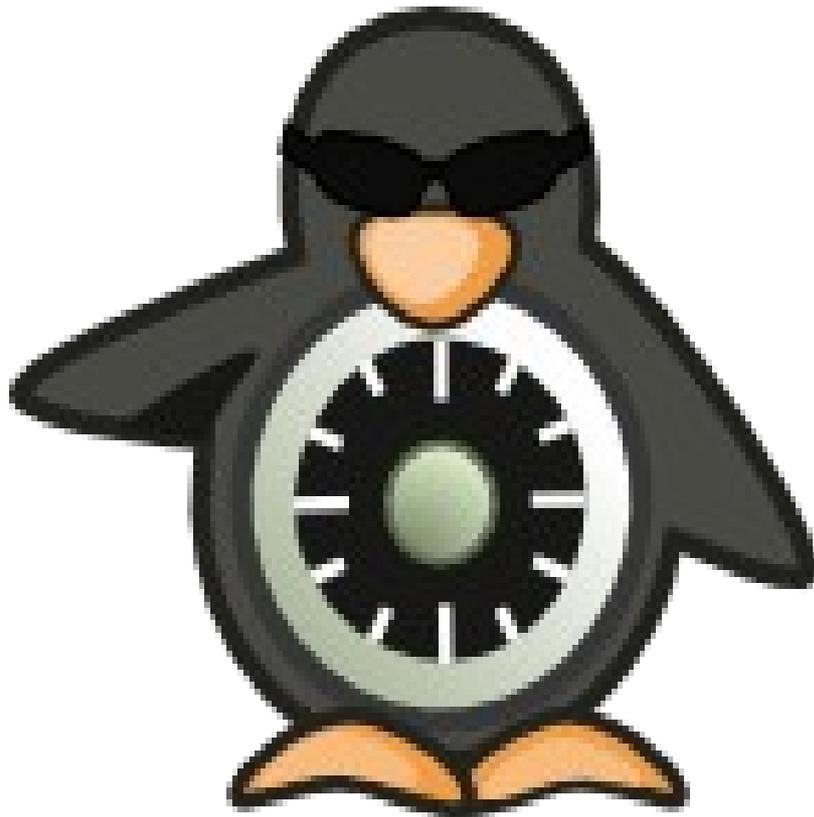


The SELinux Notebook

(4th Edition)



0. Notebook Information

0.1 Copyright Information

Copyright © 2014 [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

Edition	Date	Changes
1.0	20 th Nov '09	First released.
2.0	8 th May '10	Second release.
3.0	2 nd September '12	Third release.
4.0	30 th September '14	Fourth release.

0.3 Acknowledgements

Logo designed by [Máirín Duffy](#)

0.4 Abbreviations

AV	Access Vector
AVC	Access Vector Cache
BLP	Bell-La Padula
CC	Common Criteria
CIL	Common Intermediate Language
CMW	Compartmented Mode Workstation
DAC	Discretionary Access Control

F-20	Fedora 20
FLASK	Flux Advanced Security Kernel
Fluke	Flux μ -kernel Environment
Flux	The Flux Research Group (http://www.cs.utah.edu/flux/)
ID	Identification
LSM	Linux Security Module
LAPP	Linux, Apache, PostgreSQL, PHP / Perl / Python
LSPP	Labeled Security Protection Profile
MAC	Mandatory Access Control
MCS	Multi-Category Security
MLS	Multi-Level Security
NSA	National Security Agency
OM	Object Manager
OTA	over the air
PAM	Pluggable Authentication Module
RBAC	Role-based Access Control
rpm	Red Hat Package Manager
SELinux	Security Enhanced Linux
SID	Security Identifier
SMACK	Simplified Mandatory Access Control Kernel
SUID	Super-user Identifier
TE	Type Enforcement
UID	User Identifier
XACE	X (windows) Access Control Extension

0.5 Terminology

These give a brief introduction to the major components that form the core SELinux infrastructure.

Term	Description
Access Vector (AV)	A bit map representing a set of permissions (such as open, read, write).
Access Vector Cache (AVC)	<p>A component that stores access decisions made by the SELinux Security Server for subsequent use by Object Managers. This allows previous decisions to be retrieved without the overhead of re-computation.</p> <p>Within the core SELinux services there are two Access Vector Caches:</p> <ol style="list-style-type: none"> 1. A kernel AVC that caches decisions by the Security Server on behalf of kernel based object managers.

Term	Description
	<p>2. A userspace AVC built into libselinux that caches decisions when SELinux-aware applications use avc_open(3) with avc_has_perm(3) or avc_has_perm_noaudit(3) function calls. This will save kernel calls after the first decision has been made.</p>
Domain	<p>For SELinux this consists of one or more processes associated to the type component of a Security Context. Type Enforcement rules declared in Policy describe how the domain will interact with objects (see Object Class).</p>
Linux Security Module (LSM)	<p>A framework that provides hooks into kernel components (such as disk and network services) that can be utilised by security modules (e.g. SELinux and SMACK) to perform access control checks.</p> <p>Currently only one LSM module can be loaded, however work is in progress to stack multiple modules).</p>
Mandatory Access Control	<p>An access control mechanism enforced by the system. This can be achieved by 'hard-wiring' the operating system and applications (the bad old days - well good for some) or via a policy that conforms to a Policy. Examples of policy based MAC are SELinux and SMACK.</p>
Multi-Level Security (MLS)	<p>Based on the Bell-La & Padula model (BLP) for confidentiality in that (for example) a process running at a 'Confidential' level can read / write at their current level but only read down levels or write up levels. While still used in this way, it is more commonly used for application separation utilising the Multi-Category Security variant.</p>
Object Class	<p>Describes a resource such as files, sockets or services.</p> <p>Each 'class' has relevant permissions associated to it such as read, write or export. This allows access to be enforced on the instantiated object by their Object Manager.</p>
Object Manager	<p>Userspace and kernel components that are responsible for the labeling, management (e.g. creation, access, destruction) and enforcement of the objects under their control. Object Managers call the Security Server for an access decision based on a source and target Security Context (or SID), an Object Class and a set of permissions (or AVs). The Security Server will base its decision on whether the currently loaded Policy will allow or deny access.</p> <p>An Object Manager may also call the Security Server to compute a new Security Context or SID for an object.</p>
Policy	<p>A set of rules determining access rights. In SELinux these rules are generally written in a kernel policy language using either m4(1) macro support (e.g. Reference Policy) or the new CIL language. The Policy is then compiled into a binary</p>

Term	Description
	format for loading into the Security Server .
Role Based Access Control	SELinux users are associated to one or more roles, each role may then be associated to one or more Domain types.
Security Server	<p>A sub-system in the Linux kernel that makes access decisions and computes security contexts based on Policy on behalf of SELinux-aware applications and Object Managers.</p> <p>The Security Server does not enforce a decision, it merely states whether the operation is allowed or not according to the Policy. It is the SELinux-aware application or Object Manager responsibility to enforce the decision.</p>
Security Context	<p>An SELinux Security Context is a variable length string that consists of the following mandatory components <code>user:role:type</code> and an optional <code>[:range]</code> component.</p> <p>Generally abbreviated to 'context', and sometimes called a 'label'.</p>
Security Identifier (SID)	<p>SIDs are unique opaque integer values mapped by the kernel Security Server and userspace AVC that represent a Security Context.</p> <p>The SIDs generated by the kernel Security Server are u32 values that are passed via the Linux Security Module hooks to/from the kernel Object Managers.</p>
Type Enforcement	SELinux makes use of a specific style of type enforcement (TE) to enforce Mandatory Access Control . This is where all subjects and objects have a type identifier associated to them that can then be used to enforce rules laid down by Policy .

0.6 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 TERMINOLOGY.....	3
0.6 INDEX.....	5
1. THE SELINUX NOTEBOOK.....	15
1.1 INTRODUCTION.....	15
1.2 NOTEBOOK OVERVIEW.....	15
1.2.1 Notebook Source Overview.....	16
2. SELINUX OVERVIEW.....	17
2.1 INTRODUCTION.....	17
2.1.1 Is SELinux useful.....	17
2.2 CORE SELINUX COMPONENTS.....	19
2.3 MANDATORY ACCESS CONTROL (MAC).....	22

2.4 SELINUX USERS.....	24
2.5 ROLE-BASED ACCESS CONTROL (RBAC).....	24
2.6 TYPE ENFORCEMENT (TE).....	25
2.6.1 Constraints.....	26
2.6.2 Bounds.....	26
2.7 SECURITY CONTEXT.....	27
2.8 SUBJECTS.....	29
2.9 OBJECTS.....	29
2.9.1 Object Classes and Permissions.....	29
2.9.2 Allowing a Process Access to Resources.....	30
2.9.3 Labeling Objects.....	31
2.9.3.1 Labeling Extended Attribute Filesystems.....	32
2.9.3.1.1 Copying and Moving Files.....	32
2.9.3.2 Labeling Subjects.....	33
2.9.4 Object Reuse.....	34
2.10 COMPUTING SECURITY CONTEXTS.....	34
2.10.1 Security Context Computation for Kernel Objects.....	34
2.10.1.1 Process.....	35
2.10.1.2 Files.....	35
2.10.1.3 File Descriptors.....	36
2.10.1.4 Filesystems.....	36
2.10.1.5 Network File System (nfsv4).....	37
2.10.1.6 INET Sockets.....	37
2.10.1.7 IPC.....	37
2.10.1.8 Message Queues.....	37
2.10.1.9 Semaphores.....	38
2.10.1.10 Shared Memory.....	38
2.10.1.11 Keys.....	38
2.10.2 Using libselinux Functions.....	38
2.10.2.1 avc_compute_create and security_compute_create.....	38
2.10.2.2 avc_compute_member and security_compute_member.....	40
2.10.2.3 security_compute_relabel.....	41
2.11 COMPUTING ACCESS DECISIONS.....	42
2.12 DOMAIN AND OBJECT TRANSITIONS.....	43
2.12.1 Domain Transition.....	43
2.12.1.1 Type Enforcement Rules.....	45
2.12.2 Object Transition.....	47
2.13 MULTI-LEVEL SECURITY AND MULTI-CATEGORY SECURITY.....	48
2.13.1 Security Levels.....	49
2.13.1.1 MLS / MCS Range Format.....	50
2.13.1.2 Translating Levels.....	51
2.13.2 Managing Security Levels via Dominance Rules.....	51
2.13.3 MLS Labeled Network and Database Support.....	53
2.13.4 Common Criteria Certification.....	53
2.14 TYPES OF SELINUX POLICY.....	54
2.14.1 Example Policy.....	54
2.14.2 Reference Policy.....	54
2.14.3 Policy Functionality Based on Name or Type.....	55
2.14.4 Custom Policy.....	55
2.14.5 Monolithic Policy.....	55

2.14.6 Loadable Module Policy.....	56
2.14.6.1 Optional Policy.....	56
2.14.7 Conditional Policy.....	56
2.14.8 Binary Policy.....	57
2.14.9 Policy Versions.....	57
2.15 SELINUX PERMISSIVE AND ENFORCING MODES.....	59
2.16 AUDITING SELINUX EVENTS.....	59
2.16.1 AVC Audit Events.....	60
2.16.2 General SELinux Audit Events.....	63
2.17 POLYINSTANTIATION SUPPORT.....	65
2.17.1 Polyinstantiated Objects.....	65
2.17.2 Polyinstantiation support in PAM.....	65
2.17.2.1 namespace.conf Configuration File.....	66
2.17.2.2 Example Configurations.....	67
2.17.3 Polyinstantiation support in X-Windows.....	68
2.17.4 Polyinstantiation support in the Reference Policy.....	68
2.18 PAM LOGIN PROCESS.....	68
2.19 LINUX SECURITY MODULE AND SELINUX.....	70
2.19.1 The LSM Module.....	71
2.19.2 The SELinux Module.....	73
2.19.2.1 Fork System Call Walk-thorough.....	74
2.19.2.2 Process Transition Walk-thorough.....	76
2.19.2.3 SELinux Filesystem.....	81
2.20 LIBSELINUX LIBRARY.....	86
2.21 SELINUX NETWORKING SUPPORT.....	88
2.21.1 SECMARK.....	89
2.21.2 NetLabel - Fallback Peer Labeling.....	91
2.21.3 NetLabel - CIPSO.....	92
2.21.4 Labeled IPSec.....	92
2.21.4.1 Configuration Examples.....	94
2.22 SELINUX VIRTUAL MACHINE SUPPORT.....	96
2.22.1 KVM / QEMU Support.....	96
2.22.2 libvirt Support.....	97
2.22.3 VM Image Labeling.....	97
2.22.3.1 Dynamic Labeling.....	98
2.22.3.2 Shared Image.....	98
2.22.3.3 Static Labeling.....	101
2.22.4 Xen Support.....	103
2.23 SANDBOX SERVICES.....	104
2.24 X-WINDOWS SELINUX SUPPORT.....	106
2.24.1 Infrastructure Overview.....	106
2.24.1.1 Polyinstantiation.....	108
2.24.2 Configuration Information.....	109
2.24.2.1 Enable/Disable the OM from Policy Decisions.....	109
2.24.2.2 Determine OM X-extension Opcode.....	109
2.24.2.3 Configure OM Enforcement Mode.....	109
2.24.2.4 The x_contexts File.....	110
2.24.3 SELinux Extension Functions.....	112
2.25 SE-POSTGRESQL.....	114
2.25.1 sepgsql Overview.....	114

2.25.2	Installing SE-PostgreSQL.....	115
2.25.3	SECURITY LABEL SQL Command.....	116
2.25.4	Additional SQL Functions.....	116
2.25.5	Additional postgresql.conf Entries.....	117
2.25.6	Logging Security Events.....	118
2.25.7	Internal Tables.....	118
2.26	APACHE SELINUX SUPPORT.....	119
2.26.1	mod_selinux Overview.....	119
2.26.2	Bounds Overview.....	120
2.26.2.1	Notebook Examples.....	121
3.	SELINUX CONFIGURATION FILES.....	122
3.1	INTRODUCTION.....	122
3.1.1	Policy Store Migration.....	122
3.1.1.1	The priority Option.....	123
3.1.1.2	Converting policy packages to CIL.....	124
3.2	GLOBAL CONFIGURATION FILES.....	124
3.2.1	/etc/selinux/config File.....	125
3.2.2	/etc/selinux/semanage.conf File.....	126
3.2.3	/etc/selinux/restorecond.conf and restorecond-user.conf Files.....	131
3.2.4	/etc/selinux/newrole_pam.conf.....	131
3.2.5	/etc/sestatus.conf File.....	132
3.2.6	/etc/security/sepermit.conf File.....	132
3.3	POLICY STORE CONFIGURATION FILES.....	133
3.3.1	modules/ Files.....	134
3.3.2	modules/active/base.pp File.....	134
3.3.3	modules/active/base.linked File.....	134
3.3.4	modules/active/commit_num File.....	134
3.3.5	modules/active/file_contexts.template File.....	134
3.3.6	modules/active/file_contexts File.....	138
3.3.7	modules/active/homedir_template File.....	139
3.3.8	modules/active/file_contexts.homedirs File.....	139
3.3.9	modules/active/netfilter_contexts & netfilter.local File.....	140
3.3.10	modules/active/policy.kern File.....	140
3.3.11	modules/active/seusers.final and seusers Files.....	141
3.3.12	modules/active/users_extra, users_extra.local and users.local Files.....	143
3.3.13	modules/active/booleans.local File.....	145
3.3.14	modules/active/file_contexts.local File.....	145
3.3.15	modules/active/interfaces.local File.....	146
3.3.16	modules/active/nodes.local File.....	146
3.3.17	modules/active/ports.local File.....	146
3.3.18	modules/active/preserve_tunables File.....	146
3.3.19	modules/active/disable_dontaudit File.....	146
3.3.20	modules/active/modules Directory Contents.....	146
3.4	POLICY CONFIGURATION FILES.....	147
3.4.1	seusers File.....	148
3.4.2	booleans and booleans.local File.....	148
3.4.3	booleans.subs_dist File.....	149
3.4.4	setrans.conf File.....	150
3.4.5	secolor.conf File.....	152

3.4.6 policy/policy.<ver> File.....	153
3.4.7 contexts/customizable_types File.....	154
3.4.8 contexts/default_contexts File.....	154
3.4.9 contexts/dbus_contexts File.....	156
3.4.10 contexts/default_type File.....	156
3.4.11 contexts/failsafe_context File.....	157
3.4.12 contexts/initrc_context File.....	158
3.4.13 contexts/lxc_contexts File.....	158
3.4.14 contexts/netfilter_contexts File.....	159
3.4.15 contexts/removable_context File.....	159
3.4.16 contexts/securetty_types File.....	160
3.4.17 contexts/sepgsql_contexts File.....	160
3.4.18 contexts/systemd_contexts File.....	161
3.4.19 contexts/userhelper_context File.....	162
3.4.20 contexts/virtual_domain_context File.....	162
3.4.21 contexts/virtual_image_context File.....	163
3.4.22 contexts/x_contexts File.....	163
3.4.23 contexts/files/file_contexts File.....	165
3.4.24 contexts/files/file_contexts.local File.....	165
3.4.25 contexts/files/file_contexts.homedirs File.....	166
3.4.26 contexts/files/file_contexts.subs and file_contexts.subs_dist File.....	166
3.4.27 contexts/files/media File.....	167
3.4.28 contexts/users/[seuser_id] File.....	167
3.4.29 logins/<linuxuser_id> File.....	168
3.4.30 users/local.users File.....	169
4. SELINUX POLICY LANGUAGES.....	170
4.1 INTRODUCTION.....	170
4.1.1 CIL Overview.....	170
4.2 KERNEL POLICY LANGUAGE.....	172
4.2.1 Policy Source Files.....	172
4.2.2 Conditional, Optional and Require Statement Rules.....	175
4.2.3 MLS Statements and Optional MLS Components.....	175
4.2.4 General Statement Information.....	175
4.2.5 Section Contents.....	178
4.3 POLICY CONFIGURATION STATEMENTS.....	179
4.3.1 polycycap.....	179
4.4 DEFAULT OBJECT RULES.....	180
4.4.1 default_user.....	180
4.4.2 default_role.....	181
4.4.3 default_type.....	182
4.4.4 default_range.....	183
4.5 USER STATEMENTS.....	184
4.5.1 user.....	184
4.6 ROLE STATEMENTS.....	186
4.6.1 role.....	186
4.6.2 attribute_role.....	187
4.6.3 roleattribute.....	188
4.6.4 allow.....	189
4.6.5 role_transition.....	189

4.6.6 dominance	191
4.7 TYPE STATEMENTS.....	192
4.7.1 type	192
4.7.2 attribute	194
4.7.3 typeattribute	194
4.7.4 typealias	195
4.7.5 permissive	196
4.7.6 type_transition	197
4.7.7 type_change	200
4.7.8 type_member	201
4.8 BOUNDS RULES.....	201
4.8.1 typebounds	202
4.9 ACCESS VECTOR RULES.....	203
4.9.1 allow	204
4.9.2 dontaudit	205
4.9.3 auditallow	205
4.9.4 neverallow	205
4.10 OBJECT CLASS AND PERMISSION STATEMENTS.....	206
4.10.1 class	206
4.10.2 Associating Permissions to a Class.....	207
4.10.3 common	207
4.10.4 class	207
4.11 CONDITIONAL POLICY STATEMENTS.....	209
4.11.1 bool	209
4.11.2 if	210
4.12 CONSTRAINT STATEMENTS.....	212
4.12.1 constrain	212
4.12.2 validatetrans	215
4.12.3 mlsconstrain	216
4.12.4 mlsvalidatetrans	217
4.13 MLS STATEMENTS.....	219
4.13.1 sensitivity	220
4.13.2 dominance	221
4.13.3 category	222
4.13.4 level	222
4.13.5 range_transition	223
4.13.5.1 MLS range Definition.....	224
4.13.6 mlsconstrain	225
4.13.7 mlsvalidatetrans	225
4.14 SECURITY ID (SID) STATEMENT.....	225
4.14.1 sid	225
4.14.2 sid_context	226
4.15 FILE SYSTEM LABELING STATEMENTS.....	227
4.15.1 fs_use_xattr	227
4.15.2 fs_use_task	228
4.15.3 fs_use_trans	228
4.15.4 genfscon	229
4.16 NETWORK LABELING STATEMENTS.....	230
4.16.1 IP Address Formats.....	231
4.16.1.1 IPv4 Address Format.....	231

4.16.1.2 IPv6 Address Formats.....	231
4.16.2 netifcon	231
4.16.3 nodecon	232
4.16.4 portcon	234
4.17 MODULAR POLICY SUPPORT STATEMENTS.....	235
4.17.1 module	235
4.17.2 require	235
4.17.3 optional	237
4.18 XEN STATEMENTS.....	238
4.18.1 iomemcon	238
4.18.2 ioportcon	239
4.18.3 pcidevicecon	240
4.18.4 irqcon	240
5. THE REFERENCE POLICY.....	242
5.1 INTRODUCTION.....	242
5.2 REFERENCE POLICY OVERVIEW.....	242
5.2.1 Distributing Policies.....	244
5.2.2 Policy Functionality.....	245
5.2.3 Reference Policy Module Files.....	245
5.2.4 Reference Policy Documentation.....	248
5.3 REFERENCE POLICY SOURCE.....	249
5.3.1 Source Layout.....	249
5.3.2 Reference Policy Files and Directories.....	249
5.3.3 Source Configuration Files.....	252
5.3.3.1 Reference Policy Build Options - build.conf.....	252
5.3.3.2 Reference Policy Build Options - policy/modules.conf.....	253
5.3.3.2.1 Building the modules.conf File.....	256
5.3.4 Source Installation and Build Make Options.....	256
5.3.5 Booleans, Global Booleans and Tunable Booleans.....	258
5.3.6 Modular Policy Build Structure.....	259
5.3.7 Creating Additional Layers.....	261
5.4 INSTALLING AND BUILDING THE REFERENCE POLICY SOURCE.....	261
5.4.1 Building Standard Reference Policy.....	261
5.4.2 Building the Fedora Policy.....	263
5.5 REFERENCE POLICY HEADERS.....	266
5.5.1 Building and Installing the Header Files.....	266
5.5.2 Using the Reference Policy Headers.....	267
5.5.3 Using Fedora Supplied Headers.....	268
5.6 MIGRATING COMPILED MODULES TO CIL.....	268
5.7 REFERENCE POLICY SUPPORT MACROS.....	268
5.7.1 Loadable Policy Macros.....	270
5.7.1.1 policy_module Macro.....	270
5.7.1.2 gen_require Macro.....	271
5.7.1.3 optional_policy Macro.....	272
5.7.1.4 gen_tunable Macro.....	273
5.7.1.5 tunable_policy Macro.....	274
5.7.1.6 interface Macro.....	275
5.7.1.7 template Macro.....	277
5.7.2 Miscellaneous Macros.....	279

5.7.2.1 gen_context Macro.....	279
5.7.2.2 gen_user Macro.....	281
5.7.2.3 gen_bool Macro.....	282
5.7.3 MLS and MCS Macros.....	283
5.7.3.1 gen_cats Macro.....	283
5.7.3.2 gen_sens Macro.....	284
5.7.3.3 gen_levels Macro.....	285
5.7.3.4 System High/Low Parameters.....	286
5.7.4 ifdef / ifndef Parameters.....	286
5.7.4.1 hide_broken_symptoms.....	286
5.7.4.2 enable_mls and enable_mcs.....	287
5.7.4.3 enable_ubac.....	287
5.7.4.4 direct_sysadm_daemon.....	288
5.8 MODULE EXPANSION PROCESS.....	288
6. IMPLEMENTING SELINUX-AWARE APPLICATIONS.....	290
6.1 INTRODUCTION.....	290
6.1.1 Implementing SELinux-aware Applications.....	290
6.1.2 Implementing Object Managers.....	292
6.1.3 Reference Policy Changes.....	293
6.1.4 Adding New Object Classes and Permissions.....	294
7. SECURITY ENHANCEMENTS FOR ANDROID.....	296
7.1 INTRODUCTION.....	296
7.1.1 Terminology.....	296
7.1.2 Useful Links.....	297
7.1.3 Document Sections.....	297
7.2 SE FOR ANDROID PROJECT UPDATES.....	298
7.3 KERNEL LSM / SELINUX SUPPORT.....	301
7.4 SE FOR ANDROID CLASSES & PERMISSIONS.....	302
7.5 SELINUX COMMANDS.....	304
7.6 SELINUX PUBLIC METHODS.....	305
7.7 ANDROID INIT LANGUAGE SELINUX EXTENSIONS.....	307
7.8 DEVICE POLICY FILE LOCATIONS.....	308
7.9 BUILDING THE POLICY.....	309
7.9.1 SELinux MAC Policy Files.....	309
7.9.1.1 Policy Build Files.....	309
7.9.1.2 Policy Configuration Files.....	310
7.9.2 Install-time MMAC Policy File.....	312
7.9.3 Device Specific Policy.....	312
7.9.3.1 Managing Device Policy File.....	313
7.9.4 Build Tools.....	315
7.9.5 Miscellaneous Information.....	316
7.9.5.1 SELinux Policy Versions.....	316
7.9.5.2 SELinux Policy Booleans.....	316
7.9.5.3 Setting Permissive / Enforcing Mode.....	316
7.9.5.4 Checking File Labels.....	317
7.10 UPDATING POLICY FILES.....	317
7.10.1.1 Local Policy Update.....	317
7.11 LOGGING AND AUDITING.....	318

7.12 POLICY FILE CONFIGURATION DETAIL.....	319
7.12.1 SELinux MAC Configuration Files.....	319
7.12.1.1 seapp_contexts File.....	319
7.12.1.1.1 Default Entries.....	319
7.12.1.1.2 Entry Definitions.....	320
7.12.1.1.3 Computing a Context.....	321
7.12.1.2 property_contexts File.....	326
7.12.1.3 service_contexts File.....	327
7.12.2 Install-time MMAC Configuration File.....	328
7.12.2.1 Policy Rules.....	329
7.12.3 EOps MMAC Configuration File.....	330
7.12.4 Intent Firewall MMAC Configuration File.....	331
7.13 POLICY BUILD TOOLS.....	332
7.13.1 checkfc.....	332
7.13.2 checkseapp.....	333
7.13.3 insertkeys.py.....	333
7.13.3.1 keys.conf File.....	334
7.13.4 Build Bundle Tools.....	335
7.13.4.1 buildsebundle.....	335
7.13.4.1.1 Using an Intent Example.....	336
7.13.4.2 buildeopbundle.....	337
7.13.4.2.1 Eops Example.....	338
7.13.4.3 buildifwbundle.....	339
7.13.4.3.1 IFW Example.....	339
7.13.5 post_process_mac_perms.....	340
7.13.6 sepolicy_check.....	341
7.13.7 sepolicy-analyze.....	341
7.13.7.1 Type Equivalence.....	341
7.13.7.2 Type Difference.....	342
7.13.7.3 Duplicate Allow Rules.....	342
7.13.8 setool.....	343
7.14 SELINUX-NETWORK.SH CONFIGURATION.....	344
7.15 UID TO USERNAME UTILITY.....	345
8. APPENDIX A - OBJECT CLASSES AND PERMISSIONS.....	347
8.1 INTRODUCTION.....	347
8.2 DEFINING OBJECT CLASSES AND PERMISSIONS.....	347
8.3 COMMON PERMISSIONS.....	348
8.3.1 Common File Permissions.....	348
8.3.2 Common Socket Permissions.....	348
8.3.3 Common IPC Permissions.....	349
8.3.4 Common Database Permissions.....	349
8.3.5 Common X Device Permissions.....	350
8.4 FILE OBJECT CLASSES.....	350
8.5 NETWORK OBJECT CLASSES.....	353
8.5.1 IPSec Network Object Classes.....	355
8.5.2 Netlink Object Classes.....	356
8.5.3 Miscellaneous Network Object Classes.....	358
8.6 IPC OBJECT CLASSES.....	359
8.7 PROCESS OBJECT CLASS.....	359

<u>8.8 SECURITY OBJECT CLASS.....</u>	<u>360</u>
<u>8.9 SYSTEM OPERATION OBJECT CLASS.....</u>	<u>361</u>
<u>8.10 KERNEL SERVICE OBJECT CLASS.....</u>	<u>361</u>
<u>8.11 CAPABILITY OBJECT CLASSES.....</u>	<u>362</u>
<u>8.12 X WINDOWS OBJECT CLASSES.....</u>	<u>363</u>
<u>8.13 DATABASE OBJECT CLASSES.....</u>	<u>368</u>
<u>8.14 MISCELLANEOUS OBJECT CLASSES.....</u>	<u>370</u>
<u>9. APPENDIX B - LIBSELINUX LIBRARY FUNCTIONS.....</u>	<u>373</u>
<u>10. APPENDIX C - SELINUX COMMANDS.....</u>	<u>389</u>
<u>11. APPENDIX D - DOCUMENT REFERENCES.....</u>	<u>390</u>
<u>12. APPENDIX E - POLICY VALIDATION EXAMPLE.....</u>	<u>391</u>
<u>13. APPENDIX F - GNU FREE DOCUMENTATION LICENSE.....</u>	<u>393</u>

1. The SELinux Notebook

1.1 Introduction

This Notebook should help with explaining:

- a) SELinux and its purpose in life.
- b) The LSM / SELinux architecture, its supporting services and how they are implemented within GNU / Linux.
- c) SELinux Networking, Virtual Machine, X-Windows, PostgreSQL and Apache/SELinux-Plus SELinux-aware capabilities.
- d) The core SELinux kernel policy language and how basic policy modules can be constructed for instructional purposes.
- e) An introduction to the new Common Intermediate Language (CIL) implementation.
- f) The core SELinux policy management tools with examples of usage.
- g) The Reference Policy architecture, its supporting services and how it is implemented.
- h) The integration of SELinux within Android - SE for Android.

Note that this Notebook will not explain how the SELinux implementations are managed for each GNU / Linux distribution as they have their own supporting documentation.

While the majority of this Notebook is based on Fedora 20, all additional developments as seen on the SELinux mail list (selinux@tycho.nsa.gov) up to September '14 have been added.

1.2 Notebook Overview

This volume has the following major sections:

SELinux Overview - Gives a description of SELinux and its major components to provide Mandatory Access Control services for GNU / Linux. Hopefully it will show how all the SELinux components link together and how SELinux-aware applications / object manager have been implemented (such as Networking, X-Windows, PostgreSQL and virtual machines).

SELinux Configuration Files - Describes all known SELinux configuration files with samples. Also lists any specific SELinux commands or `libselinux` APIs used by them.

SELinux Policy Language - Gives a brief description of each policy language statement, with supporting examples taken from the Reference Policy source. Also an introduction to the new CIL language (Common Intermediate Language).

The Reference Policy - Describes the Reference Policy and its supporting macros.

SE for Android - An overview of the SELinux services used to support Android.

Object Classes and Permissions - Describes the SELinux object classes and permissions.

libselinux Functions - Describes the `libselinux` library functions.

1.2.1 Notebook Source Overview

To demonstrate some of the SELinux capabilities a supporting Notebook source tarball is available (`notebook-source-4.0.tar.gz`). The tarball contains directories and READMEs covering the following:

Building a Basic Policy - 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. These can be extended with additional modules in kernel policy language and CIL.

Example `libselinux` applications - This contains over 100 samples that use the `libselinux 2.2.1-6` functions. To save typing long context strings it makes use of a configuration file. There are also some supporting policy modules for the F-20 targeted policy to show how the functions work.

Example Android emulator device - This replaces the kernel policy language version with a CIL policy using namespaces. This is built using Android 4.4 AOSP master and will show processes as `u:r:kernel.process:s0`, `u:r:untrusted_app.process:s0:c512,c768`. and files as `u:r:bluetooth.data_file:s0`, `u:r:app.data_file:s0:c512,c768` etc..

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.

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

Note: When SELinux is installed, there are three well defined directory locations referenced. Two of these will change with the old and new locations as follows:

Description	Old Location	New Location
The SELinux filesystem that interfaces with the kernel based security server. The new location has been available since Fedora 17.	/selinux	/sys/fs/selinux
The SELinux configuration directory that holds the sub-system configuration files and policies.	/etc/selinux	No change
The SELinux policy store that holds policy modules and configuration details (see https://github.com/SELinuxProject/selinux/wiki/Policy-Store-Migration and Policy Store Migration).	/etc/selinux/ <SELINUXTYPE>/module	/var/lib/selinux/ <SELINUXTYPE>/module

2.1.1 Is SELinux useful

There are many views on the usefulness of SELinux on Linux based systems, this section gives a brief view of what SELinux is good at and what it is not (because its not designed to do it).

SELinux is not just for military or high security systems where Multi-Level Security (MLS) is required (for functionality such as 'no read up' and 'no write down'), as using the 'type enforcement' (TE) functionality applications can be confined (or contained) within domains and limited to the minimum privileges required to do their job, so in a 'nutshell':

1. If SELinux is enabled, the policy defines what access to resources and operations on them (e.g. read, write) are allowed (i.e. SELinux stops all access

unless allowed by policy). This is why SELinux is called a 'mandatory access control' (MAC) system.

2. The policy design, implementation and testing against a defined security policy or requirements is important, otherwise there could be 'a false sense of security'.
3. SELinux can confine an application within its own 'domain' and allow it to have the minimum privileges required to do its job. Should the application require access to networks or other applications (or their data), then (as part of the security policy design), this access would need to be granted (so at least it is known what interactions are allowed and what are not - a good security goal).
4. Should an application 'do something' it is not allowed by policy (intentional or otherwise), then SELinux would stop these actions.
5. Should an application 'do something' it is allowed by policy, then SELinux may contain any damage that maybe done intentional or otherwise. For example if an application is allowed to delete all of its data files or database entries and the bug, virus or malicious user gains these privileges then it would be able to do the same, however the good news is that if the policy 'confined' the application and data, all your other data should still be there.
6. User login sessions can be confined to their own domains. This allows clients they run to be given only the privileges they need (e.g. admin users, sales staff users, HR staff users etc.). This again will confine/limit any damage or leakage of data.
7. Some applications (X-Windows for example) are difficult to confine as they are generally designed to have total access to all resources. SELinux can generally overcome these issues by providing sandboxing services.
8. SELinux will not stop memory leaks or buffer over-runs (because its not designed to do this), however it may contain the damage that may be done.
9. SELinux will not stop all viruses/malware getting into the system (as there are many ways they could be introduced (including by legitimate users), however it should limit the damage or leaks they cause.
10. SELinux will not stop kernel vulnerabilities, however it may limit their effects.
11. It is easy to add new rules to an SELinux policy using tools such as **audit2allow** (1) if a user has the relevant permissions, however be aware that this may start opening holes, so check what rules are really required.
12. Finally, SELinux cannot stop anything allowed by the security policy, so good design is important.

The following maybe useful in providing a practical view of SELinux:

1. A discussion regarding Apache servers and SELinux that may look negative at first but highlights the containment points above. This is the initial study: <http://blog.ptsecurity.com/2012/08/selinux-in-practice-dvwa-test.html>, and this is a response to the study: <http://danwalsh.livejournal.com/56760.html>.

However with careful design and known security goals the SELinux [Apache / SELinux Plus](#) services could be used to build a more secure web service (also see http://code.google.com/p/sepgsql/wiki/Apache_SELinux_plus).

2. SELinux services have been added to Android, producing SE for Android. The presentation "The Case for Security Enhanced (SE)Android" [20] gives use-cases and types of Android exploits that SELinux could have overcome. The presentation and others are available at:

<http://seandroid.bitbucket.org/PapersandPresentation.html#3>

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.

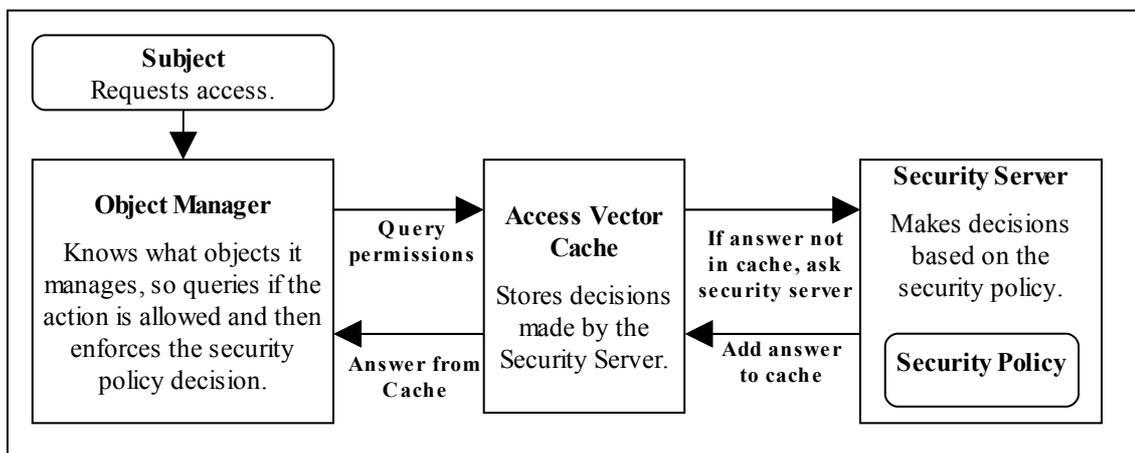


Figure 2.1: High Level Core SELinux Components - *Decisions by the Security Server are cached in the AVC to enhance performance of future requests. Note that it is the kernel and userspace Object Managers that enforce the policy.*

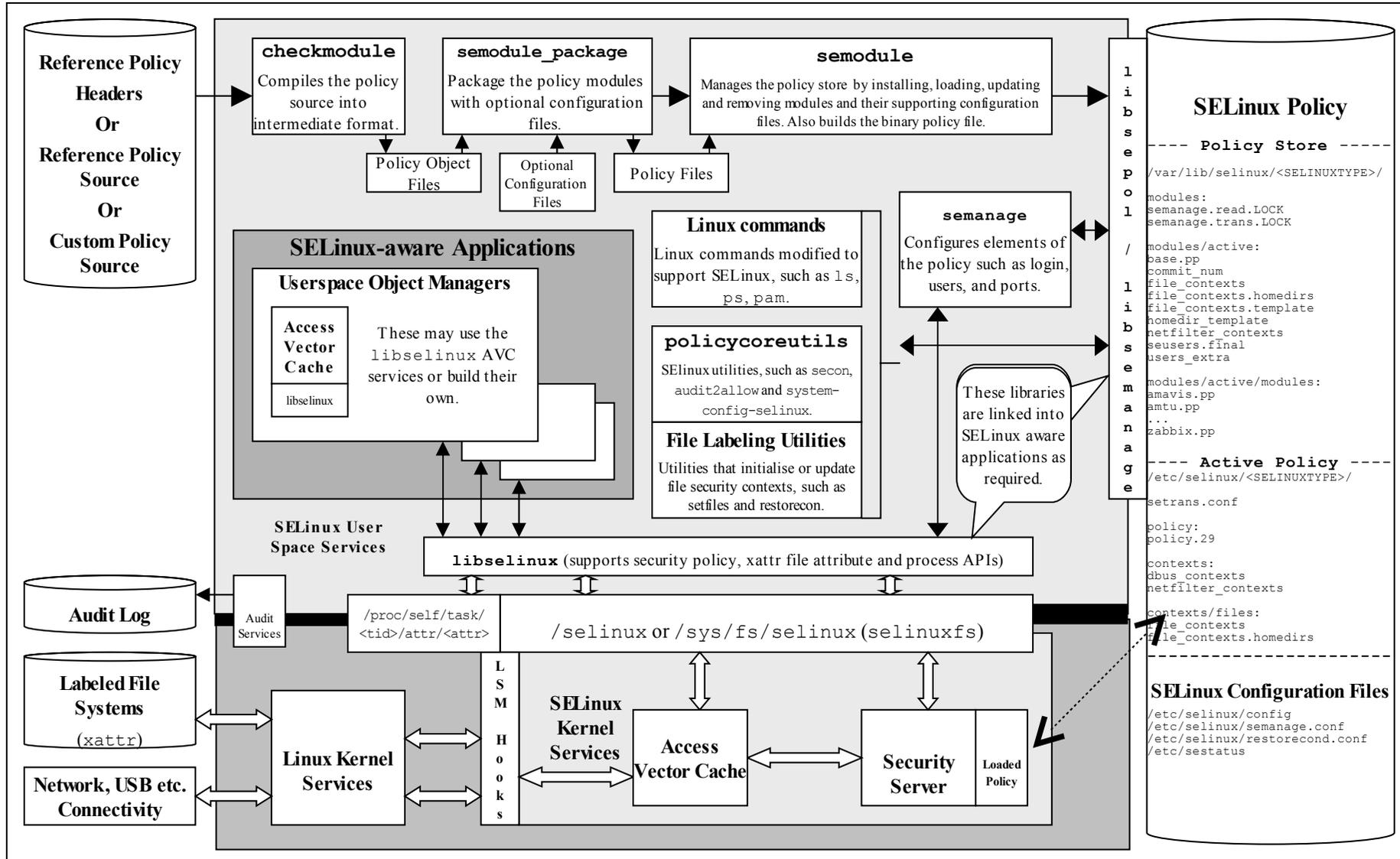


Figure 2.2: High Level SELinux Architecture - Showing the major supporting services

[Figure 2.2](#) shows a more complex diagram of kernel and userspace with a number of supporting services that are used to manage the SELinux environment. This diagram will be referenced a number of times to explain areas of SELinux, therefore starting from the bottom:

- a) In the current implementation of SELinux the security server is embedded in the kernel with the policy being loaded from userspace via a series of functions contained in the `libselinux` library (see [SELinux Userspace Libraries](#) for details).

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 the security servers response to the kernel based object managers thus speeding up access decisions should the same request be asked in future.

userspace - These object managers are provided with the application or service that requires support for MAC and are known as 'SELinux-aware' 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 (see [Implementing SELinux-aware Applications](#) for details).

- b) The SELinux security 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 has the name of the policy to be loaded (via the `SELINUXTYPE` entry) and the initial enforcement mode¹ of the policy at load time (via the `SELINUX` entry). The `/etc/selinux/<SELINUXTYPE>` directories contain policies that can be activated along with their configuration files (e.g. 'SELINUXTYPE=targeted' will have its policy and associated configuration files located at `/etc/selinux/targeted`). All known configuration files 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 source 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 where generally each supports a particular application or service. These modules are

¹ When SELinux is enabled, the policy can be running in 'permissive mode' (`SELINUX=permissive`), where all accesses are allowed. The policy can also be run in 'enforcing mode' (`SELINUX=enforcing`), where any access that is not defined in the policy is denied and an entry placed in the audit log. SELinux can also be disabled (at boot time only) by setting `SELINUX=disabled`. There is also support for the [permissive](#) statement that allows a domain to run in permissive mode while the others are still confined (instead of the all or nothing set by `SELINUX=`).

compiled, linked, and held in a 'policy store' where they can be built into a binary format that is then loaded into the security server (in the diagram the binary policy is located at `/etc/selinux/targeted/policy/policy.29`). 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 three 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 large policy developments such as the [Reference Policy](#), although the smaller SE for Android policy is written this way with some m4 macro support.
 - ii) using the [Reference Policy](#) that has high level macros to define policy rules. This is the standard way policies are now built for SELinux distributions such as Red Hat and Debian and is discussed in the [Reference Policy](#) section. Note that SE for Android also uses high level macros to define policy rules but the overall policy is much less complex.
 - iii) using CIL (Common Intermediate Language). An overview can be found at <https://github.com/SELinuxProject/cil/wiki> and the [CIL Overview](#) section.
- e) To be able to compile and link the policy source then load it into the security server requires a number of tools (top of [Figure 2.2](#)).
- f) To enable system administrators to manage policy, the SELinux environment and label file systems, tools and modified GNU / Linux commands are used. These are mentioned throughout the Notebook as needed and summarised in [Appendix C - 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 [Auditing SELinux Events](#) section describes the format of these security events.
- h) SELinux supports network services that are described in the [SELinux Networking Support](#) section.

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.
- 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 enforced.

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. The steps in the decision making chain for DAC and MAC are shown in [Figure 2.3](#).

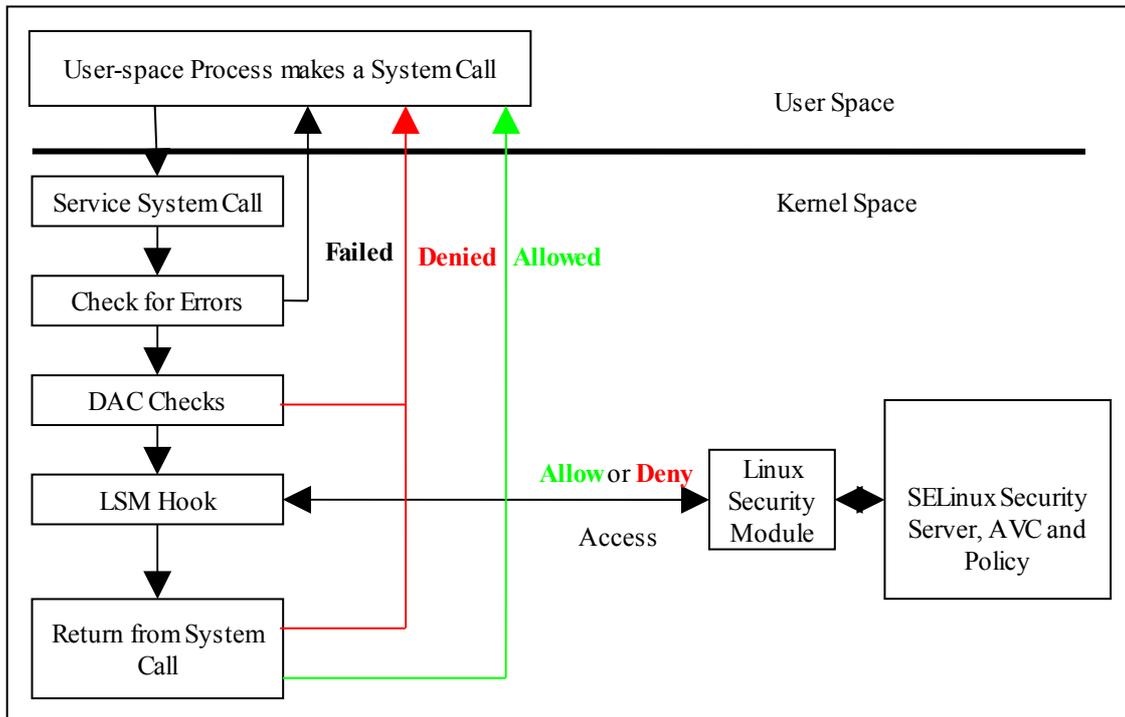


Figure 2.3: Processing a System Call - The DAC checks are carried out first, if they pass then the Security Server is consulted for a decision.

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 along with Role Based Access Control. The [Type Enforcement](#) and [Role Based Access Control](#) sections covers these 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 restricted information is separated from classified information to maintain 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 called Multi-Category Security (MCS).

The MLS / MCS services are now more generally used to maintain application separation, for example SELinux enabled:

- virtual machines use MCS categories to allow each VM to run within its own domain to isolate VMs from each other (see the [SELinux Virtual Machine Support](#) section).
- Android devices use dynamically generated MCS categories so that an app running on behalf of one user cannot read or write files created by the same app running on behalf of another user (see the [Security Enhancements for Android - Computing a Process Context](#) section).

2.4 SELinux Users

Users in GNU / Linux are generally associated to human users (such as Alice and Bob) or operator/system functions (such as admin), while this can be implemented in SELinux, SELinux user names are generally groups or classes of user. For example all the standard system users could be assigned an SELinux user name of `user_u` and administration staff under `staff_u`.

There is one special SELinux user defined that must never be associated to a GNU / Linux user as it a special identity for system processes and objects, this user is `system_u`.

The SELinux user name is the first component of a 'security context' and by convention SELinux user names end in `'_u'`, however this is not enforced by any SELinux service (i.e. it is only to identify the user component), although CIL with namespaces does make identification of an SELinux user easier for example a 'user' could be declared as `unconfined.user`.

It is possible to add constraints and bounds on SELinux users as discussed in the [Type Enforcement](#) section.

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](#).

The SELinux role name is the second component of a 'security context' and by convention SELinux roles end in `'_r'`, however this is not enforced by any SELinux service (i.e. it is only used to identify the role component), although CIL with namespaces does make identification of a role easier for example a 'role' could be declared as `unconfined.role`.

It is possible to add constraints and bounds on roles as discussed in the [Type Enforcement](#) section.

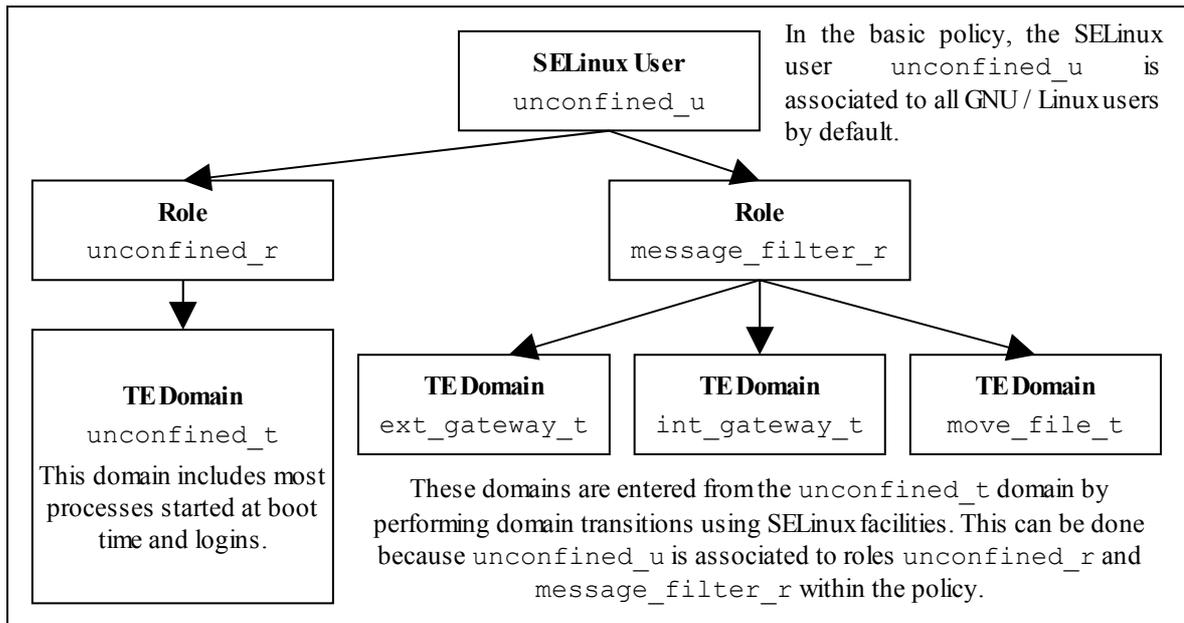


Figure 2.4: Role Based Access Control - Showing how SELinux controls access via user, role and domain type association.

2.6 Type Enforcement (TE)

SELinux makes use of a specific style of type enforcement² (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 by 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 [SELinux language statements and rules](#) used to build a policy that will, when loaded into the security server, enforce policy via the object managers.

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 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 (although CIL policies with namespaces do help in that a domain process 'type' could be declared as `msg_filter.ext_gateway.process` with object types being any others (such as `msg_filter.ext_gateway.exec`).

Basically if the type identifier is used to reference a subject it is referring to a Linux process or collection of processes (a domain or 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 `basic-selinux-policy` directory of the source tarball, all the processes on the system run in the `unconfined_t` domain (or for the CIL version in the `unconfined.process` domain), therefore every

² There are various 'type enforcement' technologies.

process is 'of type `unconfined_t`' (that means it can do whatever it likes within the limits of the standard Linux DAC policy as all access is allowed by SELinux).

It is only when additional policy statements are added to the simple policy 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 Red Hat Fedora where the majority of user space processes run under the `unconfined_t` domain (although don't think the simple policies implemented in source tarball are equivalent to the Reference Policy, they are not - so do not use them as live implementations).

The SELinux type is the third component of a 'security context' and by convention SELinux types end in '`_t`', however this is not enforced by any SELinux service (i.e. it is only used to identify the type component), although as explained above CIL with namespaces does make identification of types easier.

2.6.1 Constraints

It is possible to add constraints on users, roles, types and MLS ranges, for example within a TE environment, the way that subjects are allowed to access an object is via a TE [allow rule](#), for example:

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

This 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). Note that the constraint is based on an object class (`process` in this case) and one or more of its permissions.

The kernel policy language constraints are defined in the [Constraint Statements](#) section).

2.6.2 Bounds

It is possible to add bounds to users, roles and types, however currently only types are enforced by the kernel using the `typebounds` rule as described in the [Bounds Overview](#) section (although user and role bounds may be declared using CIL, however they are validated at compile time).

2.7 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.

Within SELinux, a security context is represented as variable-length strings that define the SELinux user³, their role, a type identifier and an optional MCS / MLS security range or level as follows:

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

Where:

user	The SELinux user identity. This can be associated to one or more roles that the SELinux user is allowed to use.
role	The SELinux role. This can be associated to one or more types the SELinux user is allowed to access.
type	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.
range	This field can also be know as a <code>level</code> and is only present if the policy supports MCS or MLS. The entry can consist of: <ul style="list-style-type: none">• A single security <code>level</code> that contains a sensitivity level and zero or more categories (e.g. <code>s0</code>, <code>s1:c0</code>, <code>s7:c10.c15</code>).• A <code>range</code> that consists of two security levels (a low and high) separated by a hyphen (e.g. <code>s0 - s15:c0.c1023</code>). These components are discussed in the Security Levels 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 the components as follows:
 - 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. It is possible to add constraints on users within policy based on their object class (an example of this is the Reference Policy UBAC (User Based Access Control) option).
 - b) the `role` is generally set to a special SELinux internal role of `object_r`, although policy version 26 with kernel 2.6.39 and above do support role transitions on any object class. It is then possible to add constraints on the role within policy based on their object class.

³ 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.

The [Computing Security Contexts](#) section describes how SELinux computes the security context components based on a source context, target context and object class.

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

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
unconfined_u:unconfined_r:unconfined_t      2539 pts/0 bash
unconfined_u:message_filter_r:ext_gateway_t  3134 pts/0 secure_server
unconfined_u:message_filter_r:int_gateway_t  3138 pts/0 secure_server
unconfined_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    /usr/message_queue/in_queue
system_u:object_r:out_queue_t   /usr/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):

/usr/message_queue/in_queue:
unconfined_u:object_r:in_file_t      Message-1
unconfined_u:object_r:in_file_t      Message-2

/usr/message_queue/out_queue:
unconfined_u:object_r:out_file_t      Message-10
unconfined_u:object_r:out_file_t      Message-11

# Note that they are instantiated with unconfined_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.
```

2.8 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 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 class⁴ 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**(8) and **semodule**(8) commands could be grouped under `semanage_t`).

Untrusted - Everything else.

2.9 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.

2.9.1 Object Classes and Permissions

Each object consists of a class identifier that defines its purpose (e.g. file, socket) along with a set of permissions⁵ 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](#).

⁴ The object class and its associated permissions are explained in the [Process Object Class](#) section.

⁵ Also known in SELinux as Access Vectors (AV).

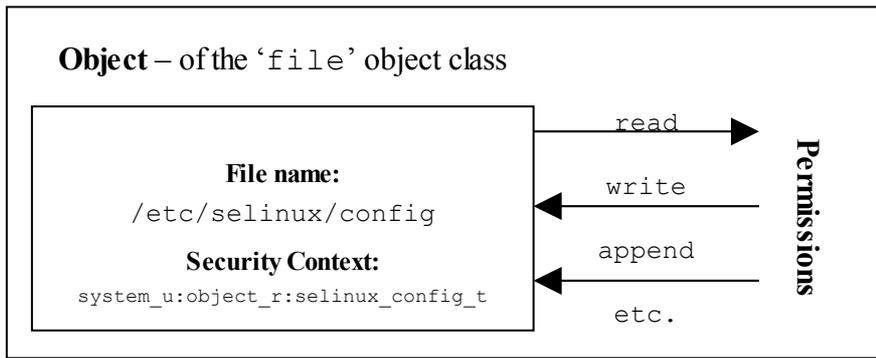


Figure 2.5: Object Class = 'file' and permissions - the policy rules would define those permissions allowed for each process that needs access to the `/etc/selinux/config` file.

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 userspace object classes for userspace object managers (for services such as X-Windows or dbus). The number of object classes and their permissions can vary depending on the features configured in the GNU / Linux release. All the known object classes and permissions are described in [Appendix A - Object Classes and Permissions](#).

2.9.2 Allowing a Process Access to Resources

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.

An SELinux policy contains many rules and statements, the majority of which are allow rules that (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.

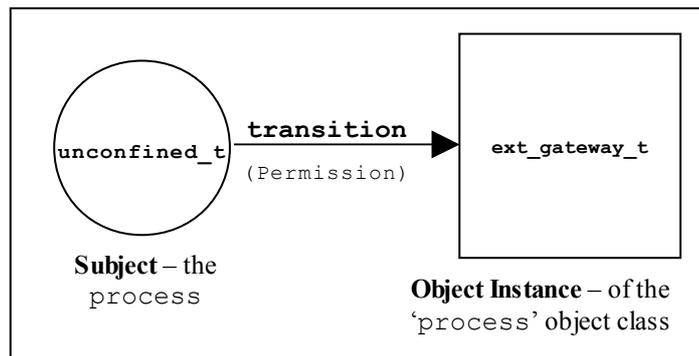


Figure 2.6: The allow rule - Showing that the subject (the processes running in the unconfined_t domain) has been given the transition permission on the ext_gateway_t 'process' object.

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.

2.9.3 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 or object.
2. The policy type, role and range 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. An object manager (OM) can enforce a default label that can either be built into the OM or obtained via a configuration file (such as those used by [X-Windows](#)).
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).

The [Computing Security Contexts](#) section gives detail on how some of the kernel based objects are computed.

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 file systems that use extended attributes (such as `ext3` and `ext4`) is discussed in the [Labeling Extended Attribute Filesystems](#) section.

2.9.3.1 Labeling Extended Attribute Filesystems

The labeling of file systems that implement extended attributes⁶ is supported by SELinux using:

1. The `fs_use_xattr` statement within the policy to identify what file systems use extended attributes. This statement is used to inform the security server how to label the filesystem.
2. A 'file contexts' file that defines what the initial contexts should be for each file and directory within the filesystem. The format of this file is described in the [modules/active/file_contexts.template file](#)⁷ 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). 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-- rch rch unconfined_u:object_r:user_home:s0 myfile
```

```
getfattr -n security.selinux myfile
# file_name: myfile
security.selinux="unconfined_u:object_r:user_home:s0

# Where -n security.selinux is the name of the extended
# attribute and 'myfile' is a file name. The security context
# (or label) held for the file is displayed.
```

2.9.3.1.1 Copying and Moving Files

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.
- 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

⁶ These file systems store the security context in an attribute associated with the file.

⁷ 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.

it is moved or copied (provided it is a valid context and specified in the [file_contexts](#) file). Note that there is also the `install(1)` command that supports a `-Z` option to specify the target context.

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 unconfined_u:object_r:unconfined_t    copied-file
-rw-r--r--  root root unconfined_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

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

# The target directory (/usr/message_queue/in_queue) is labeled "in_queue_t".
# The results of "ls -Z" on the target directory are:
#
-rw-r--r--  root root unconfined_u:object_r:in_queue_t      copied-file
-rw-r--r--  root root unconfined_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 unconfined_u:object_r:in_file_t      copied-file
-rw-r--r--  root root unconfined_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.

2.9.3.2 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 policy language supports a number of statements to assign components to security contexts 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

2.9.4 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).

2.10 Computing Security Contexts

SELinux uses a number of policy language statements and `libselinux` functions to compute a security context via the kernel security server.

When security contexts are computed, the different kernel, userspace tools and policy versions can influence the outcome. This is because patches have been applied over the years that give greater flexibility in computing contexts. For example a 2.6.39 kernel with SELinux userspace services supporting policy version 26 can influence the computed role.

The security context is computed for an object using the following components: a source context, a target context and an object class.

The `libselinux` userspace functions used to compute a security context are:

```
avc_compute_create(3) and security_compute_create(3)
avc_compute_member(3) and security_compute_member(3)
security_compute_relabel(3)
```

Note that these `libselinux` functions actually call the kernel equivalent functions in the security server (see kernel source `security/selinux/ss/services.c`: `security_compute_sid`, `security_member_sid` and `security_change_sid`) that actually compute the security context.

The kernel policy language statements that influence a computed security context are:

```
type_transition, role_transition, range_transition,
type_member and type_change, default_user, default_role,
default_type and default_range statements (their corresponding CIL
statements exclude the underscore).
```

The sections that follow give an overview of how security contexts are computed for some kernel classes and also when using the userspace `libselinux` functions.

2.10.1 Security Context Computation for Kernel Objects

Using a combination of the email thread: <http://www.spinics.net/lists/selinux/msg10746.html> and kernel 3.14 source, this is how contexts are computed by the security server for various kernel objects (also see

the [Linux Security Module and SELinux](#) section and "[Implementing SELinux as a Linux Security Module](#)" [1]).

2.10.1.1 Process

The initial task starts with the kernel security context, but the "init" process will typically transition into its own unique context (e.g. `init_t`) when the `init` binary is executed after the policy has been loaded. Some `init` programs re-exec themselves after loading policy, while in other cases the initial policy load is performed by the `initrd/initramfs` script prior to mounting the real root and executing the real `init` program.

Processes inherit their security context as follows:

1. On fork a process inherits the security context of its creator/parent.
2. On `exec`, a process may transition to another security context based on policy statements: `type_transition`, `range_transition`, `role_transition` (policy version 26), `default_user`, `default_role`, `default_range` (policy versions 27) and `default_type` (policy version 28) or if a security-aware process, by calling `setexeccon` (3) if permitted by policy prior to invoking `exec`.
3. At any time, a security-aware process may invoke `setcon` (3) to switch its security context (if permitted by policy) although this practice is generally discouraged - `exec`-based transitions are preferred.

2.10.1.2 Files

The default behavior for labeling files (actually inodes that consist of the following classes: files, symbolic links, directories, socket files, fifo's and block/character) upon creation for any filesystem type that supports labeling is as follows:

1. The user component is inherited from the creating process (policy version 27 allows a `default_user` of source or target to be defined for each object class).
2. The role component generally defaults to the `object_r` role (policy version 26 allows a `role_transition` and version 27 allows a `default_role` of source or target to be defined for each object class).
3. The type component defaults to the type of the parent directory if no matching `type_transition` rule was specified in the policy (policy version 25 allows a filename `type_transition` rule and version 28 allows a `default_type` of source or target to be defined for each object class).
4. The range/level component defaults to the low/current level of the creating process if no matching `range_transition` rule was specified in the policy (policy version 27 allows a `default_range` of source or target with the selected range being low, high or low-high to be defined for each object class).

Security-aware applications can override this default behavior by calling `setfscreatecon` (3) prior to creating the file, if permitted by policy.

For existing files the label is determined from the `xattr` value associated with the file. If there is no `xattr` value set on the file, then the file is treated as being labeled with the default file security context for the filesystem. By default, this is the "file" initial SID, which is mapped to a context by the policy. This default may be overridden via the `defcontext=mount` option on a per-mount basis as described in `mount(8)`.

2.10.1.3 File Descriptors

Inherits the label of its creator/parent.

2.10.1.4 Filesystems

Filesystems are labeled using the appropriate `fs_use` kernel policy language statement as they are mounted, they are based on the filesystem type name (e.g. `ext4`) and their behaviour (e.g. `xattr`). For example if the policy specifies the following:

```
fs_use_task pipefs system_u:object_r:fs_t:s0
```

then as the `pipefs` filesystem is being mounted, the SELinux LSM security hook `selinux_set_mnt_opts` will call `security_fs_use` that will:

- a) Look for the filesystem name within the policy (`pipefs`)
- b) If present, obtain its behaviour (`fs_use_task`)
- c) Then obtain the allocated security context (`system_u:object_r:fs_t:s0`)

Should the behaviour be defined as `fs_use_task`, then the filesystem will be labeled as follows:

1. The user component is inherited from the creating process (policy version 27 allows a `default_user` of source or target to be defined).
2. The role component generally defaults to the `object_r` role (policy version 26 allows a `role_transition` and version 27 allows a `default_role` of source or target to be defined).
3. The type component defaults to the type of the target type if no matching `type_transition` rule was specified in the policy (policy version 28 allows a `default_type` of source or target to be defined).
4. The `range/level` component defaults to the low/current level of the creating process if no matching `range_transition` rule was specified in the policy (policy version 27 allows a `default_range` of source or target with the selected range being low, high or low-high to be defined).

Notes:

1. Filesystems that support `xattr` extended attributes can be identified via the `mount` command as there will be a `'seclabel'` keyword present.

2. There are mount options for allocating various context types: `context=`, `fscontext=`, `defcontext=` and `rootcontext=`. They are fully described in the `mount` (8) man page.

2.10.1.5 Network File System (nfsv4)

If labeled NFS is implemented with `xattr` support, then the creation of inodes are treated as described in the [Files](#) section.

2.10.1.6 INET Sockets

If a socket is created by the `socket` (3) call they are labeled as follows:

1. The user component is inherited from the creating process (policy version 27 allows a `default_user` of source or target to be defined for each socket object class).
2. The role component is inherited from the creating process (policy version 26 allows a `role_transition` and version 27 allows a `default_role` of source or target to be defined for each socket object class).
3. The type component is inherited from the creating process if no matching `type_transition` rule was specified in the policy and version 28 allows a `default_type` of source or target to be defined for each socket object class).
4. The `range/level` component is inherited from the creating process if no matching `range_transition` rule was specified in the policy (policy version 27 allows a `default_range` of source or target with the selected range being low, high or low-high to be defined for each socket object class).

Security-aware applications may use `setsockcreatecon` (3) to explicitly label sockets they create if permitted by policy.

If created by a connection they are labeled with the context of the listening process.

Some sockets may be labeled with the kernel SID to reflect the fact that they are kernel-internal sockets that are not directly exposed to applications.

2.10.1.7 IPC

Inherits the label of its creator/parent.

2.10.1.8 Message Queues

Inherits the label of its sending process. However if sending a message that is unlabeled, compute a new label based on the current process and the message queue it will be stored in as follows:

1. The user component is inherited from the sending process (policy version 27 allows a `default_user` of source or target to be defined for the message object class).

2. The role component is inherited from the sending process (policy version 26 allows a `role_transition` and version 27 allows a `default_role` of source or target to be defined for the message object class).
3. The type component is inherited from the sending process if no matching `type_transition` rule was specified in the policy and version 28 allows a `default_type` of source or target to be defined for the message object class).
4. The `range/level` component is inherited from the sending process if no matching `range_transition` rule was specified in the policy (policy version 27 allows a `default_range` of source or target with the selected range being low, high or low-high to be defined for the message object class).

2.10.1.9 Semaphores

Inherits the label of its creator/parent.

2.10.1.10 Shared Memory

Inherits the label of its creator/parent.

2.10.1.11 Keys

Inherits the label of its creator/parent.

Security-aware applications may use `setkeycreatecon(3)` to explicitly label keys they create if permitted by policy.

2.10.2 Using `libselinux` Functions

2.10.2.1 `avc_compute_create` and `security_compute_create`

The table below⁸ shows how the components from the source context `scon`, target context `tcon` and class `tclass` are used to compute the new context `newcon` (referenced by SIDs for `avc_compute_create(3)`). The following notes also apply:

- a) Any valid policy `role_transition`, `type_transition` and `range_transition` enforcement rules will influence the final outcome as shown.
- b) For kernels less than 2.6.39 the context generated will depend on whether the class is `process` or any other class.
- c) For kernels 2.6.39 and above the following also applies:
 - i. Those classes suffixed by `socket` will also be included in the `process` class outcome.
 - ii. If a valid `role_transition` rule for `tclass`, then use that instead of the default `object_r`. Also requires policy version 26 or greater - see `security_policyvers(3)`.

⁸ The table only contains the kernel version, the text gives the policy version also required.

- iii. If the `type_transition` rule is classed as the 'file name transition rule' (i.e. it has an `object_name` parameter), then provided the object name in the rule matches the last component of the objects name (in this case a file or directory name), then use the rules `default_type`. Also requires policy version 25 or greater.
- d) For kernels 3.5 and above with policy version 27 or greater, the `default_user`, `default_role`, `default_range` statements will influence the `user`, `role` and `range` of the computed context for the specified class `tclass`. With policy version 28 or greater the `default_type` statement can also influence the `type` in the computed context.

<u>user</u>	<u>role</u>	<u>type</u>	<u>range</u>
<p>If kernel \geq 3.5 with a default_user <code>tclass</code> target rule then use <code>tcon user</code></p> <p>ELSE</p> <p>Use <code>scon user</code></p>	<p>If kernel \geq 2.6.39, and there is a valid role_transition rule then use the rules <code>new_role</code></p> <p>OR</p> <p>If kernel \geq 3.5 with default_role <code>tclass</code> source rule then use <code>scon role</code></p> <p>OR</p> <p>If kernel \geq 3.5 with default_role <code>tclass</code> target rule then use <code>tcon role</code></p> <p>OR</p> <p>If kernel \geq 2.6.39 and <code>tclass</code> is process or <code>*socket</code>, then use <code>scon role</code></p> <p>OR</p> <p>If kernel \leq 2.6.38 and <code>tclass</code> is process, then use <code>scon role</code></p> <p>ELSE</p> <p>Use <code>object_r</code></p>	<p>If there is a valid type_transition rule then use the rules <code>default_type</code></p> <p>OR</p> <p>If kernel \geq 3.5 with default_type <code>tclass</code> source rule then use <code>scon type</code></p> <p>OR</p> <p>If kernel \geq 3.5 with default_type <code>tclass</code> target rule then use <code>tcon type</code></p> <p>OR</p> <p>If kernel \geq 2.6.39 and <code>tclass</code> is process or <code>*socket</code>, then use <code>scon type</code></p> <p>OR</p> <p>If kernel \leq 2.6.38 and <code>tclass</code> is process, then use <code>scon type</code></p> <p>ELSE</p> <p>Use <code>tcon type</code></p>	<p>If there is a valid range_transition rule then use the rules <code>new_range</code></p> <p>OR</p> <p>If kernel \geq 3.5 with default_range <code>tclass</code> source low rule then use <code>scon low</code></p> <p>OR</p> <p>If kernel \geq 3.5 with default_range <code>tclass</code> source high rule then use <code>scon high</code></p> <p>OR</p> <p>If kernel \geq 3.5 with default_range <code>tclass</code> source low_high rule then use <code>scon range</code></p> <p>OR</p> <p>If kernel \geq 3.5 with default_range <code>tclass</code> target low rule then use <code>tcon low</code></p> <p>OR</p> <p>If kernel \geq 3.5 with default_range <code>tclass</code> target high rule then use <code>tcon high</code></p> <p>OR</p> <p>If kernel \geq 3.5 with default_range <code>tclass</code> target low_high rule then use <code>tcon range</code></p> <p>OR</p> <p>If kernel \geq 2.6.39 and <code>tclass</code> is process or <code>*socket</code>, then use <code>scon range</code></p> <p>OR</p> <p>If kernel \leq 2.6.38 and <code>tclass</code> is process, then use <code>scon range</code></p> <p>ELSE</p> <p>Use <code>scon low</code></p>

2.10.2.2 `avc_compute_member` and `security_compute_member`

The table below⁹ shows how the components from the source context, `scon` target context, `tcon` and class, `tclass` are used to compute the new context `newcon` (referenced by SIDs for `avc_compute_member` (3)). The following notes also apply:

- a) Any valid policy `type_member` enforcement rules will influence the final outcome as shown.
- b) For kernels less than 2.6.39 the context generated will depend on whether the class is `process` or any other class.
- c) For kernels 2.6.39 and above, those classes suffixed by `socket` are also included in the `process` class outcome.
- d) For kernels 3.5 and above with policy version 28 or greater, the `default_role`, `default_range` statements will influence the `role` and `range` of the computed context for the specified class `tclass`. With policy version 28 or greater the `default_type` statement can also influence the `type` in the computed context.

⁹ The table only contains the kernel version, the text gives the policy version also required.

<u>user</u>	<u>role</u>	<u>type</u>	<u>range</u>
Always uses <u>tcon user</u>	<p>If kernel \geq 3.5 with default_role <u>tclass</u> source rule then use <u>scon</u> <u>role</u></p> <p style="text-align: center;">OR</p> <p>If kernel \geq 3.5 with default_role <u>tclass</u> target rule then use <u>tcon</u> <u>role</u></p> <p style="text-align: center;">OR</p> <p>If kernel \geq 2.6.39 and <u>tclass</u> is process or *socket, then use <u>scon</u> <u>role</u></p> <p style="text-align: center;">OR</p> <p>If kernel \leq 2.6.38 and <u>tclass</u> is process, then use <u>scon</u> <u>role</u></p> <p style="text-align: center;">ELSE</p> <p>Use object_r</p>	<p>If there is a valid type_member rule then use the rules <u>member_type</u></p> <p style="text-align: center;">OR</p> <p>If kernel \geq 3.5 with default_type <u>tclass</u> source rule then use <u>scon</u> <u>type</u></p> <p style="text-align: center;">OR</p> <p>If kernel \geq 3.5 with default_type <u>tclass</u> target rule then use <u>tcon</u> <u>type</u></p> <p style="text-align: center;">OR</p> <p>If kernel \geq 2.6.39 and <u>tclass</u> is process or *socket, then use <u>scon</u> <u>type</u></p> <p style="text-align: center;">OR</p> <p>If kernel \leq 2.6.38 and <u>tclass</u> is process, then use <u>scon</u> <u>type</u></p> <p style="text-align: center;">ELSE</p> <p>Use <u>tcon</u> <u>type</u></p>	<p>If kernel \geq 3.5 with default_range <u>tclass</u> source low rule then use <u>scon</u> <u>low</u></p> <p style="text-align: center;">OR</p> <p>If kernel \geq 3.5 with default_range <u>tclass</u> source high rule then use <u>scon</u> <u>high</u></p> <p style="text-align: center;">OR</p> <p>If kernel \geq 3.5 with default_range <u>tclass</u> source low_high rule then use <u>scon</u> <u>range</u></p> <p style="text-align: center;">OR</p> <p>If kernel \geq 3.5 with default_range <u>tclass</u> target low rule then use <u>tcon</u> <u>low</u></p> <p style="text-align: center;">OR</p> <p>If kernel \geq 3.5 with default_range <u>tclass</u> target high rule then use <u>tcon</u> <u>high</u></p> <p style="text-align: center;">OR</p> <p>If kernel \geq 3.5 with default_range <u>tclass</u> target low_high rule then use <u>tcon</u> <u>range</u></p> <p style="text-align: center;">OR</p> <p>If kernel \geq 2.6.39 and <u>tclass</u> is process or *socket, then use <u>scon</u> <u>range</u></p> <p style="text-align: center;">OR</p> <p>If kernel \leq 2.6.38 and <u>tclass</u> is process, then use <u>scon</u> <u>range</u></p> <p style="text-align: center;">ELSE</p> <p>Use <u>scon</u> <u>low</u></p>

2.10.2.3 security_compute_relabel

The table below¹⁰ shows how the components from the source context, scon target context, tcon and class, tclass are used to compute the new context newcon for **security_compute_relabel** (3). The following notes also apply:

- a) Any valid policy type_change enforcement rules will influence the final outcome shown in the table.
- b) For kernels less than 2.6.39 the context generated will depend on whether the class is **process** or any other class.
- c) For kernels 2.6.39 and above, those classes suffixed by **socket** are also included in the **process** class outcome.

¹⁰ The table only contains the kernel version, the text gives the policy version also required.

- d) For kernels 3.5 and above with policy version 28 or greater, the `default_user`, `default_role`, `default_range` statements will influence the `user`, `role` and `range` of the computed context for the specified class `tclass`. With policy version 28 or greater the `default_type` statement can also influence the `type` in the computed context.

<u>user</u>	<u>role</u>	<u>type</u>	<u>range</u>
If kernel ≥ 3.5 with a <code>default_user tclass target</code> rule then use <code>tcon user</code> ELSE Use <code>scon user</code>	If kernel ≥ 3.5 with <code>default_role tclass source</code> rule then use <code>scon role</code> OR If kernel ≥ 3.5 with <code>default_role tclass target</code> rule then use <code>tcon role</code> OR If kernel $\geq 2.6.39$ and <code>tclass</code> is <code>process</code> or <code>*socket</code> , then use <code>scon role</code> OR If kernel $\leq 2.6.38$ and <code>tclass</code> is <code>process</code> , then use <code>scon role</code> ELSE Use <code>object_r</code>	If there is a valid <code>type_change</code> rule then use the rules <code>change_type</code> OR If kernel ≥ 3.5 with <code>default_type tclass source</code> rule then use <code>scon type</code> OR If kernel ≥ 3.5 with <code>default_type tclass target</code> rule then use <code>tcon type</code> OR If kernel $\geq 2.6.39$ and <code>tclass</code> is <code>process</code> or <code>*socket</code> , then use <code>scon type</code> OR If kernel $\leq 2.6.38$ and <code>tclass</code> is <code>process</code> , then use <code>scon type</code> ELSE Use <code>tcon type</code>	If kernel ≥ 3.5 with <code>default_range tclass source low</code> rule then use <code>scon low</code> OR If kernel ≥ 3.5 with <code>default_range tclass source high</code> rule then use <code>scon high</code> OR If kernel ≥ 3.5 with <code>default_range tclass source low_high</code> rule then use <code>scon range</code> OR If kernel ≥ 3.5 with <code>default_range tclass target low</code> rule then use <code>tcon low</code> OR If kernel ≥ 3.5 with <code>default_range tclass target high</code> rule then use <code>tcon high</code> OR If kernel ≥ 3.5 with <code>default_range tclass target low_high</code> rule then use <code>tcon range</code> OR If kernel $\geq 2.6.39$ and <code>tclass</code> is <code>process</code> or <code>*socket</code> , then use <code>scon range</code> OR If kernel $\leq 2.6.38$ and <code>tclass</code> is <code>process</code> , then use <code>scon range</code> ELSE Use <code>scon low</code>

2.11 Computing Access Decisions

There are a number of ways to compute access decisions within userspace SELinux-aware applications or object managers:

1. Use functions that do not cache access decisions (i.e. they do not use the `libselinux` AVC services). These require a call to the kernel for every decision using `security_compute_av(3)` or

security_compute_av_flags(3). The **avc_netlink_***(3) functions can be used to detect policy change events. Auditing would need to be implemented if required.

2. Use functions that utilise the `libselinux` userspace AVC services that are initialised with **avc_open**(3). These can be built in various configurations such as:
 - a) Using the default single threaded mode where **avc_has_perm**(3) will automatically cache entries, audit the decision and manage the handling of policy change events.
 - b) Implementing threads or a similar service that will handle policy change events and auditing in real time with **avc_has_perm**(3) or **avc_has_perm_noaudit**(3) handling decisions and caching. This has the advantage of better performance, which can be further increased by caching the entry reference.
3. Implement custom caching services with **security_compute_av**(3) or **security_compute_av_flags**(3) for computing access decisions. The **avc_netlink_***(3) functions can then be used to detect policy change events. Auditing would need to be implemented if required.
4. Use of the **selinux_check_access**(3) function is generally the recommended option provided only one permission requires the check. This utilises the AVC services defined in bullet 2, in a single call with the option to add supplemental auditing information (that is handled as described in **avc_audit**(3)).

Where performance is important when making policy decisions, then the **selinux_status_open**(3), **selinux_status_updated**(3), **selinux_status_getenforce**(3), **selinux_status_policyload**(3) and **selinux_status_close**(3) functions could be used to detect policy updates etc. as these do not require kernel system call over-heads once set up. Note that these functions are only available from `libselinux` 2.0.99, with Linux kernel 2.6.37 and above.

2.12 Domain and Object Transitions

This section discusses the `type_transition` statement that is used to:

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

These transitions can also be achieved using the `libselinux` API functions for SELinux-aware applications.

2.12.1 Domain Transition

A domain transition is where a process in one domain starts a new process in another domain under a different security context. There are two ways a process can define a domain transition:


```
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.

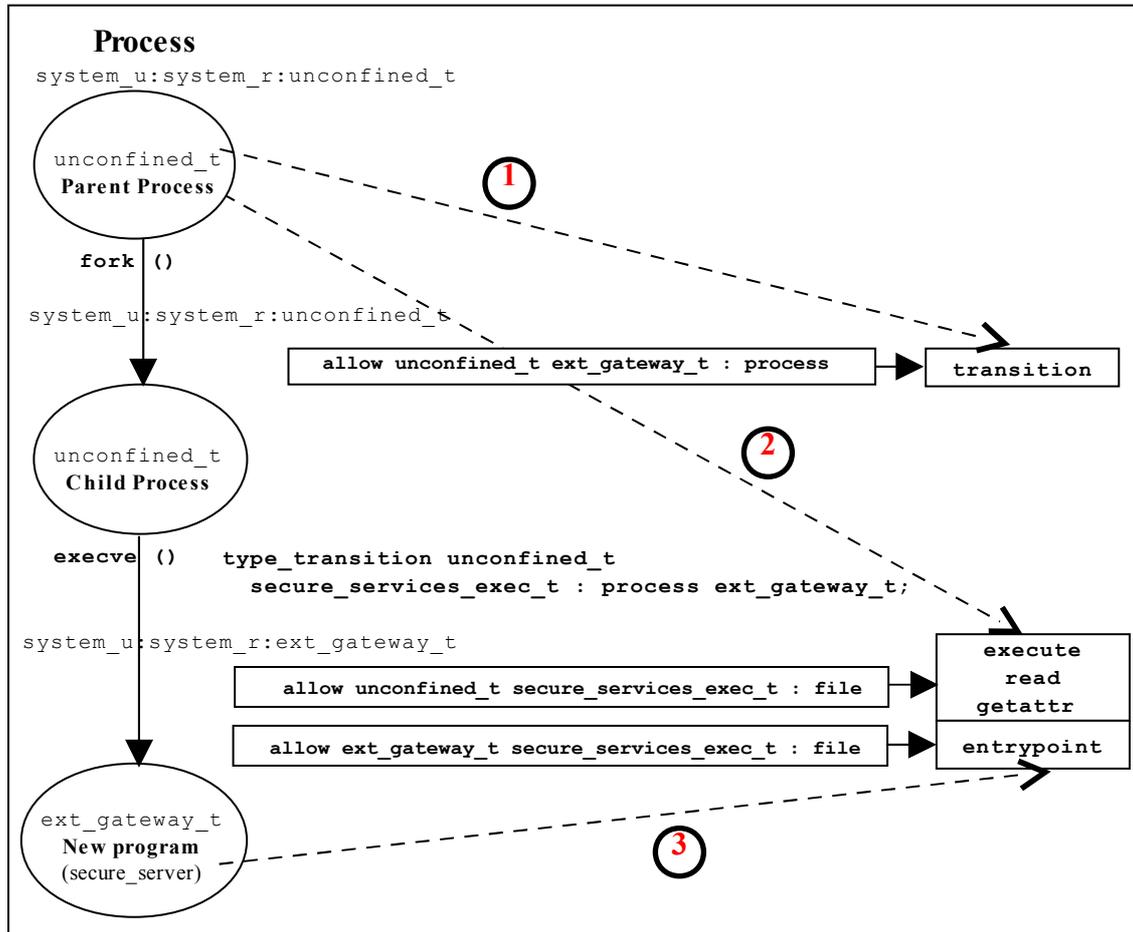


Figure 2.7: Domain Transition - Where the `secure_server` is executed within the `unconfined_t` domain and then transitioned to the `ext_gateway_t` domain.

2.12.1.1 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 allow a single 'default' type for a given source and target (see the Type Rules section). In the above case there were two `type_transition` statements with the same source and target, but different default domains. 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 (or target domains), that breaks 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'd 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 that is 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` (3)

is used to get the executable files context and `setexeccon` (3) is used to set the new process context). If all contexts are correct, then the `execvp` (2) system call is executed that exec's the `secure_server` application with the argument of '1111' into the `int_gateway_t` domain with the `message_filter_r` role. The `runcon` source can be found in the `coreutils` 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.

2.12.2 Object Transition

An object transition is where a new object requires a different label to that of its parent. For example a file is being created that requires a different label to that of its parent directory. This can be achieved automatically using a `type_transition` statement as follows:

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

The following details an object transition used in the `ext_gateway.conf` loadable module (see the source tarball) 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 unconfined_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` rule 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 create 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 unconfined_u:object_r:in_queue_t .
drwxr-xr-x root root system_u:object_r:unconfined_t ..
-rw-r--r-- root root unconfined_u:object_r:in_file_t Message-1
-rw-r--r-- root root unconfined_u:object_r:in_file_t Message-2
```

2.13 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 level or 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.

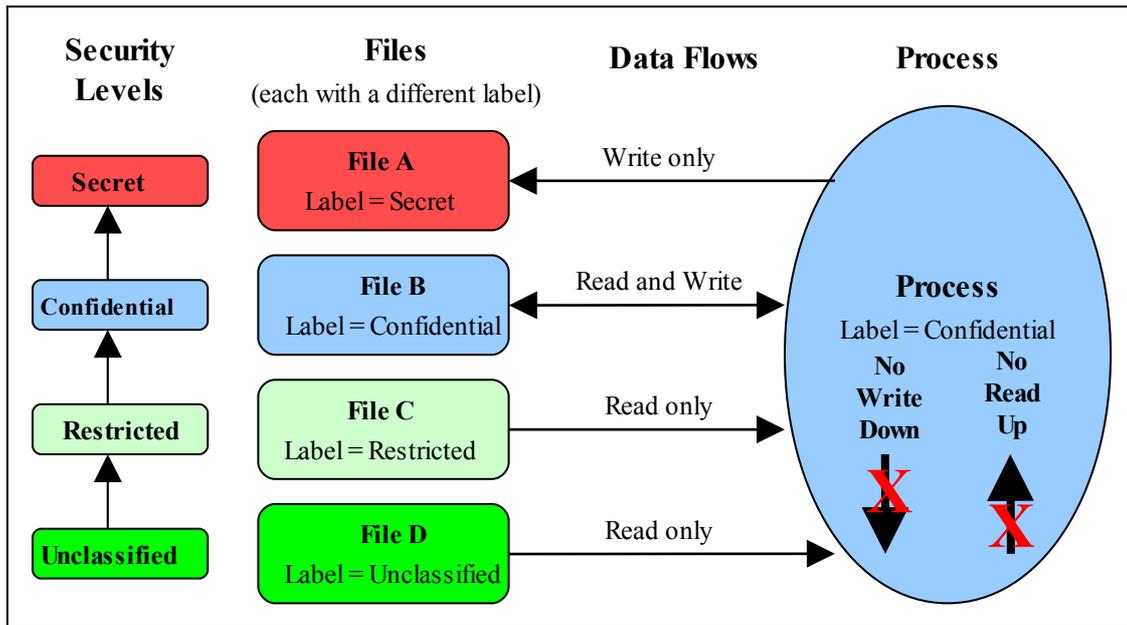


Figure 2.8: Security Levels and Data Flows - This shows how the process can only 'Read Down' and 'Write Up' within an MLS enabled system.

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 dominance rules.

2.13.1 Security Levels

[Table 1](#) shows the components that make up a security level and how two security levels form a range for 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.

	<p>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 <code>s15</code> (although the maximum value for the Reference Policy is a build time option).</p> <p>For MCS systems there is only one sensitivity defined, and that is <code>s0</code>.</p>
<code>category</code>	<p>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 <code>category</code> statement, where for example <code>c0.c3</code> represents a range (<code>c0 c1 c3</code>) and <code>c0, c3, c7</code> represent an unordered list. Traditionally the values are between <code>c0</code> and <code>c255</code> (although the maximum value for the Reference Policy is a build time option).</p>
<code>level</code>	<p>The level is a combination of the sensitivity and <code>category</code> values that form the actual security level. These values are defined using the <code>level</code> statement.</p>

2.13.1.2 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 is used by the `mcstransd`(8) daemon. For example `s0` = Unclassified, `s15` = Top Secret and `c0` = Finance, `c100` = Spy Stories. The `semanage`(8) command can be used to set up this translation and is shown in the [setrans.conf](#) configuration file section.

2.13.2 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](#) 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: MLS Security Levels - Showing the scope of a process running at a security range of `s0 - s3:c1.c5`.

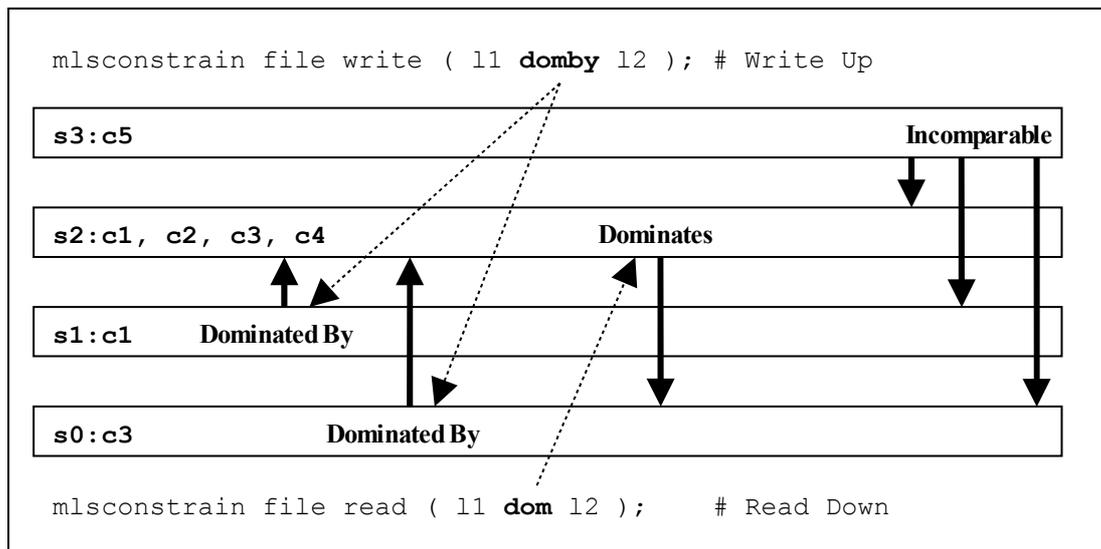


Figure 2.9: Showing the mlsconstrain Statements controlling Read Down & Write Up - This ties in with [Table 2](#) that shows a process running with a security range of `s0 - s3:c1.c5`.

Using [Figure 2.9](#):

1. 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.

2. 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.

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.

2.13.3 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 [SE-PostgreSQL](#) section.

2.13.4 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.

2.14 Types of SELinux Policy

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

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

1. Source code - These can be described as: [Example](#), [Reference Policy](#) or [Custom](#). They are generally written using either [kernel policy language](#), [m4 macro support with kernel policy language](#), or [CIL](#).
2. They can also be 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 22, 23 and 24).

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

2.14.1 Example Policy

The Example policy is the name used to describe the original SELinux policy source used to build a [monolithic](#)¹² policy produced by the NSA and is now superseded by the Reference Policy.

2.14.2 Reference Policy

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

The Reference Policy is now the standard policy source used to build Linux based 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 Linux, however each distribution makes its own specific changes to support their 'version of the Reference Policy'. For example, the F-20 distribution is based on a specific build of the standard Reference Policy that is then modified and distributed by Red Hat as a number of RPMs.

¹² 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).

2.14.3 Policy Functionality Based on Name or Type

Generally a policy is installed with a given name such as `targeted`, `mls`, `refpolicy` or `minimum` that attempts to describes its functionality. This name then becomes the entry in:

1. The directory pointing to the policy location (e.g. if the name is `targeted`, then the policy will be installed in `/etc/selinux/targeted`).
2. The `SELINUXTYPE` entry in the `/etc/selinux/config` file when it is the active policy (e.g. if the name is `targeted`, then a `SELINUXTYPE=targeted` entry would be in the `/etc/selinux/config` file).

This is how the reference policies distributed with F-20 are named, where:

`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.

`targeted` - supports a greater number of confined daemons and can also confine other areas and users. Red Hat pre-configure MCS support within this policy.

`mls` - supports server based MLS systems.

The Reference Policy also has a `TYPE` description that describes the type of policy being built by the build process, these are:

`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 server based MLS systems.

The `NAME` and `TYPE` entries are defined in the reference policy `build.conf` file that is described in the [Source Configuration Files](#) section.

2.14.4 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 e.g. Red Hat policies).
3. A policy that has been built using policy language statements to build a specific policy such as the basic policy built in the Notebook source tarball.

2.14.5 Monolithic Policy

A Monolithic policy is an SELinux policy that is compiled from one source file called (by convention) `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](#).

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

The Reference Policy supports the building of monolithic policies.

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

2.14.6 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 components 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](#)" [3].

2.14.6.1 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.

2.14.7 Conditional Policy

Conditional policies can be implemented in monolithic or loadable module policies and allow parts of the policy to be enabled or not depending on the state of a boolean flag at run time. 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.

2.14.8 Binary Policy

This is also known as the kernel policy and is the policy file that is loaded into the kernel and is located at `/etc/selinux/<SELINUXTYPE>/policy/policy.<version>`. Where `<SELINUXTYPE>` is the policy name specified in the SELinux configuration file `/etc/selinux/config` and `<version>` is the SELinux [policy version](#).

The binary policy can be built from source files supplied by the Reference Policy or custom built source files.

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.29` (as version 29 is supported by F-20):

```
/etc/selinux/targeted/policy/policy.29
```

2.14.9 Policy Versions

SELinux has a policy database (defined in 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 maximum policy version supported by the kernel in its output as follows:

```
SELinux status:          enabled
SELinuxfs mount:        /sys/fs/selinux
Loaded policy name:      targeted
Current mode:           enforcing
Mode from config file:   permissive
Policy MLS status:      enabled
Policy deny_unknown status: allowed
Max kernel policy version: 29
```

[Table 3](#) describes the different versions, although note that 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 give both version numbers).

The SELinux Notebook

<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 <code>validatetrans</code> Statement.
20	-	Reduced the size of the access vector table.
21	6	Added support for the MLS <code>range_transition</code> Statement.
22	7	Added policy capabilities that allows various kernel options to be enabled as described in the SELinux Filesystem section.
23	8	Added support for the <code>permissive</code> statement. This allows a domain to run in permissive mode while the others are still confined (instead of the all or nothing set by the <code>SELINUX</code> entry in the <code>/etc/selinux/config</code> file).
24	9 / 10	Add support for the <code>typebounds</code> statement. This was added to support a hierarchical relationship between two domains in multi-threaded web servers as described in " A secure web application platform powered by SELinux " [16].
25	11	Add support for file name transition in the <code>type_transition</code> rule. Requires kernel 2.6.39 minimum.
26	12/13	Add support for a class parameter in the <code>role_transition</code> rule. Add support for the <code>attribute_role</code> and <code>roleattribute</code> statements. These require kernel 2.6.39 minimum.
-	14	Separate tunables.
27	15	Support setting object defaults for the user, role and range components when computing a new context. Requires kernel 3.5 minimum.
28	16	Support setting object defaults for the type component when computing a new context. Requires kernel 3.5 minimum.
29	17	Support attribute names within constraints. This allows attributes as well as the types to be retrieved from a kernel policy to assist <code>audit2allow</code> (8) etc. to determine what attribute needs to be updated. Note that the attribute does not determine the constraint outcome, it is still the list of

<i>policy db Version</i>	<i>modular db Version</i>	<i>Description</i>
		types associated to the constraint. Requires kernel 3.14 minimum.

Table 3: Policy version descriptions

2.15 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 rules. This is generally used for testing as the audit log will contain the AVC denied messages as defined in the [Auditing SELinux Events](#) 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 is not enabled, therefore no policy can be 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 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/<SELINUXTYPE>/modules/active/modules/permissive_unconfined_t.pp
# and then reload the policy:

semanage permissive -a unconfined_t
```

It is also possible to set permissive mode on a userspace object manager using the `libselinux` function **avc_open**(3), for example the [X-Windows object manager](#) uses **avc_open** to set whether it will always run permissive, enforcing or follow the current SELinux enforcement mode.

The **sestatus**(8) command will show the current SELinux enforcement mode in its output, however it does not display individual domain or object manager enforcement modes.

2.16 Auditing SELinux Events

For SELinux there are two main types of audit event:

1. AVC Audit Events - These are generated by the AVC subsystem as a 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).
2. 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, relabeling etc.

The audit and event messages are generally stored in one of the following logs (in F-20 anyway):

1. The SELinux kernel boot events are logged in the `/var/log/dmesg` log.
2. The system log `/var/log/messages` contains messages generated by SELinux before the audit daemon has been loaded, although some kernel messages continue to be logged here as well¹³.
3. The audit log `/var/log/audit/audit.log` contains events that take place after the audit daemon has been loaded. The AVC audit messages of interest are described in the [AVC Audit Events](#) section with others described in the [General SELinux Audit Events](#) section. F-20 uses the audit framework `auditd`(8) as standard.

Notes:

- a) It is not mandatory for SELinux-aware applications to audit events or even log them in the audit log. The decision is made by the application designer.
- b) The format of audit messages do not need to conform to any format, however where possible applications should use the `audit_log_user_avc_message`(3) function with a suitably formatted message if using `auditd`(8). The type of audit events possible are defined in the `include/libaudit.h` and `include/linux/audit.h` files.
- c) Those libselinux library functions that output messages do so to `stderr` by default, however this can be changed by calling `selinux_set_callback`(3) and specifying an alternative log handler.

2.16.1 AVC Audit Events

[Table 4](#) describes the general format of AVC audit messages in the `audit.log` when access has been denied or an audit event has been specifically requested. Other types of events are shown in the section that follows.

<i>Keyword</i>	<i>Description</i>
type	<p>For SELinux AVC events this can be:</p> <p style="padding-left: 2em;"><code>type=AVC</code> - for kernel events</p> <p style="padding-left: 2em;"><code>type=USER_AVC</code> - for user-space object manager events</p> <p>Note that once the AVC event has been logged, another event with <code>type=SYSCALL</code> may follow that contains further information regarding the event.</p> <p>The AVC event can always be tied to the relevant <code>SYSCALL</code> event as they have the same <code>serial_number</code> in the <code>msg=audit(time:serial_number)</code> field as shown in the</p>

¹³ 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 not the audit log.

The SELinux Notebook

<i>Keyword</i>	<i>Description</i>
	<p>following example:</p> <pre style="border: 1px solid black; padding: 5px;"> 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" exe="/bin/ls" subj=system_u:object_r:unlabeled_t:s0 key=(null) </pre>
msg	This will contain the audit keyword with a reference number (e.g. msg=audit(1243332701.744:101))
avc	<p>This will be either denied when access has been denied or granted when an auditallow rule has been defined by the policy.</p> <p>The entries that follow the avc= 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 (pid) and the name of the executable file (comm).
comm	
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 (AF_INET).
lport	
faddr	
fport	
path	If a File Socket event then log the path (AF_UNIX).
saddr	If a Network event then log the Source / Destination addresses and ports with the network interface for IP4 or IP6 networks (AF_INET).
src	
daddr	
dest	
netif	
said	IPSec security association identifiers
hostname	

Keyword	Description
addr	
terminal	
resid	X-Windows resource ID and type.
restype	
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 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:oddjob_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:oddjob_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:oddjob_mkhomedir_t:s0 key=(null)
```

```
# These are example X-Windows object manager audit message:

type=USER_AVC msg=audit(1267534171.023:18): user pid=1169 uid=0 auid=4294967295
ses=4294967295 subj=system_u:unconfined_r:unconfined_t msg='avc: denied
{ getfocus } for request=X11:GetInputFocus comm=X-setest xdevice="Virtual core
keyboard" scontext=unconfined_u:unconfined_r:x_select_paste_t
tcontext=system_u:unconfined_r:unconfined_t tclass=x_keyboard :
exe="/usr/bin/Xorg" sauid=0 hostname=? addr=? terminal=?'

type=USER_AVC msg=audit(1267534395.930:19): user pid=1169 uid=0 auid=4294967295
ses=4294967295 subj=system_u:unconfined_r:unconfined_t msg='avc: denied { read
} for request=SELinux:SELinuxGetClientContext comm=X-setest resid=3c00001
restype=<unknown> scontext=unconfined_u:unconfined_r:x_select_paste_t
tcontext=unconfined_u:unconfined_r:unconfined_t tclass=x_resource :
exe="/usr/bin/Xorg" sauid=0 hostname=? addr=? terminal=?'
```

```
# 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=unconfined_u:unconfined_r:unconfined_t
tcontext=unconfined_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=unconfined_u:unconfined_r:move_file_t key=(null)
```

2.16.2 General SELinux Audit Events

This section shows a selection of non-AVC SELinux-aware services audit events taken from the `audit.log`. For a list of valid `type=` entries, the following include files should be consulted: `include/libaudit.h` and `include/linux/audit.h`.

Note that there can be what appears to be multiple events being generated for the same event. For example the kernel security server will generate a `MAC_POLICY_LOAD` event to indicate that the policy has been reloaded, but then each userspace object manager could then generate a `USER_MAC_POLICY_LOAD` event to indicate that it had also processed the event.

Policy reload - `MAC_POLICY_LOAD`, `USER_MAC_POLICY_LOAD` - These events were generated when the policy was reloaded.

```
type=MAC_POLICY_LOAD msg=audit(1336662937.117:394): policy loaded auid=0 ses=2  
type=SYSCALL msg=audit(1336662937.117:394): arch=c000003e syscall=1 success=yes  
exit=4345108 a0=4 a1=7f0a0c547000 a2=424d14 a3=7fffe3450f20 items=0 ppid=3845  
pid=3848 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts2  
ses=2 comm="load_policy" exe="/sbin/load_policy"  
subj=unconfined_u:unconfined_r:load_policy_t:s0-s0:c0.c1023 key=(null)  
  
type=USER_MAC_POLICY_LOAD msg=audit(1336662938.535:395): pid=0 uid=0  
auid=4294967295 ses=4294967295 subj=system_u:system_r:xserver_t:s0-s0:c0.c1023  
msg='avc: received policyload notice (seqno=2) : exe="/usr/bin/Xorg" sauid=0  
hostname=? addr=? terminal=?'
```

Change enforcement mode - `MAC_STATUS` - This was generated when the SELinux enforcement mode was changed:

```
type=MAC_STATUS msg=audit(1336836093.835:406): enforcing=1 old_enforcing=0  
auid=0 ses=2  
type=SYSCALL msg=audit(1336836093.835:406): arch=c000003e syscall=1 success=yes  
exit=1 a0=3 a1=7fffe743f9e0 a2=1 a3=0 items=0 ppid=2047 pid=5591 auid=0 uid=0  
gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=2  
comm="setenforce" exe="/usr/sbin/setenforce"  
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 key=(null)
```

Change boolean value - `MAC_CONFIG_CHANGE` - This event was generated when `setsebool` (8) was run to change a boolean. Note that the boolean name plus new and old values are shown in the `MAC_CONFIG_CHANGE` type event with the `SYSCALL` event showing what process executed the change.

```
type=MAC_CONFIG_CHANGE msg=audit(1336665376.629:423):  
bool=domain_paste_after_confirm_allowed val=0 old_val=1 auid=0 ses=2  
type=SYSCALL msg=audit(1336665376.629:423): arch=c000003e syscall=1 success=yes  
exit=2 a0=3 a1=7fff42803200 a2=2 a3=7fff42803f80 items=0 ppid=2015 pid=4664  
auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=2  
comm="setsebool" exe="/usr/sbin/setsebool"  
subj=unconfined_u:unconfined_r:setsebool_t:s0-s0:c0.c1023 key=(null)
```

NetLabel - MAC_UNLBL_STCADD - Generated when adding a static non-mapped label. There are many other NetLabel events possible, such as: MAC_MAP_DEL, MAC_CIPSOV4_DEL ...

```
type=MAC_UNLBL_STCADD msg=audit(1336664587.640:413): netlabel: auid=0 ses=2
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 netif=lo
src=127.0.0.1 sec_obj=system_u:object_r:unconfined_t:s0-s0:c0,c100 res=1
type=SYSCALL msg=audit(1336664587.640:413): arch=c000003e syscall=46 success=yes
exit=96 a0=3 a1=7fffde77f160 a2=0 a3=666e6f636e753a72 items=0 ppid=2015 pid=4316
auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=2
comm="netlabelctl" exe="/sbin/netlabelctl"
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 key=(null)
```

Labeled IPsec - MAC_IPSEC_EVENT - Generated when running **setkey**(8) to load IPsec configuration:

```
type=MAC_IPSEC_EVENT msg=audit(1336664781.473:414): op=SAD-add auid=0 ses=2
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 sec_alg=1 sec_doi=1
sec_obj=system_u:system_r:postgresql_t:s0-s0:c0,c200 src=127.0.0.1 dst=127.0.0.1
spi=592(0x250) res=1
type=SYSCALL msg=audit(1336664781.473:414): arch=c000003e syscall=44 success=yes
exit=176 a0=4 a1=7fff079d5100 a2=b0 a3=0 items=0 ppid=2015 pid=4356 auid=0 uid=0
gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=2 comm="setkey"
exe="/sbin/setkey" subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
key=(null)
```

SELinux kernel errors - SELINUX_ERR - These example events were generated by the kernel security server. These were generated by the kernel security server because anon_webapp_t has been give privileges that are greater than that given to the process that started the new thread (this is not allowed).

```
type=SELINUX_ERR msg=audit(1311948547.151:138): op=security_compute_av
reason=bounds scontext=system_u:system_r:anon_webapp_t:s0-s0:c0,c100,c200
tcontext=system_u:object_r:security_t:s0 tclass=dir perms=ioctl,read,lock

type=SELINUX_ERR msg=audit(1311948547.151:138): op=security_compute_av
reason=bounds scontext=system_u:system_r:anon_webapp_t:s0-s0:c0,c100,c200
tcontext=system_u:object_r:security_t:s0 tclass=file
perms=ioctl,read,write,getattr,lock,append,open
```

These were generated by the kernel security server when an SELinux-aware application was trying to use **setcon**(3) to create a new thread. To fix this a typebounds statement is required in the policy.

```
type=SELINUX_ERR msg=audit(1311947138.440:126): op=security_bounded_transition
result=denied oldcontext=system_u:system_r:httpd_t:s0-s0:c0.c300
newcontext=system_u:system_r:anon_webapp_t:s0-s0:c0,c100,c200

type=SYSCALL msg=audit(1311947138.440:126): arch=c000003e syscall=1 success=no
exit=-1 a0=b a1=7f1954000a10 a2=33 a3=6e65727275632f72 items=0 ppid=3295
pid=3473 auid=4294967295 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48
fsgid=48 tty=(none) ses=4294967295 comm="httpd" exe="/usr/sbin/httpd"
subj=system_u:system_r:httpd_t:s0-s0:c0.c300 key=(null)
```

Role changes - USER_ROLE_CHANGE - Used **newrole**(1) to set a new role that was not valid.

```
type=USER_ROLE_CHANGE msg=audit(1336837198.928:429): pid=0 uid=0 auid=0 ses=2
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 msg='newrole: old-
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 new-context=?
exe="/usr/bin/newrole" hostname=? addr=? terminal=/dev/pts/0 res=failed'
```

2.17 Polyinstantiation Support

GNU / Linux supports the polyinstantiation of directories 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](#)" [4] also gives a more detailed overview of the subject.

Polyinstantiation of objects is also supported for X-windows selections and properties that are discussed in the X-windows section. Note that sockets are not yet supported.

To clarify polyinstantiation support:

1. SELinux has `libselinux` functions and a policy rule to support polyinstantiation.
2. The polyinstantiation of directories is a function of GNU / Linux not SELinux (as more correctly, the GNU / Linux services such as PAM have been modified to support polyinstantiation of directories and have also been made SELinux-aware. Therefore their services can be controlled via policy).
3. The polyinstantiation of X-windows selections and properties is a function of the XSELinux Object Manager and the supporting XACE service.

2.17.1 Polyinstantiated Objects

Determining a polyinstantiated context for an object is supported by SELinux using the policy language `type_member` statement and the `avc_compute_member(3)` and `security_compute_member(3)` `libselinux` API functions. These are not limited to specific object classes, however only `dir`, `x_selection` and `x_property` objects are currently supported.

2.17.2 Polyinstantiation support in PAM

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

The default installation of F-20 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-20 already has these

entries configured, with an example `/etc/pam.d/gdm-password` file being:

```
auth      [success=done ignore=ignore default=bad] pam_selinux_permit.so
auth      substack      password-auth
auth      optional      pam_gnome_keyring.so
auth      include       postlogin

account   required      pam_nologin.so
account   include       password-auth

password  include       password-auth

session   required      pam_selinux.so close
session   required      pam_loginuid.so
session   optional      pam_console.so
-session  optional      pam_ck_connector.so
session   required      pam_selinux.so open
session   optional      pam_keyinit.so force revoke
session   required      pam_namespace.so
session   include       password-auth
session   optional      pam_gnome_keyring.so auto_start
session   include       postlogin
```

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-20 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-20 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 a 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.

2.17.2.1 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 <code>\$USER</code> and <code>\$HOME</code> will be replaced by the user name and home directory
---------	---

	respectively.
<code>instance_prefix</code>	A string prefix used to build the pathname for the polyinstantiated directory. The optional strings <code>\$USER</code> and <code>\$HOME</code> will be replaced by the user name and home directory respectively.
<code>method</code>	This is used to determine the method of polyinstantiation with valid entries being: <p style="margin-left: 40px;"><code>user</code> - Polyinstantiation is based on user name.</p> <p style="margin-left: 40px;"><code>level</code> - Polyinstantiation is based on the user name and MLS level.</p> <p style="margin-left: 40px;"><code>context</code> - Polyinstantiation is based on the user name and security context.</p> <p>Note that <code>level</code> and <code>context</code> are only valid for SELinux enabled systems.</p>
<code>list_of_uids</code>	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. <p>There are a number of optional flags available that are described in the namespace.conf (5) man page.</p>

2.17.2.2 Example Configurations

This section shows two sample `namespace.conf` configurations, the first uses the `method=user` and the second `method=context`. It should be noted that while polyinstantiation is enabled, the full path names will not be visible, it is only when polyinstantiation is disabled that the directories become visible.

Example 1 - `method=user`:

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

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

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

<code># The directories will contain the user name as a part of</code>
<code># the polyinstantiated directory name as follows:</code>
<code># /tmp</code>
<code>/tmp/tmp-inst/rch</code>
<code># /var/tmp:</code>

```
/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/unconfined_u:unconfined_r:unconfined_t_rch

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

# $HOME
/home/rch/rch.inst/unconfined_u:unconfined_r:unconfined_t_rch
```

2.17.3 Polyinstantiation support in X-Windows

The X-Windows SELinux object manager and XACE (X Access Control Extension) supports `x_selection` and `x_property` polyinstantiated objects as discussed in the [SELinux X-windows Support](#) section.

2.17.4 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).

The polyinstantiation of X-Windows objects (`x_selection` and `x_property`) are not currently supported by the reference policy.

2.18 PAM Login Process

Applications used to provide login services (such as `gdm` and `ssh`) in F-20 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-20 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-20 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: [value1=action1 value2=action2 ..]

	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-20 uses the `/etc/pam.d` directory configuration (in this case `gdm-password` for the Gnome login service).

```
auth      [success=done ignore=ignore default=bad] pam_selinux_permit.so
auth      substack      password-auth
auth      optional     pam_gnome_keyring.so
auth      include      postlogin

account   required     pam_nologin.so
account   include      password-auth

password  include      password-auth

session   required     pam_selinux.so close debug
session   required     pam_loginuid.so
session   optional     pam_console.so
-session  optional     pam_ck_connector.so
session   required     pam_selinux.so open debug
session   optional     pam_keyinit.so force revoke
session   required     pam_namespace.so
session   include      password-auth
session   optional     pam_gnome_keyring.so auto_start
session   include      postlogin
```

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.

2.19 Linux Security Module and SELinux

This section gives a high level overview of the LSM and SELinux internal kernel structure and workings as enabled in kernel 3.14. A more detailed view can be found in the "[Implementing SELinux as a Linux Security Module](#)" [1] that was used

extensively to develop this section (and also using 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` and `exec` system calls are followed through as an example to tie in with the transition process covered in the [Domain Transition](#) section.
4. The SELinux filesystem `/sys/fs/selinux`.
5. The `/proc` filesystem area most applicable to SELinux.

2.19.1 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 five 3rd party services that utilise the LSM:

1. SELinux - the subject of this Notebook.
2. AppArmor is a MAC service based on pathnames and does not require labeling or relabeling of filesystems. See <http://wiki.apparmor.net> for details.
3. Simplified Mandatory Access Control Kernel (SMACK). See <http://www.schaufler-ca.com/> for details.
4. Tomoyo that is a name based MAC and details can be found at <http://sourceforge.jp/projects/tomoyo/docs>.
5. Yama extends the DAC support for `ptrace`. See `Documentation/security/Yama.txt` for further details.

The basic idea behind LSM is to:

- Insert security function 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 5](#) with an example task and program execution shown in the [Fork Walk-thorough](#) and [Process Transition Walk-thorough](#) sections.
- Allow registration and initialisation services for the 3rd party security modules.
- Allow process security attributes to be available to userspace services by extending the `/proc` filesystem with a security namespace as shown in [Table 6](#). These are located at:

```
/proc/<self | pid>/attr/<attr>  
/proc/<self | pid>/task/<tid>/attr/<attr>
```

Where `<pid>` is the process id, `<tid>` is the thread id and `<attr>` is the entry described in [Table 6](#).

- Support filesystems that use extended attributes (SELinux uses `security.selinux` as explained in the [Labeling Extended Attribute Filesystems](#) section).
- 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 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 `./linux-3.14/include/linux/security.h`).*

<code>/proc/self/attr/ File Name</code>	Permissions	Function
current	-rw-rw-rw-	Contains the current process security context.
exec	-rw-rw-rw-	Used to set the security context for the next exec call.
fscreate	-rw-rw-rw-	Used to set the security context of a newly created file.
keycreate	-rw-rw-rw-	Used to set the security context for keys that are cached in the kernel.
prev	-r--r--r--	Contains the previous process security context.
sockcreate	-rw-rw-rw-	Used to set the security context of a newly created socket.

Table 6: /proc Filesystem attribute files - *These files are used by the kernel services and libselinux (for userspace) to manage setting and reading of security contexts within the LSM defined data structures.*

The major kernel source files (relative to `./linux-3.14/security`) that form the LSM are shown in [Table 7](#). However there is one major header file (`include/linux/security.h`) that describes all the LSM security hooks and structures.

Name	Function
capability.c	Some capability functions were in various kernel modules have been consolidated into these source files.
commoncap.c	
device_cgroup.c	
inode.c	This allows the 3 rd party security module to initialise a security filesystem. In the case of SELinux this would be <code>/sys/fs/selinux</code> that is defined in the <code>selinux/selinuxfs.c</code> source file.
security.c	Contains the LSM framework initialisation services that will set up the hooks described in <code>security.h</code> and those in the capability source files. It also provides functions to initialise 3 rd party modules.
lsm_audit.c	Contains common LSM audit functions.

Name	Function
min_addr.c	Minimum VM address protection from userspace for DAC and LSM.

Table 7: The core LSM source modules.

2.19.2 The SELinux Module

This section does not go into detail of all the SELinux module functionality as the [Implementing SELinux as a Linux Security Module](#) [1] does this (although a bit dated), however it attempts to highlight the way some areas work by using the [fork and transition process example](#) described in the [Domain Transition](#) section.

The major kernel SELinux source files (relative to `./linux-3.14/security/selinux`) that form the SELinux security module are shown in [Table 8](#). 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.
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 <code>security_ops</code> 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 <code>net*</code> 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.
nlmsgtab.c	
selinuxfs.c	The <code>selinuxfs</code> pseudo filesystem (<code>/sys/fs/selinux</code>) that imports/exports security policy information to/from userspace services. The services exported are shown in the SELinux Filesystem section.
xfrm.c	Contains the IPSec XFRM (transform) hooks for SELinux.
include/classmap.h	<code>classmap.h</code> contains all the kernel security classes and permissions. <code>initial_sid_to_string.h</code> contains the initial SID contexts. These are used to build the <code>flask.h</code> and <code>av_permissions.h</code> kernel configuration files when the kernel is being built (using the <code>genheaders</code> script defined in the <code>selinux/Makefile</code>). These files are built this way now to support the new dynamic security class mapping structure to remove the need for fixed class to SID mapping.
include/initial_sid_to_string.h	
ss/avtab.c	AVC table functions for inserting / deleting entries.
ss/conditional.c	Support boolean statement functions and implements a conditional AV

Name	Function
	table to hold entries.
ss/ebitmap.c	Bitmaps to represent sets of values, such as types, roles, categories, and classes.
ss/hashtab.c	Hash table.
ss/mls.c	Functions to support MLS.
ss/policydb.c	Defines the structure of the policy database. See the " SELinux Policy Module Primer " [3] 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/status.c	Interface for <code>selinuxfs/status</code> . Used by the <code>libselinux selinux_status_*(3)</code> functions.
ss/symtab.c	Maintains associations between symbol strings and their values.

Table 8: The core SELinux source modules - The `.h` files and those in the `include` directory have a number of useful comments.

2.19.2.1 Fork System Call Walk-through

This section walks through the the `fork(2)` 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` function table 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 check whether the task has permission via the `current_has_perm(current, PROCESS__FORK)` function.
4. 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 for an entry. Assuming that there is no entry in the AVC, then the `security_compute_av()` function in `services.c` is called.
5. 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 checks 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) `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.
 - g) Finally check whether there are any constraints applied via the `typebounds` rule.
6. 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.
 7. The End.

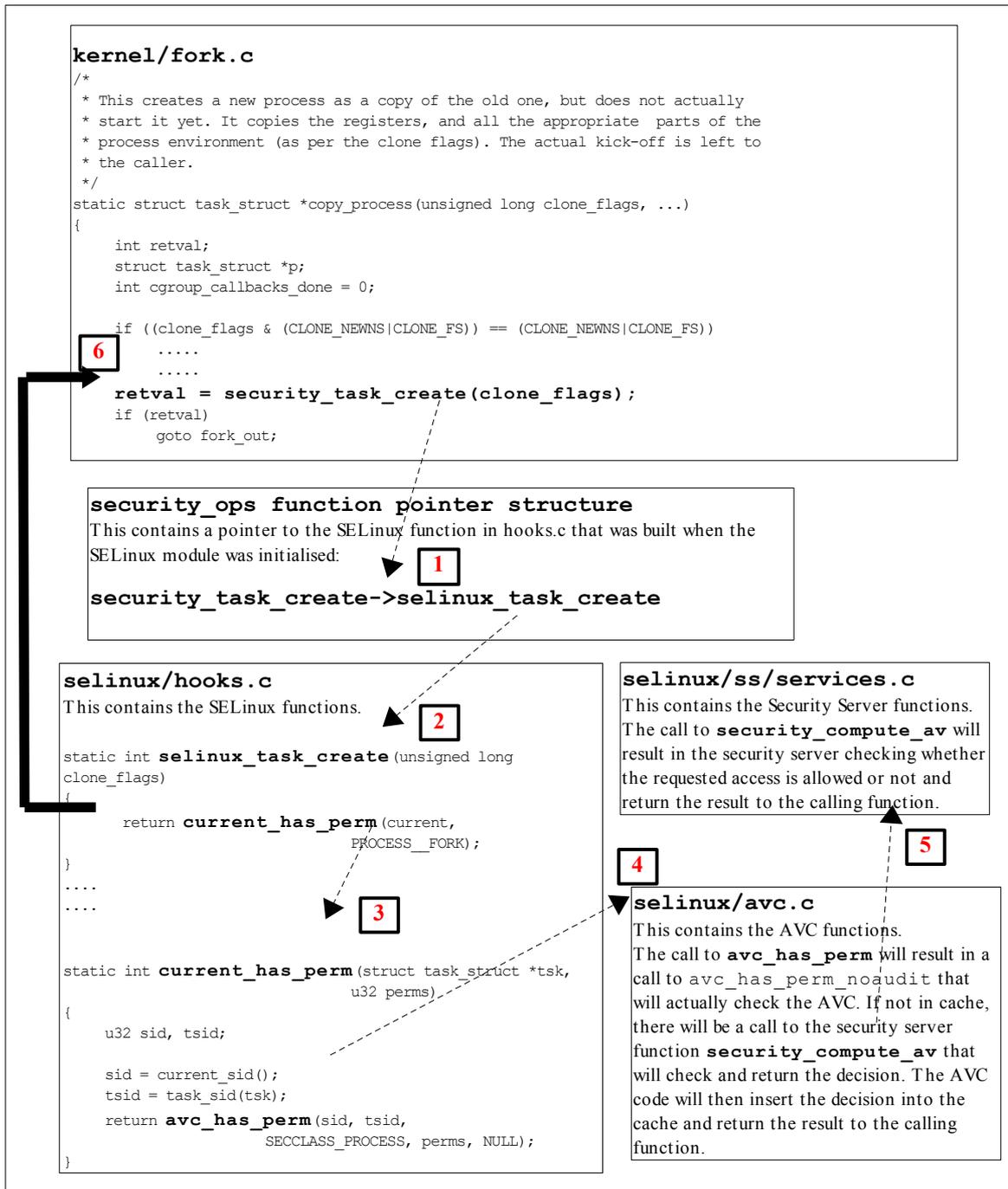


Figure 2.10: Hooks for the fork system call - This describes the steps required to check access permissions for Object Class 'process' and permission 'fork'.

2.19.2.2 Process Transition Walk-through

This section walks through the `execve(2)` and checking whether a process transition to the `ext_gateway_t` domain is allowed, and if so obtain a new SID for the context (`unconfined_u:message_filter_r:ext_gateway_t`) as shown in [Figure 2.7](#).

The process starts with the Linux operating system issuing a `do_execve14` call from the CPU specific architecture code to execute a new program (for example, from

¹⁴ This function call will pass over the file name to be run and its environment + arguments.

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 `include/linux/security.h`), and are hooked by SELinux during the initialisation process (in `security/selinux/hooks.c`). [Table 9](#) briefly describes these `security_bprm` functions that are hooks for validating program loading and execution (although see `security.h` for greater detail).

LSM / SELinux Function Name	Description
<code>security_bprm_set_creds-></code> <code>selinux_bprm_set_creds</code>	Set up security information in the <code>bprm->security</code> field based on the file to be exec'ed contained in <code>bprm->file</code> . SELinux uses this hook to check for domain transitions and the whether the appropriate permissions have been granted, and obtaining a new SID if required.
<code>security_bprm_committing_creds-></code> <code>selinux_bprm_committing_creds</code>	Prepare to install the new security attributes of the process being transformed by an <code>execve</code> operation. SELinux uses this hook to close any unauthorised files, clear parent signal and reset resource limits if required.
<code>security_bprm_committed_creds-></code> <code>selinux_bprm_committed_creds</code>	Tidy up after the installation of the new security attributes of a process being transformed by an <code>execve</code> operation. SELinux uses this hook to check whether signal states can be inherited if new SID allocated.
<code>security_bprm_secureexec-></code> <code>selinux_bprm_secureexec</code>	Called when loading libraries to check <code>AT_SECURE</code> flag for glibc secure mode support. SELinux uses this hook to check the process class <code>noatsecure</code> permission if appropriate.
<code>security_bprm_check-></code> <code>selinux_bprm_check_security</code>	This hook is not used by SELinux.

Table 9: 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).
2. Via the `prepare_binprm()` function call the UID and GIDs are checked and a call issued to `security_bprm_set_creds()` that will carry out the following:
3. Call `cap_bprm_set_creds` function in `commoncap.c`, that will set up credentials based on any configured capabilities.

If `setexeccon(3)` has been called prior to the `exec`, then that context will be used otherwise call `security_transition_sid()` function in `services.c`. This function will then call `security_compute_sid()`

to check whether a new SID needs to be computed. This function 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.
4. The `selinux_bprm_set_creds()` function will continue by checking via the `avc_has_perm()` functions (in `avc.c`) whether the file class `file_execute_no_trans` is set (in this case it is not), therefore the process class transition and file class `file_entrypoint` permissions are checked (in this case they are allowed), therefore the new SID is set, and after checking various other permissions, control is passed back to the `do_execve` function.
 5. The `exec_binprm` function will ultimately commit the credentials calling the SELinux `selinux_bprm_committing_creds` and `selinux_bprm_committed_creds`.
 6. 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 ultimately called to free the `bprm` security structure.
 7. The End.

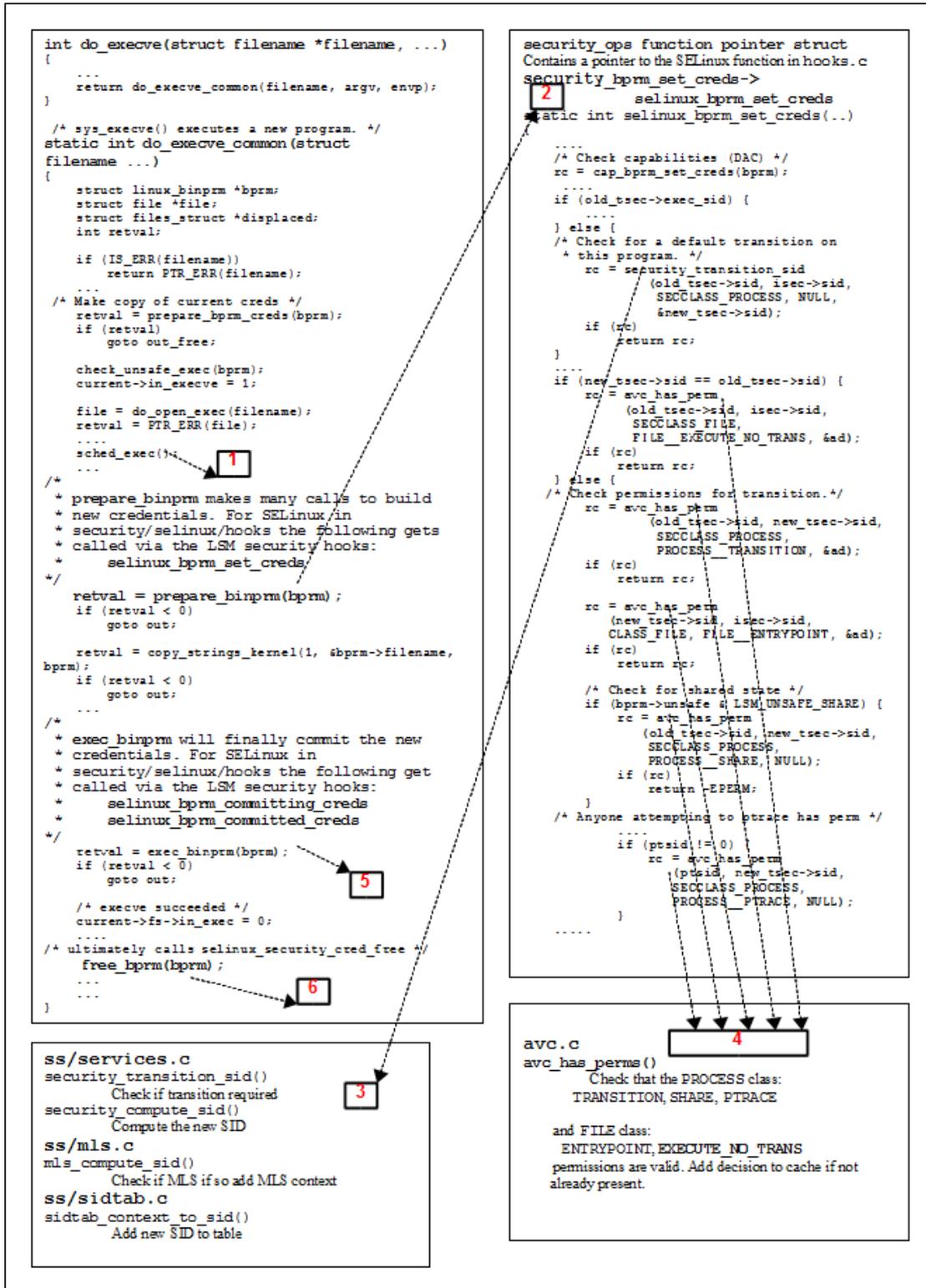


Figure 2.11: Process Transition - This shows the major steps required to check if a transition is allowed from the `unconfined_t` domain to the `ext_gateway_t` domain.

The SELinux Notebook

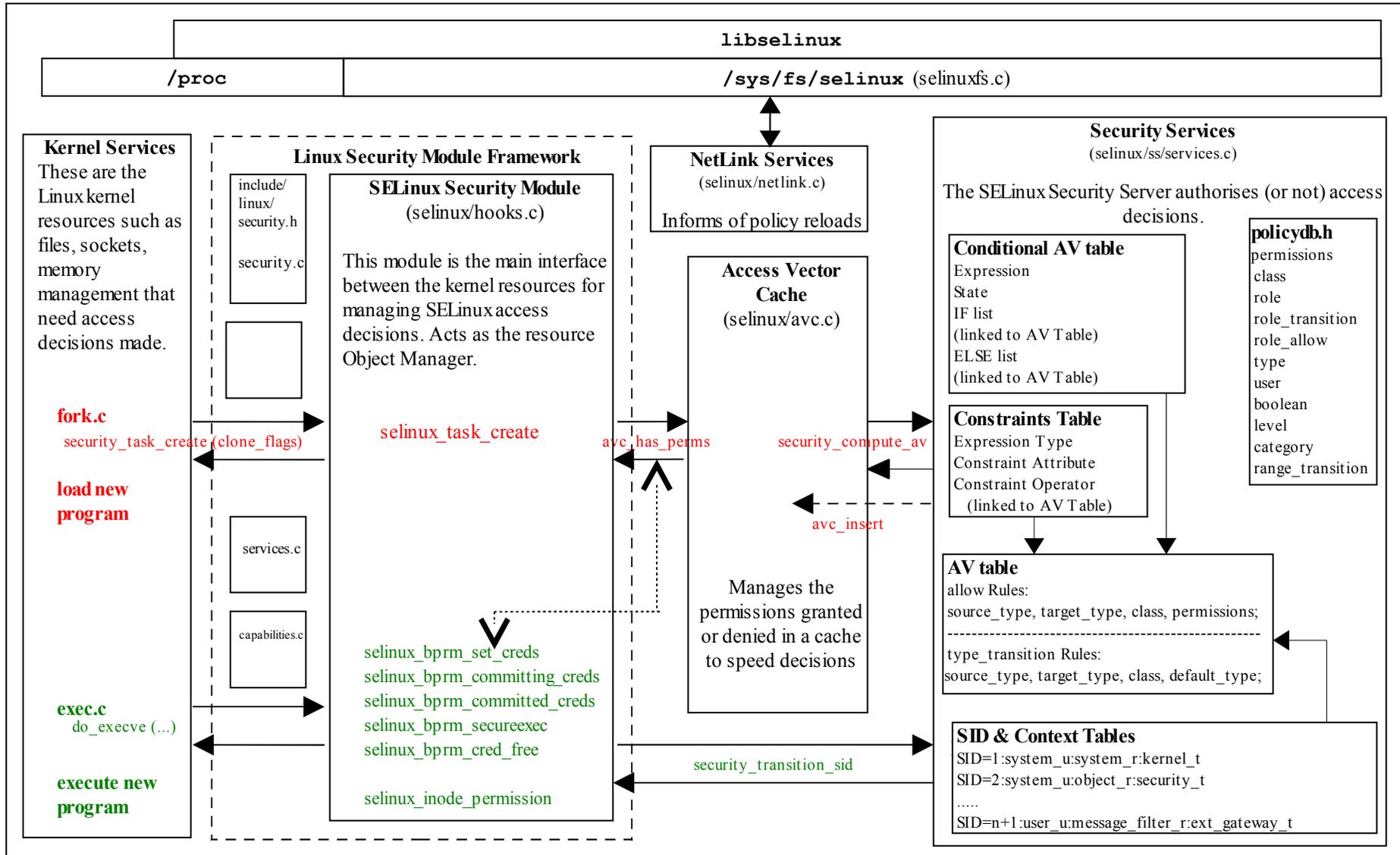


Figure 2.12: The Main LSM / SELinux Modules - The fork and exec functions link to [Figure 2.7](#) where the transition process is described.

2.19.2.3 SELinux Filesystem

[Table 10](#) shows the information contained in the SELinux filesystem (`selinuxfs`) `/sys/fs/selinux` (or `/selinux` on older systems) where the SELinux kernel exports information regarding its configuration and active policy. `selinuxfs` is a read/write interface used by SELinux library functions for userspace SELinux-aware applications and object managers. Note: while it is possible for userspace applications to read/write to this interface, it is not recommended - use the `libselinux` library.

<i>selinuxfs Directory and File Names</i>	<i>Permissions</i>	<i>Comments</i>
<code>/sys/fs/selinux</code>	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.
<code>access</code>	<code>-rw-rw-rw-</code>	Compute access decision interface that is used by the <code>security_compute_av(3)</code> , <code>security_compute_av_flags(3)</code> , <code>avc_has_perm(3)</code> and <code>avc_has_perm_noaudit(3)</code> functions. The kernel security server (see <code>services.c</code>) converts the contexts to SIDs and then calls the <code>security_compute_av_user</code> function to compute the new SID that is then converted to a context string. Requires <code>security {compute_av}</code> permission.
<code>checkreqprot</code>	<code>-rw-r--r--</code>	0 = Check requested protection applied by kernel. 1 = Check protection requested by application. This is the default. These apply to the <code>mmap</code> and <code>mprotect</code> kernel calls. Default value can be changed at boot time via the <code>checkreqprot=</code> parameter. Requires <code>security {setcheckreqprot}</code> permission.
<code>commit_pending_bools</code>	<code>--w-----</code>	Commit new boolean values to the kernel policy. Requires <code>security {setbool}</code> permission.
<code>context</code>	<code>-rw-rw-rw-</code>	Validate context interface used by the <code>security_check_context(3)</code> function. Requires <code>security {check_context}</code> permission.

The SELinux Notebook

<i>selinuxfs Directory and File Names</i>	<i>Permissions</i>	<i>Comments</i>
create	-rw-rw-rw-	<p>Compute create labeling decision interface that is used by the security_compute_create(3) and avc_compute_create(3) functions.</p> <p>The kernel security server (see <code>services.c</code>) converts the contexts to SIDs and then calls the <code>security_transition_sid_user</code> function to compute the new SID that is then converted to a context string.</p> <p>Requires <code>security {compute_create}</code> permission.</p>
deny_unknown	-r--r--r--	<p>These two files export <code>deny_unknown</code> (read by security_deny_unknown(3) function) and <code>reject_unknown</code> status to user space.</p> <p>These are taken from the <code>handle-unknown</code> parameter set¹⁵ in the /etc/selinux/semanage.conf file when policy is being built and are set as follows:</p> <pre>deny:reject 0:0 = Allow unknown object class / permissions. This will set the returned AV with all 1's. 1:0 = Deny unknown object class / permissions (the default). This will set the returned AV with all 0's. 1:1 = Reject loading the policy if it does not contain all the object classes / permissions.</pre>
reject_unknown	-r--r--r--	
disable	--w-----	Disable SELinux until next reboot.
enforce	-rw-r--r--	<p>Get or set enforcing status.</p> <p>Requires <code>security {setenforce}</code> permission.</p>
load	-rw-----	<p>Load policy interface.</p> <p>Requires <code>security {load_policy}</code> permission.</p>
member	-rw-rw-rw-	<p>Compute polyinstantiation membership decision interface that is used by the security_compute_member(3) and avc_compute_member(3) functions.</p> <p>The kernel security server (see <code>services.c</code>) converts the contexts to SIDs and then calls the <code>security_member_sid</code> function to compute the new SID that is then converted to a context string.</p> <p>Requires <code>security {compute_member}</code> permission.</p>
mls	-r--r--r--	Returns 1 if MLS policy is enabled or 0 if not.

¹⁵ This is also set in the `UNK_PERMS` entry of the Reference Policy [build.conf](#) file. The entry in `semanage.conf` will over-ride the `build.conf` entry.

The SELinux Notebook

<i>selinuxfs Directory and File Names</i>	<i>Permissions</i>	<i>Comments</i>
null	crw-rw-rw-	The SELinux equivalent of /dev/null for file descriptors that have been redirected by SELinux.
policy	-r--r--r--	Interface to upload the current running policy in kernel binary format. This is useful to check the running policy using apol (1) , dispol/sedispol etc. (e.g. <code>cat /sys/fs/selinux/policy > current-policy</code> then load it into the required tool).
policyvers	-r--r--r--	Returns supported policy version for kernel. Read by security_policyvers (3) function.
relabel	-rw-rw-rw-	Compute relabeling decision interface that is used by the security_compute_relabel (3) function. The kernel security server (see <code>services.c</code>) converts the contexts to SIDs and then calls the <code>security_change_sid</code> function to compute the new SID that is then converted to a context string. Requires <code>security {compute_relabel}</code> permission.
status	-r--r--r--	This can be used to obtain enforcing mode and policy load changes with much less over-head than using the <code>libselinux</code> netlink / call backs. This was added for Object Managers that have high volumes of AVC requests so they can quickly check whether to invalidate their cache or not. The status structure indicates the following: <ul style="list-style-type: none"> <code>version</code> - Version number of the status structure. This will increase as other entries are added. <code>sequence</code> - This is incremented for each event with an even number meaning that the events are stable. An odd number indicates that one of the events is changing and therefore the userspace application should wait before reading the status of any event. <code>enforcing</code> - 0 = Permissive mode, 1 = enforcing mode. <code>policyload</code> - This contains the policy load sequence number and should be read and stored, then compared to detect a policy reload. <code>deny_unknown</code> - 0 = Allow and 1 = Deny unknown object classes / permissions. This is the same as the <code>deny_unknown</code> entry above.

The SELinux Notebook

<i>selinuxfs Directory and File Names</i>	<i>Permissions</i>	<i>Comments</i>
user	-rw-rw-rw-	<p>Compute reachable user contexts interface that is used by the security_compute_user (3) function.</p> <p>The kernel security server (see <code>services.c</code>) converts the contexts to SIDs and then calls the <code>security_get_user_sids</code> function to compute the user SIDs that are then converted to context strings.</p> <p>Requires <code>security {compute_user}</code> permission.</p>
/sys/fs/selinux/avc	Directory	This directory contains information regarding the kernel AVC that can be displayed by the <code>avcstat</code> command.
cache_stats	-r--r--r--	Shows the kernel AVC lookups, hits, misses etc.
cache_threshold	-rw-r--r--	<p>The default value is 512, however caching can be turned off (but performance suffers) by:</p> <pre>echo 0 > /selinux/avc/cache_threshold</pre> <p>Requires <code>security {setseccap}</code> permission.</p>
hash_stats	-r--r--r--	Shows the number of kernel AVC entries, longest chain etc.
/sys/fs/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 and pending status of the boolean (0 = false or 1 = true). The getsebool (8), setsebool (8) and sestatus (8) -b commands use this interface via the <code>libselinux</code> library functions.
/sys/fs/selinux/initial_contexts	Directory	This directory contains one file for each initial SID defined in the active policy. The file name is the initial SID name with the contents containing its security context.
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>).
/sys/fs/selinux/policy_capabilities	Directory	This directory contains the policy capabilities that have been configured by default in the kernel via the <code>polycap</code> statement in the active policy. These are generally new features that can be enabled by using the <code>polycap</code> statement in policy. Their default values are false.
always_check_network	-r--r--r--	If true SECMARK and peer labeling are always enabled even if there are no SECMARK, NetLabel or Labeled IPsec rules configured. This forces checking of the <code>packet</code> class to protect the system should any rules fail to load or they get maliciously flushed. Requires kernel 3.14 minimum.

<i>selinuxfs Directory and File Names</i>	<i>Permissions</i>	<i>Comments</i>
network_peer_controls	-r--r--r--	If true the following network_peer_controls are enabled: node: sendto recvfrom netif: ingress egress peer: recv
open_perms	-r--r--r--	If true the open permissions are enabled by default on the following object classes: dir, file, fifo_file, chr_file, blk_file.
redhat1	-r--r--r--	Available in kernel 3.4 to allow finer control of ptrace (this will be named correctly one day). Requires policy support and the security class permission ptrace_child.
/sys/fs/selinux/class	Directory	This directory contains a list of classes and their permissions as defined by the policy (for the Reference Policy the order in the security_classes and access_vectors files).
/sys/fs/selinux/class/appletalk_socket	Directory	Each class has its own directory where each one is named using the appropriate class statement from the policy (i.e. class appletalk_socket). Each directory contains the following:
index	-r--r--r--	This file contains the allocated class number (e.g. appletalk_socket is the 56 th entry in the policy security_classes file).
/sys/fs/selinux/class/appletalk_socket/perms	Directory	This directory contains one file for each permission defined in the policy.
accept append bind	-r--r--r--	Each file is named by the permission assigned in the policy and contains a number that represents its position in the list (e.g. accept is the 14 th permission listed in the policy access_vector file for the appletalk_socket and therefore contains '14').

Table 10: selinux filesystem Information

Notes:

1. Kernel SIDs are not passed to userspace only the context strings.
2. The /proc filesystem exports the process security context string to userspace via /proc/<self|pid>/attr and /proc/<self|pid>/task/<tid>/attr/<attr> interfaces.

2.20 libselinux Library

`libselinux` contains all the SELinux functions necessary to build userspace SELinux-aware applications and object managers using 'C', Python, Ruby and PHP languages.

The library hides the low level functionality of (but not limited to):

- The SELinux filesystem that interfaces to the SELinux kernel security server.
- The `proc` filesystem that maintains process state information and security contexts - see `proc(5)`.
- Extended attribute services that manage the extended attributes associated to files, sockets etc. - see `attr(5)`.
- The SELinux policy and its associated configuration files.

The general category of functions available in `libselinux` are shown in Table 11, with [Appendix B](#) giving a complete list of functions.

Function Category	Description
Access Vector Cache Services	Allow access decisions to be cached and audited.
Boolean Services	Manage booleans.
Class and Permission Management	Class / permission string conversion and mapping.
Compute Access Decisions	Determine if access is allowed or denied.
Compute Labeling	Compute labels to be applied to new instances of on object.
Default File Labeling	Obtain default contexts for file operations.
File Creation Labeling	Get and set file creation contexts.
File Labeling	Get and set file and file descriptor extended attributes.
General Context Management	Check contexts are valid, get and set context components.
Key Creation Labeling	Get and set kernel key creation contexts.
Label Translation Management	Translate to/from, raw/readable contexts.
Netlink Services	Used to detect policy reloads and enforcement changes.
Process Labeling	Get and set process contexts.
SELinux Management Services	Load policy, set enforcement mode, obtain SELinux configuration information.
SELinux-aware Application Labeling	Retrieve default contexts for applications such as database and X-Windows.
Socket Creation Labeling	Get and set socket creation contexts.

User Session Management	Retrieve default contexts for user sessions.
-------------------------	--

Table 11: libselinux function types

Other SELinux userspace libraries are:

`libsepol` - To build and manipulate the contents of SELinux kernel binary policy files.

`libsemanage` - To manage the policy infrastructure.

Details of the libraries, core SELinux utilities and commands with source code are available at:

<https://github.com/SELinuxProject/selinux/wiki>

The versions of kernel and SELinux tools and libraries influence the features available, therefore it is important to establish what level of functionality is required for the application. The [Policy Versions](#) section shows the policy versions and the additional features they support.

Writing kernel based object managers is a more specialised subject and is not covered in this section.

The `libselinux` functions make use of a number of files within the SELinux sub-system:

1. The SELinux configuration file `config` that is described in the [/etc/selinux/config File](#) section.
2. The SELinux filesystem interface between userspace and kernel that is generally mounted as `/selinux` or `/sys/fs/selinux` and described in the [SELinux Filesystem](#) section.
3. The `proc` filesystem that maintains process state information and security contexts - see `proc(5)`.
4. The extended attribute services that manage the extended attributes associated to files, sockets etc. - see `attr(5)`.
5. The SELinux kernel binary policy that describes the enforcement policy.
6. A number of `libselinux` functions have their own configuration files that in conjunction with the policy, allow additional levels of configuration. These are described in the [Policy Configuration Files](#) section and also in the following man pages:

```
booleans(5), customizable_types(5),  
default_contexts(5), default_type(5),  
failsafe_context(5), file_contexts(5),  
local.users(5), media(5), removable_context(5),  
securetty_type(5), selabel_db(5), selabel_file(5),  
selabel_media(5), selabel_x(5), sepgsql_contexts(5),  
service_seusers(5), seusers(5), user_contexts(5),  
virtual_domain_context(5),  
virtual_image_context(5), x_contexts(5)
```

2.21 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 two types supported: SECMARK and NetLabel. There was a service known as 'compat_net' controls, however that was removed in kernel 2.6.30.

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. There are two types supported: Labeled IPsec and CIPSO (Commercial IP Security Option).

There are two policy capability options that can be set within policy using the `policycap` statement that affect networking configuration:

network_peer_controls - This is always enabled in the latest Reference Policy source. [Figure 2.14](#) shows the differences between the policy capability being set to 0 and 1.

always_use_network - This capability would normally be set to false. If true SECMARK and NetLabel peer labeling are always enabled even if there are no SECMARK, NetLabel or Labeled IPsec rules configured. This forces checking of the `packet` class to protect the system should any rules fail to load or they get maliciously flushed. Requires kernel 3.13 minimum.

The policy capability settings are available in userspace via the SELinux filesystem as shown in [Table 10](#).

To support peer labeling and CIPSO the NetLabel tools need to be installed:

```
yum install netlabel_tools
```

To support Labeled IPsec the IPsec tools need to be installed:

```
yum install ipsec-tools
```

It is also possible to use an alternative Labeled IPsec service that was OpenSwan but is now distributed as LibreSwan:

```
yum install libreswan
```

It is important to note that the kernel must be configured to support these services. The F-20 kernels are configured to handle all the above services.

The Linux networking package `iproute` has an SELinux aware socket statistics command `ss` (8) that will show the SELinux context of network processes (`-Z` or `--context` option) and network sockets (`-z` or `--contexts` option). Although note that the socket contexts are taken from the inode associated to the socket and not from the actual kernel socket structure (as currently there is no standard kernel/userspace interface to achieve this).

2.21.1 SECMARK

SECMARK makes use of the standard kernel NetFilter framework that underpins the GNU / Linux IP networking sub-system. NetFilter services automatically inspect 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 security extensions allow security contexts to be added to packets (SECMARK) or sessions (CONNSECMARK).

The NetFilter framework inspects and tag packets with labels as defined within **iptables** (8) and then uses the security framework (e.g. SELinux) to enforce the policy rules. Therefore SECMARK services are not SELinux specific as other security modules using 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 available¹⁶. [Figure 2.13](#) shows the basic structure with the process working as follows:

- A table called the 'security table' is used to define the parameters that identify and 'mark' packets that can then be 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 security table applying a label that can then be used to enforce policy on the packet.
- The CONNSECMARK 'marks' all packets within a session¹⁷ with the appropriate label that can then be used to enforce policy.

¹⁶ There is a very good tutorial at <http://www.frozentux.net/documents/iptables-tutorial/> [5], however it does not cover the security table that was introduced by: <http://lwn.net/Articles/267140/>. It is still possible to use the 'mangle table' to hold security labels as described in [5].

¹⁷ 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.

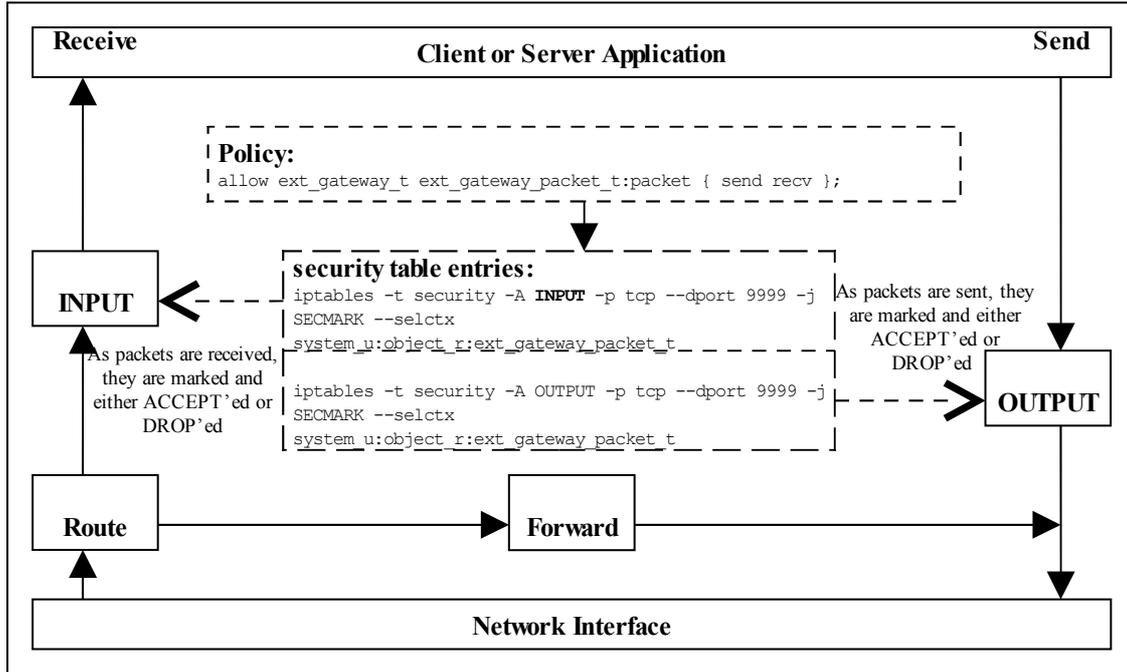


Figure 2.13: SECMARK Processing - Received packets are processed by the INPUT chain where labels are added to the appropriate packets that will either be accepted or dropped by the SECMARK process. Packets being sent are treated the same way.

An example iptables¹⁸ 'security table' entry is as follows:

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

#----- INPUT IP Stream -----#

# This INPUT rule sets all packets to msg_filter.default_packet: as it is
# executed first:
iptables -t security -A INPUT -i lo -p tcp -d 127.0.0.0/8 -j SECMARK --selctx
system.user:object_r:msg_filter.default_packet:s0

# These rules 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 security -A INPUT -i lo -p tcp --dport 9999 -j SECMARK --selctx
system.user:object_r:msg_filter.ext_gateway.packet:s0
iptables -t security -A INPUT -i lo -p tcp --sport 9999 -j SECMARK --selctx
system.user:object_r:msg_filter.ext_gateway.packet:s0
#
# The internal gateway:
iptables -t security -A INPUT -i lo -p tcp --dport 1111 -j SECMARK --selctx
system.user:object_r:msg_filter.int_gateway.packet:s0
iptables -t security -A INPUT -i lo -p tcp --sport 1111 -j SECMARK --selctx
system.user:object_r:msg_filter.int_gateway.packet:s0

iptables -t security -A INPUT -m state --state ESTABLISHED,RELATED -j
CONNSECMARK --save

#----- OUTPUT IP Stream -----#

# This OUTPUT rule sets all packets to msg_filter.default_packet: as it is
# executed first:
```

¹⁸ The tables will not load correctly if the policy does not allow the iptables domain to relabel the security table entries unless permissive mode is enabled (i.e. iptables must have the relabel permission for each entry in the table).

```

iptables -t security -A OUTPUT -o lo -p tcp -d 127.0.0.0/8 -j SECMARK --selctx
system.user:object_r:msg_filter.default_packet:s0

# These rules 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 security -A OUTPUT -o lo -p tcp --dport 9999 -j SECMARK --selctx
system.user:object_r:msg_filter.ext_gateway.packet:s0
iptables -t security -A OUTPUT -o lo -p tcp --sport 9999 -j SECMARK --selctx
system.user:object_r:msg_filter.ext_gateway.packet:s0
#
# The internal gateway:
iptables -t security -A OUTPUT -o lo -p tcp --dport 1111 -j SECMARK --selctx
system.user:object_r:msg_filter.int_gateway.packet:s0
iptables -t security -A OUTPUT -o lo -p tcp --sport 1111 -j SECMARK --selctx
system.user:object_r:msg_filter.int_gateway.packet:s0

iptables -t security -A OUTPUT -m state --state ESTABLISHED,RELATED -j
CONNSECMARK --save
    
```

An example policy that makes use of SECMARK services is described in the Notebook source tarball. There are also articles "[Transitioning to Secmark](#)" [7] and "[New secmark-based network controls for SELinux](#)" [6] that explain the services.

2.21.2 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](#)" [8] that explains their usage, the `netlabelctl` (8) man page is also a useful reference.

The example message filter has an optional module that makes use of fallback labels and can be found in the Notebook source tarball.

The network peer controls have been extended to support an additional object class of 'peer' that is enabled by default in the F-20 policy as the `network_peer_controls` in `/sys/fs/selinux/policy_capabilities` is set to '1'. [Figure 2.14](#) shows the differences between the policy capability `network_peer_controls` being set to 0 and 1.

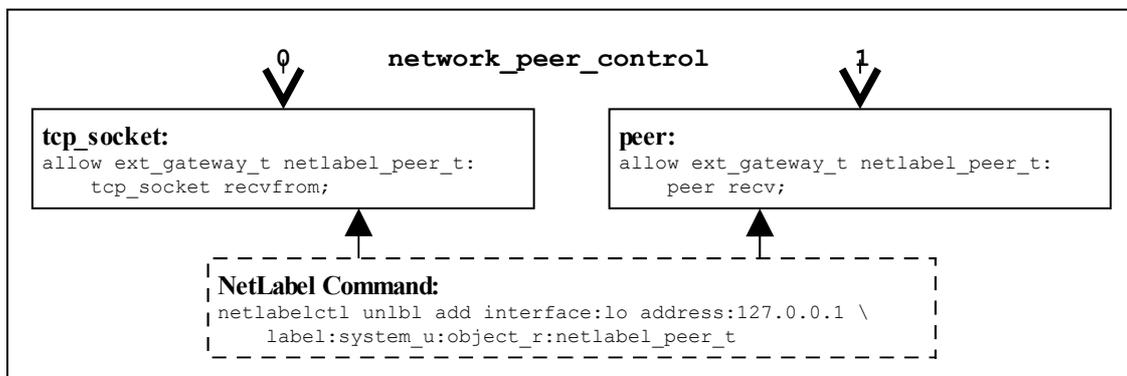


Figure 2.14: Fallback Labeling - Showing the differences between the policy capability `network_peer_controls` set to 0 and 1.

2.21.3 NetLabel - CIPSO

To allow [security levels](#) to be passed over a network between MLS systems¹⁹, the CIPSO protocol is used. This 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.

Note that only the level component of the security context is passed over the network. The exception is in loopback mode as explained in "[Full SELinux Labels Over Loopback with NetLabel and CIPSO](#)" available at <http://paulmoore.livejournal.com/7234.html>.

The protocol is implemented by the NetLabel service (see `netlabelctl(8)`) 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 2.15](#)).

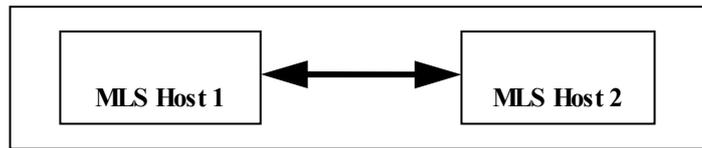


Figure 2.15: MLS Systems on the same network

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 2.16](#)).

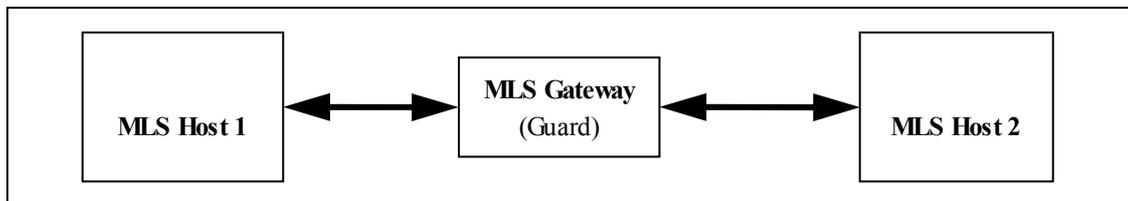


Figure 2.16: MLS Systems on different networks communicating via a gateway

2.21.4 Labeled IPsec

Labeled IPsec has been built into the standard GNU / Linux IPsec services as described in the "[Leveraging IPsec for Distributed Authorization](#)" [9]. [Figure 2.17](#) shows the basic components that form the service based on IPsec tools where it is generally used to set up either an encrypted tunnel between two machines²⁰ or an encrypted transport session. The extensions defined in [9] describe how the security context is configured and negotiated between the two systems (called security associations (SAs) in IPsec terminology).

¹⁹ Note only the security levels are passed over the network as the other components of the security context are not part of standard MLS systems (as it may be that the remote end is a Trusted Solaris system).

²⁰ Also known as a virtual private network (VPN).

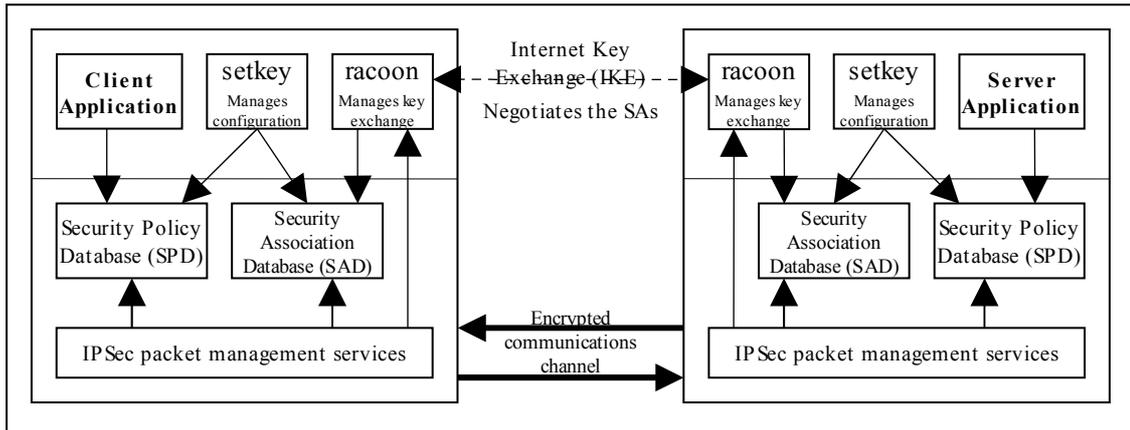


Figure 2.17: IPsec communications - *The SPD contains information regarding the security contexts to be used. These are exchanged between the two systems as part of the channel set-up.*

Basically what happens is as follows²¹:

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 with an example shown in the [Configuration Example](#) section.
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** or **pluto**²² 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 when two systems are communicating (using the above example), one system will have an SA, SA_{out} for processing outbound packets and another SA, SA_{in}, for processing the inbound packets. The other system will also create two SAs for processing its packets.

Each SA will share the same cryptographic parameters such as keys and protocol²³ (e.g. ESP (encapsulated security payload) and AH (authentication header)).

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

²¹ There is an "IPsec HOWTO" [10] at <http://www.ipsec-howto.org> that gives the gory details, however it does not cover Labeled IPsec.

²² These are the Internet Key Exchange (IKE) daemons that exchange encryption keys securely and also supports Labeled IPsec parameter exchanges.

²³ The GNU / Linux version supports a number of secure protocols, see **setkey** (8) for details.

<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/pluto</code> will need this permission to build the SAD).

When running Labeled IPsec it is recommended that the systems use the same type/version of policy to avoid any problems with them having different meanings.

There are worked examples of Labeled IPsec sessions showing manual configuration using `setkey` and IKE exchanges using `racoon`²⁴ and LibreSwan (`pluto`) configurations in the Notebook source tarball (note that the LibreSwan examples use the kernel `netkey` services).

There is a further example in the "[Secure Networking with SELinux](#)" [11] article.

There is a good reference covering "Basic Labeled IPsec Configuration" available at:

<http://www.redhat.com/archives/redhat-lspp/2006-November/msg00051.html>

2.21.4.1 Configuration Examples

There are two possible labeled IPsec solutions available:

IPsec Tools - This uses the `setkey` (8) tools and `racoon` (8) Internet Key Exchange (IKE) daemon.

LibreSwan - This uses `ipsec` (8) tools and `pluto` (8) Internet Key Exchange (IKE) daemon.

Both work in much the same way but use different configuration files with samples shown below. The one point they have in common is that to start any session for label exchange using IKE, `setkey` must be used to initially set up the labels in the security policy database (SPD) on each machine.

Another point to note is that if interoperating between `racoon` and `pluto` the IPSEC Security Association Attribute values are different:

- `racoon` has this hard-wired to a value of '10'.
- `pluto` is configurable with a default of '32001'. To interoperate with `racoon` the `ipsec.conf` (5) file must have:

```
config setup
    secctx_attr_value = 10
```

The following example configurations show the common `setkey` configuration to set up the SPD entries and then a sample supporting `racoon` and `pluto` (LibreSwan) configuration file:

²⁴ Unfortunately `racoon` core dumps when using non MCS/MLS policies.

Add label / context to SPD for loopback:

```
# setkey -f configuration file entries for RACoon 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 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 "unconfined.user:msg_filter.role:msg_filter.ext_gateway.process:s0"
-P out ipsec esp/transport//require;

spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "unconfined.user:msg_filter.role:msg_filter.ext_gateway.process:s0"
-P in ipsec esp/transport//require;

# Security policies for internal gateway:
spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "unconfined.user:msg_filter.role:msg_filter.int_gateway.process:s0"
-P out ipsec esp/transport//require;

spdadd 127.0.0.1 127.0.0.1 tcp
-ctx 1 1 "unconfined.user:msg_filter.role:msg_filter.int_gateway.process:s0"
-P in ipsec esp/transport//require;
```

racoon configuration:

```
# 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 ;
}
```

LibreSwan / pluto loopback configuration:

```
# /etc/ipsec.conf - Libreswan IPsec configuration file

version 2.0

config setup
    plutorestartoncrash=false
    protostack=netkey
    plutodebug="all"
    # A "secctx_attr_value" is optional for >= 3.6 as defaults to this:
    secctx_attr_value = 32001

conn labeled_loopback_test
    auto=start
    rekey=no
    authby=secret
    type=transport
```

```
left=127.0.0.1
right=127.0.0.1
ike=3des-sha1
phase2=esp
phase2alg=aes-sha1
loopback=yes
labeled_ipsec=yes
policy_label=unconfined.user:msg_filter.role:msg_filter.ext_gateway.process:s0
leftprotoport=tcp
rightprotoport=tcp
```

2.22 SELinux Virtual Machine Support

SELinux support is available in the KVM/QEMU and Xen virtual machine (VM) technologies²⁵ that are discussed in the sections that follow, however the package documentation should be read for how these products actually work and how they are configured.

Currently the main SELinux support for virtualisation is via `libvirt` that is an open-source virtualisation API used to dynamically load guest VMs. Security extensions were added as a part of the [Svirt](#) project and the SELinux implementation for the KVM/QEMU package (`qemu-kvm` and `libvirt` rpms) is discussed using some examples. The Xen product has Flask/TE services that can be built as an optional service, although it can also use the security enhanced `libvirt` services as well.

The sections that follow give an introduction to KVM/QEMU, then `libvirt` support with some examples using the Virtual Machine Manager to configure VMs, then an overview of the Xen implementation follows.

To ensure all dependencies are installed run:

```
yum install libvirt
yum install qemu
yum install virt-manager
```

2.22.1 KVM / QEMU Support

KVM is a kernel loadable module that uses the Linux kernel as a hypervisor and makes use of a modified QEMU emulator to support the hardware I/O emulation. The "[Kernel-based Virtual Machine](#)" [17] document gives a good overview of how KVM and QEMU are implemented. It also provides an introduction to virtualisation in general. Note that KVM requires virtualisation support in the CPU (Intel-VT or AMD-V extensions).

The SELinux support for VMs is implemented by the `libvirt` sub-system that is used to manage the VM images using a Virtual Machine Manager, and as KVM is based on Linux it has SELinux support by default. There are also Reference Policy modules to support the overall infrastructure (KVM support is in various kernel and system modules with a `virt` module supporting the `libvirt` services). [Figure 2.18](#)

²⁵ KVM (Kernel-based Virtual Machine) and Xen are classed as 'bare metal' hypervisors and they rely on other services to manage the overall VM environment. QEMU (Quick Emulator) is an emulator that emulates the BIOS and I/O device functionality and can be used standalone or with KVM and Xen.

shows a high level overview with two VMs running in their own domains. The [libvirt Support](#) section shows how to configure these and their VM image files.

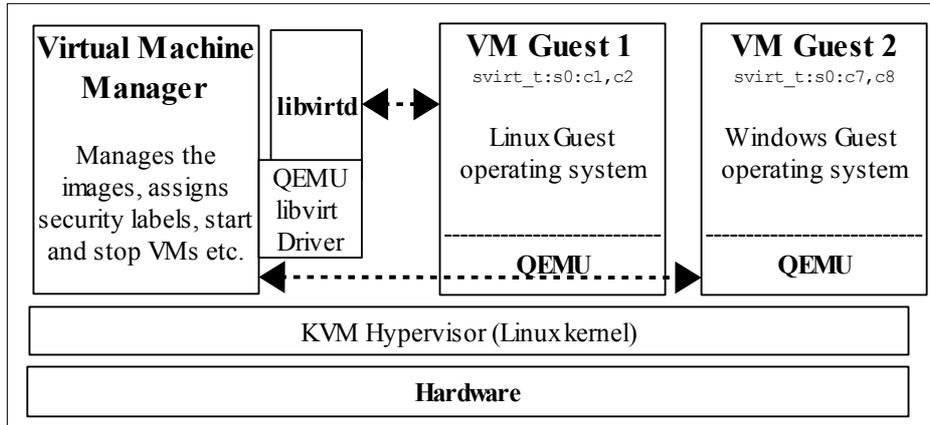


Figure 2.18: KVM Environment - KVM provides the hypervisor while QEMU provides the hardware emulation services for the guest operating systems. Note that KVM requires CPU virtualisation support.

2.22.2 libvirt Support

The Svirt project added security hooks into the libvirt library that is used by the libvirtd daemon. This daemon is used by a number of VM products (such as KVM, QEMU and Xen) to start their VMs running as guest operating systems.

The VM supplier can implement any security mechanism they require using a product specific libvirt [driver](#) that will load and manage the images. The SELinux implementation supports four methods of labeling VM images, processes and their resources with support from the Reference Policy modules/services/virt.* loadable module²⁶. To support this labeling, libvirt requires an MCS or MLS enabled policy as the [level](#) entry of the security context is used (user:role:type:level).

The link <http://libvirt.org/drvqemu.html#securityselinux> has details regarding the QEMU driver and the SELinux confinement modes it supports.

2.22.3 VM Image Labeling

This sections assumes VM images have been generated using the simple Linux kernel available at: <http://wiki.qemu.org/Testing> (the linux-0.2.img.bz2 disk image), this image was renamed to reflect each test, for example 'Dynamic_VM1.img'.

These images can be generated using the VMM by selecting the 'Create a new virtual machine' menu, 'importing existing disk image' then in step 2 Browse... selecting 'Choose Volume: Dynamic_VM1.img' with OS type: Linux, Version: Generic 2.6.x kernel and change step 4 'Name' to Dynamic_VM1.

²⁶ The various images would have been labeled by the virt module installation process (see the virt.fc module file or the policy file_contexts file libvirt entries). If not, then need to ensure it is relabeled by the most appropriate SELinux tool.

2.22.3.1 Dynamic Labeling

The default mode is where each VM is run under its own dynamically configured domain and image file therefore isolating the VMs from each other (i.e. every time the VM is run a different and unique MCS label will be generated to confine each VM to its own domain). This mode is implemented as follows:

- a) An initial context for the process is obtained from the `/etc/selinux/<SELINUXTYPE>/contexts/virtual_domain_context` file (the default is `system_u:system_r:svirt_tcg_t:s0`).
- b) An initial context for the image file label is obtained from the `/etc/selinux/<SELINUXTYPE>/contexts/virtual_image_context` file. The default is `system_u:system_r:svirt_image_t:s0` that allows read/write of image files.
- c) When the image is used to start the VM, a random MCS level is generated and added to the process context and the image file context. The process and image files are then transitioned to the context by the `libselinux` API calls `setfilecon` and `setexeccon` respectively (see `security_selinux.c` in the `libvirt` source). The following example shows two running VM sessions each having different labels:

VM Name	Object	Dynamically assigned security context
Dynamic_VM1	Process	<code>system_u:system_r:svirt_tcg_t:s0:c585,c813</code>
	File	<code>system_u:system_r:svirt_image_t:s0:c585,c813</code>
Dynamic_VM2	Process	<code>system_u:system_r:svirt_tcg_t:s0:c535,c601</code>
	File	<code>system_u:system_r:svirt_image_t:s0:c535,c601</code>

The running image `ls -Z` and `ps -eZ` are as follows, and for completeness an `ls -Z` is shown when both VMs have been stopped:

```
# Both VMs running:
ls -Z /var/lib/libvirt/images
system_u:object_r:svirt_image_t:s0:c585,c813 Dynamic_VM1.img
system_u:object_r:svirt_image_t:s0:c535,c601 Dynamic_VM2.img

ps -eZ | grep qemu
system_u:system_r:svirt_tcg_t:s0:c585,c813 8707 ? 00:00:44 qemu-system-x86
system_u:system_r:svirt_tcg_t:s0:c535,c601 8796 ? 00:00:37 qemu-system-x86

# Both VMs stopped (note that the categories are now missing AND
# the type has changed from svirt_image_t to virt_image_t):
ls -Z /var/lib/libvirt/images
system_u:object_r:virt_image_t:s0 Dynamic_VM1.img
system_u:object_r:virt_image_t:s0 Dynamic_VM2.img
```

2.22.3.2 Shared Image

If the disk image has been set to shared, then a dynamically allocated `level` will be generated for each VM process instance, however there will be a single instance of the disk image.

The Virtual Machine Manager can be used to set the image as shareable by checking the `Shareable` box as shown in [Figure 2.19](#).

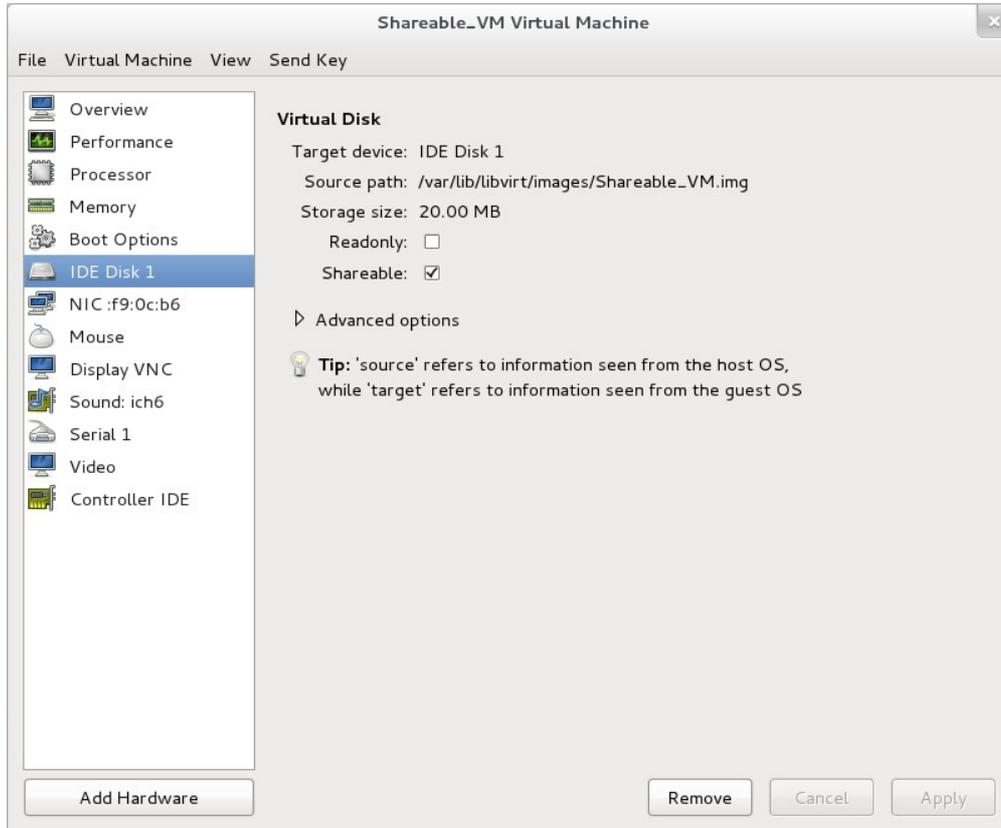
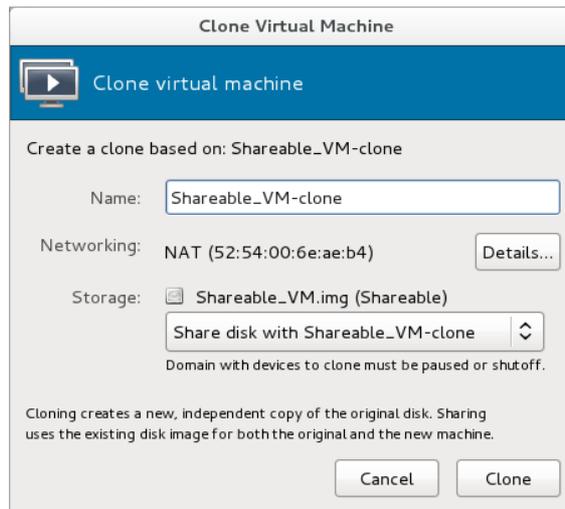


Figure 2.19: Setting the Virtual Disk as Shareable

This will set the image (`Shareable_VM.xml`) resource XML configuration file located in the `/etc/libvirt/qemu` directory `<disk>` contents as follows:

```
# /etc/libvirt/qemu/Shareable_VM.xml:
<disk type='file' device='disk'>
  <driver name='qemu' type='raw'/>
  <source file='/var/lib/libvirt/images/Shareable_VM.img'/>
  <target dev='hda' bus='ide'/>
  <shareable/>
  <address type='drive' controller='0' bus='0' unit='0'/>
</disk>
```

As the two VMs will share the same image, the `Shareable_VM` service needs to be cloned and the VM resource name selected was `Shareable_VM-clone`.



The resource XML file `<disk>` contents generated are shown - note that it has the same source file name as the `Shareable_VM.xml` above.

```
# /etc/libvirt/qemu/Shareable_VM-clone.xml:
<disk type='file' device='disk'>
  <driver name='qemu' type='raw'/>
  <source file='/var/lib/libvirt/images/Shareable_VM.img'/>
  <target dev='hda' bus='ide'/>
  <shareable/>
  <address type='drive' controller='0' bus='0' unit='0'/>
</disk>
```

With the targeted policy on F-20 the shareable option gave a error when the VMs were run as follows:

```
Could not allocate dynamic translator buffer
```

The audit log contained the following AVC message:

```
type=AVC msg=audit(1326028680.405:367): avc: denied
{ execmem } for pid=5404 comm="qemu-system-x86"
scontext=system_u:system_r:svirt_t:s0:c121,c746
tcontext=system_u:system_r:svirt_t:s0:c121,c746 tclass=process
```

To overcome this error, the following boolean needs to be enabled with `setsebool` (8) to allow access to shared memory (the `-P` option will set the boolean across reboots):

```
setsebool -P virt_use_execmem on
```

Now that the image has been configured as shareable, the following initialisation process will take place:

- a) An initial context for the process is obtained from the `/etc/selinux/<SELINUXTYPE>/contexts/virtual_domain_context` file (the default is `system_u:system_r:svirt_tcg_t:s0`).
- b) An initial context for the image file label is obtained from the `/etc/selinux/<SELINUXTYPE>/contexts/virtual_image_context` file.

The default is `system_u:system_r:svirt_image_t:s0` that allows read/write of image files.

- c) When the image is used to start the VM a random MCS level is generated and added to the process context (but not the image file). The process is then transitioned to the appropriate context by the `libselinux` API calls `setfilecon` and `setexeccon` respectively. The following example shows each VM having the same file label but different process labels:

VM Name	Object	Security context
Shareable_VM	Process	<code>system_u:system_r:svirt_tcg_t:s0:c231,c245</code>
Shareable_VM -clone	Process	<code>system_u:system_r:svirt_tcg_t:s0:c695,c894</code>
	File	<code>system_u:system_r:svirt_image_t:s0</code>

The running image `ls -Z` and `ps -eZ` are as follows and for completeness an `ls -Z` is shown when both VMs have been stopped:

```
# Both VMs running and sharing same image:
ls -Z /var/lib/libvirt/images
system_u:object_r:svirt_image_t:s0 Shareable_VM.img

# but with separate processes:
ps -eZ | grep qemu
system_u:system_r:svirt_t:s0:c231,c254 6748 ? 00:01:17 qemu-system-x86
system_u:system_r:svirt_t:s0:c695,c894 7664 ? 00:00:03 qemu-system-x86

# Both VMs stopped (note that the type has remained as svirt_image_t)
ls -Z /var/lib/libvirt/images
system_u:object_r:svirt_image_t:s0 Shareable_VM.img
```

2.22.3.3 Static Labeling

It is possible to set static labels on each image file, however a consequence of this is that the image cannot be cloned using the VMM, therefore an image for each VM will be required. This is the method used to configure VMs on MLS systems as there is a known label that would define the security level. With this method it is also possible to configure two or more VMs with the same security context so that they can share resources. A useful reference is at: <http://libvirt.org/formatdomain.html#seclabel>.

If using the Virtual Machine Manager GUI, then by default it will start each VM running as they are built, therefore they need to be stopped and restarted once configured for static labels, the image file will also need to be relabeled. An example VM configuration follows where the VM has been created as `Static_VM1` using the F-20 targeted policy in enforcing mode (just so all errors are flagged during the build):

- a) To set the required security context requires editing the `Static_VM1` configuration file using `virsh` (1) as follows:

```
virsh edit Static_VM1
```

Then add the following at the end of the file:

```
....
</devices>

<!-- The <seclabel> tag needs to be placed btween the existing
      </devices> and </domain> tags -->

<seclabel type='static' model='selinux' relabel='no'>
  <label>system_u:system_r:svirt_t:s0:c1022,c1023</label>
</seclabel>

</domain>
```

For this example `svirt_t` has been chosen as it is a valid context (however it will not run as explained in the text). This context will be written to the `Static_VM1.xml` configuration file in `/etc/libvirt/qemu`.

b) If the VM is now started an error will be shown as follows:

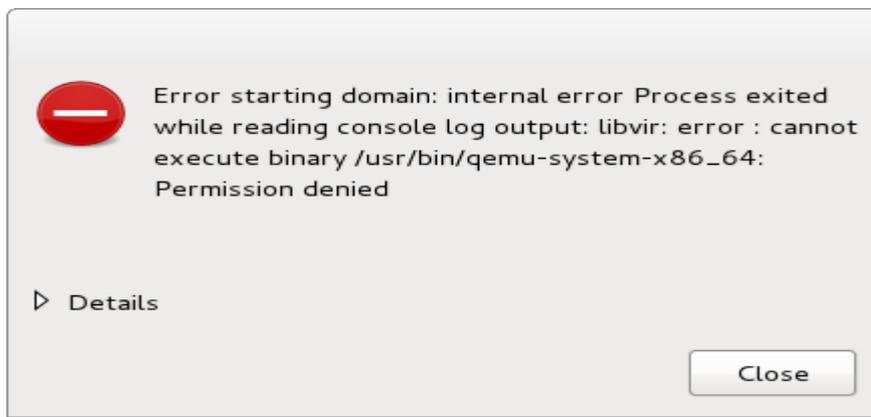


Figure 2.20: Image Start Error

This is because the image file label is incorrect as by default it is labeled `virt_image_t` when the VM image is built (and `svirt_t` does not have read/write permission for this label):

```
# The default label of the image at build time:
system_u:object_r:virt_image_t:s0 Static_VM1.img
```

There are a number of ways to fix this, such as adding an allow rule or changing the image file label. In this example the image file label will be changed using `chcon` (1) as follows:

```
# This command is executed from /var/lib/libvirt/images
#
# This sets the correct type:
chcon -t svirt_image_t Static_VM1.img
```

Optionally, the image can also be relabeled so that the `[level]` is the same as the process using `chcon` as follows:

```
# This command is executed from /var/lib/libvirt/images
#
# Set the MCS label to match the process (optional step):
chcon -l s0:c1022,c1023 Static_VM1.img
```

c) Now that the image has been relabeled, the VM can now be started.

The following example shows two static VMs (one is configured for `unconfined_t` that is allowed to run under the targeted policy - this was possible because the `'setsebool -P virt_transition_userdomain on'` boolean was set that allows `virt_d_t` domain to transition to a user domain (e.g. `unconfined_t`).

VM Name	Object	Static security context
Static_VM1	Process	system_u:system_r:svirt_t:s0:c1022,c1023
	File	system_u:system_r:svirt_image_t:s0:c1022,c1023
Static_VM2	Process	system_u:system_r:unconfined_t:s0:c11,c22
	File	system_u:system_r:virt_image_t:s0

The running image `ls -Z` and `ps -eZ` are as follows, and for completeness an `ls -Z` is shown when both VMs have been stopped:

```
# Both VMs running (Note that Static_VM2 did not have file level reset):
ls -Z /var/lib/libvirt/images
system_u:object_r:svirt_image_t:s0:c1022,c1023 Static_VM1.img
system_u:object_r:virt_image_t:s0 Static_VM2.img

ps -eZ | grep qemu
system_u:system_r:svirt_t:s0:c585,c813 6707 ? 00:00:45 qemu-system-x86
system_u:system_r:unconfined_t:s0:c11,c22 6796 ? 00:00:26 qemu-system-x86

# Both VMs stopped (note that Static_VM1.img was relabeled svirt_image_t
# to enable it to run, however Static_VM2.img is still labeled
# virt_image_t and runs okay. This is because the process is run as
# unconfined_t that is allowed to use virt_image_t):
system_u:object_r:svirt_image_t:s0:c1022,c1023 Static_VM1.img
system_u:object_r:virt_image_t:s0 Static_VM2.img
```

2.22.4 Xen Support

This is not supported by SELinux in the usual way as it is built into the actual Xen software as a 'Flask/TE' extension²⁷ for the XSM (Xen Security Module). Also the Xen implementation has its own built-in policy (`xen.te`) and supporting definitions for access vectors, security classes and initial SIDs for the policy. These Flask/TE components run in Domain 0 as part of the domain management and control supporting the Virtual Machine Monitor (VMM) as shown in [Figure 2.21](#).

²⁷ This is a version of the SELinux security server, `avc` etc. that has been specifically ported for the Xen implementation.

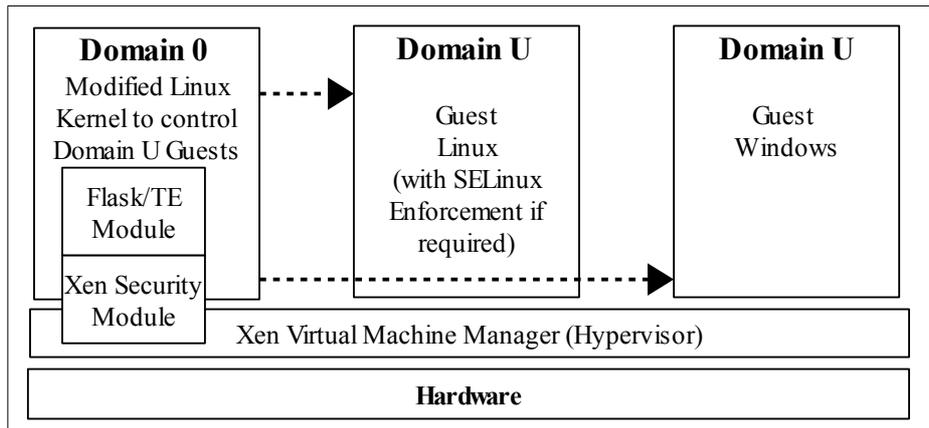


Figure 2.21: Xen Hypervisor - Using XSM and Flask/TE to enforce policy on the physical I/O resources.

The "[How Does Xen Work](#)" [18] document describes the basic operation of Xen, the "[Xen Security Modules](#)" [19] describes the XSM/Flask implementation, and the `xsm-flask.txt` file in the Xen source package describes how SELinux and its supporting policy is implemented.

However (just to confuse the issue), there is another Xen policy module (also called `xen.te`) in the Reference Policy to support the management of images etc. via the Xen console.

For reference, the Xen policy supports additional policy language statements: `iomemcon`, `ioportcon`, `pcidevicecon` and `pirqcon` that are discussed in the [Xen](#) section of [SELinux Policy Language](#).

2.23 Sandbox Services

Fedora has support for three types of sandbox services in F-20:

1. Non-GUI sandboxing (`sandbox` - see <http://danwalsh.livejournal.com/28545.html>).

There is also a good use-case with solutions at: <http://opensource.com/education/12/8/harvard-goes-paas-selinux-sandbox> that involves uploading information to web servers and access by staff and students.

2. GUI sandboxing using the Xephyr server (`sandbox-X` - see <http://danwalsh.livejournal.com/31146.html>).

This will allow isolation of X applications via nested Xephyr servers. For example running:

```
sandbox -t sandbox_web_t -i /path/to/user/home/dir/.mozilla -W metacity -X firefox
```

will load Firefox in an isolated X sandbox. The `-i` parameter stops Firefox displaying the 'welcome to Firefox' page at start-up as it will use a copy from the users current `.mozilla` directory.

Red Hat use `sandbox-X` as the preferred alternative to XSELinux when using the `targeted` policy, this is because X-clients that get a permission denied will probably abort as they expect full access to the X-server.

Both of these sandbox services are defined in the `sandbox(3)` man page and are available in the `policycoreutils` package. They make use of `seunshare(8)` that allows commands to be run in an alternate home directory, temp directory or security context. The `sandbox.conf(5)` file allows the sandbox name, cpu and memory usage to be configured. There is also a `sandbox.init` service that can be run at boot time to set up `/var/tmp` and `/tmp` as `private` (`mount --make-private`).

Note that the sandbox services require MCS policy support as a minimum as categories are used to isolate multiple sandboxes. Issuing the following command will show this usage:

```
sandbox id -Z
unconfined_u:unconfined_r:sandbox_t:s0:c421,c945
```

3. Virtualisation sandboxing of applications using either KVM/qemu or LXC²⁸ (Linux Containers) (`virt-sandbox` - see <http://people.redhat.com/berrange/fosdem-2012/libvirt-sandbox-fosdem-2012.pdf> that contains a good overview).

This service is available in the `libvirt-sandbox` package and provides an API and command line services to start sessions. There is currently limited policy support for `virt-sandbox` as its primary aim is for developers to build services and provide the appropriate policy.

The package is built on `Svirt` that provides the virtualisation with SELinux enforcement and KVM/qemu or LXC to provide the virtualisation environment. If KVM support is not available on the machine (as it requires virtualisation support in the CPU (Intel-VT or AMD-V extensions)), then LXC is the alternative to use.

An LXC example:

```
virt-sandbox -c lxc:/// /bin/sh
```

To run in enforcing mode, the following policy module was added for the `targeted` policy:

```
module lxc_example 1.0.0;

require {
    type svirt_t, virtd_lxc_t, root_t, bin_t, proc_net_t;
    type cache_home_t, user_home_t, boot_t, user_tmp_t;
    class unix_stream_socket { connectto };
    class chr_file { open read write ioctl getattr setattr };
    class file { read write open getattr entrypoint };
    class process { transition sigchld execmem };
    class filesystem getattr;
}
```

²⁸ Linux Containers do not provide a virtual machine, but a virtual environment that has its own process and network space.

```
allow virt_d_lxc_t root_t : chr_file { open read write ioctl setattr };
allow virt_d_lxc_t root_t : file { write open };
allow virt_d_lxc_t svirt_t : process { transition };
allow svirt_t bin_t : file { entrypoint };
allow svirt_t proc_net_t : file { read };
allow svirt_t virt_d_lxc_t : unix_stream_socket { connectto };
allow svirt_t virt_d_lxc_t : process { sigchld };
allow svirt_t cache_home_t : file { read getattr open };
allow svirt_t proc_net_t : file { getattr open };
allow svirt_t root_t : chr_file { read write ioctl open getattr };
allow svirt_t root_t : filesystem { getattr };
allow svirt_t user_home_t : file { read open };
```

that was built and installed as follows:

```
checkmodule -M -m lxc_example.conf -o lxc_example.mod
semodule_package -o lxc_example.pp -m lxc_example.mod
semodule -v -i lxc_example.pp
```

2.24 X-Windows SELinux Support

The SELinux X-Windows (XSELinux) implementation provides fine grained access control over the majority of the X-server objects (known as resources) using an X-Windows extension acting as the object manager (OM). The extension name is "SELinux".

This Notebook will only give a high level description of the infrastructure based on [Figure 2.22](#), however the "[Application of the Flask Architecture to the X Window System Server](#)" [14] paper has a good overview of how the object manager has been implemented, although it does not cover areas such as polyinstantiation.

The X-Windows object classes and permissions are listed in the [X Windows Object Classes](#) section and the Reference Policy modules have been updated to enforce policy using the XSELinux object manager.

On Fedora XSELinux is disabled in the targeted policy but enabled on the MLS policy. This is because Red Hat prefers to use sandboxing with the Xephyr server to isolate windows with the targeted policy, see the [Sandbox Services](#) section for details.

2.24.1 Infrastructure Overview

It is important to note that the X-Windows OM operates on the low level window objects of the X-server. A windows manager (such as Gnome or twm) would then sit above this, however they (the windows manager or even the lower level Xlib) would not be aware of the policy being enforced by SELinux. Therefore there can be situations where X-Windows applications get bitter & twisted at the denial of a service. This can result in either opening the policy more than desired, or just letting the application keep aborting, or modifying the application.

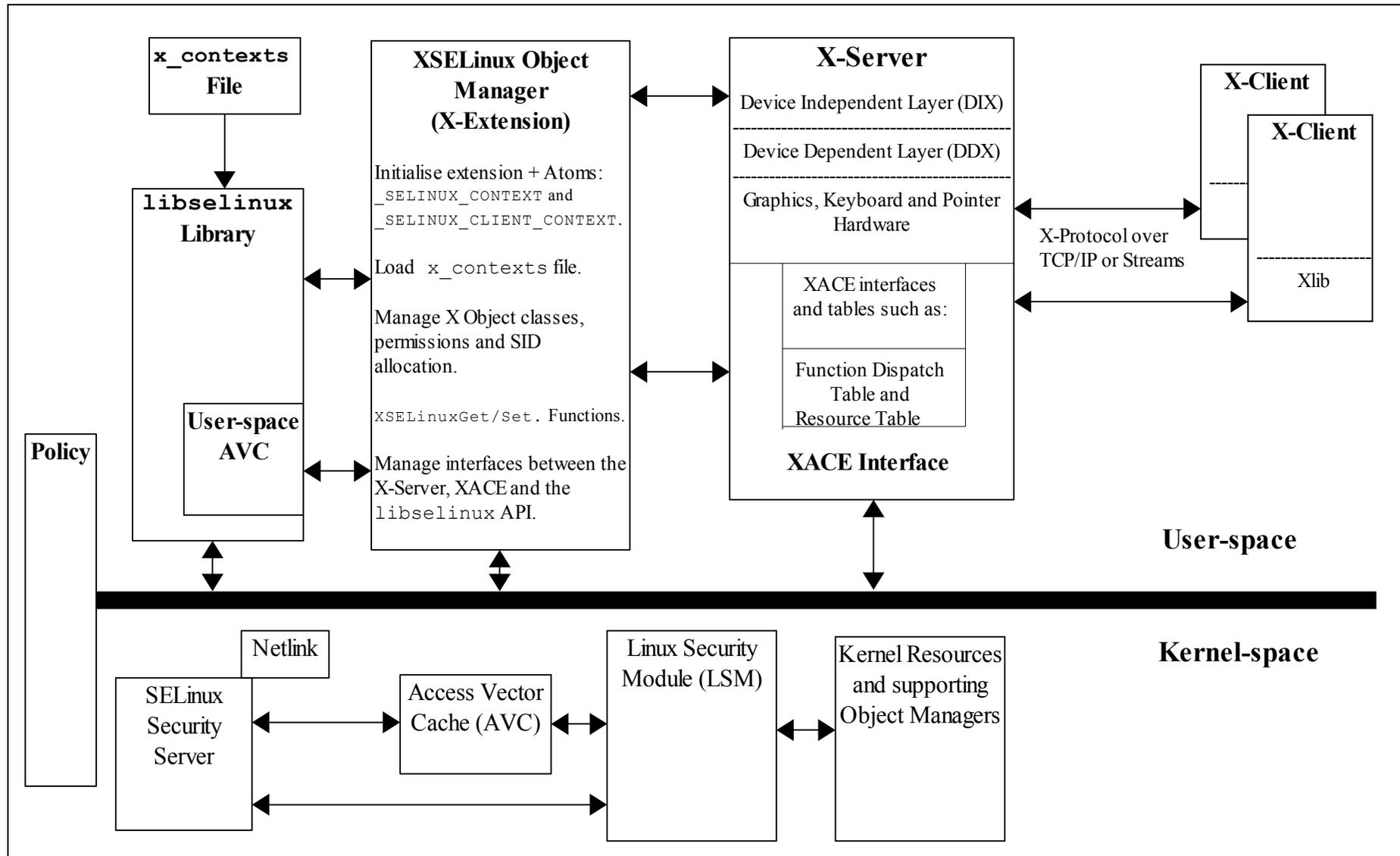


Figure 2.22: X-Server and XSELinux Object Manager - Showing the supporting services. The kernel space services are discussed in the [Linux Security Module and SELinux](#) section.

Using [Figure 2.22](#), the major components that form the overall XSELinux OM are (top left to right):

The Policy - The Reference Policy has been updated, however in Fedora the OM is enabled for mls and disabled for targeted policies via the `xserver-object-manager` boolean. Enabling this boolean also initialises the XSELinux OM extension. Important note - The boolean must be present in any policy and be set to `true`, otherwise the object manager will be disabled as the code specifically checks for the boolean.

libselinux - This library provides the necessary interfaces between the OM, the SELinux userspace services (e.g. reading configuration information and providing the AVC), and kernel services (e.g. security server for access decisions and policy update notification).

x_contexts File - This contains default context configuration information that is required by the OM for labeling certain objects. The OM reads its contents using the `selabel_lookup(3)` function.

XSELinux Object Manager - This is an X-extension for the X-server process that mediates all access decisions between the the X-server (via the XACE interface) and the SELinux security server (via `libselinux`). The OM is initialised before any X-clients connect to the X-server.

The OM has also added XSELinux functions that are described in [Table 12](#) to allow contexts to be retrieved and set by userspace SELinux-aware applications.

XACE Interface - This is an 'X Access Control Extension' (XACE) that can be used by other access control security extensions, not only SELinux. Note that if other security extensions are linked at the same time, then the X-function will only succeed if allowed by all the security extensions in the chain.

This interface is defined in the "[X Access Control Extension Specification](#)" [15]. The specification also defines the hooks available to OMs and how they should be used. The provision of polyinstantiation services for properties and selections is also discussed. The XACE interface is a similar service to the LSM that supports the kernel OMs.

X-server - This is the core X-Windows server process that handles all request and responses to/from X-clients using the X-protocol. The XSELinux OM is intercepting these request/responses via XACE and enforcing policy decisions.

X-clients - These connect to the X-server are typically windows managers such as Gnome, twm or KDE.

Kernel-Space Services - These are discussed in the [Linux Security Module and SELinux](#) section.

2.24.1.1 Polyinstantiation

The OM / XACE services support polyinstantiation of properties and selections allowing these to be grouped into different membership areas so that one group does not know of the existance of the others. To implement polyinstantiation the `poly_` keyword is used in the [x_contexts file](#) for the required selections and properties, there would then be a corresponding [type_member rule](#) in the policy to enforce the

separation by computing a new context with either `security_compute_member(3)` or `avc_compute_member(3)`.

Note that the current Reference Policy does not implement polyinstantiation, instead the MLS policy uses [mlsconstrain rules](#) to limit the scope of properties and selections.

2.24.2 Configuration Information

This section covers:

- How to enable/disable the OM X-extension.
- How to determine the OM X-extension opcode.
- How to configure the OM in a specific SELinux enforcement mode.
- The `x-contexts` configuration file.

2.24.2.1 Enable/Disable the OM from Policy Decisions

The Reference Policy has a `xserver_object_manager` boolean that enables/disables the X-server policy module and also stops the object manager extension from initialising when X-Windows is started. The following command will enable the boolean, however it will be necessary to reload X-Windows to initialise the extension (i.e. run the `init 3` and then `init 5` commands):

```
setsebool -P xserver_object_manager true
```

If the boolean is set to `false`, the x-server log will indicate that "SELinux: Disabled by boolean". Important note - If the boolean is not present in a policy then the object manager will always be enabled (therefore if not required then either do not include the object manager in the X-server build, add the boolean to the policy and set it to `false` or add a disabled entry to the `xorg.conf` file as described in the [Configure OM Enforcement Mode](#) section).

2.24.2.2 Determine OM X-extension Opcode

The object manager is treated as an X-server extension and its major opcode can be queried using Xlib `XQueryExtension` function as follows:

```
/* Get the SELinux Extension opcode */
if (!XQueryExtension(dpy, "SELinux", &opcode, &event, &error)) {
    perror("XSELinux extension not available");
    exit(1);
}
else
    printf("XQueryExtension for XSELinux Extension - Opcode: %d\n", opcode, event, error);
/* Have XSELinux Object Manager */
```

2.24.2.3 Configure OM Enforcement Mode

If the X-server object manager needs to be run in a specific SELinux enforcement mode, then the option may be added to the `xorg.conf` file (normally in `/etc/X11/xorg.conf.d`). The option entries are as follows:

```
"SELinux mode disabled"  
"SELinux mode permissive"  
"SELinux mode enforcing"
```

Note that the entry must be exact otherwise it will be ignored. An example entry is:

```
Section "Module"  
  SubSection "extmod"  
    Option "SELinux mode enforcing"  
  EndSubSection  
EndSection
```

If there is no entry, the object manager will follow the current SELinux enforcement mode.

2.24.2.4 The `x_contexts` File

The `x_contexts` file contains default context information that is required by the OM to initialise the service and then label objects as they are created. The policy will also need to be aware of the context information being used as it will use this to enforce policy or transition new objects. A typical entry is as follows:

```
# object_type object_name context  
selection     PRIMARY     system_u:object_r:clipboard_xselection_t:s0
```

or for polyinstantiation support:

```
# object_type object_name context  
poly_selection PRIMARY     system_u:object_r:clipboard_xselection_t:s0
```

The `object_name` can contain '*' for 'any' or '?' for 'substitute'.

The OM uses the `selabel` functions (such as `selabel_lookup(3)`) that are a part of `libselinux` to fetch the relevant information from the `x_contexts` file.

The valid `object_type` entries are `client`, `property`, `poly_property`, `extension`, `selection`, `poly_selection` and `events`.

The `object_name` entries can be any valid X-server resource name that is defined in the X-server source code and can typically be found in the `protocol.txt` and `BuiltInAtoms` source files (in the `dix` directory of the `xorg-server` source package), or user generated via the Xlib libraries (e.g. `XInternAtom`).

Notes:

1. The way the XSELinux extension code works (see `xselinux_label.c - SELinuxAtomToSIDLookup`) is that non-poly entries are searched for first, if an entry is not found then it searches for a matching poly entry.

The reason for this behavior is that when operating in a secure environment all objects would be polyinstantiated unless there are specific exemptions made for individual objects to make them non-polyinstantiated. There would then be a `'poly_selection *'` or `'poly_property *'` at the end of the section.

2. For systems using the Reference Policy all X-clients connecting remotely will be allocated a security context from the `x_contexts` file of:

```
# object_type  object_name  context
client        *            system_u:object_r:remote_t:s0
```

A full description of the `x_contexts` file format is given in the [x_contexts File](#) section.

2.24.3 SELinux Extension Functions

Function Name	Minor Opcode	Parameters	Comments
XSELinuxQueryVersion	0	None	Returns the XSELinux version. F-20 returns 1.1
XSELinuxSetDeviceCreateContext	1	Context+Len	Sets the context for creating a device object (x_device).
XSELinuxGetDeviceCreateContext	2	None	Retrieves the context set by XSELinuxSetDeviceCreateContext.
XSELinuxSetDeviceContext	3	DeviceID + Context+Len	Sets the context for creating the specified DeviceID object.
XSELinuxGetDeviceContext	4	DeviceID	Retrieves the context set by XSELinuxSetDeviceContext.
XSELinuxSetWindowCreateContext	5	Context+Len	Set the context for creating a window object (x_window).
XSELinuxGetWindowCreateContext	6	None	Retrieves the context set by XSELinuxSetWindowCreateContext.
XSELinuxGetWindowContext	7	WindowID	Retrieves the specified WindowID context.
XSELinuxSetPropertyCreateContext	8	Context + Len	Sets the context for creating a property object (x_property).
XSELinuxGetPropertyCreateContext	9	None	Retrieves the context set by XSELinuxSetPropertyCreateContext.
XSELinuxSetPropertyUseContext	10	Context + Len	Sets the context of the property object to be retrieved when polyinstantiation is being used.
XSELinuxGetPropertyUseContext	11	None	Retrieves the property object context set by SELinuxSetPropertyUseContext.
XSELinuxGetPropertyContext	12	WindowID + AtomID	Retrieves the context of the property atom object.
XSELinuxGetPropertyDataContext	13	WindowID + AtomID	Retrieves the context of the property atom data.
XSELinuxListProperties	14	WindowID	Lists the object and data contexts of properties associated with the selected WindowID.
XSELinuxSetSelectionCreateContext	15	Context+Len	Sets the context to be used for creating a selection object.
XSELinuxGetSelectionCreateContext	16	None	Retrieves the context set by SELinuxSetSelectionCreateContext.
XSELinuxSetSelectionUseContext	17	Context+Len	Sets the context of the selection object to be retrieved when polyinstantiation is being used. See the XSELinuxListSelections function for an example.
XSELinuxGetSelectionUseContext	18	None	Retrieves the selection object context set by SELinuxSetSelectionUseContext.

The SELinux Notebook

Function Name	Minor Opcode	Parameters	Comments
XSELinuxGetSelectionContext	19	AtomID	Retrieves the context of the specified selection atom object.
XSELinuxGetSelectionDataContext	20	AtomID	Retrieves the context of the selection data from the current selection owner (<code>x_application_data</code> object).
XSELinuxListSelections	21	None	<p>Lists the selection atom object and data contexts associated with this display. The main difference in the listings is that when (for example) the PRIMARY selection atom is polyinstantiated, multiple entries can returned. One has the context of the atom itself, and one entry for each process (or x-client) that has an active polyinstantiated entry, for example:</p> <pre> Atom: PRIMARY - label defined in the x_contexts file (this is also for non-poly listing): Object Context: system_u:object_r:primary_xselection_t Data Context: system_u:object_r:primary_xselection_t Atom: PRIMARY - Labels for client 1: Object Context: system_u:object_r:x_select_paste1_t Data Context: system_u:object_r:x_select_paste1_t Atom: PRIMARY - Labels for client 2: Object Context: system_u:object_r:x_select_paste2_t Data Context: system_u:object_r:x_select_paste2_t </pre>
XSELinuxGetClientContext	22	ResourceID	Retrieves the client context of the specified ResourceID.

Table 12: The XSELinux Extension Functions - Supported by the object manager as X-protocol extensions. Note that some functions will return the default contexts, while others (2, 6, 9, 11, 16, 18) will not return a value unless one has been set the the appropriate function (1, 5, 8, 10, 15, 17) by an SELinux-aware application.

2.25 SE-PostgreSQL

This section gives an overview of PostgreSQL version 9.3 with the `sepgsql` extension to support SELinux labeling. It assumes some basic knowledge of PostgreSQL that can be found at: http://wiki.postgresql.org/wiki/Main_Page

It is important to note that PostgreSQL from version 9.3 has the necessary infrastructure to support labeling of database objects via external 'providers'. An `sepgsql` extension has been added that provides SELinux labeling. This is not installed by default but as an option as outlined in the sections that follow. Because of these changes the original version 9.0 patches are no longer supported (i.e. the SE-PostgreSQL database engine is replaced by PostgreSQL database engine 9.3 plus the `sepgsql` extension). A consequence of this change is that row level labeling is no longer supported.

The features of `sepgsql` 9.3 and its setup are covered in the following document:

<http://www.postgresql.org/docs/9.3/static/sepgsql.html>

2.25.1 sepgsql Overview

The `sepgsql` extension adds SELinux mandatory access controls (MAC) to database objects such as tables, columns, views, functions, schemas and sequences. [Figure 2.23](#) shows a simple database with one table, two columns and three rows, each with their object class and associated security context (the [Internal Tables](#) section shows these entries from the `testdb` database in the Notebook tarball example). The database object classes and permissions are described in [Appendix A - Object Classes and Permissions](#).

database		
context = 'unconfined_u:object_r:postgresql_db_t:s0'		
This context is inherited from the database directory label- <code>ls -Z /var/lib/pgsql/data</code>		
schema (db_schema)		
security_label = 'unconfined_u:object_r:sepgsql_schema_t:s10'		
table (db_table)		
security_label = 'unconfined_u:object_r:sepgsql_table_t:s0:c20'		
	column 1 (db_column)	column 2 (db_column)
	security_label = 'unconfined_u:object_r:sepgsql_table_t:s0:c30'	security_label = 'unconfined_u:object_r:sepgsql_table_t:s0:c40'

Figure 2.23: Database Security Context Information - Showing the security contexts that can be associated to a schema, table and columns.

To use SE-PostgreSQL each GNU / Linux user must have a valid PostgreSQL database role (not to be confused with an SELinux role). The default installation automatically adds a user called `pgsql` with a suitable database role.

If a client is connecting remotely and labeled networking is required, then it is possible to use IPSec or NetLabel as discussed in the [SELinux Networking Support](#) section (the "[Security-Enhanced PostgreSQL Security Wiki](#)" [2] also covers these methods of connectivity with examples).

Using [Figure 2.24](#), the database client application (that could be provided by an API for Perl/PHP or some other programming language) connects to a database and executes SQL commands. As the SQL commands are processed by PostgreSQL, each operation performed on an object is checked by the object manager (OM) to see if the operation is allowed by the security policy or not.

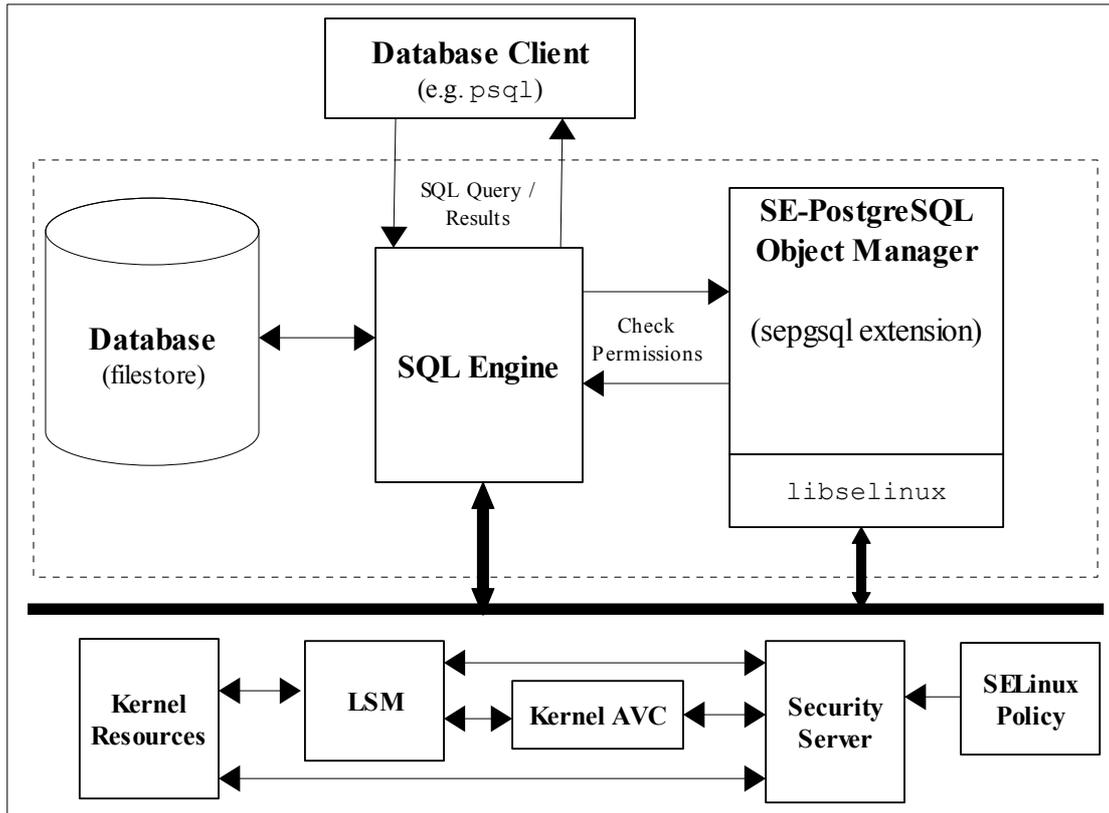


Figure 2.24: SE-PostgreSQL Services - The Object Manager checks access permissions for all objects under its control.

SE-PostgreSQL supports SELinux services via the `libselinux` library with AVC audits being logged into the standard PostgreSQL file as described in the [Logging Security Events](#) section.

2.25.2 Installing SE-PostgreSQL

The <http://www.postgresql.org/docs/devel/static/sepgsql.html> page contains all the information required to install PostgreSQL and the `sepgsql` extension, however the Notebook tarball `sepgsql-9.3/README` file also explains this and adds a simple test database.

2.25.3 SECURITY LABEL SQL Command

The 'SECURITY LABEL' SQL command has been added to PostgreSQL to allow security providers to label or change a label on database objects. The command format is:

```
SECURITY LABEL [ FOR provider ] ON
{
  TABLE object_name |
  COLUMN table_name.column_name |
  AGGREGATE agg_name (agg_type [, ...] ) |
  DATABASE object_name |
  DOMAIN object_name |
  EVENT TRIGGER object_name |
  FOREIGN TABLE object_name
  FUNCTION function_name ( [ [ argmode ] [ argname ] argtype
  [, ...] ] ) |
  LARGE OBJECT large_object_oid |
  [ PROCEDURAL ] LANGUAGE object_name |
  ROLE object_name |
  SCHEMA object_name |
  SEQUENCE object_name |
  TABLESPACE object_name |
  TYPE object_name |
  VIEW object_name
} IS 'label'
```

The full syntax is defined at <http://www.postgresql.org/docs/9.3/static/sql-security-label.html> and also in the **security_label** (7) man page. Some examples taken from the Notebook tarball are:

```
--- These set the security label on objects (default provider
--- is SELinux):
SECURITY LABEL ON SCHEMA test_ns IS
'unconfined_u:object_r:sepgsql_schema_t:s0:c10';
SECURITY LABEL ON TABLE test_ns.info IS
'unconfined_u:object_r:sepgsql_table_t:s0:c20';
SECURITY LABEL ON COLUMN test_ns.info.user_name IS
'unconfined_u:object_r:sepgsql_table_t:s0:c30';
SECURITY LABEL ON COLUMN test_ns.info.email_addr IS
'unconfined_u:object_r:sepgsql_table_t:s0:c40';
```

2.25.4 Additional SQL Functions

The following functions have been added:

sepgsql_getcon()	Returns the client security context.
sepgsql_mcstrans_in(text con)	Translates the readable range of the context into raw format provided the mcstransd daemon is running.
sepgsql_mcstrans_out(text con)	Translates the raw range of the context into readable format provided the mcstransd daemon is running.

<code>sepgsql_restorecon (text specfile)</code>	Sets security contexts on all database objects (must be superuser) according to the <code>specfile</code> . This is normally used for initialisation of the database by the <code>sepgsql.sql</code> script. If the parameter is NULL, then the default <code>sepgsql_contexts</code> file is used. See <code>selabel_db (5)</code> details.
---	--

2.25.5 Additional `postgresql.conf` Entries

The `postgresql.conf` file supports the following additional entries to enable and manage SE-PostgreSQL:

1. This entry is mandatory to enable the `sepgsql` extension to be loaded:

```
shared_preload_libraries = 'sepgsql'
```

2. These entries are optional and default to 'off'. The 'custom_variable_classes' entry must contain 'sepgsql' to enable these to be configured.

```
# This entry allows sepgsql customised entries:
custom_variable_classes = 'sepgsql'

# These are the possible entries:
# This enables sepgsql to always run in permissive mode:
sepgsql.permissive = on

# This enables printing of audit messages regardless of
# the policy setting:
sepgsql.debug_audit = on
```

To view these settings the `SHOW SQL` statement can be used (psql output shown):

```
SHOW sepgsql.permissive;
 sepgsql.permissive
-----
on
(1 row)
```

```
SHOW sepgsql.debug_audit;
 sepgsql.debug_audit
-----
on
(1 row)
```

2.25.6 Logging Security Events

SE-PostgreSQL manages its own AVC audit entries in the standard PostgreSQL log normally located within the `/var/lib/pgsql/data/pg_log` directory and by default only errors are logged (Note that there are no SE-PostgreSQL AVC entries added to the standard `audit.log`). The `'sepgsql.debug_audit = on'` can be set to log all audit events.

2.25.7 Internal Tables

To support the overall database operation PostgreSQL has internal tables in the system catalog that hold information relating to databases, tables etc. This section will only highlight the `pg_seclabel` table that holds the security label and other references. The `pg_seclabel` is described in [Table 13](http://www.postgresql.org/docs/9.3/static/catalog-pg-seclabel.html) that has been taken from <http://www.postgresql.org/docs/9.3/static/catalog-pg-seclabel.html>.

Name	Type	References	Comment
<code>objoid</code>	<code>oid</code>	any OID column	The OID of the object this security label pertains to.
<code>classoid</code>	<code>oid</code>	pg_class.oid	The OID of the system catalog this object appears in.
<code>objsubid</code>	<code>int4</code>		For a security label on a table column, this is the column number (the <code>objoid</code> and <code>classoid</code> refer to the table itself). For all other objects this column is zero.
<code>provider</code>	<code>text</code>		The label provider associated with this label. Currently only SELinux is supported.
<code>label</code>	<code>text</code>		The security label applied to this object.

Table 13: pg_seclabel Table Columns

These are entries taken from a `'SELECT * FROM pg_seclabel;'` command that refer to the example `testdb` database built using the Notebook tarball samples:

objoid	classoid	objsubid	provider	label
16390	2615	0	selinux	unconfined_u:object_r:sepgsql_schema_t:s0:c10
16391	1259	0	selinux	unconfined_u:object_r:sepgsql_table_t:s0:c20
16391	1259	1	selinux	unconfined_u:object_r:sepgsql_table_t:s0:c30
16391	1259	2	selinux	unconfined_u:object_r:sepgsql_table_t:s0:c40

The first entry is the schema, the second entry is the table itself, and the third and fourth entries are columns 1 and 2.

There is also a built-in 'view' to show additional detail regarding security labels called `'pg_seclabels'`. Using `'SELECT * FROM pg_seclabels;'` command, the entries shown above become:

objoid	classoid	objsubid	objtype	objnamespace	objname	provider	label
16390	2615	0	schema	16390	test_ns	selinux	unconfined_u:object_r:sepgsql_schema_t:s0:c10
16391	1259	0	table	16390	test_ns.info	selinux	unconfined_u:object_r:sepgsql_table_t:s0:c20
16391	1259	1	column	16390	test_ns.info.user_name	selinux	unconfined_u:object_r:sepgsql_table_t:s0:c30
16391	1259	2	column	16390	test_ns.info.email_addr	selinux	unconfined_u:object_r:sepgsql_table_t:s0:c40

2.26 Apache SELinux Support

Apache web servers are supported by SELinux using the Apache policy modules from the Reference Policy (`httpd` modules), however there is no specific Apache object manger. There is though an SELinux-aware shared library and policy that will allow finer grained access control when using Apache with threads. The additional Apache module is called `mod_selinux.so` and has a supporting policy module called `mod_selinux.pp`.

The `mod_selinux` policy module makes use of the [typebounds Statement](#) that was introduced into version 24 of the policy (requires a minimum kernel of 2.6.28). `mod_selinux` allows threads in a multi-threaded application (such as Apache) to be bound within a defined set of permissions in that the child domain cannot have greater permissions than the parent domain.

These components are known as 'Apache / SELinux Plus' and are described in the sections that follow, however a full description including configuration details is available from:

http://code.google.com/p/sepgsql/wiki/Apache_SELinux_plus

The objective of these Apache add-on services is to achieve a fully SELinux-aware web stack (although not there yet). For example, currently the LAPP²⁹ (Linux, Apache, PostgreSQL, PHP / Perl / Python) stack has the following support:

L	Linux has SELinux support.
A	Apache has partial SELinux support using the 'Apache SELinux Plus' module.
P	PostgreSQL has SELinux support using SE-PostgreSQL.
P	PHP / Perl / Python are not currently SELinux-aware, however PHP and Python do have support for libselinux functions in packages: PHP - with the <code>php-pecl-selinux</code> package, Python - with the <code>libselinux-python</code> package.

The "[A secure web application platform powered by SELinux](#)" [16] document gives a good overview of the LAPP architecture.

2.26.1 `mod_selinux` Overview

What the `mod_selinux` module achieves is to allow a web application (or a 'request handler') to be launched by Apache with a security context based on policy rather than that of the web server process itself, for example:

1. A user sends an HTTP request to Apache that requires the services of a web application (Apache may or may not apply HTTP authentication).
2. Apache receives the request and launches the web application instance to perform the task:

²⁹ This is similar to the LAMP (Linux, Apache, MySQL, PHP/Perl/Python) stack, however MySQL is not SELinux-aware.

- a) Without `mod_selinux` enabled the web applications security context is identical to the Apache web server process, it is therefore not possible to restrict its privileges.
 - b) With `mod_selinux` enabled, the web application is launched with the security context defined in the `mod_selinux.conf` file (`selinuxDomainVal <security_context> entry`). It is also possible to restrict its privileges as described in the [Bounds Overview](#) section.
3. The web application exits, handing control back to the web server that replies with the HTTP response.

2.26.2 Bounds Overview

Because multiple threads share the same memory segment, SELinux was unable to check the information flows between these different threads when using `setcon(3)` in pre 2.6.28 kernels. This meant that if a thread (the parent) should launch another thread (a child) with a different security context, SELinux could not enforce the different permissions.

To resolve this issue the `typebounds` statement was introduced with kernel support in 2.6.28 that stops a child thread (the 'bounded domain') having greater privileges than the parent thread (the 'bounding domain') i.e. the child thread must have equal or less permissions than the parent.

For example the following `typebounds` statement and `allow` rules:

```
#          parent | child
#          domain | domain
typebounds httpd_t httpd_child_t;

allow httpd_t      etc_t : file { getattr read };
allow httpd_child_t etc_t : file { read write };
```

State that the parent domain (`httpd_t`) has `file : { getattr read }` permissions. However the child domain (`httpd_child_t`) has been given `file : { read write }`. At run-time, this would not be allowed by the kernel because the parent does not have `write` permission, thus ensuring the child domain will always have equal or less privileges than the parent.

When `setcon(3)` is used to set a different context on a new thread without an associated `typebounds` policy statement, then the call will return 'Operation not permitted' and an `SELINUX_ERR` entry will be added to the audit log stating `'op=security_bounded_transition result=denied'` with the old and new context strings.

Should there be a valid `typebounds` policy statement and the child domain exercises a privilege greater than that of the parent domain, the operation will be denied and an `SELINUX_ERR` entry will be added to the audit log stating `'op=security_compute_av reason=bounds'` with the context strings and the denied class and permissions.

2.26.2.1 Notebook Examples

The Notebook source tarball contains two demonstrations using `setcon(3)` with threads and how the `typebounds` statement is used to allow a thread to be executed. These are located in the `libselinux/examples` directory and are:

- a) `setcon_thread1_example.c` - this example calls `setcon` in the main process loop but also starts a thread. If the `setcon_example.conf` policy module has been loaded and a context of `"unconfined_u:unconfined_r:user_t:s0"` selected, then an error message should be displayed as follows:

```
setcon_raw - ERROR: Operation not permitted
```

This is because the `setcon` function cannot be run in a threaded environment without a `typebounds` statement. Now load the `setcon_thread_example.conf` policy module and then re-run the example, it should now complete without error.

- b) `setcon_thread2_example.c` - this functions as example 1, however it calls `setcon` from a thread.

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 commands or `libselinux` library API function names.

Where configuration files have specific man pages, these are noted by adding the man page section (e.g. `semanage.config(5)`).

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. This Notebook will only refer to the commonly used configuration files.
2. [Policy Configuration files](#) used by an active (run time) policy and their supporting [Policy Store Configuration files](#).

The [Policy Store Configuration files](#) are 'private'³⁰ and managed by the `semanage(8)` and `semodule(8)` commands³¹. These are used to build the majority of the [Policy Configuration files](#). This store will be moving as part of a migration programme, see <https://github.com/SELinuxProject/selinux/wiki/Policy-Store-Migration> and [Policy Store Migration](#) for details.

Note that there can be multiple policy configuration areas on a system (e.g. `/etc/selinux/targeted` and `/etc/selinux/mls`), however only one can be the active policy).

3. [SELinux Kernel Configuration files](#) located under the `/sys/fs/selinux` directory and reflect the current configuration of SELinux for the active policy. This area is used extensively by the `libselinux` library for userspace 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.

3.1.1 Policy Store Migration

When distributions move to version 2.4 of `libsemanage`, `libsepol`, and `policycoreutils` the policy module store will move from `/etc/selinux/<SELINUXTYPE>/modules` to `/var/lib/selinux/<SELINUXTYPE>`. Once the libraries are upgraded, all policy stores must be migrated before any commands can be executed that modify or use the store, for example `semodule(8)` or `semanage(8)`. See <https://github.com/SELinuxProject/selinux/wiki/Policy-Store-Migration> for details.

³⁰ They should NOT be edited as together they describe the 'policy'.

³¹ 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(8)`, 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).

Once the migration is complete, it will be possible to build policies containing a mixture of Reference Policy modules, kernel policy language modules and modules written in the CIL language as shown in the following example:

```
# Compile and install a base and two modules written in kernel language:
checkmodule -o base.mod base.conf
semodule_package -o base.pp -m base.mod -f base.fc
checkmodule -m ext_gateway.conf -o ext_gateway.mod
semodule_package -o ext_gateway.pp -m ext_gateway.mod -f gateway.fc
checkmodule -m int_gateway.conf -o int_gateway.mod
semodule_package -o int_gateway.pp -m int_gateway.mod
semodule -s modular-test --priority 100 -i base.pp ext_gateway.pp int_gateway.pp

# Compile and install an updated module written in CIL:
semodule -s modular-test --priority 400 -i custom/int_gateway.cil

# Show a full listing of modules:
semodule -s modular-test --list-modules=full
400 int_gateway cil
100 base pp
100 ext_gateway pp
100 int_gateway pp

# Show a standard listing of modules:
semodule -s modular-test --list-modules=standard
base
ext_gateway
int_gateway
```

Note the use of `--priority 100` and `--priority 400` option that is available after migration for **semodule** (8). This command has a number of new options, with the most significant being:

1. Setting module priorities (`-X | --priority`), this is discussed in [The priority Option](#) section.
2. Listing modules (`--list-modules=full | standard`). The 'full' option shows all the available modules with their priority and policy format. The 'standard' option will only show the highest priority, enabled modules.

3.1.1.1 The priority Option

³²Priorities allow multiple modules with the same name to exist in the policy store, with the higher priority module included in the final kernel binary, and all lower priority modules of the same name ignored. For example:

```
semodule --priority 100 --install distribution/apache.pp
semodule --priority 400 --install custom/apache.pp
```

Both apache modules are installed in the policy store as 'apache', but only the custom apache module is included in the final kernel binary. The distribution apache module is ignored. The `--list-modules` options can be used to show these:

```
# Show a full listing of modules:
semodule --list-modules=full
400 apache pp
100 base pp
100 apache pp

# Show a standard listing of modules:
semodule --list-modules=standard
```

³² This text has been derived from: <http://marc.info/?l=selinux&m=141044198403718&w=2>.

```
base
apache
```

The main use case for this is the ability to override a distribution provided policy, while keeping the distribution policy in the store.

This makes it easy for distributions, 3rd parties, configuration management tools (e.g. puppet), local administrators, etc. to update policies without erasing each others changes. This also means that if a distribution, 3rd party etc. updates a module, providing the local customisation is installed at a higher priority, it will override the new distribution policy.

This does require that policy managers adopt some kind of scheme for who uses what priority. No strict guidelines currently exist, however the value used by the `semanage_migrate_store` script is `--priority 100` as this is assumed to be migrating a distribution. If a value is not provided, `semodule` will use a default of `--priority 400` as it is assumed to be a locally customised policy.

When `semodule` builds a lower priority module when a higher priority is already available, the following message will be given: "A higher priority <name> module exists at priority <999> and will override the module currently being installed at priority <111>".

3.1.1.2 Converting policy packages to CIL

A component of the update is to add a facility that converts compiled policy modules (known as policy packages or the `*.pp` files) to CIL format. This is achieved via a `pp` to CIL high level language conversion utility located at `/usr/libexec/selinux/hll/pp`. This utility can be used manually as follows:

```
cat module_name.pp | /usr/libexec/selinux/hll/pp > module_name.cil
```

There is no man page for 'pp', however the help text is as follows:

```
Usage: pp [OPTIONS] [IN_FILE [OUT_FILE]]

Read an SELinux policy package (.pp) and output the equivalent CIL.
If IN_FILE is not provided or is -, read SELinux policy package from
standard input. If OUT_FILE is not provided or is -, output CIL to
standard output.

Options:
-h, --help      print this message and exit
```

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. The two most important files are:

- `/etc/selinux/config` - This defines the policy to be activated and its enforcing mode.

- `/etc/selinux/semanage.conf` - This is used by the SELinux policy configuration subsystem for modular or CIL policies.

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 file man page is `selinux_config(5)`, this is because 'config' has already been taken. 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	<p>This entry can contain one of three values:</p> <p>enforcing</p> <p>SELinux security policy is enforced.</p> <p>permissive</p> <p>SELinux logs warnings (see the Auditing SELinux Events section) instead of enforcing the policy (i.e. the action is allowed to proceed).</p> <p>disabled</p> <p>No SELinux policy is loaded.</p> <p>Note that this configures the global SELinux enforcement mode. It is still possible to have domains running in permissive mode and/or object managers running as disabled, permissive or enforcing, when the global mode is enforcing or permissive.</p>
SELINUXTYPE	<p>The <code>policy_name</code> is used as the directory name where the active policy and its configuration files will be located. The system will then use this information to locate and load the policy contained within this directory structure.</p> <p>The policy directory must be located at:</p> <p style="text-align: center;"><code>/etc/selinux/<policy_name>/</code></p>
SETLOCALDEFS	<p>This optional field should be set to 0 (or the entry removed) as the policy store management infrastructure (<code>semanage(8)</code> / <code>semodule(8)</code>) is now used.</p> <p>If set to 1, then <code>init(8)</code> and <code>load_policy(8)</code> will read the local customisation for booleans and</p>

	users.
REQUIRESEUSERS	<p>This optional field can be used to fail a login if there is no matching or default entry in the seusers file or if the file is missing.</p> <p>It is checked by the <code>libselinux</code> function getseuserbyname(3) that is used by SELinux-aware login applications such as PAM(8).</p> <p>If it is set to 0 or the entry missing:</p> <p style="padding-left: 40px;">getseuserbyname(3) will return the GNU / Linux user name as the SELinux user.</p> <p>If it is set to 1:</p> <p style="padding-left: 40px;">getseuserbyname(3) will fail.</p>
AUTORELABEL	<p>This is an optional field. If set to '0' and there is a file called <code>.autorelabel</code> 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.</p> <p>If set to '1' or the parameter name is not used (the default) there is no login for manual relabeling, however should the <code>/.autorelabel</code> file exists, then the file system will be automatically relabeled using <code>fixfiles -F restore</code>.</p> <p>In both cases the <code>/.autorelabel</code> file will be removed so the relabel is not done again.</p>

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.config**(5) file controls the configuration and actions of the **semanage**(8) and **semodule**(8) 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
bzip-blocksize = 0|1..9
bzip-small true|false
usepasswd = true|false
ignoredirs dir [;dir] ...

[verify kernel]
path = <application_to_run>
args = <arguments>
[end]

[verify module]
path = <application_to_run>
args = <arguments>
[end]

[verify linked]
path = <application_to_run>
args = <arguments>
[end]

[load_policy]
path = <application_to_run>
args = <arguments>
[end]

[setfiles]
path = <application_to_run>
args = <arguments>
[end]

[sefcontext_compile]
path = <application_to_run>
args = <arguments>
[end]

[load_policy]
path = <application_to_run>
args = <arguments>
[end]

# libsepol (v2.4) with CIL support add the following:
store-root = <path>
compiler-directory = <path>
ignore-module-cache = true|false
target-platform = selinux | xen
```

Where:

module-store	The method can be one of four options: direct libsemanage will write
--------------	--

	<p>source directly to a module store. This is the default value.</p> <p>libsemanage manipulates a source SELinux policy.</p> <p>/foo/bar Write via a policy management server, whose named socket is at /foo/bar. The path must begin with a '/'. </p> <p>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.</p>
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.
expand-check	<p>This optional entry controls whether hierarchy checking on module expansion is enabled (1) or disabled (0). The default is 0.</p> <p>It is also required to detect the presence of policy rules that are to be excluded with <code>neverallow</code> rules.</p>
file-mode	This optional entry allows the file permissions to be set on runtime policy files. The format is the same as the <code>mode</code> parameter of the <code>chmod</code> command and defaults to <code>0644</code> if not present.
save-previous	This optional entry controls whether the previous module directory is saved (<code>TRUE</code>) after a successful commit to the policy store. The default is to delete the previous version (<code>FALSE</code>).
save-linked	<p>This optional entry controls whether the previously linked module is saved (<code>TRUE</code>) after a successful commit to the policy store. Note that this option will create a <code>base.linked</code> file in the module policy store.</p> <p>The default is to delete the previous module (<code>FALSE</code>).</p>
disable-genhomedircon	This optional entry controls whether the embedded <code>genhomedircon</code> function is run

	<p>when using the semanage (8) command. The default is <code>FALSE</code>.</p>
<code>handle-unknown</code>	<p>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 <code>base.conf</code> (loadable policy) or <code>policy.conf</code> (monolithic policy)).</p> <p>The options are: <code>allow</code> the permission, <code>reject</code> by not loading the policy or <code>deny</code> the permission. The default is <code>deny</code>. See the SELinux Filesystem section for how these are reported in <code>/sys/fs/selinux</code>.</p> <p>Note: to activate any change, the base policy needs to be rebuilt with the <code>semodule -B</code> command.</p>
<code>bzip-blocksize</code>	<p>This optional entry determines whether the modules are compressed or not with <code>bzip</code>. If the entry is <code>0</code>, then no compression will be used (this is required with tools such as <code>sechecker</code> and <code>apol</code>). This can also be set to a value between <code>1</code> and <code>9</code> that will set the block size used for compression (<code>bzip</code> will multiply this by <code>100,000</code>, so <code>'9'</code> is faster but uses more memory).</p>
<code>bzip-small</code>	<p>When this optional entry is set to <code>TRUE</code> the memory usage is reduced for compression and decompression (the <code>bzip -s</code> or <code>--small</code> option). If <code>FALSE</code> or no entry present, then does not try to reduce memory requirements.</p>
<code>usepasswd</code>	<p>When this optional entry is set to <code>TRUE</code> <code>semanage</code> will scan all password records for home directories and set up their labels correctly.</p> <p>If set to <code>FALSE</code> (the default if no entry present), then only the <code>/home</code> directory will be automatically re-labeled.</p>
<code>ignoredirs</code>	<p>With a list of directories to ignore (separated by <code>;</code>) when setting up users home directories. This is used by some distributions to stop labeling <code>/root</code> as a home directory.</p>
<code>[verify kernel]</code>	<p>Start an additional set of entries that can be used to validate the kernel policy with an external application during the build process. There may be multiple <code>[verify kernel]</code> entries.</p> <p>The validation process takes place before the policy is allowed to be inserted into the store with</p>

	a worked example shown in Appendix E - Policy Validation Example .
[verify module]	Start an additional set of entries that can be used to validate each module by an external application during the build process. There may be multiple [verify module] entries.
[verify linked]	Start an additional set of entries that can be used to validate module linking by an external application during the build process. There may be multiple [verify linked] entries.
[load_policy]	Replace the default load policy application with this new policy loader. Defaults are either: /sbin/load_policy or /usr/sbin/load_policy.
[setfiles]	Replace the default set files application with this new set files. Defaults are either: /sbin/setfiles or /usr/sbin/setfiles.
[sefcontexts_compile]	Replace the default file context build application with this new builder. Defaults are either: /sbin/sefcontexts_compile or /usr/sbin/sefcontexts_compile.

For libsepol (v2.4) with CIL support add the following entries:

store-root	Specify an alternative store root path to use. The default is "/var/lib/selinux".
compiler-directory	Specify an alternate directory that will hold the High Level Language (HLL) to CIL compilers. The default is "/usr/libexec/selinux/hll".
ignore-module-cache	Whether or not to ignore the cache of CIL modules compiled from HLL. The default is false.
target-platform	Target platform for generated policy. Default is "selinux", the alternate is "xen".

Example semanage.conf file contents are:

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

module-store = direct
expand-check = 0

[verify kernel]
path = /usr/local/bin/validate
```

```
args = $@  
[end]
```

3.2.3 /etc/selinux/restorecond.conf and restorecond-user.conf Files

The `restorecond.conf` file contains a list of files that may be created by applications with an incorrect security context. The `restorecond(8)` daemon will then watch for their creation and automatically correct their security context to that specified by the active policy file context configuration files³³ (located in the `/etc/selinux/<policy_name>/contexts/files` directory).

Each line of the file contains the full path of a file or directory. Entries that start with a tilde (`~`) will be expanded to watch for files 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).

Note that it is possible to run `restorecond` in a user session using the `-u` option (see `restorecond(8)`). This requires a `restorecond-user.conf` file to be installed as shown in the examples below.

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
```

Example `restorecond-user.conf` file contents are:

```
# /etc/selinux/restorecond-user.conf  
  
# This entry expands to listen for all files created for all  
# logged in users within their home directories:  
~/*  
~/public_html/*
```

3.2.4 /etc/selinux/newrole_pam.conf

The optional `newrole_pam.conf` file is used by `newrole(1)` and maps applications or commands to `PAM(8)` configuration files. Each line contains the executable file name followed by the name of a `pam` configuration file that exists in `/etc/pam.d`.

³³ The daemon uses functions in `libselinux` such as `matchpathcon(3)` to manage the context updates.

3.2.5 /etc/sestatus.conf File

The **sestatus.conf** (5) 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 file has the following parameters:

```
[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
```

3.2.6 /etc/security/sepermit.conf File

The **sepermit.conf** (5) 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 **pam_sepermit**(5), with the following example that 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 an optional argument:
# exclusive - only single login session will be allowed for
#             the user and the user's processes will be
#             killed on logout
#
# ignore - The module will never return PAM_SUCCESS status
#          for the user.
#
# An example entry for 'kiosk mode':
xguest:exclusive
```

3.3 Policy Store Configuration Files

Depending on the release being used policy stores will be located at:

- `/etc/selinux/<policy_name>/modules` - This is the default for systems that support versions `< 2.4` of `libsemanage`, `libsepol`, and `policycoreutils`.
- `/var/lib/selinux/<policy_name>/modules` - This is the default for systems that support versions `>= 2.4` of `libsemanage`, `libsepol`, and `policycoreutils`. The base (`/var/lib/selinux`) may be overridden by the `store-root` parameter defined in the [semanage.conf](#)(5) file. The migration process from previous releases is described at <https://github.com/SELinuxProject/selinux/wiki/Policy-Store-Migration>.

Note that there can be multiple policy stores on a system, each file described in this section is relative to the `./<policy_name>` as discussed above.

The Policy Store files are either installed, updated or built by the **semodule**(8) and **semanage**(8) commands as a part of the build process. The resulting files will either be copied over to the [Policy Configuration files](#) area, or used to rebuild the kernel binary policy located at `/etc/selinux/<policy_name>/policy`.

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

The command options and outputs shown in the text are based on the current F-20 build. After the migration programme, some command options and their output will change.

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, permission declarations and initial SIDs.

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 as shown in [Figure 3.1](#):

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.

Note that as a part of the `semanage` build process, these two files will also have `file_contexts.bin` and `file_contexts.homedirs.bin` files present in the [Policy Configuration Files](#) `./contexts/files` directory. This is because `semanage` requires these in the Perl compatible regular expression (PCRE) internal format. They are generated by the `sefcontext_compile(8)` utility.

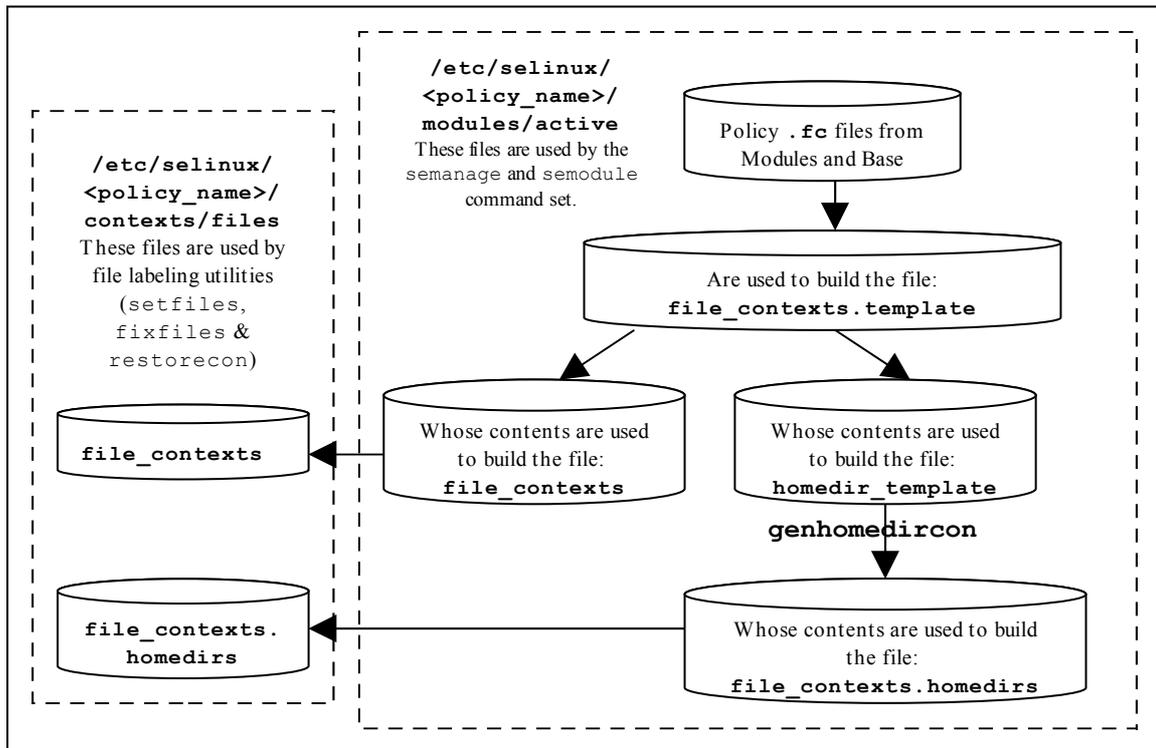


Figure 3.1: File Context Configuration Files - The two files copied to the policy area will be used by the file labeling utilities to relabel files.

The homedir_template and file_contexts files are built is as follows:

homedir_template - Any line in the file_contexts.template file that has the keywords HOME_ROOT, HOME_DIR and/or USER 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 may also have the ROLE keyword declared.

The homedir_template file will then be processed by **genhomedircon**(8)³⁴ to generate individual SELinux user entries in the file_contexts.homedirs file as discussed in the [./modules/active/file_contexts.homedirs](#) section.

These are examples of one line being processed as described above, taken from the F-20 targeted policy:

The master file_contexts.template entry:

```
HOME_DIR\./.wine(/.*)?    system_u:object_r:wine_home_t:s0
```

³⁴ The genhomedircon command has now been built into the libsemanage library as a function to build the file_contexts.homedirs file via **semanage**(8).

The `homedir_template` entry is created as:

```
HOME_DIR\./.wine(/.*)?    system_u:object_r:wine_home_t:s0
```

The `file_contexts.homedirs` entries are created by `genhomedircon` for the SELinux users extracted from the [seusers](#) file as follows:

```
# Home Context for any Linux user that is assigned
# the SELinux user unconfined_u
/home/[^/]*\./.wine(/.*)?    unconfined_u:object_r:wine_home_t:s0

# Home Context for user root
/root/\.wine(/.*)?    unconfined_u:object_r:wine_home_t:s0
```

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.

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

Each line within the file consists of the following:

```
pathname_regexp [file_type] opt_security_context
```

Where:

<code>pathname_regexp</code>	<p>An entry that defines the pathname that may be in the form of a regular expression.</p> <p>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 <code>file_contexts</code> files below).</p> <p>There are also keywords of <code>HOME_ROOT</code>, <code>HOME_DIR</code>, <code>ROLE</code> and <code>USER</code> that are used by file labeling commands (see the keyword definitions below and the ./modules/active/homedir_template file section for their usage).</p>						
<code>file_type</code>	<p>One of the following optional <code>file_type</code> entries (note if blank means "all file types"):</p> <table><tr><td>'-b' - Block Device</td><td>'-c' - Character Device</td></tr><tr><td>'-d' - Directory</td><td>'-p' - Named Pipe (FIFO)</td></tr><tr><td>'-l' - Symbolic Link</td><td>'-s' - Socket File</td></tr></table>	'-b' - Block Device	'-c' - Character Device	'-d' - Directory	'-p' - Named Pipe (FIFO)	'-l' - Symbolic Link	'-s' - Socket File
'-b' - Block Device	'-c' - Character Device						
'-d' - Directory	'-p' - Named Pipe (FIFO)						
'-l' - Symbolic Link	'-s' - Socket File						

	'--' - Ordinary file By convention this entry is known as 'file type', however it really represents the 'file object class'.
opt_security_context	This entry can be either: <ol style="list-style-type: none"> 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 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' is the default.
HOME_DIR	This keyword is replaced by the GNU / Linux users home directory, normally '/home/' is the default.
USER	This keyword will be replaced by the users GNU / Linux user id.
ROLE	<p>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:</p> <pre style="border: 1px solid black; padding: 5px;">user user_u prefix user; user staff_u prefix staff;</pre> <p>It is used for files and directories within the users home directory area.</p> <p>The prefix can be added by the semanage login command as follows (although note that the -P option is suppressed when help is displayed as it is generally it is not used (defaults to user) - see http://blog.gmane.org/gmane.linux.redhat.fedora.selinux/month=20110701 for further information):</p> <pre style="border: 1px solid black; padding: 5px;"># Add a Linux user: adduser rch # Modify staff_u SELinux user and prefix: semanage user -m -R staff_r -P staff staff_u # Associate the SELinux user to the Linux user: semanage login -a -s staff_u rch</pre>

Example file_contexts.template contents from targeted policy:

```
# ./modules/active/file_contexts.template - These sample entries
```

```
# have been taken from the targeted policy and show the
# HOME_DIR, HOME_ROOT and USER keywords whose lines will be
# extracted and added to the homedir_template file that is
# used to manage user home directory entries.

/*          system_u:object_r:default_t:s0
/[^/]+      -- system_u:object_r:etc_runtime_t:s0
/a?quota\.(user|group) -- system_u:object_r:quota_db_t:s0
/nsr(/.*)?  system_u:object_r:var_t:s0
/sys(/.*)?  system_u:object_r:sysfs_t:s0
...
/etc/ntop.* system_u:object_r:ntop_etc_t:s0
HOME_DIR/.+ system_u:object_r:user_home_t:s0
/dev/dri/.+ -c system_u:object_r:dri_device_t:s0
...
/tmp/gconfd-USER -d system_u:object_r:user_tmp_t:s0
...
/tmp/gconfd-USER/* -- system_u:object_r:gconf_tmp_t:s0
...
HOME_ROOT/\.journal <<none>>
```

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 3.1](#). 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 targeted policy.
# The keywords HOME_DIR, HOME_ROOT, USER and ROLE have been
# removed and put in the homedir_template file.

/*          system_u:object_r:default_t:s0
/[^/]+      -- system_u:object_r:etc_runtime_t:s0
/a?quota\.(user|group) -- system_u:object_r:quota_db_t:s0
/nsr(/.*)?  system_u:object_r:var_t:s0
/sys(/.*)?  system_u:object_r:sysfs_t:s0
/xen(/.*)?  system_u:object_r:xen_image_t:s0
/mnt(/[^/]*) -l system_u:object_r:mnt_t:s0
/mnt(/[^/]*)? -d system_u:object_r:mnt_t:s0
/bin/*      system_u:object_r:bin_t:s0
/dev/*      system_u:object_r:device_t:s0
/usr/*      system_u:object_r:usr_t:s0
/var/*      system_u:object_r:var_t:s0
/run/*      system_u:object_r:var_run_t:s0
/srv/*      system_u:object_r:var_t:s0
/tmp/*      <<none>>
```

```
# ./contexts/files/file_contexts - Sample entries from the
# MLS 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/[shmxd[^[^/]*) -b system_u:object_r:fixed_disk_device_t:s15:c0.c255
/var/[xgk]dm(/.*)? system_u:object_r:xserver_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 3.1](#)) 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 targeted policy and show the
# HOME_DIR, HOME_ROOT and USER keywords that are used to manage
# users home directories:

HOME_DIR/.+      system_u:object_r:user_home_t:s0
/tmp/gconfd-USER -d system_u:object_r:user_tmp_t:s0
/tmp/gconfd-USER/.* -- system_u:object_r:gconf_tmp_t:s0
HOME_ROOT/\.journal <<none>>
```

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 3.1](#). 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 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`, `ROLE` and `USER` keywords will be replaced as explained in the [keyword definitions](#) section above.

Example `file_contexts.homedirs` contents:

```
# ./modules/active/file_contexts.homedirs - These sample entries
# have been taken from the targeted policy and show that
# the HOME_DIR, HOME_ROOT and USER keywords have been replaced
# by entries as explained above.
#
# Home Context for the default user (unconfined_u)
/home/[^/]*/.+          unconfined_u:object_r:user_home_t:s0
/home/[^/]*/.maildir(/.*)? unconfined_u:object_r:mail_home_rw_t:s0
...
/tmp/gconfd-.*/*      --  unconfined_u:object_r:gconf_tmp_t:s0
/tmp/gconfd-.*      -d   unconfined_u:object_r:user_tmp_t:s0

# Home Context for user rch
/home/rch/.+         staff_u:object_r:user_home_t:s0
/home/rch/.maildir(/.*)? staff_u:object_r:mail_home_rw_t:s0
...
/tmp/gconfd-rch/*    --  staff_u:object_r:gconf_tmp_t:s0
/tmp/gconfd-rch     -d   staff_u:object_r:user_tmp_t:s0

# Home Context for user root
/root/.+            unconfined_u:object_r:user_home_t:s0
/root/.maildir(/.*)? unconfined_u:object_r:mail_home_rw_t:s0
...
/tmp/gconfd-root/*  --  unconfined_u:object_r:gconf_tmp_t:s0
/tmp/gconfd-root    -d   unconfined_u:object_r:user_tmp_t:s0
```

3.3.9 `modules/active/netfilter_contexts & netfilter.local` File

These files are not used at present. There is code to produce a `netfilter_contexts` file for use by the GNU/Linux `iptables` service³⁵ 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).

3.3.10 `modules/active/policy.kern` File

This is the binary policy file built by either the `semanage` (8) or `semodule` (8) commands (depending on the configuration action), that is then becomes the [./policy/policy.\[ver\]](#) binary policy that will be loaded into the kernel.

³⁵ This uses `SECMARK` labeling that has been utilised by SELinux as described in the [SELinux Networking Support](#) section.

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`³⁶ 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 follows³⁷:

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

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.

It is also possible to associate a GNU / Linux group of users to an SELinux user as follows:

```
semanage login -a -s staff_u %staff_group
```

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

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

Where:

<code>user_id</code>	Where <code>user_id</code> is the GNU / Linux user identity. If this is a GNU / Linux <code>group_id</code> then it will be preceded with the '%' sign as shown in the example below.
<code>seuser_id</code>	The SELinux user identity.
<code>range</code>	The optional level or range.

Example `seusers.final` file contents:

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

³⁶ 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.

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

```
__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
```

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

```
# This command will add the %user_group:user_u entry in the
# seusers file:

semanage login -a -s user_u %user_group
```

The resulting `seusers` file would be:

```
# ./modules/active/seusers

rch:user_u
%user_group: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
%user_group: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 (although no longer used by policies as discussed in the [./modules/active/file_contexts.template](#) file section).

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 follows³⁸:

```
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 updated (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:

³⁸ The Reference Policy Makefile 'Rules.modular' script uses this method to install the initial `users_extra` file.

```
user seuser_id prefix prefix_id;
```

Where:

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.

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(8)` command.

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(8)` command.

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(8)` command.

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/preserve_tunables File

This file will only exist if the policy build specified that tunables should be preserved, if so they would be converted to booleans by the policy build process.

3.3.19 modules/active/disable_dontaudit File

This file will only exist if the policy build specified that [dontaudit](#) rules should be disabled.

3.3.20 modules/active/modules Directory Contents

This directory contains loadable modules (`<module_name>.pp` or when disabled `<module_name>.pp.disabled`) that have been built by the `semodule_package` command and placed in the store by the `semodule` or `semanage module -a` commands 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
# Or:
semanage module -a -S modular-test move_file_c.pp
```

The modules within the policy store may be compressed or not depending on the value of the `bzip-blocksize` parameter in the [semanage.conf](#) file. The modules and their status can be listed using the `semanage module -l` command as shown below.

```
semanage module -l
ext_gateway      1.1.0
int_gateway      1.1.0
move_file        1.1.0
netlabel         1.0.0      Disabled
```

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**(8) or **semodule**(8) commands. 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 Notebook source tarball could run at `init 3` (i.e. no X-Windows) and only require the following configuration files:

`./policy/policy.29` - 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 two files are required:

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

`./context/x_contexts` - To allow the X-Windows service to run under SELinux.

3.4.1 seusers File

The **seusers** (5) 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 booleans and `booleans.local` File

Generally these **booleans** (5) files are not present if **semanage** (8) is being used to manage booleans (see the [modules/active/booleans.local File](#) section). However if `semanage` is not being used or there is an SELinux-aware application that uses the

libselinux functions listed below, then these files may be present (they could also be present in older Reference policies):

security_set_boolean_list(3) - Writes a `boolean.local` file if flag `permanent = '1'`.

security_load_booleans(3) - Will look for a `booleans` or `booleans.local` file here unless a specific path is specified.

Both files have the same format and contain one or more boolean names. The format is:

```
boolean_name value
```

Where:

<code>boolean_name</code>	The name of the boolean.
<code>value</code>	The default setting for the boolean that can be one of the following: <code>true false 1 0</code>

Note that if `SETLOCALDEFS` is set in the SELinux [config](#) file, then **selinux_mkload_policy**(3) will check for a `booleans.local` file in the **selinux_booleans_path**(3), and also a `local.users` file in the **selinux_users_path**(3).

3.4.3 `booleans.subs_dist` File

The `booleans.subs_dist` file (if present) will allow new boolean names to be allocated to those in the active policy. This file was added because many older booleans began with 'allow' that made it difficult to determine what they did. For example the boolean `allow_console_login` becomes more descriptive as `login_console_enabled`. If the `booleans.subs_dist` file is present, then either name maybe used. **selinux_booleans_subs_path**(3) will return the active policy path to this file and **selinux_boolean_sub**(3) will will return the translated name.

Each line within the substitution file `booleans.subs_dist` is:

```
policy_bool_name new_name
```

Where:

`policy_bool_name`

The policy boolean name.

`new_name`

The new boolean name.

Example:

```
# ./booleans.subs_dist
```

```
# policy_bool_name           new_name
allow_auditadm_exec_content  auditadm_exec_content
allow_console_login          login_console_enabled
allow_cvs_read_shadow        cvs_read_shadow
allow_daemons_dump_core     daemons_dump_core
```

When `security_get_boolean_names(3)` or `security_set_boolean(3)` is called with a boolean name and the `booleans.subs_dist` file is present, the name will be looked up and if using the `new_name`, then the `policy_bool_name` will be used (as that is what is defined in the active policy).

Supporting `libselinux` API functions are:

```
selinux_booleans_subs_path
selinux_booleans_sub
security_get_boolean_names
security_set_boolean
```

3.4.4 `setrans.conf` File

The `setrans.conf(8)` 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.

There are a number of sample configuration files within the `mcstrans` package that describe the configuration options in detail that are located at `/usr/share/mcstrans/examples`.

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

The translations can be disabled by added the following line to the file:

```
disable = 1
```

This file will also support the display of information in colour. The configuration file that controls this is called `secolor.conf` and is described in the [secolor.conf File](#) section.

The file format is described in `setrans.conf(8)` with the following giving an overview:

```
# Syntax

# A domain is a self consistent domain of translation (English, German,
Paragraph Markings ...)
Domain=NAME1

# Within a domain are a number of fixed translations
# format is raw_range=trans_range
s3:c200.c511=Confidential
# repeat as required...

# Within a domain are variable translations that are a Base + ModifierGroup +
ModifierGroup
```

```
Base=Sensitivity Levels
# raw_range=name
s1=Unclassified
# Aliases have the same name but a different translation.
# The first one is used to compute translations
s1=U
# inverse bits should appear in the base of any level that uses inverse bits
s2:c200.c511=Restricted
# repeat as required...

# Modifier Groups should be in the order of appearance in the translated range.
ModifierGroup=GROUP1
# Allowed white space can be defined
Whitespace=- ,/
# Join defines the character between multiple members of this group
Join=/
# A Prefix can be defined per group
Prefix=Releasable to
# Inverse categories (releasabilities) should always be set as Default
categories in every ModifierGroup
Default=c200.c511
# format is raw_categories=name
# ~ turns off inverse bits
~c200.c511=EVERYBODY

# Aruba - bit 201
~c200,~c201=ABW
~c200,~c201=AA
# Afghanistan - bit 202
~c200,~c202=AFG
~c200,~c202=AF
# repeat as required...

# Another Modifier Group
ModifierGroup=GROUP2
# With different white space
Whitespace=
# And different Join
Join=,
# A Suffix can be defined per group
Suffix=Eyes only
# Default categories need to be consistent
Default=c200.c511

# New domain
Domain=NAME2

# any text can be put in a separate file
Include=PATH
Include=PATH
```

Example file contents:

```
# ./setrans.conf
#
# 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
```

Supporting `libselinux` API functions are:

```
selinux_translations_path
selinux_raw_to_trans_context
selinux_trans_to_raw_context
```

3.4.5 `secolor.conf` File

The `secolor.conf` (5) file controls the colour to be associated to the components of a context when information is displayed by an SELinux colour-aware application (currently none, although there are two examples in the Notebook source tarball under the `libselinux/examples` directory). The file format is as follows:

```
color color_name = #color_mask

context_component string fg_color_name bg_color_name
```

Where:

<code>color</code>	The color keyword.
<code>color_name</code>	A descriptive name for the colour (e.g. red).
<code>color_mask</code>	A colour mask starting with a hash (#) that describes the RGB colours with black being #000000 and white being #ffffff.
<code>context_component</code>	The colour translation supports different colours on the context string components (user, role, type and range). Each component is on a separate line.
<code>string</code>	This is the <code>context_component</code> string that will be matched with the raw context component passed by <code>selinux_raw_context_to_color</code> (3) A wildcard '*' may be used to match any undefined string for the user, role and type <code>context_component</code> entries only

	A wildcard '*' may be used to match any undefined string for the user, role and type context_component entries only.
fg_color_name	The color_name string that will be used as the foreground colour. A color_mask may also be used.
bg_color_name	The color_name string that will be used as the background colour. A color_mask may also be used.

Example file contents:

```
color black = #000000
color green = #008000
color yellow = #ffff00
color blue = #0000ff
color white = #ffffff
color red = #ff0000
color orange = #ffa500
color tan = #D2B48C

user * = black white
role * = white black
type * = tan orange
range s0-s0:c0.c1023 = black green
range s1-s1:c0.c1023 = white green
range s3-s3:c0.c1023 = black tan
range s5-s5:c0.c1023 = white blue
range s7-s7:c0.c1023 = black red
range s9-s9:c0.c1023 = black orange
range s15:c0.c1023 = black yellow
```

Supporting libselinux API functions are:

```
selinux_colors_path
selinux_raw_context_to_color - this call returns the foreground
and background colours of the context string as the specified
RGB 'color' hex digits as follows:
    user      :      role      :      type      :      range
#000000 #ffffff #ffffff #000000 #d2b48c #ffa500 #000000 #008000
black  white  white  black  tan    orange black  green
```

3.4.6 policy/policy.<ver> File

This is the binary policy file that is loaded into the kernel to enforce policy and is built by either checkpolicy or semodule. 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](#)" [3] notes.

By convention the file name extension is the policy database version used to build the policy, however it is not mandatory as the true version is built into the policy file. The different policy versions are discussed in the [Policy Versions](#) section.

3.4.7 contexts/customizable_types File

The `customizable_types` (5) 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:

type	The type 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

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
```

3.4.8 contexts/default_contexts File

The `default_contexts` (5) file is 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[:range] role:type[:range] ...
```

Where:

<pre>role:type[:range]</pre>	<p>The file contains one or more lines that consist of <code>role:type[:range]</code> pairs (including the MLS / MCS level or range if applicable).</p> <p>The entry at the start of a new line corresponds to the partial <code>role:type[:range]</code> context of (generally) the login application.</p> <p>The other <code>role:type[:range]</code> entries on that line represent an ordered list of valid contexts that may be used to set the users context.</p>
------------------------------	---

Example file contents:

```
# ./contexts/default_contexts

system_r:crond_t:s0          system_r:system_crond_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
```

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 `unconfined_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. `unconfined_u:unconfined_r:ext_gateway_t`), then it was found that the login process would always set the logged in user context to

unconfined_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/unconfined_u` configuration file instead could also have achieved this.

3.4.9 contexts/dbus_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(1)` man page details the contents and the Free Desktop web site has detailed information at:

<http://dbus.freedesktop.org>

Example file contents:

```
# ./contexts/dbus_contexts
<!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
```

3.4.10 contexts/default_type File

The `default_type(5)` file allows SELinux-aware applications such as `newrole(1)` to select a default type for a role if one is not supplied.

The file format is as follows:

```
role:type
```

Where:

<code>role:type</code>	The file contains one or more lines that consist of <code>role:type</code> entries. There should be one line for each role defined within the policy.
------------------------	---

Example file contents:

```
# ./contexts/default_type
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_default_type_path
get_default_type
```

3.4.11 contexts/failsafe_context File

The **failsafe_context**(5) is used when a login process cannot determine a default context to use. The file contents will then be used to allow an administrator access to the system.

The file format is as follows:

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

Where:

<code>role:type[:range]</code>	A single line that has a valid context to allow an administrator access to the system, including the MLS / MCS level or range if applicable.
--------------------------------	--

Example file contents:

```
# ./contexts/failsafe_context - Taken from the targeted policy.
unconfined_r:unconfined_t
```

```
# ./contexts/failsafe_context - Taken from the MLS policy.
sysadm_r:sysadm_t:s0
```

Supporting libselinux API functions are:

```
selinux_context_path
selinux_failsafe_context_path
get_default_context
get_default_context_with_role
```

```
get_default_context_with_level
get_default_context_with_rolelevel
get_ordered_context_list
get_ordered_context_list_with_level
```

3.4.12 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:

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

Where:

<pre>user:role:type[:range]</pre>	The file contains one line that consists of a security context, including the MLS / MCS level or range if applicable.
-----------------------------------	---

Example file contents:

```
# ./contexts/initrc_context - Taken from the targeted policy.
system_u:system_r:initrc_t:s0
```

```
# ./contexts/initrc_context - Taken from the MLS 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
```

3.4.13 contexts/lxc_contexts File

This file supports labeling lxc containers within the libvirt library (see libvirt source `src/security/security_selinux.c`). This is similar to the [virtual_domain_context](#) and [virtual_image_context](#) used by libvirt qemu services.

The file format is as follows:

```
process = "security_context"
file = "security_context"
content = "security_context"
```

Where:

process	A single process entry that contains the lxc domain security context, including the MLS / MCS level or range if applicable.
file	A single file entry that contains the lxc file security context, including the MLS / MCS level or range if applicable.
content	A single content entry that contains the lxc content security context, including the MLS / MCS level or range if applicable.
sandbox_kvm_process sandbox_lxc_process	These entries may be present, however in F-20 they are not currently used.

Example file contents:

```
# ./contexts/lxc_contexts  
  
process = "system_u:system_r:svirt_lxc_net_t:s0"  
file = "system_u:object_r:svirt_sandbox_file_t:s0"  
content = "system_u:object_r:virt_var_lib_t:s0"
```

Supporting libselinux API functions are:

```
selinux_context_path  
selinux_lxc_context_path
```

3.4.14 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
```

3.4.15 contexts/removable_context File

The **removable_context** (5) file contains a single 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:

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

Where:

<code>user:role:type[:range]</code>	The file contains one line that consists of a security context, including the MLS / MCS level or range if applicable.
-------------------------------------	---

Example file contents:

```
# ./contexts/removable_contexts
system_u:object_r:removable_t:s0
```

Supporting libselinux API functions are:

```
selinux_removable_context_path
```

3.4.16 contexts/securetty_types File

The **securetty_types** (5) 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:

<code>type</code>	Zero or more type entries that are defined in the policy for tty devices.
-------------------	---

Example file contents:

```
# ./contexts/securetty_types
sysadm_tty_device_t
user_tty_device_t
staff_tty_device_t
```

Supporting libselinux API functions are:

```
selinux_securetty_types_path
```

3.4.17 contexts/sepgsql_contexts File

This file contains the default security contexts for SE-PostgreSQL database objects and is described in **selabel_db** (5).

The file format is as follows:

Each line within the database contexts file is as follows:

```
object_type  object_name  context
```

Where:

object_type	This is the string representation of the object type.
object_name	These are the object names of the specific database objects. The entry can contain '*' for wildcard matching or '?' for substitution. Note that if the '*' is used, then be aware that the order of entries in the file is important. The '*' on its own is used to ensure a default fallback context is assigned and should be the last entry in the <u>object_type</u> block.
context	The security context that will be applied to the object.

Example file contents:

```
# ./contexts/sepysql_contexts file
# object_type object_name context
db_database my_database system_u:object_r:my_sepysql_db_t:s0
db_database * system_u:object_r:sepysql_db_t:s0
db_schema *.* system_u:object_r:sepysql_schema_t:s0
```

3.4.18 contexts/systemd_contexts File

This file is not currently used in F-20 but seems to contain file contexts to be used by tasks run via **systemd**(8) in a later release. There are some patches in the systemd mail archive that relate to this file.

The file format is as follows:

```
service_class = security_context
```

Where:

service_class	One or more entries that relate to the systemd service (e.g. runtime, transient).
security_context	The security context, including the MLS / MCS level or range if applicable of the service to be run.

Example file contents:

```
# ./contexts/systemd_contexts
runtime=system_u:object_r:systemd_runtime_unit_file_t:s0
```

Supporting `libselinux` API functions are:

```
selinux_context_path  
selinux_systemd_contexts_path
```

3.4.19 `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:

<code>security_context</code>	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 standard  
# reference policy.  
  
system_u:sysadm_r:sysadm_t
```

```
# ./contexts/userhelper_context - Taken from the MLS/MCS  
# reference policy.  
  
system_u:sysadm_r:sysadm_t:s0
```

Supporting `libselinux` API functions are:

```
selinux_context_path
```

3.4.20 `contexts/virtual_domain_context` File

The `virtual_domain_context(5)` file is used by the virtualization API (`libvirt`) and provides the `qemu` domain contexts available in the policy (see `libvirt` source `src/security/security_selinux.c`). There may be two entries in this file, with the second entry being an alternative domain context.

Example file contents:

```
# ./contexts/virtual_domain_context - From targeted policy.  
  
system_u:system_r:svirt_t:s0
```

Supporting `libselinux` API functions are:

```
selinux_virtual_domain_context_path
```

3.4.21 contexts/virtual_image_context File

The `virtual_image_context(5)` file is used by the virtualization API (libvirt) and provides the image contexts that are available in the policy (see libvirt source `src/security/security_selinux.c`). The first entry is the image file context and the second entry is the image content context.

Example file contents:

```
# ./contexts/virtual_image_context - From targeted policy.  
  
system_u:system_r:svirt_image_t:s0  
system_u:system_r:svirtcontent_t:s0
```

Supporting libselinux API functions are:

```
selinux_virtual_image_context_path
```

3.4.22 contexts/x_contexts File

The `x_contexts(5)` file provides the default security contexts for the X-Windows SELinux security extension. The usage is discussed in the [X-windows SELinux Support](#) section. The MCS / MLS version of the file has the appropriate level or range information added.

A typical entry is as follows:

```
# object_type object_name context  
selection PRIMARY system_u:object_r:clipboard_xselection_t
```

Where:

object_type	These are types of object supported and valid entries are: client, property, poly_property, extension, selection, poly_selection and events.
object_name	These are the object names of the specific X-server resource such as PRIMARY, CUT_BUFFER0 etc. They are generally defined in the X-server source code (<code>protocol.txt</code> and <code>BuiltInAtoms</code> in the <code>dix</code> directory of the <code>xorg-server</code> source package). This can contain '*' for 'any' or '?' for 'substitute' (see the <code>CUT_BUFFER?</code> entry where the '?' would be substituted for a number between 0 and 7 that represents the number of these buffers).

context	This is the security context that will be applied to the object. For MLS/MCS systems there would be the additional MLS label.
---------	---

Example file contents:

```
#
# Config file for XSELinux extension
#

### Rules for X Clients
# 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_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 _SELINUX_*     system_u:object_r:seclabel_xproperty_t

# Clipboard and selection properties
property CUT_BUFFER?    system_u:object_r:clipboard_xproperty_t

# Default fallback type
property *              system_u:object_r: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.
#
# Restricted extensions
extension SELinux       system_u:object_r:security_xextension_t

# Standard extensions
extension *             system_u:object_r:xextension_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 PRIMARY       system_u:object_r:clipboard_xselection_t
selection CLIPBOARD     system_u:object_r:clipboard_xselection_t

# Default fallback type
selection *            system_u:object_r: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
event X11:ButtonPress   system_u:object_r:input_xevent_t
event X11:ButtonRelease system_u:object_r:input_xevent_t
event X11:MotionNotify  system_u:object_r:input_xevent_t
event XInputExtension:DeviceKeyPress system_u:object_r:input_xevent_t
event XInputExtension:DeviceKeyRelease system_u:object_r:input_xevent_t
event XInputExtension:DeviceButtonPress system_u:object_r:input_xevent_t
event XInputExtension:DeviceButtonRelease system_u:object_r:input_xevent_t
event XInputExtension:DeviceMotionNotify system_u:object_r:input_xevent_t
event XInputExtension:DeviceValuator system_u:object_r:input_xevent_t
event XInputExtension:ProximityIn system_u:object_r:input_xevent_t
event XInputExtension:ProximityOut system_u:object_r:input_xevent_t
```

```
# Client message events
event X11:ClientMessage      system_u:object_r:client_xevent_t
event X11:SelectionNotify    system_u:object_r:client_xevent_t
event X11:UnmapNotify        system_u:object_r:client_xevent_t
event X11:ConfigureNotify    system_u:object_r:client_xevent_t

# Default fallback type
event *                       system_u:object_r:xevent_t
```

Supporting `libselinux` API functions are:

```
selinux_x_context_path
selabel_open
selabel_close
selabel_lookup
selabel_stats
```

3.4.23 `contexts/files/file_contexts File`

The `file_contexts(5)` file is managed by the `semodule(8)` and `semanage(8)` commands³⁹ 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.

There may also be a `file_contexts.bin` present that is built and used by `semanage(8)`. The format of this file conforms to the Perl compatible regular expression (PCRE) internal format.

Supporting `libselinux` API functions are:

```
selinux_file_context_path
selabel_open
selabel_close
selabel_lookup
selabel_stats
```

3.4.24 `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

³⁹ 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.

defined files to be labeled correctly. The `file_contexts(5)` man page also describes this file.

Supporting `libselinux` API functions are:

```
selinux_file_context_local_path
```

3.4.25 `contexts/files/file_contexts.homedirs` File

This file is managed by the `semodule(8)` and `semanage(8)` 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.

It is fully described in the [./modules/active/file_contexts.homedirs](#) file section. The `file_contexts(5)` man page also describes this file.

There may also be a `file_contexts.homedirs.bin` present that is built and used by `semanage(8)`. The format of this file conforms to the Perl compatible regular expression (PCRE) internal format.

Supporting `libselinux` API functions are:

```
selinux_file_context_homedir_path  
selinux_homedir_context_path
```

3.4.26 `contexts/files/file_contexts.subs` and `file_contexts.subs_dist` File

These files allow substitution of file names (`.subs` for local use and `.subs_dist` for GNU / Linux distributions use) for the `libselinux` functions `matchpatchcon(3)` and `selabel_lookup(3)`. The `file_contexts(5)` man page also describes this file.

The subs files contain a list of space separated path names such as:

```
/myweb /var/www  
/myspool /var/spool/mail
```

Then (for example), when `matchpatchcon(3)` or `selabel_lookup(3)` is passed a path `/myweb/index.html` the functions will substitute the `/myweb` component with `/var/www`, with the final result being:

```
/var/www/index.html
```

Supporting `libselinux` API functions are:

```
selinux_file_context_subs_path
```

```
selinux_file_context_subs_dist_path
selabel_lookup
matchpathcon
matchpathcon_index
```

3.4.27 contexts/files/media File

The **media** (5) file is 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: cdrom, floppy, disk and usb).
<code>file_context</code>	The context to be used for the device. Note that it does not have the MLS / MCS level).

Example file contents:

```
# contexts/files/media
# Note 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
```

3.4.28 contexts/users/[seuser_id] File

These optional files are named after the SELinux user they represent. Each file has the same format as the [contexts/default_contexts](#) file and is used to assign the correct context to the SELinux user (generally during login). The **user_contexts** (5) man page also describes these entries.

Example file contents:

```
# ./contexts/users/unconfined_u - From the targeted policy.

system_r:crond_t:s0          unconfined_r:unconfined_t:s0
system_r:initrc_t:s0        unconfined_r:unconfined_t:s0
system_r:local_login_t:s0   unconfined_r:unconfined_t:s0
```

```
system_r:remote_login_t:s0    unconfined_r:unconfined_t:s0
system_r:sshd_t:s0           unconfined_r:unconfined_t:s0
system_r:sysadm_su_t:s0      unconfined_r:unconfined_t:s0
system_r:unconfined_t:s0     unconfined_r:unconfined_t:s0
system_r:initrc_su_t:s0      unconfined_r:unconfined_t:s0
unconfined_r:unconfined_t:s0 unconfined_r:unconfined_t:s0
system_r:xdm_t:s0            unconfined_r:unconfined_t:s0
```

Supporting libselinux API functions are:

```
selinux_user_contexts_path
selinux_users_path
selinux_usersconf_path
get_default_context
get_default_context_with_role
get_default_context_with_level
get_default_context_with_rolelevel
get_ordered_context_list
get_ordered_context_list_with_level
```

3.4.29 logins/<linuxuser_id> File

These optional files are used by SELinux-aware login applications such as PAM (using the `pam_selinux` module) to obtain an SELinux user name and level based on the GNU / Linux login id and service name. It has been implemented for SELinux-aware applications such as FreeIPA (Identity, Policy Audit - see http://freeipa.org/page/Main_Page for details). The `service_seusers` (5) man page also describes these entries.

The file name is based on the GNU/Linux user that is used at log in time (e.g. `ipa`).

If `getseuser` (3) fails to find an entry, then the `seusers` file is used to retrieve default information.

The file format is as follows:

```
service_name:seuser_id:level
```

Where:

<code>service_name</code>	The name of the service.
<code>seuser_id</code>	The SELinux user name.
<code>level</code>	The run level

Example file contents:

```
# ./logins/ipa example entries

ipa_service:user_u:s0
another_service:unconfined_u:s0
```

Supporting `libselinux` API functions are:

```
getseuser
```

3.4.30 `users/local.users` File

Generally the `local.users` (5) file is not present if `semanage` (8) is being used to manage users, however if `semanage` is not being used then this file may be present (it could also be present in older Reference or Example policies).

The file would contain local user definitions in the form of user statements as defined in the [modules/active/users.local](#) section.

Note that if `SETLOCALDEFS` is set in the SELinux [config](#) file, then `selinux_mkload_policy(3)` will check for a `local.users` file in the `selinux_users_path(3)`, and a `booleans.local` file in the `selinux_booleans_path(3)`.

4. SELinux Policy Languages

4.1 Introduction

This section is intended as a reference to give a basic understanding of the kernel policy language statements and rules with supporting examples taken from the Reference Policy sources. Also all of the language updates to Policy DB version 29 should have been captured. For a more detailed explanation of the policy language the "SELinux by Example" [12] book is recommended.

There is currently a project underway called the Common Intermediate Language (CIL) project that defines a new policy definition language that has an overview of its motivation and design at: <https://github.com/SELinuxProject/cil/wiki>, however some of the language statement definitions out of date. The CIL compiler source and language reference guide can be found at: <https://github.com/SELinuxProject/cil.git> and cloned via:

```
git clone https://github.com/SELinuxProject/cil.git
```

The CIL compiler language reference guide has examples for each type of statement and can be built in pdf or html formats, therefore this Notebook will not cover the CIL policy language (there is a pdf copy of the CIL Reference Guide in the Notebook tarball). There is a [migration programme](#) underway that will convert the Reference Policy to CIL via a high level language module that is discussed in the [Policy Store Migration](#) section. Once migration is complete, the CIL compiler will be in available the `libsepol` library and CIL modules will be compiled with an updated `semodule` (8) command as follows:

```
# Compile and install an updated module written in CIL:
semodule -s modular-test --priority 400 -i custom/int_gateway.cil
```

Note that any source policy file name with the `.cil` extension will automatically be built as a CIL module.

4.1.1 CIL Overview

While the CIL design web pages give the main objectives of CIL, from a language perspective it will:

- a) Apply name and usage consistency to the current kernel language statements. For example the kernel language uses `attribute` and `attribute_role` to declare identifiers, whereas CIL uses `typeattribute` and `roleattribute`. Also statements to associate types or roles have been made consistent and enhanced to allow expressions to be defined.

Examples:

Kernel	CIL
<code>attribute</code>	<code>typeattribute</code>
<code>typeattribute</code>	<code>typeattributest</code>
<code>attribute_role</code>	<code>roleattribute</code>

roleattribute	roleattributeset
allow	allow
allow (role)	roleallow
dominance	sensitivityorder

b) Additional CIL statements have been defined to enhance functionality:

`classpermission` - Declare a `classpermissionset` identifier.

`classpermissionset` - Associate class / permissions also supporting expressions.

`classmap` / `classmapping` - Statements to support declaration and association of multiple `classpermissionset`'s. Useful when defining an `allow` rule with multiple class/permissions.

`context` - Statement to declare security context.

c) Allow named and anonymous definitions to be supported.

d) Support namespace features allowing policy modules to be defined within blocks with inheritance and template features.

e) Remove the order dependency in that policy statements can be anywhere within the source (i.e. remove dependency of class, sid etc. being within a base module).

f) Able to define macros and calls that will remove any dependency on M4 macro support.

g) Directly generate the binary policy file and other configuration files - currently the `file_contexts` file.

h) Support transformation services such as `delete`, `transform` and `inherit` with exceptions.

An simple CIL policy is as follows:

```
; These CIL statements declare a user, role, type and range of:
;   unconfined.user:unconfined.role:unconfined.process:s0-s0
;
; A CIL policy requires at least one 'allow' rule and sid to be declared
; before a policy will build.
;

(handleunknown allow)
(mls true)
(policycap open_perms)

(category c0)
(categoryorder (c0))
(sensitivity s0)
(sensitivityorder (s0))
(sensitivitycategory s0 (c0))
(level systemLow (s0))
(levelrange low_low (systemLow systemLow))

(sid kernel)
(sidorder (kernel))
(sidcontext kernel unconfined.sid_context)

(classorder (file))
(class file (read write open getattr))
```

```
; Define object_r role. This must be assigned in CIL.
(role object_r)

; The unconfined namespace:
(block unconfined
  (user user)
  (userrange user (systemLow systemLow))
  (userlevel user systemLow)
  (userrole user role)

  (role role)

  (type process)
  (roletype object_r process)
  (roletype role process)
  ; Define a SID context:
  (context sid_context (user role process low_low))

  (type object)
  (roletype object_r object)

  ; An allow rule:
  (allow process object (file (read)))
)
```

There are CIL examples in the Notebook source tarball with a utility that will produce a base policy in either the kernel policy language or CIL (`notebook-tools/build-sepolicy`). The only requirement is that the `initial_sids`, `security_classes` and `access_vectors` files from the Reference policy are required, although the F-20 versions are supplied in the `basic-policy/policy-files/flask-files` directory.

```
Usage: build-sepolicy [-k] [-M] [-c|-i|-p|-s] -d flask_directory -o output_file

-k Output kernel classes only (exclude # userspace entries in the
  security_classes file).
-M Output an MLS policy.
-c Output a policy in CIL language (otherwise generate a kernel policy
  language policy).
-p Output a file containing class and classpermissionsets + their order
  for use by CIL policies.
-s Output a file containing initial SIDs + their order for use by
  CIL policies.
-i Output a header file containing class/permissions for use by
  selinux_set_mapping(3).
-o The output file that will contain the policy source or header file.
-d Directory containing the initial_sids, security_classes and
  access_vectors Flask files.
```

4.2 Kernel Policy Language

4.2.1 Policy Source Files

There are three basic types of policy source file⁴⁰ that can contain language statements and rules. The three types of policy source file⁴¹ are:

⁴⁰ It is important to note that the Reference Policy builds policy using makefiles and m4 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.

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` (8) 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` (8) or `checkmodule` (8) 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 14](#) shows the order in which the statements should appear in source files with the mandatory statements that must be present.

⁴¹ This does not include the `'file_contexts'` file as it does not contain policy statements, only default security contexts (labels) that will be used to label 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
Default user, role, type, range rules	o		
Type / Type Alias	m	Type / Type Alias	o
Roles	m	Roles	o
Policy Rules	m	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 14: Base and Module Policy Statements - *There must be at least one of each of the mandatory statements, plus at least one allow rule in a policy to successfully build.*

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 16](#) 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 16](#) 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 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. All statements available to policy version 29 have been included.
4. When multiple source and target entries are shown in a single statement or rule, the compiler (**checkpolicy** (8) or **checkmodule** (8)) 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**(8), **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 15](#) lists words reserved for the SELinux policy language.

alias	allow	and
attribute	attribute_role	auditallow
auditdeny	bool	category
cfalse	class	clone
common	constrain	ctrue
dom	domby	dominance
dontaudit	else	equals
false	filename	filesystem
fscon	fs_use_task	fs_use_trans
fs_use_xattr	genfscon	h1
h2	identifier	if
incomp	inherits	iomemcon
ioportcon	ipv4_addr	ipv6_addr
l1	l2	level
mlsconstrain	mlsvalidatetrans	module
netifcon	neverallow	nodecon
not	notequal	number
object_r	optional	or
path	pcidevicecon	permissive
pirqcon	polycycon	portcon
r1	r2	r3
range	range_transition	require
role	roleattribute	roles
role_transition	sameuser	sensitivity
sid	source	t1
t2	t3	target

true	type	typealias
typeattribute	typebounds	type_change
type_member	types	type_transition
u1	u2	u3
user	validatetrans	version_identifier
xor	default_user	default_role
default_type	default_range	low
high	low_high	

Table 15: Policy language reserved words.

7. [Table 16](#) 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
attribute_role	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
default_user	Yes	Yes	No	No	No	No
default_role	Yes	Yes	No	No	No	No
default_type	Yes	Yes	No	No	No	No
default_range	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

⁴² Only the statement keyword is allowed.

Statement / Rule	Monolithic Policy	Base Policy	Module Policy	Conditional Statements	optional Statement	require Statement
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
polycycap	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
role	Yes	Yes	Yes	No	Yes	Yes
roleattribute	Yes	Yes	Yes	No	Yes	No
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
typebounds	Yes	Yes	Yes	No	Yes	No
user	Yes	Yes	Yes	No	Yes	Yes
validatetrans	Yes	Yes	No	No	No	No

Table 16: 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 Section Contents

The policy language statement and rule sections are as follows:

- a) [Policy Configuration Statements](#)
- b) [Default Object Rules](#)
- c) [User Statements](#)

⁴³ neverallow statements are allowed in modules, however to detect these the `semanage.conf` file must have the `expand-check=1` entry present.

⁴⁴ Only if preceded by the `optional` statement.

⁴⁵ Only if preceded by the `optional` statement.

- d) [Role Statements](#)
- e) [Type Statements](#)
- f) [Bounds Rules](#)
- g) [Access Vector Rules](#)
- h) [Object Class and Permission Statements](#)
- i) [Conditional Policy Statements](#)
- j) [Constraint Statements](#)
- k) [MLS Statements](#)
- l) [Security ID \(SID\) Statement](#)
- m) [File System Labeling Statements](#)
- n) [Network Labeling Statements](#)
- o) [Modular Policy Support Statements](#)
- p) [XEN Statements](#)

4.3 Policy Configuration Statements

4.3.1 `policycap`

Policy version 22 introduced the `policycap` statement to allow new capabilities to be enabled or disabled in the kernel via policy in a backward compatible way. For example policies that are aware of a new capability can enable the functionality, while older policies would continue to use the original functionality. An example is shown in the [SELinux Networking Support](#) section using the `network_peer_controls` capability.

In the 3.14 kernel there are four policy capabilities configured as shown in the [SELinux Filesystem](#) section.

The statement definition is:

```
policycap capability;
```

Where:

<code>policycap</code>	The <code>policycap</code> keyword.
<code>capability</code>	A single <code>capability</code> identifier that will 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;
```

4.4 Default Object Rules

These rules allow a default user, role, type and/or range to be used when computing a context for a new object. These require policy version 27 or 28 with kernels 3.5 or greater.

4.4.1 default_user

Allows the default user to be taken from the source or target context when computing a new context for an object of the defined class. Requires policy version 27.

The statement definition is:

```
default_user class default;
```

Where:

default_user	The default_user rule keyword.
class	One or more class identifiers. Multiple entries consist of a space separated list enclosed in braces ({}). Entries can be excluded from the list by using the negative operator (-).
default	A single keyword consisting of either source or target that will state whether the default user should be obtained from the source or target context.

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:

```
# When computing the context for a new file object, the user
# will be obtained from the target context.
default_user file target;
```

```
# When computing the context for a new x_selection or x_property
# object, the user will be obtained from the source context.
default_user { x_selection x_property } source;
```

4.4.2 default_role

Allows the default role to be taken from the source or target context when computing a new context for an object of the defined class. Requires policy version 27.

The statement definition is:

```
default_role class default;
```

Where:

default_role	The default_role rule keyword.
class	One or more class identifiers. Multiple entries consist of a space separated list enclosed in braces ({}). Entries can be excluded from the list by using the negative operator (-).
default	A single keyword consisting of either source or target that will state whether the default role should be obtained from the source or target context.

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:

```
# When computing the context for a new file object, the role
# will be obtained from the target context.
default_role file target;
```

```
# When computing the context for a new x_selection or x_property
# object, the role will be obtained from the source context.
default_role { x_selection x_property } source;
```

4.4.3 default_type

Allows the default type to be taken from the source or target context when computing a new context for an object of the defined class. Requires policy version 28.

The statement definition is:

```
default_type class default;
```

Where:

default_type	The default_type rule keyword.
class	One or more class identifiers. Multiple entries consist of a space separated list enclosed in braces ({}). Entries can be excluded from the list by using the negative operator (-).
default	A single keyword consisting of either source or target that will state whether the default type should be obtained from the source or target context.

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:

```
# When computing the context for a new file object, the type
# will be obtained from the target context.
default_type file target;
```

```
# When computing the context for a new x_selection or x_property
```

```
# object, the type will be obtained from the source context.
default_type { x_selection x_property } source;
```

4.4.4 default_range

Allows the default range or level to be taken from the source or target context when computing a new context for an object of the defined class. Requires policy version 27.

The statement definition is:

```
default_range class default range;
```

Where:

default_range	The default_range rule keyword.
class	One or more class identifiers. Multiple entries consist of a space separated list enclosed in braces ({}). Entries can be excluded from the list by using the negative operator (-).
default	A single keyword consisting of either source or target that will state whether the default level or range should be obtained from the source or target context.
range	A single keyword consisting of either: low, high or low_high that will state what part of the range should be used.

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:

```
# When computing the context for a new file object, the lower
# level will be taken from the target context range.
default_range file target low;
```

```
# When computing the context for a new x_selection or x_property
# object, the range will be obtained from the source context.
default_type { x_selection x_property } source low_high;
```

4.5 User Statements

4.5.1 user

The `user` statement declares an SELinux user identifier within the policy and associates it 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> or <code>attribute_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 level that has been previously declared with a <code>level</code> statement. Note that the compiler only accepts the sensitivity component of the level (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
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;
```

4.6 Role Statements

Policy version 26 introduced two new role statements aimed at replacing the role dominance rule by making role relationships easier to understand. These new statements: `attribute_role` and `roleattribute` are defined in this section with examples.

4.6.1 role

The `role` statement either declares a role identifier or 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 to declare a role is:

```
role role_id;
```

The statement definition to associate a role to one or more types is:

```
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> , <code>typealias</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> , <code>typealias</code> or

	attribute 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:

```
# Declare the roles:

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 types 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;
```

4.6.2 attribute_role

The `attribute_role` statement declares a role attribute identifier that can then be used to refer to a group of roles.

The statement definition is:

```
attribute_role attribute_id;
```

Where:

<code>attribute_role</code>	The <code>attribute_role</code> keyword.
<code>attribute_id</code>	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_role statement to declare attributes that
# can then refers to a list of roles. Note that there are no
# roles associated with them yet.

attribute_role role_list_1;
attribute_role srole_list_2;
```

4.6.3 roleattribute

The roleattribute statement allows the association of previously declared roles to one or more previously declared attribute_roles.

The statement definition is:

```
roleattribute role_id attribute_id;
```

Where:

roleattribute	The roleattribute keyword.
role_id	The identifier of a previously declared role.
attribute_id	One or more previously declared attribute_role 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 roleattribute statement to associate a previously
# declared role of service_r to a previously declared
# role_list_1 attribute_role.

attribute_role role_list_1;
role service_r;

# The association using the roleattribute statement:
```

```
roleattribute service_r role_list_1;
```

4.6.4 allow

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.

Note that the role `allow` rule has the same keyword as the `allow AV` rule.

The statement definition is:

```
allow from_role_id to_role_id;
```

Where:

<code>allow</code>	The role <code>allow</code> rule keyword.
<code>from_role_id</code>	One or more <code>role</code> or <code>attribute_role</code> identifiers that identify the current role. Multiple entries consist of a space separated list enclosed in braces (<code>{ }</code>).
<code>to_role_id</code>	One or more <code>role</code> or <code>attribute_role</code> identifiers that identify the new role to be granted on the transition. Multiple entries consist of a space separated list enclosed in braces (<code>{ }</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	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;
```

4.6.5 role_transition

The `role_transition` rule specifies that a role transition is required, and if allowed, the process will run under the new role. From policy version 25, the class can now be defined.

The statement definition is:

```
role_transition current_role_id type_id new_role_id;
```

Or from Policy version 25:

```
role_transition current_role_id type_id : class new_role_id;
```

Where:

role_transition	The role_transition keyword.
current_role_id	One or more role or attribute_role identifiers that identify the current role. Multiple entries consist of a space separated list enclosed in braces ({}).
type_id	One or more type, typealias 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 (-).
class	For policy versions >= 25 an object class that applies to the role transition. If omitted defaults to the process object class.
new_role_id	A single role identifier that will become the new role.

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;
```

4.6.6 dominance

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.

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, note however that it
# has been deprecated and should not be used.

dominance { role message_filter_r { role unconfined_r };}
```

4.7 Type 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
# generally ends with _type
```

4.7.1 type

The type statement declares the type identifier and any optional associated alias or attribute identifiers. Type identifiers are a component of the [Security Context](#).

The statement definition is:

```
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 that have been previously declared by the typealias statement. 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
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;
```

4.7.2 attribute

An `attribute` statement declares an identifier that can then be used to refer to a group of type identifiers.

The statement definition is:

```
attribute attribute_id;
```

Where:

<code>attribute</code>	The <code>attribute</code> keyword.
<code>attribute_id</code>	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;
```

4.7.3 typeattribute

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;
```

Where:

<code>typeattribute</code>	The <code>typeattribute</code> keyword.
<code>type_id</code>	The identifier of a previously declared type.
<code>attribute_id</code>	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;
```

4.7.4 typealias

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 };
```

4.7.5 permissive

Policy 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** (8) 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
```

4.7.6 type_transition

The type_transition rule specifies the default type to be used for domain transition or object creation. Kernels from 2.6.39 with Policy versions from 25 also support the 'name transition rule' extension. See the [Computing Security Contexts](#)

section for more details. Note than an `allow` rule must be used to authorise the transition.

The statement definitions are:

```
type_transition source_type target_type : class default_type;
```

Policy versions 25 and above also support a 'name transition' rule however, this is only appropriate for the file classes:

```
type_transition source_type target_type : class default_type object_name;
```

Where:

<code>type_transition</code>	The <code>type_transition</code> rule keyword.
<code>source_type</code> <code>target_type</code>	One or more source / target type, typealias 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 or typealias identifier that will become the default process type for a domain transition or the type for object transitions.
<code>object_name</code>	For the 'name transition' rule this is matched against the objects name (i.e. the last component of a path). If <code>object_name</code> exactly matches the object name, then use <code>default_type</code> for the type.

The statement is valid in:

<u>Monolithic Policy</u>	<u>Base Policy</u>	<u>Module Policy</u>
Yes	Yes	Yes
<u>Conditional Policy (if) Statement</u>	<u>optional Statement</u>	<u>require Statement</u>
Yes	Yes	No

Example - Domain Transition:

```
# Using the type_transition statement to show a domain
# transition (as the statement has the process object 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 acc_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 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
# policy has:
#
allow acct_t wtmp_t:file { create open getattr setattr read
    write append rename link unlink ioctl lock };
```

Example - Name Transition:

```
# type_transition to allow using the last path component as
# part of the information in making labeling decisions for
# new objects. An example rule:
#
type_transition unconfined_t etc_t : file system_conf_t eric;

# This rule says if unconfined_t creates a file in a directory
# labeled etc_t and the last path component is "eric" (must be
# an exact strcmp) it should be labeled system_conf_t.
```

4.7.7 type_change

The `type_change` rule specifies a default type when relabeling an existing object. For example userspace SELinux-aware applications would use `security_compute_relabel(3)` and `type_change` rules in policy to determine the new context to be applied. Note that an `allow` rule must be used to authorise access. See the [Computing Security Contexts](#) section for more details.

The statement definition is:

```
type_change source_type target_type : class change_type;
```

Where:

<code>type_change</code>	The <code>type_change</code> rule keyword.
<code>source_type</code> <code>target_type</code>	One or more source / target type, typealias 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>change_type</code>	A single type or typealias identifier that will become the new type.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
Yes	Yes	No

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;
```

4.7.8 type_member

The `type_member` rule specifies a default type when creating a polyinstantiated object. For example a userspace SELinux-aware application would use `avc_compute_member(3)` or `security_compute_member(3)` with `type_member` rules in policy to determine the context to be applied. Note that an allow rule must be used to authorise access. See the [Computing Security Contexts](#) section for more details.

The statement definition is:

```
member_type source_type target_type : class member_type;
```

Where:

<code>type_member</code>	The <code>type_member</code> rule keyword.
<code>source_type</code> <code>target_type</code>	One or more source / target type, typealias 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>member_type</code>	A single type or typealias identifier that will become the polyinstantiated type.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
Yes	Yes	No

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;
```

4.8 Bounds Rules

Bounds handling was added in version 24 of the policy and consisted of adding userbounds, rolebounds and typebounds information to the policy. However only the typebounds rule is currently implemented by

checkpolicy (8) and **checkmodule** (8) with kernel support from 2.6.28. The CIL compiler does support `userbounds` and `rolebounds` but these are resolved at policy compile time, not via the kernel at run-time.

4.8.1 typebounds

The `typebounds` rule was added in version 24 of the policy. This defines a hierarchical relationship between domains where the bounded domain cannot have more permissions than its bounding domain (the parent). It requires kernel 2.6.28 and above to control the security context associated to threads in multi-threaded applications.

The statement definition is:

```
typebounds bounding_domain bounded_domain;
```

Where:

<code>typebounds</code>	The <code>typebounds</code> keyword.
<code>bounding_domain</code>	The <code>type</code> or <code>typealias</code> identifier of the parent domain.
<code>bounded_domain</code>	One or more <code>type</code> or <code>typealias</code> identifiers of the child domains. 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

Example:

```
# This example states that:
# The httpd_child_t cannot have file:{write} due to lack of
# permissions on httpd_t which is the parent. It means the
# child domains will always have equal or less privileges
# than the parent.

# The typebounds statement:
typebounds httpd_t httpd_child_t;

# The parent is allowed file 'getattr' and 'read':
allow httpd_t etc_t : file { getattr read };

# However the child process has been given 'write' access that
# will not be allowed by the kernel SELinux security server.
allow httpd_child_t etc_t : file { read write };
```

4.9 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> <code>target_type</code>	<p>One or more source / target type, <code>typealias</code> or attribute identifiers. 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>).</p> <p>The <code>target_type</code> can have the <code>self</code> keyword instead of <code>type</code>, <code>typealias</code> or attribute identifiers. This means that the <code>target_type</code> is the same as the <code>source_type</code>.</p> <p>The <code>neverallow</code> rule also supports the wildcard operator (<code>*</code>) to specify that all types are to be included and the complement operator (<code>~</code>) to specify all types are to be included except those explicitly listed.</p>
<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>	<p>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 (<code>{ }</code>).</p> <p>The optional wildcard operator (<code>*</code>) specifies that all permissions for the object class can be used.</p> <p>The complement operator (<code>~</code>) is used to specify all permissions except those explicitly listed (although the compiler issues a warning if the <code>dontaudit</code> rule has <code>'~'</code>).</p>

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

4.9.1 allow

The allow rule checks whether the operations between the `source_type` and `target_type` are allowed for the class and permissions defined. 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;
```

4.9.2 dontaudit

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;
```

4.9.3 auditallow

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;
```

4.9.4 neverallow

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`⁴⁶ 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:
```

⁴⁶ `neverallow` statements are allowed in modules, however to detect these the `semanage.conf` file must have the `expand-check=1` entry present.

```
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;
```

4.10 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.

A list of object classes used by the Reference Policy can be found in the `./policy/flask/security_classes` file.

There are two variants of the class statement for writing policy:

1. There is the `class` statement that declares the actual class identifier or name.
2. There is a further refinement of the `class` statement that associates permissions to the class as discussed in the [Associating Permissions to a Class](#) section.

4.10.1 class

Object classes are declared within a policy as follows:

The statement definition is:

```
class class_id
```

Where:

<code>class</code>	The <code>class</code> keyword.
<code>class_id</code>	The <code>class</code> 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
```

4.10.2 Associating Permissions to a Class

Permissions can be defined within policy in two ways:

1. Define a set of `common` permissions that can then be inherited by one or more object classes using further `class` statements.
2. 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).

A list of classes and their permissions used by the Reference Policy can be found in the `./policy/flask/access_vectors` file.

4.10.3 `common`

Declare a common identifier and associate one or more common permissions.

The statement definition is:

```
common common_id { perm_set }
```

Where:

<code>common</code>	The <code>common</code> keyword.
<code>common_id</code>	The common 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 }
```

4.10.4 `class`

Inherit and / or associate permissions to a perviously declared class identifier.

The statement definition is:

```
class class_id [ inherits common_set ] [ { perm_set } ]
```

Where:

class	The class keyword.
class_id	The previously declared class identifier.
inherits	The optional inherits keyword that allows a set of common permissions to be inherited.
common_set	A previously declared common identifier.
perm_set	One or more optional permission identifiers in a space separated list enclosed within braces ({}).

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
# Associating Permissions to a Class 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 }
```

4.11 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 16](#) 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 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
```

4.11.1 `bool`

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:

bool	The bool keyword.
bool_id	The boolean identifier.
default_value	Either true or false.

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;
```

4.11.2 if

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 (except 'file_name_transition'), type_change and
require.
```

The statement definition is:

```
if (conditional_expression) { true_list } [ else { false_list } ]
```

Where:

if	The if keyword.
----	-----------------

conditional_expression	<p>One or more <code>bool_name</code> identifiers that have been previously defined by the bool Statement. Multiple identifiers must be separated by the following logical operators: <code>&&</code>, <code> </code>, <code>^</code>, <code>!</code>, <code>==</code>, <code>!=</code>.</p> <p>The <code>conditional_expression</code> is enclosed in brackets <code>()</code>.</p>
true_list	<p>A list of rules enclosed within braces <code>{ }</code> that will be executed when the <code>conditional_expression</code> is <code>'true'</code>.</p> <p>Valid statements and rules are highlighted within each language definition statement.</p>
else	Optional <code>else</code> keyword.
false_list	<p>A list of rules enclosed within braces <code>{ }</code> that will be executed when the optional <code>'else'</code> keyword is present and the <code>conditional_expression</code> is <code>'false'</code>.</p> <p>Valid statements and rules are highlighted within each language definition statement.</p>

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:

```
# 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 };
    ...
}
```

4.12 Constraint Statements

4.12.1 constrain

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:
	<pre>(expression : expression) not expression expression and expression expression or expression u1 op u2 r1 role_op r2</pre>

	<pre> t1 op t2 u1 op names u2 op names r1 op names r2 op names t1 op names t2 op names </pre>
<p>Where: u1, r1, t1 = Source user, role, type u2, r2, t2 = Target user, role, type</p> <p>and: op : == != role_op : == != eq dom domby incomp names : name { name_list } name_list : name name_list name</p>	

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
);
```

4.12.2 validatetrans

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 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>(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:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Example:

```
validatetrans { file } { t1 == unconfined_t );
```

4.12.3 mlsconstrain

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>perm_set</code>	One or more permissions. Multiple entries consist of a space separated list enclosed in braces {}.
<code>expression</code>	The boolean expression of the constraint that is defined as follows:
	<pre>(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</pre>

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. 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 low security level is dominated by the targets
# low 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 high security
# level is dominated by the targets low 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 ));
```

4.12.4 mlsvalidatetrans

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:
	<pre>(expression : expression) not expression and (expression and expression or 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</pre>

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 ))));
```

4.13 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 17](#). Depending on the circumstances, there can be one level defined or a [range](#) as shown in [Table 17](#).

Security Level (or Level)	Note that SELinux uses <code>level</code> , <code>sensitivity</code> and <code>category</code> in the language statements, however when discussing these the following terms can also be used: labels, classifications, and compartments.	
Consisting of a <code>sensitivity</code> and zero or more <code>category</code> entries:		
<code>sensitivity [: category, ...]</code>		
← 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
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 17: 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(8)` command will allow this mapping to be defined as discussed in the [./setrans.conf file](#) section.

4.13.1 sensitivity

The `sensitivity` statement defines the MLS policy sensitivity identifies and optional `alias` identifiers.

The statement definition is:

```
sensitivity sens_id [alias sensitivityalias_id ...];
```

Where:

<code>sensitivity</code>	The <code>sensitivity</code> keyword.
<code>sens_id</code>	The <code>sensitivity</code> identifier.
<code>alias</code>	The optional <code>alias</code> keyword.
<code>sensitivityalias_id</code>	One or more <code>sensitivityalias</code> 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 dominance

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 { sensitivity_id ... }
```

Where:

dominance	The dominance keyword.
sensitivity_id	A space separated list of previously declared sensitivity or sensitivityalias identifiers 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

The `category` statement defines the MLS policy category identifiers⁴⁷ and optional alias identifiers.

The statement definition is:

```
category category_id [alias categoryalias_id ...];
```

Where:

<code>category</code>	The <code>category</code> keyword.
<code>category_id</code>	The category identifier.
<code>alias</code>	The optional <code>alias</code> keyword.
<code>categoryalias_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 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

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 sensitivity_id [ :category_id ];
```

Where:

⁴⁷ 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.

level	The level keyword.
sensitivity_id	A previously declared sensitivity or sensitivityalias identifier.
category_id	An optional set of zero or more previously declared category or categoryalias identifiers that are preceded by a colon (:), that can be written as follows: <ul style="list-style-type: none"> The period (.) separating two category identifiers means an inclusive set (e.g. c0.c16). The comma (,) separating two category identifiers means a non-contiguous list (e.g. c21, c36, c45). Both separators may be used (e.g. c0.c16, c21, c36, c45).

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

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_type target_type new_range;
```

or (for policy version 21 and greater):

```
range_transition source_type target_type : class new_range;
```

Where:

range_transition	The range_transition keyword.
source_type target_type	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 (-).
class	The optional object class keyword (this allows policy versions 21 and greater to specify a class other than the default of process).
new_range	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[ - high_level ]
```

Where:

<code>low_level</code>	The processes lowest level identifier that has been previously declared by a level statement. If a <code>high_level</code> is not defined, then it is taken as the same as the <code>low_level</code> .
<code>-</code>	The optional hyphen (-) separator if a <code>high_level</code> is also being defined.
<code>high_level</code>	The processes highest level identifier that has been previously declared by a level statement.

4.13.6 mlsconstrain

This is described in the [Constraints](#) section.

4.13.7 mlsvaildatetrans

This is described in the [Constraints](#) section.

4.14 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 associate an initial security context to the SID, this is used when SELinux initialises but the policy has not yet been activated or as a default context should an object have an invalid label.

4.14.1 sid

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.

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
```

4.14.2 sid context

The `sid` context statement is used to associate an initial security context to the SID.

```
sid sid_id context
```

Where:

<code>sid</code>	The <code>sid</code> keyword.
<code>sid_id</code>	The previously declared <code>sid</code> identifier.
<code>context</code>	The initial security context.

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:

```
# This is from a targeted policy:

sid unlabeled
...
sid unlabeled system_u:object_r:unlabeled_t
```

```
# This 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
...
sid unlabeled system_u:object_r:unlabeled_t:s15:c0.c255
```

4.15 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.

4.15.1 `fs_use_xattr`

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. Example 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>xf</code> s.
<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:s0;
fs_use_xattr ext2 system_u:object_r:fs_t:s0;
fs_use_xattr ext3 system_u:object_r:fs_t:s0;
```

4.15.2 fs_use_task

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. Example 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:s0;
fs_use_task pipefs system_u:object_r:fs_t:s0;
fs_use_task sockfs system_u:object_r:fs_t:s0;
```

4.15.3 fs_use_trans

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:

fs_use_trans	The fs_use_trans keyword.
fs_name	Filesystem name that supports transition rules. Example names are: mqueue, shm, tmpfs and devpts.
fs_context	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: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;
```

4.15.4 genfscon

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:

genfscon	The genfscon keyword.
fs_name	The filesystem name.

partial_path	If fs_name is proc, then the partial path (see the examples). For all other types, this must be '/'.
fs_context	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

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
```

4.16 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 MLS 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
```

4.16.1 IP Address Formats

4.16.1.1 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
```

4.16.1.2 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:

```
::
```

4.16.2 netifcon

The `netifcon` statement is used to label network interface objects (e.g. `eth0`).

It is also possible to use the `'semanage interface'` command to associate the interface to a security context.

The statement definition is:

```
netifcon netif_id netif_context packet_context
```

Where:

netifcon	The netifcon keyword.
netif_id	The network interface name (e.g. eth0).
netif_context	The security context allocated to the network interface.
packet_context	The security context allocated packets. Note that these are defined but currently unused. The iptable SECMARK services should be used to label packets.

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
# 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 netif_t eth2
```

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 eth2 system_u:object_r:netif_t:s0 system_u:object_r:netif_t:s0
```

4.16.3 nodecon

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 nodecon keyword.
subnet	The subnet or specific IP address in IPv4 or IPv6 format. Note that the subnet and netmask values are used to ensure that the node_context 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 MLS policy nodecon statement using an IPv4 address:
nodecon 127.0.0.1 255.255.255.255 system_u:object_r:lo_node_t:
s0 - s15:c0.c255
```

```
# The MLS policy nodecon statement for the multicast address
# 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 node_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.

nodecon ipv4 127.0.0.2 255.255.255.255 system_u:object_r:node_t:s0
```

4.16.4 portcon

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 MLS 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 reserved_port_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:reserved_port_t:s0
```

4.17 Modular Policy Support Statements

This section contains language statements used to support policy modules.

4.17.1 **module**

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:

<code>module</code>	The module keyword.
<code>module_name</code>	The module name.
<code>version_number</code>	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;
```

4.17.2 **require**

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:

require	The <code>require</code> keyword.
require_list	<p>One or more specific statement keywords with their required identifiers in a semi-colon (;) separated list enclosed within braces ({}).</p> <p>The valid statement keywords are:</p> <ul style="list-style-type: none"> • <code>role</code>, <code>type</code>, <code>attribute</code>, <code>user</code>, <code>bool</code>, <code>sensitivity</code> and <code>category</code>. The keyword is followed by one or more identifiers in a comma (,) separated list, with the last entry being terminated with a semi-colon (;). • <code>class</code>. 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 (;). <p>The examples below show these in detail.</p>

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 preceded 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 setseccparam 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 };
}
```

4.17.3 optional

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 } [ 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 16 .
<code>else</code>	An optional <code>else</code> keyword.
<code>rule_list</code>	As the <code>rule_list</code> 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 dhcpd_t ping_exec_t:file { getattr read execute };
    .....
    require {
        type netutils_t, netutils_exec_t;
    } # end require
    allow dhcpd_t netutils_exec_t:file { getattr read execute };
    .....
    type_transition dhcpd_t netutils_exec_t:process netutils_t;
    ...
} else {
    allow dhcpd_t self:capability setuid;
    .....
} # end optional
```

4.18 Xen Statements

Xen policy supports additional policy language statements: `iomemcon`, `ioportcon`, `pcdevicecon` and `pirqcon` that are discussed in the sections that follow.

To compile these additional statements using `semodule(8)`, ensure that the `semanage.conf(5)` file has the `policy-target=xen` entry.

4.18.1 `iomemcon`

The `sid` statement declares the actual SID identifier and is defined at the start of a policy source file.

The statement definition is:

```
iomemcon addr context;
```

Where:

iomemcon	The iomemcon keyword.
addr	The memory address to apply the context. This may also be a range that consists of a start and end address separated by a hyphen (-).
context	The security context to be applied.

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:

```
iomemcon 0xfebd9 system_u:object_r:nicP_t;  
iomemcon 0xfebe0-0xfebff system_u:object_r:nicP_t;
```

4.18.2 ioportcon

The sid statement declares the actual SID identifier and is defined at the start of a policy source file.

The statement definition is:

```
ioportcon port context;
```

Where:

ioportcon	The ioportcon keyword.
port	The port to apply the context. This may also be a range that consists of a start and end port number separated by a hyphen (-).
context	The security context to be applied.

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:

```
ioportcon 0xeac0 system_u:object_r:nicP_t;
ioportcon 0xecc0-0xecdf system_u:object_r:nicP_t;
```

4.18.3 pcidevicecon

The sid statement declares the actual SID identifier and is defined at the start of a policy source file.

The statement definition is:

```
pcidevicecon pci_id context;
```

Where:

pcidevicecon	The pcidevicecon keyword.
pci_id	The PCI identifier.
context	The security context to be applied.

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:

```
pcidevicecon 0xc800 system_u:object_r:nicP_t;
```

4.18.4 pirqcon

The sid statement declares the actual SID identifier and is defined at the start of a policy source file.

The statement definition is:

```
pirqcon irq context;
```

Where:

pirqcon	The pirqcon keyword.
irq	The interrupt request number.
context	The security context to be applied.

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:

```
pirqcon 33 system_u:object_r:nicP_t;
```

5. The Reference Policy

5.1 Introduction

The Reference Policy is now the standard policy source used to build GNU/Linux 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. Adding new modules to the build.
3. Installation as a full Reference Policy source or as Header files.
4. Impact of the migration process being used to convert compiled module files (*.pp) to CIL.
5. Modifying the configuration files to build new policies.
6. Explain the support macros.

5.2 Reference Policy Overview

Strictly speaking the 'Reference Policy' should refer to the policy taken from the master repository or the latest released version (see <https://github.com/TresysTechnology/refpolicy/wiki>). This is because most Linux distributors take a released version and then tailor it to their specific requirements, for example the Fedora distribution is built from the standard Reference Policy but modified and distributed by Red Hat as a source RPM, for example:

```
selinux-policy-3.12.1-179.fc20.src.rpm48
```

The master Reference Policy repository can be checked out using the following:

```
# Check out the core policy:
git clone https://github.com/TresysTechnology/refpolicy.git
cd refpolicy
# Add the contributed modules (policy/modules/contrib)
git submodule init
git submodule update
```

[Figure 5.1](#) shows the layout of the reference policy source tree, that once installed would be located at:

```
/etc/selinux/<NAME>/src/policy
```

Where the <NAME> entry is taken from the `build.conf` file as discussed in the [Reference Policy Build Options - build.conf](#) section. The [Installing and Building the Reference Policy Source](#) section explains a simple build plus information on building the Fedora source.

⁴⁸ These RPMs can be obtained from <http://koji.fedoraproject.org>.

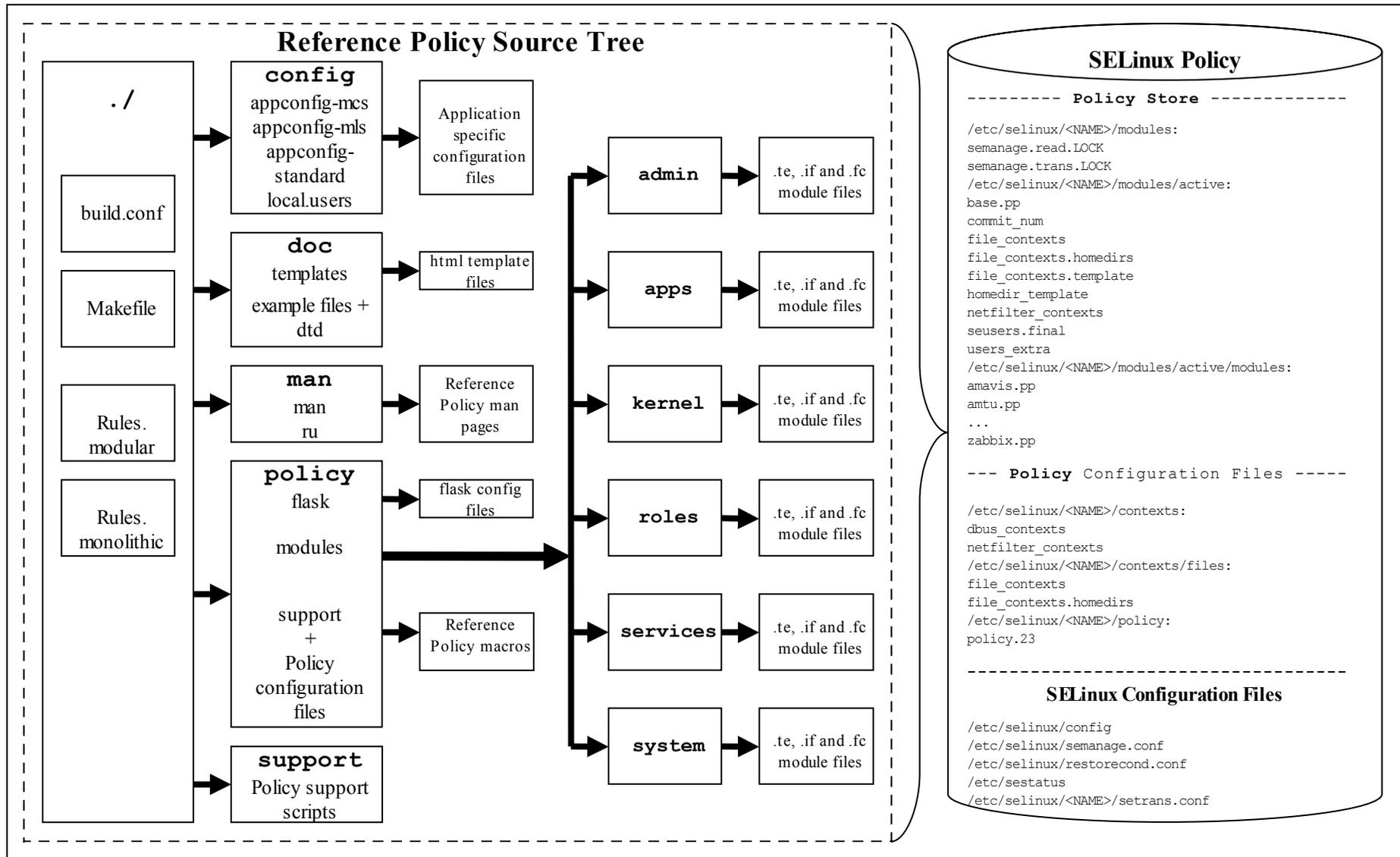


Figure 5.1: The Reference Policy Source Tree - When building a modular policy, files are added to the policy store. For monolithic builds the policy store is not used.

The Reference Policy can be used to build two different formats of policy infrastructure:

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 required⁴⁹. 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 and does not require the services of the module infrastructure (**semanage** (8) or **semodule** (8)). These are more suitable for embedded or minimal systems.

Each of the policy types are built using module files that define the specific rules required by the policy as detailed in the [Reference Policy Module Files](#) section. Note that the monolithic policy is built using the the same module files by forming a single 'base' source file.

The Reference Policy relies heavily on the **m4** (1) macro processor as the majority of supporting services are m4 macros.

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>

5.2.1 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-20 are distributed in a number of rpms as follows:

selinux-policy-3.12.1-179.fc20.noarch.rpm - Contains the SELinux `/etc/selinux/config` file, man pages and the 'Policy Header' development environment that is located at `/usr/share/selinux/devel`

⁴⁹ 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-3.12.1-179.fc20.noarch.rpm -
Contains the html policy documentation that is located at
`/usr/share/doc/selinux-policy/html`

selinux-policy-minimum-3.12.1-179.fc20.noarch.rpm

selinux-policy-mls-3.12.1-179.fc20.noarch.rpm

selinux-policy-targeted-3.12.1-179.fc20.noarch.rpm

These three rpms contain policy configuration files and the packaged policy modules (*.pp). They will be used to form the particular policy type in `/usr/share/selinux/<NAME>`, the install process will then install the policy in the appropriate `/etc/selinux/<NAME>` directory. Normally only one policy would be installed and active, however for development purposes all can be installed.

selinux-policy-sandbox-3.12.1-179.fc20.noarch.rpm

Contains the sandbox module for use by the `policycoreutils-sandbox` package. This will be installed as a module for one of the three main policies described above.

5.2.2 Policy Functionality

As can be seen from the policies distributed with F-20 above, they can be classified by the name of the functionality they support (taken from the NAME entry of the `build.conf` as shown in [Table 19](#)), for example the Fedora policies support:

`minimum` - MCS policy that supports a minimal set of confined daemons within their own domains. The remainder run in the `unconfined_t` space.

`targeted` - MCS policy that supports a greater number of confined daemons and can also confine other areas and users (this also supports the older 'strict' policy).

`mls` - MLS policy for server based systems.

5.2.3 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:

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:

- 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 5.2](#) 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, 1.4.1)

#####
#
# Declarations
#

attribute_role ada_roles;
roleattribute system_r ada_roles;

type ada_t;
type ada_exec_t;
application_domain(ada_t, ada_exec_t)
role ada_roles types ada_t;

#####
#
# Local policy
#

allow ada_t self:process { execstack execmem };

userdom_use_inherited_user_terminals(ada_t)

optional_policy(`
    unconfined_domain(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 to transition.
## </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 to transition.
## </summary>
## </param>
## <param name="role">
## <summary>
## Role allowed access.
## </summary>
## </param>
#
interface(`ada_run',`
    gen_require(`
        attribute_role ada_roles;
    `)
`)
```

```
ada_domtrans($1)
roleattribute $2 ada_roles;
')
```

ada.fc file contents:

```
/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)
```

5.2.4 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 Fedora can be viewed in a browser by <file:///usr/share/doc/selinux-policy/html/index.html> once the selinux-policy-doc rpm has been installed.

The documentation for the Reference Policy source will be available at `<location>/src/policy/doc/html` once `make html` has been executed (the `<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 Reference Policy documentation may also be available at a default location of `/usr/share/doc/refpolicy-VERSION/html` if `make install-doc` has been executed (where `VERSION` is the entry from the source `VERSION` file).

[Figure 5.2](#) shows an example screen shot of the documentation produced for the ada module interfaces.

The screenshot shows a Mozilla Firefox browser window titled "Security Enhanced Linux Reference Policy - Mozilla Firefox". The address bar shows the file path: `file:///usr/share/doc/selinux-policy/html/contrib_ada.html`. The page content is titled "Module: ada" and includes a "Description:" section stating "GNAT Ada95 compiler." Below this is an "Interfaces:" section with two entries:

- ada_domtrans(domain)**
Summary: Execute the ada program in the ada domain.
Parameters:

Parameter:	Description:
domain	Domain allowed to transition.
- ada_run(domain , role)**
Summary: Execute ada in the ada domain, and allow the specified role the ada domain.
Parameters:

Parameter:	Description:
domain	Domain allowed to transition.
role	Role allowed access.

5.3 Reference Policy Source

This section explains 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:

<https://github.com/TresysTechnology/refpolicy/wiki>

5.3.1 Source Layout

[Figure 5.1](#) shows the layout of the reference policy source tree, that once installed would be located at:

```
/etc/selinux/<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 policy to be built.

5.3.2 Reference Policy Files and Directories

[Table 18](#) 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](#) section 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.

The SELinux Notebook

File / Directory Name	Comments
<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-<type></code>	Application configuration files for all configurations of the Reference Policy where <type> is taken from the <code>build.conf</code> <code>TYPE</code> entry that are currently: <code>standard</code> , <code>MLS</code> and <code>MCS</code>). These files are used by SELinux-aware programs and described in the SELinux Configuration Files section.
<code>config/file_contexts.subs_dist</code>	Used to configure file context aliases (see the contexts/files/file_contexts.subs and file_contexts.subs_dist File section).
<code>config/local.users</code>	The file read by load policy for adding SELinux users to the policy on the fly. Note that this file is not used in the modular policy build.
<code>doc/html/*</code>	When <code>make html</code> has been executed, contains 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> <code>make</code> targets. It contains the complete XML documentation included in the policy.
<code>doc/templates/*</code>	Templates used for documentation web pages.
<code>man/*</code>	Various man pages for modules (<code>ftp</code> , <code>http</code> etc.)
<code>support/*</code>	Tools used in the build process.
<code>policy/flask/initial_sids</code>	This file has declarations for each initial SID. 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 security class. 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 common permissions and class specific permissions. The file is described in the Modular Policy Build Structure section.
<code>policy/modules/*</code>	Each directory represents a layer in Reference Policy. All of the modules are contained in one of these layers. The <code>contrib</code> modules are supplied externally to the Reference Policy, then linked into the build. The files present in each directory are: <code>metadata.xml</code> - describes the layer. <code><module_name>.te, .if & .fc</code> - contains policy source as described in the Reference Policy Module Files section. 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.
<code>policy/booleans.conf</code>	This file is generated/updated by the <code>conf</code> <code>make</code> target. It contains the booleans in the policy, and their default values. If <code>tunables</code> are implemented as <code>booleans</code> , <code>tunables</code> will also be included. This file will be installed as the

File / Directory Name	Comments
	/etc/selinux/NAME/booleans file (note that this is not true for any system that implements the modular policy - see the Booleans, Global Booleans and Tunable Booleans section).
policy/constraints	This file defines 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. (Note that this file does not contain the MLS / MCS constraints as they are in the <code>mls</code> and <code>mcs</code> files described below). The file usage in policy generation is described in the Modular Policy Build Structure section.
policy/context_defaults	This would contain any specific <code>default_user</code> , <code>default_role</code> , <code>default_type</code> and/or <code>default_range</code> rules required by the policy.
policy/global_booleans	This file defines all booleans that have a global scope, their default value, and documentation. See the Booleans, Global Booleans and Tunable Booleans section.
policy/global_tunables	This file defines all tunables that have a global scope, their default value, and documentation. See the Booleans, Global Booleans and Tunable Booleans section.
policy/mcs	This contains information used to generate the <code>sensitivity</code> , <code>category</code> , <code>level</code> and <code>mlsconstraint</code> statements used to define the MCS configuration. The file usage in policy generation is described in the Modular Policy Build Structure section.
policy/mls	This contains information used to generate the <code>sensitivity</code> , <code>category</code> , <code>level</code> and <code>mlsconstraint</code> statements used to define the MLS configuration. The file usage in policy generation is described in the Modular Policy Build Structure section.
policy/modules.conf	This file contains a listing of available modules, and how they will be used when building Reference Policy. To prevent a module from being used, set the module to "off". For monolithic policies, modules set to "base" and "module" will be included in the policy. For modular policies, modules set to "base" will be included in the base module; those set to "module" will be compiled as individual loadable modules. This file is described in the Reference Policy Build Options - modules.conf section.
policy/policy_capabilities	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.
policy/users	This file defines the users included in the policy. The file usage in policy generation is described in the

File / Directory Name	Comments
	Modular Policy Build Structure section.
<code>securetty_types</code>	These files are not part of the standard Reference Policy distribution but are added by Fedora source updates.
<code>setrans.conf</code>	

Table 18: The Reference Policy Files and Directories

5.3.3 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.

5.3.3.1 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. 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 19](#) 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 macro definitions are shown in [Table 20](#) 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
OUTPUT_POLICY	Integer	Set the version of the policy created when building a monolithic policy. This option has no effect on modular policy.
TYPE	String	Available options are <code>standard</code> (uses RBAC/TE), <code>mcs</code> (uses RBAC/TE/MCS) and <code>mls</code> (uses RBAC/TE/MLS). The <code>mls</code> and <code>mcs</code> options control the <code>enable_mls</code> , and <code>enable_mcs</code> policy blocks.
NAME	String	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.
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.
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 <code>allowed</code> , <code>denied</code> , or the policy loading can be <code>rejected</code> . See the SELinux Filesystem for more details. If not set, then it will be taken from the <code>semanage.conf</code> file.
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.

Option	Type	Comments
MONOLITHIC	Boolean (y n)	If 'y' a monolithic policy is built, otherwise a modular policy is built.
UBAC	Boolean (y n)	If 'y' User Based Access Control policy is built. The default for Red Hat is 'n'. These are defined as constraints in the <code>policy/constraints</code> file. Note Version 1 of the Reference Policy did not have this entry and defaulted to Role Based Access Control. The UBAC option is described at http://blog.siphos.be/2011/05/selinux-user-based-access-control/ .
CUSTOM_BUILDOPT	String	Space separated list of custom build options.
MLS_SENS	Integer	Set the number of sensitivities in the MLS policy. Ignored on standard and MCS policies.
MLS_CATS	Integer	Set the number of categories in the MLS policy. Ignored on standard and MCS policies.
MCS_CATS	Integer	Set the number of categories in the MCS policy. Ignored on standard and MLS 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 19: build.conf Entries

m4 Parameter Name in Makefile	From build.conf entry	Comments
<code>enable_mls</code>	TYPE	Set if MLS policy build enabled.
<code>enable_mcs</code>	TYPE	Set if MCS policy build enabled.
<code>enable_ubac</code>	UBAC	Set if UBAC set to 'y'.
<code>mls_num_sens</code>	MLS_SENS	The number of MLS sensitivities configured.
<code>mls_num_cats</code>	MLS_CATS	The number of MLS categories configured.
<code>mcs_num_cats</code>	MCS_CATS	The number of MCS categories configured.
<code>distro_\$(DISTRO)</code>	DISTRO	The distro name or blank.
<code>direct_sysadm_daemon</code>	DIRECT_INITRC	If <code>DIRECT_INITRC</code> entry set to 'y'.
<code>hide_broken_syntoms</code>		This is set up in the Makefile and can be used in modules to hide errors with <code>dontaudit</code> rules (or even allow rules).

Table 20: m4 parameters set at build time - These have been extracted from the Reference Policy Makefile file.

5.3.3.2 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 comments⁵⁰) 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 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 distributor 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).

⁵⁰ The comments are also important as they form part of the documentation when it is generated by the make html target.

```
# Layer: kernel
# Module: kernel
# Required in base
#
# Policy for kernel threads, proc filesystem, and unlabeled processes and objects.
#
kernel = base
```

Those marked as required in base are shown in [Table 21](#) (note that F-20 and the standard reference policy are different)

Layer	Module Name	Comments
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 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

Layer	Module Name	Comments
		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_'.
kernel	ubac	Disabled by Fedora but enabled on standard Ref Policy. Support user-based access control.
system	application	Enabled by Fedora but not standard Ref Policy. Defines attributes and interfaces for all user apps.
system	setrans	Enabled by Fedora but not standard Ref Policy. Support for mcstransd (8).

Table 21: Mandatory modules.conf Entries

5.3.3.2.1 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 Red Hat reference policy 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.

5.3.4 Source Installation and Build Make Options

This section explains the various make options available that have been taken from the README file. [Table 22](#) describes the general make targets, [Table 23](#) describes the modular policy make targets and [Table 24](#) 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 Makefile 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 bare 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. Note that if <code>make bare</code> has been executed before this make target, or it is a first build, then the <code>modules/kernel/corenetwork.???.in</code> files will be used to generate the <code>corenetwork.te</code> and <code>corenetwork.if</code> module files. These <code>*.in</code> files may be edited to

Make Target	Comments
	configure network ports etc. (see the # <code>network_node</code> examples entries).
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.
install-appconfig	Installs the appropriate SELinux-aware configuration files.

Table 22: 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.
install-docs	Build and install the documentation and example module source with Makefile. The default location is <code>/usr/share/doc/refpolicy-VERSION</code> , where the version is the value in the <code>VERSION</code> file.

Table 23: 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 <code>dontaudit</code> 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 24: Monolithic Policy Build Make Targets

5.3.5 Booleans, Global Booleans and Tunable Booleans

The three files `booleans.conf`, `global_booleans` and `global_tunables` are built and used as follows:

<code>booleans.conf</code>	<p>This file is generated / updated by <code>make conf</code>, and contains all the booleans in the policy with their default values. If tunable and global booleans are implemented then these are also included.</p> <p>This file can also be delivered as a part of the Fedora 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 <code>make install</code> is executed, this file will be used to set the default values.</p> <p>Note that if booleans are updated locally the policy store will contain a booleans.local file.</p> <p>In SELinux enabled systems that support the policy store features (modular policies) this file is not installed as <code>/etc/selinux/NAME/booleans</code>.</p>
<code>global_booleans</code>	<p>These are booleans that have been defined in the <code>global_tunables</code> 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 <code>false</code>):</p> <pre>secure_mode</pre>
<code>global_tunables</code>	<p>These are booleans that have been defined in module files using the gen_tunable macro and added to the <code>global_tunables</code> file by <code>make conf</code>. 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 <code>allow_execstack</code> that will allow all processes running in <code>unconfined_t</code> to make their stacks executable.</p>

5.3.6 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 macros⁵¹ in the source modules will be expanded to form actual policy language statements as described in the [SELinux Policy Language](#) section. [Figure 5.3](#) shows these temporary files that are used to form the `base.conf`⁵² 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 5.4](#).

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> statements extracted from the <code>*.te</code> and <code>*.if</code> files that form the base module.	<code>modules/**/*.te</code> <code>modules/**/*.if</code>	<code>all_attrs_types.conf</code>
Contains the global and tunable bools extracted from the <code>conf</code> files.	<code>global_bools.conf</code> <code>global_tunables.conf</code>	<code>global_bools.conf</code>
Contains the rules extracted from each of the modules <code>.te</code> and <code>.if</code> files defined in the <code>modules.conf</code> file as 'base'.	<code>base</code> modules	<code>only_te_rules.conf</code>
Contains the expanded users from the <code>users</code> file.	<code>users</code>	<code>all_post.conf</code>
Contains the expanded constraints from the <code>constraints</code> file.	<code>constraints</code>	
Contains the default SID labeling extracted from the <code>*.te</code> files.	<code>modules/**/*.te</code>	
Contains the <code>fs_use_xattr</code> , <code>fs_use_task</code> , <code>fs_use_trans</code> and <code>genfscon</code> statements extracted from each of the modules <code>.te</code> and <code>.if</code> files defined in the <code>modules.conf</code> file as	<code>modules/**/*.te</code> <code>modules/**/*.if</code>	

⁵¹ These are explained in the [Reference Policy Macros](#) section.

⁵² 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
'base'.		
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 5.3: 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 shown below.	tmp/<module_name>.tmp	<module_name>.mod
For each module defined as 'module' in the modules.conf configuration file, an expanded file context file is built from the <module_name>.fc file.	modules/**/<module_name>.fc	base.fc.tmp
<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 build.conf file.</p>		

Figure 5.4: 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).

5.3.7 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>
```

5.4 Installing and Building the Reference Policy Source

This section will give a brief overview of how to build the Reference Policy for an MCS modular build that is similar (but not the same) as the Fedora targeted policy. The Fedora F-20 version of the targeted policy build is discussed but building without using the rpm spec file is more complex.

5.4.1 Building Standard Reference Policy

This will run through a simple configuration process and build of a reference policy similar to the Fedora targeted policy. By convention the source is installed in a central location and then for each type of policy a copy of the source is installed at `/etc/selinux/<NAME>/src/policy`.

The basic steps are:

1. Install master Reference Policy Source and add the contributed modules:

```
# Check out the core policy:
git clone https://github.com/TresysTechnology/refpolicy.git
cd refpolicy
# Add the contibuted modules (policy/modules/contrib)
git submodule init
git submodule update
```

2. Edit the `build.conf` file to reflect the policy to be built, the minimum required is setting the `NAME` = entry. An example file with `NAME` = `refpolicy-test` is as follows:

```
#####
# 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
# for 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/<NAME>/src/policy directory.
NAME = refpolicy-test

# 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. Fedora use allow for all
# policies except MLS that uses 'deny'.
UNK_PERMS = allow

# 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 y here will build a monolithic policy.
MONOLITHIC = n

# User-based access control (UBAC)
# Enable UBAC for role separations. Note Fedora disables UBAC.
UBAC = n

# Custom build options. This field enables custom build options. Putting
# foo here will enable build option blocks foo. Options should be separated
# by spaces.
CUSTOM_BUILDOPT =

# 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.
# The categories will be c0 to c(MLS_CATS-1).
MLS_CATS = 1024

# Number of MCS Categories
# The categories will be c0 to c(MLC_CATS-1).
MCS_CATS = 1024

# Set this to y to only display status messages during build.
QUIET = n
```

3. Run `make install-src` to install source at policy build location.
4. Change to the `/etc/selinux/<NAME>/src/policy` directory where an unconfigured basic policy has been installed.

- Run `make conf` to build an initial `policy/booleans.conf` and `policy/modules.conf` files. For this simple configuration these files will not be edited.

This process will also build the `policy/modules/kernel/corenetwork.te / corenetwork.if` files if not already present. These would be based on the contents of `corenetwork.te.in` and `corenetwork.if.in` configuration files (for this simple configuration these files will not be edited).

- Run `make load` to build the policy, add the modules to the store and install the binary kernel policy plus its supporting configuration files.

Note that if policy stores have been migrated, the store will default to `/var/lib/selinux/refpolicy-test`, with the modules in `active/modules/400/<module_name>`, there will also be a CIL version of the module (see [Policy Store Migration](#) for details).

- The policy should now be built and can be checked using tools such as `apol` (8) or loaded by editing the `/etc/selinux/config` file, running `'touch /.autorelabel'` and rebooting the system.

5.4.2 Building the Fedora Policy

Building Fedora policies by hand is complex as they use the `rpmbuild/SPECS/selinux-policy.spec` file, therefore this section will give an overview of how this can be achieved, the reader can then experiment (the spec file gives an insight). The build process assumes that an equivalent 'targeted' policy will be built named 'targeted-179'.

Install the source as follows:

```
rpm -Uvh selinux-policy-3.12.1-179.fc20.src.rpm
```

The `rpmbuild/SOURCES` directory contents should be as follows with comments on how the files should be installed:

File Name	Comments
<code>serefpolicy-3.12.1.tgz</code>	The Reference Policy version 2.20120725 This should be unpacked into: <code>rpmbuild/SOURCES/serefpolicy-3.12.1</code>
<code>policy-f20-base.patch</code>	Fedora changes to Reference Policy version 2.20120725. These patches should be used to update the above <code>patch -p1 <policy-f20-base.patch</code>
<code>serefpolicy-contrib-3.12.1.tgz</code>	The Reference Policy contribution modules from version 2.20120725 Unpack the files, apply the <code>policy-f20-contrib.patch</code> and then installed into: <code>./serefpolicy-3.12.1/policy/modules/contrib</code>
<code>policy-f20-contrib.patch</code>	Fedora changes to Reference Policy contribution modules.

Once the above has been completed, run `make conf` from `./serefpolicy-3.12.1` to initialise the build (it creates two important files in `./serefpolicy-`

3.12.1/policy/modules/kernel called corenetwork.te and corenetwork.if).	
config.tgz	Fedora changes to Reference Policy version 2.20120725 application config files. These should be unpacked to update the ./serefpolicy-3.12.1/config versions (some will be added and others updated).
permissivedomains.pp	Contains Fedora domains currently sets to permissive. In selinux-policy-3.12.1-179.fc20 there are 31 permissive domains. Copy this to ./serefpolicy-3.12.1 as it will then be built into the policy.
selinux-policy.conf	Allows the build process to determine network information. Not required for this exercise as the corenetwork.te and corenetwork.if files have been built.
Makefile.devel	Fedora makefile when using headers. This can replace the ./serefpolicy-3.12.1/support/Makefile.devel
booleans.subs_dist	Common files installed for each policy type. Not required for this exercise.
customizable_types	
file_contexts.subs_dist	
Used to build "targeted" policy	
booleans-targeted.conf	Replace the ./serefpolicy-3.12.1/policy/booleans.conf file with this version.
modules-targeted-base.conf	Concatenate both files and copy this to become: ./serefpolicy-3.12.1/policy/modules.conf
modules-targeted-contrib.conf	
securetty_types-targeted	Replace the ./serefpolicy-3.12.1/config/appconfig-mcs/securetty_types file with this version.
setrans-targeted.conf	Not required for this exercise.
users-targeted	Replace the ./serefpolicy-3.12.1/policy/users file with this version.
Used to build "minimum" policy	
booleans-minimum.conf	
modules-targeted-base.conf	Uses the targeted modules.conf.
modules-targeted-contrib.conf	
securetty_types-minimum	
setrans-minimum.conf	
users-minimum	
Used to build "mls" policy	
booleans-mls.conf	
modules-mls-base.conf	
modules-mls-contrib.conf	
securetty_types-mls	
setrans-mls.conf	
users-mls	

The basic steps are:

1. Edit the build.conf file to reflect the policy to be built:

```
#####  
# 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  
# for 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/<NAME>/src/policy directory.  
NAME = targeted-179  
  
# 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. Fedora use allow for all  
# policies except MLS that uses 'deny'.  
UNK_PERMS = allow  
  
# 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 y here will build a monolithic policy.  
MONOLITHIC = n  
  
# User-based access control (UBAC)  
# Enable UBAC for role separations. Note Fedora disables UBAC.  
UBAC = n  
  
# Custom build options. This field enables custom build options. Putting  
# foo here will enable build option blocks foo. Options should be separated  
# by spaces.  
CUSTOM_BUILDOPT =  
  
# 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.  
# The categories will be c0 to c(MLS_CATS-1).  
MLS_CATS = 1024  
  
# Number of MCS Categories  
# The categories will be c0 to c(MLC_CATS-1).  
MCS_CATS = 1024  
  
# Set this to y to only display status messages during build.  
QUIET = n
```

2. From `rpmbuild/SOURCES/serefpolicy-3.12.1` run `make install-src` to install source at policy build location.

3. Change to the `/etc/selinux/targeted-179/src/policy` directory where the policy has been installed.
4. Run `make load` to build the policy, add the modules to the store and install the binary kernel policy plus its supporting configuration files.

Note that if policy stores have been migrated, the store will default to `/var/lib/selinux/targeted-179`, with the modules in `active/modules/400/<module_name>`, there will also be a CIL version of the module (see [Policy Store Migration](#) for details).

5. Install the `permissivedomains.pp` module as follows (this will set 31 permissive domains that are in the F-20 version of the policy):

```
semodule -s targeted-179 -i permissivedomains.pp
```

6. The policy should now be built and can be checked using tools such as `apol` (8) or loaded by editing the `/etc/selinux/config` file, running `'touch /.autorelabel'` and rebooting the system.

5.5 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 Fedora policy header installations are slightly different as described below.

5.5.1 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 in the `/usr/share/selinux/<NAME>` directory.
2. Execute the `make install-headers` 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/<NAME>/include` directory.
 - b) Produce the XML documentation set that reflects the source and place it in the `/usr/share/selinux/<NAME>/include` directory.
 - c) Copy a development `Makefile` for building from policy headers to the `/usr/share/selinux/<NAME>/include` directory.
 - d) Copy the support macros `.spt` files to the `/usr/share/selinux/<NAME>/include/support` directory.

This will also include an `all_perms.spt` file that will contain macros to allow all classes and permissions to be resolved.

- e) Copy the module interface files (`.if`) to the relevant module directories at:
`/usr/share/selinux/<NAME>/include/modules.`

5.5.2 Using the Reference Policy Headers

Note that this section describes the standard Reference Policy headers, the F-20 installation is slightly different and described in the [Using Fedora 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 `/etc/selinux/<NAME>/src/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/<NAME>/include/Makefile
```

However there is another `Makefile` that can be installed in the users home directory (`$HOME`) that will call the master `Makefile`. This is located at `/etc/selinux/<NAME>/src/policy/doc` in the reference policy source and is called `Makefile.example`. This is shown below (note that it extracts the `<policy_nNAME /etc/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 25](#) 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.
xml	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 25: Header Policy Build Make Targets

5.5.3 Using Fedora Supplied Headers

The F-20 distribution installs the headers in a slightly different manner as Fedora installs:

- A `modules-base.lst` and `modules-contrib.lst` containing a list of installed modules under `/usr/share/selinux/<NAME>`.
- The development header files are installed in the `/usr/share/selinux/devel` directory. 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 installed in the `/usr/share/doc/selinux-policy/html` directory.

5.6 Migrating Compiled Modules to CIL

As explained in the [Policy Store Migration](#) section, when `libsepol` etc. are upgraded to version 2.4, the policy stores will be migrated to the new location that will contain also contain CIL versions of the policy modules.

5.7 Reference Policy Support Macros

This section explains some of the support macros used to build reference policy source modules (see [Table 26](#) for the list). These macros are located at:

- `./policy/support` for the reference policy source.
- `/usr/share/selinux/<NAME>/include/support` for Reference Policy installed header files.
- `/usr/share/selinux/devel/support` for Fedora installed header files.

The following support macro file contents are explained:

`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.

When the header files are installed the `all_perms.spt` support macro file is also installed that describes all classes and permissions configured in the original source policy.

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 26: Support Macros described in this section

Notes:

1. The macro calls can be in any configuration file read by the build process and can be found in (for example) the `users`, `mls`, `mcs` and `constraints` files.
2. There are four 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) `enable_ubac` - this enables the user based access control within the `constraints` file.
 - d) `hide_broken_symptoms` - this is used to hide errors in modules with `dontaudit` rules.

These are also mentioned in [Table 20](#) 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:

Correct:

```
policy_module(ftp, 1.7.0)
```

Incorrect:

```
policy_module (ftp, 1.7.0)
```

5.7.1 Loadable Policy Macros

The loadable policy module support macros are located in the `loadable_module.spt` file.

5.7.1.1 `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 M.m.m format (where M = major version number and m = 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;
....
}
```

5.7.1.2 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;
}
```

5.7.1.3 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:

<code>optional_policy</code>	The <code>optional_policy</code> macro keyword.
<code>optional_statements</code>	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 ftp_t system_crond_t:fd use;
allow ftp_t system_crond_t:fifo_file { getattr read write append ioctl
lock };
allow ftp_t system_crond_t:process sigchld;
allow ftp_t crond_t:fifo_file { getattr read write append ioctl lock };
allow ftp_t crond_t:fd use;
allow ftp_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 ftp_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
```

5.7.1.4 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;
```

5.7.1.5 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 ftp_t nfs_t:dir { read getattr lock search ioctl
add_name remove_name write };
    allow ftp_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
```

5.7.1.6 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 5.6](#) and described in the [Module Expansion](#) section.

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:

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](#).

```
# 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
```

5.7.1.7 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 <code>_template</code> (e.g. <code>ftp_per_role_template</code>).
<code>template_rules</code>	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/xgquest.te
# module and extracted from tmp/xgquest.tmp.
#
##### START Expanded code segment #####
#
optional {
##### begin openoffice_per_role_template(xgquest,xgquest_usertype,xgquest_r)
  require {
    type openoffice_exec_t;
  } # end require
  type xgquest_openoffice_t; # Parameter $1

  .....
  # This is a long set of rules, therefore has been cut down.
  .....
  ....
  typeattribute xgquest_openoffice_t xgquest_usertype; # Parameter $2
  ..
  type_transition xgquest_usertype openoffice_exec_t:process xgquest_openoffice_t;
  ..
  role xgquest_r types xgquest_openoffice_t; # Parameter $3
  ....
  allow xgquest_usertype xgquest_openoffice_t:process { signal sigkill };
  allow xgquest_openoffice_t xgquest_usertype:unix_stream_socket connectto;
##### end openoffice_per_role_template(xgquest,xgquest_usertype,xgquest_r)
} # end optional
```

5.7.2 Miscellaneous Macros

These macros are in the `misc_macros.spt` file.

5.7.2.1 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:

<code>gen_context</code>	The <code>gen_context</code> macro keyword.
<code>context</code>	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
```

5.7.2.2 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;
```

5.7.2.3 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 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)
}
```

5.7.3 MLS and MCS Macros

These macros are in the `mls_mcs_macros.spt` file.

5.7.3.1 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:

<code>gen_cats</code>	The <code>gen_cats</code> macro keyword.
<code>mcs_num_cats</code> <code>mls_num_cats</code>	These are the maximum number of categories that have been extracted from the <code>build.conf</code> file <code>MCS_CATS</code> or <code>MLS_CATS</code> entries and set as <code>m4</code> 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;
```

5.7.3.2 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;
```

5.7.3.3 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 mcs_num_cats	This is the parameter that defines the number of categories to generate.

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;
```

5.7.3.4 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
```

5.7.4 `ifdef` / `ifndef` Parameters

This section contains examples of the common `ifdef` / `ifndef` parameters that can be used in module source files.

5.7.4.1 `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 };  
' )
```

5.7.4.2 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)
')
```

5.7.4.3 enable_ubac

This is used within the ./policy/constraints configuration file to set up various attributes to support user based access control (UBAC). These attributes are then used within the various modules that want to support UBAC. This support was added in version 2 of the Reference Policy.

The parameter is set up by the Makefile with information taken from the build.conf file at the start of the build process (ubac = y | ubac = n).

Example Macro:

```
# This example is from the ./policy/constraints file.
# Note that the ubac_constrained_type attribute is defined in
# modules/kernel/ubac.te module.

define(`basic_ubac_conditions',`
    ifdef(`enable_ubac',`
        u1 == u2
        or u1 == system_u
        or u2 == system_u
        or t1 != ubac_constrained_type
        or t2 != ubac_constrained_type
    ')
')
```

5.7.4.4 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)
')
```

5.8 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 5.5](#), and the resulting expanded code shown in [Figure 5.6](#) and then described in the [Module Expansion](#) section.

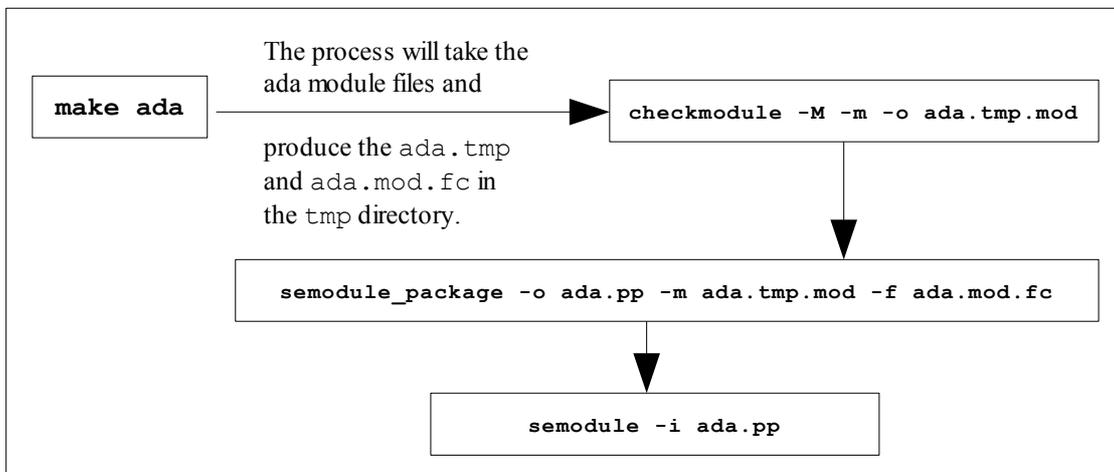


Figure 5.5: The make ada sequence of events

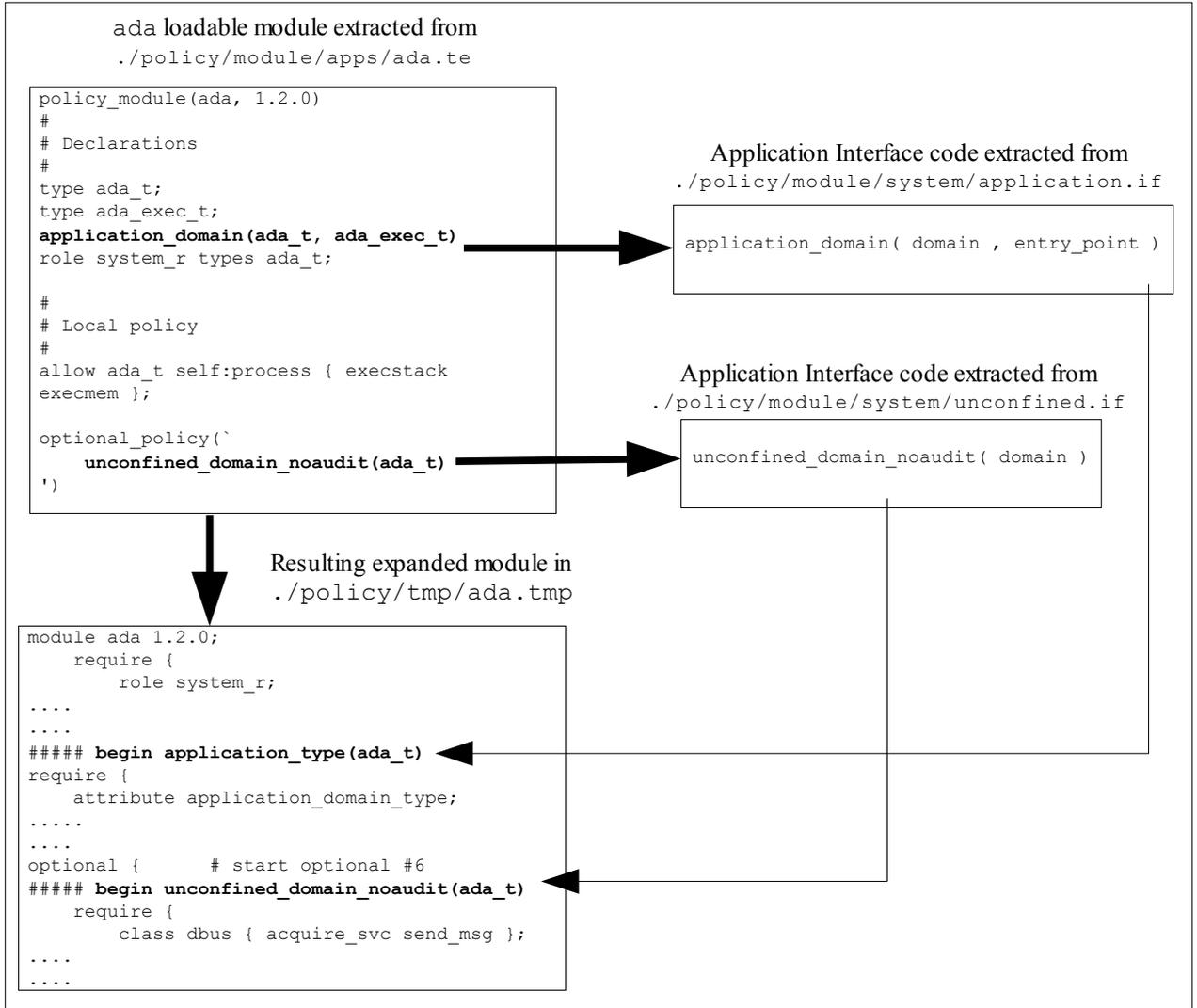


Figure 5.6: The expansion process

6. Implementing SELinux-aware Applications

6.1 Introduction

The following definitions attempt to explain the difference between the two types of userspace SELinux application (however the distinction can get 'blurred'):

SELinux-aware - Any application that provides support for SELinux. This generally means that the application makes use of SELinux libraries and/or other SELinux applications. Example SELinux-aware applications are the Pluggable Authentication Manager (**PAM**(8)) and SELinux commands such as **runcon** (1). It is of course possible to class an object manager as an SELinux-aware application.

Object Manager - Object Managers are a specialised form of SELinux-aware application that are responsible for the labeling, management and enforcement⁵³ of the objects under their control.

Generally the userspace Object Manager forms part of an application that can be configured out should the base Linux OS not support SELinux.

Example userspace Object Managers are:

- X-SELinux is an optional X-Windows extension responsible for labeling and enforcement of X-Windows objects.
- Dbus has an optional Object Manager built if SELinux is defined in the Linux build. This is responsible for the labeling and enforcement of Dbus objects.
- SE-PostgreSQL is an optional extension for PostgreSQL that is responsible for the labeling and enforcement of PostgreSQL database and supporting objects.

Therefore the basic distinction is that Object Managers manage their defined objects on behalf of an application, whereas general SELinux-aware applications do not (they rely on 'Object Managers' to do this e.g. the kernel based Object Managers such as those that manage filesystem, IPC and network labeling).

6.1.1 Implementing SELinux-aware Applications

This section puts forward various points that may be useful when developing SELinux-aware applications and object managers using `libselinux`.

1. Determine the security objectives and requirements.
2. Because these applications manage labeling and access control, they need to be trusted.

⁵³ The SELinux security server does not enforce a decision, it merely states whether the operation is allowed or not according to the policy. It is the object manager that enforces the decision of the policy / security server, therefore an object manager must be trusted. This is also true of labeling, the object manager ensures that labels are applied to their objects as defined by policy.

3. Where possible use the `libselinux *_raw` functions as they avoid the overhead of translating the context to/from the readable format (unless of course there is a requirement for a readable context - see `mcstransd(8)`).
4. Use `selinux_set_mapping(3)` to limit the classes and permissions to only those required by the application.
5. The standard output for messages generated by `libselinux` functions is `stderr`. Use `selinux_set_callback(3)` with `SELINUX_CB_LOG type` to redirect these to a log handler.
6. Do not directly reference SELinux configuration files, always use the `libselinux` path functions to return the location. This will help portability as SELinux has some changes in the pipe-line for the location of the policy configuration files and the SELinux filesystem.
7. Where possible use the `selabel_*`(3) functions to determine a files default context as they effectively replace the `matchpathcon*`(3) series of functions - see `selabel_file(5)`.
8. Do not use class IDs directly, use `string_to_security_class(3)` that will take the class string defined in the policy and return the class ID/value. Always check the value is > 0 . If `0`, then signifies that the class is unknown and the `deny_unknown` flag setting in the policy will determine the outcome of any decision - see `security_deny_unknown(3)`.
9. Do not use permission bits directly, use `string_to_av_perm(3)` that will take the permission string defined in the policy and return the permission bit mask.
10. Where performance is important when making policy decisions (i.e. using `security_compute_av(3)`, `security_compute_av_flags(3)`, `avc_has_perm(3)` or `avc_has_perm_noaudit(3)`), then use the `selinux_status_*`(3) functions to detect policy updates etc. as these do not require system call over-heads once set up. Note that the `selinux_status_*` functions are only available from `libselinux 2.0.99`, with Linux kernel 2.6.37 and above.
11. Be aware that applications being built for 32 bit systems need to specify the `CFLAG -D_FILE_OFFSET_BITS=64` as `libselinux` is built with this flag. This is particularly important if `matchpathcon_filespec_add(3)` is used as it passes over `ino_t ino` that is too small otherwise (i.e. needs to be 64 bits).
12. There are changes to the way contexts are computed for sockets in kernels 2.6.39 and above as described in the [Computing Security Contexts](#) section. The functions affected by this are: `avc_compute_create(3)`, `avc_compute_member(3)`, `security_compute_create(3)`, `security_compute_member(3)` and `security_compute_relabel(3)`.
13. It is possible to set an undefined context if the process has `capability(7)` `CAP_MAC_ADMIN` and class `capability2` with `mac_admin` permission

in the policy. This is called 'deferred mapping of security contexts' and is explained at:

<http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commit;h=12b29f34558b9b45a2c6eabd4f3c6be939a3980f>

6.1.2 Implementing Object Managers

To implement object managers for applications, an understanding of the application is essential, because as a minimum:

- What object types and their permissions are required.
- Where in the code object instances are created.
- Where access controls need to be applied.

While this section cannot help with those points, here are some notes to help during the design phase (also see the [Implementing SELinux-aware Applications](#) section):

1. Determine what objects are required and the access controls (permissions) that need to be applied.
2. Does SELinux already have some of these object classes and permissions defined. For standard Linux OS objects such as files, then these would be available. If so, the object manager should remap them with **selinux_set_mapping** (3) so only those required are available.

However, do not try to reuse a current object that may be similar to the requirements, it will cause confusion at some stage. Always generate new classes/permissions.

3. If the application has APIs or functions that integrate with other applications or scripts, then as part of the object manager implementation these may need to support the use of security contexts (examples are X-Windows and SE-PostgreSQL that provide functions for other applications to use). Therefore if required, provide common functions that can be used to label the objects.
4. Determine how the initial objects will be labeled. For example will a configuration file be required for default labels, if so how will this be introduced into the SELinux userspace build. Examples of these are the X-Windows (**selabel_x** (5)), SE-PostgreSQL (**selabel_db** (3)), and file context series of files (**selabel_file** (5)).
5. Will the labeling need to be persistent across policy and system reloads or not. X-Windows is an example of a non-persistent, and SE-PostgreSQL is an example of a persistent object manager.
6. Will support for the standard audit log or its own be required (the `libselinux` functions default to `stderr`). Use **selinux_set_callback** (3) to manage logging services.
7. Decide whether an AVC cache is required or not. If the object manager handles high volumes of requests then an AVC will be required. See the [Types of Object Manager](#) section for details.
8. Will the object manager need to do additional processing when policy or enforcement changes are detected. This could be clearing any caches or

resetting variables etc.. If so, then `selinux_set_callback(3)` will be used to set up these functions. These events are detected via the `netlink(7)` services, see `avc_open(3)` and `avc_netlink_open(3)` for the various options available.

9. If possible implement a service like XACE for the application, and use it to interface with the applications SELinux object manager. The XACE interface acts like the LSM which supports SELinux as well as other providers such as SMACK. The XACE interface is defined in the "[X Access Control Extension Specification](#)" [15], and for reference, the SE-PostgreSQL service also implements a similar interface.

The XACE specification is available from:

<http://www.x.org/releases/X11R7.5/doc/security/XACE-Spec.pdf>

6.1.3 Reference Policy Changes

When adding a new object manager to SELinux, it will require at least a new policy module to be added. This section assumes that the SELinux Reference Policy is in use and gives some pointers, however any detail is beyond the scope of this section. Further information can be found at:

<https://github.com/TresysTechnology/refpolicy/wiki>

The latest Reference Policy source can be obtained as follows:

```
git clone https://github.com/TresysTechnology/refpolicy.git
```

The main points to note when adding to the Reference Policy are:

1. Create sample Reference Policy policy modules (`*.te`, `*.if` and `*.fc` module files) that provide rules for managing the new objects as described in the [Reference Policy Module Files](#) section.

The SE-PostgreSQL modules provide an example, see the `./refpolicy/policy/modules/services/postgresql.*` files in the Reference Policy source.

2. Create any new policy classes and permissions for the Reference Policy, these will need to be built into the base module as described in the [Adding New Object Classes and Permissions](#) section.

Note, that if no new object classes, permissions or constraints are being added to the policy, then the Reference Policy source code does not require modification, and supplying the module files (`*.te`, `*.if` and `*.fc`) should suffice.

3. Create any constraints required as these need to be built into the base module of the Reference Policy. They are added to the `./refpolicy/policy/constraints`, `mcs` and `mls` files. Again the SE-PostgreSQL entries in these files give examples (find the `db_*` class entries).
4. Create any SELinux configuration files (context, user etc.) that need to be added to the policy at build time.

5. Either produce an updated Reference Policy source or module patch, depending on whether new classes/constraints have been added. Note that by default a new module will be generated as a 'module', if it is required that the module is in the base (unusual), then add an entry `<required val='true'>` to the start of the interface file as shown below:

```
## <summary>
##Comment regarding interface file
## </summary>
## <required val="true">
##Comment on reason why required in base
## </required>
```

6.1.4 Adding New Object Classes and Permissions

Because userspace object managers do not require their new classes and permissions to be built into the kernel, the configuration is limited to the actual policy (generally the Reference Policy) and the application object manager code. New classes are added to the Reference Policy `security_classes` file and permissions to the `access_vectors` file.

The class configuration file is at:

```
./refpolicy/policy/flask/security_classes
```

and each entry must be added to the end of the file in the following format:

```
class object_name # userspace
```

Where **class** is the class keyword and `object_name` is the name of the object. The **# userspace** is used by build scripts to detect userspace objects.

The permissions configuration file is at:

```
./refpolicy/policy/flask/access_vectors
```

and each entry must be added to the end of the file in the following format:

```
class object_name
{
    perm_name
    [.....]
}
```

Where **class** is the class keyword, `object_name` is the name of the object and `perm_name` is the name given to each permission in the class (there is a limit of 32 permissions within a class). It is possible to have a common permission section within this file, see the file object entry in the `access_vectors` file for an example.

The same principle applies to adding new class/permissions to Android although the flask files are located in the `external/sepolicy` directory.

Note that CIL policies do not use flask files and class/permissions must be declared using the `class`, `classpermission`, and `classorder` statements (see the

device/demo_vendor/cil_device/external/cil_sepolicy/classes_and_perms.cil file in the Notebook tarball).

For reference, http://selinuxproject.org/page/Adding_New_Permissions describes how new kernel object classes and permissions are added to the system and is summarised as follows for kernels $\geq 2.6.33$ that use dynamic class/perm discovery:

1. Edit `security/selinux/include/classmap.h` in the kernel tree and add the required definition. This will define the class and/or permission for use in the kernel; the corresponding symbol definitions will be automatically generated during the kernel build. If not defined in the policy, then the class and/or permission will be handled in accordance with the policy's `handle_unknown` definition, which can be `reject` (refuse to load the policy), `deny` (deny the undefined class/permission), or `allow` (allow the undefined class/permission). `handle_unknown` is set to `allow` in Fedora policies.
2. Edit `refpolicy/policy/flask/security_classes` and/or `access_vectors` in the refpolicy tree and add your definition. This will define the class and permission for use in the policy. These are generally added to the class and/or permission at the end of the existing list of classes or permissions for that class for backward compatibility with older kernels. The class and/or permission definition in policy need not line up with the definition in the kernel's classmap, as the values will be dynamically mapped by the kernel. Then add allow rules as appropriate to the policy for the new permissions.

The email thread <http://marc.info/?l=seandroid-list&m=139056956927985&w=2> describes how the CAN sockets could be added to the kernel along with possible hooks required in `security/selinux/hooks.c`.

7. Security Enhancements for Android

7.1 Introduction

This section gives an overview of the enhancements made to Android to add SELinux services to Security Enhancements for Android™ (SE for Android).

The main objective of this document is to provide a reference for the tools, commands, policy building tools and file formats of SE for Android based on the 4.4 release. The builds discussed are from AOSP master and SEAndroid master repositories (as September '14).

The AOSP git repositories can be found at <https://android.gogglesource.com> and the SEAndroid enhancements at <https://bitbucket.org/seandroid>.

For up to date information on the status of SE for Android the following should be consulted: <http://seandroid.bitbucket.org/>.

The Notebook tarball has a sample emulator device with a CIL policy that uses namespaces. This has been tested on AOSP 4.4 September '14 but will be obsolete by the time anyone tries it, therefore only useful as a reference.

7.1.1 Terminology

This section describes how the terms SE for Android, AOSP and SEAndroid are used in this document.

SE for Android	Used to describe the overall framework for implementing SELinux mandatory access control (MAC) and Middleware mandatory access control (MMAC) on Android.
AOSP	<p>The Android code base distributed by Google (see http://source.android.com/source/downloading.html). Release 4.4 contains SELinux support that is described at http://source.android.com/devices/tech/security/se-linux.html.</p> <p>AOSP contains the core SELinux MAC functionality with the Install-time MMAC framework and policy as described in the Building the Policy section (also see http://seandroid.bitbucket.org/MergeStatus.html#2 for the latest status).</p> <p>AOSP also contains services to allow the updating of Intent Firewall policies, however currently no files are installed (although SEAndroid supplies a sample and update tools).</p>
SEAndroid	<p>The SEAndroid project enhancements are decreasing as more features move into AOSP. The additional SEAndroid features are:</p> <ul style="list-style-type: none"> a) Enhanced MAC policy (although this is almost in line with AOSP). b) Enhanced Install time MMAC

	<ul style="list-style-type: none">c) Installation of Enterprise Operations (EOps) configuration files.d) Sample EOps and Intent Firewall configuration files (the actual services are supplied by AOSP, replacing the SEAndroid Intent MMAC, Content Provider MMAC and Revoke Permissions services that are now obsolete).e) Tools to manage bundles for policy, EOps and Intent Firewall updates. <p>See the SE for Android project page for up-to-date details at http://seandroid.bitbucket.org/</p>
--	---

7.1.2 Useful Links

The following link describes how to validate SELinux in Android:

<http://source.android.com/devices/tech/security/se-linux.html>

The <http://seandroid.bitbucket.org/> pages describe the current merge status with AOSP, how to obtain the code, install SE for Android and the features that have been implemented. It also has useful reference papers with "Security Enhanced (SE) Android: Bringing Flexible MAC to Android" available at http://www.internetsociety.org/sites/default/files/02_4.pdf being a recommended read.

The white paper "[An Overview of Samsung KNOX](#)" also gives an overview of how SE for Android is being integrated with other security services (such as secure boot and integrity measurement) to help provide a more secure mobile platform.

7.1.3 Document Sections

The sections that follow cover:

1. Overview of Android package additions and updates to support MAC
2. Additional kernel LSM / SELinux support
3. SE for Android Classes & Permissions
4. SELinux commands and methods to support SE for Android
5. SELinux extensions for `init`
6. Policy construction and build
 - Build file locations
 - Policy files
 - Build tools
7. Logging and auditing
8. SE for Android `libselinux` additional functions
9. Object labeling configuration file details

7.2 SE for Android Project Updates

This gives a high level view of the new and updated projects to support SE for Android services and covers AOSP with any additional SEAndroid functions noted. These are not a complete set of updates, but give some idea of the scope.

`external/libselinux`

Provides the SELinux userspace function library that is installed on the device. It is based on the Linux version but has additional functions to support Android, for example:

`selinux_android_setcontext`

Sets the correct domain context when launching applications using **`setcon`** (3). Information contained in the `seapp_contexts` file is used to compute the correct context.

It is called by `frameworks/base/core/jni/com_android_internal_os_Zygote.cpp` when forking a new process and the `system/core/run-as/run-as.c` utility.

`selinux_android_setfilecon`

Sets the correct context on application directory / files using **`setfilecon`** (3). Information contained in the `seapp_contexts` file is used to compute the correct context.

The function is used by the package installer within `frameworks/native/cmds/installd/commands.c` via the `package_install()` and `make_user_data()` functions.

`selinux_android_restorecon`

`selinux_android_restorecon_pkgdir`

Basically these functions are used to label files and directories based on entries from the `file_contexts` and/or `seapp_contexts` files. They call a common handler (`selinux_android_restorecon_common()`) that will then relabel the requested directories and files. It will also handle recursive labeling of directories and files should a new app, `file_contexts` or `seapp_contexts` be installed (see the [Checking File Labels](#) section for further information).

The **`selinux_android_restorecon`** function is used by:

- `frameworks/native/cmds/installd/installd.c` when installing a new app.
- `frameworks/base/core/jni/android_os_SELinux.cpp` for the Java `native_restorecon` method.
- `frameworks/native/cmds/dumpstate/utils.c` when dumping Dalvik and stack traces to ensure correct label.

The **`selinux_android_restorecon_pkgdir`** function is used by:

- `frameworks/native/cmds/installd/commands.c` for the package `restorecon_data()` and `make_user_data()` functions.

selinux_android_seapp_context_reload

Loads the `seapp_contexts` file for `frameworks/native/cmds/installd/installd.c` when the package installer is loaded.

selinux_android_load_policy

Mounts the SELinux filesystem if SELinux is enabled and then calls **selinux_android_reload_policy** to load the policy into the kernel. Used by `system/core/init/init.c` to initialise SELinux.

selinux_android_reload_policy

Reloads the policy into the kernel. Used by `system/core/init/init.c` `selinux_reload_policy()` to reload policy after setting the `selinux.reload_policy` property.

selinux_android_use_data_policy

Used by `system/core/init/init.c` to decide which policy directory to load the `property_contexts` file from.

There is also a new labeling service for **selabel_lookup**(3) to query the Android `property_contexts` and `service_contexts` files.

Various Android services will also call (not a complete list):

- **selinux_status_updated**(3), **is_selinux_enabled**(3), to check whether anything changed within the SELinux environment (e.g. updated configuration files).
- **selinux_check_access**(3) to check if the source context has access permission for the class on the target context.
- **selinux_label_open**(3), **selabel_lookup**(3), **selinux_android_file_context_handle**, **selinux_android_prop_context_handle**, **setfilecon**(3), **setfscreatecon**(3) to manage file labeling.
- **selinux_lookup_best_match** called by `system/core/init/devices.c` when `ueventd` creates a device node as it may also create one or more symlinks (for block and PCI devices). Therefore a "best match" look-up for a device node is based on its real path, plus any links that may have been created (see <https://android.googlesource.com/platform/system/core/+b0ab94b7d5a888f0b6920b156e5c6a075fa0741a>, patches

<https://android.googlesource.com/platform/system/core/+b4c5200f51c3568f604a4557119ab545a6ddac94> and <https://android.googlesource.com/platform/external/libselinux/+be7f5e8814c4954aca51d3f95455c5d9d527658c>).

external/libsepol

Provides the policy userspace library. There are no specific updates to support SE for Android, also this library is not available on the device.

external/checkpolicy

Provides the policy build tool. Added support for MacOS X (darwin). Not available on the device as policy rebuilds are done in the development environment.

external/sepolicy

This is a policy specifically for the core components of SE for Android that looks much like the reference policy, but is contained in one directory that has the policy modules (*.te files), class / permission files etc.. The policy is built by the Android.mk file and the resulting policy is installed on the target device (as sepolicy) along with its supporting configuration files.

Device specific policy may be defined under the device directory as discussed in the [Device Specific Policy](#) section.

The policy can be updated along with its configuration files as discussed in the [Updating Policy](#) section.

The policy files are discussed in the [SELinux MAC Policy Files](#) section and support tools in [Policy Build Tools](#).

The Android specific object classes are described in the [SE for Android Classes & Permissions](#) section.

The directory also contains sample MMAC configuration files.

external/yaffs2

mkyaffs2image support for labeling and extended attributes (xattr)

packages/apps/SEAdmin

This is an Android application to manage the SE for Android environment (such as loading a new policy). Only available on SEAndroid build.

packages/apps/Settings

SELinux settings for the settings manager application.

bionic

Bionic is the Android libc that is a derived from the BSD standard C library code. It contains enhancements to support security providers such as SELinux.

bootable/recovery

Changes to manage file labeling on recovery.

build

Changes to build SE for Android and manage file labeling on images and OTA (over the air) target files.

frameworks/base

JNI - Add SELinux support functions such as `isSELinuxEnabled` and `setFSCreateCon`.

SELinux Java class and method definitions.

Checking Zygote connection contexts.

Managing file permissions for the package manager and wallpaper services.

SELinux additions to support install / run time MMAC and for SEAndroid the MMAC services.

system/core

SELinux support services for toolbox (e.g. `load_policy`, `runcon`).

SELinux support for system initialisation (e.g. `init`, `init.rc`).

SELinux support for auditing `avc`'s (`auditd`).

system/extras

SELinux support for the `ext4` file system. Note that the `make_ext4fs` utility is used to build these file systems and relies on the `file_contexts` file having all the relevant entries, if not, it will be unable to set the `security.selinux` xattr on the inode and fail.

kernel

There are a number of kernels that have been enhanced to support Linux Security Module (LSM) and SELinux services that are listed at:

<http://seandroid.bitbucket.org/BuildingKernels.html#9>

Note that the Android kernels are based on various versions (currently 3.4 for Goldfish used by the emulator), therefore the latest SELinux enhancements may not always be present. The [Kernel LSM / SELinux Support](#) section describes the Android kernel changes.

device

Build information for each device, details regarding SEAndroid supported devices can be found at:

<http://seandroid.bitbucket.org/BuildingKernels.html#9>

Device specific policy can be added as discussed in the [Building the Policy](#) and [Device Specific Policy](#) sections.

7.3 Kernel LSM / SELinux Support

The paper "Security Enhanced (SE) Android: Bringing Flexible MAC to Android" available at http://www.internetsociety.org/sites/default/files/02_4.pdf gives a good review of what did and didn't change in the kernel to support Android. This section briefly describes the only major change that was to support the Binder IPC service that consists of the following:

1. LSM hooks in the binder code (`drivers/staging/android/binder.c`) and (`include/linux/security.h`)
2. Default support for capabilities (`security/capability.c`) in case no other module is loaded.
3. Hooks in the LSM security module (`security/security.c`).
4. SELinux support for the binder object class and permissions (`security/selinux/include/classmap.h`) that are shown in the [SE for Android Classes & Permissions](#) section. Support for these permission checks are added to `security/selinux/hooks.c`.

7.4 SE for Android Classes & Permissions

SE for Android currently requires the kernel classes and permissions shown in [Appendix A - Object Classes and Permissions](#), and also specific Android classes and permissions that are shown in the following tables:

binder class - This is a kernel object to manage the Binder IPC service.	
Permission	Description (4 unique permissions)
<code>call</code>	Perform a binder IPC to a given target process (can A call B?).
<code>impersonate</code>	Perform a binder IPC on behalf of another process (can A impersonate B on an IPC?). Not currently used in policy but kernel (<code>selinux/hooks.c</code>) checks permission in <code>selinux_binder_transaction</code> call.
<code>set_context_mgr</code>	Register self as the Binder Context Manager aka <code>servicemanager</code> (global name service). Can A set the context manager to B, where normally $A = B$. See policy module <code>servicemanager.te</code> .
<code>transfer</code>	Transfer a binder reference to another process (can A transfer a binder reference to B?).

zygote class – This is a userspace object to manage the Android application loader. See Java <code>SELinux.checkSELinuxAccess()</code> in <code>frameworks/base/core/java/com/android/internal/os/ZygoteConnection.java</code>	
Permission	Description (4 unique permissions)
<code>specifyuids</code>	Peer may specify uid's or gid's.
<code>specifyrlimits</code>	Peer may specify rlimits.
<code>specifyinvokewith</code>	Peer may specify <code>--invoke-with</code> to launch Zygote with a wrapper command.
<code>specifyseinfo</code>	Specify a <code>seinfo</code> string for use in determining the app security label.

property_service class – This is a userspace object to manage the Android Property Service. See <code>check_mac_perms()</code> in <code>system/core/init/property_service.c</code>	
Permission	Description (1 unique permission)
<code>set</code>	Set a property.

service_manager class – This is a userspace object to manage the loading of Android services. See <code>check_mac_perms()</code> in <code>frameworks/native/cmds/servicemanager/service_manager.c</code>	
Permission	Description (3 unique permissions)
add	Add a service.
find	Find a service.
list	List services.

keystore_key class – This is a userspace object to manage the Android keystores. See <code>system/security/keystore/keystore.cpp</code>	
Permission	Description (16 unique permissions)
test	Test if keystore okay.
get	Get key.
insert	Insert / update key.
delete	Delete key.
exist	Check if key exists.
saw	Search for matching string.
reset	Reset keystore for primary user.
password	Generate new keystore password for primary user.
lock	Lock keystore.
unlock	Unlock keystore.
zero	Check if keystore empty.
sign	Sign data.
verify	Verify data.
grant	Add or remove access.
duplicate	Duplicate the key.
clear_uid	Clear keys for this uid.
reset_uid	Reset keys for this uid.
sync_uid	Sync keys for this uid.
password_uid	Generate new keystore password for this uid.

debuggerd class – This is a userspace object to allow file dumps. See <code>system/core/debuggerd/debuggerd.cpp</code>	
Permission	Description (2 unique permissions)
dump_tombstone	Write tombstone file.
dump_backtrace	Write backtrace file.

drmservice class – This is a userspace object to allow finer access control of the Digital Rights Management services. See <code>frameworks/av/drm/drmserver/DrmManagerService.cpp</code>	
Permission	Description (8 unique permissions)
<code>consumeRights</code>	Consume rights for content.
<code>setPlaybackStatus</code>	Set the playback state.
<code>openDecryptSession</code>	Open the DRM session for the requested DRM plugin.
<code>closeDecryptSession</code>	Close DRM session.
<code>initializeDecryptUnit</code>	Initialise the decrypt resources.
<code>decrypt</code>	Decrypt data stream.
<code>finalizeDecryptUnit</code>	Release DRM resources.
<code>pread</code>	Read the data stream.

7.5 SELinux Commands

A subset of the Linux SELinux commands have been implemented in SE for Android and are listed in Table 27. They are available as Toolbox commands (see `system/core/toolbox`) and can be run via `adb shell`, for example:

```
adb shell su 0 setenforce permissive
```

Table 27: SELinux enabled adb shell commands (in Android toolbox)

Command	Comment
<code>chcon</code>	Change security context of file: <code>chcon context path</code>
<code>getenforce</code>	Returns the current enforcing mode.
<code>getsebool</code>	Returns SELinux boolean value(s): <code>getsebool [-a boolean_name]</code>
<code>id</code>	If SELinux is enabled then the security context is automatically displayed.
<code>load_policy</code>	Load new policy into kernel: <code>load_policy policy-file</code>
<code>ls</code>	Supports <code>-Z</code> option to display security context.
<code>ps</code>	Supports <code>-Z</code> option to display security context.
<code>restorecon</code>	Restore file default security context as defined in the <code>file_contexts</code> or <code>seapp_contexts</code> files. The options are: <code>D</code> - data files, <code>F</code> - Force reset, <code>n</code> - do not change, <code>R/r</code> - Recursive change, <code>v</code> - Show changes. <code>restorecon [-DFnrRv] pathname</code>
<code>runcon</code>	Run command in specified security context: <code>runcon context program args...</code>
<code>setenforce</code>	Modify the SELinux enforcing mode: <code>setenforce [enforcing permissive 1 0]</code>
<code>setsebool</code>	Set SELinux boolean to a value (note that the cmd does not set the boolean across reboots):

	setsebool boolean_name [1 true on 0 false off]
--	--

7.6 SELinux Public Methods

The public methods implemented are equivalent to `libselinux` functions and are show in Table 28. They have been taken from `frameworks/base/core/java/android/os/SELinux.java`.

The SELinux class and its methods are not available in the Android SDK, however if developing SELinux enabled apps within AOSP then reflection would be used (see the `proguard.flags` and `Android.mk` files in `packages/apps/SEAdmin`).

Table 28: SELinux class public methods

<p><code>boolean isSELinuxEnabled()</code> Determine whether SELinux is enabled or disabled. Return <code>true</code> if SELinux is enabled.</p>
<p><code>boolean isSELinuxEnforced()</code> Determine whether SELinux is permissive or enforcing. Returns <code>true</code> if SELinux is enforcing.</p>
<p><code>boolean setSELinuxEnforce(boolean value)</code> Set whether SELinux is in permissive or enforcing modes. <code>value</code> of <code>true</code> sets SELinux to enforcing mode. Returns <code>true</code> if the desired mode was set.</p>
<p><code>boolean setFSContext(String context)</code> Sets the security context for newly created file objects. <code>context</code> is the security context to set. Returns <code>true</code> if the operation succeeded.</p>
<p><code>boolean setFileContext(String path, String context)</code> Change the security context of an existing file object. <code>path</code> represents the path of file object to relabel. <code>context</code> is the new security context to set . Returns <code>true</code> if the operation succeeded.</p>
<p><code>String getFileContext(String path)</code> Get the security context of a file object. <code>path</code> the pathname of the file object. Returns the requested security context or null.</p>
<p><code>String getPeerContext(FileDescriptor fd)</code> Get the security context of a peer socket. <code>FileDescriptor</code> is the file descriptor class of the peer socket. Returns the peer socket security context or null.</p>
<p><code>String getContext()</code> Gets the security context of the current process. Returns the current process security context or null.</p>
<p><code>String getPidContext(int pid)</code> Gets the security context of a given process id.</p>

`pid` an `int` representing the process id to check.
Returns the security context of the given `pid` or null.

String[] getBooleanNames()

Gets a list of the SELinux boolean names.
Return an array of strings containing the SELinux boolean names.

boolean getBooleanValue(String name)

Gets the value for the given SELinux boolean name.
`name` is the name of the SELinux boolean.
Returns true or false indicating whether the SELinux boolean is set or not.

boolean setBooleanValue(String name, boolean value)

Sets the value for the given SELinux boolean name. Note that this will be set the boolean permanently across reboots.
`name` is the name of the SELinux boolean.
`value` is the new value of the SELinux boolean.
Returns true if the operation succeeded.

boolean checkSELinuxAccess(String scon, String tcon, String tclass, String perm)

Check permissions between two security contexts.
`scon` is the source or subject security context.
`tcon` is the target or object security context.
`tclass` is the object security class name.
`perm` is the permission name.
Returns true if permission was granted.

boolean native_restorecon(String pathname)

Restores a file to its default SELinux security context. If the system is not compiled with SELinux, then true is automatically returned. If SELinux is compiled in, but disabled, then true is returned.
`pathname` is the pathname of the file to be relabeled.
Returns true if the relabeling succeeded.

boolean restorecon(String pathname)

Restores a file to its default SELinux security context. If the system is not compiled with SELinux, then true is automatically returned. If SELinux is compiled in, but disabled, then true is returned.
`pathname` is the pathname of the file to be relabeled.
Returns true if the relabeling succeeded.
exception `NullPointerException` if the `pathname` is a null object.

boolean restorecon(File file)

Restores a file to its default SELinux security context. If the system is not compiled with SELinux, then true is automatically returned. If SELinux is compiled in, but disabled, then true is returned.
`file` is the file object representing the path to be relabeled.
Returns true if the relabeling succeeded.
exception `NullPointerException` if the `file` is a null object.

7.7 Android Init Language SELinux Extensions

The Android init process language has been expanded to support SELinux as shown in Table 29. The complete Android init language description is available in the `system/core/init/readme.txt` file.

Table 29: SELinux init extensions

<p>seclabel <securitycontext></p> <p>service option: Change to security context before exec'ing this service. Primarily for use by services run from the rootfs, e.g. ueventd, adbd. Services on the system partition can instead use policy defined transitions based on their file security context. If not specified and no transition is defined in policy, defaults to the init context.</p>
<p>restorecon <path></p> <p>action command: Restore the file named by <path> to the security context specified in the <code>file_contexts</code> configuration. Not required for directories created by the <code>init.rc</code> as these are automatically labeled correctly by init.</p>
<p>restorecon_recursive <path> [<path>]*</p> <p>action command: Recursively restore the directory tree named by <path> to the security context specified in the <code>file_contexts</code> configuration. Do NOT use this with paths leading to shell-writable or app-writable directories, e.g. <code>/data/local/tmp</code>, <code>/data/data</code> or any prefix thereof.</p> <p>See the Managing Policy Updates section for further details.</p>
<p>setcon <securitycontext></p> <p>action command: Set the current process security context to the specified string. This is typically only used from <code>early-init</code> to set the init context before any other process is started (see <code>init.rc</code> example above).</p>
<p>setenforce 0 1</p> <p>action command: Set the SELinux system-wide enforcing status. 0 is permissive (i.e. log but do not deny), 1 is enforcing.</p>
<p>setsebool <name> <value></p> <p>action command: Set SELinux boolean <name> to <value>.</p> <p><value> may be <code>1 true on</code> or <code>0 false off</code></p>

Examples of their usage are shown in the following `init.rc` file segments:

```

system/core/rootdir/init.rc
...
on early-init
...

# Set the security context for the init process.
# This should occur before anything else (e.g. ueventd) is started.
setcon u:r:init:s0

# Set the security context of /adb_keys if present.
restorecon /adb_keys

start ueventd

```

```

...
on post-fs-data
...
# Reload policy from /data/security if present.
setprop selinux.reload_policy 1

# Set SELinux security contexts on upgrade or policy update.
restorecon_recursive /data
...
service ueventd /sbin/ueventd
class core
critical
seclabel u:r:ueventd:s0

```

7.8 Device Policy File Locations

Table 30 shows the SE for Android policy files with their default location when the device is built, and their alternate locations when devices are updated by other methods (such as OTA or via adb). The alternate locations are always checked first as if present they override the default location as discussed in the comments section of Table 30.

The `init` process will initially load the SELinux set of policy files from root (/). Once the `/data` partition setup has been completed (see `init.rc`) a policy reload is performed. This will check whether there is a valid policy at `/data/security/current` and load that if valid.

If safe mode, then only the root policy files will be loaded. A factory reset will wipe `/data` and will therefore revert to the original root policy files.

Table 30: Policy file locations

Default Location	Alternate Location	Comments
<code>/sepolicy</code>	<code>/data/security/current</code>	Any or all these files may be in the alternate directory as each component that requires them will look in the alternate first and then the default, however: <ol style="list-style-type: none"> 1. During a policy reload, if there is an <code>selinux_version</code> file in the alternate location, then the default location will be over-riden. If the policy has been updated via the <code>buildsebundle / SEAdmin</code> app process then this would be the case. 2. The alternate directory may be a symbolic link to another directory. For example the <code>buildsebundle / SEAdmin</code> app process adds a link to <code>/data/security/context</code> that holds the policy files 3. If the policy has been updated via the <code>buildsebundle / SEAdmin</code> app process, then the following will also be present: <ul style="list-style-type: none"> • <code>/data/security/bundle</code> will contain the <code>sepolicy_bundle</code> (the packed files) and a metadata directory containing a version file holding the last version number. • There will be <code>*_backup</code> policy files of the previous version that could be restored if
<code>/file_contexts</code>	<code>/data/security/current</code>	
<code>/seapp_contexts</code>	<code>/data/security/current</code>	
<code>/property_contexts</code>	<code>/data/security/current</code>	
<code>/service_contexts</code>	<code>/data/security/current</code>	
<code>/selinux_version</code>	<code>/data/security/current</code>	
<code>/system/etc/security/mac_permissions.xml</code>	<code>/data/security/current</code>	

		required. See the Build Bundle Tools - buildsebundle section for a worked example.
/system/etc/security/eops.xml	/data/security/eops	If the policy has been updated via the <code>buildeopbundle</code> / SEAdmin app process, then the following will also be present in the alternative location: <ul style="list-style-type: none"> • / data/security/eops/eops_metadata/version file holding the last version number. See the Build Bundle Tools - buildeopbundle section for a worked example.
/data/system/ifw/ifw.xml	/data/secure/system/ifw (default for encrypted systems)	This file is not installed by default and note that the Intent Firewall service will read any file from <code>/data/system/ifw/</code> so long as it has an <code>.xml</code> extension. If required would be built and delivered by the <code>buildifwbundle</code> / SEAdmin app process, with the following also present in the default location: <ul style="list-style-type: none"> • / data/system/ifw/metadata/gservices.version file holding the last version number. See the Build Bundle Tools - buildifwbundle section for a worked example.
/system/etc/sepolicy.recovery	none	Only used for recovery.

7.9 Building the Policy

This section covers building of SELinux MAC and Install-time MMAC policies. The file formats of SE for Android specific configuration files are detailed in the [Policy File Configuration Detail](#) section with examples.

7.9.1 SELinux MAC Policy Files

The policy files are contained in the `external/sepolicy` directory, however there may also be additional policy configuration files to enable specific device features under the `device/<vendor>/<device>/sepolicy` directory (see the [Device Specific Policy](#) section). Once generated the policy and its supporting configuration files are installed on the device as part of the build process.

7.9.1.1 Policy Build Files

The following files are used to build the kernel binary policy file that is named `sepolicy` and installed by default in the root directory.

```
access_vectors, security_classes
```

These have been modified to support the new SE for Android classes and permissions (although they still contain the unused Linux userspace items).

```
initial_sids, initial_sids_contexts
```

Contains the system initialisation (before policy is loaded) and failsafe (for objects that would not otherwise have a valid label).

`fs_use, genfs_contexts, port_contexts`

For flexibility of policy building, these files have been separated to allow additional policy files to be defined for specific devices as discussed below.

`users, roles`

These define the only user (`u`) and role (`r`) used by the policy, although there is no reason why others cannot be added.

`mls`

Contains the constraints to be applied to the defined classes and permissions.

`global_macros, mls_macro, te_macros`

These contain the m4 macros that expand the policy files to build a policy in the kernel policy language as described in the [Kernel Policy Language](#) section. The policy can then be compiled by `checkpolicy` (8).

`attributes`

Contains the attribute names (forming the `attribute` statements) that will be used to group `type_identifiers` defined by the policy.

`policy_capabilities`

Contains the policy capabilities enabled for the kernel policy (see `politycap` statement).

`*.te`

The `*.te` files are the core policy module definition files. These are the same format as the standard reference policy and are expanded by the m4 macros. There is (generally) one `.te` file for each domain/service defined containing the policy rules.

7.9.1.2 Policy Configuration Files

These files will be installed on the device and used to compute SE for Android security contexts (see the [Checking File Labels](#) section for further information).

`file_contexts`

Contains default file contexts for setting the filesystem as Linux based SELinux (note that it does not contain entries for labeling apps or their data stores, the `seapp_contexts` file is used for that purpose). The format of this file is defined in [file_contexts](#) (5). The file is installed by default in the root directory. SE for Android services (such as [restorecon](#)) will first check for this file at (this is where updated files would be placed):

`/data/security/current/file_contexts`

If not present they will then check the root directory:

`/file_contexts`

`property_contexts`

Contains default contexts for Android property services as discussed in the [property_contexts File](#) section. The file is installed by default in the root directory. The SE for Android initialisation / reload process will first check for this file at (this is where updated files would be placed):

```
/data/security/property_contexts
```

If not present they will then check the root directory:

```
/property_contexts
```

service_contexts

Contains default contexts for Android services as discussed in the [service_contexts File](#) section. The file is installed by default in the root directory. The SE for Android initialisation / reload process will first check for this file at (this is where updated files would be placed):

```
/data/security/service_contexts
```

If not present they will then check the root directory:

```
/service_contexts
```

seapp_contexts

Contains information to allow domain or file contexts to be computed based on parameters as discussed in the [seapp_contexts File](#) section. The file is installed by default in the root directory. The SE for Android initialisation / reload process will first check for this file at (this is where updated files would be placed):

```
/data/security/current/seapp_contexts
```

If not present they will then check the root directory:

```
/seapp_contexts
```

selinux-network.sh

This will not be processed by the SE for Android build, it must be specifically added to the device make file if required. See the [selinux-network.sh Configuration](#) section for details on configuring this file.

The following files will be built as part of the build process and installed on the device:

sepolicy

The kernel binary policy. The SE for Android initialisation / reload process will first check for this file at (this is where updated files would be placed):

```
/data/security/current/sepolicy
```

If not present they will then check the root directory:

```
/sepolicy
```

For reference, the policy text file is available at:

```
out/target/product/<device>/obj/ETC/sepolicy_intermediates/policy.conf
```

The compiled kernel policy (sepolicy) is also in this directory along with `policy.conf.dontaudit` and `sepolicy.dontaudit` files that have the `dontaudit` rules removed.

`sepolicy.recovery`

A recovery policy is installed at `system/etc/sepolicy.recovery`. It is build with the macro `target_recovery = true` that will add additional rules defined in the `recovery.te` module (see `Android.mk` and `te_macros`). For reference the recovery policy text file is available at:

```
out/target/product/<device>/obj/ETC/sepolicy.recovery_intermediates/policy_recovery.conf
```

`selinux_version`

The `selinux_version` file is generated containing the `BUILD_FINGERPRINT` that the policy was built against. Its existence is used at boot time, policy upgrades or reloads to determine whether the policy configuration files should be read from `/data/security/current` or root (`/`). The `mac_permissions.xml` would also be read from either `/data/security/current` or `/system/etc/security`.

7.9.2 Install-time MMAC Policy File

The Install-time MMAC is part of AOSP and SEAndroid policy build that is always enabled. The file that configures policy is `mac_permissions.xml` and its format is discussed in the [Install-time MMAC Configuration File](#) section. The file is installed by default at:

```
/system/etc/security/mac_permissions.xml
```

The SE for Android initialisation / reload process will first check for this file at:

```
/data/security/current/mac_permissions.xml
```

This file can be replaced through `BOARD_SEPOLICY_REPLACE` or appended to by the `BOARD_SEPOLICY_UNION` variable as described in the [Device Specific Policy](#) section.

This file can be updated along with all other MAC policy files as described in the [Updating Policy](#) section.

The main code for the service is `frameworks/base/services/java/com/android/server/pm/SELinuxMMAC.java`, however it does hook into other Android services such as `PackageManagerService.java`. Note that AOSP and SEAndroid builds only differ in that SEAndroid will not install or load an app if there is no matching entry in the `mac_permissions.xml` file when there is no `<default>` entry.

7.9.3 Device Specific Policy

Some of this section has been extracted from the `external/sepolicy/README` file that should be checked in case there have been updates. It describes how files in `external/sepolicy` can be manipulated during the build process to reflect

requirements of different device vendors whose policy files would normally be located in the `device/<vendor>/<device>/sepolicy` directory.

Important Note: SE for Android policy has a number of `neverallow` rules defined in the core policy to ensure that `allow` rules are never added to domains that would weaken security. However developers may need to customise their device policies, and as a consequence they may fail one or more of these rules. If so, then this thread may be useful:

<http://marc.info/?l=seandroid-list&m=141116103611797&w=2>

7.9.3.1 Managing Device Policy File

Additional per device policy files may be added or removed during the policy build and are configured through the use of the following four variables that would be added to the device `BoardConfig.mk` file:

```
BOARD_SEPOLICY_DIRS
BOARD_SEPOLICY_UNION
BOARD_SEPOLICY_REPLACE
BOARD_SEPOLICY_IGNORE
```

They are used as follows:

BOARD_SEPOLICY_DIRS

`BOARD_SEPOLICY_DIRS` contains a list of directories to search for files listed by the `BOARD_SEPOLICY_UNION` and `BOARD_SEPOLICY_REPLACE` variables. Order matters in this list. e.g. If the following is defined:

```
BOARD_SEPOLICY_UNION := widget.te
```

and there are two instances of `widget.te` files on the `BOARD_SEPOLICY_DIRS` search path, the first one found (at the first search directory containing the file) gets processed first. Reviewing the devices `policy.conf`⁵⁴ will help sort out ordering issues and is located at:

```
out/target/product/<device>/obj/ETC/sepolicy_intermediates/policy.conf
```

BOARD_SEPOLICY_UNION

`BOARD_SEPOLICY_UNION` is a list of files that will be "unioned", i.e. concatenated at the END of their respective files in `external/sepolicy`

Note to add a unique/new file this variable would be used.

BOARD_SEPOLICY_REPLACE

`BOARD_SEPOLICY_REPLACE` is a list of files that will be used instead of the corresponding file in `external/sepolicy`.

BOARD_SEPOLICY_IGNORE

`BOARD_SEPOLICY_IGNORE` is a list of paths (directory + filename) of files that are not to be included in the resulting policy. This list is passed to `filter-out`

⁵⁴ The `policy.conf` file contains the policy language statements as described the [Kernel Policy Language](#) section. These define the policy that will be enforced and devices labeled.

to remove any paths to be ignored. This is useful if there are numerous configuration directories that contain a file, and that file is NOT to be included in the resulting policy, either by `BOARD_SEPOLICY_UNION` or `BOARD_SEPOLICY_REPLACE`.

For example, suppose the following:

```
BOARD_SEPOLICY_DIRS += X Y
BOARD_SEPOLICY_REPLACE += A
BOARD_SEPOLICY_IGNORE += X/A
```

with directories X and Y containing a copy of file A. The resulting policy is created by using Y/A only, thus X/A was ignored.

Error Handling:

1. It is an error to specify a `BOARD_POLICY_REPLACE` file that does not exist in `external/sepolicy`.
2. It is an error to specify a `BOARD_POLICY_REPLACE` file that appears multiple times on the policy search path defined by `BOARD_SEPOLICY_DIRS`.

For example, if `shell.te` is specified in `BOARD_SEPOLICY_REPLACE` and `BOARD_SEPOLICY_DIRS` is set to:

```
vendor/widget/common/sepolicy device/widget/x/sepolicy
```

and `shell.te` appears in both locations, it is an error. Unless it is in `BOARD_SEPOLICY_IGNORE` to be filtered out. See `BOARD_SEPOLICY_IGNORE` for more details.

3. It is an error to specify the same file name in both `BOARD_POLICY_REPLACE` and `BOARD_POLICY_UNION`.
4. It is an error to specify a `BOARD_SEPOLICY_DIRS` that has no entries when specifying `BOARD_SEPOLICY_REPLACE`.

Examples:

Two example `BoardConfig.mk` entries showing the use of `BOARD_SEPOLICY_UNION` that will take files referenced in `BOARD_SEPOLICY_DIRS` and add their contents to the end of the respective files in `external/sepolicy`, it will also include those not in `external/sepolicy`, and `BOARD_SEPOLICY_REPLACE` that will replace those files in `external/sepolicy`.

Example 1:

```
BOARD_SEPOLICY_DIRS := \
    device/samsung/tuna/sepolicy

BOARD_SEPOLICY_UNION := \
    genfs_contexts \
    file_contexts \
    sepolicy.te
```

Example 2:

```
BOARD_SEPOLICY_DIRS := \  
    device/demo_vendor/se4a_device/sepolicy  
  
BOARD_SEPOLICY_UNION := \  
    netclient_server.te \  
    secmark.te \  
    seapp_contexts \  
    keys.conf \  
    mac_permissions.xml  
  
BOARD_SEPOLICY_REPLACE := \  
    selinux-network.sh
```

7.9.4 Build Tools

The kernel policy is compiled using **checkpolicy**(8) via the `external/sepolicy/Android.mk` file. There are also a number of SE for Android specific tools used to assist in policy configuration that are described in [Policy Build Tools](#), with a summary as follows:

`checkfc` - Used to parse the `file_contexts` file against the binary policy `sepolicy`. This is to ensure all file contexts are valid for the policy. There is a `-p` option that is used to validate the contexts defined in the `property_contexts` or `service_contexts` file.

`checkseapp` - Used to validate the `seapp_contexts` file entries against the binary policy `sepolicy`.

`insertkeys.py` - Used to replace keywords in the signature sections of the `mac_permissions.xml` file with information obtained from pem files. This uses information contained in the `external/sepolicy/keys.conf` file that is detailed in the [insertkeys.py](#) tools section.

Note that the tools listed below are not built as part of the standard build process, therefore use `make <tool_name>` except where indicated.

`post_process_mac_perms` - Assists in generating new entries in an existing `mac_permissions.xml` file (also see `setool`). There is no make target for this python script, so either move to `HOST_EXECUTABLE` or execute directly (e.g. `$PREFIX/external/sepolicy/tools/post_process_mac_perms`).

`sepolicy-analyze` - Used to analyze the kernel policy file (`sepolicy`) for equivalent or different type pairs, or duplicate allow rules.

`sepolicy-check` - Used to check the kernel policy file (`sepolicy`) for allow rules based on source / target types, class and a single permission.

`build<???`>`bundle` - Used to build bundles for `sepolicy` et al., `eop.xml` or `ifw.xml` files to handle policy updates. Not available on AOSP.

`setool` - Assists in generating new entries for the `mac_permissions.xml` file. It will extract certificates from one or more packages then generate the package sections. Its output may need to be modified before inclusion in the master file as detailed in the [setool](#) tools section. Not available on AOSP.

7.9.5 Miscellaneous Information

7.9.5.1 SELinux Policy Versions

The default SELinux policy version is 26 that requires a kernel ≥ 3.0 and is set in `external/sepolicy/Android.mk` as follows:

```
POLICYVERS ?= 26
```

If an older kernel must be supported `POLICYVERS` can be set as an environment variable as follows:

```
export POLICYVERS=24
```

Information regarding policy versions can be found in the [Policy Versions](#) section that also gives information on the kernel versions required.

7.9.5.2 SELinux Policy Booleans

AOSP does not allow the use of booleans and the Android Compatibility Test Suite will specifically check and fail if they are present. They may still be defined in SEAndroid policy though.

7.9.5.3 Setting Permissive / Enforcing Mode

Version 4.4 is always started in enforcing mode, although some domains may be running in 'per-domain' permissive mode due to the [permissive](#) statement being present in the policy. Also in 4.4 there is a `permissive_or_unconfined` macro (see `te_macros` policy file) that can be controlled via the `FORCE_PERMISSIVE_TO_UNCONFINED` flag defined in the policy `Android.mk` file (see comments in `Android.mk` for the detail).

These are ways to set permissive or enforcing mode:

1. To set across reboots, add the `setenforce` command to `init.rc` or `init.<board>.rc` files.
2. Using `adb` to run the `setenforce` command (not set across reboots):

```
# 1 = enforcing 0 = permissive
adb shell su 0 setenforce 1
```

If running the emulator the following may also be used:

```
emulator -selinux permissive
```

```
emulator -qemu -append androidboot.selinux=permissive
```

7.9.5.4 Checking File Labels

Checks on file labels take place at boot time, policy upgrades / reloads, app installation / upgrade, and via adb using restorecon/chcon. Depending on whether data, app or system areas are being labeled by the various restorecon services, there are two files involved: `file_contexts` for all areas other than `/data/data` and `/data/user` where the `seapp_contexts` file is used. Their use and format are described in the [Policy Configuration Files](#) and [seapp_contexts File](#) sections.

To determine whether either of these two files have changed:

1. The `file_contexts` file has an SHA hash taken when loaded. This will be used when a recursive restorecon request is made and will be written to the `xattr` inode entry of "security.restorecon_last" as files are labeled (except `/sys` files). When restorecon is run again (policy reload/update etc.), the `xattr` hash will be compared to the loaded `file_contexts` file hash, thus allowing automatic relabeling should the file change.
2. The `seapp_contexts` file has an SHA hash taken when loaded and stored as `/data/system/seapp_hash` by `SELinuxMMAC.java`. This is used to determine whether a recursive restorecon should be carried out on the `/data/data` and `data/user` directories by the package manager.

7.10 Updating Policy Files

This is covered at <http://seandroid.bitbucket.org/PolicyUpdates.html> in some detail and there are worked examples in the following sections:

- [Build Bundle Tools - buildsebundle](#) - This includes using an intent to update policy.
- [Build Bundle Tools - buildeopbundle](#)
- [Build Bundle Tools - buildifwbundle](#)

There are also details in the [Device Policy File Locations](#) section.

The Android services that manage the updates are contained in the following java source files within the `frameworks/base/services/java/com/android/server/updates` directory:

- `SELinuxPolicyInstallReceiver.java`
- `IntentFirewallInstallReceiver.java`
- `EopsInstallReceiver.java`

7.10.1.1 Local Policy Update

An example of loading a different policy via adb is described at <http://seandroid.bitbucket.org/AddressingHiddenDenials.html#13>, however this is an alternate method:

1. Modify the required policy source files including the relevant device policy modules. Rebuild the kernel policy file by:

```
make sepolicy
```

2. Copy the policy file to the device (it copies the new policy to the alternate directory so that it is picked up by the reload property):

```
adb push out/target/product/<device>/root/sepolicy /data/security/current
```

3. Then load the new policy by:

```
adb shell su setprop selinux.reload_policy 1
```

7.11 Logging and Auditing

SE for Android 4.4 now supports auditing of SELinux events via the AOSP logger service that can be viewed using `logcat`, for example:

```
adb logcat > logcat.log
```

Example SELinux audit events (avc denials) are:

```
W/iptables( 92): type=1400 audit(0.0:18): avc: denied { relabelto } for
scontext=u:r:init:s0 tcontext=u:object_r:net_apps_packet:s0 tclass=packet
W/iptables( 92): type=1300 audit(0.0:18): arch=40000028 syscall=294 per=800000 success=no
exit=-13 a0=4 a1=0 a2=40 a3=b845a468 items=0 ppid=54 auid=4294967295 uid=0 gid=0 euid=0
suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=(none) ses=4294967295 exe="/system/bin/iptables"
subj=u:r:init:s0 key=(null)
...
...
...
W/com.se4android.netclient( 3168): type=1400 audit(0.0:200): avc: denied { send } for
comm=4173796E635461736B202331 saddr=10.0.2.15 src=43397 daddr=10.0.2.15 dest=9999 netif=lo
scontext=u:r:netclient_app:s0:c15,c256 tcontext=u:object_r:unlabeled:s0 tclass=packet
W/com.se4android.netclient( 3168): type=1300 audit(0.0:200): arch=40000028 syscall=283
per=800000 success=no exit=-111 a0=14 a1=abf4e6c4 a2=1c a3=b6f98e98 items=0 ppid=66
auid=4294967295 uid=10015 gid=10015 euid=10015 fsuid=10015 egid=10015 sgid=10015
fsgid=10015 tty=(none) ses=4294967295 comm=4173796E635461736B202331
exe="/system/bin/app_process32" subj=u:r:netclient_app:s0:c15,c256 key=(null)
...
...
E/SE4A-NetClient( 3141): java.net.ConnectException: failed to connect to /10.0.2.15 (port
9999): connect failed: ECONNREFUSED (Connection refused)
```

The `audit2allow` (1) command can be used to create policy rules as follows:

```
audit2allow -p out/target/product/<device>/root/sepolicy < logcat.log > policy.te
```

The result from the above avc denials would be:

```
#===== init =====
allow init net_apps_packet:packet relabelto;

#===== netclient_app =====
allow netclient_app unlabeled:packet send;
```

As the requirement of the app is to only accept packets labeled `net_apps_packet` via `iptables` (8) `SECMARK`, the `relabelto allow` rule was added to the device policy (see the [selinux-network.sh Configuration](#) section regarding `SECMARK`).

Note that before the auditing daemon is loaded messages will be logged in the kernel buffers that can be read using `dmesg` (1):

```
adb shell su 0 dmesg
```

7.12 Policy File Configuration Detail

This section details the specific SE for Android policy configuration files (i.e. those not used by 'standard' Linux based SELinux). Where those files are used to compute contexts using the SE for Android `libselinux` functions, those functions are also described with examples.

7.12.1 SELinux MAC Configuration Files

7.12.1.1 seapp_contexts File

This file is loaded and sorted into memory using the precedence rules explained below on first use by of one of the following SE for Android `libselinux` functions:

`selinux_android_setcontext` - Computes process security contexts.

`selinux_android_setfilecon` - Computes file/directory security contexts.

`selinux_android_seapp_context_reload` will reload the file.

The build process supports additional `seapp_contexts` files to allow devices to specify their entries as described in the [Device Specific Policy](#) section.

The following sections will show:

1. The default `external/sepolicy/seapp_contexts` file entries.
2. A description of the `seapp_contexts` entries and their usage.
3. A brief description of how a context is computed using either the `selinux_android_setcontext` or `selinux_android_setfilecon` function using the `seapp_contexts` file entries.
4. Examples of computed domain and directory contexts for various apps.

7.12.1.1.1 Default Entries

The default SEAndroid `external/sepolicy/seapp_contexts` file contains the following entries:

```
isSystemServer=true domain=system_server
user=system domain=system_app type=system_app_data_file
user=bluetooth domain=bluetooth type=bluetooth_data_file
user=nfc domain=nfc type=nfc_data_file
user=radio domain=radio type=radio_data_file
```

```
user=shared_relro domain=shared_relro
user=shell domain=shell type=shell_data_file
user=_isolated domain=isolated_app levelFrom=user
user=_app seinfo=platform domain=platform_app type=app_data_file levelFrom=user
user=_app domain=untrusted_app type=app_data_file levelFrom=user
```

7.12.1.1.2 Entry Definitions

The following has been extracted from the default file with some additional comments that describe the parameters and how they are used to compute a context:

Input selectors from seapp_contexts file:

```
isSystemServer (boolean)
isOwner (boolean)
user (string)
seinfo (string)
name (string) - A package name e.g. com.example.demo
path (string) - A path name (added to ensure correct labeling of some files).
sebool (string) - The boolean must be 'active' (enabled/true)
```

Notes:

isSystemServer=true can only be used once.

An unspecified isSystemServer defaults to false.

isOwner=true will only match for owner/primary user.

isOwner=false will only match for secondary users.

If unspecified, the entry can match either case.

An unspecified string selector will match any value.

A user string selector that ends in * will perform a prefix match.

user=_app will match any regular app UID.

user=_isolated will match any isolated service UID.

All specified input selectors in an entry must match (i.e. logical AND).

Matching is case-insensitive.

Precedence rules:

- 1) isSystemServer=true before isSystemServer=false.
- 2) Specified isOwner= before unspecified isOwner=boolean.
- 3) Specified user= string before unspecified user= string.
- 4) Fixed user= string before user= prefix (i.e. ending in *).
- 5) Longer user= prefix before shorter user= prefix.
- 6) Specified seinfo= string before unspecified seinfo= string.
- 7) Specified name= string before unspecified name= string.
- 8) Specified path= string before unspecified path= string.
- 9) Specified sebool= string before unspecified sebool= string.

Outputs:

```
domain (string) - The type component of a process context.
type (string) - The type component of a file/directory context.
levelFrom (string; one of none, all, app, or user) - A level that will be
automatically computed based on the parameter.
level (string) - A predefined level (e.g. s0:c1022.c1023)
```

Notes:

Only entries that specify `domain=` will be used for app process labeling.

Only entries that specify `type=` will be used for app directory labeling.

`levelFrom=user` is only supported for `_app` or `_isolated` UIDs.

`levelFrom=app` or `levelFrom=all` is only supported for `_app` UIDs.

`level` may be used to specify a fixed level for any UID.

7.12.1.1.3 Computing a Context

This section explains the process to compute a context using parameters supplied by the `selinux_android_setcontext`, `selinux_android_setfilecon`, `selinux_android_restorecon` and `selinux_android_restorecon_pkgdir` functions plus the precedence sorted contents of the `seapp_contexts` file, some examples are then shown.

The context is computed first by converting the `uid` parameter to a string that is used to match the user component in the `seapp_contexts` entry as follows:

- a) If an Android system service, the `uid` parameter is converted to a username string via an internal Android table (e.g. "radio", "system").
- b) If an isolated service the `_isolated` string is used as the username.
- c) For any other app or service `_app` string is used as the username.

Then cycling through each precedence sorted `seapp_contexts` entry, check each component as follows until a match is found or generate an error log entry:

- The `isSystemServer` component is matched against the `isSystemServer` parameter. If a match or `isSystemServer` not present check remaining components, else skip entry.
- The `isOwner` boolean determines whether the remaining components should be checked or skip this entry. The rules are:
 - a) If `isOwner` not present then check remaining components.
 - b) If set `true` and the `uid` computes to the owner or primary user then check remaining components, else skip this entry.
 - c) If set `false` and the `uid` computes to a secondary user then check remaining components, else skip this entry.
- The computed `username` is matched against the `user` component. If a match or `user` not present check remaining components, else skip entry.
- The `seinfo` component is matched against the `seinfo` parameter. If a match or `seinfo` not present check remaining components, else skip entry.
- The `name` component is matched against the `pkgname` parameter. If a match or `name` not present check remaining components, else skip entry.

- The `path` component is matched against a computed path of a file having its context restored via one of the `restorecon` functions. If a match or `path` not present check remaining components, else skip entry.
- The `domain` component is used to set the process context for the `selinux_android_setcontext` function and must match a type declared in the policy. If `domain` not present skip this entry.
- The `type` component is used to set the file context for the `selinux_android_setfilecon` function and must match a type declared in the policy. If `type` not present skip this entry.
- The `sebool` parameter if present will be matched against the SELinux boolean name list. If `sebool` present then boolean must be active.
- The `levelFrom` and `level` components if present will be used to determine the `level` component of the security context as follows:
 - a) if `levelFrom=none` then use current level.
 - b) else if `levelFrom=app` then compute a category pair based on a derived app id with a starting base of `c512, c768` base.
 - c) else if `levelFrom=user` then compute a category pair based on a derived user id with a starting base of `c0, c256` base.
 - d) else if `levelFrom=all` then compute a category pair based on a derived app id with a starting base of `c512, c768` base, and also compute another category pair based on a derived user id with a starting base of `c0, c256` base.
 - e) else if `level` has a value use this as the context level.

The overall objective is that the computed levels should never be the same for different apps, users, or a combination of both. By encoding each ID as a category pair, up to 2^{16} app IDs and up to 2^{16} user IDs within the 1024 categories can be represented, including the `levelFrom=all` or mixed usage of `levelFrom=app` and `levelFrom=user` without concern.

If a valid entry is found, then:

1. If a context for the `selinux_android_setcontext` function has been computed, it is validated against policy, if correct `setcon(3)` is used to set the process context.
2. If a context for `selinux_android_setfilecon`, `selinux_android_restorecon` or `selinux_android_restorecon_pkgdir` functions have been computed, it is validated against policy, if correct `setfilecon(3)` or `lsetfilecon(3)` are used to set the context for labeling the file.

If a valid entry is not found an error is generated in the log currently formatted as follows:

```
seapp_context_lookup: No match for app with uid uid, seinfo seinfo,  
name pkgname
```

Computing process context examples:

The following is an example taken as the system server is loaded:

```
selinux_android_setcontext() parameters:  
uid          1000  
isSystemServer true  
seinfo       null  
pkgname      null  
  
seapp_contexts lookup parameters:  
uid          1000  
isSystemServer true  
seinfo       null  
pkgname      null  
path         null  
  
Matching seapp_contexts entry:  
  isSystemServer=true domain=system_server  
  
Outputs:  
  domain      system_server  
  level       s0  
  
Computed context = u:r:system_server:s0  
username computed from uid = system  
  
Result using ps -Z command:  
LABEL                USER      PID  PPID  NAME  
u:r:system_server:s0 system    836  63   system_server
```

This is the 'radio' application that is part of the platform:

```
selinux_android_setcontext() parameters:  
uid          1001  
isSystemServer false  
seinfo       platform  
pkgname      com.android.phone  
  
seapp_contexts lookup parameters:  
uid          1001 (computes user=radio entry)  
isSystemServer false  
seinfo       platform  
pkgname      com.android.phone  
path         null  
  
Matching seapp_contexts entry:  
  user=radio domain=radio type=radio_data_file  
  
Outputs:  
  domain      radio  
  level       s0  
  
Computed context = u:r:radio:s0  
username computed from uid = radio
```

```
Result using ps -Z command:
LABEL      USER    PID   PPID  NAME
u:r:radio:s0 radio  619   62    com.android.phone
```

This is the 'SEAndroid Admin Manager' application that is part of the seandroid release, however it is treated as an untrusted app (it is installed as a privileged app):

```
selinux_android_setcontext() parameters:
uid          10013
isSystemServer false
seinfo       default
pkgname      com.android.seandroid_admin

seapp_contexts lookup parameters:
isSystemServer false
uid          10013 (computes user=_app entry)
seinfo       default
pkgname      com.android.seandroid_admin
path         null

Matching seapp_contexts entry:
user=_app domain=untrusted_app type=app_data_file levelFrom=user

Outputs:
domain       untrusted_app
level        s0:c512,c768

Computed context = u:r:untrusted_app:s0:c512,c768
username computed from uid = u0_a13

Result using ps -Z command:
LABEL                                USER  PID  PPID  NAME
u:r:untrusted_app:s0:c512,c768 u0_a13 827  45    com.android.seandroid_admin
```

This is a third party app (com.example.runisolatedservice) to run an isolated service that has been installed as a privileged app (com.se4android.isolatedservice):

```
selinux_android_setcontext() parameters:
uid          10054
isSystemServer false
seinfo       default
pkgname      com.example.runisolatedservice

seapp_contexts lookup parameters:
isSystemServer false
uid          10054 (computes user=_app entry)
seinfo       default
pkgname      com.example.runisolatedservice
path         null

Matching seapp_contexts entry:
user=_app domain=untrusted_app type=app_data_file levelFrom=user

Outputs:
```

```
domain untrusted_app
level s0:c512,c768

Computed context = u:r:untrusted_app:s0:c512,c768
username computed from uid = u0_a54

Result using ps -Z command:
LABEL                USER  PID  PPID  NAME
u:r:untrusted_app:s0:c512,c768 u0_a54 1138 64   com.example.runisolatedservice
```

This is the isolated service installed as a privileged app (com.se4android.isolatedservice):

```
selinux_android_setcontext() parameters:
uid          99000
isSystemServer false
seinfo      default
pkgname     com.se4android.isolatedservice

seapp_contexts lookup parameters:
uid          99000 (computes user=_isolated entry)
isSystemServer false
seinfo      default
pkgname     com.se4android.isolatedservice
path        null

Matching seapp_contexts entry:
user=_isolated domain=isolated_app levelFrom=user
Note that uid's 99000-99999 are reserved for isolated services - see:
system/core/include/private/android_filesystem_config.h

Outputs:
domain isolated_app
level s0:c512,c768

Computed context = u:r:isolated_app:s0:c512,c768
username computed from uid = u0_i0

Result using ps -Z command:
LABEL                USER  PID  PPID  NAME
u:r:isolated_app:s0:c512,c768 u0_i0 1140 62   com.se4android.isolatedservice
```

Computing file context examples:

The following example is from the third party isolated app:

```
selinux_android_setfilecon() parameters:
pkgdir /data/data/com.example.runisolatedservice
pkgname com.example.runisolatedservice
seinfo default
uid     10046

seapp_contexts lookup parameters:
uid          10046 (computes user=_app entry)
isSystemServer false
seinfo      default
pkgname     com.example.runisolatedservice
path        null
```

```
Matching seapp_contexts entry:
  user=_app domain=untrusted_app type=app_data_file levelFrom=user

Outputs:
  type    app_data_file
  level   s0:c512,c768

Computed context = u:object_r:app_data_file:s0:c512,c768
username computed from uid = u0_a46

Result from /data/data directory using ls -Z command:
drwxr-x--x u0_a46 u0_a46 u:object_r:app_data_file:s0:c512,c768
com.example.runisolatedservice
```

7.12.1.2 property_contexts File

This file holds property service keys and their contexts that are matched against property names using `selabel_lookup(3)`. The returned context will then be used as the target context as described in the example below to determine whether the property is allowed or denied (see `system/core/init/property_service.c` and `init.c`).

The build process supports additional `property_contexts` files to allow devices to specify their entries as described in the [Device Specific Policy](#) section.

When `selabel_open(3)` is called specifying this file it will be read into memory and sorted using `qsort(3)`, subsequent calls using `selabel_lookup(3)` will then retrieve the appropriate context based on matching the `property_key`.

Example:

Use adb to reload the SELinux policy:

```
adb shell su 0 setprop selinux.reload_policy 1
```

Sample `property_contexts` file entries are:

```
# property_key          context to be applied on match
net.rmnet               u:object_r:net_radio_prop:s0
net.gprs                u:object_r:net_radio_prop:s0
net.ppp                 u:object_r:net_radio_prop:s0
net.qmi                 u:object_r:net_radio_prop:s0
net.lte                 u:object_r:net_radio_prop:s0
net.cdma                u:object_r:net_radio_prop:s0
net.dns                 u:object_r:net_radio_prop:s0
sys.usb.config          u:object_r:system_radio_prop:s0
ril                     u:object_r:radio_prop:s0
gsm                     u:object_r:radio_prop:s0
persist.radio           u:object_r:radio_prop:s0

debug                   u:object_r:debug_prop:s0
debug.db                u:object_r:debuggerd_prop:s0
log                     u:object_r:shell_prop:s0
service.adb.root        u:object_r:shell_prop:s0
service.adb.tcp.port    u:object_r:shell_prop:s0

persist.audio           u:object_r:audio_prop:s0
persist.logd            u:object_r:logd_prop:s0
persist.sys             u:object_r:system_prop:s0
persist.service         u:object_r:system_prop:s0
persist.service.bdroid  u:object_r:bluetooth_prop:s0
```

```
persist.security.      u:object_r:system_prop:s0
# selinux non-persistent properties
selinux.              u:object_r:security_prop:s0
# default property context (* is wild card match)
*                     u:object_r:default_prop:s0
```

The property service will call `selabel_lookup` with parameters consisting of the handle passed from `selabel_open`, a buffer to hold the returned context, and the object name "selinux.reload_policy" to look-up (the final parameter is not used):

```
selabel_lookup(handle, &context, "selinux.reload_policy", 1);
```

The following context will be returned as the look-up process will search for a match based on the length of the `property_key` (and will therefore match against "selinux."):

```
u:object_r:security_prop:s0
```

The property service will then validate whether the service has permission by issuing an `selinux_check_access` (3) call with the following parameters:

```
source context: u:r:su:s0
target context: u:object_r:security_prop:s0
class:         property_service
permission:    set
```

The policy would then decide whether to allow or deny the property request. Using the [sepolicy-check](#) tool will show that this will be denied by the current policy (a `dontaudit` rule is in the policy, however `su` runs permissive anyway):

```
sepolicy-check -s su -t security_prop -c property_service \
-p set -P out/target/product/generic/root/sepolicy
echo $?
1
```

7.12.1.3 service_contexts File

This file holds binder service keys and their contexts that are matched against binder object names using `selabel_lookup` (3). The returned context will then be used as the target context as described in the example below to determine whether the binder service is allowed or denied (see `frameworks/native/cmds/servicemanager/servicemanager.c`).

The build process supports additional `service_contexts` files to allow devices to specify their entries as described in the [Building the Policy](#) section.

When `selabel_open` (3) is called specifying this file it will be read into memory and sorted using `qsort` (3), subsequent calls using `selabel_lookup` (3) will then retrieve the appropriate context based on matching the `service_key`.

Example:

The healthd process wants to start a binder service "batterypropreg" (see frameworks/base/services/java/com/android/server/BatteryService.java).

Sample service_contexts file entries are:

```
# service_key          context to be applied on match
batteryproperties      u:object_r:healthd_service:s0
batterystats           u:object_r:system_server_service:s0
battery                u:object_r:system_server_service:s0

# default service context (* is wild card match)
*                      u:object_r:default_android_service:s0
```

The service manager will call selabel_lookup with parameters consisting of the handle passed from selabel_open, a buffer to hold the returned context, and the object name "batterypropreg" to look-up (the final parameter is not used):

```
selabel_lookup(handle, &context, "batterypropreg", 1);
```

The following context will be returned as the look-up process will search for a match based on the length of the service_key (and will therefore match against "battery"):

```
u:object_r:system_server_service:s0
```

The service manager will then validate whether the service has permission by issuing an **selinux_check_access** (3) call with the following parameters:

```
source context: u:r:healthd:s0
target context: u:object_r:system_server_service:s0
class:          service_manager
permission:     add
```

The policy would then decide whether to allow or deny the service. Using the [sepolicy-check](#) tool will show that this will be allowed by the current policy:

```
sepolicy-check -s healthd -t system_server_service \
-c service_manager -p add \
-P out/target/product/generic/root/sepolicy
Match found!
```

7.12.2 Install-time MMAC Configuration File

The mac_permissions.xml file is used to configure Install-time MMAC policy and provides x.509 certificate to seinfo string mapping so that Zygote spawns an app in the correct domain. See the [Computing a Process Context](#) section for how this is achieved using information also contained in the seapp_contexts file (AOSP and SEAndroid).

An example AOSP mac_permissions.xml file that shows the <default> entry is:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<policy>
  <!-- Platform dev key in AOSP -->
  <signer signature="@PLATFORM" >
    <seinfo value="platform" />
  </signer>

  <!-- All other keys -->
  <default>
    <seinfo value="default" />
  </default>
</policy>
```

The `<signer signature=` entry may have the public base16 signing key present in the string or it may have an entry starting with `@`, then a keyword as shown that allows the key to be extracted from a pem file as discussed in the [insertkeys.py](#) section. If a base16 key is required, it can be extracted from a package using the [post_process_mac_perms](#) and [setool](#) utilities.

The build process supports additional `mac_permissions.xml` files to allow devices to specify their entries as described in the [Device Specific Policy](#) section. An example SEAndroid test device `mac_permissions.xml` file is:

```
<?xml version="1.0" encoding="utf-8"?>
<policy>

  <!-- NET_APPS key and seinfo for SE4A-NetClient & SE4A-NetServer apps.
  Note that if these had to be signed as @PLATFORM apps, then these
  entries would be added to the external/sepolicy/mac_permissions.xml
  file <signer signature="@PLATFORM" > entry (or the master file
  replaced by using BOARD_SEPOLICY_REPLACE in BoardConfig.mk). This is
  because multiple signer entries with the same signature are not
  allowed.
  -->
  <signer signature="@NET_APPS" >
    <package name="com.se4android.netclient" >
      <seinfo value="netclient" />
    </package>
    <package name="com.se4android.netserver" >
      <seinfo value="netserver" />
    </package>
  </signer>
</policy>
```

7.12.2.1 Policy Rules

The following rules have been extracted from the SEAndroid `mac_permissions.xml` file:

1. A signature is a hex encoded X.509 certificate or a tag defined in `keys.conf` and is required for each `signer` tag.
2. A `signer` tag may contain a `seinfo` tag and multiple package stanzas.
3. A `default` tag is allowed that can contain policy for all apps not signed with a previously listed cert. It may not contain any inner package stanzas.

4. Each `signer/default/package` tag is allowed to contain one `seinfo` tag. This tag represents additional info that each app can use in setting a SELinux security context on the eventual process.
5. When a package is installed the following logic is used to determine what `seinfo` value, if any, is assigned:
 - a) All signatures used to sign the app are checked first.
 - b) If a `signer` stanza has inner `package` stanzas, those stanza will be checked to try and match the package name of the app. If the package name matches then that `seinfo` tag is used. If no inner package matches then the outer `seinfo` tag is assigned.
 - c) The `default` tag is consulted last if needed.
 - d) If none of the cases apply then the app is denied install on the device.

7.12.3 EOps MMAC Configuration File

The following text has been taken from the `SEAndroid/external/sepolicy/eops.xml` file (so check if any changes) with a few minor additions (there is also a simple example in the [EOps Example](#) section section).

EOps (enterprise operations) is a security extension to the App Operations (AppOps) feature already present on Android 4.3+ devices. AppOps lets users fine tune certain functionality requested by apps by allowing the user to toggle these access rights.

EOps seeks to provide an extension whereby a hardcoded set of rules explicitly denies certain access rights to groups of installed apps. This feature will allow an enterprise like control over certain operations. EOps is not a frontend for SELinux which somehow ties app permissions to SELinux contexts. Rather, it is an extension of the middleware MAC (MMAC) controls that currently exist on Android 4.3+ devices. EOps uses the `seinfo` labels that are already assigned to apps upon install.

The list of viable `op` tag names can be found in `AppOpsManager.java`. Just use the string version of each `op` without the `OP_` prefix in your policy tags. These are the current entries (July '14):

ACCESS_NOTIFICATIONS	AUDIO_ALARM_VOLUME	AUDIO_BLUETOOTH_VOLUME
AUDIO_MASTER_VOLUME	AUDIO_MEDIA_VOLUME	AUDIO_NOTIFICATION_VOLUME
AUDIO_RING_VOLUME	AUDIO_VOICE_VOLUME	CALL_PHONE
CAMERA	COARSE_LOCATION	FINE_LOCATION
GPS	MONITOR_HIGH_POWER_LOCATION	MONITOR_LOCATION
NEIGHBORING_CELLS	PLAY_AUDIO	POST_NOTIFICATION
READ_CALENDAR	READ_CALL_LOG	READ_CLIPBOARD
READ_CONTACTS	READ_ICC_SMS	READ_SMS
RECEIVE_EMERGENCY_SMS	RECEIVE_MMS	RECEIVE_SMS
RECEIVE_WAP_PUSH	RECORD_AUDIO	SEND_SMS
SYSTEM_ALERT_WINDOW	TAKE_AUDIO_FOCUS	TAKE_MEDIA_BUTTONS
VIBRATE	WAKE_LOCK	WIFI_SCAN
WRITE_CALENDAR	WRITE_CALL_LOG	WRITE_CLIPBOARD

WRITE_CONTACTS	WRITE_ICC_SMS	WRITE_SETTINGS
WRITE_SMS		

All operations listed in the policy will have a mode of ignored. This means that empty data sets are returned to the caller when an operation is requested. This shadow data will then allow certain apps to presumably still operate. However, AOSP currently is not constructed to return these empty data sets and therefore acts as if ignored operations are completely denied (blocked). Because of this some apps might crash or behave oddly if you apply certain eops policy. In addition, while AOSP seems to have hooked the proper places to check operations against policy some of those hooks fail to follow through with the denial and still allow the operation to occur. Because of this, EOps will also fail to make those distinctions and likewise fail to enforce certain operations. Once the AOSP pieces are in place to return legitimate fake data and enforce all operations then of course eops, by its design, will also do the same.

So, as long as AppOps is beta so too will EOps.

A debug tag is also allowed which flips on the global debugging log functionality inside AppOps.

Each stanza is grouped according to the `seinfo` tag that is assigned during install and thus creates a dependency with the `mac_permissions.xml` file. Each `seinfo` tag can then include any number of `op` tags. By including the `op(s)` you are simply removing that operation from working for all apps that have been installed with the listed `seinfo` label. These operations are restricted regardless of what any user controlled app ops policy may say. Any `op` not listed is therefore still subject to user control as normal.

Lastly, there is no permissive mode for EOps, once a policy is in place all ops listed are enforced.

The following is an example `eops.xml` policy file that will stop the camera being used by any system or default app. The file installation is shown in the [Build Bundle Tools - buildeopbundle](#) section:

```
<?xml version="1.0"?>
<app-ops>

  <debug/>

  <seinfo name="default">
    <op name="CAMERA"/>
  </seinfo>

  <seinfo name="system">
    <op name="CAMERA"/>
  </seinfo>

</app-ops>
```

7.12.4 Intent Firewall MMAC Configuration File

The example `external/sepolicy/ifw.xml` file has some comments regarding the tags, there is also an overview at <http://www.cis.syr.edu/~wedu/android/IntentFirewall/>.

The following is an example `ifw.xml` policy file that will stop the `DemoIsolatedService` being used by any app other than system apps or apps with the same signature. The file installation is shown in the [Build Bundle Tools - buildifwbundle](#) section:

```
<?xml version="1.0"?>
<rules>
  <!-- This will stop any app that is not a system app or
  does not have a matching signature from running the
  DemoIsolatedService service
  -->
  <service log="true" block="true">
    <not><sender type="system|signature"/></not>
    <intent-filter />
    <component-filter name="com.se4android.isolatedservice/.DemoIsolatedService"/>
  </service>
</rules>
```

The events will be in the event log under the `'ifw_intent_matched'` tag, for example:

```
adb logcat -b events
...
...
I/ifw_intent_matched( 390):[2,com.se4android.isolatedservice/.DemoIsolatedService
,10058,1,NULL,NULL,NULL,NULL,0]
...
...
```

7.13 Policy Build Tools

This section covers the policy build tools located at `external/sepolicy/tools`. They are `checkfc`, `checkseapp` and `insertkeys.py`. There is also `setool` that is not used as part of the build process but generates `mac_permissions.xml` entries from packages.

7.13.1 checkfc

The `checkfc` utility is used during the build process to validate the `file_contexts`, `property_contexts` and `service_contexts` files against policy. If validation fails `checkfc` will exit with an error.

Usage:

```
usage: checkfc [OPTIONS] sepolicy context_file
Parses a context file and checks for syntax errors.
The context_file is assumed to be a file_contexts file
unless explicitly switched by an option.

OPTIONS:
  -p : context file represents a property_context file.
```

Example validating `file_contexts` file (note: no `-p` parameter):

```
checkfc out/target/product/generic/root/sepolicy
out/target/product/generic/root/file_contexts
```

Example validating `property_contexts` file:

```
checkfc -p out/target/product/generic/root/sepolicy
out/target/product/generic/root/property_contexts
```

7.13.2 checkseapp

The `checkseapp` utility is used during the build process to validate the `seapp_contexts` file against policy. If validation fails `checkseapp` will exit with an error. `checkseapp` also consolidates matching entries and outputs the valid file stripped of comments.

Usage:

```
checkseapp [options] <input file>
Processes an seapp_contexts file specified by argument <input file> (default
stdin) and allows later declarations to override previous ones on a match.
Options:
  -h - print this help message
  -s - enable strict checking of duplicates. This causes the program to exit on
      a duplicate entry with a non-zero exit status
  -v - enable verbose debugging informations
  -p policy file - specify policy file for strict checking of output selectors
                  against the policy
  -o output file - specify output file, default is stdout
```

An example command with output to `stdout` is:

```
checkseapp -p out/target/product/se4a_device/root/sepolicy \
out/target/product/se4a_device/root/seapp_contexts

isSystemService=true domain=system_server
user=system domain=system_app type=system_data_file
user=bluetooth domain=bluetooth type=bluetooth_data_file
user=nfc domain=nfc type=nfc_data_file
user=radio domain=radio type=radio_data_file
user=shared_relo domain=shared_relo
user=shell domain=shell type=shell_data_file
user=_isolated domain=isolated_app
user=_app seinfo=platform domain=platform_app type=app_data_file
user=_app domain=untrusted_app type=app_data_file
user=_app seinfo=netclient domain=netclient_app type=net_apps_log_file levelFrom=app
user=_app seinfo=netserver domain=netserver_app type=net_apps_log_file levelFrom=app
```

7.13.3 insertkeys.py

The `insertkeys.py` utility is used during the build process to insert signing keys into the `mac_permissions.xml` file. The keys are obtained from pem files and the entries to be replaced start with an `@` followed by a keyword. The `external/sepolicy/keys.conf` file contains corresponding entries that allow mapping of pem files to signatures as discussed in the [keys.conf](#) section.

`insertkeys.py` generates base16 encodings from the base64 pem files as this is required by the Android Package Manager Service. The resulting `mac_permissions.xml` file will also be stripped of comments and whitespace.

Usage:

```
Usage: insertkeys.py [options] CONFIG_FILE MAC_PERMISSIONS_FILE [MAC_PERMISSIONS_FILE...]

This tool allows one to configure an automatic inclusion of signing keys into the
mac_permission.xml file(s) from the pem files. If multiple mac_permission.xml files are
included then they are unioned to produce a final version.

Options:
--version          show program's version number and exit
-h, --help        show this help message and exit
-v, --verbose     Print internal operations to stdout
-o FILE, --output=FILE Specify an output file, default is stdout
-c DIR, --cwd=DIR Specify a root (CWD) directory to run this from, itchdirs'
                  AFTER loading the config file
-t TARGET_BUILD_VARIANT, --target-build-variant=TARGET_BUILD_VARIANT
                  Specify the TARGET_BUILD_VARIANT, defaults to eng
-d KEY_DIRECTORY, --key-directory
                  Specify a parent directory for keys
```

7.13.3.1 keys.conf File

The `keys.conf` file is used by `insertkeys.py` for mapping the "@ . . ." tags in `mac_permissions.xml`, `mmac_types.xml` and `content_provider.xml` signature entries with public keys found in pem files. The configuration file can be used in `BOARD_SEPOLICY_UNION` and `BOARD_SEPOLICY_REPLACE` variables and is processed via m4 macros.

`insertkeys.py` allows for mapping any string contained in `TARGET_BUILD_VARIANT` with a specific path to a pem file. Typically `TARGET_BUILD_VARIANT` is either `user`, `eng` or `userdebug`. Additionally "ALL" may be specified to map a path to any string specified in `TARGET_BUILD_VARIANT`. All tags are matched verbatim and all options are matched lowercase. The options are "tolowered" automatically for the user, it is convention to specify tags and options in all uppercase and tags start with @.

An example `keys.conf` file is as follows:

```
#
# Maps an arbitrary tag [TAGNAME] with the string contents found in
# TARGET_BUILD_VARIANT. Common convention is to start TAGNAME with an @ and
# name it after the base file name of the pem file.
#
# Each tag (section) then allows one to specify any string found in
# TARGET_BUILD_VARIANT. Typically this is user, eng, and userdebug. Another
# option is to use ALL which will match ANY TARGET_BUILD_VARIANT string.
#

[@PLATFORM]
ALL : $DEFAULT_SYSTEM_DEV_CERTIFICATE/platform.x509.pem

[@MEDIA]
ALL : $DEFAULT_SYSTEM_DEV_CERTIFICATE/media.x509.pem

[@SHARED]
ALL : $DEFAULT_SYSTEM_DEV_CERTIFICATE/shared.x509.pem

# Example of ALL TARGET_BUILD_VARIANTS
[@RELEASE]
```

```
ENG      : $DEFAULT_SYSTEM_DEV_CERTIFICATE/testkey.x509.pem
USER     : $DEFAULT_SYSTEM_DEV_CERTIFICATE/testkey.x509.pem
USERDEBUG : $DEFAULT_SYSTEM_DEV_CERTIFICATE/testkey.x509.pem
```

The following is an example entry that will use a device specific key during the build process:

```
[@NET_APPS]
ALL : $ANDROID_BUILD_TOP/device/demo_vendor/se4a_device/security/net_apps.x509.pem
```

7.13.4 Build Bundle Tools

The following tools will produce an Android "bundle" for updating MAC/MMAC policy within a zip file suitable for installation by the SEAdmin app. SEAdmin is currently hard-coded to look for these zip files in the SD Card device (/sdcard/).

The [buildsebundle](#) section also shows how a policy can be updated by broadcasting an intent instead of using SEAdmin.

7.13.4.1 buildsebundle

The `buildsebundle` tool will produce an Android "bundle" for updating the core SE for Android policy within an `selinux_bundle.zip` file, suitable for installation by the SEAdmin app, although it is possible to update using an intent as described in the [Using an Intent Example](#) section.

To be able to build the bundle the following mandatory files are required:

```
selinux_version, sepolicy, file_contexts, seapp_contexts,
property_contexts, service_contexts, mac_permissions.xml
```

Usage:

```
usage: buildsebundle -k <private key.pk8> [-v <version>] [-r <previous hash>] \
[-h] -- <selinux_version> <file_contexts> <property_contexts> \
<sepolicy> <seapp_contexts> <service_contexts> <mac_permissions.xml>
```

This script builds a selinux policy bundle and supporting metadata file capable of being loaded via the ConfigUpdate mechanism. It takes a pkcs8 DER encoded RSA private key that is then used to sign the bundle. For AOSP development you'll typically want to use the key from the source tree at:

```
build/target/product/security/testkey.pk8
```

The built bundle will be written to `selinux_bundle.zip` which will include the signature metadata file of the bundle.

OPTIONS:

```
-h      Show this message.
-v      Version of the built bundle. Defaults to 1.
-r      SHA-512 hash of the bundle to replace. Defaults to 'NONE'.
```

The following is an example where a new policy has been built with all required files. The wildcard can be used as `buildsebundle` will always use the mandatory list:

```
buildsebundle -k $ANDROID_BUILD_TOP/build/target/product/security/testkey.pk8 \
-v 3 -- $ANDROID_BUILD_TOP/device/demo_device/se4a_device/new_sepolicy/*

adb push selinux_bundle.zip /sdcard/
```

Once built, the bundle is pushed to the SD card and SEAdmin is used to update the policy (note that SEAdmin only reads the bundle from /sdcard).

7.13.4.1.1 Using an Intent Example

This example shows how to update a policy by broadcasting an intent in the same way as SEAdmin.

Extract the `selinux_bundle` files from the `selinux_bundle.zip` file:

```
unzip selinux_bundle.zip
Archive:  selinux_bundle.zip
  inflating: update_bundle
  inflating: update_bundle_metadata
```

The two files contain:

update_bundle - Contains hex encoded policy files to be installed.

update_bundle_metadata - This is used by SEAdmin to form the intent and contains a hash of the bundle to replace or "NONE", the signature of the `update_bundle` and the bundle version (in this case "3"). Example contents are:

```
NONE:I6E0cZ8WbF6kJWkDozJCfckw5xuZhXuE0iqrbszszsxi7S4Z3DrR7RiH/aomRQxeskvMv9B/
+G7JXfxFAQlV1CWZihnefkHGnei4atKnBLPK/g3gmf0Wb0jjizc4yb4uvu/XQAZvybKcsTvTiegfqTHMFWPG
Kgoq97RKAjk2kT2fa3liArylTrLl7OfRtKq6mNjQNnfVrte9e/aJptiAmOwDNDQydfRwhewrKPE6rM+YNUHJ
aJ+h28dNecQtCn9TabTxn8I1G+10d5/wmjggXq6MdfEMQZ+
+H4ZIaL4bTduOQVdFeMsnFLA3hjLGf3BXpHmG84s7iD0158V0kbXikzA==:3
```

Push the `update_bundle` to the device:

```
adb push update_bundle /data/update_bundle
```

Build an intent to broadcast via `adb` by including the bundle location, with the hash, signature and version from the `update_bundle_metadata` as follows:

```
adb shell am broadcast -a android.intent.action.UPDATE_SEPOLICY -e "CONTENT_PATH"
"/data/update_bundle" -e "REQUIRED_HASH" "NONE" -e "SIGNATURE"
"I6E0cZ8WbF6kJWkDozJCfckw5xuZhXuE0iqrbszszsxi7S4Z3DrR7RiH/aomRQxeskvMv9B/
+G7JXfxFAQlV1CWZihnefkHGnei4atKnBLPK/g3gmf0Wb0jjizc4yb4uvu/XQAZvybKcsTvTiegfqTHMFWPGKgoq
97RKAjk2kT2fa3liArylTrLl7OfRtKq6mNjQNnfVrte9e/aJptiAmOwDNDQydfRwhewrKPE6rM+YNUHJaJ+h28dN
ecQtCn9TabTxn8I1G+10d5/wmjggXq6MdfEMQZ+
+H4ZIaL4bTduOQVdFeMsnFLA3hjLGf3BXpHmG84s7iD0158V0kbXikzA==" -e "VERSION" "3"
```

When the intent has been broadcast there will be a response, however that does not indicate that the policy was updated, just that the intent was broadcast:

```
Broadcasting: Intent { act=android.intent.action.UPDATE_SEPOLICY (has extras) }
Broadcast completed: result=0
```

`logcat` should show whether it was successful:

```
I/ConfigUpdateInstallReceiver( 908): Found new update, installing...
I/ConfigUpdateInstallReceiver( 908): Installation successful
I/SELinuxPolicyInstallReceiver( 908): Applying SELinux policy
```

If the update failed because of versioning then an error is given (however if signature incorrect fails silently).

The following show various policy information after the third update:

```
adb shell ls -l /data/security/current
lrwxrwxrwx system system 2014-07-19 10:41 current -> /data/security/contexts
```

```
adb shell ls -l /data/security/contexts
-rw-r--r-- system system 10512 2014-07-19 10:41 file_contexts
-rw-r--r-- system system 10656 2014-07-19 09:01 file_contexts_backup
-rw-r--r-- system system 4203 2014-07-19 10:41 mac_permissions.xml
-rw-r--r-- system system 4203 2014-07-19 09:01 mac_permissions.xml_backup
-rw-r--r-- system system 2549 2014-07-19 10:41 property_contexts
-rw-r--r-- system system 2549 2014-07-19 09:01 property_contexts_backup
-rw-r--r-- system system 641 2014-07-19 10:41 seapp_contexts
-rw-r--r-- system system 641 2014-07-19 09:01 seapp_contexts_backup
-rw-r--r-- system system 78 2014-07-19 10:41 selinux_version
-rw-r--r-- system system 78 2014-07-19 09:01 selinux_version_backup
-rw-r--r-- system system 115831 2014-07-19 10:41 sepolicy
-rw-r--r-- system system 116438 2014-07-19 09:01 sepolicy_backup
-rw-r--r-- system system 7748 2014-07-19 10:41 service_contexts
-rw-r--r-- system system 7748 2014-07-19 09:01 service_contexts_backup
```

```
adb shell ls -l /data/security
drwx----- system system 2014-07-19 10:41 bundle
drwx----- system system 2014-07-19 10:41 contexts
lrwxrwxrwx system system 2014-07-19 10:41 current ->/data/security/contexts
drwx----- system system 2014-07-19 10:37 eops
```

```
adb shell ls -l /data/security/bundle
drwx----- system system 2014-07-19 10:41 metadata
-rw-r--r-- system system 191271 2014-07-19 10:41 sepolicy_bundle
```

```
adb shell ls -l /data/security/bundle/metadata
-rw-r--r-- system system 1 2014-07-19 10:41 version
```

```
adb shell cat /data/security/bundle/metadata/version
3
```

The loaded policy can be extracted from the device if required by:

```
adb pull /sys/fs/selinux/policy sepolicy-v3
```

7.13.4.2 buildeopbundle

The `buildeopbundle` tool will produce an Android "bundle" for updating the Enterprise Operations policy within an `eops_bundle.zip` file suitable for installation by the SEAdmin app, although it is possible to update using an intent as described in the [Using an Intent Example](#) section.

To be able to build the bundle an `eops.xml` file is required.

Usage:

```
usage: buildeopbundle -k <private key.pk8> [-v <version>] [-r <previous hash>] \
[-h] -- <eops.xml>
```

```
This script builds a eops policy bundle and supporting metadata file capable of being loaded via the ConfigUpdate mechanism. It takes a pkcs8 DER encoded RSA private key that is then used to sign the bundle. For AOSP development you'll typically want to use the key from the source tree at:
```

```
build/target/product/security/testkey.pk8
```

```
If building your own cert you should probably use a key size of at least 1024 or greater. The bundle requires that the eops.xml file be included and with that exact basename. The built bundle will be written to eop_bundle.zip which will include the signature metadata file of the bundle.
```

```
OPTIONS:
```

```
-h Show this message.
-v Version of the built bundle. Defaults to 1.
-r SHA-512 hash of the bundle to replace. Defaults to 'NONE'.
```

7.13.4.2.1 Eops Example

The following is an example where a new `eops.xml` file has been produced, bundled, then pushed to the SD card. SEAdmin is then used to update the policy (note that SEAdmin only reads the bundle from `/sdcard`):

```
buildeopbundle -k $ANDROID_BUILD_TOP/build/target/product/security/testkey.pk8
-v 1 -- eops.xml

adb push eops_bundle.zip /sdcard/
```

logcat should show if it was successful:

```
D/SEAdminConfigUpdateFragment( 904): android.intent.action.UPDATE_EOPS intent being broadcast. Bundle[{CONTENT_PATH=/cache/eops_bundle, SIGNATURE=qZJ8I07MHFTXaII2jhPMoorLzejArUI0qsvkteG9nzEzgzjwyh8RWUaaRil6xrQsPb5g+qWj+nfQCkH7DI Eow/WF8S1sTeReS8G/z+hPQi0MHgWGKH0kCIfxN6yqqEri3+DnolblvHVuM7t/0mszCvtjqfq5GwbHZc1xYSgMQJXqrfzSqa2zvO4+7zEOGszfuZXwt9QHci9C1IJ5B50URmmg4TDIuhfISWW9vYkEctwARIyCLhfYiZzIQOwzPj3oSII1AUWMHxbbpADFzCumZ1WdfpA0txow8rDM+01qkKGtcAsNs8me2FAPz28tckQ9ea6QwAzDCSP3PzQC1Horg==, REQUIRED_HASH=NONE, VERSION=1}]
I/ConfigUpdateInstallReceiver( 395): Couldn't find current metadata, assuming first update
I/ConfigUpdateInstallReceiver( 395): Failed to read current content, assuming first update!
I/ConfigUpdateInstallReceiver( 395): Found new update, installing...
I/ConfigUpdateInstallReceiver( 395): Installation successful
D/AppOps ( 381): Eops policy: system [ CAMERA]
D/AppOps ( 381): Eops policy: default [ CAMERA]
```

The new file and its supporting metadata are:

```
adb shell su 0 ls -lR /data/security/eops

/data/security/eops:
-rw-r--r-- system system 189 2014-07-20 14:15 eops.xml
drwx----- system system 2014-07-20 14:15 eops_metadata

/data/security/eops/eops_metadata:
-rw-r--r-- system system 1 2014-07-20 14:15 version
```

The version number after the update is:

```
adb shell su 0 cat /data/security/eops/eops_metadata/version
1
```

Because the Eops policy specified an `seinfo` of `system` and the operation `CAMERA`, if the Camera app is now started it will load however, it will not be possible to take pictures as logcat will show:

```
D/AppOps ( 381): startOperation: reject #1 for code 26 (26) uid 10026 package
com.android.camera2
I/CameraService( 60): Camera 0: Access for "com.android.camera2" has been revoked
```

7.13.4.3 buildifwbundle

The buildifwbundle tool will produce an Android "bundle" for updating the Intent Firewall policy within an ifw_bundle.zip file suitable for installation by the SEAdmin app, although it is possible to update using an intent as described in the [Using an Intent Example](#) section.

To be able to build the bundle an ifw.xml file is required, although note that the Intent Firewall service will read any file so long as it has the .xml extension.

Usage:

```
usage: buildifwbundle -k <private key.pk8> [-v <version>] [-r <previous hash>] \
[-h] -- <ifw.xml>

This script builds an intent firewall policy bundle and supporting metadata file
capable of being loaded via the ConfigUpdate mechanism. It takes a pkcs8 DER
encoded RSA private key that is then used to sign the bundle. For AOSP
development you'll typically want to use the key from the source tree at:
  build/target/product/security/testkey.pk8
If building your own cert you should probably use a key size of at least 1024 or
greater. The bundle requires that the ifw.xml file be included and with that
exact basename. The built bundle will be written to ifw_bundle.zip which will
include the signature metadata file of the bundle.

OPTIONS:
  -h      Show this message.
  -v      Version of the built bundle. Defaults to 1.
  -r      SHA-512 hash of the bundle to replace. Defaults to 'NONE'.
```

7.13.4.3.1 IFW Example

The following is an example where a new ifw.xml file has been produced, bundled, and then pushed to the SD card. SEAdmin is then used to update the policy (note that SEAdmin only reads the bundle from /sdcard):

```
buildsifwbundle -k $ANDROID_BUILD_TOP/build/target/product/security/testkey.pk8
-v 1 -- eops.xml

adb push ifw_bundle.zip /sdcard/
```

logcat should show whether it was successful:

```
D/SEAdminConfigUpdateFragment( 904): android.intent.action.UPDATE_INTENT_FIREWALL intent
being broadcast. Bundle[{CONTENT_PATH=/cache/ifw_bundle,
SIGNATURE=tfQONpeZbL1Y6sXj1BY98T04izK2IyeqO9Hko5tZygE77zry98RGmU5BAAIFs21G9G7WpAcPTR7TGe4LR
MpB7SKeZ1Xh+4B+U+30TnHkwXp9HRigIJcN5Kqiyp/UPAjEJjYmBzk+yM5FLYcMCQS082wfpC9c+gRQc16AYuSmiiyv
jgc1d33rtfB7Hd40LF30mBZyyiUJc5YFlddaITBbL/CCkMfblfBqadZtmCN7xGUIJEHQWPnuEvsckatKOLgZa+35ZXfl
2WkD/DsGkwocXM9akjDONJY9WZJpzwAHQPdQFXN6nthrsV8kiC7OUFvK/PK119oetiYtSEEVH5JlMnA==,
REQUIRED_HASH=NONE, VERSION=1}]
I/ConfigUpdateInstallReceiver( 395): Couldn't find current metadata, assuming first update
I/ConfigUpdateInstallReceiver( 395): Failed to read current content, assuming first
update!
I/ConfigUpdateInstallReceiver( 395): Found new update, installing...
I/ConfigUpdateInstallReceiver( 395): Installation successful
I/IntentFirewall( 395): Read new rules (A:0 B:0 S:1)
```

The new file and its supporting metadata are:

```
adb shell su 0 ls -lR /data/system/ifw

/data/system/ifw:
-rw-r--r-- system system 454 2014-07-20 13:14 ifw.xml
drwx----- system system 2014-07-20 13:14 metadata

/data/system/ifw/metadata:
-rw-r--r-- system system 1 2014-07-20 13:14 gservices.version
```

The version number after the update is:

```
adb shell su 0 cat /data/system/ifw/metadata/gservices.version
1
```

7.13.5 post_process_mac_perms

This tool will modify an existing `mac_permissions.xml` with additional app certs not already found in that policy. This becomes useful when a directory containing apps is searched and the certs from those apps are added to the policy not already explicitly listed.

There is no make target for this tool (python script), so either move to `HOST_EXECUTABLE` or execute directly (e.g. `$PREFIX/external/sepolicy/tools/post_process_mac_perms`).

Usage:

```
post_process_mac_perms [-h] -s SEINFO -d DIR -f POLICY

-s SEINFO, --seinfo SEINFO seinfo tag for each generated stanza
-d DIR, --dir DIR Directory to search for apks
-f POLICY, --file POLICY mac_permissions.xml policy file
```

Example:

```
post_process_mac_perms -s netapps -d ./APK -f mac_permissions.xml
```

Before:

```
<?xml version="1.0" encoding="utf-8"?>
<policy>
  <signer signature="- certificate here -" ><seinfo value="platform"/></signer>
  <default><seinfo value="default"/></default>
</policy>
```

After:

```
<?xml version="1.0" encoding="utf-8"?>
<policy>
  <signer signature="- certificate here -" ><seinfo value="platform"/></signer>
  <default><seinfo value="default"/></default>
  <signer signature="- certificate here -" ><seinfo value="netapps"/></signer>
</policy>
```

7.13.6 sepolicy_check

A tool for auditing a sepolicy file for any allow rule that grants a given permission.

Usage:

```
sepolicy-check -s <domain> -t <type> -c <class> -p <permission>
-P out/target/product/<board>/root/sepolicy
```

The output will be "Match found!" or silent if not. sepolicy_check will return 0 for found, 1 for not found and -1 for an error.

Examples:

```
sepolicy-check -s healthd -t system_server_service \
-c service_manager -p add \
-P out/target/product/generic/root/sepolicy
Match found!
```

```
sepolicy-check -s su -t security_prop -c property_service \
-p set -P out/target/product/generic/root/sepolicy
echo $?
1
```

7.13.7 sepolicy-analyze

This is the text from the external/sepolicy/tools/README that describes the tool for performing various kinds of analysis on a sepolicy file. The analysis currently supported include:

7.13.7.1 Type Equivalence

```
sepolicy-analyze -e -P out/target/product/<board>/root/sepolicy
```

Display all type pairs that are "equivalent", i.e. they are identical with respect to allow rules, including indirect allow rules via attributes and default-enabled conditional rules (i.e. default boolean values yield a true conditional expression).

Equivalent types are candidates for being coalesced into a single type. However, there may be legitimate reasons for them to remain separate, for example: - the types may differ in a respect not included in the current analysis, such as default-disabled conditional rules, audit-related rules (auditallow or dontaudit), default type transitions, or constraints (e.g. mls), or - the current policy may be overly permissive with respect to one or the other of the types and thus the correct action may be to tighten access to one or the other rather than coalescing them together, or - the domains that would in fact have different accesses to the types may not yet be defined or may be unconfined in the policy you are analyzing.

Example output:

```
sepolicy-analyze -e -P out/target/product/se4a_device/root/sepolicy
```

```
Types abbd_socket and mdns_socket are equivalent.
Types rild_debug_socket and init_tmpfs are equivalent.
Types rild_debug_socket and qemud_tmpfs are equivalent.
Types surfaceflinger_service and mediaserver_service are equivalent.
Types surfaceflinger_service and inputflinger_service are equivalent.
```

7.13.7.2 Type Difference

```
sepolicy-analyze -d -P out/target/product/<board>/root/sepolicy
```

Display type pairs that differ and the first difference found between the two types. This may be used in looking for similar types that are not equivalent but may be candidates for coalescing.

Example output:

```
sepolicy-analyze -d -P out/target/product/se4a_device/root/sepolicy

Types abbd_socket and functionfs differ, starting with:
  allow abbd_socket rootfs:filesystem { associate };
  allow functionfs self:filesystem { associate };

Types abbd_socket and hci_attach_exec differ, starting with:
  allow system_server abbd_socket:sock_file { ioctl read write getattr lock append
open };
  allow debuggerd hci_attach_exec:file { ioctl read getattr lock open };

Types abbd_socket and system_server differ, starting with:
  allow abbd_socket rootfs:filesystem { associate };
  allow system_server rootfs:filesystem { getattr };
```

7.13.7.3 Duplicate Allow Rules

```
sepolicy-analyze -D -P out/target/product/<board>/root/sepolicy
```

Displays duplicate allow rules, i.e. pairs of allow rules that grant the same permissions where one allow rule is written directly in terms of individual types and the other is written in terms of attributes associated with those same types. The rule with individual types is a candidate for removal. The rule with individual types may be directly represented in the source policy or may be a result of expansion of a type negation (e.g. `domain -foo -bar` is expanded to individual allow rules by the policy compiler). Domains with `unconfineddomain` will typically have such duplicate rules as a natural side effect and can be ignored.

Example output:

```
sepolicy-analyze -D -P out/target/product/se4a_device/root/sepolicy

Duplicate allow rule found:
  allow init hci_attach_exec:file { read getattr execute open };
  allow unconfineddomain exec_type:file { ioctl read getattr lock execute open };

Duplicate allow rule found:
  allow ueventd device:dir { write add_name remove_name };
  allow ueventd dev_type:dir { ioctl read write create getattr setattr unlink link rename
add_name remove_name reparent search rmdir open };
```

7.13.8 setool

The setool utility is not used during the build process and is intended only to produce entries for the `mac_permissions.xml` file and verify a correctly formatted file. It is not supplied in AOSP.

Usage:

```
Usage: setool [flags] <--build keys|package OR --policy policyFile> <apk> [ <apk> ]*

Tool to help build and verify MMAC install policies.

--build  Generate an MMAC style policy stanza with a given --seinfo string.
         The resulting stanza can then be used as an entry in the mac_permissions.xml
         file.
  package Policy entry that contains the package name inside the signature stanza.
  keys     Print just a signer tag which contains the hex encoded X.509 certs of the app.

--policy Determine if the apks pass the supplied policy by printing the seinfo tag
         that would be assigned or null otherwise.

apk      An apk to analyze. All supplied apks must be absolute paths or relative to
         --apkdir (which defaults to the current directory).

Flags:
--help   Prints this message and exits.
--apkdir Directory to search for supplied apks (default to current directory).
--verbose Increase the amount of debug statements.
--outfile Dump output to the given file (defaults to stdout).
--seinfo Create an seinfo tag for all generated policy stanzas. This is a required
         flag if using the --build option.
```

The following examples show the generation and verification process:

```
setool --build package --seinfo service_app \
--outfile sepolicy/mac_permissions.xml \
RunIsolatedService.apk
```

The output will be:

```
<signer signature="- certificate will be here -">
  <package name="com.example.runisolatedservice">
    <seinfo value="service_app" />
  </package>
</signer>
```

Note that for verification via setool requires the segment to be included within a correctly formatted `mac_permissions.xml` file (i.e. have the `<policy>` `</policy>` tags present:

```
setool --policy sepolicy/mac_permissions.xml RunIsolatedService.apk
```

The output will then be:

```
seinfo tag service_app assigned to ./RunIsolatedService.apk
```

7.14 selinux-network.sh Configuration

This file may become obsolete, however to enable and configure it for loading **iptables** (8) with SECMARK information as part of the policy build, the following will need to be carried out.

Add an entry in the device make file:

```
PRODUCT_PACKAGES += selinux-network.sh
```

Replace the default version in `external/sepolicy` by an entry in the `BoardConfig.mk` file (assumes there is a `BOARD_SEPOLICY_DIRS` entry):

```
BOARD_SEPOLICY_REPLACE += selinux-network.sh
```

Then either load via `adb` or add to the `init.rc` file:

```
## Daemon process to be run by init.
##
...
# Load iptables configuration.
service netlabels /system/bin/selinux-network.sh
    class core
    oneshot
```

During the build the file will be installed at `/system/bin/selinux-network.sh` and may be executed at system initialisation time or via `adb`.

Example `selinux-network.sh` entries:

```
#!/system/bin/sh

##### IPTABLES FOR V4 Using security table #####
IPTABLES="/system/bin/iptables"

# Common rules that copy connection labels to established and related packets:
$IPTABLES -t security -A INPUT -m state --state ESTABLISHED,RELATED -j CONNSECMARK --restore
$IPTABLES -t security -A OUTPUT -m state --state ESTABLISHED,RELATED -j CONNSECMARK --restore

# Create a chain for the NetLabelDemo app:
$IPTABLES -t security -N SELINUX_NET_APPS
# Add rules to mark the demo packets:
$IPTABLES -t security -A SELINUX_NET_APPS -j SECMARK --selctx u:object_r:net_apps_packet:s0
$IPTABLES -t security -A SELINUX_NET_APPS -j CONNSECMARK --save
$IPTABLES -t security -A SELINUX_NET_APPS -j ACCEPT
$IPTABLES -t security -A OUTPUT -p tcp --dport 9999 -j SELINUX_NET_APPS
$IPTABLES -t security -A INPUT -p tcp --sport 9999 -j SELINUX_NET_APPS
```

Notes:

1. Adding entries to this file will also require additional policy rules to be added for the device.
2. Kernels supplied as part of AOSP or SEAndroid may not have the kernel build parameters to support all the SECMARK features. The following additional kernel parameters will enable these:

a) Enable iptables 'security' table in kernel (although the mangle table may be used instead):

```
CONFIG_IP_NF_SECURITY=y
CONFIG_IP6_NF_SECURITY=y
```

b) Enable SECMARK/CONNSECMARK in kernel:

```
CONFIG_NETWORK_SECMARK=y
CONFIG_NF_CONNTRACK_SECMARK=y
CONFIG_NETFILTER_XT_TARGET_CONNSECMARK=y
CONFIG_NETFILTER_XT_TARGET_SECMARK=y
```

7.15 uid To username Utility

This utility will take an Android uid and convert it to a username. The code is a modified version from `bionic/libc/bionic/stubbs.cpp` that converts an Android uid to username.

To compile this utility:

```
cc -std=gnu99 uid_to_username.c -o uid_to_username -include \
$ANDROID_BUILD_TOP/system/core/include/private/android_filesystem_config.h
```

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
    uid_t uid;
    if (argc != 2) {
        printf("Converts an Android uid to username\n");
        printf("usage: %s uid\n\n", argv[0]);
        exit(1);
    }
    uid = atoi(argv[1]);
    uid_t appid = uid % AID_USER;
    uid_t userid = uid / AID_USER;

    if (appid >= AID_ISOLATED_START) {
        printf("username: u%u_i%u\n", userid, appid - AID_ISOLATED_START);
    } else if (userid == 0 && appid >= AID_SHARED_GID_START) {
        printf("username: all_a%u\n", appid - AID_SHARED_GID_START);
    } else if (appid < AID_APP) {
        for (size_t n = 0; n < android_id_count; n++) {
            if (android_ids[n].aid == appid) {
                printf("username: u%u_%s\n", userid, android_ids[n].name);
                printf("Note that only \"%s\" will be shown in 'ps' etc.\n",
                    android_ids[n].name);
                exit(0);
            }
        }
    }
}
```

```
    }
    printf("Failed - invalid uid\n");
} else {
    printf("username: u%u_a%u\n", userid, appid - AID_APP);
}
exit(0);
}
```

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-20 policy sources. There are also additional entries for Xen. The SEAndroid specific classes and permissions are shown in the [Security Enhancements for Android](#) section.

All objects are kernel objects unless marked as user space objects.

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'). Some 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 location and would probably be more up-to-date:

<http://selinuxproject.org/page/ObjectClassesPerms>

8.2 Defining Object Classes and Permissions

The Reference Policy already contains 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](#).

8.3 Common Permissions

8.3.1 Common File Permissions

[Table 31](#) 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 31: Common File Permissions

8.3.2 Common Socket Permissions

[Table 32](#) 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 32: Common Socket Permissions

8.3.3 Common IPC Permissions

[Table 33](#) 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 33: Common IPC Permissions

8.3.4 Common Database Permissions

[Table 34](#) describes the common database permissions that are inherited by a number of object classes. The "[Security-Enhanced PostgreSQL Security Wiki](#)" [2] 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 (DROP) a database object.

getattr	Get metadata - needed to reference an object (e.g. SELECT ... FROM ...).
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 (e.g. ALTER ...).

Table 34: Common PostgreSQL Database Permissions

8.3.5 Common X_Device Permissions

[Table 35](#) describes the common `x_device` permissions that are inherited by the X-
Windows `x_keyboard` and `x_pointer` object classes.

Permissions	Description (19 permissions)
add	
bell	
create	
destroy	
force_cursor	Get window focus.
freeze	
get_property	Required to create a device context. (source code)
getattr	
getfocus	
grab	Set window focus.
list_property	
manage	
read	
remove	
set_property	
setattr	
setfocus	
use	
write	

Table 35: Common X_Device Permissions

8.4 File Object Classes

Class	filesystem - A mounted filesystem
Permissions	Description (10 unique permissions)
associate	Use <code>type</code> 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.

The SELinux Notebook

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.

Class	dir - Directory
Permissions	Description (Inherit 17 common file permissions + 7 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.
audit_access	The rules for this permission work as follows: If a process calls access() or faccessat() and SELinux denies their request there will be a check for a dontaudit rule on the audit_access permission. If there is a dontaudit rule on audit_access an AVC event will not be written. If there is no dontaudit rule an AVC event will be written for the permissions requested (read, write, or exec). Notes: 1) There will never be a denial message with the audit_access permission as this permission does not control security decisions. 2) allow and auditallow rules with this permission are therefore meaningless, however the kernel will accept a policy with such rules, but they will do nothing.
execmod	Make executable a file that has been modified by copy-on-write.
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 + 5 unique)
Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
audit_access	See the dir class for details
entrypoint	Entry point permission for a domain transition.
execute_no_trans	Execute in the caller's domain (i.e. no domain transition).
execmod	Make executable a file that has been modified by copy-on-write.
open	Added in 2.6.26 Kernel to control the open permission.

Class	lnk_file - Symbolic links
Permissions	Description (Inherit 17 common file permissions + 3 unique)
Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
audit_access	See the dir class for details

The SELinux Notebook

execmod	Make executable a file that has been modified by copy-on-write.
open	Added in 2.6.26 Kernel to control the open permission.

Class	chr_file - Character files
Permissions	Description (Inherit 17 common file permissions + 5 unique)
Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
audit_access	See the dir class for details
entrypoint	Entry point permission for a domain transition.
execute_no_trans	Execute in the caller's domain (i.e. no domain transition).
execmod	Make executable a file that has been modified by copy-on-write.
open	Added in 2.6.26 Kernel to open a character device.

Class	blk_file - Block files
Permissions	Description (Inherit 17 common file permissions + 3 unique)
Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
audit_access	See the dir class for details
execmod	Make executable a file that has been modified by copy-on-write.
open	Added in 2.6.26 Kernel to control the open permission.

Class	sock_file - UNIX domain sockets
Permissions	Description (Inherit 17 common file permissions + 3 unique)
Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
audit_access	See the dir class for details
execmod	Make executable a file that has been modified by copy-on-write.
open	Added in 2.6.26 Kernel to control the open permission.

Class	fifo_file - Named pipes
Permissions	Description (Inherit 17 common file permissions + 3 unique)
Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounon, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
audit_access	See the dir class for details
execmod	Make executable a file that has been modified by copy-on-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 <code>ingress</code> permission.
sendto	Network interface and address check permission for use with the <code>egress</code> 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 <code>egress</code> access control. Also requires the <code>node sendto</code> permission.
ingress	Each packet entering the system must pass an <code>ingress</code> access control. Also requires the <code>node 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.

The SELinux Notebook

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	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	tcp_socket - Protocol: PF_INET, PF_INET6 Family Type: SOCK_STREAM
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
acceptfrom	Accept connection from client socket.
connectto	Connect to server socket.
name_connect	Connect to a specific port type.
newconn	Create new connection.
node_bind	Bind to a node.

Class	udp_socket - Protocol: PF_INET, PF_INET6 Family Type: SOCK_DGRAM
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	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,

The SELinux Notebook

	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

Class	tun_socket - TUN is Virtual Point-to-Point network device driver to support IP tunneling.
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
attach_queue	

8.5.1 IPsec Network Object Classes

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.
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	accept, append, bind, connect, create, getattr,

The SELinux Notebook

Socket Permissions	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, setattr, 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, setattr, 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, setattr, 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.

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, setattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

The SELinux Notebook

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, rcv_msg, rcvfrom, 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, rcv_msg, rcvfrom, 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, rcv_msg, rcvfrom, 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, rcv_msg, rcvfrom, 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.
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	accept, append, bind, connect, create, getattr,

Socket Permissions	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, setattr, 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, setattr, 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.
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	accept, append, bind, connect, create, setattr, getopt, ioctl, listen, lock, name_bind, read,

Permissions	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 - Interprocess communications
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
--------------	--

The SELinux Notebook

	system.
Permissions	Description (31 unique permissions)
dyntransition	Dynamically transition to a new context using setcon (3) .
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 (ptrace (2)).
ptrace_child	Trace program execution of 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 (3) .
setfscreate	Set security context by setfscreatecon (3) .
setkeycreate	Set security context by setkeycreatecon (3) .
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 (3) .
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 (12 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 and class.
compute_create	Determine context to use when querying the security server about a

	transition rule (<code>type_transition</code>).
<code>compute_member</code>	Determine context to use when querying the security server about a membership decision (<code>type_member</code> for a polyinstantiated object).
<code>compute_relabel</code>	Determines the context to use when querying the security server about a relabeling decision (<code>type_change</code>).
<code>compute_user</code>	Determines the context to use when querying the security server about a user decision (<code>user</code>).
<code>load_policy</code>	Load the security policy into the kernel (the security server).
<code>read_policy</code>	Read the kernel policy to userspace.
<code>setbool</code>	Change a boolean value within the active policy.
<code>setcheckreqprot</code>	Set if SELinux will check original protection mode or modified protection mode (read-implies-exec) for <code>mmap</code> / <code>mprotect</code> .
<code>setenforce</code>	Change the enforcement state of SELinux (permissive or enforcing).
<code>setseccparam</code>	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 (12 unique permissions)
<code>disable</code>	Allow services to be disabled.
<code>enable</code>	Allow services to be enabled.
<code>halt</code>	Allow the system to be halted.
<code>ipc_info</code>	Get info about an IPC object.
<code>module_request</code>	Request the kernel to load a module.
<code>reboot</code>	Allow system to be rebooted.
<code>reload</code>	Allow services to be reloaded.
<code>status</code>	Get system status information.
<code>syslog_console</code>	Control output of kernel messages to the console with <code>syslog(2)</code> .
<code>syslog_mod</code>	Clear kernel message buffer with <code>syslog(2)</code> .
<code>syslog_read</code>	Read kernel message with <code>syslog(2)</code> .
<code>undefined</code>	Allow an undefined operation.

8.10 Kernel Service Object Class

Class	kernel_service - Used to add kernel services.
Permissions	Description (2 unique permissions)
<code>use_as_override</code>	Grant a process the right to nominate an alternate process SID for the kernel to use as an override for the SELinux subjective security when accessing information on behalf of another process. For example, CacheFiles when accessing the cache on behalf of a process accessing an NFS file needs to use a subjective security ID appropriate to the cache rather than the one the calling process is using. The <code>cachefilesd</code> daemon will nominate the security ID to be used.
<code>create_files_as</code>	Grant a process the right to nominate a file creation label for a kernel service to use.

8.11 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 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; nfservctl; VM86_REQUEST_IRQ; to read/write pci config on alpha; irix_prctl 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 <code>sysctl</code> .
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 ioperm (2) and iopl (2) 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
Permissions	Description (7 unique permissions)
block_suspend	Prevent system suspends (was <code>epollwakeup</code>)
compromise_kernel	Allow tasks that can modify the running kernel (Secure Boot).
mac_admin	Allow MAC configuration state changes. For SELinux allow contexts not defined in the policy to be assigned. This is called 'deferred mapping of security contexts' and is explained at: http://www.nsa.gov/research/selinux/list-archive/0805/26046.shtml
mac_override	Allow MAC policy to be overridden.
syslog	Allow configuration of kernel <code>syslog</code> (<code>printk</code> behaviour).
wake_alarm	Trigger the system to wake up

8.12 X Windows Object Classes

These are userspace objects managed by XSELinux.

Class	x_drawable - The <code>drawable</code> parameter specifies the area into which the text will be drawn. It may be either a pixmap or a window. Some of the permission information has been extracted from an email describing them in terms of an MLS system.
--------------	--

The SELinux Notebook

Permissions	Description (19 unique permissions)
add_child	Add new window. Normally SystemLow for MLS systems.
blend	There are two cases: 1) Allow a non-root window to have a transparent background. 2) The application is redirecting the contents of the window and its sub-windows into a memory buffer when using the Composite extension. Only SystemHigh processes should have the blend permission on the root window.
create	Create a drawable object. Not applicable to the root windows as it cannot be created.
destroy	Destroy a drawable object. Not applicable to the root windows as it cannot be destroyed.
get_property	Read property information. Normally SystemLow for MLS systems.
getattr	Get attributes from a drawable object. Most applications will need this so SystemLow.
hide	Hide a drawable object. Not applicable to the root windows as it cannot be hidden.
list_child	Allows all child window IDs to be returned. From the root window it will show the client that owns the window and their stacking order. If hiding this information is required then processes should be SystemHigh.
list_property	List property associated with a window. Normally SystemLow for MLS systems.
manage	Required to create a context, move and resize windows. Not applicable to the root windows as it cannot be resized etc.
override	Allow setting the <code>override-redirect</code> bit on the window. Not applicable to the root windows as it cannot be overridden.
read	Read window contents. Note that this will also give read permission to all child windows, therefore (for MLS), only SystemHigh processes should have read permission on the root window.
receive	Allow receiving of events. Normally SystemLow for MLS systems (but could leak information between clients running at different levels, therefore needs investigation).
remove_child	Remove child window. Normally SystemLow for MLS systems.
send	Allow sending of events. Normally SystemLow for MLS systems (but could leak information between clients running at different levels, therefore needs investigation).
set_property	Set property. Normally SystemLow for MLS systems (but could leak information between clients running at different levels, therefore needs investigation. Polyinstantiation may be required).
setattr	Allow window attributes to be set. This permission protects operations on the root window such as setting the background image or colour, setting the colormap and setting the mouse cursor to display when the cursor is in the window, therefore only SystemHigh processes should have the <code>setattr</code> permission.
show	Show window. Not applicable to the root windows as it cannot be hidden.
write	Draw within a window. Note that this will also give write permission to all child windows, therefore (for MLS), only SystemHigh processes should have <code>write</code> permission on the root window.

The SELinux Notebook

Class	x_screen - The specific screen available to the display (X-server) (hostname:display_number.screen)
Permissions	Description (8 unique permissions)
getattr	
hide_cursor	
saver_getattr	
saver_hide	
saver_setattr	
saver_show	
setattr	
show_cursor	

Class	x_gc - The graphics contexts allows the X-server to cache information about how graphics requests should be interpreted. It reduces the network traffic.
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	Allow GC contexts to be used.

Class	x_font - An X-server resource for managing the different fonts.
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.

Class	x_colormap - An X-server resource for managing colour mapping. A new colormap can be created using XCreateColormap.
Permissions	Description (10 unique permissions)
add_color	Add a colour
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	Remove a colour
uninstall	Remove a virtual colormap from the display hardware.
use	Use a colormap

The SELinux Notebook

write	Change color cells in colormap.
-------	---------------------------------

Class	x_property - An InterClient Communications (ICC) service where each property has a name and ID (or Atom). Properties are attached to windows and can be uniquely identified by the windowID and propertyID. XSELinux supports polyinstantiation of properties.
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 - An InterClient Communications (ICC) service that allows two parties to communicate about passing information. The information uses properties to define the the format (e.g. whether text or graphics). XSELinux supports polyinstantiation of selections.
Permissions	Description (4 unique permissions)
getattr	Get selection owner (XGetSelectionOwner).
read	Read the information from the selection owner
setattr	Set the selection owner (XSetSelectionOwner).
write	Send the information to the selection requestor.

Class	x_cursor - The cursor on the screen
Permissions	Description (7 unique permissions)
create	Create an arbitrary cursor object.
destroy	Free (dereference) a cursor object.
getattr	Get attributes of the cursor.
read	Read the cursor.
setattr	Set attributes of the cursor.
use	Associate a cursor object with a window.
write	Write a cursor

Class	x_client - The X-client connecting to the X-server.
Permissions	Description (4 unique permissions)
destroy	Close down a client.
getattr	Get attributes of X-client.
manage	Required to create an X-client context. (source code)
setattr	Set attributes of X-client.

Class	x_device - These are any other devices used by the X-server as the keyboard and pointer devices have their own object classes.
--------------	---

The SELinux Notebook

Permissions	Description (Inherit 19 common <code>x_device</code> permissions)
Inherit Common X_Device Permissions	<code>add, bell, create, destroy, force_cursor, freeze, get_property, getattr, getfocus, grab, list_property, manage, read, remove, set_property, setattr, setfocus, use, write</code>

Class	x_server - The X-server that manages the display, keyboard and pointer.
Permissions	Description (6 unique permissions)
<code>debug</code>	
<code>getattr</code>	
<code>grab</code>	
<code>manage</code>	Required to create a context. (source code)
<code>record</code>	
<code>setattr</code>	

Class	x_extension - An X-Windows extension that can be added to the X-server (such as the XSELinux object manager itself).
Permissions	Description (2 unique permissions)
<code>query</code>	Query for an extension.
<code>use</code>	Use the extensions services.

Class	x_resource - These consist of Windows, Pixmaps, Fonts, Colormaps etc. that are classed as resources.
Permissions	Description (2 unique permissions)
<code>read</code>	Allow reading a resource.
<code>write</code>	Allow writing to a resource.

Class	x_event - Manage X-server events.
Permissions	Description (2 unique permissions)
<code>receive</code>	Receive an event
<code>send</code>	Send an event

Class	x_synthetic_event - Manage some X-server events (e.g. <code>confignotify</code>). Note the <code>x_event</code> permissions will still be required (its magic).
Permissions	Description (2 unique permissions)
<code>receive</code>	Receive an event
<code>send</code>	Send an event

Class	x_application_data - Not specifically used by XSELinux, however is used by userspace applications that need to manage copy and paste services (such as the <code>CUT_BUFFERS</code>).
Permission	Description (3 unique permissions)
<code>copy</code>	Copy the data

paste	Paste the data
paste_after_confirm	Need to confirm that the paste is allowed.

Class	x_pointer - The mouse or other pointing device managed by the X-server.
Permissions	Description (Inherit 19 common x_device permissions)
Inherit Common X_Device Permissions	add, bell, create, destroy, force_cursor, freeze, get_property, getattr, getfocus, grab, list_property, manage, read, remove, set_property, setattr, setfocus, use, write

Class	x_keyboard - The keyboard managed by the X-server.
Permissions	Description (Inherit 19 common x_device permissions)
Inherit Common X_Device Permissions	add, bell, create, destroy, force_cursor, freeze, get_property, getattr, getfocus, grab, list_property, manage, read, remove, set_property, setattr, setfocus, use, write

8.13 Database Object Classes

These are userspace objects - The PostgreSQL database supports these with their SE-PostgreSQL database extension. The "[Security-Enhanced PostgreSQL Security Wiki](#)" [2] explains the objects, their permissions and how they should be used in detail.

Class	db_database
Permission	Description (Inherit 6 common database permissions + 3 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 SE-PostgreSQL client.
install_module	Required to install a dynamic link library.
load_module	Required to load a dynamic link library.

Class	db_table
Permission	Description (Inherit 6 common database permissions + 5 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.

The SELinux Notebook

Class	db_schema
Permission	Description (Inherit 6 common database permissions + 3 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
search	Search for an object in the schema.
add_name	Add an object to the schema.
remove_name	Remove an object from the schema.

Class	db_procedure
Permission	Description (Inherit 6 common database permissions + 3 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
entrypoint	Required for any functions defined as Trusted Procedures.
execute	Required for functions executed with SQL queries.
install	

Class	db_column
Permission	Description (Inherit 6 common database permissions + 3 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, 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.

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.

The SELinux Notebook

use	Controls usage of system objects that require permission to "use" objects such as data types, tablespaces and operators.
-----	--

Class	db_blob
Permission	Description (Inherit 6 common database permissions + 4 unique)
Inherit Common Database Permissions	create, drop, getattr, 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 object with the lowrite() function.

Class	db_view
Permission	Description (Inherit 6 common database permissions + 1 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
expand	Allows the expansion of a 'view'.

Class	db_sequence - A sequential number generator
Permission	Description (Inherit 6 common database permissions + 3 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
get_value	Get a value from the sequence generator object.
next_value	Get and increment value.
set_value	Set an arbitrary value.

Class	db_language - Support for script languages such as Perl and Tcl for SQL Procedures
Permission	Description (Inherit 6 common database permissions + 2 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
implement	Whether the language can be implemented or not for the SQL procedure.
execute	Allow the execution of a code block using a 'DO' statement.

8.14 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.

The SELinux Notebook

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 (12 unique permissions)
admin	Allow the nscd daemon to be shut down.
getgrp	Get group information.
gethost	Get host information.
getnetgrp	
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. ??
shmemnetgrp	
shmempwd	
shmemserv	

Class	dbus - This is a userspace object for the D-BUS Messaging service that is required to run various 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.
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.

The SELinux Notebook

--	--

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 (/proc/sys/vm/mmap_min_addr). Setting this value to 0 will disable the checks. The " SELinux hardening for mmap_min_addr protections " [13] describes additional checks that will be added to the kernel to protect against some kernel exploits (by requiring CAP_SYS_RAWIO (root) and the SELinux memprotect / mmap_zero permission instead of only one or the other).

Class	service - This is a userspace object to manage systemd services.
Permission	Description (8 unique permissions)
disable	Disable services.
enable	Enable services.
kill	Kill services.
load	Load services
reload	Restart systemd services.
start	Start systemd services.
status	Read service status.
stop	Stop systemd services.

Class	proxy - This is a userspace object for gssd services.
Permission	Description (1 unique permission)
read	Read credentials.

9. Appendix B - libselinux Library Functions

These functions have been taken from the following header files of libselinux version 2.3:

```
/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.

Num.	Function Name	Description	Header File
1.	avc_add_callback	Register a callback for security events.	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 (3) after a permission check, but can also be called directly by callers who use avc_has_perm_noaudit (3) 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. Logs 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
4.	avc_cache_stats	Get cache access statistics. Fill the supplied structure with information about AVC activity since the last call to avc_init (3) or avc_reset (3).	avc.h
5.	avc_cleanup	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.	avc.h
6.	avc_compute_create	Compute SID for labeling a new object. Call the security server to obtain a	avc.h

The SELinux Notebook

Num.	Function Name	Description	Header File
		context for labeling a new object. Look up the context in the SID table, making a new entry if not found.	
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.	<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. Store a pointer to the SID structure into the memory referenced by <code>sid</code> , returning 0 on success or -1 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(3)</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(3)</code> , which stores cache entry references in them. They can increase performance on repeated queries.	<code>avc.h</code>
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(3)</code> and then call <code>avc_context_to_sid(3)</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 <code>requested</code> permissions are granted for the SID pair (<code>ssid</code> , <code>tsid</code>), interpreting the permissions based on <code>tclass</code> , and call the security server on a cache miss to obtain a new decision and add it to the cache. Update <code>aeref</code> 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 <code>requested</code> permissions are granted, -1 with <code>errno</code> set to <code>EACCES</code> 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 <code>requested</code> permissions are granted for the SID pair (<code>ssid</code> ,	<code>avc.h</code>

The SELinux Notebook

Num.	Function Name	Description	Header File
		<p><code>tsid</code>), interpreting the permissions based on <code>tclass</code>, and call the security server on a cache miss to obtain a new decision and add it to the cache. Update <code>aeref</code> to refer to an AVC entry with the resulting decisions, and return a copy of the decisions in <code>avd</code>. Return 0 if all requested permissions are granted, -1 with <code>errno</code> set to <code>EACCES</code> if any permissions are denied, or to another value upon other errors. This function is typically called by <code>avc_has_perm(3)</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.</p>	
14.	<code>avc_init</code> (deprecated)	<p>Use <code>avc_open</code></p> <p>Initialize the AVC. Initialize the access vector cache. Return 0 on success or -1 with <code>errno</code> set on failure. If <code>msgprefix</code> is NULL, use "uavc". If any callback structure references are NULL, use default methods for those callbacks (see the definition of the callback structures).</p>	<code>avc.h</code>
15.	<code>avc_netlink_acquire_fd</code>	Create a netlink socket and connect to the kernel.	<code>avc.h</code>
16.	<code>avc_netlink_check_nb</code>	Wait for netlink messages from the kernel.	<code>avc.h</code>
17.	<code>avc_netlink_close</code>	Close the netlink socket.	<code>avc.h</code>
18.	<code>avc_netlink_loop</code>	Acquire netlink socket fd. Allows the application to manage messages from the netlink socket in its own main loop.	<code>avc.h</code>
19.	<code>avc_netlink_open</code>	Release netlink socket fd. Returns ownership of the netlink socket to the library.	<code>avc.h</code>
20.	<code>avc_netlink_release_fd</code>	Check netlink socket for new messages. Called by the application when using <code>avc_netlink_acquire_fd(3)</code> to process kernel netlink events.	<code>avc.h</code>
21.	<code>avc_open</code>	Initialize the AVC. This function is identical to <code>avc_init(3)</code> except the message prefix is set to "avc" and any callbacks desired should be specified via <code>selinux_set_callback(3)</code> .	<code>avc.h</code>
22.	<code>avc_reset</code>	Flush the cache and reset statistics. Remove all entries from the cache and reset all access statistics (as returned by <code>avc_cache_stats(3)</code>) to zero. The SID mapping is not affected. Return 0 on success, -1 with <code>errno</code> set on error.	<code>avc.h</code>
23.	<code>avc_sid_stats</code>	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.	<code>avc.h</code>

The SELinux Notebook

Num.	Function Name	Description	Header File
24.	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 (3)</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
25.	checkPasswdAccess (deprecated)	Use <code>selinux_check_passwd_access (3)</code> or preferably <code>selinux_check_access (3)</code> Check a permission in the <code>passwd</code> class. Return 0 if granted or -1 otherwise.	selinux.h
26.	context_free	Free the storage used by a context.	context.h
27.	context_new	Return a new context initialized to a context string.	context.h
28.	context_range_get	Get a pointer to the <code>range</code> .	context.h
29.	context_range_set	Set the <code>range</code> component. Returns nonzero if unsuccessful.	context.h
30.	context_role_get	Get a pointer to the <code>role</code> .	context.h
31.	context_role_set	Set the <code>role</code> component. Returns nonzero if unsuccessful.	context.h
32.	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
33.	context_type_get	Get a pointer to the <code>type</code> .	context.h
34.	context_type_set	Set the <code>type</code> component. Returns nonzero if unsuccessful.	context.h
35.	context_user_get	Get a pointer to the <code>user</code> .	context.h
36.	context_user_set	Set the <code>user</code> component. Returns nonzero if unsuccessful.	context.h
37.	fgetfilecon fgetfilecon_raw	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> .	selinux.h
38.	fini_selinuxmnt	Clear <code>selinuxmnt</code> variable and free allocated memory.	selinux.h
39.	freecon	Free the memory allocated for a context by any of the <code>get*</code> calls.	selinux.h
40.	freeconary	Free the memory allocated for a context array by <code>security_compute_user (3)</code> .	selinux.h
41.	fsetfilecon fsetfilecon_raw	Wrapper for the <code>xattr</code> API - Set file context.	selinux.h

The SELinux Notebook

Num.	Function Name	Description	Header File
42.	get_default_context	Get the default security context for a user session for 'user' spawned by 'fromcon' and set *newcon 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 freecon.	get_context_list.h
43.	get_default_context_with_level	Same as get_default_context (3), but use the provided MLS level rather than the default level for the user.	get_context_list.h
44.	get_default_context_with_role	Same as get_default_context (3), but only return a context that has the specified role.	get_context_list.h
45.	get_default_context_with_rolelevel	Same as get_default_context (3), but only return a context that has the specified role and level.	get_context_list.h
46.	get_default_type	Get the default type (domain) for 'role' and set 'type' to refer to it. Caller must free via free (3). Return 0 on success or -1 otherwise.	get_default_type.h
47.	get_ordered_context_list	Get an ordered list of authorized security contexts for a user session for 'user' spawned by 'fromcon' and set *conary 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 *conary. If 'fromcon' is NULL, defaults to current context. Caller must free via freeconary (3).	get_context_list.h
48.	get_ordered_context_list_with_level	Same as get_ordered_context_list (3), but use the provided MLS level rather than the default level for the user.	get_context_list.h
49.	getcon getcon_raw	Get current context, and set *con to refer to it. Caller must free via freecon (3).	selinux.h
50.	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 (3).	selinux.h
51.	getfilecon getfilecon_raw	Wrapper for the xattr API - Get file context, and set *con to refer to it. Caller must free via freecon (3).	selinux.h
52.	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 (3).	selinux.h

The SELinux Notebook

Num.	Function Name	Description	Header File
53.	getkeycreatecon getkeycreatecon_raw	Get <code>keycreate</code> context, and set <code>*con</code> to refer to it. Sets <code>*con</code> to NULL if no key create context has been set, i.e. using default. If non-NULL, caller must free via freecon (3).	selinux.h
54.	getpeercon getpeercon_raw	Wrapper for the socket API - Get context of peer socket, and set <code>*con</code> to refer to it. Caller must free via freecon (3).	selinux.h
55.	getpidcon getpidcon_raw	Get context of process identified by <code>pid</code> , and set <code>*con</code> to refer to it. Caller must free via freecon (3).	selinux.h
56.	getprevcon getprevcon_raw	Get previous context (prior to last <code>exec</code>), and set <code>*con</code> to refer to it. Caller must free via freecon (3).	selinux.h
57.	getseuser	Get the SELinux username and level to use for a given Linux username and service. These values may then be passed into the <code>get_ordered_context_list*</code> and <code>get_default_context*</code> functions to obtain a context for the user. Returns 0 on success or -1 otherwise. Caller must free the returned strings via free (3).	selinux.h
58.	getseuserbyname	Get the SELinux username and level to use for a given Linux username. These values may then be passed into the <code>get_ordered_context_list*</code> and <code>get_default_context*</code> functions to obtain a context for the user. Returns 0 on success or -1 otherwise. Caller must free the returned strings via free (3).	selinux.h
59.	getsockcreatecon getsockcreatecon_raw	Get <code>sockcreate</code> context, and set <code>*con</code> to refer to it. Sets <code>*con</code> to NULL if no socket create context has been set, i.e. using default. If non-NULL, caller must free via freecon (3).	selinux.h
60.	init_selinuxmnt	There is a man page for this, however it is not a user accessible function (internal use only - although the <code>fini_selinuxmnt</code> is reachable).	-
61.	is_context_customizable	Returns whether a file context is customizable, and should not be relabeled.	selinux.h
62.	is_selinux_enabled	Return 1 if running on a SELinux kernel, or 0 if not or -1 for error.	selinux.h
63.	is_selinux_mls_enabled	Return 1 if we are running on a SELinux MLS kernel, or 0 otherwise.	selinux.h
64.	lgetfilecon lgetfilecon_raw	Wrapper for the <code>xattr</code> API - Get file context, and set <code>*con</code> to refer to it. Caller must free via freecon (3).	selinux.h
65.	lsetfilecon lsetfilecon_raw	Wrapper for the <code>xattr</code> API- Set file context for symbolic link.	selinux.h

The SELinux Notebook

Num.	Function Name	Description	Header File
66.	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 (3). Returns 0 on success or -1 otherwise.	get_context_list.h
67.	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
68.	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 (3) has not already been called, then this function will call it upon its first invocation with a NULL path.	selinux.h
69.	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
70.	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
71.	matchpathcon_filespec_destroy	Destroy any inode associations that have been added, e.g. to restart for a new filesystem.	selinux.h
72.	matchpathcon_filespec_eval	Display statistics on the hash table usage for the associations.	selinux.h
73.	matchpathcon_fini	Free the memory allocated by matchpathcon_init.	selinux.h
74.	matchpathcon_index	Same as matchpathcon (3), but return a specification index for later use in a matchpathcon_filespec_add (3) call.	selinux.h
75.	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 (3). 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
76.	matchpathcon_init_prefix	Same as matchpathcon_init (3), but only load entries with regexes that have stems that are prefixes of 'prefix'.	selinux.h

The SELinux Notebook

Num.	Function Name	Description	Header File
77.	mode_to_security_class	Translate mode_t to a security class string name (e.g. S_ISREG = "file").	selinux.h
78.	print_access_vector	Display an access vector in a string representation.	selinux.h
79.	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 (3). Returns 0 on success or -1 otherwise.	get_context_list.h
80.	realpath_not_final	Resolve all of the symlinks and relative portions of a pathname, but NOT the final component (same as realpath (3) unless the final component is a symlink. Resolved path must be a path of size PATH_MAX + 1.	selinux.h
81.	rpm_execon	Execute a helper for rpm in an appropriate security context.	selinux.h
82.	security_av_perm_to_string	Convert access vector permissions to string names.	selinux.h
83.	security_av_string	Returns an access vector in a string representation. User must free the returned string via free (3).	selinux.h
84.	security_canonicalize_context security_canonicalize_context_raw	Canonicalize a security context. Returns a pointer to the canonical (primary) form of a security context in <u>canoncon</u> that the kernel is using rather than what is provided by the userspace application in <u>con</u> .	selinux.h
85.	security_check_context security_check_context_raw	Check the validity of a security context.	selinux.h
86.	security_class_to_string	Convert security class values to string names.	selinux.h
87.	security_commit_booleans	Commit the pending values for the booleans.	selinux.h
88.	security_compute_av security_compute_av_raw	Compute an access decision. Queries whether the policy permits the source context <u>scon</u> to access the target context <u>tcon</u> via class <u>tclass</u> with the <u>requested</u> access vector. The decision is returned in <u>avd</u> .	selinux.h
89.	security_compute_av_flags security_compute_av__flags_raw	Compute an access decision and return the flags. Queries whether the policy permits the source context <u>scon</u> to access the target context <u>tcon</u> via class <u>tclass</u> with the <u>requested</u> access vector. The decision is returned in <u>avd</u> . that has an additional flags entry. Currently the only flag defined is SELINUX_AVD_FLAGS_PERMISSIVE that indicates the decision was computed on a permissive domain (i.e. the permissive policy language statement has been used in policy or semanage (8) has been used to set the domain in permissive mode). Note	selinux.h

The SELinux Notebook

Num.	Function Name	Description	Header File
		this does not indicate that SELinux is running in permissive mode, only the scon domain.	
90.	security_compute_create security_compute_create_raw	Compute a labeling decision and set *newcon to refer to it. Caller must free via freecon (3).	selinux.h
91.	security_compute_create_name security_compute_create_name_raw	This is identical to security_compute_create (3) but also takes the name of the new object in creation as an argument. When a type_transition rule on the given class and the scon / tcon pair has an object name extension, newcon will be returned according to the policy. Note that this interface is only supported on the kernels 2.6.40 or later. For older kernels the object name is ignored.	selinux.h
92.	security_compute_member security_compute_member_raw	Compute a polyinstantiation member decision and set *newcon to refer to it. Caller must free via freecon (3).	selinux.h
93.	security_compute_relabel security_compute_relabel_raw	Compute a relabeling decision and set *newcon to refer to it. Caller must free via freecon (3).	selinux.h
94.	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 (3).	selinux.h
95.	security_deny_unknown	Get the behavior for undefined classes / permissions.	selinux.h
96.	security_disable	Disable SELinux at runtime (must be done prior to initial policy load).	selinux.h
97.	security_get_boolean_active	Get the active value for the boolean.	selinux.h
98.	security_get_boolean_names	Get the boolean names	selinux.h
99.	security_get_boolean_pending	Get the pending value for the boolean.	selinux.h
100.	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 (3).	selinux.h
101.	security_getenforce	Get the enforce flag value.	selinux.h
102.	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
103.	security_load_policy	Load a policy configuration.	selinux.h
104.	security_policyvers	Get the policy version number.	selinux.h
105.	security_set_boolean	Set the pending value for the boolean.	selinux.h

The SELinux Notebook

Num.	Function Name	Description	Header File
106.	<code>security_set_boolean_list</code>	Save a list of booleans in a single transaction.	<code>selinux.h</code>
107.	<code>security_setenforce</code>	Set the <code>enforce</code> flag value.	<code>selinux.h</code>
108.	<code>selabel_close</code>	Destroy the specified handle, closing files, freeing allocated memory, etc. The handle may not be further used after it has been closed.	<code>label.h</code>
109.	<code>selabel_lookup</code> <code>selabel_lookup_raw</code>	Perform a labeling lookup operation. Return 0 on success, -1 with <code>errno</code> set on failure. The <code>key</code> and <code>type</code> 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 <code>con</code> and must be freed by <code>freecon</code> .	<code>label.h</code>
110.	<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. <code>SELABEL_CTX_DB</code> - SE-PostgreSQL contexts. <code>SELABEL_CTX_ANDROID_PROP</code> - property 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>
111.	<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>

The SELinux Notebook

Num.	Function Name	Description	Header File
112.	<code>selinux_binary_policy_path</code>	Return path to the binary <code>policy</code> file under the policy root directory.	<code>selinux.h</code>
113.	<code>selinux_booleans_path</code>	Return path to the <code>booleans</code> file under the policy root directory.	<code>selinux.h</code>
114.	<code>selinux_boolean_sub</code>	Reads the <code>/etc/selinux/TYPE/booleans.subs_dist</code> file looking for a record with <code>boolean_name</code> . If a record exists <code>selinux_boolean_sub</code> (3) returns the translated name otherwise it returns the original name. The returned value needs to be freed. On failure NULL will be returned.	<code>selinux.h</code>
115.	<code>selinux_booleans_subs_path</code>	Returns the path to the <code>booleans.subs_dist</code> configuration file.	<code>selinux.h</code>
116.	<code>selinux_check_access</code>	Used to check if the source context has the access permission for the specified class on the target context. Note that the permission and class are reference strings. The <code>aux</code> parameter may reference supplemental auditing information. Auditing is handled as described in <code>avc_audit</code> (3). See <code>security_deny_unknown</code> (3) for how the <code>deny_unknown</code> flag can influence policy decisions.	<code>selinux.h</code>
117.	<code>selinux_check_passwd_access</code>	Check a permission in the <code>passwd</code> class. Return 0 if granted or -1 otherwise. Replaced by <code>selinux_check_access</code> (3)	<code>selinux.h</code>
118.	<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>
119.	<code>selinux_colors_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
120.	<code>selinux_contexts_path</code>	Return path to <code>contexts</code> directory under the policy root directory.	<code>selinux.h</code>
121.	<code>selinux_current_policy_path</code>	Return path to the current policy.	<code>selinux.h</code>
122.	<code>selinux_customizable_types_path</code>	Return path to <code>customizable_types</code> file under the policy root directory.	<code>selinux.h</code>
123.	<code>selinux_default_context_path</code>	Return path to <code>default_context</code> file under the policy root directory.	<code>selinux.h</code>
124.	<code>selinux_default_type_path</code>	Return path to <code>default_type</code> file.	<code>get_default_type.h</code>
125.	<code>selinux_failsafe_context_path</code>	Return path to <code>failsafe_context</code> file under the policy root directory.	<code>selinux.h</code>
126.	<code>selinux_file_context_cmp</code>	Compare two file contexts, return 0 if equivalent.	<code>selinux.h</code>
127.	<code>selinux_file_context_homedir_path</code>	Return path to <code>file_context.homedir</code> file under the policy root directory.	<code>selinux.h</code>

The SELinux Notebook

Num.	Function Name	Description	Header File
128.	<code>selinux_file_context_local_path</code>	Return path to <code>file_context.local</code> file under the policy root directory.	<code>selinux.h</code>
129.	<code>selinux_file_context_path</code>	Return path to <code>file_context</code> file under the policy root directory.	<code>selinux.h</code>
130.	<code>selinux_file_context_subs_path</code>	Return path to <code>file_context.subs</code> file under the policy root directory.	<code>selinux.h</code>
131.	<code>selinux_file_context_subs_dist_path</code>	Return path to <code>file_context.subs_dist</code> file under the policy root directory.	<code>selinux.h</code>
132.	<code>selinux_file_context_verify</code>	Verify the context of the file 'path' against policy. Return 0 if correct.	<code>selinux.h</code>
133.	<code>selinux_get_callback</code>	Used to get a pointer to the callback function of the given <code>type</code> . Callback functions are set using <code>selinux_set_callback(3)</code> .	<code>selinux.h</code>
134.	<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>
135.	<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>
136.	<code>selinux_homedir_context_path</code>	Return path to file under the policy root directory. Note that this file will only appear in older versions of policy at this location. On systems that are managed using <code>semanage(8)</code> this is now in the policy store.	<code>selinux.h</code>
137.	<code>selinux_init_load_policy</code>	Perform the initial policy load. 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. 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.	<code>selinux.h</code>
138.	<code>selinux_lsetfilecon_default</code>	This function sets the file context to the system defaults. Returns 0 on success.	<code>selinux.h</code>
139.	<code>selinux_lxc_contexts_path</code>	Return the path to the <code>lxc_contexts</code> configuration file.	<code>selinux.h</code>
140.	<code>selinux_media_context_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>

The SELinux Notebook

Num.	Function Name	Description	Header File
141.	<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(3)</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(3)</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>
142.	<code>selinux_netfilter_context_path</code>	Returns path to the <code>netfilter_context</code> file under the policy root directory.	<code>selinux.h</code>
143.	<code>selinux_path</code>	Returns path to the policy root directory.	<code>selinux.h</code>
144.	<code>selinux_policy_root</code>	Reads the <code>/etc/selinux/config</code> file and returns the top level directory.	<code>selinux.h</code>
145.	<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(3)</code>. Returns -1 upon an error or 0 otherwise.</p>	<code>selinux.h</code>
146.	<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(3)</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>
147.	<code>selinux_removable_context_path</code>	Return path to <code>removable_context</code> file under the policy root directory.	<code>selinux.h</code>
148.	<code>selinux_securetty_types_path</code>	Return path to the <code>securetty_types</code> file under the policy root directory.	<code>selinux.h</code>
149.	<code>selinux_sepgsql_context_path</code>	Return path to <code>sepgsql_context</code> file under the policy root directory.	<code>selinux.h</code>
150.	<code>selinux_set_callback</code>	Sets the callback according to the type: <code>SELINUX_CB_LOG</code> , <code>SELINUX_CB_AUDIT</code> , <code>SELINUX_CB_VALIDATE</code> , <code>SELINUX_CB_SETENFORCE</code> , <code>SELINUX_CB_POLICYLOAD</code>	<code>selinux.h</code>
151.	<code>selinux_set_mapping</code>	Userspace class mapping support that establishes a mapping from a user-provided ordering of object classes and permissions to the numbers actually used by the loaded system policy.	<code>selinux.h</code>

The SELinux Notebook

Num.	Function Name	Description	Header File
152.	<code>selinux_set_policy_root</code>	Sets an alternate policy root directory path under which the compiled policy file and context configuration files exist.	<code>selinux.h</code>
153.	<code>selinux_status_open</code>	Open and map SELinux kernel status page.	<code>avc.h</code>
154.	<code>selinux_status_close</code>	Unmap and close kernel status page.	<code>avc.h</code>
155.	<code>selinux_status_updated</code>	Inform whether the kernel status has been updated.	<code>avc.h</code>
156.	<code>selinux_status_getenforce</code>	Get the enforce flag value.	<code>avc.h</code>
157.	<code>selinux_status_policyload</code>	Get the number of policy loads.	<code>avc.h</code>
158.	<code>selinux_status_deny_unknown</code>	Get behaviour for undefined classes/permissions.	<code>avc.h</code>
159.	<code>selinux_systemd_contexts_path</code>	Returns the path to the <code>systemd_contexts</code> configuration file.	<code>selinux.h</code>
160.	<code>selinux_reset_config</code>	Force a reset of the loaded configuration. WARNING: This is not thread safe. Be very sure that no other threads are calling into <code>libselinux</code> when this is called.	<code>selinux.h</code>
161.	<code>selinux_trans_to_raw_context</code>	Perform context translation between the human-readable format ("translated") and the internal system format ("raw"). Caller must free the resulting context via <code>freecon(3)</code> . Returns -1 upon an error or 0 otherwise. If passed NULL, sets the returned context to NULL and returns 0.	<code>selinux.h</code>
162.	<code>selinux_translations_path</code>	Return path to <code>setrans.conf</code> file under the policy root directory.	<code>selinux.h</code>
163.	<code>selinux_user_contexts_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
164.	<code>selinux_users_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
165.	<code>selinux_usersconf_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
166.	<code>selinux_virtual_domain_context_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
167.	<code>selinux_virtual_image_context_path</code>	Return path to file under the policy root directory.	<code>selinux.h</code>
168.	<code>selinux_x_context_path</code>	Return path to <code>x_context</code> file under the policy root directory.	<code>selinux.h</code>
169.	<code>selinuxfs_exists</code>	Check if <code>selinuxfs</code> exists as a kernel filesystem.	<code>selinux.h</code>
170.	<code>set_matchpathcon_canoncon</code>	Same as <code>set_matchpathcon_invalidcon(3)</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(3)</code> .	<code>selinux.h</code>
171.	<code>set_matchpathcon_flags</code>	Set flags controlling operation of <code>matchpathcon_init(3)</code> or	<code>selinux.h</code>

The SELinux Notebook

Num.	Function Name	Description	Header File
		<p>matchpathcon (3):</p> <p>MATCHPATHCON_BASEONLY - Only process the base file_contexts file.</p> <p>MATCHPATHCON_NOTRANS - Do not perform any context translation.</p> <p>MATCHPATHCON_VALIDATE - Validate/canonicalize contexts at init time.</p>	
172.	set_matchpathcon_invalidcon	Set the function used by matchpathcon_init (3) when checking the validity of a context in the file_contexts configuration. If not set, then this defaults to a test based on security_check_context (3). The function is also responsible for reporting any such error, and may include the 'path' and 'lineno' in such error messages.	selinux.h
173.	set_matchpathcon_printf	Set the function used by matchpathcon_init (3) when displaying errors about the file_contexts configuration. If not set, then this defaults to <code>fprintf(stderr, fmt, ...)</code> .	selinux.h
174.	set_selinuxmnt	Set the path to the selinuxfs mount point explicitly. Normally, this is determined automatically during libselinux initialization, but this is not always possible, e.g. for /sbin/init which performs the initial mount of selinuxfs.	selinux.h
175.	setcon setcon_raw	Set the current security context to con. 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 exec-based transitions performed via setexeccon (3). When possible, decompose your application and use setexeccon (3) + execve (3) instead. Note that the application may lose access to its open descriptors as a result of a setcon (3) unless policy allows it to use descriptors opened by the old context.	selinux.h
176.	setexeccon setexeccon_raw	Set exec security context for the next execve (3). Call with NULL if you want to reset to the default.	selinux.h
177.	setexecfilecon	Set an appropriate security context based on the filename of a helper program, falling back to a new context with the specified type.	selinux.h
178.	setfilecon setfilecon_raw	Wrapper for the xattr API - Set file context.	selinux.h

The SELinux Notebook

Num.	Function Name	Description	Header File
179.	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.	<code>selinux.h</code>
180.	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.	<code>selinux.h</code>
181.	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.	<code>selinux.h</code>
182.	<code>sidget</code> (deprecated)	From 2.0.86 this is a no-op.	<code>avc.h</code>
183.	<code>sidput</code> (deprecated)	From 2.0.86 this is a no-op.	<code>avc.h</code>
184.	<code>string_to_av_perm</code>	Convert string names to access vector permissions.	<code>selinux.h</code>
185.	<code>string_to_security_class</code>	Convert string names to security class values.	<code>selinux.h</code>

10. Appendix C - 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:

<https://github.com/SELinuxProject/selinux/wiki/Tools>

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 semanage (8), 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 semanage (8) / semodule (8) 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 init 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.

11. Appendix D - Document References

<i>Ref.</i>	<i>Title</i>	<i>Author</i>
1.	Implementing SELinux as a Linux Security Module	S. Smalley, C. Vance, W. Salamon
2.	Security-Enhanced PostgreSQL Security Wiki	K. Kohei
3.	SELinux Policy Module Primer	J. Brindle
4.	Polyinstantiation of directories in an SELinux system	R. Coker
5.	Iptables Tutorial	O. Andreasson
6.	New secmark-based network controls for SELinux	J. Morris
7.	Transitioning to Secmark	Paul Moore
8.	Fallback Label Configuration Example	Paul Moore
9.	Leveraging IPSec for Distributed Authorization	Trent Jaeger
10.	IPSec HOWTO	Ralf Spenneberg
11.	Secure Networking with SELinux	J. Brindle
12.	SELinux by Example	F. Mayer K Macmillan D Caplan
13.	SELinux hardening for mmap_min_addr protections	E. Paris
14.	Application of the Flask Architecture to the X Window System Server	E. Walsh
15.	X Access Control Extension Specification	E. Walsh
16.	A secure web application platform powered by SELinux	K. Kohei
17.	Kernel-based Virtual Machine	Red Hat
18.	How Does Xen Work	Xen Project
19.	Xen Security Modules	G. Coker
20.	The Case for Security Enhanced (SE)Android	S. Smalley

12. Appendix E - Policy Validation Example

This example has been taken from <http://selinuxproject.org/page/PolicyValidate>.

libsemanage is the library responsible for building a kernel policy from policy modules. It has many features but one that is rarely mentioned is the policy validation hook. This example will show how to make a basic validator and tell libsemanage to run it before allowing any policy updates.

The sample validator uses **sesearch** (1) to search for a rule between `user_t` and `shadow_t`. The purpose of this validator is to never allow a policy update that allows `user_t` to access `shadow_t`.

To use the script below requires the `setools-console` package to be installed.

Make a file in `/usr/local/bin/validate` that contains the following (run `chmod +x` or **semodule** (8) will fail):

```
#!/bin/bash

# Usage: validate <policy file>

# The following searches for a file rule with user_t as the source and shadow_t
# as the target.
# If the output of sesearch has "Found", meaning matching rules were found, then
# grep will return 0 otherwise it will return 1. This is actually the reverse of the
# logic required, so it will be reversed.

sesearch --allow -s user_t -t shadow_t -c file $1 | grep "Found" > /dev/null

if [ $? == 1 ]; then
    exit 0
fi

exit 1
```

Then add the validation script to `/etc/selinux/semanage.conf`

```
[verify kernel]
path = /usr/local/bin/validate
args = $@
[end]
```

Next try rebuilding the policy with no changes:

```
# semodule -B
```

It should succeed, therefore build a module that would violate this rule:

```
module badmod 1.0;
require {
    type user_t, shadow_t;
    class file { read };
}

allow user_t shadow_t : file read;
```

Do the standard compilation steps:

```
# checkmodule -o badmod.mod badmod.te -m -M
checkmodule: loading policy configuration from badmod.te
checkmodule: policy configuration loaded
checkmodule: writing binary representation (version 17) to badmod.mod
# semodule_package -m badmod.mod -o badmod.pp
```

And then attempt to insert it:

```
# semodule -i badmod.pp
semodule: Failed!
```

Now run `sesearch` to ensure that there is no matching rule:

```
# sesearch --allow -s user_t -t shadow_t -c file
```

Note that there are also `[verify module]` and `[verify linked]` options as described in the [semanage.conf](#) file section.

13. Appendix F - 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

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.