# AND the module linker will not allow two type_transition statements
# using the secure_services_exec_t target with a different process type
# i.e. There cannot be in the overall policy:
#   type_transition unconfined_t secure_services_exec_t :
#     process ext_gateway_t;
#   type_transition unconfined_t secure_services_exec_t :
#     process int_gateway_t;
#
# Therefore run this as follows:
# runcon -t int_gateway_t -r message_filter_r secure_server 9999
# runcon -t int_gateway_t -r message_filter_r secure_client \
#                                     127.0.0.1 9999
#
#####

require {
    type unconfined_t, secure_services_exec_t;
    role unconfined_r;
    attribute message_filter_domains;
    class packet { send recv relabelto };
    class process { fork sigchld transition siginh rlimitinh noatsecure };
    class file { entrypoint read getattr execute relabelto unlink write
create };
    class filesystem { getattr associate };
    class chr_file { read write getattr };
    class dir { read search getattr write add_name remove_name };
    class fd use;
    class lnk_file read;
    class tcp_socket { write listen node_bind name_bind accept bind read
name_connect connect create getopt };
    class association recvfrom;
    class unix_stream_socket { create connect };
}

# The internal gateway will run in this domain:
type int_gateway_t;

# The internal gateway will have a SECMARK in the iptables of this label:
type int_gateway_packet_t;

# Add the gateway domain to the attribute:
typeattribute int_gateway_t message_filter_domains;

# Use message_filter_r role and role transition for the gateway:
role message_filter_r types int_gateway_t;
allow unconfined_r message_filter_r;
role_transition unconfined_r secure_services_exec_t message_filter_r;

# Allow unconfined_t to relabel the secure ports. This is needed so that
# iptables can be updated easily. Note: Against security policy, however
# these need to be loaded at boot time when the policy is in enforcing
# mode so no choice !!
allow unconfined_t int_gateway_packet_t : packet relabelto;

# Allow gateway access only to secure ports:
allow int_gateway_t int_gateway_packet_t : packet { send recv };

# Allow the internal gateway to transition to int_gateway_t. Note that the
# type_transition statement is commented out and runcon is used to
# transition this gateway (see the "Type Enforcement Rules" section of
# the SELinux Notebook for gory details):
allow unconfined_t secure_services_exec_t : file { read execute getattr };
allow unconfined_t int_gateway_t : process { transition };
allow int_gateway_t secure_services_exec_t : file { entrypoint };
#type_transition unconfined_t secure_services_exec_t : process
int_gateway_t;

# Stop segmentation faults
allow int_gateway_t unconfined_t : filesystem associate;
allow unconfined_t int_gateway_t : process noatsecure;
```

```
dontaudit unconfined_t int_gateway_t : process { siginh rlimitinh };

# Allow int_gateway_t access to areas under unconfined_t domain:
allow int_gateway_t unconfined_t : packet { recv send };
allow int_gateway_t unconfined_t : chr_file { read write getattr };
allow int_gateway_t unconfined_t : dir search;
allow int_gateway_t unconfined_t : fd use;
allow int_gateway_t unconfined_t : filesystem getattr;
allow int_gateway_t unconfined_t : tcp_socket name_connect;
allow int_gateway_t unconfined_t : association recvfrom;
dontaudit int_gateway_t unconfined_t : process sigchld;
allow int_gateway_t self : dir search;
allow int_gateway_t self : tcp_socket { read create connect };

# For client and server to access the shared libc:
allow int_gateway_t unconfined_t : file { read getattr execute };
dontaudit int_gateway_t unconfined_t : dir { getattr };
allow int_gateway_t unconfined_t : lnk_file read;

# Required if use host name instead of the IP address (e.g. localhost
# instead of 127.0.0.1) in the client command line:
dontaudit int_gateway_t self : unix_stream_socket { create connect };

# Required to get context information when using the libselinux api calls
# getcon() and getpeercon():
allow int_gateway_t self : file read;
allow int_gateway_t self : tcp_socket getopt;

# These entries are for the server only:
allow int_gateway_t self : tcp_socket { listen write accept bind };
allow int_gateway_t unconfined_t : tcp_socket { name_bind node_bind };
```

3. Compile the policy with `checkmodule` to produce an intermediate binary policy file:

```
checkmodule -m int_gateway.conf -o int_gateway.mod
```

The output from the compilation should be:

```
checkmodule: loading policy configuration from base.conf
checkmodule: policy configuration loaded
checkmodule: writing binary representation (version 8) to base.mod
```

4. Package the policy with `semodule_package`, this will produce a policy module file (note – if successful there are no output messages):

```
semodule_package -o int_gateway.pp -m int_gateway.mod
```

5. Install the loadable module with `semodule` (note – if successful there are no output messages):

```
semodule -v -s modular-test -i int_gateway.pp
```

6. If there are no errors reported, then the loadable module has been added to the policy store and loaded as a part of the policy. The policy module can be checked by:

```
semodule -s modular-test -l
```

The results should be:

```
ext_gateway 1.0.0
int_gateway 1.0.0
netlabel    1.0.0
```

The file mover application will now be built.

5.8.2 File Move Application

This 'C' program will move files from one directory to another and works in conjunction with the `move_file.conf` loadable module that will apply the policy rules.

The following steps need to be followed to build the file move application and it is assumed that the services are installed in `./notebook-source/message_filter/move_file`:

1. With an editor produce the `move_file.c` program as follows:

```
/*
 * *****
 */
/* This is the file mover component for the Notebook demo modular policy. */
/* It moves the files created as a part of the SECMARK tests from the */
/* /usr/message_queue/in_queue to the /usr/message_queue/out_queue. */
/* The move_file.conf module ensures that the output queue files are */
/* correctly labeled out_file_t by an object type_transition. */
/*
 * Copyright (C) 2009 Richard Haines
 */
/*
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 */
/*
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 */
/*
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */
/*
 * *****
 */
/*
 * The move_file program is compiled as:
 * gcc -o move_file move_file.c
 */
/*
 * The move_file application can optionally be called with a timer value
 * (in seconds) so that it can be run forever checking the in_queue
 * every X seconds: move_file [timer]
 */
/*
 * For the tests, the binary should be installed in /usr/local/bin and
 * then the mk-dir script run to create the following directories:
 * mkdir -p /usr/message_queue/in_queue
 * mkdir -p /usr/message_queue/out_queue
 */
/*
 * Install the move_file loadable module by:
 * checkmodule -m move_file.conf -o move_file.mod
 * semodule_package -o move_file.pp -m move_file.mod -f move_file.fc
 * semodule -v -s modular-test -i move_file.pp
 */
/*
 * Finally label the binary and message queue directories by:
 * restorecon -r -f restorecon_move_file
 */
/*
 * The server.c file describes how the files are are created etc.
 */
/*
 * *****
 */
#include <stdio.h>
```

```
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/dir.h>
#include <sys/param.h>

#define MAXBUFFERSIZE 256

#define FALSE 0
#define TRUE !FALSE

extern int alphasort();

// variable to store current path
char in_path[] = "/usr/message_queue/in_queue";
char out_path[] = "/usr/message_queue/out_queue";

main (int argc, char *argv [])
{
    FILE *fp1;
    FILE *fp2;
    char buffer [MAXBUFFERSIZE];
    char in_file_name [MAXPATHLEN];
    char out_file_name [MAXPATHLEN];
    int timer, count, length, i;
    struct direct **files;
    int select_file();

    // Use arg as the timer to use
    if (argc == 2)
        timer = atoi (argv [1]);
    else
        timer = 0;

    while (TRUE) {
        count = 0;
        count = scandir (in_path, &files, select_file, alphasort);
        printf ("Count = %d Timer = %d\n", count, timer);
        if ((count <= 0 && timer == 0))
            break;
        else
            sleep (timer);

        for (i=1; i<count+1; ++i) {
            // Build file name and clear the buffer
            sprintf (in_file_name,"%s/%s", in_path, files [i-1]->d_name);
            memset (buffer, 0, sizeof (buffer));

            // Open INQ file
            if ((fp1 = fopen (in_file_name, "r")) == 0) {
                perror (fp1);
                exit (1);
            }

            /* Read Contents of File */
            if (fread (buffer, sizeof (buffer), 1, fp1) != 0) {
                perror (fp1);
                exit (1);
            }

            // Build output file name
            sprintf (out_file_name,"%s/%s", out_path, files [i-1]->d_name);

            // Open OUTQ file
            if ((fp2 = fopen (out_file_name, "w")) == 0) {
                perror (fp2);
                exit (1);
            }

            // Get buffer length
            strcat (buffer, "(FILE MOVED TO OUT QUEUE)");
            length = strlen (buffer);
```

```
// Write Contents of File
if (fwrite (buffer, length, 1, fp2) != 1) {
    perror (fp2);
    exit (1);
}

unlink (in_file_name);
fclose (fp1);
fclose (fp2);
}
}
}

int select_file (struct direct *entry)
{
    if ((strcmp (entry->d_name, ".") == 0) || (strcmp (entry->d_name, "..") ==
0))
        return (FALSE);
    else
        return (TRUE);
}
```

2. Compile the `move_file.c` program:

```
gcc -o move_file move_file.c
```

3. Move the binary to `/usr/local/bin`:

```
mv move_file /usr/local/bin
```

To complete the message filter, the file mover loadable module will now be built.

5.8.3 File Mover Loadable Policy Module

This loadable module will allow a file to be moved from one directory to another using the file mover application built above with minimum privileges. The policy applies `dontaudit` rules for those permissions known not to cause problems.

Note that in the policy there is a statement that allows a counter to be displayed on the console for testing purposes.

The following steps need to be followed to build the file mover module and it is assumed that the services are installed in `./notebook-source/message_filter/move_file`:

1. Ensure you are logged on as 'root' and SELinux is running in permissive mode (`setenforce 0`) to perform the build process.
2. Produce a `move_file.conf` file with a text editor (such as `vi` or `gedit`) containing the contents shown below:

```
module move_file 1.0.0;

#####
#
# This Loadable Module will allow files to be moved from one directory #
# (or queue) to another using the move_file 'C' program with minimum #
# permissions. The policy applies dontaudit rules for those permissions #
# known not to cause problems. #
# #
#####
```



```
require {
    role unconfined_r;
    type unconfined_t;
    attribute message_filter_domains;
    class file { entrypoint getattr execute create read write unlink };
    class dir { read search getattr write add_name remove_name };
    class process { transition siginh noatsecure sigchld rlimitinh };
    class fd use;
    class chr_file { read write getattr };
    class lnk_file read;
    class filesystem associate;
}

# Define type identifiers for the process / domain:
type move_file_t;
typeattribute move_file_t message_filter_domains;

# Define the executable type:
type move_file_exec_t;

# These are the file directory types:
type in_queue_t;
type out_queue_t;

# These are the file types:
type in_file_t;
type out_file_t;

# Use message_filter_r role and then allow role transition
role message_filter_r types { move_file_t };
allow unconfined_r message_filter_r;
role_transition unconfined_r move_file_exec_t message_filter_r;

# Need permission for the program to transition:
allow unconfined_t move_file_t : process transition;
auditallow unconfined_t move_file_t : process transition;
allow unconfined_t move_file_exec_t : file { read execute getattr };
allow move_file_t move_file_exec_t : file { entrypoint };
type_transition unconfined_t move_file_exec_t : process move_file_t;

#
# The move_file application reads then deletes the file:
type_transition move_file_t in_queue_t : file in_file_t;
allow move_file_t in_file_t : file { read unlink };
allow move_file_t in_queue_t : dir { read getattr search write
remove_name };
dontaudit move_file_t in_file_t : file getattr;

# Need these if the files are labeled user_u:object_r:in_queue_t
# This happens if restorecond is not running with setenforce 0 and use
# vi to create the files for testing (as they are not relabeled)
# allow move_file_t in_queue_t:file { read getattr unlink };

# The move_file application then writes the file to the out queue:
type_transition move_file_t out_queue_t : file out_file_t;
allow move_file_t out_file_t : file { create write };
allow move_file_t out_queue_t : dir { search write add_name };
dontaudit move_file_t out_file_t : file getattr;

# Do not need these:
dontaudit move_file_t unconfined_t : process sigchld;
# HOWEVER - The move_file application has a printf with:
#     printf ("Count = %d Timer = %d\n", count, timer);
# that can be seen on the console IF this allow is enabled:
allow move_file_t unconfined_t : chr_file { read write getattr };
# OR it can be disabled from printing this on the console by:
# dontaudit move_file_t unconfined_t : chr_file { read write getattr };

# Need these as /usr/move_file dir is unconfined_t
allow move_file_t unconfined_t : dir search;
allow move_file_t unconfined_t : fd use;

# Need these to run libc.so shared library
dontaudit move_file_t unconfined_t : dir getattr;
allow move_file_t unconfined_t : lnk_file read;
```

```
allow move_file_t unconfined_t : file { read getattr execute };

# Need these to stop Segmentation faults
allow out_file_t unconfined_t : filesystem associate;
allow unconfined_t move_file_t : process noatsecure;
dontaudit unconfined_t move_file_t : process { siginh rlimitinh };

# Don't need these:
dontaudit unconfined_t in_queue_t : dir { read getattr search };
dontaudit unconfined_t out_file_t : file getattr;
dontaudit unconfined_t out_queue_t : dir { read getattr search };
dontaudit unconfined_t in_file_t : file getattr;
```

3. Produce a `move_file.fc` file (a segment that will be added to `file_contexts` file during the build) with the contents shown below. This will be used to relabel application files and directories.

```
# The Move File process makes use of two directory structures
# (in & out) that are labeled as follows:

/usr/message_queue/in_queue -d system_u:object_r:in_queue_t
/usr/message_queue/out_queue -d system_u:object_r:out_queue_t

# Ensure that any files are also relabeled:
/usr/message_queue/in_queue(/.*)? -- system_u:object_r:in_file_t
/usr/message_queue/out_queue(/.*)? -- system_u:object_r:out_file_t

# The Move File 'C' application is labeled:
/usr/local/bin/move_file -- system_u:object_r:move_file_exec_t
```

4. Produce a `restorecon_files` file with the contents shown below. This will be used by the `restorecon` command to relabel application files and directories after any updates.

```
/usr/message_queue/in_queue
/usr/message_queue/out_queue
/usr/local/bin/move_file
```

5. Compile the policy with `checkmodule` to produce an intermediate binary policy file:

```
checkmodule -m move_file.conf -o move_file.mod
```

The output from the compilation should be:

```
checkmodule: loading policy configuration from base.conf
checkmodule: policy configuration loaded
checkmodule: writing binary representation (version 8) to base.mod
```

6. Package the policy with `semodule_package`, this will produce a policy module file (note – if successful there are no output messages):

```
semodule_package -o move_file.pp -m move_file.mod -f move_file.fc
```

7. Make the directories required by the application. These need to be created because when `semodule` loads the policy, it will run `setfiles` to set the file contexts correctly (using the contents of the `move_file.fc` file produced in step 3).

```
mkdir -p /usr/message_queue/in_queue
mkdir -p /usr/message_queue/out_queue
```

8. Install the loadable module with `semodule` (note – if successful there are no output messages):

```
semodule -v -s modular-test -i move_file.pp
```

9. If there are no errors reported, then the loadable module has been added to the policy store and loaded as a part of the policy. The policy module can be checked by:

```
semodule -s modular-test -l
```

The results should be:

```
ext_gateway 1.0.0
int_gateway 1.0.0
move_file   1.0.0
netlabel    1.0.0
```

10. Uncomment the internal gateway entries in the `iptables` file (`./notebook-source/message_filter/gateways/iptables_secmark`) that was produced in step 13 of the [Building the SECMARK Test Loadable Module](#) section:

```
....
# These are not required until using the internal gateway:
iptables -t mangle -A INPUT -i lo -p tcp --dport 1111 -j SECMARK --selctx
system_u:object_r:int_gateway_packet_t
iptables -t mangle -A INPUT -i lo -p tcp --sport 1111 -j SECMARK --selctx
system_u:object_r:int_gateway_packet_t
....
....
#----- OUTPUT IP Stream -----#
....
#
# These are not required until using the internal gateway:
iptables -t mangle -A OUTPUT -o lo -p tcp --dport 1111 -j SECMARK --selctx
system_u:object_r:int_gateway_packet_t
iptables -t mangle -A OUTPUT -o lo -p tcp --sport 1111 -j SECMARK --selctx
system_u:object_r:int_gateway_packet_t
....
```

11. Ensure all the files are correctly labeled by running the `restorecon` command using the input file produced in step 4 above:

```
restorecon -r -f restorecon_file
```

12. Run enforcing mode:

```
setenforce 1
```

The message filter should now be ready to test.

5.8.4 Testing the Message Filter Build

To test the message filter it is recommended that four virtual terminal sessions are opened (as shown in [Figure 5-25](#)) for:

1. Running the external gateway client as it will display status messages if successful. This is shown on bottom left hand side using port 9999. Note that this process is run directly from the command line by `secure_server 9999` as it will automatically transition to the `ext_gateway_t` domain by the policy rules.
1. Running the internal gateway client as it will display status messages if successful.. This is shown on bottom right hand side using port 1111. Note that this process (and the secure server for the internal gateway) has to be run via the `runcon` command because of the type enforcement rules discussed in the [Type Enforcement Rules](#) section.
2. Running the servers as they display messages when connections are made with the clients.
3. Viewing the audit log file. Note that the module has `auditallow` rules on `packet { send recv }` so that these events can be seen. This is top left.
4. Starting and viewing the file mover application as this will be run to display a count of the files being moved. This is top right.

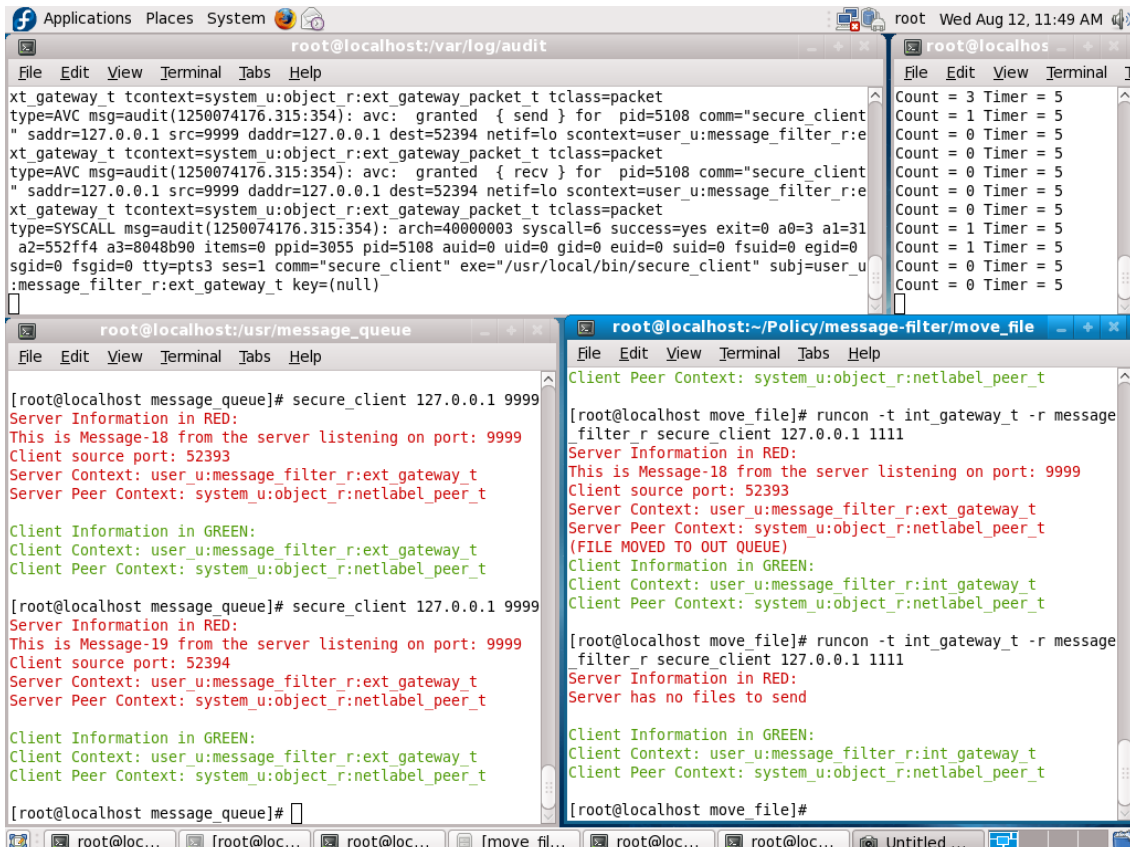


Figure 5-25: Testing the message filter service

If there are four terminal sessions logged in as root as shown in [Figure 5-25](#), then the follow commands will need to be executed to show the message filter is working:

1. In the session that will display the audit log, execute the following command:

```
tail -f /var/log/audit/audit.log.
```

2. In a session run the following command to load the iptables (it is assumed that the current directory is where the file is located):

```
./iptables_secmark
```

3. Each of the server processes for the gateways will be run in background using one of the sessions with the following commands:

```
# Start the external gateway in background with the 'in' argument
# so that files are created in the in_queue with the communications
# traffic:
secure_server 9999 in &
```

```
# Start the internal gateway in background using the runcon command
# with the 'out' argument so that files are read from the out_queue:
runcon -t int_gateway_t -r message_filter_r secure_server 1111 out &
```

4. In a session start the file mover application with a time in seconds argument so that it will loop and display the number of files moved:

```
move_file 5
```

5. In a session start the secure external gateway client:

```
secure_client 127.0.0.1 9999
```

6. In a session start the secure internal gateway client using the runcon command:

```
runcon -t int_gateway_t -r message_filter_r secure_client 127.0.0.1 1111
```

7. Keep repeating the client commands and the messages should be displayed in each window as the clients are run.

If the external gateway client is run a number of times, the messages will be read from the `in_queue` by the file mover and queued to the `out_queue`, the internal gateway client can then be run to read each message off the `out_queue`. The queues can be investigated for their context by using `ls -Z`, however to do this, enforcing mode must be off otherwise `unconfined_t` (that is the logon sessions domain) cannot read these areas.

6. Policy Investigation Tools

6.1 Introduction

This section describes the tools used to investigate the modular-test (base + ext_gateway + netlabel + int_gateway + move_file modules) policy for the message filter project during its development, debugging and validation.

Points to note:

1. When viewing a policy via investigation tools such as `apol`, the rules and statements that contain permissions, roles etc. have been resolved and will therefore not look the same as in the original source code. For example in the `ext_gateway.conf` module there are two separate but common rules (as they specify permissions required for that part of the policy):

```
# Allow the client/server to send/recv packets:
allow unconfined_t default_secmark_packet_t : packet { send recv };

# Required to allow the iptables to load as needs to relabel:
allow unconfined_t default_secmark_packet_t : packet relabelto;
```

When they are viewed via the investigation tools, the two rules would have been amalgamated and displayed as:

```
allow unconfined_t default_secmark_packet_t : packet { send recv relabelto };
```

2. When investigating loadable modules that have had additional configuration added via `semanage` (user, port etc.), then the binary policy file will contain this information. However when using the tools to view the packaged module source, these changes will not appear (see [sechecker](#) for an example).

6.2 Using `audit2allow` and `audit2why`

`audit2allow` is a very useful tool, however it needs to be used with caution as it gives permissions that are not always required; does not define any types; shows role transitions as process transitions; and has various other features.

The process used to debug most of the policy revolved around `audit2allow` and monitoring the audit log, updating the policy with the results and removing any permissions thought not to be required, testing the policy and then re-running `audit2allow` and so on.

There were times when `audit2allow` (or anything else for that matter) was of no help, particularly when the system hung during boot or at login time. It was just a case of keeping track of the changes, determining the differences and finding a fix.

`audit2why` is also useful, however it does not like AVC granted messages in the audit log.

6.3 Using seaudit and setroubleshoot

During the development these audit tools were initially used, however it was found that after a while it was easier to clear the audit log (>audit.log), then tail it (tail -f audit.log) and 'see' the problem flashing past, run audit2allow and then interpret the results.

6.4 Using sediffx

The sediff(1) (command line) and sediffx(1) (GUI) compares two policies and finds the differences between them. These are part of the SETools package and have extensive help (see the /usr/share/setools-3.3 directory) and man pages (man sediff and man sediffx) that should be read before using the tools.

The GUI version was used for testing the modular-test policy as the differences in two policies needed to be found for the following reason:

- When testing the ext_gateway module, a new role was created called message_filter_r. This role needed to be associated with a user and can be achieved by one of two ways:
 1. Add a [user statement](#) to the policy and associate the role. This worked with no problems, however as also experimenting with MLS policies it would mean having two gateway modules (as one user statement needs level and range, the other does not).
 2. Use semanage to associate the user to the new role. This did not work as expected using the following command:

```
semanage user -m -R "message_filter_r" user_u
```

The result was that the policy caused SELinux to lock the system so no login was possible, therefore the repair disk had to be used to change the policy, as the system had been configured with the save-previous = true set in the [/etc/selinux/semanage.conf file](#). Therefore the original (good) binary policy was available and copied over to the ./modular-test/policy directory, the system restarted, and the differences investigated.

To check the differences between the two policies, the original (good) and current (bad) were loaded as shown in [Figure 6-26](#):

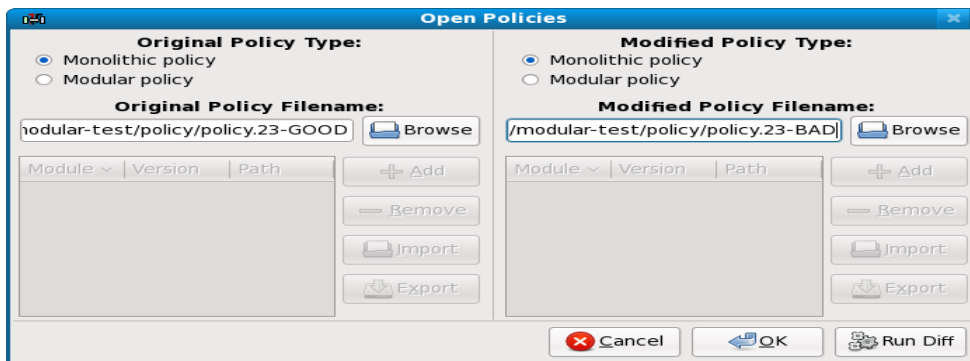


Figure 6-26: Opening the two policies in sediffx

The ‘Run Diff’ was run and [Figure 6-27](#) shows the differences between these two policies. As can be seen, the `unconfined_r` role has been removed by the `semanage` command:

```
semanage user -m -R "message_filter_r" user_u
```

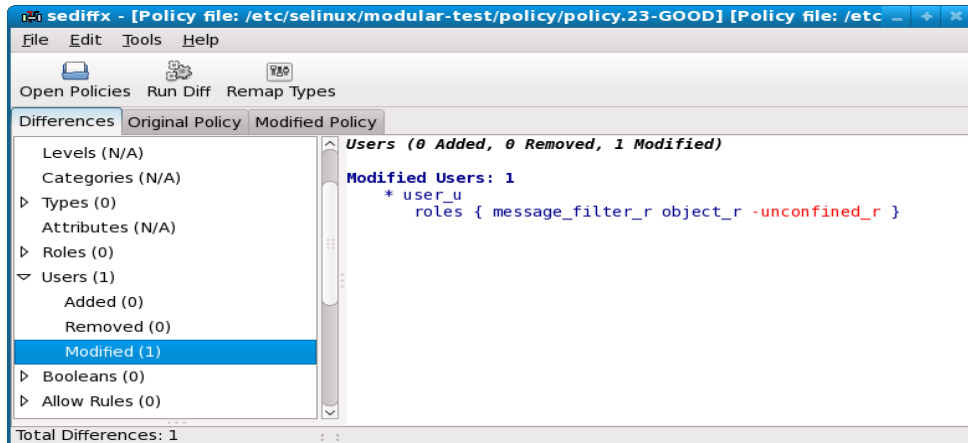


Figure 6-27: sediffx showing the differences in the two policies

The fix for this is to add all the roles when updating a user with `semanage` as follows:

```
semanage user -m -R "message_filter_r unconfined_r" user_u
```

6.5 Using sechecker

This command line application is part of the SETools package and is used to analyse a policy for various flaws. It has extensive help and man pages (`man sechecker`) that should be read before using the tool.

The `sechecker(8)` command has a set of pre-built modules⁵⁰ that can be run individually or from a profile containing a list of modules (a number of profiles are supplied – see the `/usr/share/setools-3.3` directory that also contains help files). Each of these modules will check for a specific set of flaws (e.g. find users without roles).

There are also ‘utility modules’ that find basic information (e.g. find domains) and are used by the modules when checking for flaws. Each module function is described in [Table 6-2](#) and [Table 6-3](#) along with comments on the test results for the `modular-test` policy. New modules can be written for `sechecker`, however the source code is required (that contains a module template source file to help with the development).

Note that some modules will work on packaged modules and source files only (as they have the attribute identifiers available). [Table 6-2](#) and [Table 6-3](#) has a column that specifies what type of policy (module, source or binary) each module supports.

⁵⁰ Not to be confused with the policy ‘loadable modules’.

6.5.1 Testing the Policy

For testing the modular-test policy, sechecker was run using the modular source⁵¹ policy with the modular-test.profile and using the binary policy with the modular-test-binary.profile. The two profiles are shown in [Table 6-1](#) (note that the modular-test.profile is in fact a copy of the all-checks-no-mls.sechecker that is supplied with sechecker).

The reason for running on both types of policy is to show the differences, as the module source does not contain the user association with the message_filter_r role, and the binary policy does not show that an attribute is not used by any rules.

modular-test.profile	modular-test-binary.profile
<pre> <sechecker version="1.1"> <profile> <module name="find_domains"> <output value="quiet"/> <option name="domain_attribute"> <item value="domain"/> </option> </module> <module name="find_file_types"> <output value="quiet"/> <option name="file_type_attribute"> <item value="file_type"/> </option> </module> <module name="domain_and_file"> <output value="short"/> </module> <module name="attribs_wo_types"> <output value="short"/> </module> <module name="roles_wo_types"> <output value="short"/> </module> <module name="users_wo_roles"> <output value="short"/> </module> <module name="roles_wo_allow"> <output value="short"/> </module> <module name="types_wo_allow"> <output value="short"/> </module> <module name="attribs_wo_rules"> <output value="short"/> </module> <module name="roles_wo_users"> <output value="short"/> </module> <module name="spurious_audit"> <output value="short"/> </module> <module name="inc_mount"> <output value="short"/> </module> <module name="domains_wo_roles"> <output value="short"/> </module> <module name="inc_dom_trans"> <output value="short"/> </module> <module name="find_net_domains"> <output value="quiet"/> <option name="net_obj"> <item value="netif"/> <item value="tcp_socket"/> <item value="udp_socket"/> <item value="node"/> <item value="association"/> </option> </module> <module name="find_port_types"> <output value="quiet"/> </module> <module name="find_node_types"> <output value="quiet"/> </module> <module name="find_netif_types"> <output value="quiet"/> </module> </pre>	<pre> <sechecker version="1.1"> <profile> <module name="roles_wo_types"> <output value="short"/> </module> <module name="users_wo_roles"> <output value="short"/> </module> <module name="roles_wo_allow"> <output value="short"/> </module> <module name="types_wo_allow"> <output value="short"/> </module> <module name="roles_wo_users"> <output value="short"/> </module> <module name="spurious_audit"> <output value="short"/> </module> <module name="inc_mount"> <output value="short"/> </module> <module name="inc_net_access"> <output value="short"/> </module> </profile> </sechecker> </pre>

⁵¹ Actually the packaged modules (base.pp, gateway.pp, netlabel.pp and move_file.pp.

```
<module name="inc_net_access">
  <output value="short"/>
</module>
<module name="unreachable_doms">
  <output value="short"/>
</module>
</profile>
</sechecker>
```

Table 6-1: sechecker profiles – *The profiles used to check the modular-test packages and the binary policy files.*

The sechecker commands were each run twice, once with `-v` (for verbose output that will detail any issues found in gory detail) and once without the `-v` option:

```
sechecker --fcfile=/etc/selinux/modular-test/contexts/files/file_contexts -p
modular-test.profile modular-test.list > modular-test-results.txt

sechecker --fcfile=/etc/selinux/modular-test/contexts/files/file_contexts -v -p
modular-test.profile modular-test.list > modular-test-verbose-results.txt

sechecker --fcfile=/etc/selinux/modular-test/contexts/files/file_contexts -p
modular-test-binary.profile /etc/selinux/modular-test/policy/policy.23 > modular-
test-binary-results.txt

sechecker --fcfile=/etc/selinux/modular-test/contexts/files/file_contexts -v -p
modular-test-binary.profile /etc/selinux/modular-test/policy/policy.23 > modular-
test-binary-verbose-results.txt
```

Note that the binary policy is referenced by its full path name, but the modular policy is referenced by a file called `modular-test.list`. The contents of this file is as follows, and can be built by the [apol tool](#) described later:

```
# modular-test lists the modules to be tested:
#
policy_list 1 modular
/etc/selinux/modular-test/modules/active/base.pp
/etc/selinux/modular-test/modules/active/modules/ext_gateway.pp
/etc/selinux/modular-test/modules/active/modules/int_gateway.pp
/etc/selinux/modular-test/modules/active/modules/move_file.pp
/etc/selinux/modular-test/modules/active/modules/netlabel.pp
```

6.5.2 The Results

The output from the `modular-test-binary.profile` without the `-v` option is shown below, however the main results are shown in:

- [Table 6-2](#) that describes the results for each module and the authors interpretation / action regarding any policy changes.
- [Table 6-3](#) that describes the results from the utility modules.

```
Module name: inc_mount Severity: med
This module finds domains that have incomplete mount permissions.
In order for a mount operation to be allowed by the policy the following rules
must be present:
  1) allow somedomain_d sometype_t : filesystem { mount };
  2) allow somedomain_d sometype_t : dir { mounton };

This module finds domains that have only one of the rules listed above.

Found 0 types.
-----
Module name: inc_net_access Severity: med
This module finds all network domains in a policy which do not have the
required permissions needed to facilitate network communication. For network
domains to communicate, the following conditions must be true:
```

- 1) the domain must have read or receive permissions on a socket of the same type
- 2) the domain must have send or receive permissions on an IPsec association (see `find_assoc_types`)
- 3) the domain must have send or receive permissions on netif objects for a netif type (see `find_netif_types`)
- 4) the domain must have send or receive permissions on node objects for a node type (see `find_node_types`)
- 5) the domain must have send or receive permissions on port objects for a port type (see `find_port_types`)

Found 3 network domains with insufficient permissions.

`int_gateway_t, ext_gateway_t, unconfined_t`

Module name: roles_wo_allow Severity: low

This module finds roles defined in the policy that are not used in any role allow rules. It is not possible to transition to or from any role that does not have any role allow rules.

Found 0 roles.

Module name: roles_wo_types Severity: low

This module finds roles in the policy that have no types. A role with no types cannot form a valid context.

Found 0 roles.

Module name: roles_wo_users Severity: low

This module finds roles that are not assigned to users. If a role is not assigned to a user it cannot form a valid context.

Found 0 roles.

Module name: spurious_audit Severity: low

This module finds audit rules in the policy which do not affect the auditing of the policy. This could happen in the following situations:

- 1) there is an allow rule with the same key and permissions for a dontaudit rule
- 2) there is an auditallow rule without an allow rule with the same key or with permissions that do not appear in an allow rule with the same key.

Found 1 rules.

`dontaudit ext_gateway_t unconfined_t : filesystem getattr ;`

Module name: types_wo_allow Severity: low

This module finds types defined in the policy that are not used in any allow rules. A type that is never granted an allow rule in the policy is a dead type. This means that all attempted access to the type will be denied including attempts to relabel to a (usable) type. The type may need to be removed from the policy or some intended access should be granted to the type.

Found 1 types.

`socket_t`

Module name: users_wo_roles Severity: low

This module finds all the SELinux users in the policy that have no associated roles. Users without roles may appear in the label of a file system object; however, these users cannot login to the system or run any process. Since these users cannot be used on the system, a policy change is recommended to remove the users or provide some intended access.

Found 0 users.

The SELinux Notebook - The Foundations

<i>Module Name</i>	<i>Module Description</i>	<i>Valid for Binary, Module or Source files</i>	<i>Comments on running sechecker on the modular-test policy with -v (verbose) option</i>
attrs_wo_rules	This module finds attributes in the policy that are not used in any rules; These attributes will get thrown out by the compiler and have no effect on the security environment. They are unnecessary and should be removed.	Modules and Source only	This module found an attribute called <code>message_filter_domains</code> that is not used (it was added to the modules and had the domain types added). Decision: The attribute can be removed from the policy.
attrs_wo_types	This module finds attributes in the policy that are not associated with any types. Attributes without types can cause type fields in rules to expand to empty sets and thus become unreachable. This makes for misleading policy source files.	Modules and Source only	This module did not find any attributes without types.
domain_and_file	This module finds all types in the policy treated as both a domain and a file type. See <code>find_domains</code> and <code>find_file_types</code> modules for details about the heuristics used to determine these types. It is considered bad security practice to use the same type for a domain and its data objects because it requires that less restrictive access be granted to these types.	Modules and Source only	This module found three types associated to domains and files (<code>unconfined_t</code> , <code>ext_gateway</code> and <code>int_gateway_t</code>). This probably occurred in the policy because the base is all <code>unconfined_t</code> . Decision: Without building a more complex policy it is thought that this is an acceptable risk.
domains_wo_roles	This module finds all domains in the policy not associated with a role. These domains cannot have a valid security context. The <code>object_r</code> role is not considered in this check.	Modules and Source only	This module did not find any domains without roles.
imp_range_trans	This module finds impossible range transitions in a policy. A range transition is possible if and only if all of the following conditions are satisfied: 1) there exist TE rules allowing the range transition to occur. 2) there exist RBAC rules allowing the range transition to occur. 3) at least one user must be able to transition to the target MLS range.	Binary, Modules and Source	As the modular-test policy is not MLS, then this was not run.

The SELinux Notebook - The Foundations

<i>Module Name</i>	<i>Module Description</i>	<i>Valid for Binary, Module or Source files</i>	<i>Comments on running sechecker on the modular-test policy with -v (verbose) option</i>
inc_dom_trans	<p>This module finds potential domain transitions missing key permissions. A valid domain transition requires the following:</p> <ol style="list-style-type: none"> 1) the starting domain can transition to the end domain for class process. 2) the end domain has some type as an entrypoint. 3) the starting domain can execute that entrypoint type. 4) (optional) a type_transition rule specifying these three types. 	Modules and Source only	This module did not find any incomplete domain transitions.
inc_mount	<p>This module finds domains that have incomplete mount permissions. In order for a mount operation to be allowed by the policy the following rules must be present:</p> <ol style="list-style-type: none"> 1) allow somedomain_d sometype_t : filesystem { mount }; 2) allow somedomain_d sometype_t : dir { mounon }; <p>This module finds domains that have only one of the rules listed above.</p>	Binary, Modules and Source	This module did not find any domains with incomplete mount permissions.
inc_net_access	<p>This module finds all network domains in a policy which do not have the required permissions needed to facilitate network communication. For network domains to communicate, the following conditions must be true:</p> <ol style="list-style-type: none"> 1) the domain must have read or receive permissions on a socket of the same type. 2) the domain must have send or receive permissions on an IPsec association (see find_assoc_types). 3) the domain must have send or receive permissions on netif objects for a netif type (see find_netif_types). 4) the domain must have send or receive permissions on node objects for a node type (see find_node_types). 5) the domain must have send or receive permissions on port objects for a port type (see find_port_types). 	Binary, Modules and Source	<p>This module found three network domains with insufficient permissions (unconfined_t, ext_gateway_t and int_gateway_t).</p> <p>Decision: As the policy modules were built for minimum privilege, adding additional (and not required) permissions would add no value to the policy.</p> <p>-----</p> <p>Note: Try running this module with the NetLabel loadable module detailed in Appendix E – NetLabel Module Support for network_peer_controls as there will then be an additional network domain found (network_peer_t).</p>

The SELinux Notebook - The Foundations

<i>Module Name</i>	<i>Module Description</i>	<i>Valid for Binary, Module or Source files</i>	<i>Comments on running sechecker on the modular-test policy with -v (verbose) option</i>
roles_wo_allow	This module finds roles defined in the policy that are not used in any role allow rules. It is not possible to transition to or from any role that does not have any role allow rules.	Binary, Modules and Source	This module did not find any roles without an allow rule.
roles_wo_types	This module finds roles in the policy that have no types. A role with no types cannot form a valid context.	Binary, Modules and Source	This module did not find any roles without types.
roles_wo_users	This module finds roles that are not assigned to users. If a role is not assigned to a user it cannot form a valid context.	Binary, Modules and Source	On the modular policy files sechecker reported one role without a user (<code>message_filter_r</code>). The reason for this is because the user association (<code>user_u</code>) was added with <code>semanage</code> . Note: Running <code>sechecker</code> on the binary policy does not report this error as <code>semanage</code> has added the association. Decision: Leave as it is, however <code>sechecker</code> could be modified at some stage to check the policy store !!.
spurious_audit	This module finds audit rules in the policy which do not affect the auditing of the policy. This could happen in the following situations: 1) there is an <code>allow</code> rule with the same key and permissions for a <code>dontaudit</code> rule. 2) there is an <code>auditallow</code> rule without an <code>allow</code> rule with the same key or with permissions that do not appear in an <code>allow</code> rule with the same key.	Binary, Modules and Source	This module found one spurious audit rule: <code>dontaudit ext_gateway_t unconfined_t : filesystem getattr ;</code> Decision: Review policy and update the <code>getattr</code> permission as required.
types_wo_allow	This module finds types defined in the policy that are not used in any allow rules. A type that is never granted an allow rule in the policy is a dead type. This means that all attempted access to the type will be denied including attempts to relabel to a (usable) type. The type may need to be removed from the policy or some intended access should be granted to the type.	Binary, Modules and Source	This module found one type without and allow rule (<code>socket_t</code>). This was added to the <code>netlabel.conf</code> module but never used. Decision: Remove <code>socket_t</code> .

The SELinux Notebook - The Foundations

<i>Module Name</i>	<i>Module Description</i>	<i>Valid for Binary, Module or Source files</i>	<i>Comments on running sechecker on the modular-test policy with -v (verbose) option</i>
unreachable_doms	<p>This module finds all domains in a policy which are unreachable. A domain is unreachable if any of the following apply:</p> <ol style="list-style-type: none"> 1) There is insufficient type enforcement policy to allow a transition. 2) There is insufficient RBAC policy to allow a transition. 3) There are no users with proper roles to allow a transition. <p>However, if any of the above rules indicate an unreachable domain, yet the domain appears in the system default contexts file, it is considered reachable.</p>	Modules and Source only	<p>This module found no unreachable domains.</p> <p>-----</p> <p>Note: Try running this module with the NetLabel loadable module detailed in Appendix E – NetLabel Module Support for network_peer_controls as there will then be one unreachable domain found (netlabel_peer_t).</p> <p>This was never intended as a domain, only a label for the NetLabel test. It is suspected that they were found by the find_domains utility module (Bullet 2 - it is the source of a TE rule for object class other than filesystem).</p>
users_wo_roles	<p>This module finds all the SELinux users in the policy that have no associated roles. Users without roles may appear in the label of a file system object; however, these users cannot login to the system or run any process. Since these users cannot be used on the system, a policy change is recommended to remove the users or provide some intended access.</p>	Binary, Modules and Source	<p>This module did not find any users without roles.</p>

Table 6-2: Modules in Version 1.1 of sechecker (8) – *The Comments column covers the authors interpretation of the test results on the modular-test policy base and loadable modules using the sechecker modules and profiles.*

The SELinux Notebook - The Foundations

<i>Module Name</i>	<i>Module Description</i>	<i>Valid for Binary, Module or Source files</i>	<i>Comments on running sechecker on the modular-test policy with -v (verbose) option</i>
find_assoc_types	This module finds types with an unlabeled SID.	Binary, Modules and Source	This module does not output a report using the standard profiles, however it can be run with the <code>-v</code> and <code>-m</code> options as follows: <pre>sechecker -v -m find_assoc_types <policy></pre> Running this on the modular-test policy will result in finding <code>unconfined_t</code> as the unlabeled SID.
find_domains	This is a utility module which finds types in a policy that are treated as a domain. A <code>type</code> is considered a domain if any of the following is true: 1) it has an <code>attribute</code> associated with domains. 2) it is the source of a TE rule for object class other than <code>filesystem</code> . 3) it is the default type in a <code>type_transition</code> rule for object class <code>process</code> . 4) it is associated with a role other than <code>object_r</code> .	Modules and Source only	This module does not output a report using the standard profiles, however it can be run with the <code>-v</code> and <code>-m</code> options as follows: <pre>sechecker -v -m find_domains <policy></pre> Running this on the modular-test policy will result in finding four domains (<code>move_file_t</code> , <code>ext_gateway_t</code> , <code>int_gateway_t</code> and <code>unconfined_t</code>).
find_file_types	This module finds all types in the policy treated as a file type. A <code>type</code> is considered a file type if any of the following is true: 1) it has an <code>attribute</code> associated with file types. 2) it is the source of a rule to allow <code>filesystem</code> <code>associate</code> permission. 3) it is the default type of a <code>type transition</code> rule with an object class other than <code>process</code> . 4) it is specified in a context in the <code>file_contexts</code> file.	Modules and Source only	This module does not output a report using the standard profiles, however it can be run with the <code>-v</code> and <code>-m</code> options as follows: <pre>sechecker -v -m find_file_types <policy></pre> Running this on the modular-test policy will result in finding nine file types (<code>out_file_t</code> , <code>out_queue_t</code> , <code>move_file_exec_t</code> , <code>in_file_t</code> , <code>in_queue_t</code> , <code>secure_services_exec_t</code> , <code>ext_gateway_t</code> , <code>int_gateway_t</code> and <code>unconfined_t</code>).

The SELinux Notebook - The Foundations

<i>Module Name</i>	<i>Module Description</i>	<i>Valid for Binary, Module or Source files</i>	<i>Comments on running sechecker on the modular-test policy with -v (verbose) option</i>
find_net_domains	<p>This module finds all types in a policy considered to be network domains. A type is considered a network domain if it is the subject of TE rules involving certain object classes, which are currently defined as:</p> <ol style="list-style-type: none"> 1) netif 2) tcp_socket 3) udp_socket 4) node 5) association <p>These values can be overridden in this module's profile.</p>	Binary, Modules and Source	<p>This module does not output a report using the standard profiles, however it can be run with the <code>-v</code> and <code>-m</code> options as follows:</p> <pre>sechecker -v -m find_net_domains <policy></pre> <p>Running this on the modular-test policy will result in finding three net domains (ext_gateway_t, int_gateway_t and unconfined_t).</p>
find_netif_types	<p>This module finds all types in a policy treated as a netif type. A type is considered a netif type if it is used in the context of a netifcon statement or the context of the netif initial SID.</p>	Binary, Modules and Source	<p>This module does not output a report using the standard profiles, however it can be run with the <code>-v</code> and <code>-m</code> options as follows:</p> <pre>sechecker -v -m find_netif_types <policy></pre> <p>Running this on the modular-test policy will result in finding that the only use of netif is in the initial SID for unconfined_t.</p>
find_node_types	<p>This module finds all types in a policy treated as a node type. A type is considered a node type if it is used in the context of a nodecon statement or the context of the node initial SID.</p>	Binary, Modules and Source	<p>This module does not output a report using the standard profiles, however it can be run with the <code>-v</code> and <code>-m</code> options as follows:</p> <pre>sechecker -v -m find_node_types <policy></pre> <p>Running this on the modular-test policy will result in finding that the only use of node is in the initial SID for unconfined_t.</p>

<i>Module Name</i>	<i>Module Description</i>	<i>Valid for Binary, Module or Source files</i>	<i>Comments on running sechecker on the modular-test policy with -v (verbose) option</i>
find_port_types	This module finds all types in a policy treated as a port type. A type is considered a port type if it is used in the context of a portcon statement or the context of the port initial SID.	Binary, Modules and Source	This module does not output a report using the standard profiles, however it can be run with the -v and -m options as follows: <pre>sechecker -v -m find_port_types <policy></pre> Running this on the modular-test policy will result in finding that the only use of port is in the initial SID for unconfined_t.

Table 6-3: Utility Modules in Version 1.1 of sechecker (8) – *The Comments column covers the authors interpretation of the test results on the modular-test policy base and loadable modules using the sechecker modules and profiles.*

6.6 Using apol

The apol application is part of the SETools package and has extensive help (see the ‘Help’ tab or information in the /usr/share/setools-3.3 directory).

The author had problems displaying all the apol window on the screen but resolved this as described in [Appendix H - Bugs and Features](#).

The application analyses many different aspects of a policy that are not covered in this Notebook (however the apol documentation covers all aspects), however to attempt some analysis of the message filter policy, two scenarios are presented from the ‘Analysis’ tab⁵² that were carried out using the binary policy file:

1. [Direct Relabel](#) - To show what can be relabeled by unconfined_t as the [security policy](#) stated minimum required.
2. [Transitive Information Flow](#) - This shows other paths that may be available to allow information to flow that is not directly enabled by the policy, and could therefore be used to allow unauthorised access.

The majority of text comes directly from apol as it has a facility to copy the analysis information to the clipboard.

6.6.1 General Information

The binary policy file was used as this has a complete picture of the policy. The policy source file could have been used however apol only supports the base or monolithic source. The other alternative is to use the packaged files that can be opened directly from the File>Open tab, selecting the Modular policy option, and then selecting the base module first then Add the other modules to the list as shown in [Figure 6-28](#). This list may then be exported for future use, with a sample as follows:

```
policy_list 1 modular
/etc/selinux/modular-test/modules/active/base.pp
/etc/selinux/modular-test/modules/active/modules/ext_gateway.pp
/etc/selinux/modular-test/modules/active/modules/int_gateway.pp
/etc/selinux/modular-test/modules/active/modules/move_file.pp
/etc/selinux/modular-test/modules/active/modules/netlabel.pp
```

⁵² The ‘Analysis’ tab requires a ‘permissions’ file to be loaded (via Tools>Open Default Perm Map) that adds weighting to object permissions (see the apol Help>Information Flow Analysis tab). However the one supplied (apol_perm_mapping_ver21) does not have all the new object classes added. The author updated this file (available in source package), however it made no difference to the findings (not that any were expected !!!).

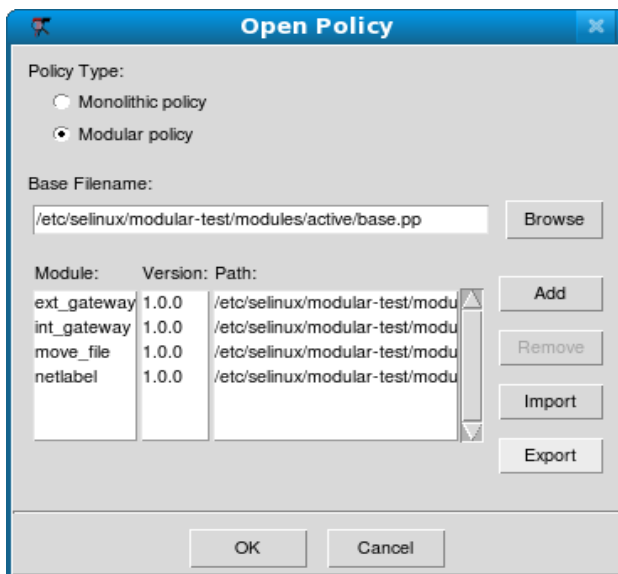
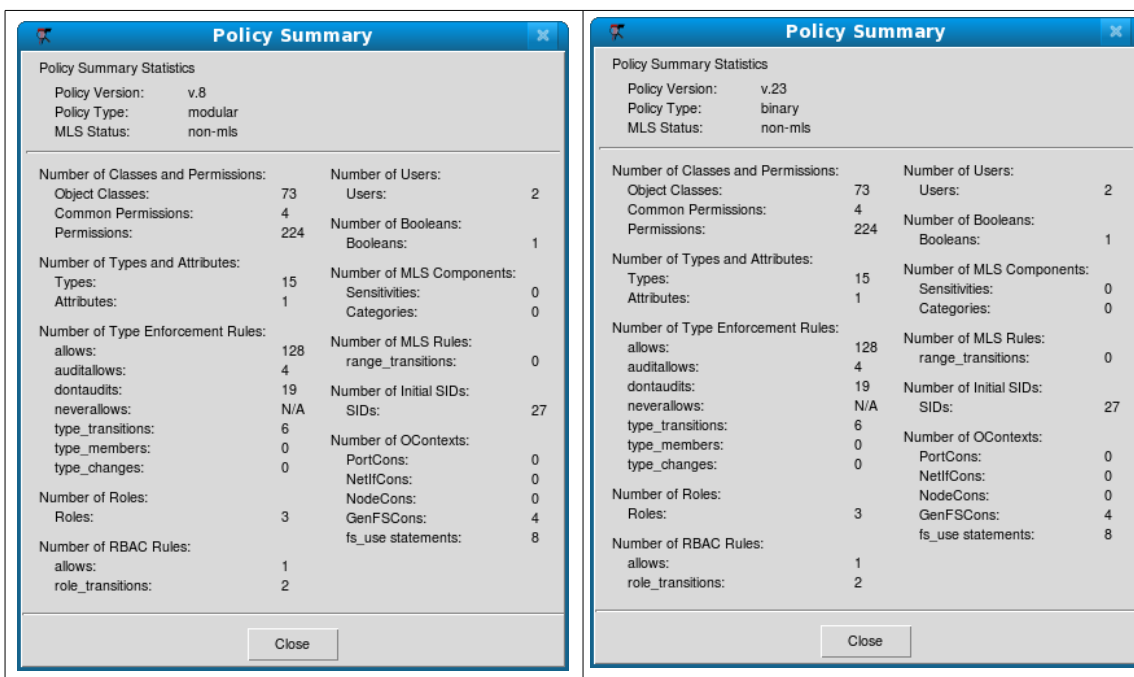


Figure 6-28: Opening the package policy files for analysis

Once the policy has been opened, a summary can be displayed using the Query>Policy Summary tabs. Figure 6-29 shows one for each of the possible options: the packages, the binary policy and the base module source. As stated earlier the binary policy will be used for analysis.



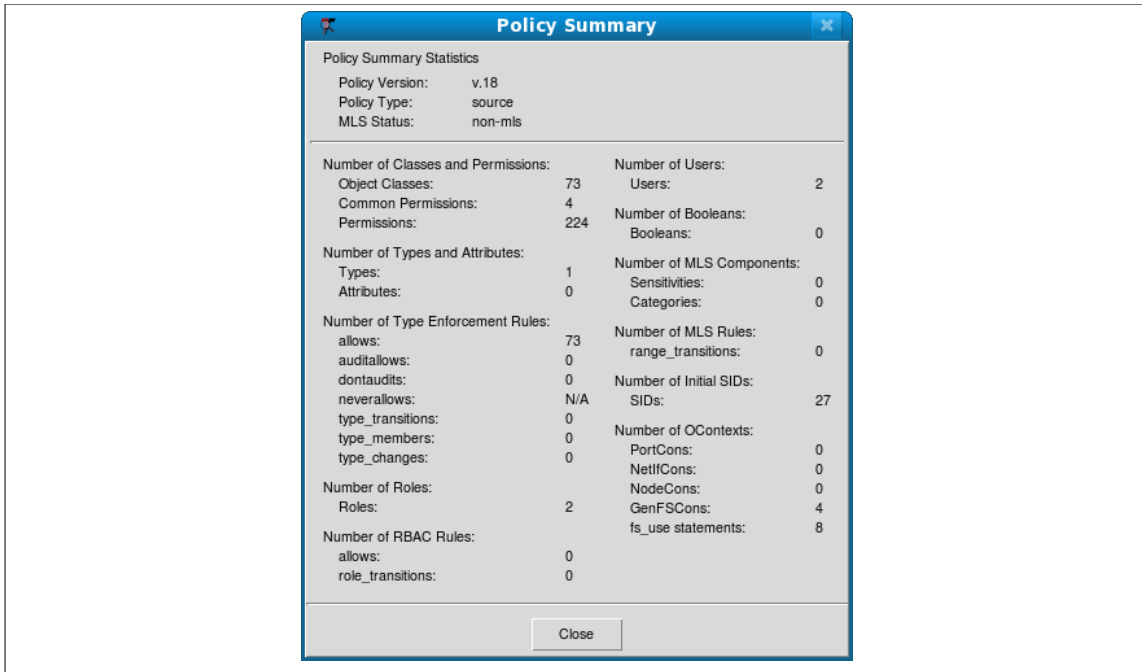


Figure 6-29: The Modular, Binary and Base Source Policy Summaries

6.6.2 Type Enforcement Rules

Figure 6-30 and Figure 6-31 show how flexible `apol` can be in searching and analysing a policy. They show a search for TE rules using a regular expression with the source set to `^un` and the target `^in` that will find all rules starting with these characters. Figure 6-30 shows that five rules were found, however when the Class/Permissions tab is set to select the `process` class (Figure 6-31), only two are found.

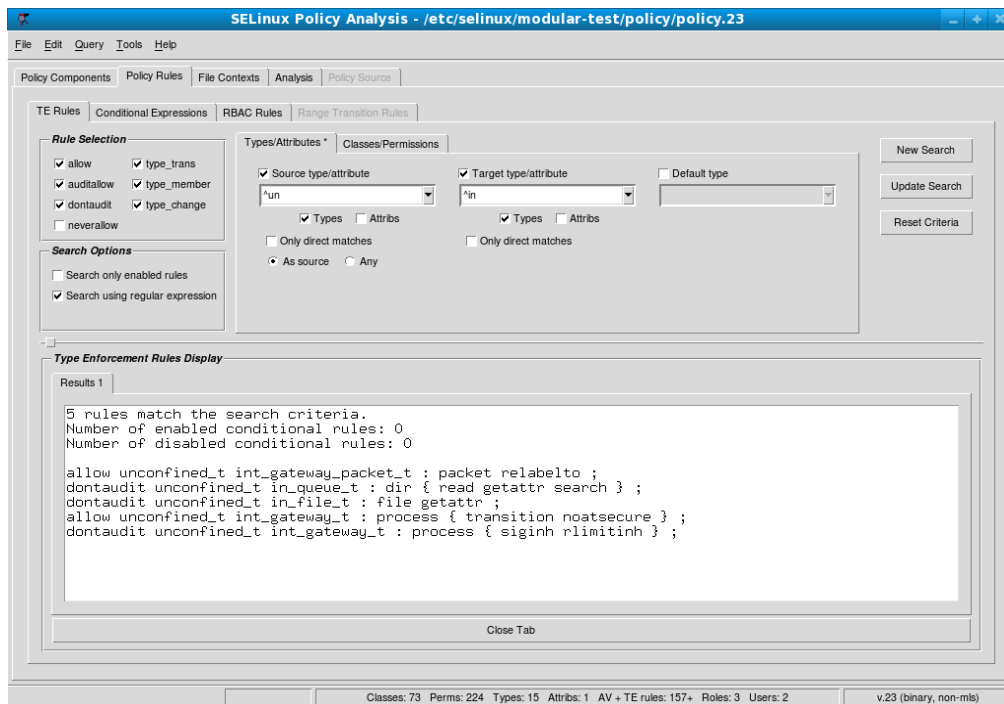


Figure 6-30: Type Enforcement Rules (1) - Finding TE rules using a regular expression.

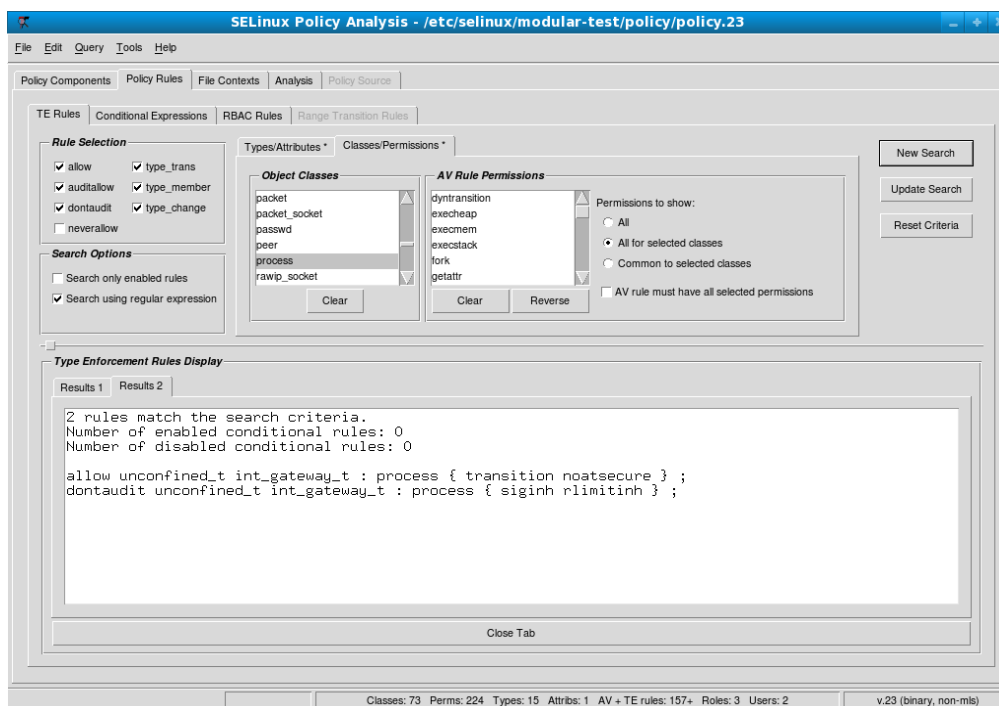


Figure 6-31: Type Enforcement Rules (2) - Finding TE rules using a regular expression with Class/Permissions tab set to show only the process class.

6.6.3 Direct Relabel

The objective of the direct relabel analysis is to show what can be relabeled by `unconfined_t` as this needed to be the least possible. When the policy was written it was decided to only allow the message filter packets to be relabeled as the `iptables` needed to be loaded under `unconfined_t` and therefore required these permissions.

6.6.3.1 ap01 Direct Relabel Analysis

Direct Relabel Analysis: Subject: `unconfined_t`

`unconfined_t` can relabel to **3** type(s) and relabel from **0** type(s).

This tab provides the results of a Direct Relabel Analysis for the subject above. The results of the analysis are presented in tree form with the root of the tree (this node) being the starting point for the analysis.

Each child node in the To and From subtrees represents a type in the current policy which the chosen subject can relabel.

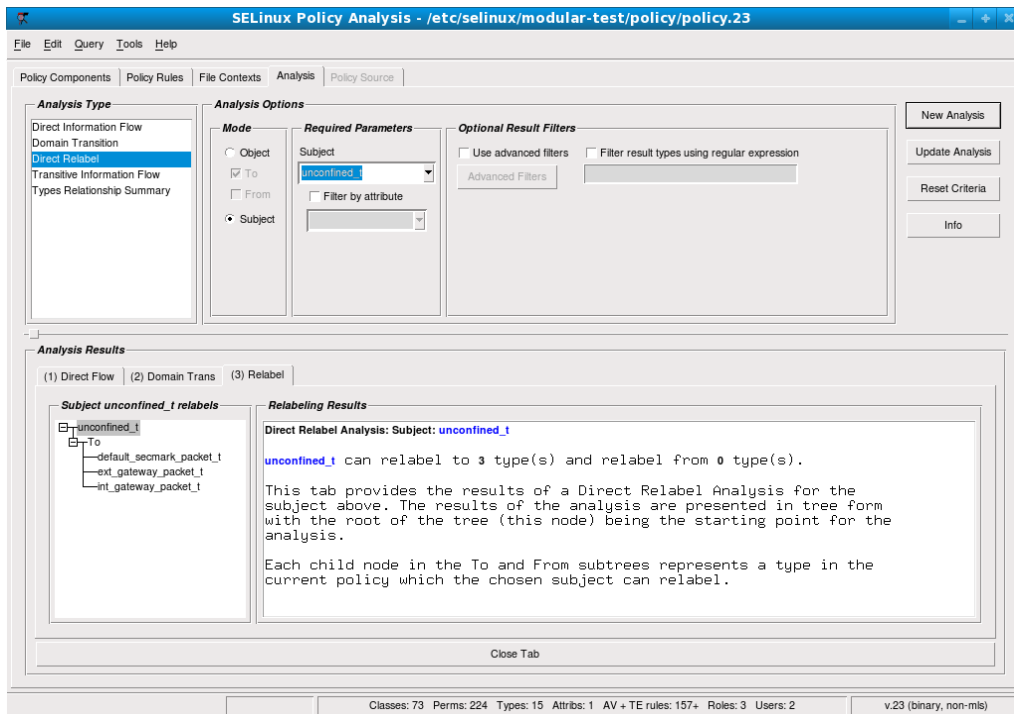


Figure 6-32: Direct Relabel - Subject: unconfined_t

Each of the nodes for the unconfined_t subject are as follows:

default_secmark_packet_t:

```
unconfined_t can relabel to default_secmark_packet_t
allow unconfined_t default_secmark_packet_t : packet { send recv relabelto } ;
```

ext_gateway_packet_t:

```
unconfined_t can relabel to ext_gateway_packet_t
allow unconfined_t ext_gateway_packet_t : packet relabelto ;
```

int_gateway_packet_t:

```
unconfined_t can relabel to int_gateway_packet_t
allow unconfined_t int_gateway_packet_t : packet relabelto ;
```

6.6.4 Transitive Information Flows

This shows other paths that may be available to allow information to flow that is not directly enabled by the policy, and could therefore be used to allow unauthorised access. The in_file_t to / from unconfined_t was analysed in an attempt to write an application that would ‘plant’ a file in the message filters ‘in_queue’ when in enforcing mode (and assuming no access to the policy build tools). The author failed miserably⁵³ – any offers !!!!

⁵³ The kernel exploit from [Brad Spengler](#) is known but was not used (also see “[SELinux hardening for mmap_min_addr protections](#)” [Ref. 16]).

6.6.4.1 apol Transitive Information Flows Analysis

Transitive Information Flow Analysis: Starting type: `in_file_t` ([To](#) and [From](#))

This tab provides the results of a Transitive Information Flow analysis beginning from the starting type selected above. The results of the analysis are presented in tree form with the root of the tree (this node) being the start point for the analysis.

Each child node in the tree represents a type in the current policy for which there is a transitive information flow to or from its parent node.

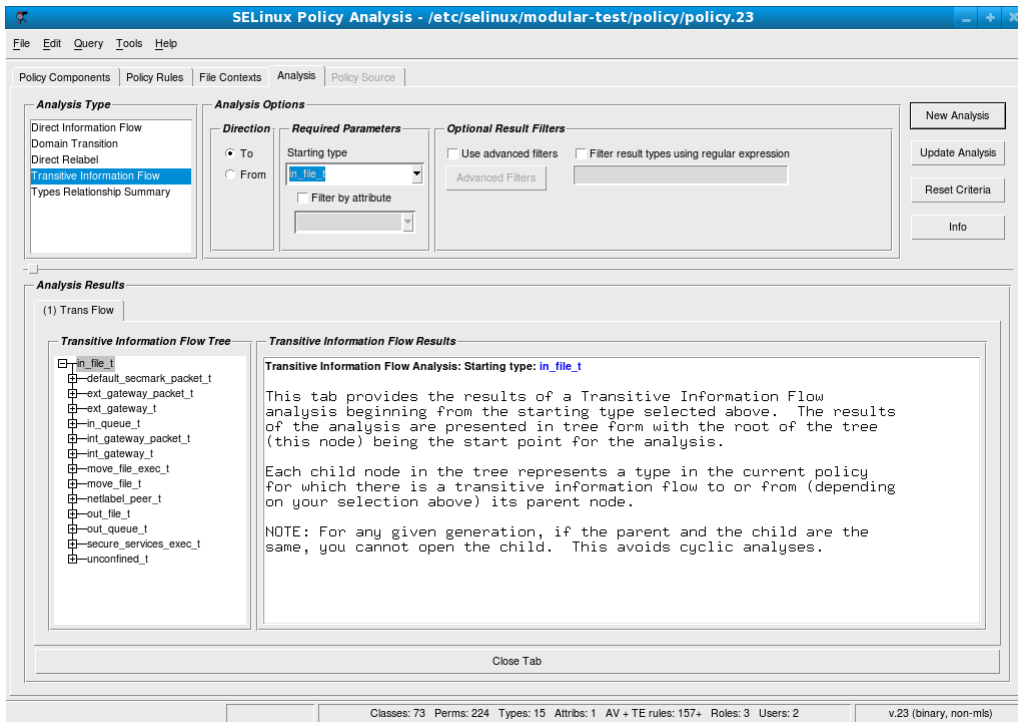


Figure 6-33: Transitive Information Flow - Starting type : `in_file_t` showing the 'to' direction.

Note that only the node for `unconfined_t` has been shown.

The first entry is with the 'To' direction selected, and the second with the 'From' direction selected (this entry has been edited⁵⁴ as it lists all objects that `unconfined_t` is allowed to relabel i.e. all of the object classes with a relabel permission).

TO `in_file_t` FROM `unconfined_t`:

```

Information flows to in_file_t from unconfined_t (find more flows)

Apol found the following number of information flows: 2

Flow 1 requires 3 steps(s).
  unconfined_t -> ext_gateway_packet_t -> ext_gateway_t -> in_file_t
  allow unconfined_t ext_gateway_packet_t : packet relabelto ;
  allow ext_gateway_t ext_gateway_packet_t : packet { send recv } ;
  allow ext_gateway_t in_file_t : file { write create getattr } ;

Flow 2 requires 2 steps(s).
    
```

⁵⁴ By selecting the 'Use advanced filters' check box and then 'Advanced Filters', it is possible to refine the search, for example excluding all permission weights below 10 (that will get permissions such as read, write and relabel).


```
unconfined_t -> ext_gateway_t -> in_file_t
allow ext_gateway_t unconfined_t : filesystem { getattr associate } ;
allow ext_gateway_t unconfined_t : association recvfrom ;
allow ext_gateway_t unconfined_t : chr_file { read write getattr } ;
allow ext_gateway_t unconfined_t : dir search ;
allow ext_gateway_t unconfined_t : fd use ;
allow ext_gateway_t unconfined_t : file { read getattr execute } ;
allow ext_gateway_t unconfined_t : lnk_file read ;
allow ext_gateway_t unconfined_t : packet { send rcv } ;
allow ext_gateway_t in_file_t : file { write create getattr } ;
```

FROM in_file_t TO unconfined_t:

Information flows from in_file_t to unconfined_t (find more flows)

Apol found the following number of information flows: 2

Flow 1 requires 2 steps(s).

```
in_file_t -> move_file_t -> unconfined_t
allow move_file_t in_file_t : file { read unlink } ;
allow move_file_t unconfined_t : fd use ;
allow move_file_t unconfined_t : chr_file { read write getattr } ;
```

Flow 2 requires 3 steps(s).

```
in_file_t -> move_file_t -> unconfined_t -> unconfined_t
allow move_file_t in_file_t : file { read unlink } ;
allow move_file_t unconfined_t : fd use ;
allow move_file_t unconfined_t : chr_file { read write getattr } ;
allow unconfined_t unconfined_t : process { fork transition sigchld sigkill
sigstop signull signal ptrace getsched setsched getsession getpgid setpgid getcap
setcap share getattr setexec setfscreate noatsecure siginh setrlimit rlimitinh
dyntransition setcurrent execmem execstack execheap setkeycreate setsockcreate } ;
.....
.....
.....
allow unconfined_t unconfined_t : unix_stream_socket { ioctl read write
create getattr setattr lock relabelfrom relabelto append bind connect listen
accept getopt setopt shutdown recvfrom sendto rcv_msg send_msg name_bind
connectto newconn acceptfrom } ;
```

7. The Reference Policy

7.1 Introduction

The Reference Policy is now the standard policy source used to build SELinux policies. This provides a single source tree with supporting documentation that can be used to build policies for different purposes such as: confining important daemons, supporting MLS / MCS type policies and locking down systems so that all processes are under SELinux control.

This section details how the Reference Policy is:

1. Constructed and types of policy builds supported.
2. Installation as a full Reference Policy source or as Header files.
3. Modifying the configuration files to build new policies.
4. Adding new modules to the build.

7.1.1 Notebook Reference Policy Information

This section makes use of the F-10 distribution that is built from the standard [Reference Policy VERSION=20081014](#)⁵⁵. This is modified and distributed by Red Hat as the following RPM:

selinux-policy-3.5.13-70.fc10.src.rpm⁵⁶

This core source code is then used to build various policy RPMs that are distributed by Red Hat as:

selinux-policy-3.5.13-70.fc10.noarch - Contains the SELinux /etc/selinux/config file, man pages and the 'Policy Header' development environment that is located at /usr/share/selinux/devel

selinux-policy-doc-3.5.13-70.fc10.noarch - Contains the html policy documentation that is located at /usr/share/doc/selinux-policy-3.5.13/html

selinux-policy-minimum-3.5.13-70.fc10.noarch

selinux-policy-mls-3.5.13-70.fc10.noarch

selinux-policy-targeted-3.5.13-70.fc10.noarch

These three rpms contain policy configuration files and the packaged policy modules (*.pp). These will be used to build the particular policy type in /usr/share/selinux/<policy_name> and the install process will then install the policy in the /etc/selinux/<policy_name> directory.

⁵⁵ The full source code and details are at the following site: <http://oss.tresys.com/projects/refpolicy>.

⁵⁶ This RPM can be obtained from the <http://koji.fedoraproject.org> web site.

7.2 Reference Policy Overview

The Reference Policy can be used to build two different formats of a policy:

1. [Loadable Module Policy](#) – A policy that has a base module for core services and has the ability to load / unload modules to support applications as required⁵⁷. This is now the standard used by GNU / Linux distributions.
2. [Monolithic Policy](#) – A policy that has all the required policy information in a single base policy.

Each of the policy types are built using module files that define the specific modules policy as detailed in the [Reference Policy Module Files](#). Note that the monolithic policy is built using the the same module files, however they are all assembled into a single ‘base’ source file.

The Reference Policy is now used by all major distributions of SELinux, however each distribution makes its own specific changes to support their ‘version of the Reference Policy’ (as this section should show as the Red Hat F-10 policy distribution has a slightly different build to the standard [Reference Policy VERSION=20081014](#)).

There are tools such as SLIDE (SELinux integrated development environment) that can be used to make the task of policy development and testing easier when using the Reference Policy source or headers. SLIDE is an [Eclipse](#) plugin and details can be found at:

<http://oss.tresys.com/projects/slide>

7.2.1 Distributing Policies

It is possible to distribute the Reference Policy in two forms:

1. As source code that is then used to build policies. This is not the general way policies are distributed as it contains the complete source that most administrators do not need. The [Reference Policy Source](#) section describes the source and the [Installing and Building the Reference Policy Source](#) section describes how to install the source and build a policy.
2. As ‘Policy Headers’. This is the most common way to distribute the Reference Policy. Basically, the modules that make up ‘the distribution’ are pre-built and then linked to form a base and optional modules. The ‘headers’ that make-up the policy are then distributed along with makefiles and documentation. A policy writer can then build policy using the core modules supported by the distribution, and using development tools they can add their own policy modules. The [Reference Policy Headers](#) section describes how these are installed and used to build modules.

The policy header files for F-10 are distributed in a number of rpms as follows:

selinux-policy-3.5.13-70.fc10.noarch – This package contains the SELinux `/etc/selinux/config` file, man pages and the ‘Policy Header’ development environment that is located at `/usr/share/selinux/devel`

⁵⁷ These can be installed by system administrators as required. The dynamic loading / unloading of policies as applications are loaded is not yet supported.

selinux-policy-doc-3.5.13-70.fc10.noarch – This package contains the html policy documentation that is located at `/usr/share/doc/selinux-policy-3.5.13/html`

selinux-policy-minimum-3.5.13-70.fc10.noarch

selinux-policy-mls-3.5.13-70.fc10.noarch

selinux-policy-targeted-3.5.13-70.fc10.noarch

These three packages contain policy configuration files and policy modules (*.pp files) for the particular policy type to be installed.

These files are used to build the policy type in `/usr/share/selinux/<policy_name>` and then install the policy in the `/etc/selinux/<policy_name>` directory.

Normally only one policy would be installed and active, however for development purposes all three can be installed.

7.2.2 Policy Type Functionality

As can be seen from the policies distributed with F-10 above, they can also be classified by the type of functionality they support, for example the Red Hat policies support⁵⁸:

`minimum` – supports a minimal set of confined daemons within their own domains. The remainder run in the `unconfined_t` space.

`mls` – supports server based MLS systems.

`targeted` – supports a greater number of confined daemons and can also confine other areas and users (this targeted version also supports the older ‘strict’ version).

For information, the Reference Policy from the Tresys repository supports the following types:

`standard` – supports confined daemons and can also confine other areas and users (this is an amalgamated version of the older ‘targeted’ and ‘strict’ versions).

`mcs` – As `standard` but supports MCS labels.

`mls` – supports MLS labels and confines server processes (as currently MLS does not support desktop applications (but should be in F-12)).

7.2.3 Reference Policy Module Files

The reference policy modules are constructed using a mixture of [policy language statements](#), [support macros](#) and [access interface calls](#) using three principle types of source file (note that all three must exist even if empty):

1. A private policy file that contains statements required to enforce policy on the specific GNU / Linux service being defined within the module. These files are named `<module_name>.te`.

For example the `ada.te` file shown below has two statements:

⁵⁸ Note that Red Hat pre-configure MCS support within all their policies.

- a) one to state that the `ada_t` process has permission to write to the stack and memory allocated to a file.
 - b) one that states that if the `unconfined` module is loaded, then allow the `ada_t` domain unconfined access. Note that if the flow of this statement is followed it will be seen that many more interfaces and macros are called to build the final raw SELinux language statements. An expanded module source is shown in the [Module Expansion Process](#) section.
2. An external interface file that defines the services available to other modules. These files are named `<module_name>.if`.

For example the `ada.if` file shown below has two interfaces defined for other modules to call (see also [Figure 7-34](#) that shows a screen shot of the documentation that can be automatically generated):

- a) `ada_domtrans` - that allows another module (running in domain `$1`) to run the `ada` application in the `ada_t` domain.
- b) `ada_run` - that allows another module to run the `ada` application in the `ada_t` domain (via the `ada_domtrans` interface), then associate the `ada_t` domain to the caller defined role (`$2`) and terminal (`$3`).

Provided of course that the caller domain has permission.

It should be noted that there are two types of interface specification:

Access Interfaces – These are the most common and define interfaces that `.te` modules can call as described in the `ada` examples. They are generated by the `interface` macro as detailed in the [interface Macro](#) section.

Template Interfaces – These are required whenever a module is required in different domains and allows the type(s) to be redefined by adding a prefix supplied by the calling module. The basic idea is to set up an application in a domain that is suitable for the defined SELinux user and role to access but not others. These are generated by the `template` macro as detailed in the [template Macro](#) section that also explains the `openoffice.if` template.

3. A file labeling file that defines the labels to be added to files for the specified module. These files are named `<module_name>.fc`. The build process will amalgamate all the `.fc` files and finally form the [file_contexts](#) file that will be used to label the filesystem.

For example the `ada.fc` file shown below requires that the specified files are all labeled `system_u:object_r:ada_exec_t:s0`.

The `<module_name>` must be unique within the reference policy source tree and should reflect the specific GNU / Linux service being enforced by the policy.

The module files are constructed using a mixture of:

1. Policy language statements as defined in the [SELinux Policy Language](#) section.
2. Reference Policy macros that are defined in the [Reference Policy Macros](#) section.

3. External interface calls defined within other modules (.te and .if only).

An example of each file taken from the ada module is as follows:

ada.te file contents:

```
policy_module(ada, 1.2.0)

#####
#
# Declarations
#

type ada_t;
type ada_exec_t;
application_domain(ada_t, ada_exec_t)
role system_r types ada_t;

#####
#
# Local policy
#

allow ada_t self:process { execstack execmem };

optional_policy(`
    unconfined_domain_noaudit(ada_t)
`)
```

ada.if file contents:

```
## <summary>GNAT Ada95 compiler</summary>

#####
## <summary>
## Execute the ada program in the ada domain.
## </summary>
## <param name="domain">
## <summary>
## Domain allowed access.
## </summary>
## </param>
#
interface(`ada_domtrans',`
    gen_require(`
        type ada_t, ada_exec_t;
    `)

    corecmd_search_bin($1)
    domtrans_pattern($1, ada_exec_t, ada_t)
`)
```

```
#####
## <summary>
## Execute ada in the ada domain, and
## allow the specified role the ada domain.
## </summary>
## <param name="domain">
## <summary>
## Domain allowed access.
## </summary>
## </param>
## <param name="role">
## <summary>
## The role to be allowed the ada domain.
## </summary>
## </param>
## <param name="terminal">
## <summary>
## The type of the terminal allow the ada domain to use.
## </summary>
## </param>
#
```

```
interface(`ada_run',`
  gen_require(`
    type ada_t;
  ')

  ada_domtrans($1)
  role $2 types ada_t;
  allow ada_t $3:chr_file rw_term_perms;
')
```

ada.fc file contents:

```
#
# /usr
#
/usr/bin/gnatbind -- gen_context(system_u:object_r:ada_exec_t,s0)
/usr/bin/gnatls -- gen_context(system_u:object_r:ada_exec_t,s0)
/usr/bin/gnatmake -- gen_context(system_u:object_r:ada_exec_t,s0)
/usr/libexec/gcc(/.*)?/gnat1 -- gen_context(system_u:object_r:ada_exec_t,s0)
```

7.2.4 Reference Policy Documentation

One of the advantages of the reference policy is that it is possible to automatically generate documentation as a part of the build process. This documentation is defined in XML and generated as HTML files suitable for viewing via a browser.

The documentation for F-10 can be found in the following locations:

Distributed as Policy Headers - /usr/share/doc/selinux-policy-<version>/html. Where <version> is the version number of the Red Hat release, for the build used in this Notebook the location is:

```
/usr/share/doc/selinux-policy-3.5.13/html
```

Distributed as Policy Source - <location>/src/policy/doc/html. Where <location> is the location of the installed source after make install-src has been executed as described in the [Installing The Reference Policy Source](#) section. The documentation can then be generated using make html, where for the build used in this Notebook the location is:

```
/etc/selinux/targeted-70/src/policy/doc/html
```

[Figure 7-34](#) shows an example screen shot of the documentation produced for the ada module interfaces.

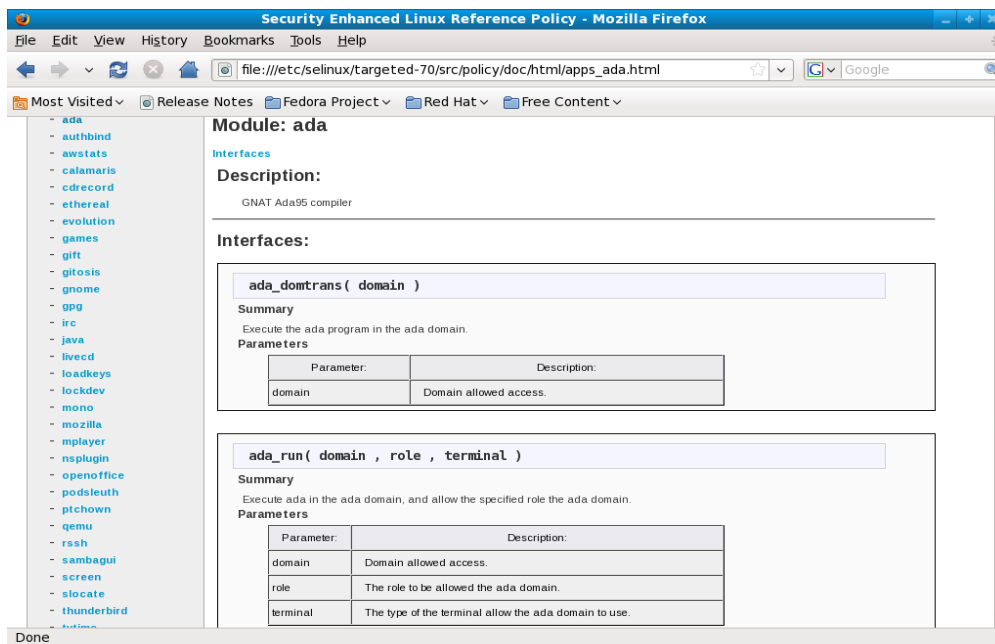


Figure 7-34: Example Documentation Screen Shot

7.3 Reference Policy Source

This section will explain the source layout and configuration files, with the actual installation and building covered in the [Installing and Building the Reference Policy Source](#) section.

The source has a README file containing information on the configuration and installation processes that has been used within this section (and updated with the authors comments as necessary). There is also a VERSION file that contains the Reference Policy release date which can be used to obtain the original source from the repository located at:

<http://oss.tresys.com/projects/refpolicy>

7.3.1 Source Layout

Figure 7-35 shows the layout of the reference policy source tree, that once installed would be located at:

```
/etc/selinux/<policy_name>/src/policy
```

The following sections detail the source contents:

- [Reference Policy Files and Directories](#) – Describes the files and their location.
- [Source Configuration Files](#) – Details the contents of the build.conf and modules.conf configuration files.
- [Source Installation and Build Make Options](#) – Describes the make targets.
- [Modular Policy Build Process](#) – Describes how the various source files are linked together to form a base policy module (base.conf) during the build process.

The [Installing and Building the Reference Policy Source](#) section then describes how the initial source is installed and configured to allow a version of the F-10 targeted policy to be built.

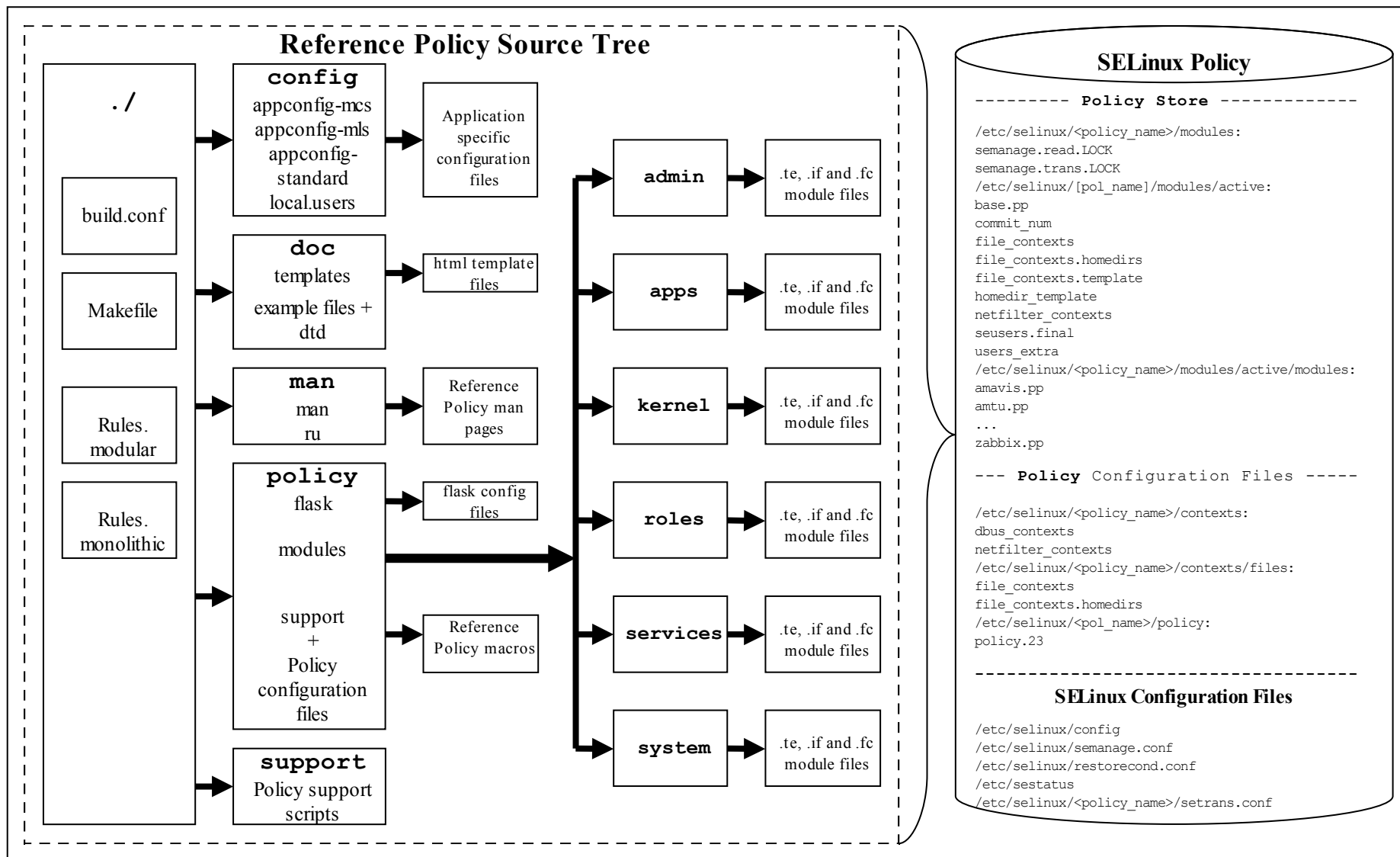


Figure 7-35: The Reference Policy Source Tree – When building a modular policy files are added to the policy store, for monolithic builds the policy store is not used.

7.3.2 Reference Policy Files and Directories

[Table 7-1](#) shows the major files and their directories with a description of each taken from the README file. All directories are relative to the root of the Reference Policy source directory `./policy`.

Two of these configuration files (`build.conf` and `modules.conf`) are further detailed in the [Source Configuration Files](#) as they define how the policy will be built.

During the build process, a file is generated in the `./policy` directory called either `policy.conf` or `base.conf` depending whether a monolithic or modular policy is being built. This file is explained in the [Modular Policy Build Structure](#) section.

File / Directory Name	Comments
<code>Makefile</code>	General rules for building the policy.
<code>Rules.modular</code>	Makefile rules specific to building loadable module policies.
<code>Rules.monolithic</code>	Makefile rules specific to building monolithic policies.
<code>build.conf</code>	Options which influence the building of the policy, such as the policy type and distribution. This file is described in the Reference Policy Build Options - build.conf section.
<code>config/appconfig-*</code>	Application configuration files for all configurations of the Reference Policy (<code>standard</code> , <code>MLS</code> or <code>MCS</code>). These are used by SELinux-aware programs. These files are described in the SELinux Configuration Files section.
<code>config/local.users</code>	The file read by load policy for adding SELinux users to the policy on the fly. This file is described in the section. Note that this file is not used in the F-10 modular policy build.
<code>doc/html/*</code>	This contains the contents of the in-policy XML documentation, presented in web page form.
<code>doc/policy.dtd</code>	The <code>doc/policy.xml</code> file is validated against this DTD.
<code>doc/policy.xml</code>	This file is generated/updated by the <code>conf</code> and <code>html</code> make targets. It contains the complete XML documentation included in the policy.
<code>doc/templates/*</code>	Templates used for documentation web pages.
<code>support/*</code>	Tools used in the build process.
<code>policy/flask/initial_sids</code>	This file has declarations for each <code>initial SID</code> . The file usage in policy generation is described in the Modular Policy Build Structure section.
<code>policy/flask/security_classes</code>	This file has declarations for each <code>security class</code> . The file usage in policy generation is described in the Modular Policy Build Structure section.
<code>policy/flask/access_vectors</code>	This file defines the <code>access vectors</code> . Common prefixes for <code>access vectors</code> may be defined at the beginning of the file. After the common prefixes are defined, an <code>access vector</code> may be defined for each

File / Directory Name	Comments
	<p>security class.</p> <p>The file usage in policy generation is described in the Modular Policy Build Structure section.</p>
policy/modules/*	<p>Each directory represents a layer in Reference Policy all of the modules are contained in one of these layers.</p> <p>The files present are:</p> <p>metadata.xml – describes the layer.</p> <p><module_name>.te, .if & .fc – contains policy source as described in the Reference Policy Module Files section.</p> <p>The file usage in policy generation is described in the Modular Policy Build Structure section.</p>
policy/booleans.conf	<p>This file is generated/updated by the conf make target. It contains the booleans in the policy, and their default values. If tunables are implemented as booleans, tunables will also be included. This file will be installed as the /etc/selinux/NAME/booleans file (note that this is not true for F-10 or any system that implements the modular policy - see the Booleans, Global Booleans and Tunable Booleans section).</p> <p>This file is also included in the F-10 source updates as described in the Installing and Building the Reference Policy Source section.</p>
policy/constraints	<p>This file defines additional constraints on permissions in the form of boolean expressions that must be satisfied in order for specified permissions to be granted. These constraints are used to further refine the type enforcement rules and the role allow rules. Typically, these constraints are used to restrict changes in user identity or role to certain domains.</p> <p>(Note that this file does not contain the MLS / MCS constraints as they are in the mls and mcs files described below).</p> <p>The file usage in policy generation is described in the Modular Policy Build Structure section.</p>
policy/global_booleans	<p>This file defines all booleans that have a global scope, their default value, and documentation. See the Booleans, Global Booleans and Tunable Booleans section.</p>
policy/global_tunables	<p>This file defines all tunables that have a global scope, their default value, and documentation. See the Booleans, Global Booleans and Tunable Booleans section.</p>
policy/mcs	<p>This contains information used to generate the sensitivity, category, level and mlsconstraint statements used to define the MCS configuration.</p> <p>The file usage in policy generation is described in the Modular Policy Build Structure section.</p>
policy/mls	<p>This contains information used to generate the sensitivity, category, level and mlsconstraint statements used to define the MLS</p>

File / Directory Name	Comments
	configuration. The file usage in policy generation is described in the Modular Policy Build Structure section.
<code>policy/modules.conf</code>	This file contains a listing of available modules, and how they will be used when building Reference Policy. This file is described in the Reference Policy Build Options - modules.conf section, it is also updated by the F-10 source updates as described in the Installing and Building the Reference Policy Source section.
<code>policy/policy_capabilities</code>	This file defines the policy capabilities that can be enabled in the policy. The file usage in policy generation is described in the Modular Policy Build Structure section.
<code>policy/rolemap</code>	This file contains prefix and user domain type that corresponds to each user role. The contents of this file will be used to expand the per-user domain templates for each module. <i>Note this does not seem to be used in the Reference Policy supported by F-10 (commented out in makefiles).</i>
<code>securetty_types</code>	This file is not part of the standard distribution but is added by the F-10 source updates as described in the Installing and Building the Reference Policy Source section.
<code>setrans.conf</code>	This file is not part of the standard distribution but is added by the F-10 source updates as described in the Installing and Building the Reference Policy Source section.
<code>policy/users</code>	This file defines the users included in the policy. The file usage in policy generation is described in the Modular Policy Build Structure section.
<code>policy/support/*</code>	Reference Policy support macros. These are described in the Reference Policy Macros section.

Table 7-1: The Reference Policy Files and Directories

7.3.3 Source Configuration Files

There are two major configuration files (`build.conf` and `modules.conf`) that define the policy to be built and are detailed in this section.

7.3.3.1 Reference Policy Build Options - `build.conf`

This file defines the policy type to be built that will influence its name and where the source will be located once it is finally installed. It also configures the MCS / MLS sensitivity and category maximum values. An example file content is shown in the [Installing and Building the Reference Policy Source](#) section where it is used to install and then build the policy.

[Table 7-2](#) explains the fields that can be defined within this file, however there are a number of m4 macro parameters that are set up when this file is read by the build process makefiles. These definitions are shown in [Table 7-3](#) and are also used within

the module source files to control how the policy is built with examples shown in the [ifdef/ifndef Parameters](#) section.

Option	Type	Comments
TYPE	String	Available options are <code>standard</code> , <code>mls</code> , and <code>mcs</code> . For a type enforcement only system, set <code>standard</code> . This optionally enables multi-level security (MLS) or multi-category security (MCS) features. This option controls <code>enable_mls</code> , and <code>enable_mcs</code> policy blocks.
NAME	String (optional)	Sets the name of the policy; the <code>NAME</code> is used when installing files to e.g., <code>/etc/selinux/NAME</code> and <code>/usr/share/selinux/NAME</code> . If not set, the policy type field (<code>TYPE</code>) is used. The policy built under this directory is also controlled by the <code>modules.conf</code> that is described in the Reference Policy Build Options – policy/modules.conf section.
DISTRO	String (optional)	Enable distribution-specific policy. Available options are <code>redhat</code> , <code>rhel4</code> , <code>gentoo</code> , <code>debian</code> , and <code>suse</code> . This option controls <code>distro_redhat</code> , <code>distro_rhel4</code> , <code>distro_suse</code> policy blocks.
MONOLITHIC	Boolean (y n)	If 'y' a monolithic policy is built, otherwise a modular policy is built.
DIRECT_INITRC	Boolean (y n)	If 'y' <code>sysadm</code> will be allowed to directly run <code>init</code> scripts, instead of requiring the <code>run_init</code> tool. This is a build option instead of a tunable since role transitions do not work in conditional policy. This option controls <code>direct_sysadm_daemon</code> policy blocks.
OUTPUT_POLICY	Integer	Set the version of the policy created when building a monolithic policy. This option has no effect on modular policy.
UNK_PERMS	String	Set the kernel behaviour for handling of permissions defined in the kernel but missing from the policy. The permissions can either be allowed, denied, or the policy loading can be rejected. See the SELinux Filesystem for more details.
MLS_SENS	Integer	Set the number of sensitivities in the MLS policy. Ignored on <code>standard</code> and <code>MCS</code> policies.
MLS_CATS	Integer	Set the number of categories in the MLS policy. Ignored on <code>standard</code> and <code>MCS</code> policies.
MCS_CATS	Integer	Set the number of categories in the MCS policy. Ignored on <code>standard</code> and <code>MLS</code> policies.
QUIET	Boolean (y n)	If 'y' the build system will only display status messages and error messages. This option has no effect on policy.

Table 7-2: build.conf Entries

Parameter Name	From build.conf entry	Comments
enable_mls	TYPE	Set if MLS policy build enabled.
enable_mcs	TYPE	Set if MCS policy build enabled.
mls_num_sens	MLS_SENS	The number of MLS sensitivities configured.
mls_num_cats	MLS_CATS	The number of MLS categories configured.
mcs_num_cats	MCS_CATS	The number of MCS categories configured.
distro_\$	DISTRO	The distro can be: redhat, debian or blank.
direct_sysadm_daemon	DIRECT_INITRC	As defined in the DIRECT_INITRC entry above.
hide_broken_syptoms		This is set up in the Makefile and can be used in modules to hide errors with dontaudit rules (or even allow rules).

Table 7-3: m4 parameters set at build time

7.3.3.2 Reference Policy Build Options – policy/modules.conf

This file controls what modules are built within the policy with example entries as follows:

```
# Layer: kernel
# Module: kernel
# Required in base
#
# Policy for kernel threads, proc filesystem, and unlabeled processes and
objects.
#
kernel = base

# Module: amanda
#
# Automated backup program.
#
amanda = module

# Layer: admin
# Module: ddcprobe
#
# ddcprobe retrieves monitor and graphics card information
#
ddcprobe = off
```

As can be seen the only active line (those without comments⁵⁹) is:

```
<module_name> = base | module | off
```

Where:

- module_name The name of the module to be included within the build.
- base The module will be in the base module for a modular policy build (build.conf entry MONOLITHIC = n).
- module The module will be built as a loadable module for a modular

⁵⁹ The comments are also important as they form part of the documentation when it is generated by the make html target.

policy build. If a monolithic policy is being built (build.conf entry MONOLITHIC = y), then this module will be built into the base module.

off The module will not be included in any build.

Generally it is up to the policy writer to decide which modules are in the base and those that are loadable, however there are some modules that **MUST** be in the base module. To highlight this there is a special entry at the start of the modules interface file (.if) that has the entry <required val="true"> as shown below (taken from the kernel.if file):

```
## <summary>
## Policy for kernel threads, proc filesystem,
## and unlabeled processes and objects.
## </summary>
## <required val="true">
## This module has initial SIDs.
## </required>
```

The modules.conf file will also reflect that a module is required in the base by adding a comment 'Required in base' when the make conf target is executed (as all the .if files are checked during this process and the modules.conf file updated).

```
# Layer: kernel
# Module: kernel
# Required in base
#
# Policy for kernel threads, proc filesystem, and unlabeled processes and
objects.
#
kernel = base
```

There are 13 modules in the F-10 reference policy source marked as required and are shown in [Table 7-4](#).

Layer	Module Name	Comments
system	application	Policy for user executable applications. All the interface calls start with 'application_'.
system	setrans	Policy for the SELinux MLS/MCS label translation service. All the interface calls start with 'setrans_'.
kernel	corecommands	Core policy for shells, and generic programs in: /bin, /sbin, /usr/bin, and /usr/sbin. The .fc file sets up the labels for these items. All the interface calls start with 'corecmd_'.
kernel	corenetwork	Policy controlling access to network objects and also contains the initial SIDs for these. The .if file is large and automatically generated. All the interface calls start with 'corenet_'.
kernel	devices	This module creates the device node concept and provides the policy for many of the device files. Notable exceptions are the mass storage and terminal devices that are covered by other

Layer	Module Name	Comments
		modules (that is a char or block device file, usually in /dev). All types that are used to label device nodes should use the dev_node macro. Additionally this module controls access to three things: <ol style="list-style-type: none"> 1. the device directories containing device nodes. 2. device nodes as a group 3. individual access to specific device nodes covered by this module. All the interface calls start with 'dev_'.
kernel	domain	Contains the core policy for forming and managing domains. All the interface calls start with 'domain_'.
kernel	files	This module contains basic filesystem types and interfaces and includes: <ol style="list-style-type: none"> 1. The concept of different file types including basic files, mount points, tmp files, etc. 2. Access to groups of files and all files. 3. Types and interfaces for the basic filesystem layout (/ , /etc, /tmp, /usr, etc.). 4. Contains the file initial SID. All the interface calls start with 'files_'.
kernel	filesystem	Contains the policy for filesystems and the initial SID. All the interface calls start with 'fs_'.
kernel	kernel	Contains the policy for kernel threads, proc filesystem, and unlabeled processes and objects. This module has initial SIDs. All the interface calls start with 'kernel_'.
kernel	mcs	Policy for Multicategory security. The .te file only contains attributes used in MCS policy. All the interface calls start with 'mcs_'.
kernel	mls	Policy for Multilevel security. The .te file only contains attributes used in MLS policy. All the interface calls start with 'mls_'.
kernel	selinux	Contains the policy for the kernel SELinux security interface (selinuxfs). All the interface calls start with 'selinux_'.
kernel	terminal	Contains the policy for terminals. All the interface calls start with 'term_'.

Table 7-4: required modules.conf Entries

7.3.3.2.1 Building the modules.conf File

The file can be created by an editor, however it is generally built initially by `make conf` that will add any additional modules to the file. The file can then be edited to configure the required modules as base, module or off.

As will be seen in the [Installing and Building the Reference Policy Source](#) section, the F-10 source comes with a number of pre-configured files that are used to produce the required policy including multiple versions of the `modules.conf` file.

7.3.4 Source Installation and Build Make Options

This section explains the various make options available that have been taken from the README file. [Table 7-5](#) describes the general make targets, [Table 7-6](#) describes the modular policy make targets and [Table 7-7](#) describes the monolithic policy make targets.

Make Target	Comments
install-src	Install the policy sources into <code>/etc/selinux/NAME/src/policy</code> , where <code>NAME</code> is defined in the <code>build.conf</code> file. If it is not defined, then <code>TYPE</code> is used instead. If a <code>build.conf</code> does not have the information, then the <code>Makefile</code> will default to the current entry in the <code>/etc/selinux/config</code> file or default to <code>refpolicy</code> . A pre-existing source policy will be moved to <code>/etc/selinux/NAME/src/policy.bak</code> .
conf	Regenerate <code>policy.xml</code> , and update/create <code>modules.conf</code> and <code>booleans.conf</code> . This should be done after adding or removing modules, or after running the <code>bare</code> target. If the configuration files exist, their settings will be preserved. This must be run on policy sources that are checked out from the CVS repository before they can be used.
clean	Delete all temporary files, compiled policies, and <code>file_contexts</code> . Configuration files are left intact.
bare	Do the <code>clean</code> make target and also delete configuration files, web page documentation, and <code>policy.xml</code> .
html	Regenerate <code>policy.xml</code> and create web page documentation in the <code>doc/html</code> directory.

Table 7-5: General Build Make Targets

Make Target	Comments
base	Compile and package the <code>base</code> module. This is the default target for modular policies.
modules	Compile and package all Reference Policy modules configured to be built as loadable modules.
MODULENAME.pp	Compile and package the <code>MODULENAME</code> Reference Policy module.
all	Compile and package the <code>base</code> module and all Reference Policy modules configured to be built as loadable modules.
install	Compile, package, and install the <code>base</code> module and Reference Policy modules configured to be built as loadable modules.
load	Compile, package, and install the <code>base</code> module and Reference Policy modules configured to be built as loadable modules, then insert them into the module store.
validate	Validate if the configured modules can successfully link and expand.
install-headers	Install the policy headers into <code>/usr/share/selinux/NAME</code> . The headers are sufficient for building a policy module locally, without requiring the complete Reference Policy sources. The <code>build.conf</code> settings for this policy configuration should be set before using this target.

Table 7-6: Modular Policy Build Make Targets

Make Target	Comments
policy	Compile a policy locally for development and testing. This is the default target for monolithic policies.
install	Compile and install the policy and file contexts.
load	Compile and install the policy and file contexts, then load the policy.
enableaudit	Remove all dontaudit rules from <code>policy.conf</code> .
relabel	Relabel the filesystem.
checklabels	Check the labels on the filesystem, and report when a file would be relabeled, but do not change its label.
restorelabels	Relabel the filesystem and report each file that is relabeled.

Table 7-7: Monolithic Policy Build Make Targets

7.3.5 Booleans, Global Booleans and Tunable Booleans

The three files `booleans.conf`, `global_booleans` and `global_tunables` are built and used as follows:

`booleans.conf` This file is generated / updated by `make conf`, and contains all the booleans in the policy with their default values. If tunable and global booleans are implemented then these are also included.

This file can also be delivered as a part of the reference policy source as shown in the [Installing and Building the Reference Policy Source](#) section. This is generally because other default values are used for booleans and not those defined within the modules themselves (i.e. distribution specific booleans). When the `make install` is executed, then this file will be used to set the default values.

Note that if booleans are updated locally then the policy store will contain a `booleans.local` file.

In SELinux enabled systems that support the policy store features (modular policies) this file is not installed as `/etc/selinux/NAME/booleans`.

`global_booleans` These are booleans that have been defined in the `global_tunables` file using the `gen_bool` macro. They are normally booleans for managing the overall policy and currently consist of the following (where the default values are `false`):

```
secure_mode
secure_mode_insmod
secure_mode_policyload
```

`global_tunables` These are booleans that have been defined in module files using the [gen_tunable](#) macro and added to the `global_tunables` file by `make conf`. The [tunable_policy](#) macros are defined in each module where policy statements or interface calls are required. They are booleans for managing specific areas of policy that are global in scope. An example is `allow_execstack` that will allow all processes running in `unconfined_t` to make their stacks executable.

7.3.6 Modular Policy Build Structure

This section explains the way a modular policy is constructed, this does not really need to be known but is used to show the files used that can then be investigated if required.

When `make all` or `make load` or `make install` are executed the `build.conf` and `modules.conf` files are used to define the policy name and what modules will be built in the base and those as individual loadable modules.

Basically the source modules (`.te`, `.if` and `.fc`) and core flask files are rebuilt in the `tmp` directory where the reference policy macros⁶⁰ in the source modules will be expanded to form actual policy language statements as described in the [SELinux Policy Language](#) section. [Figure 7-36](#) shows these temporary files that are used to form the `base.conf`⁶¹ file during policy generation.

The `base.conf` file will consist of language statements taken from the module defined as `base` in the `modules.conf` file along with the constraints, users etc. that are required to build a complete policy.

The individual loadable modules are built in much the same way as shown in [Figure 7-37](#).

Base Policy Component Description	Policy Source File Name (relative to <code>./policy/policy</code>)	<code>./policy/tmp</code> File Name
The object classes supported by the kernel.	<code>flask/security_classes</code>	<code>pre_te_files.conf</code>
The initial SIDs supported by the kernel.	<code>flask/initial_sids</code>	
The object class permissions supported by the kernel.	<code>flask/access_vectors</code>	
This is either the expanded mls or mcs file depending on the type of policy being built.	<code>mls</code> or <code>mcs</code>	
These are the policy capabilities that can be configured / enabled to support the policy.	<code>policy_capabilities</code>	
This area contains all the <code>attribute</code> , <code>bool</code> , <code>type</code> and <code>typealias</code>	<code>modules/**/*.te</code> <code>modules/**/*.if</code>	<code>all_attrs_types.conf</code>

⁶⁰ These are explained in the [Reference Policy Macros](#) section.

⁶¹ The `base.conf` gets built for modular policies and a `policy.conf` file gets built for a monolithic policy.

Base Policy Component Description	Policy Source File Name (relative to ./policy/policy)	./policy/tmp File Name
statements extracted from the *.te and *.if files that form the base module.		
Contains the global and tunable bools extracted from the conf files.	global_bools.conf global_tunables.conf	global_bools.conf
Contains the rules extracted from each of the modules .te and .if files defined in the modules.conf file as 'base'.	base modules	only_te_rules.conf
Contains the expanded users from the users file.	users	all_post.conf
Contains the expanded constraints from the constraints file.	constraints	
Contains the default SID labeling extracted from the *.te files.	modules/**/*.te	
Contains the fs_use_xattr, fs_use_task, fs_use_trans and genfscon statements extracted from each of the modules .te and .if files defined in the modules.conf file as 'base'.	modules/**/*.te modules/**/*.if	
Contains the netifcon, nodecon and portcon statements extracted from each of the modules .te and .if files defined in the modules.conf file as 'base'.	modules/**/*.te modules/**/*.if	
Contains the expanded file context file entries extracted from the *.fc files defined in the modules.conf file as 'base'.	modules/**/*.fc	base.fc.tmp
Expanded seusers file.	seusers	seusers
<p>These are the commands used to compile, link and load the base policy module:</p> <pre>checkmodule base.conf -o tmp/base.mod semodule_package -o base.conf -m base_mod -f base_fc -u users_extra -s tmp/seusers semodule -s \$(NAME) -b base.pp -i and each module .pp file</pre> <p>The 'NAME' is that defined in the build.conf file.</p>		

Figure 7-36: Base Module Build – This shows the temporary build files used to build the base module 'base.conf' as a part of the 'make' process. Note that the modules marked as base in modules.conf are built here.

Base Policy Component Description	Policy Source File Name (relative to ./policy/policy)	./policy/tmp File Name
For each module defined as 'module' in the modules.conf configuration file, a source module is produced that has been extracted from the *.te and *.if file for that module.	modules/**/<module_name>.te modules/**/<module_name>.if	<module_name>.tmp
For each module defined as 'module' in the modules.conf configuration file, an object module is produced from executing the checkmodule command	tmp/<module_name>.tmp	<module_name>.mod

shown below.		
For each module defined as 'module' in the <code>modules.conf</code> configuration file, an expanded file context file is built from the <code><module_name>.fc</code> file.	<code>modules/*/<module_name>.fc</code>	<code>base.fc.tmp</code>
<p>This command is used to compile each module:</p> <pre>checkmodule tmp/<module_name>.tmp -o tmp/<module_name>.mod</pre> <p>Each module is packaged and loaded with the base module using the following commands:</p> <pre>semodule_package -o base.conf -m base_mod -f base_fc -u users_extra -s tmp/seusers semodule -s \$(NAME) -b base.pp) -i and each module .pp file</pre> <p>The 'NAME' is that defined in the <code>build.conf</code> file.</p>		

Figure 7-37: Module Build – This shows the module files and the temporary build files used to build each module as a part of the 'make' process (i.e. those modules marked as module in `modules.conf`).

7.3.7 Creating Additional Layers

One objective of the reference policy is to separate the modules into different layers reflecting their 'service' (e.g. kernel, system, app etc.). While it can sometimes be difficult to determine where a particular module should reside, it does help separation, however because the way the build process works, each module must have a unique name.

If a new layer is required, then the following will need to be completed:

1. Create a new layer directory `./policy/modules/LAYERNAME` that reflects the layer's purpose.
2. In the `./policy/modules/LAYERNAME` directory create a `metadata.xml` file. This is an XML file with a `summary` tag and optional `desc` (long description) tag that should describe the purpose of the layer and will be used as a part of the documentation. An example is as follows:

```
<summary>ABC modules for the XYZ components.</summary>
```

7.4 Installing and Building the Reference Policy Source

This section explains how to install the F-10 reference policy source that is distributed by Red Hat (however the same principle is followed for the source taken directly from the [Tresys repository](http://trsys.fedoraproject.org), except that it will not build a compatible policy to that discussed in this section).

Any F-10 policy source rpm will suffice and can be obtained from the <http://koji.fedoraproject.org> web site, however it is assumed that the source rpm is:

`selinux-policy-3.5.13-70.fc10.src.rpm`

The objective of this exercise is to show that the policy built from the above source rpm is an exact replica of the targeted policy distributed as header files in the F-10 targeted rpm:

`selinux-policy-targeted-3.5.13-70.fc10.noarch.rpm`

7.4.1 Installation and Configuration

Install the source by:

```
rpm -Uvh selinux-policy-3.5.13-70.fc10.src.rpm
```

The source will be installed in the users home directory under `./rpmbuild/SOURCES` where the `serefpolicy-3.5.13.tgz` will need to be unpacked:

```
cd $HOME/rpmbuild/SOURCES
tar -xzf serefpolicy.tgz
```

The SOURCES directory contents will then look like this:

```
booleans-minimum.conf    modules-olpc.conf        securetty_types-targeted
booleans-mls.conf        modules-targeted.conf    serefpolicy-3.5.13
booleans-olpc.conf       policy-20080710.patch   serefpolicy-3.5.13.tgz
booleans-targeted.conf   policygentool            setrans-minimum.conf
Makefile.devel           securetty_types-minimum  setrans-mls.conf
modules-minimum.conf     securetty_types-mls     setrans-olpc.conf
modules-mls.conf         securetty_types-olpc    setrans-targeted.conf
```

The files with `minimum`, `targeted`, `mls` and `olpc` within their names are the specific configuration files used within the Reference Policy for that particular Red Hat policy type.

As the ‘targeted’ policy is being built, the files shown in [Table 7-8](#) left hand column need to be copied to the location and named as shown in the right hand column.

Configuration File:	Installed as Reference Policy Configuration File:
<code>modules-targeted.conf</code>	<code>./serefpolicy-3.5.13/policy/modules.conf</code>
<code>booleans-targeted.conf</code>	<code>./serefpolicy-3.5.13/policy/booleans.conf</code>
<code>securetty_types-targeted</code>	<code>./serefpolicy-3.5.13/policy/securetty_types</code>
<code>setrans-targeted.conf</code>	<code>./serefpolicy-3.5.13/policy/setrans.conf</code>

Table 7-8: Red Hat specific policy configuration files – This example builds a ‘targeted’ policy.

The latest patches now need to be applied to the source tree as follows:

```
patch -p0 <policy-20080710.patch
```

The `serefpolicy-3.5.13` directory will now contain the source code with the latest patches for this release (`3.5.13-70`) of the Red Hat Reference Policy and the correct configuration files for a targeted policy.

The `./serefpolicy-3.5.13/build.conf` must now be modified to allow the source to be installed in its final location and have the correct parameters set for

the build. The entries that need to be updated in the `build.conf` file are highlighted below⁶²:

```
#
# Policy build options
#

# Policy version
# By default, checkpolicy will create the highest version policy it supports.
# Setting this will override the version. This only has an effect for
monolithic
# policies.
#OUTPUT_POLICY = 18

# Policy Type
# standard, mls, mcs. Note Red Hat always build the MCS Policy Type
# as their 'targeted' version.
TYPE = mcs

# Policy Name
# If set, this will be used as the policy name. Otherwise the policy type will
be
# used for the name. This entry is also used by the 'make install-src' process
# to copy the source to the /etc/selinux/targeted-70/src/policy directory.
NAME = targeted-70

# Distribution
# Some distributions have portions of policy for programs or configurations
# specific to the distribution. Setting this will enable options for the
# distribution. redhat, gentoo, debian, suse, and rhel4 are current options.
# Fedora users should enable redhat.
DISTRO = redhat

# Unknown Permissions Handling
# The behaviour for handling permissions defined in the kernel but missing from
# the policy. The permissions can either be allowed, denied, or the policy
# loading can be rejected.
# allow, deny, and reject are current options.
#UNK_PERMS = deny

# Direct admin init
# Setting this will allow sysadm to directly run init scripts, instead of
# requiring run_init. This is a build option, as role transitions do not work in
# conditional policy.
DIRECT_INITRC = n

# Build monolithic policy. Putting n here will build a loadable module policy.
MONOLITHIC = n

# Number of MLS Sensitivities
# The sensitivities will be s0 to s(MLS_SENS-1). Dominance will be in increasing
# numerical order with s0 being lowest.
MLS_SENS = 16

# Number of MLS Categories. Note Red Hat use 1024 categories for MLS and MCS.
# The categories will be c0 to c(MLS_CATS-1).
MLS_CATS = 1024

# Number of MCS Categories
# The categories will be c0 to c(MLS_CATS-1).
MCS_CATS = 1024

# Set this to y to only display status messages during build.
QUIET = n
```

The policy source is now in a position to be installed at its default location that will be derived from the `NAME = targeted-70` entry and will therefore be located at:

⁶² The README file in this directory contains helpful information on installation of the source, headers, documentation etc. The only point the README will not cover are the Red Hat specific configuration files that need to be copied over as shown in [Table 7-8](#).


```
/etc/selinux/targeted-70/src/policy
```

7.4.2 Building the targeted Policy Type

From the `./serefpolicy-3.5.13` directory run the following command:

```
make install-src
```

This will copy the source code to its final location making any directories required.

Once the copy process is complete the policy can be built and the modules loaded into the policy store⁶³ by running the following commands:

```
# Go to the source location:  
cd /etc/selinux/targeted-70/src/policy
```

```
# To ensure a clean source build:  
make clean
```

```
# Finally build policy modules and load into the policy store:  
make load
```

The policy will now be built as a `targeted` policy that will be an exact copy of the policy distributed in the following rpm:

```
selinux-policy-targeted-3.5.13-70.fc10.noarch.rpm
```

7.4.3 Checking the Build

Now that the targeted policy has been built, the policy binary file can be compared to the one that is distributed and built by the following rpm:

```
selinux-policy-targeted-3.5.13-70.fc10.noarch
```

The binary files sizes of both policies should be 3,491,012 bytes.

```
ls -l /etc/selinux/targeted/policy  
-rw-r--r-- root root policy.23      3491012  
  
ls -l /etc/selinux/targeted-70/policy  
-rw-r--r-- root root policy.23      3491012
```

Note that the binaries would not be an exact comparison due to time stamps etc., therefore the SETools `sediffx` utility should be run against the two binary policies⁶⁴ which should show that they are the same and give the results shown in [Figure 7-38](#).

⁶³ Note that the term ‘load’ is not loading the policy as the active policy, but just building the base policy + the modules and installing them ready to be activated if required

⁶⁴ Be aware that comparing these two policies on a low specification machine will take hours. It is best to select a few items for comparison first.

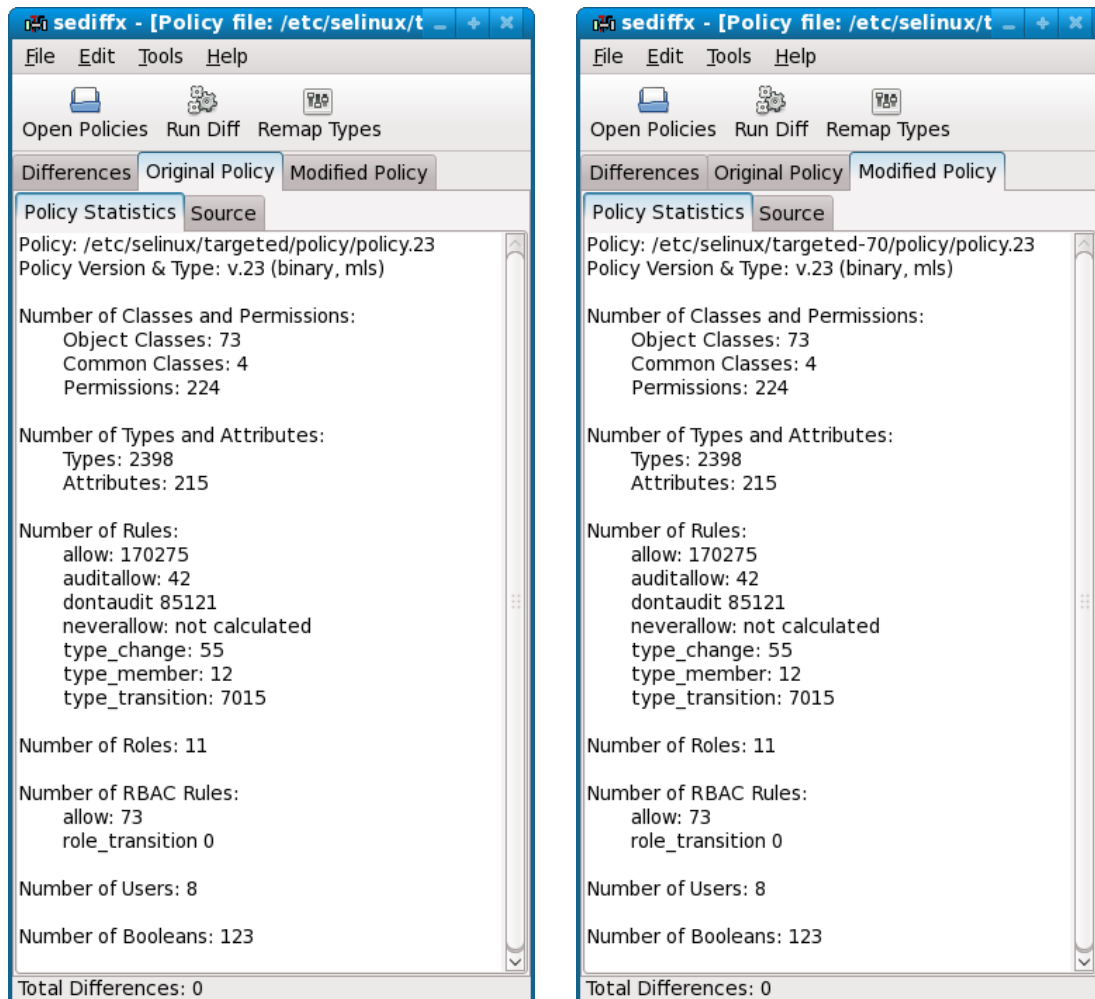


Figure 7-38: The two ‘targeted’ policies should be the same using sediffx

7.4.4 Running with the new Policy

To run the system using the new targeted-70 build edit the /etc/selinux/config file entry to read SELINUXTYPE=targeted-70, and then run the following commands:

```
touch /.autorelabel
shutdown -r now
```

During reboot, the system will be relabeled and the policy loaded (hopefully with no errors).

7.5 Reference Policy Headers

This method of building policy and adding new modules is used for distributions that do not require access to the source code.

Note that the Reference Policy header and the [Red Hat F-10 policy header installations](#) are slightly different as described below.

7.5.1 Building and Installing the Header Files

To be able to fully build the policy headers from the reference policy source two steps are required:

1. Ensure the source is installed and configured as described in the [Installing and Building the Reference Policy Source](#) section. This is because the `make load` (or `make install`) command will package all the modules as defined in the `modules.conf` file, producing a `base.pp` and the relevant `.pp` packages. The build process will then install these files in the `/usr/share/selinux/<policy_name>` directory.
2. Execute the `make install-headers` command that will:
 - a) Produce a `build.conf` file that represents the contents of the master `build.conf` file and place it in the `/usr/share/selinux/<policy_name>/include` directory.
 - b) Produce the XML documentation set that reflects the source and place it in the `/usr/share/selinux/<policy_name>/include` directory.
 - c) Copy a development Makefile for building from policy headers to the `/usr/share/selinux/<policy_name>/include` directory.
 - d) Copy the support macros `.spt` files to the `/usr/share/selinux/<policy_name>/include/support` directory.
 - e) Copy the module interface files (`.if`) to the relevant module directories at:
`/usr/share/selinux/<policy_name>/include/modules.`

The directory structure for the `targeted-70` build generated above (edited for readability) would be:

```
The policy packages:
targeted-70/ada.pp
....
....
targeted-70/base.pp

Build / Configuration files:
targeted-70/include/build.conf
targeted-70/include/Makefile
targeted-70/include/rolemap # Note this file is not used by F-10

XML Documentation:
targeted-70/include/global_tunables.xml
targeted-70/include/global_booleans.xml
targeted-70/include/apps.xml
targeted-70/include/roles.xml
targeted-70/include/system.xml
targeted-70/include/kernel.xml
targeted-70/include/services.xml
targeted-70/include/admin.xml

Support Macros:
targeted-70/include/support/ipc_patterns.spt
...
...
The module interface files in their relevant directories:
```

```
targeted-70/include/admin/acct.if
..
..
targeted-70/include/apps/ada.if
..
...
targeted-70/include/kernel/terminal.if
..
...
targeted-70/include/roles/auditadm.if
..
..
targeted-70/include/services/arpwatch.if
...
...
targeted-70/include/system/ipsec.if
..
..
...
```

7.5.2 Using the Header Files

Note that this section describes the standard Reference Policy headers, the F-10 installation is slightly different and described in the [Using F-10 Supplied Headers](#) section.

Once the headers are installed as defined above, new modules can be built in any local directory. An example set of module files are located in the reference policy source at `./policy/doc` and are called `example.te`, `example.if`, and `example.fc`.

During the header build process a Makefile was included in the headers directory. This Makefile can be used to build the example modules by using `make -f` option as follows (assuming that the example module files are in the local directory):

```
make -f /usr/share/selinux/<policy_name>/include/Makefile
```

However there is another Makefile that can be installed in the local directory that will call the master Makefile. This is located at `./policy/doc` in the reference policy source and is called `Makefile.example`. This is shown below (note that it extracts the `<policy_name>` from the SELinux config file):

```
AWK ?= gawk

NAME ?= $(shell $(AWK) -F= '/^SELINUXTYPE/{ print $$2 }'
/etc/selinux/config)
SHAREDIR ?= /usr/share/selinux
HEADERDIR := $(SHAREDIR)/$(NAME)/include

include $(HEADERDIR)/Makefile
```

[Table 7-9](#) shows the make targets for modules built from headers.

Make Target	Comments
MODULENAME.pp	Compile and package the MODULENAME local module.
all	Compile and package the modules in the current directory.
load	Compile and package the modules in the current directory, then insert them into the module store.
refresh	Attempts to reinsert all modules that are currently in the module store from the local and system module packages.

Make Target	Comments
<code>xml</code>	Build a <code>policy.xml</code> from the XML included with the base policy headers and any XML in the modules in the current directory.

Table 7-9: Header Policy Build Make Targets

7.5.3 Using F-10 Supplied Headers

The F-10 distribution installs the headers in a slightly different manner as Red Hat installs:

- The packaged files under the `/usr/share/selinux/<policy_name>`, these files may be `.pp` files or `.pp.bz2` depending on the version of `rpm` installed (later ones compressed the packages). They are installed by the `selinux-policy-<policy_name>-3.5.13-70.fc10.noarch` type rpms.
- The development header files are installed in the `/usr/share/selinux/devel` directory by the `selinux-policy-3.5.13-70.fc10.noarch` rpm. Red Hat also include an additional application called `policygentool` that allows users to generate policy by answering various questions. This tool is described in the [Fedora 10 SELinux User Guide](#) [Ref. 1]. The example modules are also in this directory and the `Makefile` is also slightly different to that used by the Reference Policy source.
- The documentation is supplied in the `selinux-policy-doc-3.5.13-70.fc10.noarch` type rpms and would be installed (for this version), in the `/usr/share/doc/selinux-policy-3.5.13/html` directory.

7.6 Reference Policy Macros

This section explains some of the common macros used to build reference policy source modules (see [Table 7-10](#) for the list). [Appendix D - Reference Policy Macros](#) contains a list of all the macros as a reference but does not detail their usage.

These macros are located at:

`./policy/policy/support` for the reference policy source.

`/usr/share/selinux/<policy_name>/include/support` for reference policy supplied header files.

`/usr/share/selinux/devel/support` for Red Hat supplied header files.

They consist of the following files:

`loadable_module.spt` - Loadable module support.

`misc_macros.spt` - Generate users, bools and security contexts.

`mls_mcs_macros.spt` - MLS/MCS support.

`file_patterns.spt` - Sets up allow rules via parameters for files and directories.

`ipc_patterns.spt` - Sets up allow rules via parameters for Unix domain sockets.

`misc_patterns.spt` - Domain and process transitions.

`obj_perm_sets.spt` - Object classes and permissions.

Macro Name	Function	Macro file name
<code>policy_module</code>	For adding the module statement and mandatory require block entries.	<code>loadable_module.spt</code>
<code>gen_require</code>	For use in interfaces to optionally insert a require block	
<code>template</code>	Generate <code>template</code> interface block	
<code>interface</code>	Generate the access interface block	
<code>optional_policy</code>	Optional policy handling	
<code>gen_tunable</code>	Tunable declaration	
<code>tunable_policy</code>	Tunable policy handling	
<code>gen_user</code>	Generate an SELinux user	<code>misc_macros.spt</code>
<code>gen_context</code>	Generate a security context	
<code>gen_bool</code>	Generate a boolean	
<code>gen_cats</code>	Declares categories <code>c0</code> to <code>c (N-1)</code>	<code>mls_mcs_macros.spt</code>
<code>gen_sens</code>	Declares sensitivities <code>s0</code> to <code>s (N-1)</code> with dominance in increasing numeric order with <code>s0</code> lowest, <code>s (N-1)</code> highest.	
<code>gen_levels</code>	Generate levels from <code>s0</code> to <code>(N-1)</code> with categories <code>c0</code> to <code>(M-1)</code>	
<code>mls_systemlow</code>	Basic level names for system low and high	
<code>mls_systemhigh</code>		
<code>mcs_systemlow</code>		
<code>mcs_systemhigh</code>		
<code>mcs_allcats</code>	Allocates all categories	

Table 7-10: Macros listed in this section

Notes:

1. These macros can be in any configuration file read by the build process and are contained in (for example) the `users`, `mls`, `mcs` and `constraints` files.
2. There are three main m4 `ifdef` parameters used within modules:
 - a) `enable_mcs` - this is used to test if the MCS policy is being built.
 - b) `enable_mls` - this is used to test if the MLS policy is being built.
 - c) `hide_broken_symptoms` - this is used to hide errors in modules with `dontaudit` rules (or even allow rules).

These are also mentioned in [Table 7-3](#) as they are set by the initial build process with examples shown in the [ifdef / ifndef Parameters](#) section.

3. The macro examples in this section have been taken from the reference policy module files and shown in each relevant “**Example Macro**” section. The

macros are then expanded by the build process to form modules containing the policy language statements and rules in the `tmp` directory. These files have been extracted and modified for readability, then shown in each relevant “**Expanded Macro**” section.

4. An example policy that has had macros expanded is shown in the [Module Expansion Process](#) section.
5. Be aware that spaces between macro names and their parameters are not allowed:

Allowed:

```
policy_module(ftp, 1.7.0)
```

Not allowed:

```
policy_module (ftp, 1.7.0)
```

7.6.1 Loadable Policy Macros

The loadable policy module support macros are located in the `loadable_module.spt` file.

7.6.1.1 `policy_module` Macro

This macro will add the [module statement](#) to a loadable module, and automatically add a [require Statement](#) with pre-defined information for all loadable modules such as the `system_r` role, kernel classes and permissions, and optionally MCS / MLS information (`sensitivity` and `category` statements).

The macro definition is:

```
policy_module(module_name,version)
```

Where:

<code>policy_module</code>	The <code>policy_module</code> macro keyword.
<code>module_name</code>	The module identifier that must be unique in the module layers.
<code>version_number</code>	The module version number in <code>M.m.m</code> format (where <code>M</code> = major version number and <code>m</code> = minor version numbers).

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	No	No

Example Macro:

```
# This example is from the modules/services/ftp.te module:
#
policy_module(ftp, 1.7.0)
```

Expanded Macro:

```
# This is the expanded macro from the tmp/ftp.tmp file:
#
module ftp 1.7.0;
require {
    role system_r;
    class security {compute_av compute_create .... };
    ....
    class capability2 (mac_override mac_admin );

# If MLS or MCS configured then the:
sensitivity s0;
....
category c0;
....
}
```

7.6.1.2 gen_require Macro

For use within module files to insert a require block.

The macro definition is:

```
gen_require(`require_statements`)
```

Where:

gen_require	The gen_require macro keyword.
require_statements	These statements consist of those allowed in the policy language require Statement .

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	Yes	No

Example Macro:

```
# This example is from the modules/services/ftp.te module:
#
gen_require(`type ftp_script_exec_t;`)
```

Expanded Macro:

```
# This is the expanded macro from the tmp/ftp.tmp file:
#
require {
```



```
type ftp_script_exec_t;
}
```

7.6.1.3 optional_policy Macro

For use within module files to insert an `optional` block that will be expanded by the build process only if the modules containing the access or template interface calls that follow are present. If one module is present and the other is not, then the optional statements are not included (need to check).

The macro definition is:

```
optional_policy(`optional_statements`)
```

Where:

- `optional_policy` The `optional_policy` macro keyword.
- `optional_statements` These statements consist of those allowed in the policy language [optional Statement](#). However they can also be [interface](#), [template](#) or support macro calls.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	Yes	No

Example Macro:

```
# This example is from the modules/services/ftp.te module and
# shows the optional_policy macro with two levels.
#
optional_policy(`
    corecmd_exec_shell(ftp_t)
    files_read_usr_files(ftp_t)
    cron_system_entry(ftp_t, ftpd_exec_t)

    optional_policy(`
        logrotate_exec(ftp_t)
    `)
`)
```

Expanded Macro:

```
# This is the expanded macro from the tmp/ftp.tmp file showing
# the policy language statements with both optional levels
# expanded.
#
##### Start optional_policy - Level 1 #####
optional {
##### begin corecmd_exec_shell(ftp_t)
    require {
```

```
    type bin_t, shell_exec_t;
} # end require
allow ftpd_t bin_t:dir { getattr search };
allow ftpd_t bin_t:dir { getattr search read lock ioctl };
allow ftpd_t bin_t:dir { getattr search };
allow ftpd_t bin_t:lnk_file { getattr read };
allow ftpd_t shell_exec_t:file { { getattr read execute ioctl } ioctl lock
execute_no_trans };
##### end corecmd_exec_shell(ftp_t)

##### begin files_read_usr_files(ftp_t)
require {
    type usr_t;
} # end require
allow ftpd_t usr_t:dir { getattr search read lock ioctl };
allow ftpd_t usr_t:dir { getattr search };
allow ftpd_t usr_t:file { getattr read lock ioctl };
allow ftpd_t usr_t:dir { getattr search };
allow ftpd_t usr_t:lnk_file { getattr read };
##### end files_read_usr_files(ftp_t)

##### begin cron_system_entry(ftp_t, ftp_exec_t)
require {
    type crond_t, system_crond_t;
} # end require
allow system_crond_t ftp_exec_t:file { getattr read execute };
allow system_crond_t ftp_t:process transition;
dontaudit system_crond_t ftp_t:process { noatsecure siginh rlimitinh };
type_transition system_crond_t ftp_exec_t:process ftp_t;
# cjp: perhaps these four rules from the old
# domain_auto_trans are not needed?
allow ftpd_t system_crond_t:fd use;
allow ftpd_t system_crond_t:fifo_file { getattr read write append ioctl
lock };
allow ftpd_t system_crond_t:process sigchld;
allow ftpd_t crond_t:fifo_file { getattr read write append ioctl lock };
allow ftpd_t crond_t:fd use;
allow ftpd_t crond_t:process sigchld;
role system_r types ftp_t;
##### end cron_system_entry(ftp_t, ftp_exec_t)

##### Start optional_policy - Level 2 #####
optional {
##### begin logrotate_exec(ftp_t)
require {
    type logrotate_exec_t;
} # end require
allow ftpd_t logrotate_exec_t:file { { getattr read execute ioctl } ioctl
lock execute_no_trans };
##### end logrotate_exec(ftp_t)
} # end optional 2nd level

} # end optional 1st level
```

7.6.1.4 gen_tunable Macro

This macro defines booleans that are global in scope. The corresponding [tunable_policy](#) macro contains the supporting statements allowed or not depending on the value of the boolean. These entries are extracted as a part of the build process (by the `make conf` target) and added to the `global_tunables` file where they can then be used to alter the default values for the `make load` or `make install` targets.

Note that the comments shown in the example **MUST** be present as they are used to describe the function and are extracted for the [documentation](#).

The macro definition is:

```
gen_tunable(boolean_name,boolean_value)
```

Where:

gen_tunable	The gen_tunable macro keyword.
boolean_name	The boolean identifier.
boolean_value	The boolean value that can be either true or false.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	Yes	No

Example Macro:

```
# This example is from the modules/services/ftp.te module:
#
## <desc>
## <p>
## Allow ftp servers to use nfs
## for public file transfer services.
## </p>
## </desc>
gen_tunable(allow_ftpd_use_nfs, false)
```

Expanded Macro:

```
# This is the expanded macro from the tmp/ftp.tmp file:
#
bool allow_ftpd_use_nfs false;
```

7.6.1.5 tunable_policy Macro

This macro contains the statements allowed or not depending on the value of the boolean defined by the [gen_tunable](#) macro.

The macro definition is:

```
tunable_policy(`gen_tunable_id', `tunable_policy_rules`)
```

Where:

tunable_policy	The tunable_policy macro keyword.
gen_tunable_id	This is the boolean identifier defined by the gen_tunable macro. It is possible to have multiple entries separated by && or as shown in the example.
tunable_policy_rules	These are the policy rules and statements as defined in the if statement policy language

section.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	Yes	No

Example Macro:

```
# This example is from the modules/services/ftp.te module
# showing the use of the boolean with the && operator.
#
tunable_policy(`allow_ftp_use_nfs && allow_ftp_anon_write', `
    fs_manage_nfs_files(ftp_t)
')
```

Expanded Macro:

```
# This is the expanded macro from the tmp/ftp.tmp file.
#
if (allow_ftp_use_nfs && allow_ftp_anon_write) {
##### begin fs_manage_nfs_files(ftp_t)
    require {
        type nfs_t;
    } # end require

    allow ftpd_t nfs_t:dir { read getattr lock search ioctl
add_name remove_name write };
    allow ftpd_t nfs_t:file { create open getattr setattr read
write append rename link unlink ioctl lock };
##### end fs_manage_nfs_files(ftp_t)
} # end if
```

7.6.1.6 interface Macro

Access interface macros are defined in the interface module file (.if) and form the interface through which other modules can call on the modules services (as shown in [Figure 7-40](#) and described in the [Module Expansion](#) section.

Note that the comments shown in the example **MUST** be present as they are used to describe the function and are extracted for the [documentation](#).

The macro definition is:

```
interface(`name`, `interface_rules`)
```

Where:

- interface The interface macro keyword.
- name The interface identifier that should be named to reflect the module identifier and its purpose.

interface_rules This can consist of the support macros, policy language statements or other interface calls as required to provide the service.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
No	Yes	No

Example Interface Definition:

```
# This example is from the modules/services/ftp.if module
# showing the 'ftp_read_config' interface.
#
#####
## <summary>
##     Read ftpd etc files
## </summary>
## <param name="domain">
## <summary>
##     Domain allowed access.
## </summary>
## </param>
#
interface(`ftp_read_config', `
    gen_require(`
        type ftpd_etc_t;
    ')

    files_search_etc($1)
    allow $1 ftpd_etc_t:file { getattr read };
')
```

Expanded Macro: (taken from the base.conf file):

```
# Access Interfaces are only expanded at policy compile time
# if they are called by a module that requires their services.
#
# In this example the ftp_read_config interface is called from
# the init.te module via the optional_policy macro as shown
# below with the expanded code shown afterwards.
#
##### From ./policy/policy/modules/system/init.te #####
#
# optional_policy(`
#     ftp_read_config(initrc_t)
# ')
#
#
##### Expanded policy statements taken #####
##### from the base.conf file that #####
##### forms the base policy. #####
#
optional { # Start optional_policy segment for ftp interface
#
# This is the resulting output contained the base.conf file
# where init calls the ftp_read_config ($1) interface from
```

```
# init.te with the parameter initrc_t:
#
require {
    type ftpd_etc_t;
}

#
# Call the files_search_etc ($1) interface contained in the
# ftp.if file with the parameter initrc_t:
#
require {
    type etc_t;
}
allow initrc_t etc_t:dir { getattr search };
#
# end files_search_etc(initrc_t)
#
# This is the allow $1 ftpd_etc_t:file { getattr read };
# statement with the initrc_t parameter resolved:
#
allow initrc_t ftpd_etc_t:file { getattr read };
#
# end ftp_read_config(initrc_t)
} # End optional_policy segment for this ftp interface
```

7.6.1.7 template Macro

A template interface is used to help create a domain and set up the appropriate rules and statements to run an application / process. The basic idea is to set up an application in a domain that is suitable for the defined SELinux user and role to access but not others. Should a different user / role need to access the same application, another domain would be allocated (these are known as ‘derived domains’ as the domain name is derived from caller information).

The application template shown in the example below is for `openoffice.org` where the domain being set up to run the application is based on the SELinux user `xguest` (parameter `$1`) therefore a domain type is initialised called `xguest_openoffice_t`, this is then added to the user domain attribute `xguest_usertype` (parameter `$2`). Finally the role `xguest_r` (parameter `$3`) is allowed access to the domain type `xguest_openoffice_t`. If a different user / role required access to `openoffice.org`, then by passing different parameters (i.e. `user_u`), a different domain would be set up.

The main differences between an application interface and a template interface are:

- An access interface is called by other modules to perform a service.
- A template interface allows an application to be run in a domain based on user / role information to isolate different instances.

Note that the comments shown in the example **MUST** be present as they are used to describe the function and are extracted for the [documentation](#).

The macro definition is:

```
template(`name`, `template_rules`)
```

Where:

`template`

The template macro keyword.

name The template identifier that should be named to reflect the module identifier and its purpose. By convention the last component is `_template` (e.g. `ftp_per_role_template`).

template_rules This can consist of the support macros, policy language statements or interface calls as required to provide the service.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
No	Yes	No

Example Macro:

```
# This example is from the modules/apps/openoffice.if module
# showing the 'openoffice_per_role_template' template interface.
#
#####
## <summary>
## The per role template for the openoffice module.
## </summary>
## <desc>
## <p>
## This template creates a derived domains which are used
## for openoffice applications.
## </p>
## </desc>
## <param name="userdomain_prefix">
## <summary>
## The prefix of the user domain (e.g., user
## is the prefix for user_t).
## </summary>
## </param>
## <param name="user_domain">
## <summary>
## The type of the user domain.
## </summary>
## </param>
## <param name="user_role">
## <summary>
## The role associated with the user domain.
## </summary>
## </param>
#
template(`openoffice_per_role_template',`
    gen_require(`
        type openoffice_exec_t;
    ')

    type $1_openoffice_t;
    domain_type($1_openoffice_t)
    domain_entry_file($1_openoffice_t, openoffice_exec_t)
    role $3 types $1_openoffice_t;

    domain_interactive_fd($1_openoffice_t)

    userdom_unpriv_usertype($1, $1_openoffice_t)
    userdom_exec_user_home_content_files($1, $1_openoffice_t)

    allow $1_openoffice_t self:process { getsched sigkill execheap execmem
execstack };

```

```
allow $2 $1_openoffice_t:process { getattr ptrace signal_perms noatsecure
siginh rlimitinh };
allow $1_openoffice_t $2:tcp_socket { read write };

domtrans_pattern($2, openoffice_exec_t, $1_openoffice_t)

dev_read_urand($1_openoffice_t)
dev_read_rand($1_openoffice_t)

fs_dontaudit_rw_tmpfs_files($1_openoffice_t)

allow $2 $1_openoffice_t:process { signal sigkill };
allow $1_openoffice_t $2:unix_stream_socket connectto;
')
```

Expanded Macro:

```
# Template Interfaces are only expanded at policy compile time
# if they are called by a module that requires their services.
# This has been expanded as a part of the roles/xguest.te
# module and extracted from tmp/xguest.tmp.
#
##### START Expanded code segment #####
#
optional {

##### begin openoffice_per_role_template(xguest,xguest_usertype,xguest_r)
require {
    type openoffice_exec_t;
} # end require
type xguest_openoffice_t; # Parameter $1

.....
# This is a long set of rules, therefore has been cut down.
.....
....
typeattribute xguest_openoffice_t xguest_usertype; # Parameter $2
..
type_transition xguest_usertype openoffice_exec_t:process xguest_openoffice_t;
..
role xguest_r types xguest_openoffice_t; # Parameter $3
....
allow xguest_usertype xguest_openoffice_t:process { signal sigkill };
allow xguest_openoffice_t xguest_usertype:unix_stream_socket connectto;
##### end openoffice_per_role_template(xguest,xguest_usertype,xguest_r)

} # end optional
```

7.6.2 Miscellaneous Macros

These macros are in the `misc_macros.spt` file.

7.6.2.1 gen_context Macro

This macro is used to generate a valid security context and can be used in any of the module files. Its most general use is in the `.fc` file where it is used to set the files security context.

The macro definition is:

```
gen_context(context[,mls | mcs])
```

Where:

gen_context The gen_context macro keyword.

context The security context to be generated. This can include macros that are relevant to a context as shown in the example below.

mls | mcs MLS or MCS labels if enabled in the policy.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	Yes	Yes

Example Macro:

```
# This example shows gen_context being used to generate a
# security context for the security initial sid in the
# selinux.te module:

sid security gen_context(system_u:object_r:security_t:mls_systemhigh)
```

Expanded Macro:

```
# This is the expanded entry built into the base.conf source
# file for an MLS policy:

sid security system_u:object_r:security_t:s15:c0.c255
```

Example File Context .fc file:

```
# This is from the modules/apps/gnome.fc file. Note that the
# HOME_DIR and USER parameters will be entered during
# the file\_contexts.homedirs file build as described in the
# modules/active/file\_contexts.template File section.
#

HOME_DIR/.gnome2(/.*)?
    gen_context(system_u:object_r:gnome_home_t,s0)
HOME_DIR/\.config/gtk-.*
    gen_context(system_u:object_r:gnome_home_t,s0)
HOME_DIR/\.gconf(d)?(/.*)?
    gen_context(system_u:object_r:gconf_home_t,s0)
HOME_DIR/\.local.*
    gen_context(system_u:object_r:gconf_home_t,s0)

/tmp/gconfd-USER/.* --
    gen_context(system_u:object_r:gconf_tmp_t,s0)

HOME_DIR/.pulse(/.*)?
    gen_context(system_u:object_r:gnome_home_t,s0)
```

Expanded File Context .fc file:

```
# The resulting expanded tmp/gnome.mod.fc file. This will be
# concatenated with the main file_contexts file during the
# policy build process.
#
HOME_DIR/.gnome2(/.*)?      system_u:object_r:gnome_home_t:s0
HOME_DIR/.config/gtk-.*    system_u:object_r:gnome_home_t:s0
HOME_DIR/.gconf(d)?(/.*)? system_u:object_r:gconf_home_t:s0
HOME_DIR/.local.*         system_u:object_r:gconf_home_t:s0

/tmp/gconfd-USER/.* -- system_u:object_r:gconf_tmp_t:s0

HOME_DIR/.pulse(/.*)?     system_u:object_r:gnome_home_t:s0
```

7.6.2.2 gen_user Macro

This macro is used to generate a valid [user statement](#) and add an entry in the [users_extra](#) configuration file if it exists.

The macro definition is:

```
gen_user(username, prefix, role_set, mls_defaultlevel,
mls_range, [mcs_categories])
```

Where:

gen_user	The gen_user macro keyword.
username	The SELinux user id.
prefix	SELinux users without the prefix will not be in the users_extra file. This is added to user directories by the genhomedircon as discussed in the modules/active/file_contexts.template File section.
role_set	The user roles.
mls_defaultlevel	The default level if MLS / MCS policy.
mls_range	The range if MLS / MCS policy.
mcs_categories	The categories if MLS / MCS policy.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	No	No

Example Macro:

```
# This example has been taken from the policy/policy/users file:
#
```

```
gen_user(root, user, unconfined_r sysadm_r staff_r
ifdef(`enable_mls', `secadm_r auditadm_r') system_r, s0, s0 -
mls_systemhigh, mcs_allcats)
```

Expanded Macro:

```
# The expanded gen_user macro from the base.conf for an MLS
# build. Note that the prefix is not present. This is added to
# the users_extra file as shown below.
#
user root roles { unconfined_r sysadm_r staff_r secadm_r
auditadm_r system_r } level s0 range s0 - s15:c0.c1023;
```

```
# users_extra file entry:
#
user root prefix user;
```

7.6.2.3 gen_bool Macro

This macro defines a boolean and requires the following steps:

1. Declare the [boolean](#) in the [global_booleans](#) file.
2. Use the boolean in the module files with an [if / else statement](#) as shown in the example.

Note that the comments shown in the example **MUST** be present as they are used to describe the function and are extracted for the [documentation](#).

The macro definition is:

```
gen_bool (name, default_value)
```

Where:

gen_bool	The gen_bool macro keyword.
name	The boolean identifier.
default_value	The value true or false.

The macro is only valid in in the `global_booleans` file but the boolean declared can be used in the following module types:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	Yes	No

Example Macro (in `global_booleans`):

```
# This example is from the global_booleans file where the bool
# is declared. The comments must be present as it is used to
# generate the documentation.
```

```
#  
  
## <desc>  
## <p>  
## Disable transitions to insmod.  
## </p>  
## </desc>  
gen_bool(secure_mode_insmod,false)
```

```
# Example usage from the system/modutils.te module:  
#  
if( ! secure_mode_insmod ) {  
    kernel_domtrans_to(insmod_t,insmod_exec_t)  
}
```

Expanded Macro:

```
# This has been taken from the base.conf source file after  
# expansion by the build process of the modutils.te module.  
#  
if( ! secure_mode_insmod ) {  
##### begin kernel_domtrans_to(insmod_t,insmod_exec_t)  
    allow kernel_t insmod_exec_t:file { getattr read execute };  
    allow kernel_t insmod_t:process transition;  
    dontaudit kernel_t insmod_t:process { noatsecure siginh  
rlimitinh };  
    type_transition kernel_t insmod_exec_t:process insmod_t;  
    allow insmod_t kernel_t:fd use;  
    allow insmod_t kernel_t:fifo_file { getattr read write append  
ioctl lock };  
    allow insmod_t kernel_t:process sigchld;  
##### end kernel_domtrans_to(insmod_t,insmod_exec_t)  
}
```

7.6.3 MLS and MCS Macros

These macros are in the `mls_mcs_macros.spt` file.

7.6.3.1 gen_cats Macro

This macro will generate a [category statement](#) for each category defined. These are then used in the `base.conf` / `policy.conf` source file and also inserted into each module by the [policy module Macro](#). The `policy/policy/mcs` and `mls` configuration files are the only files that contain this macro in the current reference policy.

The macro definition is:

```
gen_cats(mcs_num_cats | mls_num_cats)
```

Where:

`gen_cats`

The `gen_cats` macro keyword.

mcs_num_cats These are the maximum number of categories
mls_num_cats that have been extracted from the build.conf
 file MCS_CATS or MLS_CATS entries and set as
 m4 parameters.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
na	na	na

Example Macro:

```
# This example is from the policy/policy/mls configuration file.  
#  
gen_cats(mls_num_cats)
```

Expanded Macro:

```
# This example has been extracted from the base.conf source  
# file.  
  
category c0;  
category c1;  
...  
category c1023;
```

7.6.3.2 gen_sens Macro

This macro will generate a [sensitivity statement](#) for each sensitivity defined. These are then used in the base.conf / policy.conf source file and also inserted into each module by the [policy_module Macro](#). The policy/policy/mcs and mls configuration files are the only files that contain this macro in the current reference policy (note that the mcs file has gen_sens (1) as only one sensitivity is required).

The macro definition is:

```
gen_sens(mls_num_sens)
```

Where:

gen_sens The gen_sens macro keyword.
mls_num_sens These are the maximum number of sensitivities
 that have been extracted from the build.conf
 file MLS_SENS entries and set as an m4
 parameter.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
na	na	na

Example Macro:

```
# This example is from the policy/policy/mls configuration file.
#
gen_cats(mls_num_sens)
```

Expanded Macro:

```
# This example has been extracted from the base.conf source
# file.

sensitivity s0;
sensitivity s1;
...
sensitivity s15;
```

7.6.3.3 gen_levels Macro

This macro will generate a [level statement](#) for each level defined. These are then used in the `base.conf / policy.conf` source file. The `policy/policy/mcs` and `mls` configuration files are the only files that contain this macro in the current reference policy.

The macro definition is:

```
gen_levels(mls_num_sens,mls_num_cats)
```

Where:

- `gen_levels` The `gen_levels` macro keyword.
- `mls_num_sens` This is the parameter that defines the number of sensitivities to generate. The MCS policy is set to '1'.
- `mls_num_cats` This is the parameter that defines the number of categories to generate.
- `mcs_num_cats`

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
na	na	na

Example Macro:

```
# This example is from the policy/policy/mls configuration file.
#
```

```
gen_levels(mls_num_sens,mls_num_cats)
```

Expanded Macro:

```
# This example has been extracted from the base.conf source
# file. Note that the all categories are allocated to each
# sensitivity.

level s0:c0.c1023;
level s1:c0.c1023;
...
level s15:c0.c1023;
```

7.6.3.4 System High/Low Parameters

These macros define system high etc. as shown.

```
mls_systemlow
# gives:
s0

mls_systemhigh
# gives:
s15:c0.c1023

mcs_systemlow
# gives:
s0

mcs_systemhigh
# gives:
s0:c0.c1023

mcs_allcats
# gives:
c0.c1023
```

7.6.4 `ifdef` / `ifndef` Parameters

This section contains examples of the common `ifdef` / `ifndef` parameters that can be used in module source files.

7.6.4.1 `hide_broken_symptoms`

This is used within modules as shown in the example. The parameter is set up by the Makefile at the start of the build process.

Example Macro:

```
# This example is from the modules/kernel/domain.te module.
#
ifdef(`hide_broken_symptoms',`
    cron_dontaudit_rw_tcp_sockets(domain)
```

```
allow domain domain:key { link search };
')
```

7.6.4.2 enable_mls and enable_mcs

These are used within modules as shown in the example. The parameters are set up by the Makefile with information taken from the build.conf file at the start of the build process.

Example Macros:

```
# This example is from the modules/kernel/kernel.te module.
#
ifdef(`enable_mls',`
    role secadm_r;
    role auditadm_r;
')
```

```
# This example is from the modules/kernel/kernel.if module.
#
ifdef(`enable_mcs',`
    range_transition kernel_t $2:process $3;
    ')

ifdef(`enable_mls',`
    range_transition kernel_t $2:process $3;
    mls_rangetrans_target($1)
    ')
```

7.6.4.3 direct_sysadm_daemon

This is used within modules as shown in the example. The parameter is set up by the Makefile with information taken from the build.conf file at the start of the build process (if DIRECT_INITRC = y).

Example Macros:

```
# This example is from the modules/system/selinuxutil.te module.
#
ifndef(`direct_sysadm_daemon',`
    ifdef(`distro_gentoo',`
        # Gentoo integrated run_init:
        init_script_file_entry_type(run_init_t)
    ')
')
```

```
# This example is from the modules/system/userdomain.te module.
#
ifdef(`direct_sysadm_daemon',`
    domain_system_change_exemption($1_t)
    ')
```


7.7 Module Expansion Process

The objective of this section is to show how the modules are expanded by the reference policy build process to form files that can then be compiled and then loaded into the policy store by using the `make MODULENAME.pp` target.

The files shown are those produced by the build process using the `ada` policy modules from the Reference Policy source tree (`ada.te`, `ada.if` and `ada.fc`) that are shown in the [Reference Policy Module Files](#) section.

The initial build process will build the source text files in the `policy/tmp` directory as `ada.tmp` and `ada.mod.fc` (that are basically build equivalent `ada.conf` and `ada.fc` formatted files). The basic steps are shown in [Figure 7-39](#), and the resulting expanded code shown in [Figure 7-40](#) and then described in the [Module Expansion](#) section.

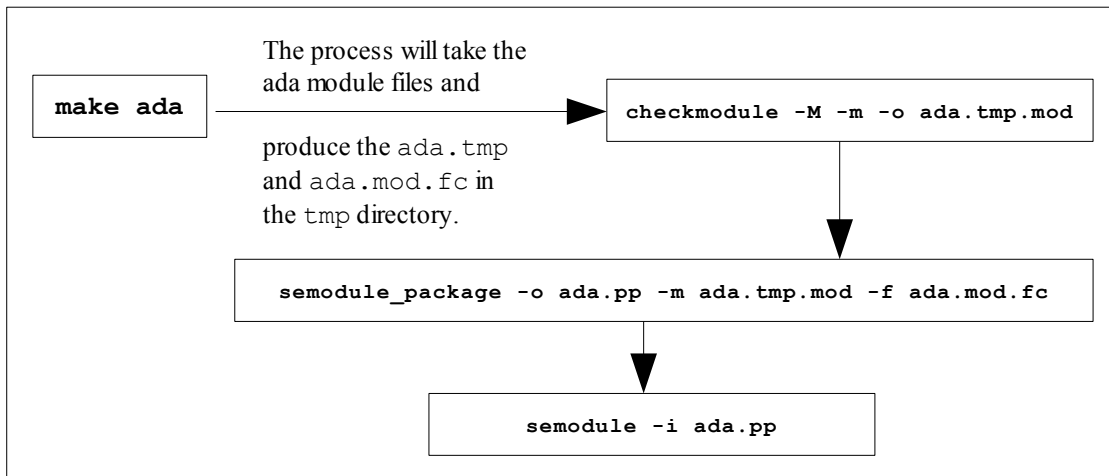


Figure 7-39: The `make ada` sequence of events

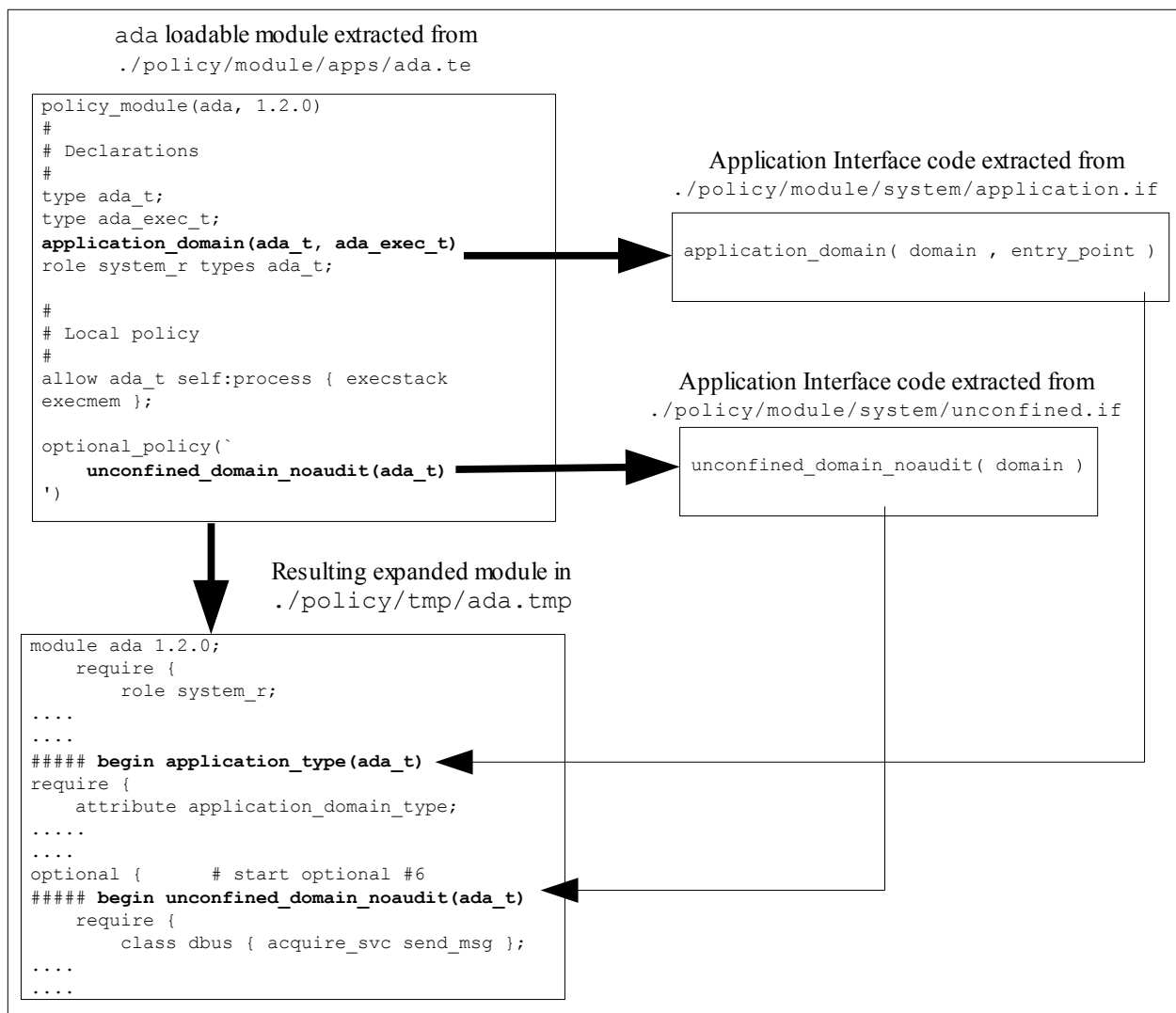


Figure 7-40: The Resulting Code - The expanded code in the [Module Expansion](#) section.

7.7.1 Module Expansion

The ada.te module is expanded as shown below. Note that the ada.if module is not expanded here. The ada.if module would only be expanded if another module calls these interfaces, where they would then be expanded into the calling module.

```

#
#####
#
# This is the start of the ada.te file that is expanded by the build
# process. Note the following:
#
# 1) The macros have been expanded to transform the 'ada.te' file into an
# 'ada.conf' file.
#
# 2) The 'ada.if' file Application Interface calls have NOT been expanded
# into this file simply because the ada.te does not call them (but
# they would be expanded into any other policy module that called them).
#
# 3) The module calls the "application_domain(ada_t,ada_exec_t)" that is
# one of the mandatory modules that MUST be included in the base.
# Note that this then calls other Application Interface macros.
#
#
    
```

```
# 4) All the build information that was in the original ada.tmp build      #
#   file have been removed for readability.                             #
#                                                                           #
#####
#
# The "policy_module(ada, 1.2.0)" macro has been expanded by the build process
# and the predefined 'require { }' entries are added (system_r role, all
# kernel classes and the sensitivity and category statements).
#
module ada 1.2.0;
  require {
    role system_r;
# These are the kernel class statements. There are many of them, therefore most
# have been removed for readability.
#
    class security { compute_av compute_create compute_member check_context
load_policy compute_relabel compute_user setenforce setbool setseccparam
setcheckreqprot };
    class peer { recv };
    class capability2 { mac_override mac_admin };
# End of classes

# As this is built as an MCS policy, there is only one sensitivity.
    sensitivity s0;

    category c0;
# This would contain many lines, one for each of the 1024 category statements
# defined in the policy. These have been removed for clarity.
    category c1023;
  } # END REQUIRE

#####
#
# Declarations
#

type ada_t;
type ada_exec_t;

#
#####
#
# This is the "application_domain(ada_t, ada_exec_t)" Application Interface
# call that is expanded to policy language statements by the build process.
# Note that there are many 'optional { }' statements, these have been
# marked numbered for easy reference.
#
##### START application_domain(ada_t,ada_exec_t) SEQUENCE #####
#
#### begin application_type(ada_t)
require {
  attribute application_domain_type;
} # end require

typeattribute ada_t application_domain_type;

#### begin domain_type(ada_t)
#### begin domain_base_type(ada_t)
require {
  attribute domain;
} # end require

typeattribute ada_t domain;
##### end domain_base_type(ada_t)

optional { # start optional #1
  #### begin unconfined_use_fds(ada_t)
  require {
    type unconfined_t;
  } # end require

  allow ada_t unconfined_t:fd use;
  ##### end unconfined_use_fds(ada_t)
```

```
} # end optional #1

# send init a sigchld and signull
optional { # start optional #2
##### begin init_sigchld(ada_t)
    require {
        type init_t;
    } # end require
    allow ada_t init_t:process sigchld;
##### end init_sigchld(ada_t)

##### begin init_signull(ada_t)
    require {
        type init_t;
    } # end require

    allow ada_t init_t:process signull;
#### end init_signull(ada_t)
} # end optional #2

# these seem questionable:
optional { # start optional #3
##### begin rpm_use_fds(ada_t)
    require {
        type rpm_t;
    } # end require

    allow ada_t rpm_t:fd use;
##### end rpm_use_fds(ada_t)

##### begin rpm_read_pipes(ada_t)
    require {
        type rpm_t;
    } # end require

    allow ada_t rpm_t:fifo_file { getattr read lock ioctl };
##### end rpm_read_pipes(ada_t)
} # end optional #3

optional { # start optional #4
##### begin selinux_dontaudit_getattr_fs(ada_t)
    require {
        type security_t;
    } # end require

    dontaudit ada_t security_t:filesystem getattr;
##### end selinux_dontaudit_getattr_fs(ada_t)

##### begin selinux_dontaudit_read_fs(ada_t)
    require {
        type security_t;
    } # end require

##### begin selinux_dontaudit_getattr_fs(ada_t)
    require {
        type security_t;
    } # end require

    dontaudit ada_t security_t:filesystem getattr;
##### end selinux_dontaudit_getattr_fs(ada_t)
    dontaudit ada_t security_t:dir { getattr search };
    dontaudit ada_t security_t:file { getattr read };
##### end selinux_dontaudit_read_fs(ada_t)
} # end optional #4

optional { # start optional #5
##### begin seutil_dontaudit_read_config(ada_t)
    require {
        type selinux_config_t;
    } # end require
    dontaudit ada_t selinux_config_t:dir { getattr search };
    dontaudit ada_t selinux_config_t:file { getattr read };

```

```
##### end seutil_dontaudit_read_config(ada_t)
} # end optional #5

##### end domain_type(ada_t)
##### end application_type(ada_t)

##### begin application_executable_file(ada_exec_t)
require {
    attribute application_exec_type;
} # end require

typeattribute ada_exec_t application_exec_type;

##### begin corecmd_executable_file(ada_exec_t)
require {
    attribute exec_type;
} # end require

typeattribute ada_exec_t exec_type;

##### begin files_type(ada_exec_t)
require {
    attribute file_type, non_security_file_type;
} # end require

typeattribute ada_exec_t file_type, non_security_file_type;
##### end files_type(ada_exec_t)
##### end corecmd_executable_file(ada_exec_t)
##### end application_executable_file(ada_exec_t)

##### begin domain_entry_file(ada_t,ada_exec_t)
require {
    attribute entry_type;
} # end require
allow ada_t ada_exec_t:file entrypoint;
allow ada_t ada_exec_t:file { { getattr read execute ioctl } ioctl lock };
typeattribute ada_exec_t entry_type;

##### begin corecmd_executable_file(ada_exec_t)
require {
    attribute exec_type;
} # end require
typeattribute ada_exec_t exec_type;

##### begin files_type(ada_exec_t)
require {
    attribute file_type, non_security_file_type;
} # end require
typeattribute ada_exec_t file_type, non_security_file_type;
##### end files_type(ada_exec_t) depth: 3
##### end corecmd_executable_file(ada_exec_t)
##### end domain_entry_file(ada_t,ada_exec_t)
##### end application_domain(ada_t,ada_exec_t)

#
##### END application_domain(ada_t,ada_exec_t) INSERT #####
#

role system_r types ada_t;

#####
#
# Local policy
#

# This is the only allow statement in the ada.te file:
allow ada_t self:process { execstack execmem };

#
# The "optional_policy('unconfined_domain_noaudit(ada_t)')" is the next
# line that expands into the lines between START and END OPTIONAL comments.
# NOTE: If the unconfined module was NOT part of the build then this optional
# policy section would not be present.
#
##### START OPTIONAL_POLICY STATEMENT #####
```

```
optional { # start optional #6
##### begin unconfined_domain_noaudit(ada_t)
    require {
        class dbus { acquire_svc send_msg };
        class nscd { getpwd getgrp gethost getstat admin shmempwd shmemgrp
shmehost getserv shmemserv };
        class passwd { passwd chfn chsh rootok crontab };
    } # end require

# Use any Linux capability.
    allow ada_t self:capability { chown dac_override dac_read_search fowner fsetid
kill setgid setuid setpcap linux_immutable net_bind_service net_broadcast
net_admin net_raw ipc_lock ipc_owner sys_module sys_rawio sys_chroot sys_ptrace
sys_pacct sys_admin sys_boot sys_nice sys_resource sys_time sys_tty_config mknod
lease audit_write audit_control setfcap };

    allow ada_t self:fifo_file { create open getattr setattr read write append
rename link unlink ioctl lock };

# Transition to myself, to make get_ordered_context_list happy.
    allow ada_t self:process transition;
# Write access is for setting attributes under /proc/self/attr.
    allow ada_t self:file { getattr read write append ioctl lock };
    allow ada_t self:dir { read getattr lock search ioctl add_name remove_name
write };
# Userland object managers
    allow ada_t self:nscd { getpwd getgrp gethost getstat admin shmempwd shmemgrp
shmehost getserv shmemserv };
    allow ada_t self:dbus { acquire_svc send_msg };
    allow ada_t self:passwd { passwd chfn chsh rootok crontab };
    allow ada_t self:association { sendto recvfrom setcontext polmatch };
##### begin kernel_unconfined(ada_t)
    require {
        attribute kern_unconfined;
    } # end require

    typeattribute ada_t kern_unconfined;
##### end kernel_unconfined(ada_t)

##### begin corenet_unconfined(ada_t)
    require { attribute corenet_unconfined_type;
    } # end require

    typeattribute ada_t corenet_unconfined_type;
##### end corenet_unconfined(ada_t)

##### begin dev_unconfined(ada_t)
    require {
        attribute devices_unconfined_type;
    } # end require

    typeattribute ada_t devices_unconfined_type;
##### end dev_unconfined(ada_t)

##### begin domain_unconfined(ada_t)
    require {
        attribute set_curr_context;
        attribute can_change_object_identity;
        attribute unconfined_domain_type;
        attribute process_uncond_exempt;
    } # end require

    typeattribute ada_t unconfined_domain_type;
# pass constraints
    typeattribute ada_t can_change_object_identity;
    typeattribute ada_t set_curr_context;
    typeattribute ada_t process_uncond_exempt;
##### end domain_unconfined(ada_t)

##### begin domain_dontaudit_read_all_domains_state(ada_t)
    require {
        attribute domain;
    } # end require
```

```
dontaudit ada_t domain:dir { getattr search read lock ioctl };
dontaudit ada_t domain:lnk_file { getattr read };
dontaudit ada_t domain:file { getattr read lock ioctl };
# cjp: these should be removed:
dontaudit ada_t domain:sock_file { getattr read };dontaudit ada_t
domain:fifo_file { getattr read lock ioctl };
##### end domain_dontaudit_read_all_domains_state(ada_t)

##### begin domain_dontaudit_ptrace_all_domains(ada_t)
require {
    attribute domain;
} # end require

dontaudit ada_t domain:process ptrace;
##### end domain_dontaudit_ptrace_all_domains(ada_t)

##### begin files_unconfined(ada_t)
require {
    attribute files_unconfined_type;
} # end require

typeattribute ada_t files_unconfined_type;
##### end files_unconfined(ada_t)

##### begin fs_unconfined(ada_t)
require {
    attribute filesystem_unconfined_type;
} # end require

typeattribute ada_t filesystem_unconfined_type;
##### end fs_unconfined(ada_t)

##### begin selinux_unconfined(ada_t)
require {
    attribute selinux_unconfined_type;
} # end require

typeattribute ada_t selinux_unconfined_type;
##### end selinux_unconfined(ada_t)

##### begin domain_mmap_low_type(ada_t)
require {
    attribute mmap_low_domain_type;
} # end require

typeattribute ada_t mmap_low_domain_type;
##### end domain_mmap_low_type(ada_t)

require {
    bool allow_unconfined_mmap_low;
} # end require
if (allow_unconfined_mmap_low) {
##### begin domain_mmap_low(ada_t)
    allow ada_t self:memprotect mmap_zero;
##### end domain_mmap_low(ada_t)
}

require {
    bool allow_execheap;
} # end require

if (allow_execheap) {
# Allow making the stack executable via mprotect.
    allow ada_t self:process execheap;
}

require {
    bool allow_execmem;
} # end require

if (allow_execmem) {
# Allow making anonymous memory executable, e.g.
# for runtime-code generation or executable stack.
    allow ada_t self:process execmem;
}
}
```

```
require {
    bool allow_execstack;
} # end require

if (allow_execstack) {
    # Allow making the stack executable via mprotect;
    # execstack implies execmem;
    allow ada_t self:process { execstack execmem };
    #
    auditallow ada_t self:process execstack;}

optional { # start optional #7
##### begin auth_unconfined(ada_t)
    require {
        attribute can_read_shadow_passwords;
        attribute can_write_shadow_passwords;
        attribute can_relabelto_shadow_passwords;
    } # end require

typeattribute ada_t can_read_shadow_passwords;typeattribute ada_t
can_write_shadow_passwords;typeattribute ada_t can_relabelto_shadow_passwords;
##### end auth_unconfined(ada_t) depth: 1
} # end optional #7

optional { # start optional #8
# Communicate via dbus.
##### begin dbus_system_bus_unconfined(ada_t)
    require {
        type system_dbusd_t;
        class dbus { acquire_svc send_msg };
    } # end require

    allow ada_t system_dbusd_t:dbus *;
##### end dbus_system_bus_unconfined(ada_t)
##### begin dbus_unconfined(ada_t) depth: 2
    require {
        attribute dbusd_unconfined;
    } # end require

    typeattribute ada_t dbusd_unconfined;
##### end dbus_unconfined(ada_t)
} # end optional #8

optional {# start optional #9
##### begin ipsec_setcontext_default_spd(ada_t)
    require {
        type ipsec_spd_t;
    } # end require

    allow ada_t ipsec_spd_t:association setcontext;
##### end ipsec_setcontext_default_spd(ada_t)

##### begin ipsec_match_default_spd(ada_t)
    require { type ipsec_spd_t;
    } # end require

    allow ada_t ipsec_spd_t:association polmatch;
    allow ada_t self:association sendto;
##### end ipsec_match_default_spd(ada_t)
} # end optional #9

optional { # start optional #10
# this is to handle execmod on shared
# libs with text relocations
##### begin libs_use_shared_libs(ada_t)
    require { type lib_t, textrel_shlib_t;
    } # end require
##### begin files_list_usr(ada_t)
    require {
        type usr_t;
    } # end require

    allow ada_t usr_t:dir { getattr search read lock ioctl };
```



```
##### end files_list_usr(ada_t)

    allow ada_t lib_t:dir { getattr search read lock ioctl };
    allow ada_t lib_t:dir { getattr search };allow ada_t { lib_t
textrel_shlib_t }:lnk_file { getattr read };
    allow ada_t lib_t:dir { getattr search };allow ada_t { lib_t
textrel_shlib_t }:file { getattr read execute ioctl };
    allow ada_t textrel_shlib_t:file execmod;
##### end libs_use_shared_libs(ada_t)
} # end optional #10

optional { # start optional #11
##### begin nscd_unconfined(ada_t)
    require {
        type nscd_t;
        class nscd { getpwd getgrp gethost getstat admin shmempwd shmemgrp
shmehost getserv shmemserv };
    } # end require

    allow ada_t nscd_t:nscd *;
##### end nscd_unconfined(ada_t)
} # end optional #11

optional { # start optional #12
##### begin postgresql_unconfined(ada_t)
    require {
        attribute sepgsql_unconfined_type;
    } # end require

    typeattribute ada_t sepgsql_unconfined_type;
##### end postgresql_unconfined(ada_t)
} # end optional #12

optional { # start optional #13
##### begin seutil_create_bin_policy(ada_t)
    require {
#
        attribute can_write_binary_policy;
        type selinux_config_t, policy_config_t;
    } # end require

##### begin files_search_etc(ada_t)
    require {
        type etc_t;
    } # end require

    allow ada_t etc_t:dir { getattr search };
##### end files_search_etc(ada_t)
    allow ada_t selinux_config_t:dir { getattr search };
    allow ada_t policy_config_t:dir { getattr search lock ioctl write add_name
};
    allow ada_t policy_config_t:file { getattr create open };
    allow ada_t policy_config_t:dir { getattr search };
    allow ada_t policy_config_t:file { getattr write append lock ioctl };

    # typeattribute ada_t can_write_binary_policy;
##### end seutil_create_bin_policy(ada_t)

##### begin seutil_relabelto_bin_policy(ada_t)

    require {
        attribute can_relabelto_binary_policy;    type policy_config_t;
    } # end require
    allow ada_t policy_config_t:file relabelto;typeattribute ada_t
can_relabelto_binary_policy;
##### end seutil_relabelto_bin_policy(ada_t)
} # end optional #13

optional { # start optional #14
##### begin storage_unconfined(ada_t)
    require {
        attribute storage_unconfined_type;
    } # end require

    typeattribute ada_t storage_unconfined_type;
```

```
##### end storage_unconfined(ada_t)
} # end optional #14

optional { # start optional #15
##### begin xserver_unconfined(ada_t)
    require {
        attribute xserver_unconfined_type, x_domain;
    } # end require
    typeattribute ada_t xserver_unconfined_type, x_domain;
##### end xserver_unconfined(ada_t)
} # end optional #15
##### end unconfined_domain_noaudit(ada_t)
} # end optional #6

##### END OPTIONAL_POLICY STATEMENT #####
#
```

7.7.2 File Context Expansion

As can be seen the `gen_context` macro has been expanded to build the security context:

```
#
# /usr
#
/usr/bin/gnatbind -- system_u:object_r:ada_exec_t:s0
/usr/bin/gnatls -- system_u:object_r:ada_exec_t:s0
/usr/bin/gnatmake -- system_u:object_r:ada_exec_t:s0
/usr/libexec/gcc(/.*)?/gnat1 -- system_u:object_r:ada_exec_t:s0
```

8. Appendix A - Object Classes and Permissions

8.1 Introduction

This section contains a list of object classes and their associated permissions that have been taken from the Fedora F-10 policy sources.

All objects are kernel objects unless marked as user space objects.

Deprecated permissions and objects still in the policy sources are shown in the tables as italic text.

The deprecated permissions are:

transition, swapon, enforce_dest, connectto, acceptfrom and newconn

The deprecated object is:

ipc

In most cases the permissions are self explanatory as they are those used in the standard Linux function calls (such as ‘create a socket’ or ‘write to a file’). The SELinux specific permissions are:

<code>relabelfrom</code>	Used on most objects to allow the objects security context to be changed from the current type.
<code>relabelto</code>	Used on most objects to allow the objects security context to be changed to the new type.
<code>entrypoint</code>	Used for files to indicate that they can be used as an entry point into a domain via a domain transition.
<code>execute_no_trans</code>	Used for files to indicate that they can be used as an entry point into the calling domain (i.e. does not require a domain transition).
<code>execmod</code>	Generally used for files to indicate that they can execute the modified file in memory.

Where possible the specific object class permissions are explained, however for some permissions it is difficult to determine what they are used for (or if used at all) so a ‘?’ has been added when doubt exists. There are lists of object classes and permissions at the following locations and would probably be more up-to-date:

<http://oss.tresys.com/projects/refpolicy/wiki/ObjectClassesPerms>

<http://selinuxproject.org/page/ObjectClassesPerms>

8.2 Defining Object Classes and Permissions

The Fedora [Reference Policy](#) (and the basic policy explained in the [Building a Basic Policy](#) section) already contain the default object classes and permissions required to manage the system and supporting services.

For those who write or manager SELinux policy, there is no need to define new objects and their associated permissions as these would be done by those who actually design and/or write object managers.

The [Object Classes](#) and [Permissions](#) sections explain how these are defined within the [SELinux Policy Language](#).

The sections that follow contain the entries defined in the F-10 `./policy/flask/access_vectors` file of the [Reference Policy](#) source.

8.3 Common Permissions

8.3.1 Common File Permissions

[Table 8-1](#) describes the common file permissions that are inherited by a number of object classes.

Permissions	Description (17 permissions)
append	Append to file.
create	Create new file.
execute	Execute the file with domain transition.
getattr	Get file attributes.
ioctl	I/O control system call requests.
link	Create hard link.
lock	Set and unset file locks.
mounton	Use as mount point.
quotaon	Enable quotas.
read	Read file contents.
relabelfrom	Change the security context based on existing type.
relabelto	Change the security context based on the new type.
rename	Rename file.
setattr	Change file attributes.
swapon	Allow file to be used for paging / swapping space. (not used ?)
unlink	Delete file (or remove hard link).
write	Write or append file contents.

Table 8-1: Common File Permissions

8.3.2 Common Socket Permissions

[Table 8-2](#) describes the common socket permissions that are inherited by a number of object classes.

Permissions	Description (22 Permissions)
accept	Accept a connection.
append	Write or append socket contents.
bind	Bind to a name.
connect	Initiate a connection.

create	Create new socket.
getattr	Get socket information.
getopt	Get socket options.
ioctl	Get and set attributes via <code>ioctl</code> call requests.
listen	Listen for connections.
lock	Lock and unlock socket file descriptor.
name_bind	AF_INET - Controls relationship between a socket and the port number. AF_UNIX - Controls relationship between a socket and the file.
read	Read data from socket.
recv_msg	Receive datagram.
recvfrom	Receive datagrams from socket.
relabelfrom	Change the security context based on existing type.
relabelto	Change the security context based on the new type.
send_msg	Send datagram.
sendto	Send datagrams to socket.
setattr	Change attributes.
setopt	Set socket options.
shutdown	Terminate connection.
write	Write data to socket.

Table 8-2: Common Socket Permissions

8.3.3 Common IPC Permissions

[Table 8-3](#) describes the common IPC permissions that are inherited by a number of object classes.

Permissions	Description (9 Permissions)
associate	shm – Get shared memory ID. msgq – Get message ID. sem – Get semaphore ID.
create	Create.
destroy	Destroy.
getattr	Get information from IPC object.
read	shm – Attach shared memory to process. msgq – Read message from queue. sem – Get semaphore value.
setattr	Set IPC object information.
unix_read	Read.
unix_write	Write or append.
write	shm – Attach shared memory to process. msgq – Send message to message queue. sem – Change semaphore value.

Table 8-3: Common IPC Permissions

8.3.4 Common Database Permissions

[Table 8-4](#) describes the common database permissions that are inherited by a number of object classes. The “[Security-Enhanced PostgreSQL Security Wiki](#)” [Ref. 3] explains the objects, their permissions and how they should be used in detail.

Permissions	Description (6 Permissions)
create	Create a database object such as a ‘TABLE’.
drop	Delete a database object.
getattr	Get metadata – needed to reference an object.
relabelfrom	Change the security context based on existing type.
relabelto	Change the security context based on the new type.
setattr	Set metadata – this permission is required to update information in the database.

Table 8-4: Common PostgreSQL Database Permissions

To support security context information the following PostgreSQL statements have been extended:

```
CREATE DATABASE, ALTER DATABASE, CREATE TABLE, ALTER TABLE,
CREATE FUNCTION and ALTER FUNCTION.
```

Also three new PostgreSQL functions have been created:

```
sepgsql_getcon() – Obtain a clients security context.
lo_get_security() and lo_set_security() – Get and set security
context on binary large objects (blobs).
```

8.4 File Object Classes

Class	filesystem – A mounted filesystem
Permissions	Description (10 unique permissions)
associate	Use type as label for file.
getattr	Get file attributes.
mount	Mount filesystem.
quotaget	Get quota information.
quotamod	Modify quota information.
relabelfrom	Change the security context based on existing type.
relabelto	Change the security context based on the new type.
remount	Remount existing mount.
transition	Transition to a new SID (change security context).
unmount	Unmount filesystem.

The SELinux Notebook - The Foundations

Class	dir - Directory
Permissions	Description (Inherit 17 common file permissions + 6 unique)
Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
add_name	Add entry to the directory.
open	Added in 2.6.26 Kernel to control the open permission.
remove_name	Remove an entry from the directory.
reparent	Change parent directory.
rmdir	Remove directory.
search	Search directory.

Class	file - Ordinary file
Permissions	Description (Inherit 17 common file permissions + 4 unique)
Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
entrypoint	Entry point permission for a domain transition.
execmod	Make executable a file that has been modified by copy-on-write.
execute_no_trans	Execute in the caller's domain (i.e. no domain transition).
open	Added in 2.6.26 Kernel to control the open permission.

Class	lnk_file - Symbolic links
Permissions	Description (Inherit 17 common file permissions)
Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write

Class	chr_file - Character files
Permissions	Description (Inherit 17 common file permissions + 4 unique)
Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
entrypoint	Entry point permission for a domain transition.
execmod	Make executable a file that has been modified by copy-on-write.
execute_no_trans	Execute in the caller's domain (i.e. no domain transition).
open	Added in 2.6.26 Kernel to open a character device.

Class	blk_file - Block files
Permissions	Description (Inherit 17 common file permissions + 1 unique)
Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
open	Added in 2.6.26 Kernel to control the open permission.

The SELinux Notebook - The Foundations

Class	sock_file – UNIX domain sockets
Permissions	Description (Inherit 17 common file permissions)
Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write

Class	fifo_file – Named pipes
Permissions	Description (Inherit 17 common file permissions + 1 unique)
Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
open	Added in 2.6.26 Kernel to control the open permission.

Class	fd – File descriptors
Permissions	Description (1 unique permission)
use	1) Inherit fd when process is executed and domain has been changed. 2) Receive fd from another process by Unix domain socket. 3) Get and set attribute of fd.

8.5 Network Object Classes

Class	node – IP address or range of IP addresses
Permissions	Description (11 unique permissions)
dccp_recv	Allow Datagram Congestion Control Protocol receive packets.
dccp_send	Allow Datagram Congestion Control Protocol send packets.
enforce_dest	Ensure that destination node can enforce restrictions on the destination socket.
rawip_recv	Receive raw IP packet.
rawip_send	Send raw IP packet.
recvfrom	Network interface and address check permission for use with the ingress permission.
sendto	Network interface and address check permission for use with the egress permission.
tcp_recv	Receive TCP packet.
tcp_send	Send TCP packet.
udp_recv	Receive UDP packet.
udp_send	Send UDP packet.

Class	netif – Network Interface (e.g. eth0)
Permissions	Description (10 unique permissions)
dccp_recv	Allow Datagram Congestion Control Protocol receive packets.
dccp_send	Allow Datagram Congestion Control Protocol send packets.
egress	Each packet leaving the system must pass an egress access control. Also requires the node sendto permission.

The SELinux Notebook - The Foundations

ingress	Each packet entering the system must pass an <code>ingress</code> access control. Also requires the node <code>recvfrom</code> permission.
rawip_recv	Receive raw IP packet.
rawip_send	Send raw IP packet.
tcp_recv	Receive TCP packet.
tcp_send	Send TCP packet.
udp_recv	Receive UDP packet.
udp_send	Send UDP packet.

Class	socket – Socket that is not part of any other specific SELinux socket object class.
Permissions	Description (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	<code>accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write</code>

Class	tcp_socket – Protocol: <code>PF_INET, PF_INET6</code> Family Type: <code>SOCK_STREAM</code>
Permissions	Description (Inherit 22 common socket permissions + 5 unique)
Inherit Common Socket Permissions	<code>accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write</code>
<code>acceptfrom</code>	Accept connection from client socket.
<code>connectto</code>	Connect to server socket.
<code>name_connect</code>	Connect to a specific port type.
<code>newconn</code>	Create new connection.
<code>node_bind</code>	Bind to a node.

Class	udp_socket – Protocol: <code>PF_INET, PF_INET6</code> Family Type: <code>SOCK_DGRAM</code>
Permissions	Description (Inherit 22 common socket permissions + 1 unique)
Inherit Common Socket Permissions	<code>accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write</code>
<code>node_bind</code>	Bind to a node.

The SELinux Notebook - The Foundations

Class	rawip_socket – Protocol: PF_INET, PF_INET6 Family Type: SOCK_RAW
Permissions	Description (Inherit 22 common socket permissions + 1 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
node_bind	Bind to a node.

Class	packet_socket – Protocol: PF_PACKET Family Type: All.
Permissions	Description (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	unix_stream_socket – Communicate with processes on same machine. Protocol: PF_STREAM Family Type: SOCK_STREAM
Permissions	Description (Inherit 22 common socket permissions + 3 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
acceptfrom	Accept connection from client socket.
connectto	Connect to server socket.
newconn	Create new socket for connection.

Class	unix_dgram_socket – Communicate with processes on same machine. Protocol: PF_STREAM Family Type: SOCK_DGRAM
Permissions	Description (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

8.5.1 IPSec Network Object Classes

[Appendix E – Labeled IPSec Module Example](#) contains a worked IPSec example.

Class	association – IPSec security association
Permissions	Description (4 unique permissions)
polmatch	Match IPSec Security Policy Database (SPD) context (-ctx) entries to an SELinux domain (contained in the Security Association Database (SAD)).
recvfrom	Receive from an IPSec association.
sendto	Send to an IPSec association.

The SELinux Notebook - The Foundations

setcontext	Set the context of an IPSec association on creation.

Class	key_socket – IPSec key management. Protocol: PF_KEY Family Type: All
Permissions	Description (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	netlink_xfrm_socket - Netlink socket to maintain IPSec parameters.
Permissions	Description (Inherit 22 common socket permissions + 2 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
nlmsg_read	Get IPSec configuration information.
nlmsg_write	Set IPSec configuration information.

8.5.2 Netlink Object Classes

Netlink sockets communicate between userspace and the kernel.

Class	netlink_socket - Netlink socket that is not part of any specific SELinux Netlink socket class. Protocol: PF_NETLINK Family Type: All other types that are not part of any other specific netlink object class.
Permissions	Description (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	netlink_route_socket – Netlink socket to manage and control network resources.
Permissions	Description (Inherit 22 common socket permissions + 2 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
nlmsg_read	Read kernel routing table.
nlmsg_write	Write to kernel routing table.

The SELinux Notebook - The Foundations

Class	netlink_firewall_socket – Netlink socket for firewall filters.
Permissions	Description (Inherit 22 common socket permissions + 2 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
nlmsg_read	Read netlink message.
nlmsg_write	Write netlink message.

Class	netlink_tcpdiag_socket - Netlink socket to monitor TCP connections.
Permissions	Description (Inherit 22 common socket permissions + 2 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
nlmsg_read	Request information about a protocol.
nlmsg_write	Write netlink message.

Class	netlink_nflog_socket - Netlink socket for Netfilter logging
Permissions	Description (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	netlink_selinux_socket - Netlink socket to receive SELinux events such as a policy or boolean change.
Permissions	Description (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	netlink_audit_socket - Netlink socket for audit service.
Permissions	Description (Inherit 22 common socket permissions + 5 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
nlmsg_read	Query status of audit service.
nlmsg_readpriv	List auditing configuration rules.
nlmsg_relay	Send userspace audit messages to theaudit service.

The SELinux Notebook - The Foundations

nlmsg_tty_audit	Control TTY auditing.
nlmsg_write	Update audit service configuration.

Class	netlink_ip6fw_socket - Netlink socket for IPv6 firewall filters.
Permissions	Description (Inherit 22 common socket permissions + 2 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
nlmsg_read	Read netlink message.
nlmsg_write	Write netlink message.

Class	netlink_dnrt_socket - Netlink socket for DECnet routing
Permissions	Description (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	netlink_kobject_uevent_socket - Netlink socket to send kernel events to userspace.
Permissions	Description (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

8.5.3 Miscellaneous Network Object Classes

Class	peer - NetLabel and Labeled IPsec have separate access controls, the network peer label consolidates these two access controls into a single one (see http://paulmoore.livejournal.com/1863.html for details).
Permissions	Description (1 unique permission)
recv	Receive packets from a labeled networking peer.

Class	packet - Supports 'secmark' services where packets are labeled using iptables to select and label packets, SELinux then enforces policy using these packet labels.
Permissions	Description (7 unique permissions)
flow_in	Receive external packets. (deprecated)
flow_out	Send packets externally. (deprecated)
forward_in	Allow inbound forwarded packets.
forward_out	Allow outbound forwarded packets.
recv	Receive inbound locally consumed packets.

The SELinux Notebook - The Foundations

relabelto	Control how domains can apply specific labels to packets.
send	Send outbound locally generated packets.

Class	appletalk_socket - Appletalk socket
Permissions	Description (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	dccp_socket - Datagram Congestion Control Protocol (DCCP)
Permissions	Description (Inherit 22 common socket permissions + 2 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
name_connect	Allow DCCP name connect().
node_bind	Allow DCCP bind().

8.6 IPC Object Classes

Class	ipc – No longer used
Permissions	Description (Inherit 9 common IPC permissions)
Inherit Common IPC Permissions	associate, create, destroy, getattr, read, setattr, unix_read, unix_write, write

Class	sem - Semaphores
Permissions	Description (Inherit 9 common IPC permissions)
Inherit Common IPC Permissions	associate, create, destroy, getattr, read, setattr, unix_read, unix_write, write

Class	msgq – IPC Message queues
Permissions	Description (Inherit 9 common IPC permissions + 1 unique)
Inherit Common IPC Permissions	associate, create, destroy, getattr, read, setattr, unix_read, unix_write, write
enqueue	Send message to message queue.

Class	msg – Message in a queue
Permissions	Description (2 unique permissions)
receive	Read (and remove) message from queue.
send	Add message to queue.

Class	shm – Shared memory segment
Permissions	Description (Inherit 9 common IPC permissions + 1 unique)
Inherit Common IPC Permissions	associate, create, destroy, getattr, read, setattr, unix_read, unix_write, write
lock	Lock or unlock shared memory.

8.7 Process Object Class

Class	process – An object is instantiated for each process created by the system.
Permissions	Description (30 unique permissions)
dyntransition	Dynamically transition to a new context (domain).
execheap	Make the heap executable.
execmem	Make executable an anonymous mapping or private file mapping that is writable.
execstack	Make the main process stack executable.
fork	Create new process using fork(2).
getattr	Get process security information.
getcap	Get Linux capabilities of process.
getpgid	Get group Process ID of another process.
getsched	Get scheduling information of another process.
getsession	Get session ID of another process.
noatsecure	Disable secure mode environment cleansing.
ptrace	Trace program execution of parent or child (ptrace(2)).
rlimitinh	Inherit rlimit information from parent process.
setcap	Set Linux capabilities of process.
setcurrent	Set the current process context.
setexec	Set security context of executed process by setexecon() call.
setfscreate	Set security context by setfscreatecon() call.
setkeycreate	Set security context by setkeycreatecon() call.
setpgid	Set group Process ID of another process.
setrlimit	Change process rlimit information.
setsched	Modify scheduling information of another process.
setsockcreate	Set security context by setsockcreatecon() call.
share	Allow state sharing with cloned or forked process.
sigchld	Send SIGCHLD signal.
siginh	Inherit signal state from parent process.
sigkill	Send SIGKILL signal.
signal	Send a signal other than SIGKILL, SIGSTOP, or SIGCHLD.
signull	Test for existence of another process without sending a signal
sigstop	Send SIGSTOP signal
transition	Transition to a new context on exec().

8.8 Security Object Class

Class	security - This is the security server object and there is only one instance of this object (for the SELinux security server).
Permissions	Description (11 unique permissions)
check_context	Determine whether the context is valid by querying the security server.
compute_av	Compute an access vector given a source/target/class.
compute_create	Determine context to use when querying the security server about a transition rule (type_transition).
compute_member	Determine context to use when querying the security server about a membership decision (type_member for a polyinstantiated object).
compute_relabel	Determines the context to use when querying the security server about a relabeling decision (type_change).
compute_user	Determines the context to use when querying the security server about a user decision (user).
load_policy	Load the security policy into the kernel (the security server).
setbool	Change a boolean value within the active policy.
setcheckreqprot	Set if SELinux will check original protection mode or modified protection mode (read-implies-exec) for mmap / mprotect.
setenforce	Change the enforcement state of SELinux (permissive or enforcing).
setseccomparam	Set kernel access vector cache tuning parameters.

8.9 System Operation Object Class

Class	system - This is the overall system object and there is only one instance of this object.
Permissions	Description (4 unique permissions)
ipc_info	Get info about an IPC object.
syslog_console	Control output of kernel messages to the console with syslog(2).
syslog_mod	Clear kernel message buffer with syslog(2).
syslog_read	Read kernel message with syslog(2).

8.10 Capability Object Classes

Class	capability – Used to manage the Linux capabilities granted to root processes. Taken from the header file: /usr/include/linux/capability.h
Permissions	Description (32 unique permissions)
audit_control	Change auditing rules. Set login UID.
audit_write	Send audit messages from user space.
chown	Allow changing file and group ownership.
dac_override	Overrides all DAC including ACL execute access.
dac_read_search	Overrides DAC for read and directory search.
fowner	Grant all file operations otherwise restricted due to different ownership except where FSETID capability is applicable. DAC and MAC accesses are not overridden.
fsetid	Overrides the restriction that the real or effective user ID of a process

The SELinux Notebook - The Foundations

	sending a signal must match the real or effective user ID of the process receiving the signal.
ipc_lock	Grants the capability to lock non-shared and shared memory segments.
ipc_owner	Grant the ability to ignore IPC ownership checks.
kill	Allow signal raising for any process.
lease	Grants ability to take leases on a file.
linux_immutable	Grant privilege to modify S_IMMUTABLE and S_APPEND file attributes on supporting filesystems.
mknod	Grants permission to creation of character and block device nodes.
net_admin	Allow the following: interface configuration; administration of IP firewall; masquerading and accounting; setting debug option on sockets; modification of routing tables; setting arbitrary process / group ownership on sockets; binding to any address for transparent proxying; setting TOS (type of service); setting promiscuous mode; clearing driver statistics; multicasting; read/write of device-specific registers; activation of ATM control sockets.
net_bind_service	Allow low port binding. Port < 1024 for TCP/UDP. VCI < 32 for ATM.
net_raw	Allows opening of raw sockets and packet sockets.
netbroadcast	Grant network broadcasting and listening to incoming multicasts.
setfcap	Allow the assignment of file capabilities.
setgid	Allow setgid(2) allow setgroups(2) allow fake gids on credentials passed over a socket.
setpcap	Transfer capability maps from current process to any process.
setuid	Allow all setsuid(2) type calls including fsuid. Allow passing of forged pids on credentials passed over a socket.
sys_admin	Allow the following: configuration of the secure attention key; administration of the random device; examination and configuration of disk quotas; configuring the kernel's syslog; setting the domainname; setting the hostname; calling bdflush(); mount() and umount(), setting up new smb connection; some autofs root ioctls; nfsservctl; VM86_REQUEST_IRQ; to read/write pci config on alpha; irix_pctl on mips (setstacksize); flushing all cache on m68k (sys_cacheflush); removing semaphores; locking/unlocking of shared memory segment; turning swap on/off; forged pids on socket credentials passing; setting readahead and flushing buffers on block devices; setting geometry in floppy driver; turning DMA on/off in xd driver; administration of md devices; tuning the ide driver; access to the nvram device; administration of apm_bios, serial and bttv (TV) device; manufacturer commands in isdn CAPI support driver; reading non-standardized portions of pci configuration space; DDI debug ioctl on sbpcd driver; setting up serial ports; sending raw qic-117 commands; enabling/disabling tagged queuing on SCSI controllers and sending arbitrary SCSI commands; setting encryption key on loopback filesystem; setting zone reclaim policy.
sys_boot	Grant ability to reboot the system.
sys_chroot	Grant use of the chroot(2) call.
sys_module	Allow unrestricted kernel modification including but not limited to loading and removing kernel modules. Allows modification of kernel's bounding capability mask. See sysctl.
sys_nice	Grants privilege to change priority of any process. Grants change of scheduling algorithm used by any process.

sys_pacct	Allow modification of accounting for any process.
sys_ptrace	Allow ptrace of any process.
sys_rawio	Grant permission to use <code>ioperm(2)</code> and <code>iopl(2)</code> as well as the ability to send messages to USB devices via <code>/proc/bus/usb</code> .
sys_resource	Override the following: resource limits; quota limits; reserved space on <code>ext2</code> filesystem; size restrictions on IPC message queues; max number of consoles on console allocation; max number of keymaps. Set resource limits. Modify data journaling mode on <code>ext3</code> filesystem, Allow more than 64hz interrupts from the real-time clock.
sys_time	Grant permission to set system time and to set the real-time lock.
sys_tty_config	Grant permission to configure tty devices.

Class	capability2 – This is currently unused by SELinux
Permissions	Description (2 unique permissions)
mac_admin	Unused
mac_override	Unused

8.11 X Windows Object Classes

These are userspace objects.

Class	x_drawable -
Permissions	Description (19 unique permissions)
add_child	Add new window.
blend	
create	Create a drawable object.
destroy	Destroy a drawable object.
get_property	
getattr	Get attributes from a drawable object.
hide	Hide a drawable object.
list_child	
list_property	List property from a window.
manage	Required to create a device context. (source code)
override	
read	
receive	
remove_child	
send	
set_property	
setattr	
show	
write	

The SELinux Notebook - The Foundations

Class	x_screen -
Permissions	Description (8 unique permissions)
getattr	
hide_cursor	
saver_getattr	
saver_hide	
saver_setattr	
saver_show	
setattr	
show_cursor	

Class	x_gc - The GC object class represents graphics contexts.
Permissions	Description (5 unique permissions)
create	Create Graphic Contexts object.
destroy	Free (dereference) a Graphics Contexts object.
getattr	Get attributes from Graphic Contexts object.
setattr	Set attributes for Graphic Contexts object.
use	

Class	x_font -
Permissions	Description (6 unique permissions)
add_glyph	Create glyph for cursor
create	Load a font.
destroy	Free a font.
getattr	Obtain font names, path, etc.
remove_glyph	Free glyph
use	Use a font for drawing.

Class	x_colormap -
Permissions	Description (10 unique permissions)
add_color	
create	Create a new Colormap.
destroy	Free a Colormap.
getattr	Get the color gamut of a screen.
install	Copy a virtual colormap into the display hardware.
read	Read color cells of colormap.
remove_color	
uninstall	Remove a virtual colormap from the display hardware.
use	
write	Change color cells in colormap.

The SELinux Notebook - The Foundations

Class	x_property -
Permissions	Description (7 unique permissions)
append	Append a property.
create	Create property object.
destroy	Free (dereference) a property object.
getattr	Get attributes of a property.
read	Read a property.
setattr	Set attributes of a property.
write	Write a property.

Class	x_selection -
Permissions	Description (4 unique permissions)
getattr	
read	
setattr	
write	

Class	x_cursor -
Permissions	Description (7 unique permissions)
create	Create an arbitrary cursor object.
destroy	Free (dereference) a cursor object.
getattr	Get attributes of the cursor.
read	
setattr	Set attributes of the cursor.
use	Associate a cursor object with a window.
write	

Class	x_client -
Permissions	Description (4 unique permissions)
destroy	Close down a client.
getattr	Get attributes of X client.
manage	Required to create a device context. (source code)
setattr	Set attributes of X client.

Class	x_device -
Permissions	Description (12 unique permissions)
bell	
force_cursor	
freeze	
getattr	
getfocus	Get window focus.
grab	

The SELinux Notebook - The Foundations

manage	Required to create a device context. (source code)
read	
setattr	
setfocus	Set window focus.
use	
write	

Class	x_server -
Permissions	Description (6 unique permissions)
debug	
getattr	
grab	
manage	Required to create a device context. (source code)
record	
setattr	

Class	x_extension -
Permissions	Description (2 unique permissions)
query	
use	

Class	x_resource -
Permissions	Description (2 unique permissions)
read	
write	

Class	x_event -
Permissions	Description (2 unique permissions)
receive	
send	

Class	x_synthetic_event -
Permissions	Description (2 unique permissions)
receive	
send	

Class	x_application_data -
Permission	Description (3 unique permissions)
copy	
paste	
paste_after_confirm	

8.12 Database Object Classes

These are userspace objects – The PostgreSQL database supports these with their SE-PostgreSQL database extension. The “[Security-Enhanced PostgreSQL Security Wiki](#)” [Ref. 3] explains the objects, their permissions and how they should be used in detail.

Class	db_database
Permission	Description (Inherit 6 common database permissions + 5 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
access	Required to connect to the database – this is the minimum permission required by an PostgreSQL client.
get_param	Deprecated - Allow reference of run-time parameters with the SHOW statement.
install_module	Required to install a dynamic link library.
load_module	Required to load a dynamic link library.
set_param	Deprecated - Set run-time parameters with the SET statement.

Class	db_table
Permission	Description (Inherit 6 common database permissions + 6 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
delete	Required to delete from a table with a DELETE statement, or when removing the table contents with a TRUNCATE statement.
insert	Required to insert into a table with an INSERT statement, or when restoring it with a COPY FROM statement.
lock	Required to get a table lock with a LOCK statement.
select	Required to refer to a table with a SELECT statement or to dump the table contents with a COPY TO statement.
update	Required to update a table with an UPDATE statement.
use	Deprecated - Required on tables and columns when referring to a conditional SQL statement clause (WHERE, JOIN~ON, or HAVING clauses), GROUP BY clause, or ORDER BY clause.

Class	db_procedure
Permission	Description (Inherit 6 common database permissions + 2 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
entrypoint	Required for any functions defined as Trusted Procedures (see [Ref. 3]).
execute	Required for functions executed with SQL queries.

Class	db_column
Permission	Description (Inherit 6 common database permissions + 4 unique)
Inherit Common	create, drop, getattr, relabelfrom, relabelto,

The SELinux Notebook - The Foundations

Database Permissions	setattr
insert	Required to insert a new entry using the INSERT statement.
select	Required to reference columns.
update	Required to update a table with an UPDATE statement.
use	Deprecated - Required on tables and columns when referring to a conditional SQL statement clause (WHERE, JOIN~ON, or HAVING clauses), GROUP BY clause, or ORDER BY clause.

Class	db_tuple
Permission	Description (7 unique)
delete	Required to delete entries with a DELETE or TRUNCATE statement.
insert	Required when inserting a entry with an INSERT statement, or restoring tables with a COPY FROM statement.
relabelfrom	The security context of an entry can be changed with an UPDATE to the security_context column at which time relabelfrom and relabelto permission is evaluated. The client must have relabelfrom permission to the security context before the entry is changed, and relabelto permission to the security context after the entry is changed.
relabelto	
select	Required when: reading entries with a SELECT statement, returning entries that are subjects for updating queries with a RETURNING clause, or dumping tables with a COPY TO statement. Entries that the client does not have select permission on will be filtered from the result set.
update	Required when updating an entry with an UPDATE statement. Entries that the client does not have update permission on will not be updated.
use	Deprecated - Required on tables and columns when referring to a conditional SQL statement clause (WHERE, JOIN~ON, or HAVING clauses), GROUP BY clause, or ORDER BY clause. Entries that the client does not have use permission on will be filtered from the result set and from being subject to update / deletion.

Class	db_blob
Permission	Description (Inherit 6 common database permissions + 4 unique)
Inherit Common Database Permissions	create, drop, setattr, relabelfrom, relabelto, setattr
export	Export a binary large object by calling the lo_export() function.
import	Import a file as a binary large object by calling the lo_import() function.
read	Read a binary large object the loread() function.
write	Write a binary large objecty with the lowrite() function.

8.13 Miscellaneous Object Classes

Class	passwd - This is a userspace object for controlling changes to passwd information.
Permissions	Description (5 unique permissions)
chfn	Change another users finger info.
chsh	Change another users shell.
crontab	crontab another user.
passwd	Change another users passwd.
rootok	pam_rootok check – skip authentication.

Class	nscd - This is a userspace object for the Name Service Cache Daemon.
Permission	Description (10 unique permissions)
admin	Allow the nscd daemon to be shut down.
getgrp	Get group information.
gethost	Get host information.
getpwd	Get password information.
getserv	Get ?? information.
getstat	Get the AVC stats from the nscd daemon.
shmemgrp	Get shmem group file descriptor.
shmemhost	Get shmem host descriptor. ??
shmempwd	
shmemserv	

Class	dbus - This is a userspace object for the D-BUS Messaging service that is required to run X-Windows (and other services).
Permission	Description (2 unique permissions)
acquire_svc	Open a virtual circuit (communications channel).
send_msg	Send a message.

Class	context - This is a userspace object for the translation daemon mcstransd. These permissions are required to allow translation and querying of level and ranges for MCS and MLS systems.
Permission	Description (2 unique permissions)
contains	Calculate a MLS/MCS subset - Required to check what the configuration file contains.
translate	Translate a raw MLS/MCS label - Required to allow a domain to translate contexts.

Class	key – This is a kernel object to manage Keyrings.
Permission	Description (7 unique permissions)
create	Create a keyring.
link	Link a key into the keyring.

The SELinux Notebook - The Foundations

read	Read a keyring.
search	Search a keyring.
setattr	Change permissions on a keyring.
view	View a keyring.
write	Add a key to the keyring.

Class	memprotect – This is a kernel object to protect lower memory blocks.
Permission	Description (1 unique permission)
mmap_zero	Security check on mmap operations to see if the user is attempting to mmap to low area of the address space. The amount of space protected is indicated by a proc tunable (<code>/proc/sys/vm/mmap_min_addr</code>). Setting this value to 0 will disable the checks. The “ SELinux hardening for mmap_min_addr protections ” [Ref. 16] describes additional checks that will be added to the kernel to protect against some kernel exploits (by requiring <code>CAP_SYS_RAWIO</code> (root) and the SELinux <code>memprotect / mmap_zero</code> permission instead of only one or the other).

9. Appendix B – SELinux Commands

This section gives a brief explanation of the SELinux specific commands. Some of these have been used within this Notebook, however the appropriate man pages do give more detail and the SELinux project site has a page that details all the available tools and commands at:

<http://userspace.selinuxproject.org/trac/wiki/SelinuxTools>

Command	Man Page	Purpose
audit2allow	1	Generates policy allow rules from the audit.log file.
audit2why	8	Describes audit.log messages and why access was denied.
avcstat	8	Displays the AVC statistics.
chcat	8	Change or remove a category from a file or user.
chcon	1	Changes the security context of a file.
checkmodule	8	Compiles base and loadable modules from source.
checkpolicy	8	Compiles a monolithic policy from source.
fixfiles	8	Update / correct the security context of for filesystems that use extended attributes.
genhomedircon	8	Generates file configuration entries for users home directories. This command has also been built into <code>semanage</code> , therefore when using the policy store / loadable modules this does not need to be used.
getenforce	1	Shows the current enforcement state.
getsebool	8	Shows the state of the booleans.
load_policy	8	Loads a new policy into the kernel. Not required when using <code>semanage / semodule</code> commands.
matchpathcon	8	Show a files path and security context.
newrole	1	Allows users to change roles - runs a new shell with the new security context.
restorecon	8	Sets the security context on one or more files.
run_init	8	Runs an <code>init</code> script under the correct context.
runcon	1	Runs a command with the specified context.
selinuxenabled	1	Shows whether SELinux is enabled or not.
semanage	8	Used to configure various areas of a policy within a policy store.
semodule	8	Used to manage the installation, upgrading etc. of policy modules.
semodule_expand	8	Manually expand a base policy package into a kernel binary policy file.
semodule_link	8	Manually link a set of module packages.
semodule_package	8	Create a module package with various configuration files (file context etc.)
sestatus	8	Show the current status of SELinux and the loaded policy.
setenforce	1	Sets / unsets enforcement mode.
setfiles	8	Initialise the extended attributes of filesystems.
setsebool	8	Sets the state of a boolean to on or off persistently across reboots or for this session only.
selinux-relabel	-	This is a script that is called using <code>'service selinux-relabel cmd'</code> , where <code>cmd</code> is <code>help, status, relabel, switch <policy_name> or cancel.</code>

10. Appendix C – API Summary for libselinux

These functions have been taken from the following header files delivered in the “libselinux-devel-2.0.78-1.fc10.i386” rpm, and sorted in alphabetical order:

```
/usr/include/selinux/avc.h
/usr/include/selinux/context.h
/usr/include/selinux/get_context_list.h
/usr/include/selinux/get_default_type.h
/usr/include/selinux/label.h
/usr/include/selinux/selinux.h
```

The appropriate man (3) pages should be consulted for detailed usage, also the libselinux source code (see the “libselinux-2.0.78-1.fc10.src” rpm) has a number of sample applications showing the API usage, and the “SELinux Support for Userspace Object Managers” [Ref. 17] illustrates the use of a userspace AVC.

Num.	Function Name	Description	Header File
1.	avc_add_callback	Register a callback for security events. Register a callback function for events in the set @events related to the SID pair (@ssid, @tsid) and the permissions @perms, interpreting @perms based on @tclass. Returns %0 on success or -%1 if insufficient memory exists to add the callback.	avc.h
2.	avc_audit	Audit the granting or denial of permissions in accordance with the policy. This function is typically called by avc_has_perm() after a permission check, but can also be called directly by callers who use avc_has_perm_noaudit() in order to separate the permission check from the auditing. For example, this separation is useful when the permission check must be performed under a lock, to allow the lock to be released before calling the auditing code.	avc.h
3.	avc_av_stats	Log AV table statistics. Log a message with information about the size and distribution of the access vector table. The audit callback is used to print the message.	avc.h

The SELinux Notebook - The Foundations

Num.	Function Name	Description	Header File
4.	<code>avc_cache_stats</code>	Get cache access statistics. Fill the supplied structure with information about AVC activity since the last call to <code>avc_init()</code> or <code>avc_reset()</code> . See the structure definition for details.	<code>avc.h</code>
5.	<code>avc_cleanup</code>	Remove unused SIDs and AVC entries. Search the SID table for SID structures with zero reference counts, and remove them along with all AVC entries that reference them. This can be used to return memory to the system.	<code>avc.h</code>
6.	<code>avc_compute_create</code>	Compute SID for labeling a new object. Call the security server to obtain a context for labeling a new object. Look up the context in the SID table, making a new entry if not found. Increment the reference counter for the SID. Store a pointer to the SID structure into the memory referenced by <code>@newsid</code> , returning <code>%0</code> on success or <code>-%1</code> on error with <code>@errno</code> set.	<code>avc.h</code>
7.	<code>avc_compute_member</code>	Compute SID for polyinstantiation. Call the security server to obtain a context for labeling an object instance. Look up the context in the SID table, making a new entry if not found. Increment the reference counter for the SID. Store a pointer to the SID structure into the memory referenced by <code>@newsid</code> , returning <code>%0</code> on success or <code>-%1</code> on error with <code>@errno</code> set.	<code>avc.h</code>
8.	<code>avc_context_to_sid</code> <code>avc_context_to_sid_raw</code>	Get SID for context. Look up security context <code>@ctx</code> in SID table, making a new entry if <code>@ctx</code> is not found. Increment the reference counter for the SID. Store a pointer to the SID structure into the memory referenced by <code>@sid</code> , returning <code>%0</code> on success or <code>-%1</code> on error with <code>@errno</code> set.	<code>avc.h</code>
9.	<code>avc_destroy</code>	Free all AVC structures. Destroy all AVC structures and free all allocated memory. User-supplied locking, memory, and audit callbacks will be retained, but security-event callbacks will not. All SID's will be invalidated. User must call <code>avc_init()</code> if further use of AVC is desired.	<code>avc.h</code>
10.	<code>avc_entry_ref_init</code>	Initialize an AVC entry reference. Use this macro to initialize an <code>avc</code> entry reference structure before first use. These structures are passed to <code>avc_has_perm()</code> , which stores cache entry references in them. They can increase performance on repeated queries.	<code>avc.h</code>

The SELinux Notebook - The Foundations

Num.	Function Name	Description	Header File
11.	<code>avc_get_initial_sid</code>	Get SID for an initial kernel security identifier. Get the context for an initial kernel security identifier specified by @name using <code>security_get_initial_context()</code> and then call <code>avc_context_to_sid()</code> to get the corresponding SID.	<code>avc.h</code>
12.	<code>avc_has_perm</code>	Check permissions and perform any appropriate auditing. Check the AVC to determine whether the @requested permissions are granted for the SID pair (@ssid, @tsid), interpreting the permissions based on @tclass, and call the security server on a cache miss to obtain a new decision and add it to the cache. Update @aeref to refer to an AVC entry with the resulting decisions. Audit the granting or denial of permissions in accordance with the policy. Return %0 if all @requested permissions are granted, -%1 with @errno set to %EACCES if any permissions are denied or to another value upon other errors.	<code>avc.h</code>
13.	<code>avc_has_perm_noaudit</code>	Check permissions but perform no auditing. Check the AVC to determine whether the @requested permissions are granted for the SID pair (@ssid, @tsid), interpreting the permissions based on @tclass, and call the security server on a cache miss to obtain a new decision and add it to the cache. Update @aeref to refer to an AVC entry with the resulting decisions, and return a copy of the decisions in @avd. Return %0 if all @requested permissions are granted, -%1 with @errno set to %EACCES if any permissions are denied, or to another value upon other errors. This function is typically called by <code>avc_has_perm()</code> , but may also be called directly to separate permission checking from auditing, e.g. in cases where a lock must be held for the check but should be released for the auditing.	<code>avc.h</code>
14.	<code>avc_init</code>	Initialize the AVC. Initialize the access vector cache. Return %0 on success or -%1 with @errno set on failure. If @msgprefix is NULL, use "uavc". If any callback structure references are NULL, use default methods for those callbacks (see the definition of the callback structures).	<code>avc.h</code>

The SELinux Notebook - The Foundations

Num.	Function Name	Description	Header File
15.	avc_open	Initialize the AVC. This function is identical to <code>avc_init()</code> except the message prefix is set to "avc" and any callbacks desired should be specified via <code>selinux_set_callback()</code> .	avc.h
16.	avc_reset	Flush the cache and reset statistics. Remove all entries from the cache and reset all access statistics (as returned by <code>avc_cache_stats()</code>) to zero. The SID mapping is not affected. Return %0 on success, -%1 with <code>@errno</code> set on error.	avc.h
17.	avc_sid_stats	Log SID table statistics. Log a message with information about the size and distribution of the SID table. The audit callback is used to print the message.	avc.h
18.	avc_sid_to_context avc_sid_to_context_raw	Get copy of context corresponding to SID. Return a copy of the security context corresponding to the input <code>@sid</code> in the memory referenced by <code>@ctx</code> . The caller is expected to free the context with <code>freecon()</code> . Return %0 on success, -%1 on failure, with <code>@errno</code> set to <code>%ENOMEM</code> if insufficient memory was available to make the copy, or <code>%EINVAL</code> if the input SID is invalid.	avc.h
19.	checkPasswdAccess	Check a permission in the <code>passwd</code> class. Return 0 if granted or -1 otherwise.	selinux.h
20.	context_free	Free the storage used by a context.	context.h
21.	context_new	Return a new context initialized to a context string.	context.h
22.	context_range_get	Get a pointer to the <code>range</code> .	context.h
23.	context_range_set	Set the <code>range</code> component. Returns nonzero if unsuccessful.	context.h
24.	context_role_get	Get a pointer to the <code>role</code> .	context.h
25.	context_role_set	Set the <code>role</code> component. Returns nonzero if unsuccessful.	context.h
26.	context_str	Return a pointer to the string value of <code>context_t</code> . Valid until the next call to <code>context_str</code> or <code>context_free</code> for the same <code>context_t*</code> .	context.h
27.	context_type_get	Get a pointer to the <code>type</code> .	context.h
28.	context_type_set	Set the <code>type</code> component. Returns nonzero if unsuccessful.	context.h
29.	context_user_get	Get a pointer to the <code>user</code> .	context.h

The SELinux Notebook - The Foundations

Num.	Function Name	Description	Header File
30.	<code>context_user_set</code>	Set the <code>user</code> component. Returns nonzero if unsuccessful.	<code>context.h</code>
31.	<code>fgetfilecon</code> <code>fgetfilecon_raw</code>	Wrapper for the <code>xattr</code> API - Get file context, and set <code>*con</code> to refer to it. Caller must free via <code>freecon</code> .	<code>selinux.h</code>
32.	<code>freecon</code>	Free the memory allocated for a context by any of the <code>get*</code> calls.	<code>selinux.h</code>
33.	<code>freeconary</code>	Free the memory allocated for a context array by <code>security_compute_user</code> .	<code>selinux.h</code>
34.	<code>fsetfilecon</code> <code>fsetfilecon_raw</code>	Wrapper for the <code>xattr</code> API- Set file context.	<code>selinux.h</code>
35.	<code>get_default_context</code>	Get the default security context for a user session for 'user' spawned by 'fromcon' and set <code>*newcon</code> to refer to it. The context will be one of those authorized by the policy, but the selection of a default is subject to user customizable preferences. If 'fromcon' is NULL, defaults to current context. Returns 0 on success or -1 otherwise. Caller must free via <code>freecon</code> .	<code>get_context_list.h</code>
36.	<code>get_default_context_with_level</code>	Same as <code>get_default_context</code> , but use the provided MLS level rather than the default level for the user.	<code>get_context_list.h</code>
37.	<code>get_default_context_with_role</code>	Same as <code>get_default_context</code> , but only return a context that has the specified role.	<code>get_context_list.h</code>
38.	<code>get_default_context_with_rolelevel</code>	Same as <code>get_default_context</code> , but only return a context that has the specified role and level.	<code>get_context_list.h</code>
39.	<code>get_default_type</code>	Get the default type (domain) for 'role' and set 'type' to refer to it. Caller must free via <code>free()</code> . Return 0 on success or -1 otherwise.	<code>get_default_type.h</code>
40.	<code>get_ordered_context_list</code>	Get an ordered list of authorized security contexts for a user session for 'user' spawned by 'fromcon' and set <code>*conary</code> to refer to the NULL-terminated array of contexts. Every entry in the list will be authorized by the policy, but the ordering is subject to user customizable preferences. Returns number of entries in <code>*conary</code> . If 'fromcon' is NULL, defaults to current context. Caller must free via <code>freeconary</code> .	<code>get_context_list.h</code>
41.	<code>get_ordered_context_list_with_level</code>	Same as <code>get_ordered_context_list</code> , but use the provided MLS level rather than the default level for the user.	<code>get_context_list.h</code>

The SELinux Notebook - The Foundations

Num.	Function Name	Description	Header File
42.	getcon getcon_raw	Get current context, and set *con to refer to it. Caller must free via freecon.	selinux.h
43.	getexeccon getexeccon_raw	Get exec context, and set *con to refer to it. Sets *con to NULL if no exec context has been set, i.e. using default. If non-NULL, caller must free via freecon.	selinux.h
44.	getfilecon getfilecon_raw	Wrapper for the xattr API - Get file context, and set *con to refer to it. Caller must free via freecon.	selinux.h
45.	getfscreatecon getfscreatecon_raw	Get fscreate context, and set *con to refer to it. Sets *con to NULL if no fs create context has been set, i.e. using default. If non-NULL, caller must free via freecon.	selinux.h
46.	getkeycreatecon getkeycreatecon_raw	Get keycreate context, and set *con to refer to it. Sets *con to NULL if no key create context has been set, i.e. using default. If non-NULL, caller must free via freecon.	selinux.h
47.	getpeercon getpeercon_raw	Wrapper for the socket API - Get context of peer socket, and set *con to refer to it. Caller must free via freecon.	selinux.h
48.	getpidcon getpidcon_raw	Get context of process identified by pid, and set *con to refer to it. Caller must free via freecon.	selinux.h
49.	getprevcon getprevcon_raw	Get previous context (prior to last exec), and set *con to refer to it. Caller must free via freecon.	selinux.h
50.	getseuser	Get the SELinux username and level to use for a given Linux username and service. These values may then be passed into the get_ordered_context_list* and get_default_context* functions to obtain a context for the user. Returns 0 on success or -1 otherwise. Caller must free the returned strings via free().	selinux.h
51.	getseuserbyname	Get the SELinux username and level to use for a given Linux username. These values may then be passed into the get_ordered_context_list* and get_default_context* functions to obtain a context for the user. Returns 0 on success or -1 otherwise. Caller must free the returned strings via free().	selinux.h

The SELinux Notebook - The Foundations

Num.	Function Name	Description	Header File
52.	getsockcreatecon getsockcreatecon_raw	Get sockcreate context, and set *con to refer to it. Sets *con to NULL if no socket create context has been set, i.e. using default. If non-NULL, caller must free via freecon.	selinux.h
53.	is_context_customizable	Returns whether a file context is customizable, and should not be relabeled.	selinux.h
54.	is_selinux_enabled	Return 1 if running on a SELinux kernel, or 0 if not or -1 for error.	selinux.h
55.	is_selinux_mls_enabled	Return 1 if we are running on a SELinux MLS kernel, or 0 otherwise.	selinux.h
56.	lgetfilecon lgetfilecon_raw	Wrapper for the xattr API - Get file context, and set *con to refer to it. Caller must free via freecon.	selinux.h
57.	lsetfilecon lsetfilecon_raw	Wrapper for the xattr API- Set file context for symbolic link.	selinux.h
58.	manual_user_enter_context	Allow the user to manually enter a context as a fallback if a list of authorized contexts could not be obtained. Caller must free via freecon. Returns 0 on success or -1 otherwise.	get_context_list.h
59.	matchmediacon	Match the specified media and against the media contexts configuration and set *con to refer to the resulting context. Caller must free con via freecon.	selinux.h
60.	matchpathcon	Match the specified pathname and mode against the file context sconfiguration and set *con to refer to the resulting context. 'mode' can be 0 to disable mode matching. Caller must free via freecon. If matchpathcon_init has not already been called, then this function will call it upon its first invocation with a NULL path.	selinux.h
61.	matchpathcon_checkmatches	Check to see whether any specifications had no matches and report them. The 'str' is used as a prefix for any warning messages.	selinux.h
62.	matchpathcon_filespec_add	Maintain an association between an inode and a specification index, and check whether a conflicting specification is already associated with the same inode (e.g. due to multiple hard links). If so, then use the latter of the two specifications based on their order in the file contexts configuration. Return the used specification index.	selinux.h
63.	matchpathcon_filespec_destroy	Destroy any inode associations that have been added, e.g. to restart for a new filesystem.	selinux.h
64.	matchpathcon_filespec_eval	Display statistics on the hash table usage for the associations.	selinux.h
65.	matchpathcon_fini	Free the memory allocated by matchpathcon_init.	selinux.h

The SELinux Notebook - The Foundations

Num.	Function Name	Description	Header File
66.	matchpathcon_index	Same as 'matchpathcon', but return a specification index for later use in a matchpathcon_filespec_add() call.	selinux.h
67.	matchpathcon_init	Load the file contexts configuration specified by 'path' into memory for use by subsequent matchpathcon calls. If 'path' is NULL, then load the active file contexts configuration, i.e. the path returned by selinux_file_context_path(). Unless the MATCHPATHCON_BASEONLY flag has been set, this function also checks for a 'path'.homedirs file and a 'path'.local file and loads additional specifications from them if present.	selinux.h
68.	matchpathcon_init_prefix	Same as matchpathcon_init, but only load entries with regexes that have stems that are prefixes of 'prefix'.	selinux.h
69.	print_access_vector	Display an access vector in a string representation.	selinux.h
70.	query_user_context	Given a list of authorized security contexts for the user, query the user to select one and set *newcon to refer to it. Caller must free via freecon. Returns 0 on success or -1 otherwise.	get_context_list.h
71.	rpm_execon	Execute a helper for rpm in an appropriate security context.	selinux.h
72.	security_av_perm_to_string	Convert access vector permissions to string names.	selinux.h
73.	security_av_string	Returns an access vector in a string representation. User must free the returned string via free().	selinux.h
74.	security_canonicalize_context security_canonicalize_context_raw	Canonicalize a security context.	selinux.h
75.	security_check_context security_check_context_raw	Check the validity of a security context.	selinux.h
76.	security_class_to_string	Convert security class values to string names.	selinux.h
77.	security_commit_booleans	Commit the pending values for the booleans.	selinux.h
78.	security_compute_av security_compute_av_raw	Compute an access decision.	selinux.h
79.	security_compute_create security_compute_create_raw	Compute a labeling decision and set *newcon to refer to it. Caller must free via freecon.	selinux.h

The SELinux Notebook - The Foundations

Num.	Function Name	Description	Header File
80.	security_compute_member security_compute_member_raw	Compute a polyinstantiation member decision and set *newcon to refer to it. Caller must free via freecon.	selinux.h
81.	security_compute_relabel security_compute_relabel_raw	Compute a relabeling decision and set *newcon to refer to it. Caller must free via freecon.	selinux.h
82.	security_compute_user security_compute_user_raw	Compute the set of reachable user contexts and set *con to refer to the NULL-terminated array of contexts. Caller must free via freeconary.	selinux.h
83.	security_disable	Disable SELinux at runtime (must be done prior to initial policy load).	selinux.h
84.	security_get_boolean_active	Get the active value for the boolean.	selinux.h
85.	security_get_boolean_names	Get the boolean names	selinux.h
86.	security_get_boolean_pending	Get the pending value for the boolean.	selinux.h
87.	security_get_initial_context security_get_initial_context_raw	Get the context of an initial kernel security identifier by name. Caller must free via freecon.	selinux.h
88.	security_getenforce	Get the enforce flag value.	selinux.h
89.	security_load_booleans	Load policy boolean settings. Path may be NULL, in which case the booleans are loaded from the active policy boolean configuration file.	selinux.h
90.	security_load_policy	Load a policy configuration.	selinux.h
91.	security_policyvers	Get the policy version number.	selinux.h
92.	security_set_boolean	Set the pending value for the boolean.	selinux.h
93.	security_set_boolean_list	Save a list of booleans in a single transaction.	selinux.h
94.	security_setenforce	Set the enforce flag value.	selinux.h
95.	selabel_close	Destroy the specified handle, closing files, freeing allocated memory, etc. The handle may not be further used after it has been closed.	label.h
96.	selabel_lookup selabel_lookup_raw	Perform a labeling lookup operation. Return %0 on success, -%1 with @errno set on failure. The key and type arguments are the inputs to the lookup operation; appropriate values are dictated by the backend in use. The result is returned in the memory pointed to by @con and must be freed by the user with freecon().	label.h

The SELinux Notebook - The Foundations

Num.	Function Name	Description	Header File
97.	<code>selabel_open</code>	<p>Create a labeling handle.</p> <p>Open a labeling backend for use. The available backend identifiers are:</p> <ul style="list-style-type: none"> <code>SELABEL_CTX_FILE</code> - file contexts. <code>SELABEL_CTX_MEDIA</code> - media contexts. <code>SELABEL_CTX_X</code> - x contexts. <p>Options may be provided via the <code>opts</code> parameter; available options are:</p> <ul style="list-style-type: none"> <code>SELABEL_OPT_UNUSED</code> - no-op option, useful for unused slots in an array of options. <code>SELABEL_OPT_VALIDATE</code> - validate contexts before returning them (boolean value). <code>SELABEL_OPT_BASEONLY</code> - don't use local customizations to backend data (boolean value). <code>SELABEL_OPT_PATH</code> - specify an alternate path to use when loading backend data. <code>SELABEL_OPT_SUBSET</code> - select a subset of the search space as an optimization (file backend). <p>Not all options may be supported by every backend. Return value is the created handle on success or NULL with <code>@errno</code> set on failure.</p>	<code>label.h</code>
98.	<code>selabel_stats</code>	Log a message with information about the number of queries performed, number of unused matching entries, or other operational statistics. Message is backend-specific, some backends may not output a message.	<code>label.h</code>
99.	<code>selinux_binary_policy_path</code>	Return path to the binary <code>policy</code> file under the policy root directory.	<code>selinux.h</code>
100.	<code>selinux_booleans_path</code>	Return path to the <code>booleans</code> file under the policy root directory.	<code>selinux.h</code>
101.	<code>selinux_check_passwd_access</code>	Check a permission in the <code>passwd</code> class. Return 0 if granted or -1 otherwise.	<code>selinux.h</code>
102.	<code>selinux_check_securetty_context</code>	Check if the <code>tty_context</code> is defined as a <code>securetty</code> . Return 0 if secure, < 0 otherwise.	<code>selinux.h</code>
103.	<code>selinux_colors_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
104.	<code>selinux_contexts_path</code>	Return path to <code>contexts</code> directory under the policy root directory.	<code>selinux.h</code>
105.	<code>selinux_customizable_types_path</code>	Return path to <code>customizable_types</code> file under the policy root directory.	<code>selinux.h</code>
106.	<code>selinux_default_context_path</code>	Return path to <code>default_context</code> file under the policy root directory.	<code>selinux.h</code>

The SELinux Notebook - The Foundations

Num.	Function Name	Description	Header File
107.	<code>selinux_failsafe_context_path</code>	Return path to <code>failsafe_context</code> file under the policy root directory.	<code>selinux.h</code>
108.	<code>selinux_file_context_cmp</code>	Compare two file contexts, return 0 if equivalent.	<code>selinux.h</code>
109.	<code>selinux_file_context_homedir_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
110.	<code>selinux_file_context_local_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
111.	<code>selinux_file_context_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
112.	<code>selinux_file_context_verify</code>	Verify the context of the file 'path' against policy. Return 0 if correct.	<code>selinux.h</code>
113.	<code>selinux_getenforcemode</code>	Reads the <code>/etc/selinux/config</code> file and determines whether the machine should be started in enforcing (1), permissive (0) or disabled (-1) mode.	<code>selinux.h</code>
114.	<code>selinux_getpolicytype</code>	Reads the <code>/etc/selinux/config</code> file and determines what the default policy for the machine is. Calling application must free <code>policytype</code> .	<code>selinux.h</code>
115.	<code>selinux_homedir_context_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
116.	<code>selinux_init_load_policy</code>	<p>Perform the initial policy load.</p> <p>This function determines the desired enforcing mode, sets the the <code>*enforce</code> argument accordingly for the caller to use, sets the SELinux kernel enforcing status to match it, and loads the policy. It also internally handles the initial <code>selinuxfs</code> mount required to perform these actions.</p> <p>The function returns 0 if everything including the policy load succeeds. In this case, <code>init</code> is expected to re-exec itself in order to transition to the proper security context. Otherwise, the function returns -1, and <code>init</code> must check <code>*enforce</code> to determine how to proceed. If enforcing (<code>*enforce > 0</code>), then <code>init</code> should halt the system. Otherwise, <code>init</code> may proceed normally without a re-exec.</p>	<code>selinux.h</code>
117.	<code>selinux_lsetfilecon_default</code>	This function sets the file context on to the system defaults returns 0 on success.	<code>selinux.h</code>
118.	<code>selinux_media_context_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>

The SELinux Notebook - The Foundations

Num.	Function Name	Description	Header File
119.	<code>selinux_mkload_policy</code>	<p>Make a policy image and load it.</p> <p>This function provides a higher level interface for loading policy than <code>security_load_policy</code>, internally determining the right policy version, locating and opening the policy file, mapping it into memory, manipulating it as needed for current boolean settings and/or local definitions, and then calling <code>security_load_policy</code> to load it.</p> <p>'preservebooleans' is a boolean flag indicating whether current policy boolean values should be preserved into the new policy (if 1) or reset to the saved policy settings (if 0). The former case is the default for policy reloads, while the latter case is an option for policy reloads but is primarily for the initial policy load.</p>	<code>selinux.h</code>
120.	<code>selinux_netfilter_context_path</code>	Returns path to the <code>netfilter_context</code> file under the policy root directory.	<code>selinux.h</code>
121.	<code>selinux_path</code>	Returns path to the policy root directory.	<code>selinux.h</code>
122.	<code>selinux_policy_root</code>	Reads the <code>/etc/selinux/config</code> file and returns the top level directory.	<code>selinux.h</code>
123.	<code>selinux_raw_context_to_color</code>	<p>Perform context translation between security contexts and display colors. Returns a space-separated list of ten ten hex RGB triples prefixed by hash marks, e.g. "#ff0000". Caller must free the resulting string via <code>free()</code>. Returns -1 upon an error or 0 otherwise.</p>	<code>selinux.h</code>
124.	<code>selinux_raw_to_trans_context</code>	<p>Perform context translation between the human-readable format ("translated") and the internal system format ("raw"). Caller must free the resulting context via <code>freecon</code>. Returns -1 upon an error or 0 otherwise. If passed NULL, sets the returned context to NULL and returns 0.</p>	<code>selinux.h</code>
125.	<code>selinux_removable_context_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
126.	<code>selinux_securetty_types_path</code>	Return path to the <code>securetty_types</code> file under the policy root directory.	<code>selinux.h</code>
127.	<code>selinux_set_mapping</code>	Userspace class mapping support.	<code>selinux.h</code>
128.	<code>selinux_trans_to_raw_context</code>	<p>Perform context translation between the human-readable format ("translated") and the internal system format ("raw"). Caller must free the resulting context via <code>freecon</code>. Returns -1 upon an error or 0 otherwise. If passed NULL, sets the returned context to NULL and returns 0.</p>	<code>selinux.h</code>
129.	<code>selinux_translations_path</code>	Return path to <code>setrans.conf</code> file under the policy root directory.	<code>selinux.h</code>

The SELinux Notebook - The Foundations

Num.	Function Name	Description	Header File
130.	<code>selinux_user_contexts_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
131.	<code>selinux_users_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
132.	<code>selinux_usersconf_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
133.	<code>selinux_virtual_domain_context_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
134.	<code>selinux_virtual_image_context_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
135.	<code>selinux_x_context_path</code>	Return path to <code>x_context</code> file under the policy root directory.	<code>selinux.h</code>
136.	<code>set_matchpathcon_canoncon</code>	Same as <code>'set_matchpathcon_invalidcon'</code> , but also allows canonicalization of the context, by changing <code>*context</code> to refer to the canonical form. If not set, and <code>invalidcon</code> is also not set, then this defaults to calling <code>security_canonicalize_context()</code> .	<code>selinux.h</code>
137.	<code>set_matchpathcon_flags</code>	Set flags controlling operation of <code>matchpathcon_init</code> or <code>matchpathcon</code> : <code>MATCHPATHCON_BASEONLY</code> - Only process the base file_contexts file. <code>MATCHPATHCON_NOTRANS</code> - Do not perform any context translation. <code>MATCHPATHCON_VALIDATE</code> - Validate/canonicalize contexts at init time.	<code>selinux.h</code>
138.	<code>set_matchpathcon_invalidcon</code>	Set the function used by <code>matchpathcon_init</code> when checking the validity of a context in the <code>file_contexts</code> configuration. If not set, then this defaults to a test based on <code>security_check_context()</code> . The function is also responsible for reporting any such error, and may include the 'path' and 'lineno' in such error messages.	<code>selinux.h</code>
139.	<code>set_matchpathcon_printf</code>	Set the function used by <code>matchpathcon_init</code> when displaying errors about the <code>file_contexts</code> configuration. If not set, then this defaults to <code>fprintf(stderr, fmt, ...)</code> .	<code>selinux.h</code>
140.	<code>set_selinuxmnt</code>	Set the path to the <code>selinuxfs</code> mount point explicitly. Normally, this is determined automatically during <code>libselinux</code> initialization, but this is not always possible, e.g. for <code>/sbin/init</code> which performs the initial mount of <code>selinuxfs</code> .	<code>selinux.h</code>

The SELinux Notebook - The Foundations

Num.	Function Name	Description	Header File
141.	setcon setcon_raw	Set the current security context to <code>con</code> . Note that use of this function requires that the entire application be trusted to maintain any desired separation between the old and new security contexts, unlike <code>exec</code> -based transitions performed via <code>setexeccon</code> . When possible, decompose your application and use <code>setexeccon()</code> + <code>execve()</code> instead. Note that the application may lose access to its open descriptors as a result of a <code>setcon()</code> unless policy allows it to use descriptors opened by the old context.	selinux.h
142.	setexeccon setexeccon_raw	Set <code>exec</code> security context for the next <code>execve</code> . Call with <code>NULL</code> if you want to reset to the default.	selinux.h
143.	setfilecon setfilecon_raw	Wrapper for the <code>xattr</code> API - Set file context.	selinux.h
144.	setfscreatecon setfscreatecon_raw	Set the <code>fscreate</code> security context for subsequent file creations. Call with <code>NULL</code> if you want to reset to the default.	selinux.h
145.	setkeycreatecon setkeycreatecon_raw	Set the <code>keycreate</code> security context for subsequent key creations. Call with <code>NULL</code> if you want to reset to the default.	selinux.h
146.	setsockcreatecon setsockcreatecon_raw	Set the <code>sockcreate</code> security context for subsequent socket creations. Call with <code>NULL</code> if you want to reset to the default.	selinux.h
147.	sidget	Increment SID reference counter. Increment the reference counter for <code>@sid</code> , indicating that <code>@sid</code> is in use by an (additional) object. Return the new reference count, or zero if <code>@sid</code> is invalid (has zero reference count). Note that <code>avc_context_to_sid()</code> also increments reference counts.	avc.h
148.	sidput	Decrement SID reference counter. Decrement the reference counter for <code>@sid</code> , indicating that a reference to <code>@sid</code> is no longer in use. Return the new reference count. When the reference count reaches zero, the SID is invalid, and <code>avc_context_to_sid()</code> must be called to obtain a new SID for the security context.	avc.h
149.	string_to_av_perm	Convert string names to access vector permissions.	selinux.h
150.	string_to_security_class	Convert string names to security class values.	selinux.h

11. Appendix D - Reference Policy Macros

This section contains the macros that are used by the Reference Policy within modules and then expanded into policy language statements. The macros in the first table are described in the [Reference Policy Macros](#) section with the remainder listed for reference. These macros are located in the source tree at `./policy/policy/support` and are coded in the following files:

```
loadable_module.spt
misc_macros.spt
mls_mcs_macros.spt
file_patterns.spt
ipc_patterns.spt
misc_patterns.spt
obj_perm_sets.spt
```

Macro Name	Function	Macro file name
policy_module	For adding the module statement and mandatory require block entries.	loadable_module.spt
gen_require	For use in interfaces to optionally insert a require block	
template	Genrate template interface block	
interface	Generate the access interface block	
optional_policy	Optional policy handling	
gen_tunable	Tunable declaration	
tunable_policy	Tunable policy handling	
gen_user	Generate an SELinux user	
gen_context	Genrate a security context	
gen_bool	Generate a boolean	
gen_cats	Declares categories c0 to c (N-1)	mls_mcs_macros.spt
gen_sens	Declares sensitivites s0 to s (N-1) with dominance in increasing numeric order with s0 lowest, s (N-1) highest.	
gen_levels	Generate levels from s0 to (N-1) with categories c0 to (M-1)	
mls_systemlow	Basic level names for system low and high	
mls_systemhigh		
mcs_systemlow		
mcs_systemhigh		
mcs_allcats	Allocates all categories	

file_patterns.spt

Sets up 'allow \$1 \$2/\$3 : dir { <perms> }' statements for directory patterns (dir)		
Parameters: 1) domain type 2) container (directory) type 3) directory type		
getattr_dirs_pattern	setattr_dirs_pattern	search_dirs_pattern
list_dirs_pattern	add_entry_dirs_pattern	del_entry_dirs_pattern
rw_dirs_pattern	create_dirs_pattern	delete_dirs_pattern
rename_dirs_pattern	manage_dirs_pattern	relabelfrom_dirs_pattern
relabelto_dirs_pattern	relabel_dirs_pattern	

Sets up 'allow \$1 \$2 \$3 : dir file { <perms> }' statements for regular file patterns (file)		
Parameters: 1) domain type 2) container (directory) type 3) file type		
getattr_files_pattern	setattr_files_pattern	read_files_pattern
mmap_files_pattern	exec_files_pattern	append_files_pattern
write_files_pattern	rw_files_pattern	create_files_pattern
delete_files_pattern	rename_files_pattern	manage_files_pattern
relabelfrom_files_pattern	relabelto_files_pattern	relabel_files_pattern

Sets up 'allow \$1 \$2 \$3 : dir file { <perms> }' statements for symbolic link patterns (lnk_file).		
Parameters: 1) domain type 2) container (directory) type 3) file type		
getattr_lnk_files_pattern	setattr_lnk_files_pattern	read_lnk_files_pattern
append_lnk_files_pattern	write_lnk_files_pattern	rw_lnk_files_pattern
create_lnk_files_pattern	delete_lnk_files_pattern	rename_lnk_files_pattern
manage_lnk_files_pattern	relabelfrom_lnk_files_pattern	relabelto_lnk_files_pattern
relabel_lnk_files_pattern		

Sets up 'allow \$1 \$2 \$3 : dir file { <perms> }' statements for (Un)named Pipes / FIFO patterns (fifo_file).	
Parameters: 1) domain type 2) container (directory) type 3) file type	
getattr_fifo_files_pattern	setattr_fifo_files_pattern
read_fifo_files_pattern	append_fifo_files_pattern
write_fifo_files_pattern	rw_fifo_files_pattern
create_fifo_files_pattern	delete_fifo_files_pattern
rename_fifo_files_pattern	manage_fifo_files_pattern
relabelfrom_fifo_files_pattern	relabelto_fifo_files_pattern
relabel_fifo_files_pattern	

Sets up 'allow \$1 \$2 \$3 : dir sock_file { <perms> }' statements for (Un)named sockets patterns (sock_file).	
Parameters: 1) domain type 2) container (directory) type 3) file type	
getattr_sock_files_pattern	setattr_sock_files_pattern
read_sock_files_pattern	write_sock_files_pattern
rw_sock_files_pattern	create_sock_files_pattern
delete_sock_files_pattern	rename_sock_files_pattern
manage_sock_files_pattern	relabelfrom_sock_files_pattern
relabelto_sock_files_pattern	relabel_sock_files_pattern

The SELinux Notebook - The Foundations

Sets up 'allow \$1 \$2 \$3 : dir blk_file { <perms> }' statements for block device node patterns (blk_file).	
Parameters: 1) domain type 2) container (directory) type 3) file type.	
getattr_blk_files_pattern	setattr_blk_files_pattern
read_blk_files_pattern	append_blk_files_pattern
write_blk_files_pattern	rw_blk_files_pattern
create_blk_files_pattern	delete_blk_files_pattern
rename_blk_files_pattern	manage_blk_files_pattern
relabelfrom_blk_files_pattern	relabelto_blk_files_pattern
relabel_blk_files_pattern	

Sets up 'allow \$1 \$2 \$3 : dir chr_file { <perms> }' statements for Character device node patterns (chr_file).	
Parameters: 1) domain type 2) container (directory) type 3) file type	
getattr_chr_files_pattern	setattr_chr_files_pattern
read_chr_files_pattern	append_chr_files_pattern
write_chr_files_pattern	rw_chr_files_pattern
create_chr_files_pattern	delete_chr_files_pattern
rename_chr_files_pattern	manage_chr_files_pattern
relabelfrom_chr_files_pattern	relabelto_chr_files_pattern
relabel_chr_files_pattern	

Sets up statements for file type_transition patterns:		
allow \$1 \$2 : dir { <perms> };		
type_transition \$1 \$2 : \$3 \$4;		
filetrans_add_pattern	filetrans_pattern	admin_pattern

ipc_patterns.spt

unix domain socket patterns \$1, \$2, \$3 and \$4	
stream_connect_pattern	dgram_send_pattern

misc_patterns.spt

Domain transition patterns		
domain_transition_pattern	spec_domtrans_pattern	domain_auto_transition_pattern
domtrans_pattern	ps_process_pattern	

obj_perm_sets.spt

Support macros for sets of object classes and permissions		
dir_file_class_set	file_class_set	notdevfile_class_set
devfile_class_set	socket_class_set	dgram_socket_class_set
stream_socket_class_set	unpriv_socket_class_set	stat_file_perms
x_file_perms	r_file_perms	rx_file_perms
ra_file_perms	link_file_perms	rw_dir_perms
mount_fs_perms	rw_socket_perms	create_socket_perms
rw_stream_socket_perms	create_stream_socket_perms	connected_socket_perms
connected_stream_socket_perms	create_netlink_socket_perms	rw_netlink_socket_perms
r_netlink_socket_perms	signal_perms	packet_perms
r_sem_perms	rw_sem_perms	create_sem_perms
r_msgq_perms	rw_msgq_perms	create_msgq_perms
r_shm_perms	rw_shm_perms	create_shm_perms

The SELinux Notebook - The Foundations

Directory (dir)		
getattr_dir_perms	setattr_dir_perms	search_dir_perms
list_dir_perms	add_entry_dir_perms	del_entry_dir_perms
create_dir_perms	rename_dir_perms	delete_dir_perms
manage_dir_perms	relabelfrom_dir_perms	relabelto_dir_perms
relabel_dir_perms		

Regular file (file)		
getattr_file_perms	setattr_file_perms	read_file_perms
mmap_file_perms	exec_file_perms	append_file_perms
write_file_perms	rw_file_perms	create_file_perms
rename_file_perms	delete_file_perms	manage_file_perms
relabelfrom_file_perms	relabelto_file_perms	relabel_file_perms

Symbolic link (lnk_file)		
getattr_lnk_file_perms	setattr_lnk_file_perms	read_lnk_file_perms
append_lnk_file_perms	write_lnk_file_perms	rw_lnk_file_perms
create_lnk_file_perms	rename_lnk_file_perms	delete_lnk_file_perms
manage_lnk_file_perms	relabelfrom_lnk_file_perms	relabelto_lnk_file_perms
relabel_lnk_file_perms		

(Un)named Pipes/FIFOs (fifo_file)		
getattr_fifo_file_perms	setattr_fifo_file_perms	read_fifo_file_perms
append_fifo_file_perms	write_fifo_file_perms	rw_fifo_file_perms
create_fifo_file_perms	rename_fifo_file_perms	delete_fifo_file_perms
manage_fifo_file_perms	relabelfrom_fifo_file_perms	relabelto_fifo_file_perms
relabel_fifo_file_perms		

(Un)named Sockets (sock_file)		
getattr_sock_file_perms	setattr_sock_file_perms	read_sock_file_perms
write_sock_file_perms	rw_sock_file_perms	create_sock_file_perms
rename_sock_file_perms	delete_sock_file_perms	manage_sock_file_perms
relabelfrom_sock_file_perms	relabelto_sock_file_perms	relabel_sock_file_perms

Block device nodes (blk_file)		
getattr_blk_file_perms	setattr_blk_file_perms	read_blk_file_perms
append_blk_file_perms	write_blk_file_perms	rw_blk_file_perms
create_blk_file_perms	rename_blk_file_perms	delete_blk_file_perms
manage_blk_file_perms	relabelfrom_blk_file_perms	relabelto_blk_file_perms
relabel_blk_file_perms		

Character device nodes (chr_file)		
getattr_chr_file_perms	setattr_chr_file_perms	read_chr_file_perms
append_chr_file_perms	write_chr_file_perms	rw_chr_file_perms
create_chr_file_perms	rename_chr_file_perms	delete_chr_file_perms
manage_chr_file_perms	relabelfrom_chr_file_perms	relabelto_chr_file_perms
relabel_chr_file_perms		

Miscellaneous permissions		
rw_term_perms	client_stream_socket_perms	server_stream_socket_perms
all_capabilities	all_nscd_perms	all_dbus_perms
all_passwd_perms	all_association_perms	manage_key_perms

12. Appendix E – NetLabel Module Support for network_peer_controls

12.1 Introduction

This is an enhanced NetLabel module to enable a NetLabel netlabel_peer_t label to be added to the network connection.

The [previous NetLabel module](#) used the standard F-10 Policy Capabilities⁶⁵ network_peer_controls (set to '0'). This exercise will set the network_peer_controls to '1' by updating the base module with a [policycap](#) statement, allowing the use of these new controls.

12.2 Configuration

The following steps are required to build the enhanced NetLabel module, it is assumed that the NetLabel services have already been installed from the previous NetLabel module exercise.

1. Ensure you are logged on as 'root' and SELinux is running in permissive mode (setenforce 0) to perform the build process.
2. Edit the ./notebook-source/modular-base-policy/base.conf file to remove the '#' from the policycap statement as shown:

```
#
# This policycap statement will be used in a netlabel module exercise
# to show network_peer_controls. For now comment out:
policycap network_peer_controls;
```

3. Compile and link the base module so that the network_peer_controls are enabled:

```
checkmodule -o base.mod base.conf
semodule_package -o base.pp -m base.mod -f base.fc -s seusers -u users_extra
semodule -s modular-test -b base.pp
```

4. The following command will return '1' if the policy enabled the network_peer_controls:

```
cat /selinux/policy_capabilities/network_peer_controls
1
```

5. Produce a netlabel_policycap.conf loadable module file with a text editor containing the contents shown below:

```
module netlabel 2.0.0;

#
#####
#
# This Loadable Module will allow the netlabels to be added and checked #
# within the client / server applications that form part of the SECMARK #
```

⁶⁵ See the [SELinux Filesystem](#) section.

```
# test examples. #
# #
# The following needs to happen to enable Netlabel to work as it is not #
# installed by default in F-10: #
# #
# (1) Download and install netlabel_tools-018-1.fc10.i386.rpm (or equiv) #
# #
# (2) Install this loadable module. #
# #
# (3) Run the following netlabelctl command: #
#     netlabelctl unlbl add interface:lo address:127.0.0.1 \ #
#         label:system_u:object_r:netlabel_peer_t #
# #
# (4) Run netlabelctl -p unlbl list command to check all is okay. #
# #
# (5) Run the secure and standard client/server that should now display #
#     the netlabel_peer_t as the peer context. #
# #
# Important note: The polycap network_peer_controls; statement must be #
#     added to the base policy before the peer object can be #
#     used, otherwise the tcp_socket object will be used #
#     instead: #
# /selinux/policy_capabilities/network_peer_controls = 0 (use tcp_socket) #
# /selinux/policy_capabilities/network_peer_controls = 1 (use peer) #
# #
#####
#
require {
    type ext_gateway_t, unconfined_t;
    class peer { recv };
    class netif { ingress egress };
    class node { recvfrom sendto};
}

# Use this to label the peer level:
type netlabel_peer_t;

# These are used when /selinux/policy_capabilities/network_peer_controls =
1

# These are for unconfined_t ports:
allow unconfined_t netlabel_peer_t : peer recv;
allow netlabel_peer_t unconfined_t : netif ingress;
allow netlabel_peer_t unconfined_t : node recvfrom;

# These are for the external gateway port:
allow ext_gateway_t netlabel_peer_t : peer recv;
allow ext_gateway_t unconfined_t : netif egress;
allow ext_gateway_t unconfined_t : node sendto;

#
##### START OPTIONAL SECTION #####
#
optional {
    require {
        # This is defined in the int_gateway.conf module:
        type int_gateway_t;
    }
    allow int_gateway_t netlabel_peer_t : peer recv;
    allow int_gateway_t unconfined_t : netif egress;
    allow int_gateway_t unconfined_t : node sendto;
}
#
##### END OPTIONAL SECTION #####
#
```

6. Compile and link the new NetLabel module:

```
checkmodule -m netlabel_policycap.conf -o netlabel.mod
semodule_package -o netlabel.pp -m netlabel.mod
```

```
semodule -v -s modular-test -i netlabel.pp
```

8. Run the following command to add the `netlabel_peer_t` label as follows:

```
netlabelctl unlbl add interface:lo address:127.0.0.1 \  
label:system_u:object_r:netlabel_peer_t
```

9. Run enforcing mode:

```
setenforce 1
```

10. Run either the client / server or `secure_client` / `secure_server` applications as shown in the [SECMARK tests](#). There should now be a peer context displayed as shown in the ‘With Peer Context (NetLabel)’ section of [Figure 11-41](#).

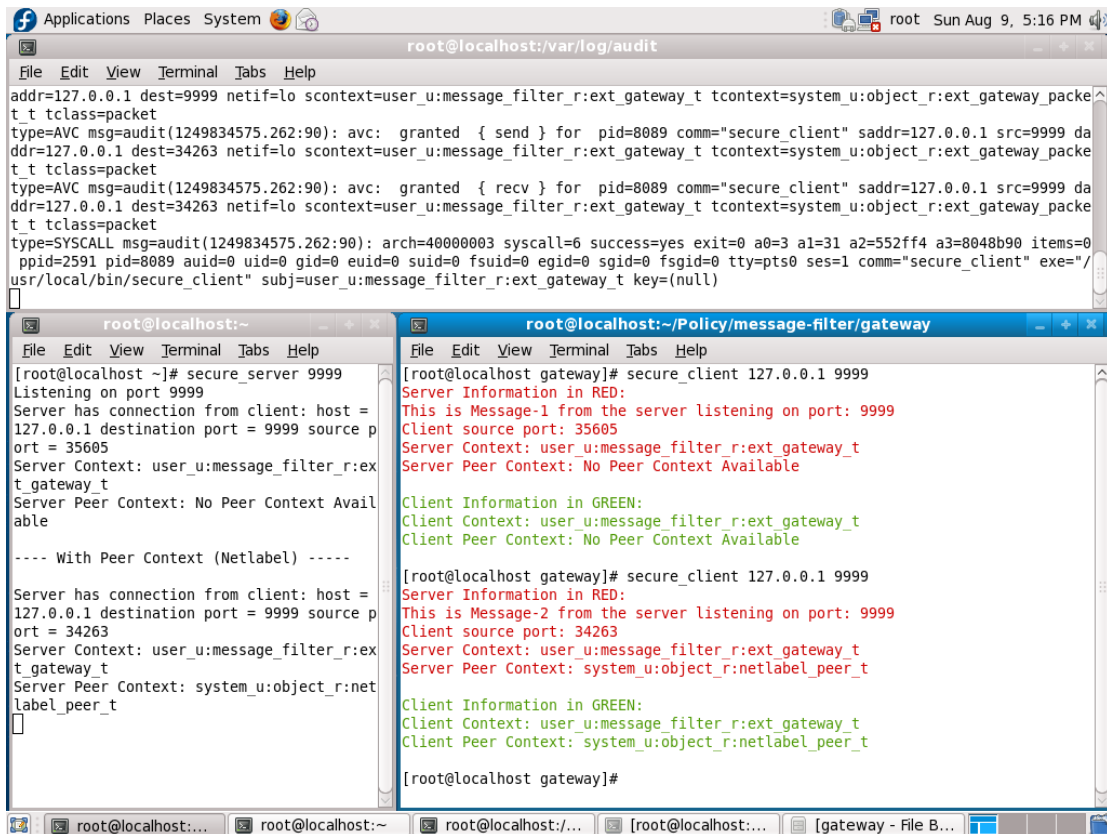


Figure 11-41: Running the secure client / server – Once with no NetLabel and once with NetLabel enabled.

13. Appendix F – Labeled IPSec Module Example

13.1 Introduction

This section shows a sample IPSec module and configuration files that have been built to support the simple message filter. It is in two parts:

Manual Configuration – This shows the files required to configure IPSec manually making all the entries in the SAD and SPD databases. Important note: The encryption keys are pre-generated. If this type of configuration is to be used, then generate new keys as described in the [IPSec-HOWTO](#) [Ref. 12].

Key Exchange Configuration – This shows the configuration files required for `racoon` to manage the key exchange and security context. Unfortunately, `racoon` core dumps on F-10 using the `modular-test` policy (but does work with Red Hat targeted policy - The reason seems to be linked with using loopback to run IPSec. When an MCS / MLS policy is used with loopback it works, however if MCS or MLS is not configured it core-dumps).

Notes:

1. F-10 does not have IPSec tools installed as standard, therefore `yum` can be used to install it as shown below:

```
yum install ipsec_tools

# yum will then install ipsec_tools-0.7.2-1.fc10.i386.rpm
# or a later version.
```

2. The IPSec configuration files have entries for the Internal Gateway (`int_gateway_t`). If this module is not loaded, then the entries need to be removed.
3. F-10 does not have IPSec services enabled for loopback by default, therefore the following commands need to be run:

```
echo 0 > /proc/sys/net/ipv4/conf/lo/disable_xfrm
echo 0 > /proc/sys/net/ipv4/conf/lo/disable_policy
```

Be aware though that this re-configuration will only be valid until the next re-boot.

13.2 Manual IPSec Configuration

The steps required to install the module and configure IPSec are as follows:

1. Ensure you are logged on as 'root' and SELinux is running in permissive mode (`setenforce 0`) to perform the build process.
2. Produce a `ipsec.conf` loadable module file with a text editor containing the contents shown below:

```
module ipsec 1.0.0;
#
#####
```



```
# #
# This Loadable Module will allow Labeled IPsec to manage labeling for #
# the client / server applications that form part of the test examples #
# in the SELinux Notebook. #
# #
#####
#
require {
    type ext_gateway_t, unconfined_t;
    class association { setcontext polmatch sendto recvfrom };
}

# This allows unconfined to set the SPD and SAD context entries:
allow unconfined_t ext_gateway_t : association { setcontext };

# This allows the external gateway to work with Labeled IPsec:
allow ext_gateway_t self : association { setcontext polmatch sendto
recvfrom };

# Allows Racoon running in unconfined_t to polmatch (in fact
# racoon needs to polmatch all entries):
allow unconfined_t ext_gateway_t:association polmatch;

#
##### START OPTIONAL SECTION (for internal gateway) #####
#
optional {
    require {
        # This is defined in the int_gateway.conf module:
        type int_gateway_t;
    }
    allow unconfined_t int_gateway_t:association { setcontext polmatch };
    allow int_gateway_t self : association { setcontext polmatch sendto
recvfrom };
}
#
##### END OPTIONAL SECTION #####
#
```

3. Compile and link the new IPsec module:

```
checkmodule -m ipsec.conf -o ipsec.mod
semodule_package -o ipsec.pp -m ipsec.mod
semodule -v -s modular-test -i ipsec.pp
```

4. Create an IPsec configuration file (`ipsec_manual_SA`) that will generate both the SAD and SPD database entries that allow IPsec to be configured manually:

```
# setkey -f configuration file entries for MANUAL SA configuration
#
# If the Internal Gateway module (int_gateway.conf) is not loaded,
# then the entries should be removed from this file.
#
# Flush the SAD and SPD
flush;
spdf flush;

#
##### Security Association Database entries #####
#
# Important notes:
# 1) The security context (-ctx) entries MUST match
#    the actual running context of the process or it will fail to
#    match (therefore racoon is the best configuration option as
#    these are automatically exchanged).
# 2) If the manual configuration is used in a live environment,
```

```
# then DO NOT use these encryption keys, generate your own.
#
# Authentication Header info
# AH SAs using 128 bit long keys
add 127.0.0.1 127.0.0.1 ah 0x200
-ctx 1 1 "user_u:message_filter_r:ext_gateway_t"
-A hmac-md5 0xc0291ff014dccdd03874d9e8e4cdf3e6;

add 127.0.0.1 127.0.0.1 ah 0x250
-ctx 1 1 "user_u:message_filter_r:int_gateway_t"
-A hmac-md5 0xc0291ff014dccdd03874d9e8e4cdf3e6;

add 127.0.0.1 127.0.0.1 ah 0x300
-ctx 1 1 "user_u:unconfined_r:unconfined_t"
-A hmac-md5 0x96358c90783bbfa3d7b196ceabe0536b;
#
# Encapsulated Security Payload info
# The -ctx context MUST be exact else get "connect: No such process"
# message when running client:
# ESP SAs using 192 bit long keys (168 + 24 parity)
add 127.0.0.1 127.0.0.1 esp 0x201
-ctx 1 1 "user_u:message_filter_r:ext_gateway_t"
-E 3des-cbc 0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831;

add 127.0.0.1 127.0.0.1 esp 0x251
-ctx 1 1 "user_u:message_filter_r:int_gateway_t"
-E 3des-cbc 0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831;

add 127.0.0.1 127.0.0.1 esp 0x301
-ctx 1 1 "user_u:unconfined_r:unconfined_t"
-E 3des-cbc 0xf6ddb555acfd9d77b03ea3843f2653255afe8eb5573965df;

#
##### Security Policy Database entries #####
#
# Note that the only part of the security context matched against is
# the 'type' (e.g. ext_gateway_t).

# Security policies for external gateway:
spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "system_u:object_r:ext_gateway_t"
-P out ipsec esp/transport//require
ah/transport//require;

spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "system_u:object_r:ext_gateway_t"
-P in ipsec esp/transport//require
ah/transport//require;

# Security policies for internal gateway:
spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "system_u:object_r:int_gateway_t"
-P out ipsec esp/transport//require
ah/transport//require;

spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "system_u:object_r:int_gateway_t"
-P in ipsec esp/transport//require
ah/transport//require;

# Security policies for unconfined_t:
spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "system_u:object_r:unconfined_t"
-P out ipsec esp/transport//require
ah/transport//require;

spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "system_u:object_r:unconfined_t"
-P in ipsec esp/transport//require
ah/transport//require;
```

5. Activate the IPSec configuration by running setkey:

```
setkey -f ipsec_manual_SA
```

6. The configuration can be checked using the setkey commands as shown:

```
# This command will list the Security Association
# Database entries:

setkey -D

# A list should follow that starts:

127.0.0.1 127.0.0.1
ah mode=transport spi=512(0x00000200) reqid=0(0x00000000)
A: hmac-md5 c0291ff0 14dccdd0 3874d9e8 e4cdf3e6
seq=0x00000000 replay=0 flags=0x00000000 state=mature
created: Sep 14 16:48:48 2009 current: Sep 14 16:49:10 2009
diff: 22(s)hard: 0(s) soft: 0(s)
last:
current: 0(bytes) hard: 0(bytes) soft: 0(bytes)
allocated: 0 hard: 0 soft: 0
security context doi: 1
security context algorithm: 1
security context length: 38
security context: user_u:message_filter_r:ext_gateway_t
sadb_seq=1 pid=3216 refcnt=0
.....
....
```

```
# This command will list the Security Policy
# Database entries:

setkey -DP

# A list should follow that starts:

127.0.0.1[any] 127.0.0.1[any] tcp
out prio def ipsec
esp/transport//require
ah/transport//require
created: Sep 14 16:48:48 2009 lastused:
lifetime: 0(s) validtime: 0(s)
security context doi: 1
security context algorithm: 1
security context length: 32
security context: system_u:object_r:ext_gateway_t
spid=209 seq=1 pid=3219
refcnt=1
.....
.....
```

7. Because the IPSec service has not been enabled in F-10 for loopback, the following commands need to be run:

```
# The default for F-10 is that ipsec is disabled for
# loopback. These commands will enable until a re-boot:

echo 0 > /proc/sys/net/ipv4/conf/lo/disable_xfrm
echo 0 > /proc/sys/net/ipv4/conf/lo/disable_policy
```

8. Run enforcing mode:

```
setenforce 1
```

- Run either the client / server or secure_client / secure_server applications as shown in the [SECMARK tests](#). There should now be a peer context displayed as shown in the ‘With Labeled IPsec Peer Context’ section of [Figure 12-42](#).

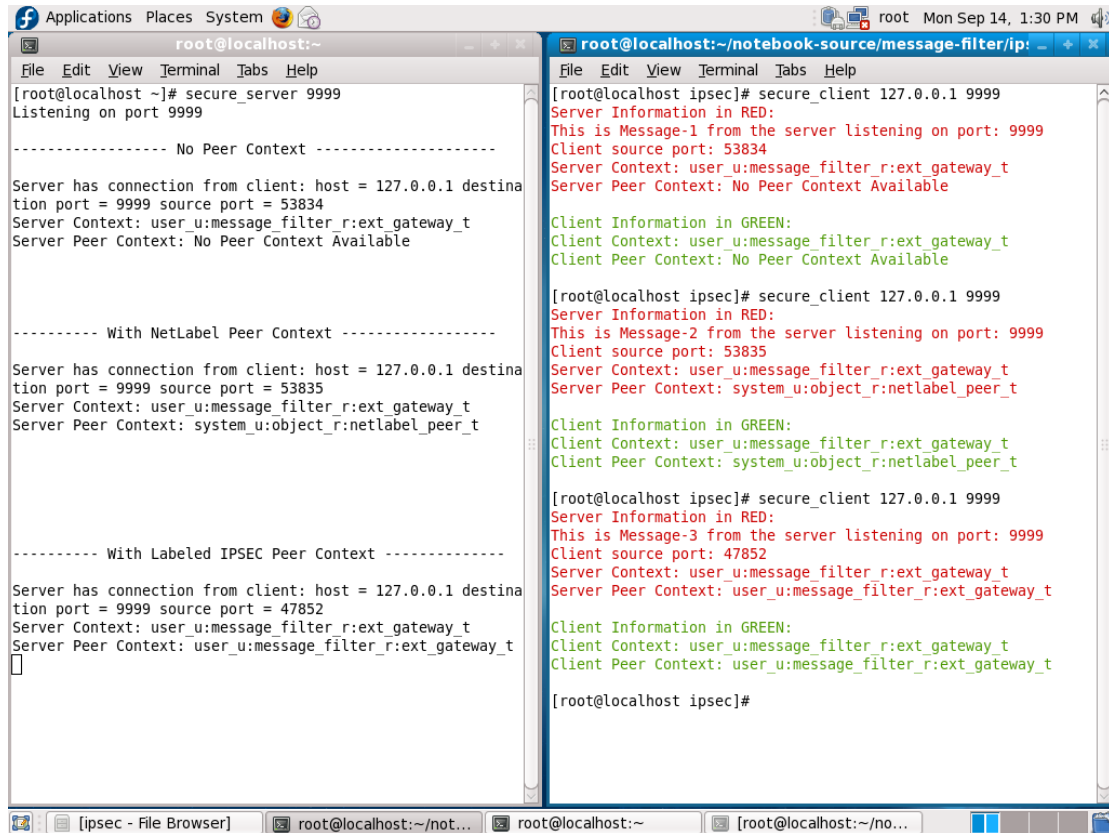


Figure 12-42: Running the secure client / server – Once with no NetLabel, once with NetLabel enabled and once with Labeled IPsec enabled (note that this label takes precedence over the Fallback NetLabel).

13.3 Key Exchange IPsec Configuration

The configuration requirements are shown below, however as mentioned above the racoon IKE daemon will core dump when using the simple policy configured as shown in the [Building a Basic Policy](#) section. The steps required to install the module and configure IPsec are as follows:

- Perform steps 1, 2 and 3 as for the manual configuration to build the IPsec module.
- Create an IPsec configuration file (`ipsec_racoon_SA`) that will generate only the SPD database entries. The SAD entries will be populated by racoon as it exchanges the key and context information:

```
# setkey -f configuration file entries for RACOON SA configuration
#
# Flush the SAD and SPD
flush;
spdflush;

#
##### Security Policy Database entries #####
#
```

```
# Security policies for external gateway:
spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "system_u:object_r:ext_gateway_t"
-P out ipsec esp/transport//require
ah/transport//require;

spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "system_u:object_r:ext_gateway_t"
-P in ipsec esp/transport//require
ah/transport//require;

# Security policies for internal gateway:
spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "system_u:object_r:int_gateway_t"
-P out ipsec esp/transport//require
ah/transport//require;

spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "system_u:object_r:int_gateway_t"
-P in ipsec esp/transport//require
ah/transport//require;

# Security policies for unconfined_t:
spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "system_u:object_r:unconfined_t"
-P out ipsec esp/transport//require
ah/transport//require;

spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "system_u:object_r:unconfined_t"
-P in ipsec esp/transport//require
ah/transport//require;
```

3. Activate the IPsec configuration by running setkey:

```
setkey -f ipsec_racoon_SA
```

4. The configuration can be checked using the setkey commands as shown:

```
# This command will list the Security Association
# Database entries:

setkey -D
No SAD entries.

# Note that there should be NO SAD entries as racoon will
# add these during the key exchange process.
```

```
# This command will list the Security Policy
# Database entries:

setkey -DP

# A list should follow that starts:
```

5. Because the IPsec service has not been enabled in F-10 for loopback, the following commands need to be run:

```
# The default for F-10 is that ipsec is disabled for
# loopback. These commands will enable until a re-boot:
```

```
echo 0 > /proc/sys/net/ipv4/conf/lo/disable_xfrm
echo 0 > /proc/sys/net/ipv4/conf/lo/disable_policy
```

6. Check the `/etc/racoon/racoon.conf` file. For F-10 its contents should resemble that shown (the commented out sections have been removed). This describes the key exchange as an anonymous exchange and therefore should work with loopback. If the contents of the `racoon.conf` file are different, then save the file and replace the contents with that below:

```
# Racoon IKE daemon configuration file.
# See 'man racoon.conf' for a description of the format and
entries.

path include "/etc/racoon";
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";
path script "/etc/racoon/scripts";

sainfo anonymous
{
  lifetime time 1 hour ;
  encryption_algorithm 3des, blowfish 448, rijndael ;
  authentication_algorithm hmac_shal, hmac_md5 ;
  compression_algorithm deflate ;
}
```

7. Start a new virtual terminal session so that the `racoon` service can be run using the command below (in the foreground with debug). If F-10 is being used with the `ipsec_tools-0.7.2-1.fc10.i386` tools, then the chances are `racoon` will core dump once the client tries to contact the server !!

```
racoon -Fd
```

8. Run enforcing mode:

```
setenforce 1
```

9. Run either the client / server or `secure_client` / `secure_server` applications as shown in the [SECMARK tests](#). There should be a peer context displayed as shown in the 'With Labeled IPSec Peer Context' section of [Figure 12-42](#) (however `racoon` core dumps once the client tries to contact the server).

14. Appendix G – Implementing a Constraint

14.1 Introduction

The objective of this section is to show how a constraint can further limit access permissions. The example given will add a simple role constraint to the base policy described in the [Building the Base Module Policy](#) section by adding the following:

```
constrain process transition ( r1 == r2 );
```

The impact of this constraint will be that when a transition is required, the role of the source (or current) process must be the same as the role of the target process (or new process being exec'ed).

While the majority of applications will load and execute when using the example base policy, when attempting to load the external gateway module described in the [Building the SECMARK Test Loadable Module](#) section, the result will be:

```
# Ensure enforcement mode:
setenforce 1

# Run the external gateway:
secure_server 9999

# The following will be displayed with the constraint enforced:
bash: /usr/local/bin/secure_server: Permission denied
```

This is because the roles are not equal as the source process is the shell running with `unconfined_r` and the external gateway with `message_filter_r`.

14.2 Configuration

The following steps are required to add the constraint to the base policy and test the results:

1. Ensure you are logged on as 'root' and SELinux is running in permissive mode (`setenforce 0`) to perform the build process.
2. Edit the `./notebook-source/modular-base-policy/base.conf` file to remove the '#' from the constraint statement as shown:

```
#
# This role constraint statement will be used to show limiting
# a role transition in the external gateway. For now comment out:
#
constrain process transition ( r1 == r2 );
```

3. Compile and link the base module so that the constraint is enabled:

```
checkmodule -o base.mod base.conf
semodule_package -o base.pp -m base.mod -f base.fc -s seusers -u users_extra
semodule -s modular-test -b base.pp
```

4. Run enforcing mode:

```
setenforce 1
```

5. Run the external gateway as shown:

```
secure_server 9999
```

6. The following error should be displayed as the source and target process roles are not equal:

```
bash: /usr/local/bin/secure_server: Permission denied
```

7. The role constraint (or any other if required) can be tried using different operators such as `!=`, `dom`, `domby` or `incomp` as described in the [constrain statement](#) section. Note the only other operator that will stop the transition is ‘`domby`’, however this can be overcome by adding a further line to the base policy to implement the [role dominance rule](#) as shown (but note that this rule has been deprecated and `checkmodule` will issue a warning):

```
# Add the dominance rule after the following statement:
allow unconfined_t self:x_application_data *;
#
dominance { role message_filter_r { role unconfined_r };}
```

8. Once testing has been completed it is recommended that the `constrain` and `dominance` statements are removed and the policy rebuilt.

14.3 Reference Policy Constraints Information

The reference policy source has all the constraints listed in the `policy/constraints` file and the MLS / MCS constraints listed in the `policy/mls` and `policy/mcs` files respectively. All these constraints make extensive use of attributes to hold the types to be managed.

For example in the `policy/constraints` file used by F-10, role changing for transitions are managed by the `constrain` statement as shown:

```
#
# SELinux process role change constraint:
#
constrain process transition
(
  r1 == r2
  or ( t1 == can_change_process_role and t2 == process_user_target )
    or ( t1 == cron_source_domain and t2 == cron_job_domain )
    or ( t1 == can_system_change and r2 == system_r )
    or ( t1 == process_uncond_exempt )
);
```

As can be seen the ‘`r1 == r2`’ is what was used in the external gateway example above, however to allow other scenarios, attributes are used to ‘hold’ the types that can change the constraint conclusion. For example, if the external gateway module was written as a reference policy source module, then to allow the role change:

- The `unconfined_t` domain type could be added to the `can_change_process_role` attribute and the `ext_gateway_t` domain type added to the `process_user_target` attribute.

or

- The `unconfined_t` domain type could be added to the `process_uncond_exempt` attribute.

Either of these would allow the transition to take place (need to check these are right way round !!!).

15. Appendix H - Bugs and Features

This section lists the bugs and / or features found while writing this Notebook.

15.1 OpenOffice.org - Writer

This Notebook is written using OpenOffice.org Writer 3.1.1 (Build 9420). The main problem found is that when the document is loaded, sometimes (by magic) the page formatting gets screwed in that pages in portrait switch to landscape. Sometimes re-loading fixes the problem, other times pages have to be re-formatted. Also some of the diagrams tend to straddle pages at random.

15.2 semanage – node configuration

The `semanage node` command fails to add an entry. The example used and the results were as follows (this has been fixed in a later release):

```
semanage node -a -t unconfined_t -p ipv4 -M 255.255.255.0 127.0.0.1

Traceback (most recent call last):
  File "/usr/sbin/semanage", line 477, in <module>
    process_args(sys.argv[1:])
  File "/usr/sbin/semanage", line 371, in process_args
    OBJECT.add(target, mask, proto, serange, setype)
  File "/usr/lib/python2.5/site-packages/seobject.py", line 1093, in add
    self.__add(self, addr, mask, proto, serange, ctype)
TypeError: __add() takes exactly 6 arguments (7 given)
```

15.3 semanage - roles get deleted

The `semanage user` command deletes roles already assigned. To add a role of `message_filter_r` it seems logical to do the following:

```
semanage user -m -R "message_filter_r" user_u
```

However need to know the other roles and add these as well, otherwise `semanage` will remove them from the current policy:

```
semanage user -m -R "message_filter_r unconfined_r" user_u
```

15.4 checkmodule - neverallow not picked up in modules

The `checkmodule` compiler does not pick `neverallow` statements in modules even though they are valid (from what the author can determine anyway !!). Works fine in base module.

15.5 apol not showing all screen in window

On the authors system the policy analysis tool `apol` never displayed all the information in the window. Various configuration items were tried (editing the

.apol file, changing screen formats etc.), but nothing worked. After further investigation (recompiling source etc.) it was found that the easiest way was to edit the /usr/bin/apol file (that is a text file) to use smaller font sizes. To achieve this use the editors search facility to find all 'helvetica' entries in the /usr/bin/apol file (there should be 30 of them) and change all the font sizes to '8'. Once this is complete, then load apol and the before and after results are shown in [Figure 15-43](#).

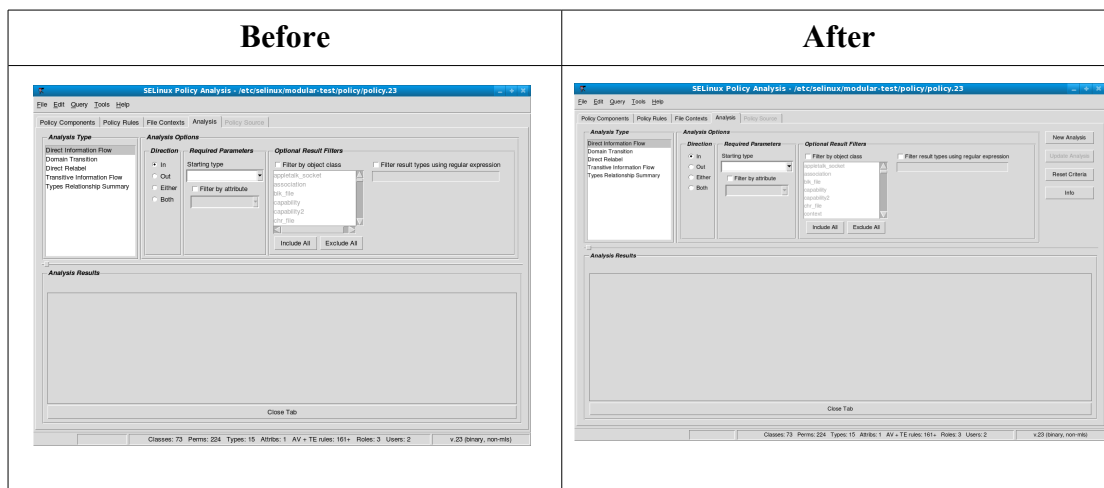


Figure 15-43: apol before and after the font size change.

15.6 racoon core dumps

As detailed in [Appendix F – Labeled IPsec Module Example](#) racoon core dumps when using loopback with policies that do not support MCS or MLS services. Some of the tests listed in the <http://sourceforge.net/projects/audit-test> audit-test-2090-1.src.rpm were carried out and those using ports without MCS / MLS being configured do not work on loopback either.

Running racoon -Fd as described in [Appendix F – Labeled IPsec Module Example](#) provided the following trace:

```
[root@localhost ipsec]# racoon -Fd
Foreground mode.
2009-09-13 15:52:34: INFO: @(#)ipsec-tools 0.7.2 (http://ipsec-tools.sourceforge.net)
2009-09-13 15:52:34: INFO: @(#)This product linked OpenSSL 0.9.8g 19 Oct 2007 (http://www.openssl.org/)
2009-09-13 15:52:34: INFO: Reading configuration from "/etc/racoon/racoon.conf"
2009-09-13 15:52:34: DEBUG: call pfkey_send_register for AH
2009-09-13 15:52:34: DEBUG: call pfkey_send_register for ESP
2009-09-13 15:52:34: DEBUG: call pfkey_send_register for IPCOMP
2009-09-13 15:52:34: DEBUG: reading config file /etc/racoon/racoon.conf
2009-09-13 15:52:34: DEBUG: compression algorithm can not be checked because sadb message doesn't support
it.
2009-09-13 15:52:34: DEBUG: getsainfo params: loc='ANONYMOUS', rmt='ANONYMOUS', peer='NULL', id=0
2009-09-13 15:52:34: DEBUG: getsainfo pass #2
2009-09-13 15:52:34: DEBUG: filename: /etc/racoon/127.0.0.1.conf
2009-09-13 15:52:34: DEBUG: reading config file /etc/racoon/127.0.0.1.conf
2009-09-13 15:52:34: DEBUG: hmac(modp1024)
2009-09-13 15:52:34: DEBUG: open /var/racoon/racoon.sock as racoon management.
2009-09-13 15:52:34: DEBUG: my interface: ::1 (lo)
2009-09-13 15:52:34: DEBUG: my interface: 127.0.0.1 (lo)
2009-09-13 15:52:34: DEBUG: configuring default isakmp port.
2009-09-13 15:52:34: DEBUG: 2 addr's are configured successfully
2009-09-13 15:52:34: INFO: 127.0.0.1[500] used as isakmp port (fd=7)
2009-09-13 15:52:34: INFO: 127.0.0.1[500] used for NAT-T
2009-09-13 15:52:34: INFO: ::1[500] used as isakmp port (fd=8)
2009-09-13 15:52:34: DEBUG: pk_recv: retry[0] recv()
2009-09-13 15:52:34: DEBUG: get pfkey X_SPDDUMP message
2009-09-13 15:52:34: DEBUG: pk_recv: retry[0] recv()
2009-09-13 15:52:34: DEBUG: get pfkey X_SPDDUMP message
2009-09-13 15:52:34: DEBUG: sub:0xbf9fe5bc: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=fwd
sec_ctx:doi=1,alg=1,len=31,str=system_u:object_r:unconfined_t
```

```
2009-09-13 15:52:34: DEBUG: db :0x1fa69a8: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=in
sec_ctx:doi=1,alg=1,len=31,str=system_u:object_r:unconfined_t
2009-09-13 15:52:34: DEBUG: pk_recv: retry[0] recv()
2009-09-13 15:52:34: DEBUG: get pfkey X_SPDDUMP message
2009-09-13 15:52:34: DEBUG: sub:0xbf9fe5bc: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=out
sec_ctx:doi=1,alg=1,len=31,str=system_u:object_r:unconfined_t
2009-09-13 15:52:34: DEBUG: db :0x1fa69a8: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=in
sec_ctx:doi=1,alg=1,len=31,str=system_u:object_r:unconfined_t
2009-09-13 15:52:34: DEBUG: sub:0xbf9fe5bc: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=out
sec_ctx:doi=1,alg=1,len=31,str=system_u:object_r:unconfined_t
2009-09-13 15:52:34: DEBUG: db :0x1fa77e8: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=fwd
sec_ctx:doi=1,alg=1,len=31,str=system_u:object_r:unconfined_t
2009-09-13 15:52:39: DEBUG: pk_recv: retry[0] recv()
2009-09-13 15:52:39: DEBUG: get pfkey ACQUIRE message
2009-09-13 15:52:39: INFO: security context doi: 1
2009-09-13 15:52:39: INFO: security context algorithm: 1
2009-09-13 15:52:39: INFO: security context length: 33
2009-09-13 15:52:39: INFO: security context: user u:unconfined_r:unconfined_t
2009-09-13 15:52:39: DEBUG: suitable outbound SP found: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=out
sec_ctx:doi=1,alg=1,len=31,str=system_u:object_r:unconfined_t.
2009-09-13 15:52:39: DEBUG: sub:0xbf9fe5c0: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=in
sec_ctx:doi=1,alg=1,len=33,str=user_u:unconfined_r:unconfined_t
2009-09-13 15:52:39: DEBUG: db :0x1fa69a8: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=in
sec_ctx:doi=1,alg=1,len=31,str=system_u:object_r:unconfined_t
2009-09-13 15:52:39: DEBUG: sub:0xbf9fe5c0: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=in
sec_ctx:doi=1,alg=1,len=33,str=user_u:unconfined_r:unconfined_t
2009-09-13 15:52:39: DEBUG: db :0x1fa77e8: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=fwd
sec_ctx:doi=1,alg=1,len=31,str=system_u:object_r:unconfined_t
2009-09-13 15:52:39: DEBUG: sub:0xbf9fe5c0: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=in
sec_ctx:doi=1,alg=1,len=33,str=user_u:unconfined_r:unconfined_t
2009-09-13 15:52:39: DEBUG: db :0x1fa8740: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=out
sec_ctx:doi=1,alg=1,len=31,str=system_u:object_r:unconfined_t
2009-09-13 15:52:39: NOTIFY: no in-bound policy found: 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=in
sec_ctx:doi=1,alg=1,len=33,str=user_u:unconfined_r:unconfined_t
2009-09-13 15:52:39: DEBUG: new acquire 127.0.0.1/32[0] 127.0.0.1/32[0] proto=any dir=out
sec_ctx:doi=1,alg=1,len=31,str=system_u:object_r:unconfined_t
Segmentation fault
```

Running the core dump with `gdb racoon core.3451` gives:

```
#0 0x0078616b in pk_recvacquire (mhp=0xbfc7a774) at pfkey.c:2048
2048          newpr->reqid_in = sp_in->req->saidx.reqid;
```

15.7 Red Hat SELinux Config Utility

The `system-config-selinux` utility does not like non MCS / MLS policies.

16. Appendix I - Useful Information

16.1 Building the Source rpm

To create the source rpm for the Notebook source (assuming the source is in \$HOME/notebook-source):

1. Create the notebook-source compressed archive using the tar command:

```
tar -cvzf notebook-source-1.0.0-1.tar.gz notebook-source/
```

2. Move the archive to the \$HOME/rpmbuild/SOURCES directory:

```
mv notebook-source-1.0.0-1.tar.gz rpmbuild/SOURCES
```

3. Create the notebook.spec file using an editor in the \$HOME/rpmbuild/SPECS directory with the following content:

```
Summary: Software for The SELinux Notebook - The
Foundations
Name: notebook-source
Version: 1.0.0
Release: 1
License: GPLv2
Group: Experimental
Source0: %{name}-%{version}-%{release}.tar.gz
%description
This contains the sample source code and modules that are
detailed in The SELinux Notebook - The Foundations
%prep

%files
%defattr(-,root,root,-)
```

4. Build the rpm file from the \$HOME/rpmbuild/SPECS directory by running the rpmbuild command as follows, the results are also shown:

```
rpmbuild -ba notebook.spec

Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.41L8I5
+ umask 022
+ cd /root/rpmbuild/BUILD
+ LANG=C
+ export LANG
+ unset DISPLAY
+ exit 0
Processing files: notebook-source-1.0.0-1
Checking for unpackaged file(s): /usr/lib/rpm/check-
files /root/rpmbuild/BUILDROOT/notebook-source-1.0.0-1.i386
Wrote: /root/rpmbuild/SRPMS/notebook-source-1.0.0-1.src.rpm
Wrote: /root/rpmbuild/RPMS/i386/notebook-source-1.0.0-
1.i386.rpm
```

The only file of interest is the `/root/rpmbuild/SRPMS/notebook-source-1.0.0-1.src.rpm` as that contains the source files, the `i386` rpm will be empty as no code was built for distribution.

The rpm can be checked, installed and uncompressed on a different system by executing the following commands:

```
# This command will show the contents of the rpm:
rpm -qlp notebook-source-1.0.0-1.src.rpm

notebook-source-1.0.0-1.tar.gz
notebook.spec
```

```
# This command will extract the rpm contents into:
#     $HOME/rpmbuild/SOURCES:

rpm -Uvh notebook-source-1.0.0-1.src.rpm
```

```
# This command will uncompress the archive and install the
# contents:
tar -xzf notebook-source-1.0.0-1.tar.gz
```

17. Appendix J – References

17.1 Document References

<i>Ref. No.</i>	<i>Title</i>	<i>Author</i>
1.	Fedora 10 – SELinux User Guide	Red Hat
2.	Gentoo SELinux Handbook	Gentoo
3.	Security-Enhanced PostgreSQL Security Wiki	SE-Postgre
4.	SELinux Policy Module Primer	J. Brindle
5.	Polyinstantiation of directories in an SELinux system	R. Coker
6.	Implementing SELinux as a Linux Security Module	S. Smalley, C. Vance, W. Salamon
7.	Iptables Tutorial	O. Andreasson
8.	New secmark-based network controls for SELinux	J. Morris
9.	Transitioning to Secmark	Paul Moore
10.	Fallback Label Configuration Example	Paul Moore
11.	Leveraging IPsec for Distributed Authorization	Trent Jaeger
12.	IPsec HOWTO	Ralf Spenneberg
13.	Secure Networking with SELinux	J. Brindle
14.	SELinux by Example	F. Mayer, K Macmillan, D. Caplan
15.	SELinux From Scratch	S. Hallyn
16.	SELinux hardening for mmap_min_addr protections	E. Paris
17.	SELinux Support for Userspace Object Managers (this seems to have disappeared from the Internet, however the example userspace AVC code is in the Notebook source rpm).	E. Walsh

17.2 Useful Websites

<http://fedora.redhat.com/>

<http://selinuxnews.org/wp/>

http://selinuxproject.org/page/Main_Page

<http://selinuxnews.org/planet/>

<http://oss.tresys.com/projects/refpolicy/wiki/ObjectClassesPerms>

<http://kernelnewbies.org/>

<http://www.commoncriteriaportal.org/>

<http://www.nsa.gov/>

18. Appendix K - GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. Applicability and Definitions

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. Copying In Quantity

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

The SELinux Notebook - The Foundations

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. Collections Of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. Future Revisions Of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. Relicensing

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.