The SELinux Notebook



Volume 1 The Foundations

(2nd Edition)

0. Notebook Information

0.1 Copyright Information

Copyright © 2010 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 "<u>GNUFree Documentation</u> <u>License</u>".

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 <<u>http://www.gnu.org/licenses/</u>>.

Edition	Date	Changes
1.0	20 th Nov '09	First released.
2.0	8 th May '10	Split the Notebook into two volumes:
		 The Foundations - covers SELinux and its supporting services.
		 Sample Policy Source - contains sample application and policy source code to build a simple message filter and experiment with X- Windows.
		In this volume:
		• Updated all relevant sections to reflect Fedora 12 release and correct errors.
		• Added secolors.conf and VM configuration files, typebounds Policy language statement.
		Added sections on:
		• XSELinux for X-Windows support and sample polyinstantiation code and policy.
		• Virtual machine support with examples.

0.2 Revision History

• SE-PostgreSQL support with an example database.
• Apache / SELinux-Plus support.

0.3 Acknowledgements

Logo designed by Máirín Duffy

0.4 Abbreviations

Term	Definition	
apol	Policy analysis tool	
AV	Access Vector	
AVC	Access Vector Cache	
BLP	Bell-La Padula	
CC	Common Criteria	
CMW	Compartmented Mode Workstation	
DAC	Discretionary Access Control	
F-12	Fedora 12	
FLASK	Flux Advanced Security Kernel - A security-enhanced version of the Fluke kernel and OS developed by the Utah Flux team and the US Department of Defence.	
Fluke	Flux μ -kernel Environment - A specification and implementation of a micro kernel and operating system architecture.	
Flux	The Flux Research Group (<u>http://www.cs.utah.edu/flux/</u>)	
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	
ОМ	Object Manager	
PAM	Pluggable Authentication Module	
RBAC	Role-based Access Control	
rpm	Red Hat Package Manager	
SELinux	Security-Enhanced Linux	
SID	Security Identifier	
SL	Security Level	
SLIDE	SELinux Integrated Development Environment	

Term	Definition
SMACK	Simplified Mandatory Access Control Kernel
SUID	Super-user Identifier
ТЕ	Type Enforcement
UID	User Identifier
XACE	X (windows) Access Control Extension

0.5 Index

0. NOTEBOOK INFORMATION	2
0.1 Copyright Information	2
0.2 Revision History.	
0.3 Acknowledgements	
0.4 Abbreviations.	
0.5 INDEX	
1. THE SELINUX NOTEBOOK	
1.1 Introduction.	
1.2 Volume 1 - The Foundations Overview	
1.3 VOLUME 2 - SAMPLE POLICY SOURCE OVERVIEW	
1.4 Relevant F-12 Packages	
2. SELINUX OVERVIEW	
2.1 INTRODUCTION	
2.2 Core SELinux Components	
2.3 MANDATORY ACCESS CONTROL (MAC).	
2.4 Type Enforcement (TE)	
2.4.1 Constraints	
2.5 Role-Based Access Control (RBAC)	
2.6 Security Context.	
2.7 Subjects	
2.8 Objects.	
2.8.1 Object Classes and Permissions	23
2.8.2 Allowing a Process Access to Resources	24
2.8.3 Labeling Objects	25
2.8.3.1 Labeling Extended Attribute Filesystems	
2.8.3.1.1 Copying and Moving Files	<u>26</u>
2.8.3.2 Labeling Subjects	
2.8.4 Object Reuse	
2.9 Domain and Object Transitions	
2.9.1 Domain Transition	
2.9.1.1 Type Enforcement Rules	30
2.9.2 Object Transition	
2.10 Multi-Level Security and Multi-Category Security	
2.10.1 Security Levels	
2.10.1.1 MLS / MCS Range Format	
2.10.1.2 Translating Levels	
2.10.2 Managing Security Levels via Dominance Rules	

2.10.3 MLS Labeled Network and Database Support	<u>38</u>
2.10.4 Common Criteria Certification	<u>38</u>
2.11 Types of SELinux Policy	39
2.11.1 Example Policy	39
2.11.2 Reference Policy.	39
2.11.2.1 Policy Functionality Based on Name or Type	40
2.11.3 Custom Policy	41
2.11.4 Monolithic Policy	41
2.11.5 Loadable Module Policy	41
2.11.5.1 Optional Policy	
2.11.6 Conditional Policy	
2.11.7 Binary Policy.	
2.11.8 Policy Versions	
2.12 SELINUX PERMISSIVE AND ENFORCING MODES	
2.13 Audit Logs	
2.13.1 SELinux-aware Application Events	
2.13.2 AVC Audit Events	
2.14 Polyinstantiation.	
2.14.1 Polyinstantiated Objects	
2.14.2 Polyinstantiation support in PAM	
2.14.2.1 namespace.conf Configuration File	
2.14.2.2 Example Configurations	<u>51</u> 51
2.14.3 Polyinstantiation support in X-Windows	
2.14.4 Polyinstantiation support in the Reference Policy	
2.15 PAM LOGIN PROCESS.	
2.16 Linux Security Module and SELinux	
2.16.1 The LSM Module	
2.16.2 The SELinux Module	
2.16.2.2 Process Transition Walk-thorough	
2.16.2.3 SELinux Filesystem	
2.16.2.4 SELinux Boot Process.	
2.17 SELINUX NETWORKING SUPPORT.	69
<u>2.17.1 compat_net Controls</u>	
<u>2.17.2 SECMARK</u>	
2.17.3 NetLabel - Fallback Peer Labeling	
2.17.4 Labeled IPSec.	
<u>2.17.5 NetLabel - CIPSO</u>	
2.18 SELINUX VIRTUAL MACHINE SUPPORT.	75
2.18.1 KVM / QEMU Support	76
2.18.2 libvirt Support.	
2.18.2.1 Default Mode	
2.18.2.2 Shared Image Mode.	
2.18.2.3 Readonly Image Mode.	79
2.18.2.4 Static Labeling.	
2.18.2.4.1 Configuring the unconfined_t image	
	<u>84</u>
2.19 SELINUX X-WINDOWS SUPPORT.	
2.19.1 Infrastructure Overview	
2.19.2 Polyinstantiation	88

2.19.3 Configuration Information	88
2.19.3.1 Determine OM X-extension Opcode	88
2.19.3.2 Configure OM in Permissive Mode	88
2.19.3.3 Disable the OM.	
2.19.3.4 The x contexts File.	
2.20 SELINUX POSTGRESQL SUPPORT.	92
2.20.1 SE-PostgreSQL Overview	
2.20.2 SE-PostgreSOL Extensions	
2.20.2.1 Extended SQL Statements	
2.20.2.2 Additional SQL Functions.	
2.20.2.3 Additional Utilities	
2.20.2.4 Additional postgresql.conf Entries.	
	96
2.20.2.6 Logging Security Events.	
2.20 Apache SELINUX SUPPORT.	
2.21.1 mod selinux Overview.	
—	102
3. SELINUX CONFIGURATION FILES	<u>103</u>
3.1 Introduction.	103
3.2 GLOBAL CONFIGURATION FILES.	
3.2.1 /etc/selinux/config File.	
3.2.2 /etc/selinux/semanage.conf File.	
3.2.3 /etc/selinux/restorecond.conf File	102
<u>3.2.5 /etc/settinux/restorecond.conj File</u>	100
3.2.4 /etc/sestatus.conf File 3.2.5 /etc/security/sepermit.conf File	
3.3 POLICY STORE CONFIGURATION FILES.	
3.3.1 modules/ Files	$\frac{110}{110}$
3.3.2 modules/active/base.pp File	$\frac{110}{110}$
3.3.3 modules/active/base.linked File.	
<u>3.3.4 modules/active/commit_num File.</u>	
3.3.5 modules/active/file_contexts.template File	
	<u>114</u>
3.3.7 modules/active/homedir_template File	
3.3.8 modules/active/file_contexts.homedirs File	
3.3.9 modules/active/netfilter_contexts & netfilter.local File	
3.3.10 modules/active/policy.kern File	
3.3.11 modules/active/seusers_final and seusers Files	
3.3.12 modules/active/users_extra, users_extra.local and users.local Files	
3.3.13 modules/active/booleans.local File	<u>121</u>
3.3.14 modules/active/file_contexts.local File	
3.3.15 modules/active/interfaces.local File	<u>122</u>
3.3.16 modules/active/nodes.local File	<u>122</u>
3.3.17 modules/active/ports.local File	<u>123</u>
3.3.18 modules/active/modules Directory Contents	<u>123</u>
3.4 Policy Configuration Files.	123
3.4.1 seusers File	<u>124</u>
3.4.2 setrans.conf File	<u>125</u>
3.4.3 secolor.conf File	
3.4.4 policy/policy. <ver> File</ver>	

3.4.5 contexts/customizable_types File	129
3.4.6 contexts/default_contexts File	129
3.4.7 contexts/debus_contexts File	<i>131</i>
3.4.8 contexts/default_type File	132
3.4.9 contexts/failsafe_context File	133
3.4.10 contexts/initrc_context File	
3.4.11 contexts/netfilter_contexts File	134
3.4.12 contexts/removable_contexts File	134
3.4.13 contexts/securetty_types File	
3.4.14 contexts/userhelper_context File	
3.4.15 contexts/virtual_domain_context File	
3.4.16 contexts/virtual_image_context File	
<u>3.4.17 contexts/x_contexts File</u>	
3.4.18 contexts/files/file_contexts File	
3.4.19 contexts/files/file_contexts.local File	
3.4.20 contexts/files/file_contexts.homedirs File	
3.4.21 contexts/files/media File	<u>140</u>
3.4.22 contexts/users/[seuser_id] File	<i>141</i>
4. SELINUX POLICY LANGUAGE	142
4.1 Introduction.	
4.2 Policy Statements and Rules	
4.2.1 Policy Source Files	
4.2.2 Conditional, Optional and Require Statement Rules	
4.2.3 MLS Statements and Optional MLS Components	
4.2.4 General Statement Information	
4.2.5 SELinux Indentifier Naming Conventions	
4.2.6 Section Contents	
4.3 Type Enforcement and Attribute Statements	
4.3.1 type Statement.	
4.3.2 attribute Statement	
4.3.3 typeattribute Statement	151
4.3.4 typealias Statement.	152
4.4 Type Enforcement Rules	153
4.4.1 type transition Statement	
4.4.2 type change Statement	<u>155</u>
4.4.3 type_member Statement	<u>156</u>
4.4.4 typebounds Statement	<u>156</u>
4.5 Access Vector Rules	157
4.5.1 allow Rule	<u>158</u>
4.5.2 dontaudit Rule	
4.5.3 auditallow Rule	
4.5.4 neverallow Rule	
<u>4.6 User Statement</u>	160
4.6.1 user Statement	<u>160</u>
<u>4.7 Role Statement</u>	
4.7.1 role Statement	162
4.8 Role Rules	
4.8.1 Role allow Rule	
4.8.2 role transition Rule	164

4.8.3 Role dominance Rule	<u>166</u>
4.9 Conditional Policy Statements	166
4.9.1 bool Statement	167
4.9.2 if Statement	<u>168</u>
4.10 Constraint Statements	170
4.10.1 constrain Statement	170
4.10.2 validatetrans Statement	
4.11 FILE SYSTEM LABELING STATEMENTS	
4.11.1 fs use xattr Statements	
4.11.2 fs_use_task Statement	
4.11.3 fs_use_trans Statement	
4.11.4 genfscon Statements	
4.12 Network Labeling Statements	
<u>4.12.1 IP Address Formats.</u>	
4.12.1.1 IPv4 Address Format.	179
4.12.1.2 IPv6 Address Formats	
4.12.2 netifcon Statement	
4.12.3 nodecon Statement.	
4.12.4 portcon Statement.	
4.13 MLS STATEMENTS.	
4.13.1 sensitivity Statement	<u>104</u> 194
<u>4.13.3 category Statement.</u>	
<u>4.13.4 level Statement.</u>	
<u>4.13.5 range_transition Statement.</u>	
4.13.5.1 MLS range Definition.	
4.13.6 mlsconstrain Statement.	
<u>4.13.7 mlsvalidatetrans Statement.</u>	
4.14 POLICY SUPPORT STATEMENTS.	
<u>4.14.1 module Statement.</u>	
<u>4.14.2 require Statement</u>	
4.14.3 optional Statement	
4.14.4 policycap Statement	
<u>4.14.5 permissive Statement</u>	
4.15 OBJECT CLASS AND PERMISSION STATEMENTS	
<u>4.15.1 Object Classes</u>	
4.15.2 Permissions	
4.15.2.1 Defining common Permissions	199
4.16 Security ID (SID) Statement	
<u>4.16.1 sid Statement</u>	<u>200</u>
4.16.2 sid context Statement	<u>201</u>
5. THE REFERENCE POLICY.	203
5.1 INTRODUCTION.	
5.1.1 Notebook Reference Policy Information	
5.2 Reference Policy Overview.	
5.2.1 Distributing Policies	
5.2.2 Policy Functionality	
5.2.3 Reference Policy Module Files	
5.2.4 Reference Policy Documentation	

5.3 Reference Policy Source	
5.3.1 Source Layout	209
5.3.2 Reference Policy Files and Directories	<u>212</u>
5.3.3 Source Configuration Files	<u>214</u>
5.3.3.1 Reference Policy Build Options - build.conf.	214
5.3.3.2 Reference Policy Build Options – policy/modules.conf	
5.3.3.2.1 Building the modules.conf File	
5.3.4 Source Installation and Build Make Options	
5.3.5 Booleans, Global Booleans and Tunable Booleans	
5.3.6 Modular Policy Build Structure	221
5.3.7 Creating Additional Layers	
5.4 Installing and Building the Reference Policy Source	
5.4.1 Installation and Configuration	
5.4.2 Building the targeted Policy Type	
5.4.3 Checking the Build.	
5.4.4 Running with the new Policy	
5.5 Reference Policy Headers	
5.5.1 Building and Installing the Header Files	
5.5.2 Using the Header Files	230
5.5.3 Using F-12 Supplied Headers	
5.6 Reference Policy Support Macros	
5.6.1 Loadable Policy Macros	
5.6.1.1 policy module Macro	
5.6.1.2 gen require Macro.	
5.6.1.3 optional policy Macro	
5.6.1.4 gen tunable Macro	
5.6.1.5 tunable_policy Macro	
5.6.1.6 interface Macro.	
5.6.1.7 template Macro	
5.6.2 Miscellaneous Macros	
5.6.2.1 gen context Macro.	
5.6.2.2 gen user Macro	
5.6.2.3 gen_bool Macro	
5.6.3 MLS and MCS Macros	
5.6.3.1 gen_cats Macro.	
5.6.3.2 gen sens Macro	
5.6.3.3 gen levels Macro.	
5.6.3.4 System High/Low Parameters.	
5.6.4 ifdef / ifndef Parameters.	
5.6.4.1 hide broken symptoms	
5.6.4.2 enable mls and enable mcs	
5.6.4.3 enable ubac	250
5.6.4.4 direct sysadm daemon	
5.7 Module Expansion Process.	
5.7.1 Module Expansion	<u>253</u>
5.7.2 File Context Expansion	
6. APPENDIX A - OBJECT CLASSES AND PERMISSIONS	261
6.1 Introduction	
6.2 Defining Object Classes and Permissions	

6.3 Common Permissions	
6.3.1 Common File Permissions	
6.3.2 Common Socket Permissions	
6.3.3 Common IPC Permissions	263
6.3.4 Common Database Permissions	
6.3.5 Common X_Device Permissions	
6.4 File Object Classes	
6.5 Network Object Classes	
6.5.1 IPSec Network Object Classes	
6.5.2 Netlink Object Classes	
6.5.3 Miscellaneous Network Object Classes	
<u>6.6 IPC Object Classes</u>	
6.7 PROCESS OBJECT CLASS.	
6.8 Security Object Class.	
6.9 System Operation Object Class.	
6.10 Kernel Service Object Class.	
6.11 CAPABILITY OBJECT CLASSES.	
6.12 X WINDOWS OBJECT CLASSES.	
6.13 DATABASE OBJECT CLASSES.	
6.14 Miscellaneous Object Classes	
7. APPENDIX B – SELINUX COMMANDS	
8. APPENDIX C – API SUMMARY FOR LIBSELINUX	
8. APPENDIX C – API SUMMARY FOR LIBSELINUX	
8. APPENDIX C – API SUMMARY FOR LIBSELINUX 9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE	
	301
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE 9.1 Introduction 9.2 SE-PostgreSQL Walk-through	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE 9.1 INTRODUCTION 9.2 SE-PostgreSQL Walk-through 9.2.1 SE-PostgreSQL Functions 9.2.1.1 Get / Set Security Context Components	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE 9.1 INTRODUCTION 9.2 SE-PostgreSQL Walk-through 9.2.1 SE-PostgreSQL Functions 9.2.1.1 Get / Set Security Context Components	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE. 9.1 INTRODUCTION. 9.2 SE-POSTGRESQL WALK-THROUGH. 9.2.1 SE-PostgreSQL Functions. 9.2.1.1 Get / Set Security Context Components. 9.2.1.2 Get Connection Information. 9.2.1.3 Reclaiming Unused Labels.	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE. 9.1 INTRODUCTION. 9.2 SE-POSTGRESQL WALK-THROUGH. 9.2.1 SE-PostgreSQL Functions. 9.2.1.1 Get / Set Security Context Components. 9.2.1.2 Get Connection Information. 9.2.1.3 Reclaiming Unused Labels. 10. APPENDIX E - GENERAL INFORMATION. 10.1 Bugs and Features.	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE. 9.1 INTRODUCTION. 9.2 SE-PostgreSQL WALK-THROUGH. 9.2.1 SE-PostgreSQL Functions. 9.2.1.1 Get / Set Security Context Components. 9.2.1.2 Get Connection Information. 9.2.1.3 Reclaiming Unused Labels. 10. APPENDIX E - GENERAL INFORMATION. 10.1 Bugs and Features. 10.1.1 semanage - roles get deleted.	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE. 9.1 INTRODUCTION	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE. 9.1 INTRODUCTION. 9.2 SE-POSTGRESQL WALK-THROUGH. 9.2.1 SE-PostgreSQL Functions. 9.2.1.1 Get / Set Security Context Components. 9.2.1.2 Get Connection Information. 9.2.1.3 Reclaiming Unused Labels. 10. APPENDIX E - GENERAL INFORMATION. 10.1 Bugs and Features. 10.1.1 semanage - roles get deleted. 10.1.2 apol Not Loading. 10.1.3 apol not showing all screen in window.	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE. 9.1 INTRODUCTION. 9.2 SE-POSTGRESQL WALK-THROUGH. 9.2.1 SE-PostgreSQL Functions. 9.2.1.1 Get / Set Security Context Components. 9.2.1.2 Get Connection Information. 9.2.1.3 Reclaiming Unused Labels. 10. APPENDIX E - GENERAL INFORMATION. 10.1 Bugs and Features. 10.1.1 semanage - roles get deleted. 10.1.2 apol Not Loading. 10.1.3 apol not showing all screen in window. 10.1.4 racoon coredumps.	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE. 9.1 INTRODUCTION. 9.2 SE-POSTGRESQL WALK-THROUGH. 9.2.1 SE-PostgreSQL Functions. 9.2.1.1 Get / Set Security Context Components. 9.2.1.2 Get Connection Information. 9.2.1.3 Reclaiming Unused Labels. 10. APPENDIX E - GENERAL INFORMATION. 10.1 Bugs and Features. 10.1.1 semanage - roles get deleted. 10.1.2 apol Not Loading. 10.1.3 apol not showing all screen in window. 10.1.4 racoon coredumps. 10.1.5 Red Hat SELinux Config Utility. 10.2 Root Login from GNOME. 10.3 BUILDING A SOURCE RPM.	
9. APPENDIX D - SE-POSTGRESQL DATABASE EXAMPLE	

1. The SELinux Notebook

1.1 Introduction

This Notebook is split into two volumes:

- 1. **The Foundations** that describes Security-Enhanced Linux (SELinux) services as built into the Fedora 12 release¹ of GNU / Linux.
- 2. **Sample Policy Source** that contains sample policy and application code to build a simple policy to experiment with a message filter and X-Windows polyinstantiation.

These 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) The Virtual Machine, X-Windows, SE-PostgreSQL and Apache/SELinux-Plus SELinux-aware capabilities.
- d) The core SELinux policy language and how basic policy modules can be constructed for instructional purposes.
- e) The core SELinux policy management tools with examples of usage.
- f) The Reference Policy architecture, its supporting services and how it is implemented.

Note that his Notebook will not explain how the SELinux implementations are managed within each GNU / Linux distribution as they have their own supporting documentation (Fedora has the Fedora 12 SELinux User Guide [Ref. 1] and Gentoo has the <u>Gentoo SELinux Handbook</u> [Ref. 2]).

Major parts of these Notebooks have been added to the <u>SELinux Project</u> web site as part of the SELinux documentation project.

1.2 Volume 1 - The Foundations Overview

This volume has the following sections:

SELinux Overview - Gives a high level 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 and their object managers have been implemented (such as X-Windows, SE-PostgreSQL and virtual machines).

SELinux Configuration Files - Describes all the known SELinux configuration files in F-12 with samples. Also lists any specific commands or libselinux APIs used to manage them.

SELinux Policy Language - Gives a brief description of each policy language statement, with supporting examples taken from the Reference Policy source.

¹ This Notebook uses Fedora 12 simply because that is what is installed on the authors test system.

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

Object Classes and Permissions - Describes the SELinux object classes and permissions. These have been updated to reflect those in the 20091117 Reference Policy build.

SELinux Commands - Describes each of the core SELinux commands.

API Summary for libselinux - Contains a sorted alphabetical list of libselinx library functions with comments extracted from the header files.

SE-PostgreSQL Database Example - Walks through setting up a simple database with each object created having a unique security context to demonstrate how they are implemented. Also shows the additional SE-PostgreSQL functions.

General Information - This section contains information about some minor problems encountered and information that could be useful.

References - List of references used within this Notebook.

1.3 Volume 2 - Sample Policy Source Overview

For reference Volume 2 - Sample Policy Source has the following sections:

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.

Building the Message Filter Loadable Modules - Describes how to build a simple network and file handling application with policy using SECMARK and NetLabel services.

Experimenting with X-Windows - Builds sample copy and paste application and policy to demonstrate polyinstantiated selections. Also has a simple test application for the XSElinux extension Get/Set functions.

Policy Investigation Tools - Investigate the sample message filter application policy using the Tresys SETools apol, sechecker and sediff.

NetLabel Module Support for network_peer_controls - This builds on the modules developed in the Building the Message Filter section to implement an enhanced module to support the network peer controls.

Labeled IPSec Module Example - This builds on the modules developed in the Building the Message Filter section to implement Labeled IPSec.

Implementing a constraint - This builds on the modules developed in the Building a Basic Policy section to show a simple constraint statement and its impact on the policy.

1.4 Relevant F-12 Packages

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

```
checkpolicy-2.0.19-3.fc12.i686
checkpolicy-2.0.19-3.fc12.src
coreutils-7.6-8.f12.src
ipsec-tools-0.7.3-4.fc12.i686
kernel-2.6.31.5-127.fc12.i686
kernel-2.6.31.5-127.fc12.src
libselinux-2.0.90-5.fc12.i686
libselinux-devel-2.0.90-5.fc12.i686
libselinux-python-2.0.90-5.fc12.i686
libselinux-utils-2.0.90-5.fc12.i686
libsemanage-2.0.45-1.fc12.i686
libsemanage-devel-2.0.45-1.fc12.i686
libsemanage-python-2.0.45-1.fc12.i686
libsepol-2.0.41-3.fc12.i686
libsepol-devel-2.0.41-3.fc12.i686
libsepol-static-2.0.41-3.fc12.i686
libsepol-2.0.41-3.fc12.src
libvirt-0.7.1-15.f12.src
mcstrans-0.3.1-3.fc12.i686
mod selinux-2.2.2015-3.fc12.src
netlabel tools-0.19-3.fc12.i686
policycoreutils-2.0.79-1.fc12.i686
policycoreutils-gui-2.0.79-1.fc12.i686
policycoreutils-sandbox-2.0.79-1.fc12.i686
policycoreutils-python-2.0.79-1.fc12.i686
policycoreutils-newrole-2.0.79-1.fc12.i686
postgresql-libs-8.4.3-1.fc12.i686
postgresql-8.4.3-1.fc12.i686
postgresql-server-8.4.3-1.fc12.i686
qemu-0.12.3-2.fc12.src
selinux-policy-3.6.32-103.fc12.src
selinux-policy-3.6.32-103.fc12.noarch
selinux-policy-doc-3.6.32-103.fc12.noarch
selinux-policy-minimum-3.6.32-103.fc12.noarch
selinux-policy-mls-3.6.32-103.fc12.noarch
selinux-policy-targeted-3.6.32-103.fc12.noarch
sepostgresql-8.4.2-2583.fc12.i686
setools-3.3.6-4.fc12.i686
setools-console-3.3.6-4.fc12.i686
setools-gui-3.3.6-4.fc12.i686
setools-libs-3.3.6-4.fc12.i686
setools-libs-java-3.3.6-4.fc12.i686
setools-libs-tcl-3.3.6-4.fc12.i686
xen-3.4.2-1.fc12.src
```

The gcc tools will be required to compile and link the test 'C' applications used in some of the scenarios (gcc-4.4.2-20.i686 and libgcc-4.4.2-20.i686 rpms are installed on the test machine that is using the kernel-2.6.31.5-127.fcl2.i686 rpm).

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 <u>Flux</u> and <u>NSA</u> websites.

This Notebook will concentrate on describing SELinux as delivered in the Fedora F-12 release.

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

2.2 Core SELinux Components

Figure 2.1 shows a high level diagram of the SELinux core components that manage enforcement of the policy and comprise of the following:

- 1. A <u>subject</u> that must be present to cause an action to be taken by an <u>object</u> (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.

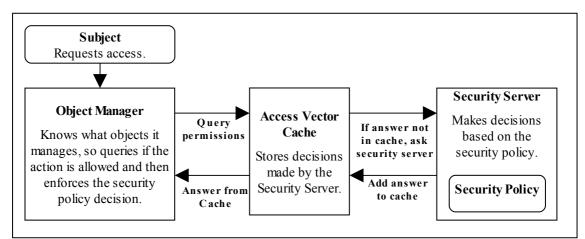


Figure 2.1: High Level Core SELinux Components - *Decisions by the Security Server are cached in the AVC to enhance performance of future requests.*

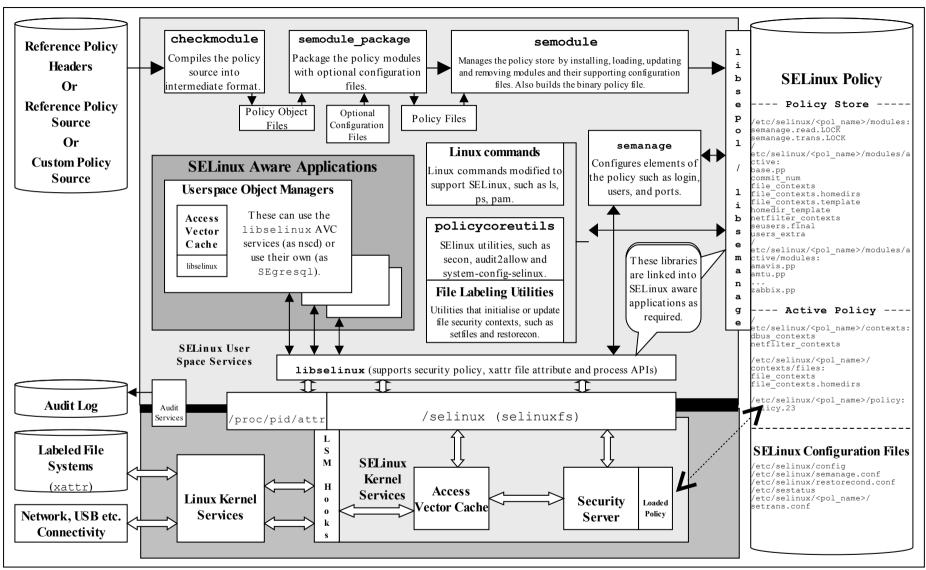


Figure 2.2: High Level SELinux Architecture – Showing the major supporting services

Figure 2.2 shows a more complex diagram of kernel and userspace with a number of supporting services which are used to manage the SELinux environment. This diagram will be referenced a number of times to explain areas of SELinux, therefore using Figure 2.2 as the reference and starting from the bottom:

a) In the current implementation of SELinux, the security server is embedded in the kernel² with the policy being loaded from userspace via a series of functions contained in a library (<u>libselinux</u>).

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

kernel space – These object manages 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 <u>LSM</u> section. The SELinux kernel AVC service is used to cache their requests and the security servers response.

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

- b) The loadable policy (right hand side of Figure 2.2) and its supporting configuration files are contained in the /etc/selinux directory. This directory contains the main SELinux configuration file (config) that names the policy to be loaded and the initial status of SELinux at boot time (enforcing⁴ the policy or not). The area also holds all policies that can be activated in their respective directories /etc/selinux/<policy_name> (e.g. /etc/selinux/targeted would hold the 'targeted' F-12 policy and all its configuration files). All know configuration files for F-12 SELinux are shown in the <u>SELinux Configuration Files</u> section.
- c) SELinux supports a 'modular policy', this means that a policy does not have to be one large policy, but can be built from modules. A modular policy consists of a base policy that contains the mandatory information (such as object classes, permissions etc.), and zero or more policy modules that generally support a particular application or service. These modules are compiled, linked, and held in a 'policy store' where they can be built into a binary format that is then loaded into the security server. The types of policy and their construction are covered in the Types of SELinux Policy section.

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

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

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

- d) To be able to build the policy in the first place, policy source is required (top left hand side of Figure 2.2). This can be supplied in two basic ways:
 - i) as source code written using the <u>SELinux Policy Language</u>. This is how the simple policies have been written to support the examples in this Notebook, however it is not recommended for real-world policy development.
 - ii) using the Reference Policy that uses high level macros to define policy rules as well as the policy language. This is the standard way policies are now built for SELinux distributions such as F-12 and is discussed in the <u>Reference Policy</u> section.
- e) To be able to compile and link the source code and load it into the security server requires a number of tools (top of <u>Figure 2.2</u>). These are used to build the <u>sample policy modules</u> where their use is described.
- f) To enable system administrators to manage the policy, the SELinux environment and label file systems also requires tools and modified GNU / Linux commands. These are mentioned throughout the Notebook as needed and summarised in <u>Appendix B – SELinux Commands</u>. 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 <u>Audit Logs</u> section describes the format of these AVC security events.
- h) SELinux supports network services that are described in the <u>SELinux</u> <u>Networking Support</u> section, and the message filter (described in volume 2) is built using some of these features.

The <u>Linux Security Module and SELinux</u> 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:

- <u>subjects</u> are processes.
- <u>objects</u> are system resources such as files, sockets, etc.
- security attributes are the <u>security context</u>.
- Security Server within the Linux kernel authorizes access (or not) using the:
 - security policy (or policy) that describes rules that must be obeyed.

Note that the subject (and therefore the user) cannot decide to bypass the policy rules being enforced by the MAC policy with SELinux enabled. Contrast this to standard

Linux Discretionary Access Control (DAC), which also governs the ability of subjects to access objects, however it allows users to make policy decisions.

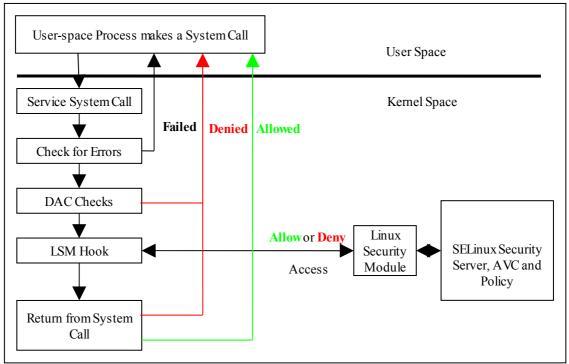


Figure 2.3: Processing a System Call – *The DAC checks are carried out first, if they pass then the Security Server is consulted for a decision.*

SELinux supports two forms of MAC:

Type Enforcement – Where processes run in domains and the actions on objects are controlled by the policy. This is the implementation used for general purpose MAC within SELinux. The <u>Type Enforcement</u> section covers this in more detail.

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

2.4 Type Enforcement (TE)

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

The SELinux type identifier is a simple variable-length string that is defined in the policy and then associated to a <u>security context</u>. It is also used in the majority of <u>SELinux language statements and rules</u> used to <u>build a policy</u> that will, when loaded into the security server, enforce the policy.

⁵ There are various 'type enforcement' technologies.

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

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

While SELinux refers to a subject as being an active process that is associated to a domain type, the scope of an SELinux type enforcement domain can vary widely. For example in the simple policy built in the Building a Basic Policy section of volume 2, all the processes on the system run in the unconfined_t domain, therefore every process is 'of type unconfined_t' (that means it can do whatever it likes within the limits of the standard Linux DAC policy).

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

2.4.1 Constraints

Within a TE environment the way that subjects are allowed to access an object is via an <u>allow rule</u>, for example:

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

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

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

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

There are a number of different constraint statements within the policy language to support areas such as MLS (see the <u>Constraint Statements</u> and <u>MLS Statements</u> sections).

2.5 Role-Based Access Control (RBAC)

To further control access to TE domains SELinux makes use of role-based access control (RBAC). This feature allows SELinux users to be associated to one or more roles, where each role is then associated to one or more domain types as shown in Figure 2.4. Note that GNU / Linux users are not a direct part of the RBAC feature, they are associated to SELinux users via SELinux specific commands⁶ such as:

semanage login - That manages the association of GNU / Linux users (or groups of users) to SELinux users.

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

Figure 2.4 shows how the SELinux user and roles are associated within the basic loadable modules that form the simple message filter exercise described in Volume 2.

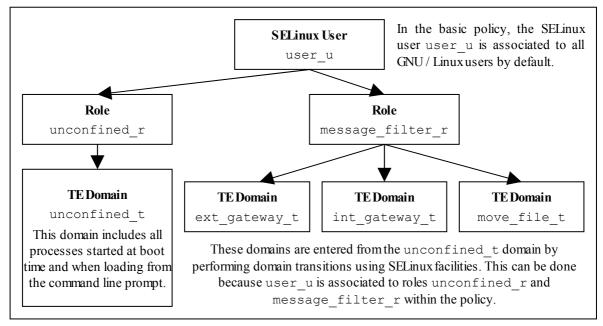


Figure 2.4: Role Based Access Control – Showing how SELinux controls access via user, role and domain type association.

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

2.6 Security Context

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

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

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

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

```
user:role:type[:level]
```

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.
level	 This field can also be know as a range and is only present if the policy supports MCS or MLS. The entry can consist of: A single security level that contains a sensitivity level and zero or more categories (e.g. s0, s1:c0, s7:c10.c15).
	 A range that consists of two security levels (a low and high) separated by a hyphen (e.g. s0 - s15:c0.c1023). These components are discussed in the <u>Security Levels</u> section.

However note that:

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

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

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

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

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

Example Process Security Context:

```
# These are process security contexts taken from a ps -Z command
# (edited for clarity) that show four processes:
LABEL
                                     PID TTY CMD
user u:unconfined r:unconfined t 2539 pts/0 bash
user u:message filter r:ext gateway t 3134 pts/0 secure server
user u:message filter r:int_gateway_t 3138 pts/0 secure_server
user u:unconfined r:unconfined t
                                     3146 pts/0 ps
# Note the bash and ps processes are running under the
# unconfined t domain, however the secure server has two instances
# running under two different domains (ext gateway t and
# int gateway t). Also note that they are using the
# message filter r role whereas bash and ps use unconfined r.
# These results were obtained by running the system in permissive
# mode (as in enforcing mode the gateway processes would not
# be shown).
```

Example Object Security Context:

```
# These are the message queue directory object security contexts
# taken from an ls -Zd command (edited for clarity):
system_u:object_r:in_queue_t /user/message_queue/in_queue
system u:object r:out queue t /user/message queue/out queue
# Note that they are instantiated with system u and object r
# These are the message queue file object security contexts
# taken from an ls -Z command (edited for clarity):
/user/message_queue/in_queue:
user_u:object_r:in_file_t
                                Message-1
user u:object r:in file t
                                 Message-2
/user/message queue/out queue:
user_u:object_r:out_file_t
                                  Message-10
user_u:object_r:out_file t
                                  Message-11
# Note that they are instantiated with user u as that was the
# SELinux user id of the process that created the files (see the
# process example above). The role remained as object r.
```

2.7 Subjects

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

Within SELinux a subject is generally an active process and has a <u>security context</u> associated with it, however a process can also be referred to as an object depending on the context in which it is being taken, for example:

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

An example is given of the above scenarios in the <u>Allowing a Process Access to an</u> <u>Object</u> section.

In SELinux subjects can be:

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

Untrusted – Everything else.

2.8 Objects

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

2.8.1 Object Classes and Permissions

Each object consists of a class identifier that defines its purpose (e.g. file, socket) along with a set of permissions⁹ that describe what services the object can handle (read, write, send etc.). When an object is instantiated it will be allocated a name (e.g. a file could be called config or a socket my_connection) and a security context (e.g. system_u:object_r:selinux_config_t) as shown in Figure 2.5.

⁸ The object class and its associated permissions are explained in the <u>Process Object Class</u> section.

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

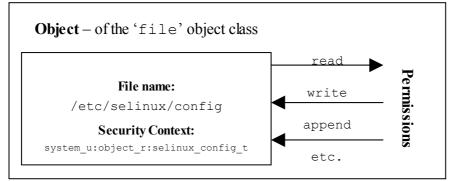


Figure 2.5: Object Class = 'file' and permissions – the policy rules would define those permissions allowed for each process that needs access to the /etc/selinux/configfile.

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

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

The object classes consist of kernel object classes (for handling files, sockets etc.) plus user space object classes used by user space object managers (for services such as the name service cache daemon (nscd), X-Windows and debus). The number of object classes and their permissions can vary depending on the features configured in the GNU / Linux release. The F-12 object classes and permissions are described in <u>Appendix A - Object Classes and Permissions</u>.

2.8.2 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.

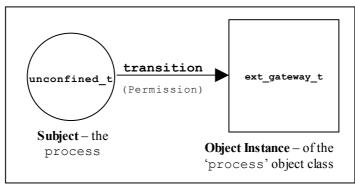
An SELinux policy contains many rules and statements, the majority of which are <u>allow rules</u> that (simply) allows processes to be given access permissions to an objects resources.

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:

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.



It should be noted that there is more to a domain transition than described above, for a more detailed explanation, see the <u>Domain Transition</u> section.

2.8.3 Labeling Objects

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

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

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

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

The SELinux policy language supports object labeling statements for file and network services that are defined in the <u>File System Labeling Statements</u> and <u>Network Labeling</u> <u>Statements</u> sections.

An overview of the process required for labeling files systems that use extended attributes (such as ext3 and ext4) is discussed in the <u>Labeling Extended Attribute</u> <u>Filesystems</u> section.

2.8.3.1 Labeling Extended Attribute Filesystems

The labeling of file systems that implement extended attributes¹¹ is supported by SELinux using:

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

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

```
ls -Z myfile
-rw-r--r- root root unconfined u:object r:admin home t:s0 myfile
```

```
getfattr -n security.selinux <file_name>
#file_name: rpmbuild
security.selinux="unconfined_u:object_r:admin_home_t:s0\000"
# Where -n security.selinux is the name of the attribute and
# rpmbuild is the file name.
# The security context (or label) for the file is:
# system_u:object_r:admin_home_t:s0
```

2.8.3.1.1 Copying and Moving Files

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

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

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

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

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

However, if the restorecond daemon is running and the <u>restorecond.conf</u> file is correctly configured, then other security contexts can be associated to the file as it is moved or copied (provided it is a valid context and specified in the <u>file contexts</u> file).

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

```
# These are the test files in the /root directory and their current security
# context:
#
-rw-r--r- root root user_u:object_r:unconfined_t copied-file
-rw-r--r- root root user_u:object_r:unconfined_t moved-file
# These are the commands used to copy / move the files:
# Standard copy file:
cp copied-file /usr/message_queue/in_queue
# Copy using -Z to set the files context:
cp -Z user_u:object_r:unconfined t copied-file /usr/message queue/in queue/copied-
file-with-Z
# Standard move file:
mv moved-file /usr/message_queue/in_queue
# The target directory (/usr/message_queue/in_queue) is label "in_queue_t".
# The results of "ls -Z" on target the directory are:
                                                                  copied-file
-rw-r--r-- root root user u:object r:in queue t
-rw-r--r- root root user_u:object_r:unconfined_t copied-file
-rw-r--r- root root user_u:object_r:unconfined_t moved-file
```

However, if the restorecond daemon is running:

```
# If the restorecond daemon is running with a restorecond.conf file entry of:
#
/usr/message_queue/in_queue/*
# AND the file_context file has an entry of:
#
/usr/message_queue/in_queue(/.*)? -- system_u:object_r:in_file_t
# Then all the entries would be set as follows when the daemon detects the files
# creation:
#
-rw-r--r-- root root user_u:object_r:in_file_t copied-file
-rw-r--r-- root root user_u:object_r:in_file_t copied-file
-rw-r--r-- root root user_u:object_r:in_file_t moved-file
# This is because the restorecond process will set the contexts defined in
# the file_contexts file to the context specified as it is created in the
# new directory.
```

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

2.8.3.2 Labeling Subjects

On a running GNU / Linux system, processes inherit the security context of the parent process. If the new process being spawned has permission to change its context, then a 'type transition' is allowed that is discussed in the <u>Domain Transition</u> section.

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

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

<u>user</u>, <u>role</u> and <u>type</u> statements.

and manage their scope:

role allow and constrain

and manage their transition:

type_transition, role_transition and range_transition

One point to note is that the current Reference Policy does not support role transitions / changes as these are 'constrained' by the policy. To change to a different role, the newrole(1) command needs to be used (although there are ways around this as described in the Implementing a Constraint section of the Sample Policy Source volume).

2.8.4 Object Reuse

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

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

2.9 Domain and Object Transitions

This section discusses the <u>type transition statement</u> that is used for:

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

These transitions can also be achieved using the <u>libselinux API</u> functions, however they are beyond the scope of this Notebook as is dynamically changing a processes security context using the dyntransition permission.

2.9.1 Domain Transition

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

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

type_transition unconfined_t secure_services_exec_t : process ext_gateway_t;

2. SELinux-aware applications can specify the domain of the new process using the <u>libselinux API</u> call setexeccon. To achieve this the SELinux-aware application must also have the setexec permission by:

allow crond_t self : process setexec;

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

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

The following is a type_transition statement taken from the example loadable module message filter ext_gateway.conf (described in volume 2) that will be used to explain the transition process¹³:

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

However, as stated above to be able to *transition* to the *ext_gateway_t* domain, the following minimum permissions must be granted in the policy using <u>allow rules</u>, where (note that the bullet numbers correspond to the numbers shown in Figure 2.7):

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

allow unconfined_t ext_gateway_t : process transition;

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

allow unconfined_t secure_services_exec_t : file { execute read getattr };

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

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

allow ext_gateway_t secure_services_exec_t : file entrypoint;

These are shown in Figure 2.7 where unconfined_t forks a child process, that then exec's the new program into a new domain called $ext_gateway_t$. Note that because the type_transition statement is being used, the transition is automatically carried out by the SELinux enabled kernel.

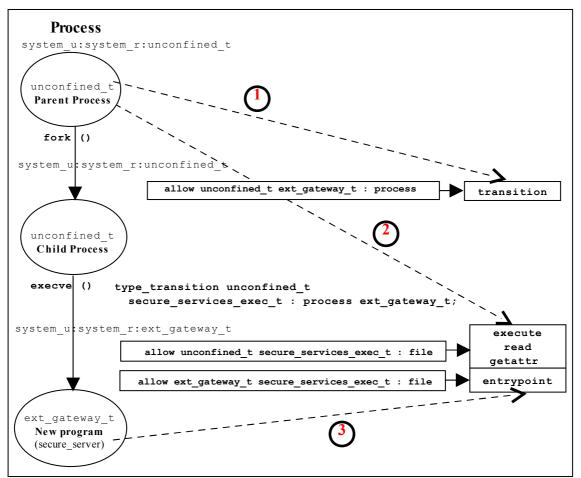


Figure 2.7: Domain Transition – Where the secure server is executed within the unconfined t domain and then transitioned to the ext gateway t domain.

2.9.1.1 Type Enforcement Rules

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

type_transition unconfined_t secure_services_exec_t : process ext_gateway_t;

and the int gateway t statement would be:

type_transition unconfined_t secure_services_exec_t : process int_gateway_t;

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

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

This happened because the type enforcement rules will only handle a single 'default' type for a given source and target (see the <u>Type Enforcement Rules</u> section). In the above case there were two type_transition statements with the same source and target, but different target 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, that break the type enforcement rule.

It was decided to resolve this by:

1. Keeping the type_transition rule for the 'default' type of ext_gateway_t and allow the secure server process to be exec'ed from unconfined_t as shown in Figure 2.7, by simply running the command from the prompt as follows:

```
# Run the external gateway 'secure server' application on port 9999 and
# let the policy transition the process to the ext_gateway_t domain:
secure_server 99999
```

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

```
# Run the internal gateway 'secure server' application on port 1111 and
# use runcon to transition the process to the int_gateway_t domain:
runcon -t int_gateway_t -r message_filter_r secure_server 1111
# Note - The role is required as a role transition is also defined in the
# policy.
```

The runcon command makes use of a number of <u>libselinux API</u> functions to check the current context and set up the new context (for example getfilecon is used to check the executable files context and setexeccon is used to ensure the setexec permission is allowed). If the all contexts are correct, then the execvp(2) system call is executed that exec's the secure_server application with the argument of '1111' in the int_gateway_t domain with the message_filter_r role. The runcon source can be found in the coreutils 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 (see volume 2) 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 <u>template Macro</u> section.

It was decided to use runcon as it demonstrates the command usage better than reading the man pages.

2.9.2 Object Transition

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

1. Using a <u>type_transition statement</u> to perform an object transition (relabel) for programs that are not SELinux-aware. This is the most common method and would be in the form of the following statement:

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

2. Using a <u>type change statement</u> to perform an object transition for programs that are SELinux-aware.

type_change sysadm_t server_ptynode : chr_file sysadm_devpts_t;

The <u>libselinux API</u> call security_compute_relabel would be used to compute the new context.

The following details an object transition used in the ext_gateway.conf loadable module (shown in volume 2) where by default, files would be labeled in_queue_t when created by the gateway application as this is the label attached to the parent directory as shown:

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

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

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

However, as stated above to be able to relabel the file, the following minimum permissions need to be granted in the policy using <u>allow rules</u>, where:

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

allow ext_gateway_t in_queue_t : dir { write search add_name };

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

allow ext_gateway_t in_file_t : file { write create getattr };

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

type transition ext gateway t in queue t:file in file t;

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

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

2.10 Multi-Level Security and Multi-Category Security

As stated in the <u>Mandatory Access Control (MAC</u>) 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.

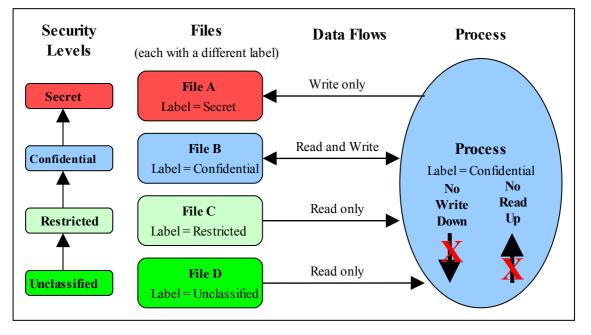


Figure 2.8: Security Levels and Data Flows – *This shows how the process can only 'Read Down' and 'Write Up' within an MLS enabled system.*

To achieve this level of control, the MLS extensions to SELinux make use of constraints similar to those described in the type enforcement <u>Constraints</u> section, except that the statement is called <u>mlsconstrain</u>.

However, as always life is not so simple as:

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

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

2.10.1 Security Levels

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

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

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

 Table 2-1: Level, Label, Category or Compartment – this table shows the meanings

 depending on the context being discussed.

The format used in the policy language statements is fully described in the <u>MLS</u> <u>Statements</u> section, however a brief overview follows.

2.10.1.1 MLS / MCS Range Format

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

		-	<pre>sensitivity [:category,]</pre>
•	level	•	l level
		range	e

Where:

sensitivity	Sensitivity levels are hierarchical with (traditionally) s0 being the lowest. These values are defined using the <u>sensitivity</u> language statement. To define their hierarchy, the <u>dominance</u> statement is used.	
	For MLS systems the highest sensitivity is the last one defined in the dominance statement (low to high). Traditionally the maximum for MLS systems is s15 (although the maximum value for the Reference Policy is a compile time option).	
	For MCS systems there is only one sensitivity defined, and that is $s0$.	
category	Categories are optional (i.e. there can be zero or more categories) and they form unordered and unrelated lists of 'compartments'. These values are defined using the <u>category</u> statement, where for example $c0.c3$ represents a range ($c0 c1 c3$) and $c0$, $c3$, $c7$ represent an unordered list. Traditionally the values are between $c0$ and $c255$ (although the maximum value for the Reference Policy is a compile time option).	
level	The level is a combination of the sensitivity and category values that form the actual security level. These values are defined using the <u>level</u> statement.	

2.10.1.2 Translating Levels

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

2.10.2 Managing Security Levels via Dominance Rules

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

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

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

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

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

To illustrate the usage of these rules, <u>Table 2-2</u> 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 <u>MLS dominance statement</u> 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 <u>mlsconstrain statement</u> as illustrated in <u>Figure 2.9</u>.

	Category →	с0	c1	c2	c3	c4	с5	сб	с7
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
^	^		↑ File Labels ↑						
Sensitivity	Security Level	A process running with a range of s0 - s3:c1.c5 has access to the files			the files				
	(sensitivity:category)	within the grey boxed area.							
	aka: classification								

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

Using Figure 2.9:

1. To allow write-up, the source level (11) must be **dominated by** the target level (12):

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 (11) must **dominate** the target level (12):

Source level = s2:c1.c4

Target level = s0:c3

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

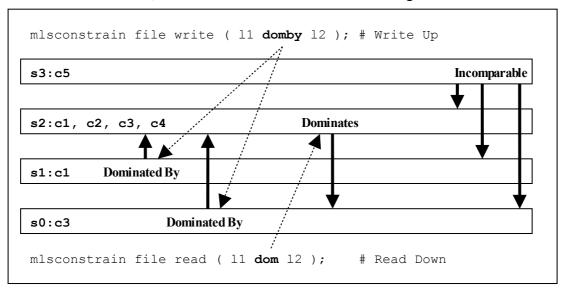


Figure 2.9: Showing the mlsconstrain Statements controlling Read Down & Write Up – This ties in with <u>Table 2-2</u> that shows a process running with a security range of s0 - s3:c1.c5.

However in the real world the SELinux MLS Reference Policy does not allow the write-up unless the process has a special privilege (by having the domain type added to an attribute), although it does allow the read-down. The default is to use 11 eq 12 (i.e. the levels are equal). The reference policy MLS source file (policy/mls) shows these mlsconstrain statements.

2.10.3 MLS Labeled Network and Database Support

Networking for MLS is supported via the NetLabel CIPSO (commercial IP security option) service as discussed in the <u>SELinux Networking Support</u> section.

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

2.10.4 Common Criteria Certification

While the <u>Common Criteria</u> 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 (<u>LSPP</u>) This describes how systems that implement security labels (i.e. MLS) should function.
- Controlled Access Protection Profile (<u>CAPP</u>) This describes how systems that implement DAC should function.

An interesting point:

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

2.11 Types of SELinux Policy

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

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

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

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

2.11.1 Example Policy

The Example policy is the name used to describe the original SELinux policy source used to build a <u>monolithic</u>¹⁴ policy produced by the NSA and is now superseded by the Reference Policy.

2.11.2 Reference Policy

Note that this section only gives an introduction to the reference policy, the installation, configuration and building of a policy using the source code is contained in the Reference Policy.

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

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

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

Policy'. For example, the F-12 distribution is based on a specific build of the standard Reference Policy that is then modified and distributed by Red Hat as an RPM. The release numbers will vary however this Notebook uses:

selinux-policy-3.6.32-103.fc12.src.rpm

For information, the policy RPMs installed on the authors test machine for F-12 are as follows (the contents are explained in the main <u>Reference Policy</u> section):

```
selinux-policy-3.6.32-103.fc12.noarch
selinux-policy-doc-3.6.32-103.fc12.noarch
selinux-policy-minimum-3.6.32-103.fc12.noarch
selinux-policy-mls-3.6.32-103.fc12.noarch
selinux-policy-targeted-3.6.32-103.fc12.noarch
```

2.11.2.1 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 normally 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 the policy (e.g. if name is targeted, then а SELINUXTYPE=targeted be entry would in the /etc/selinux/config file).

This is how the reference policies distributed with F-12 shown above 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 (this targeted version also supports the older 'strict' version). 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 MLS labels as discussed in the <u>Multi-Level Security and Multi-Category Security</u> section.

The NAME and TYPE entries are defined in the reference policy build.conf file that is described in the <u>Source Configuration Files</u> section.

2.11.3 Custom Policy

This generally refers to a policy source that is either:

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

2.11.4 Monolithic Policy

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

An example monolithic policy is the NSAs original <u>Example Policy</u>. A simple monolithic policy is shown in the <u>Building the Monolithic Policy</u> section and <u>Table 4-1</u> shows the order of language statements that can be in a source file.

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

The Reference Policy supports the building of monolithic policies.

In some cases the policy binary file (see the <u>Binary Policy</u> section) is also called a monolithic policy.

2.11.5 Loadable Module Policy

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

There are number of parts that form the infrastructure:

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

Figure 2.2 shows these components along the top of the diagram. The files contained in the policy store are detailed in the <u>Policy Store Configuration Files</u> section.

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

2.11.5.1 Optional Policy

The loadable module policy infrastructure supports an <u>optional policy statement</u> that allows policy rules to be defined but only enabled in the binary policy once the conditions have been satisfied. The example loadable modules shown in the Building a Basic Policy section of volume 2 use this feature.

2.11.6 Conditional Policy

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

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

setsebool -P ext_gateway_audit=false

The conditional policy language statements are the <u>bool Statement</u> that defines the boolean flag identifier and its initial status, and the <u>if Statement</u> that allows certain rules to be executed depending on the state of the boolean value or values.

2.11.7 Binary Policy

The binary policy is the policy file that is loaded into the kernel and is always located at /etc/selinux/<policy_name>/policy and is called policy.<version>. Where <policy_name> is the policy name specified in the SELinux configuration file /etc/selinux/config and <version> is the SELinux policy version supported by the kernel and SELinux tools.

The binary policy can be built from source files supplied by the <u>Example Policy</u>, the Reference Policy or custom built source files as described in the Building a Basic Policy section of volume 2.

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.24 (as version 24 is supported by F-12):

```
/etc/selinux/targeted/policy/policy.24
```

2.11.8 Policy Versions

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

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

```
SELinux status:enabledSELinuxfs mount:/selinuxCurrent mode:enforcingMode from config file:permissivePolicy version:24Policy from config file:modular-test
```

The F-12 policy version is '24' with <u>Table 2-3</u> describing the different versions. There is also another version that applies to the modular policy, however the main policy database version is the one that is generally quoted (some SELinux utilities (e.g. apol) give both version numbers).

policy db Version	modular db Version	Description
15	4	The base version when SELinux was merged into the kernel.
16	-	Added Conditional Policy support (the bool feature).
17	-	Added support for IPv6.
18	-	Added Netlink support.
19	5	Added MLS support, plus the <u>validatetrans</u> <u>Statement</u> .
20	-	Reduced the size of the access vector table.
21	6	Added support for the MLS <u>range_transition</u> <u>Statement</u> .
22	7	Added <u>policy capabilities</u> . Allows various kernel options to be enabled as described in the <u>SELinux Filesystem</u> section.
23	8	Added support for the <u>permissive Statement</u> . This allows a module to run in permissive mode while the others are still confined (instead of the all or nothing set by the SELINUX entry in the <u>/etc/selinux/config</u> file).
24	9 / 10	Add support for the typebounds Statement. This was added to support a hierarchical relationship between two domains in multi-threaded web servers as described in " <u>A</u> secure web application platform powered by SELinux" [Ref. 20].

Table 2-3: Policy version descriptions

2.12 SELinux Permissive and Enforcing Modes

SELinux has three major modes of operation:

Enforcing – SELinux is enforcing the loaded policy.

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

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

These flags are set in the /etc/selinux/config file as described in the <u>Global</u> <u>Configuration Files</u> section.

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

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

The <u>sestatus</u>(8) command will show the current policy mode in its output as follows:

```
SELinux status:enabledSELinuxfs mount:/selinuxCurrent mode:permissiveMode from config file:enforcingPolicy version:24Policy from config file:modular-test
```

2.13 Audit Logs

For SELinux there are two main types of audit event:

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

The audit and event messages can be stored in one of two places (in F-12 anyway):

1. The system log located at /var/log/messages that contains the syslog boot and runtime events. The AVC messages logged here are those generated by SELinux before the audit daemon has been loaded, as F-12 uses the audit framework (auditd) as standard. However, some SELinux-aware audit messages are logged here as well¹⁵. Note that the detailed SELinux kernel boot events are logged in the /var/log/dmesg file.

2. The audit log located at /var/log/audit/audit.log. Audit events that take place after the audit daemon has been loaded are in this log file as are some SELinux system messages. The AVC audit messages of interest are those starting with type=AVC and are described in the <u>AVC Audit Events</u> section.

Note that SE-PostgreSQL does not send its AVC events to the standard audit.log, but to its own log file as discussed in the <u>SELinux PostgreSQL Support</u> section.

2.13.1 SELinux-aware Application Events

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

```
# Example audit.log entries for adding a new SELinux user with:
#
#
        semanage user -a -R staff r -P STAFF test u
#
# Note that the audit messages appear in the following sequence:
#
# 1) That USER AVC has received a policy reload message.
# 2) A notice that the policy is changing by MAC POLICY LOAD.
#
     Note that this event is followed by another audit message
#
     with type=SYSCALL with further information. These two
#
     events are tied by having the same serial number in the
     msg=audit(time:serial number) field.
# 3) That semanage carried out a USER ROLE CHANGE.
type=USER AVC msg=audit(1243855640.568:29): user pid=1543 uid=81 auid=4294967295
ses=4294967295 subj=system_u:system_r:system_dbusd_t:s0-s0:c0.c1023 msg='avc:
received policyload notice (seqno=3) : exe="?" (sauid=81, hostname=?, addr=?,
terminal=?)'
type=MAC POLICY LOAD msg=audit(1243855634.660:30): policy loaded auid=0 ses=1
type=SYSCALL msg=audit(1243855634.660:30): arch=40000003 syscall=4 success=yes
exit=3480819 a0=4 a1=b7c74000 a2=351cf3 a3=bfbd9dc8 items=0 ppid=2731 pid=2732
auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=1
comm="load policy" exe="/usr/sbin/load policy"
subj=unconfined u:unconfined r:load policy t:s0-s0:c0.c1023 key=(null)
type=USER_ROLE_CHANGE msg=audit(1243855646.618:31): user pid=2731 uid=0 auid=0
ses=1 subj=unconfined_u:unconfined_r:semanage_t:s0-s0:c0.c1023 msg='op=add SELinux
user record acct="test u" old-seuser=? old-role=? old-range=? new-seuser=test u
new-role=staff r new-range=? exe=/usr/sbin/semanage (hostname=?, addr=?,
terminal=pts/0 res=success)
# Example audit.log entries for setting enforcing mode by:
#
```

#
setenforce 1
#
Note that the audit messages appear in the following sequence:
#

¹⁵ For example if the iptables are loaded and there are SECMARK security contexts present, BUT the contexts are invalid (i.e. not in the policy), then the event is logged in the messages log and nothing will appear in the audit log.

type=MAC_STATUS msg=audit(1243856597.296:32): enforcing=1 old_enforcing=0 auid=0
ses=1
type=USER_AVC msg=audit(1243856597.312:33): user pid=1543 uid=81 auid=4294967295
ses=4294967295 subj=system_u:system_r:system_dbusd_t:s0-s0:c0.c1023 msg='avc:
received setenforce notice (enforcing=1) : exe="?" (sauid=81, hostname=?, addr=?,
terminal=?)'
type=SYSCALL msg=audit(1243856597.296:32): arch=40000003 syscall=4 success=yes
exit=1 a0=3 a1=bf95b554 a2=1 a3=bf95b554 items=0 ppid=2643 pid=2761 auid=0 uid=0
gid=0 euid=0 suid=0 fsuid=0 egid=0 fsgid=0 fsgid=0 ty=pts0 ses=1 comm="setenforce"
exe="/usr/sbin/setenforce" subj=unconfined u:unconfined r:unconfined t:s0-

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

s0:c0.c1023 key=(null)

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

```
# This is an example entry of an AVC denial and corresponding
# SYSCALL from the /var/log/messages file that were captured
# before the audit daemon was loaded:
May 26 12:34:16 localhost kernel: type=1400 audit(1243337656.638:1358): avc:
denied { read } for pid=3033 comm="rsyslogd" path="/proc/kmsg" dev=proc
ino=4026531848 scontext=system_u:system_r:syslogd_t:s0
tcontext=system_u:object_r:unlabeled_t:s0 tclass=file
May 26 12:34:16 localhost kernel: type=1300 audit(1243337656.638:1358):
arch=40000003 syscall=3 success=yes exit=230 a0=7 a1=11f7e0 a2=fff a3=11f7e0
items=0 ppid=1 pid=3033 auid=4294967295 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0
sgid=0 fsgid=0 tty=(none) ses=4294967295 comm="rsyslogd" exe="/sbin/rsyslogd"
subj=system_u:system_r:syslogd_t:s0 key=(null)
```

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

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

2.13.2 AVC Audit Events

<u>Table 2-4</u> shows the general format of AVC audit message within the /var/log/audit/audit.log file. These appear when access has been denied or an audit event has been specifically requested.

Keyword	Description
type	For SELinux AVC events this can be:
	type=AVC - for kernel events
	type=USER_AVC - for user-space object manager events
	Note that once the AVC event has been logged, another event with type=SYSCALL will follow that contains further information regarding the event.
	The AVC event can always be tied to the relevant SYSCALL event as they have the same serial_number in the msg=audit(time:serial_number) field as shown in the following example:
	<pre>type=AVC msg=audit(1243332701.744:101): avc: denied { getattr } for pid=2714 comm="ls" path="/usr/lib/locale/locale-archive" dev=dm-0 ino=353593 scontext=system_u:object_r:unlabeled_t:s0 tcontext=system_u:object_r:locale_t:s0 tclass=file</pre>
	<pre>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	This will be either denied when access has been denied or granted when the <u>auditallow rule</u> has been executed by the AVC system.
	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.
pid	If a task, then log the process id (pid) and the name of the
comm	executable file (comm).
key	If an IPC event then log the identifier.
capability	If a Capability event then log the identifier.
path	If a File System event then log the relevant information. Note that the
name	name field may not always be present.
dev	

Keyword	Description
ino	
laddr	If a Socket event then log the Source / Destination addresses and
lport	ports for IP4 or IP6 sockets (AF_INET).
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
src	ports with the network interface for IP4 or IP6 networks (AF_INET).
daddr	
dest	-
netif	
sauid	IPSec security association identifiers
hostname	
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 2-4: AVC Audit Message Description – The keywords in **bold** are in all AVC audit messages, the others depend on the type of event being audited.

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

```
# This is an example denied message note that there are two
# type=AVC calls, but only one corresponding type=SYSCALL entry.
type=AVC msg=audit(1242575005.122:101): avc: denied { rename } for pid=2508
comm="canberra-gtk-p1" 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-p1" 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
tty=(none) ses=1 comm="canberra-gtk-p1" 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=user_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=user_u:unconfined_r:x_select_paste_t tcontext=user_u:unconfined_r:unconfined_r:x_select_paste_t seuid=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=user_u:unconfined_r:unconfined_t tcontext=user_u:unconfined_r:move_file_t
tclass=process
```

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

2.14 Polyinstantiation

GNU / Linux supports the polyinstantiation of directories that can be utilised by SELinux via the Pluggable Authentication Module (PAM) that is explained in the next section. The "Polyinstantiation of directories in an SELinux system" [Ref. 5] also gives a more detailed overview of the subject.

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. 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).
- 2. The polyinstantiation of X-windows selections and properties is a function of the XSELinux Object Manager and the supporting XACE service. These two services are effectively X-windows extensions that can be disabled if required.
- 3. SELinux has a <u>type_member</u> rule that supports polyinstantiated objects. An example using X-windows selections is shown in the Experimenting with X-Windows section of volume 2.

2.14.1 Polyinstantiated Objects

Polyinstantiation is supported by SELinux using the <u>type_member Statement</u>. This statement is not limited to specific object classes, however GNU / Linux currently only supports dir, x_selection and x_property objects.

The following libselinux API functions support polyinstantiation as detailed in Appendix C – API Summary for libselinux:

```
avc_compute_member
security_compute_member
security_compute_member raw
```

2.14.2 Polyinstantiation support in PAM

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

The default installation of F-12 does not enable polyinstantiated directories, therefore this section will show the configuration required to enable the feature and some <u>examples</u>.

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-12 already has these entries configured, with an example /etc/pam.d/gdm file being:

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

2. Entries added to the /etc/security/namespace.conf file that defines the directories to be polyinstantiated by PAM (and other services that may need to use the namespace service). The entries are explained in the <u>namespace.conf Configuration File</u> section, with the default entries in F-12 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-12 configuration also includes an /etc/security/namespace.init script that is used to initialise the namespace every time a new directory instance is set-up. This script receives four parameters: the polyinstantiated directory path, the instance directory path, a flag to indicate if new instance, and the user name. If a new instance is being set up, the directory permissions are set and the restorecon(8) command is run to set the correct file contexts.

2.14.2.1 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 \$USER and \$HOME will be replaced by the user name and home directory respectively.
instance_prefix	A string prefix used to build the pathname for the polyinstantiated directory. The optional strings \$USER and \$HOME will be replaced by the user name and home directory respectively.
method	This is used to determine the method of polyinstantiation with valid entries being:
	user - Polyinstantiation is based on user name.
	level - Polyinstantiation is based on the user name and MLS level.
	context - Polyinstantiation is based on the user name and security context.
	Note that level and context are only valid for SELinux enabled systems.
list_of_uids	A comma separated list of user names that will not have polyinstantiated directories. If blank, then all users are polyinstantiated. If the list is preceded with an '~' character, then only the users in the list will have polyinstantiated directories.
	There are a number of optional flags available that are described in the namespace.conf(5) man page.

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

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

```
# The directories will contain the user name as a part of
# the polyinstantiated directory name as follows:
# /tmp
/tmp/tmp-inst/rch
# /var/tmp:
/var/tmp/tmp-inst/rch
# $HOME
/home/rch/rch.inst/rch
```

Example 2 - method=context:

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

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

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

```
# The directories will contain the security context and
# user name as a part of the polyinstantiated directory
# name as follows:
# /tmp
/tmp/tmp-inst/user_u:unconfined_r:unconfined_t_rch
# /var/tmp:
/var/tmp/tmp-inst/user_u:unconfined_r:unconfined_t_rch
# $HOME
/home/rch/rch.inst/user_u:unconfined_r:unconfined_t_rch
```

2.14.3 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 <u>SELinux X-windows Support</u> section.

2.14.4 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, however the Experimenting with X-Windows section in volume 2 shows an x_selection example.

2.15 PAM Login Process

Applications used to provide login services (such as gdm and ssh) in F-12 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-12 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:

service	The service name such as gdm and login reflecting the login
	application. If there is a /etc/pam.d directory, then this is
	the name of a configuration file name under this directory.
	Alternatively, a configuration file called /etc/pam.conf

	can be used. F-12 uses the /etc/pam.d configuration.
type	These are the management groups used by PAM with valid entries being: account, auth, password and session that correspond to the descriptions given above. Where there are multiple entries of the same 'type', the order they appear could be significant.
control	This entry states how the module should behave when the requested task fails. There can be two formats: a single keyword such as required, optional, and include; 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 pam.conf man pages for the gory details.
	the pain, contributing pages for the gory details.
module-path	Either the full path name of the module or its location relative to /lib/security (but does depend on the system architecture).
arguments	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-12 uses the /etc/pam.d directory configuration (in this case qdm for the Gnome login service).

# SERVICE =	= file name	guration rule entry. (gdm) PATH ARGUMENTS
#%PAM-1.0		
auth [s	success=done	<pre>ignore=ignore default=bad] pam selinux permit.so</pre>
# auth	required	pam succeed if.so user != root quiet
auth	required	pam env.so
auth	substack	system-auth
auth	optional	pam_gnome_keyring.so
account	required	pam_nologin.so
account	include	system-auth
password	include	system-auth
session	required	pam_selinux.so close
session	required	pam_loginuid.so
session	optional	pam_console.so
session	required	pam_selinux.so open
session	optional	pam_keyinit.so force revoke
session	required	pam_namespace.so
session	optional	pam_gnome_keyring.so auto_start
session	include	system-auth

The core services are provided by PAM, however other library modules can be written to manage specific services such as support for SELinux. The SELinux PAM modules use the libselinux API to obtain its configuration information and the three SELinux PAM entries highlighted in the above configuration file perform the following functions: **pam_selinux_permit.so** - Allows pre-defined users the ability to logon without a password provided that SELinux is in enforcing mode (see the /etc/security/sepermit.conffilesection).

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 <u>seusers configuration file</u> section.

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

2.16 Linux Security Module and SELinux

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

- 1. How the LSM and SELinux modules work together.
- 2. The major SELinux internal services.
- 3. The fork system call and exec are followed through as an example to tie in with the transition process covered in the <u>Domain Transition</u> section.
- 4. The SELinux filesystem / selinux.
- 5. The boot sequences that are relevant to SELinux.

2.16.1 The LSM Module

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

The basic idea behind the LSM is to:

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

It should be noted that the LSM does not provide any security services itself, only the hooks and structures for supporting 3rd party modules. If no 3rd party module is loaded,

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	

the capabilities module becomes the default module thus allowing standard DAC access control.

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

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

Name	Function	
capability.c	Some capability functions were in various kernel modules have been consolidated into these source files. These are now (from Kernel 2.6.27) always linked into the kernel. This means the dummy.c security module (mentioned in [Ref. 6]) is no longer required.	
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 /selinux that is defined in the selinux/selinuxfs.c source file.	
root_plug.c	This is a sample 3 rd party module and therefore not used.	
security.c	Contains the LSM framework initialisation services that will set up the hooks described in security. h and those in the capability source files. It also provides functions to initialise 3^{rd} party modules.	

Table 2-6: The core LSM source modules.

2.16.2 The SELinux Module

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

The major kernel SELinux source files (relative to ./kernel-2.6.27/security/selinux) that form the SELinux security module are shown in <u>Table 2-7</u>. The diagrams shown in <u>Figure 2.2</u> and <u>Figure 2.12</u> can be used to see how some of these kernel source modules fit together.

Name	Function		
avc.c	Access Vector Cache functions and structures. The function calls are for the kernel services, however they have been ported to form the libselinux userspace library detailed in <u>Appendix C – API Summary for</u> <u>libselinux</u> .		
exports.c	Exported SELinux services for SECMARK (as there is SELinux specific code in the netfilter source tree).		
hooks.c	Contains all the SELinux functions that are called by the kernel resources via the security_ops function table (they form the kernel resource object managers). There are also support functions for managing process exec's, managing SID allocation and removal, interfacing into the AVC and Security Server.		
netif.c	These manage the mapping between labels and SIDs for the net* language		
netnode.c	statements when they are declared in the active policy.		
netport.c			
netlabel.c	The interface between NetLabel services and SELinux.		
netlink.c	Manages the notification of policy updates to resources including userspace		
nlmsgtab.c	applications via libselinux.		
selinuxfs.c	The selinuxfs pseudo filesystem (/selinux) that exports the security policy to userspace services via libselinux. The services exported are shown in the <u>SELinux Filesystem</u> section.		
xfrm.c	Contains the IPSec XFRM hooks for SELinux.		
include/flask.h	This contains all the kernel security classes and initial SIDs. Note that the Reference Policy source (policy/flask directory) contains a list of all the kernel and userspace security classes and permissions.		
ss/avtab.c	AVC table functions for inserting / deleting entries.		
ss/conditional.c	Support boolean statement functions and implements a conditional AV table to hold entries.		
ss/avtab.c	AVC table functions for inserting / deleting entries.		
ss/ebitmap.c	Bitmaps to represent sets of values, such as types, roles, categories, and classes.		
ss/hashtab.c	Hash table.		
ss/mls.c	Functions to support MLS.		
ss/policydb.c	Defines the structure of the policy database. See the " <u>SELinux Policy</u> <u>Module Primer</u> " [Ref. 4] article for details on the structure.		
ss/services.c	This contains the supporting services for kernel hooks defined in hooks.c, the AVC and the Security Server. For example the security_transition_sid that computes the SID for a new subject / object shown in Figure 2.12.		
ss/sidtab.c	The SID table contains the security context indexed by its SID value.		
ss/symtab.c	Maintains associations between symbol strings and their values.		

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

2.16.2.1 Fork System Call Walk-thorough

This section walks through the the fork system call shown in Figure 2.7 starting at the kernel hooks that link to the SELinux services. The way the SELinux hooks are initialised into the LSM security_ops and secondary_ops function tables are also described.

Using Figure 2.10, the major steps to check whether the unconfined_t process has permission to use the fork permission are:

- 4. The kernel/fork.c has a hook that links it to the LSM function security_task_create() that is called to check access permissions.
- 5. 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.
- 6. The selinux_task_create() function will first call the capabilities code in capability.c via the secondary_ops function table to check the DAC permission.
- 7. This is simply a return 0;, therefore no error would be generated.
- 8. The selinux_task_create() function will then check whether the task has permission via the task_has_perm(current_process, current_process, PROCESS_FORK) function.
- 9. This will result in a call to the AVC via the avc_has_perm() function in avc.c that checks whether the permission has been granted or not. First (via avc_has_perm_noaudit()) the cache is checked to for an entry. Assuming that there is no entry in the AVC, then the security_compute_av() function in services.c is called.
- 10. The security_compute_av() function will search the SID table for source and target entries, and if found will then call the context_struct_compute_av() function.

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

- a) Initialise the AV structure so that it is clear.
- b) Check the object class and permissions are correct. It also checks the status of the allow_unknown flag (see the <u>SELinux Filesystem</u>, <u>/etc/selinux/semanage.conf file</u> and <u>Reference Policy Build</u> <u>Options build.conf UNK_PERMS</u> sections).
- c) Checks if there are any type enforcement rules (ALLOW, AUDIT_ALLOW, AUDIT_DENY).
- d) Check whether any conditional statements are involved via the cond_compute_av() function in conditional.c.
- e) Remove permissions that are defined in any constraint via the constraint_expr_eval() function call (in services.c). This function will also check any MLS constraints.

- f) Finally context_struct_compute_av() checks if a process transition is being requested (it is not). If it were, then the TRANSITION and DYNTRANSITION permissions are checked and whether the role is changing.
- 11. 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.
- 12. The End.

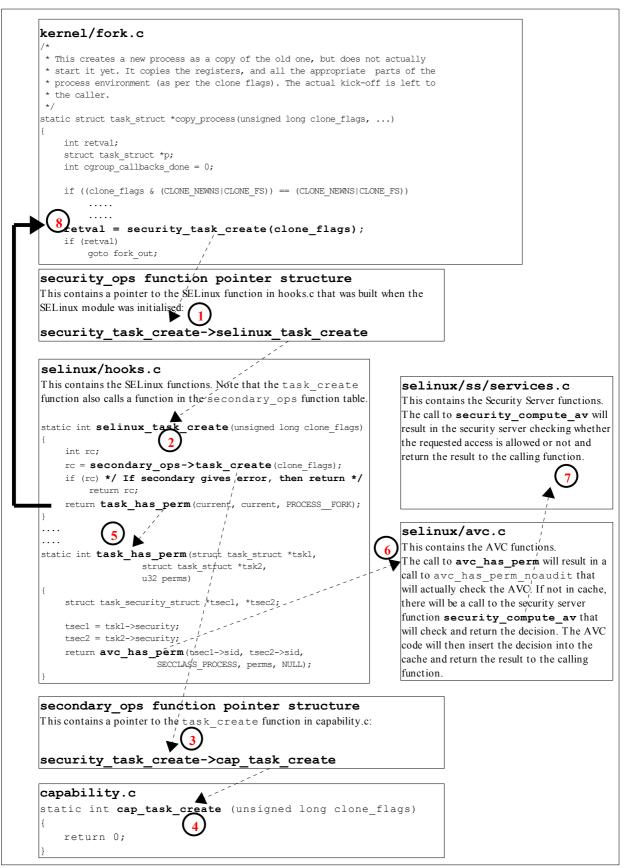


Figure 2.10: Hooks for the fork system call - This describes the steps required to check access permissions for Object Class 'process' and permission 'fork'.

2.16.2.2 Process Transition Walk-thorough

This section walks through the <code>execve()</code> and checking whether a process transition to the <code>ext_gateway_t</code> domain is allowed, and if so obtain a new SID for the context (<code>user_u:message_filter_r:ext_gateway_t</code>) as shown in Figure 2.7.

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

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

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

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

Therefore starting at the do_execve() function and using Figure 2.11, the following major steps will be carried out to check whether the unconfined_t process has permission to transition the secure_server executable to the ext_gateway_t domain:

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

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

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

- 2. Via the prepare_binprm() function call the UID and GIDs are checked and a call issued to security bprm set() that will carry out the following:
 - a) The selinux_bprm_set_security() function will call the secondary_ops->bprm_set_security function in capability.c, that is effectively a no-op.
 - b) The bsec->set flag will be checked and if 1 will return as this function can be called multiple times during the exec process.
 - c) The target SID is checked to see whether a transition is required (in this case it is), therefore a call will be made to the security_transition_sid() function in services.c. This function will compute the SID for a new subject or object (subject in this case) via the security_compute_sid() function that will (assuming there are no errors):
 - i. Search the SID table for the source and target SIDs.
 - ii. Sets the SELinux user identity.
 - iii. Set the source role and type.
 - iv. Checks that a type_transition rule exists in the AV table and / or the conditional AV table (see Figure 2.12).
 - v. If a type_transition, then also check for a role_transition (there is a role change in the ext_gateway.conf policy module), set the role.
 - vi. Check if any MLS attributes by calling mls_compute_sid() in mls.c. It also checks whether MLS is enabled or not, if so sets up MLS contexts.
 - vii. Check whether the contexts are valid by calling compute_sid_handle_invalid_context() that will also log an audit message if the context is invalid.
 - viii. Finally obtains a SID for the new context by calling sidtab_context_to_sid() in sidtab.c that will search the SID table (see Figure 2.12) and insert a new entry if okay or log a kernel event if invalid.
 - d) The selinux_bprm_set_security() function will then continue by checking via the avc_has_perm() function (in avc.c) whether the file_execute_no_trans is set (in this case it is not), therefore the process_transition and file_entrypoint permissions are checked (in this case they are), therefore the new SID is set, the bsec->set flag is set to 1 so that this part of the function is not executed again for this exec, finally control is passed back to the do execve function:

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

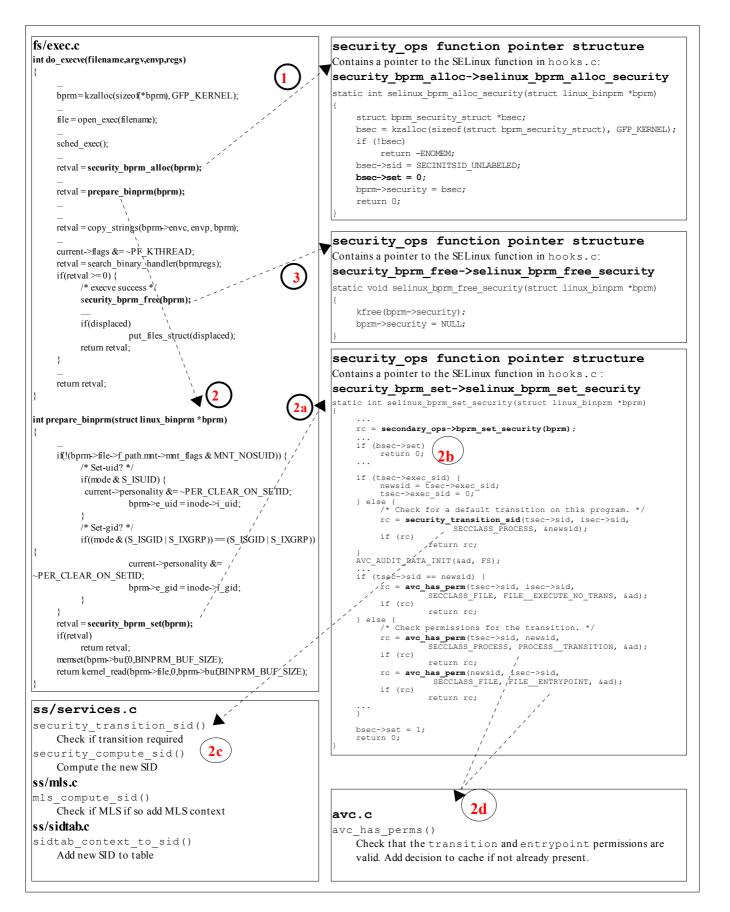


Figure 2.11: Process Transition - This shows the major steps required to check if a transition is allowed from the unconfined_t domain to the ext_gateway_t domain.

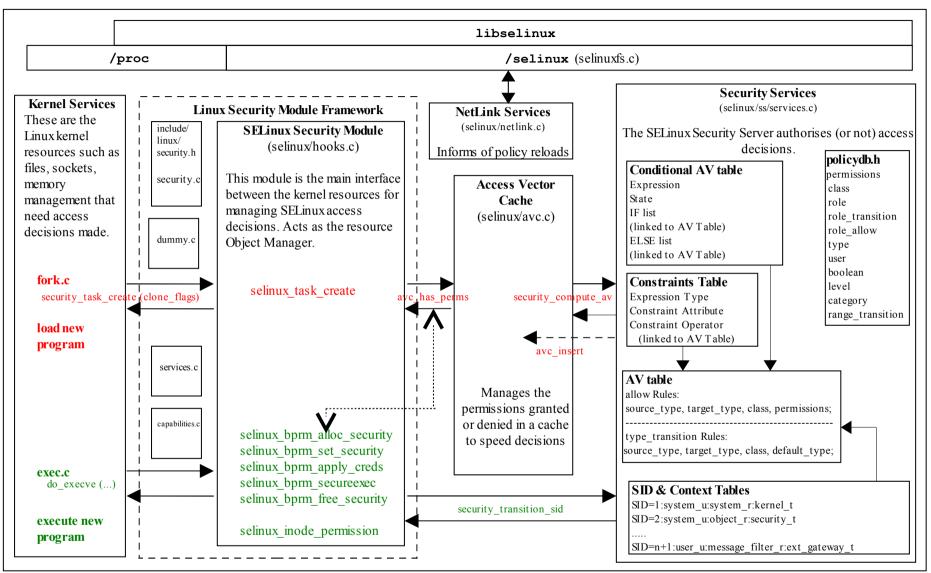


Figure 2.12: The Main LSM / SELinux Modules – The fork and exec functions link to Figure 2.7 where the transition process is described.

2.16.2.3 SELinux Filesystem

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

Directory / File Name	Permissions	Comments
/selinux	Directory	This is the root directory where the SELinux kernel exports relevant information regarding its configuration and active policy for use by the libselinux library (that is used by user space object managers and SELinux-aware applications).
access	-rw-rw-rw-	Compute access decision interface.
checkreqprot	-rw-rr	Check requested protection, not kernel-applied one.
commit_pending_bools	w	Commit new boolean values to the kernel policy.
compat_net	-rw-rr	Whether SECMARK is enabled or not: 0 = SECMARK enabled (F-12 default). 1 = SECMARK disabled At some stage (kernel 2.6.30 and greater), the older compat_net network controls listed in the Network Labeling Statements section will be removed and replaced by SECMARK.
context	-rw-rw-rw-	Validate context interface.
create	-rw-rw-rw-	Compute create labeling decision interface.
deny_unknown	-rrr	These two files export unknown deny and
reject_unknown	-rr	reject handling status to user space. This is taken from the handle-unknown parameter set ¹⁷ in the <u>/etc/selinux/semanage.conf file</u> and are set as follows: 0:0 = allow, 1:0 = deny and 1:1 = reject.
disable	w	Disable SELinux until next reboot.
enforce	-rw-rr	Get or set enforcing status. Used by the setenforce (8) command.
load	-rw	Load policy interface.
member	-rw-rw-rw-	Compute polyinstantiation membership decision interface.
mls	-rrr	Returns 1 if MLS policy is enabled or 0 if not.
null	crw-rw-rw-	
policyvers	-rrr	Returns policy version for this kernel ($F-12 = 24$).
relabel	-rw-rw-rw-	Compute relabeling decision interface.
user	-rw-rw-rw-	Compute reachable user contexts interface.
/selinux/avc	Directory	This directory contains information regarding the kernel AVC that can be displayed by the avcstat command.
cache_stats	-rrr	Shows the AVC lookups, hits, misses etc.

¹⁷ That is taken from the UNK_PERMS entry in the Reference Policy <u>build.conf</u> file.

Directory / File Name	Permissions	Comments
cache_threshold	-rw-rr	The default value is 512, however caching can be turned off (but performance suffers) by: echo 0 > /selinux/avc/cache_threshold
hash_stats	-rrr	Shows the number of AVC entries, longest chain etc.
/selinux/booleans	Directory	This directory contains one file for each boolean defined in the active policy.
secmark_audit 	-rw-rr	Each file contains the current status of the boolean (0 = false or 1 = true). The getsebool(8), setsebool(8) and sestatus -b commands use this information.
/selinux/initial_contexts	Directory	This directory contains one file for each initial SID defined in the active policy.
any_socket devnull 	-rrr	Each file contains the initial context of the initial SID as defined in the active policy (e.g. any_socket was assigned system_u:object_r:unconfined_t).
/selinux/policy_capabilities	Directory	This directory contains the policy capabilities that have been configured by default in the kernel. They are generally new features that can be enabled for testing by using the <u>policycap Statement</u> in a monolithic or base policy.
network_peer_controls	-rr	For the F-12 kernel, this file contains '0' (false) which means that the following network_peer_controls are not enabled by default: node: sendto recvfrom netif: ingress egress peer: recv See the <u>NetLabel Loadable Module</u> section for an example with this enabled.
open_perms	-rr	For the F-12 kernel, this file contains '0' (false) which means that open permissions are not enabled by default on the following objects: dir, file, fifo_file, chr_file, blk_file.
/selinux/class	Directory	This directory contains a list of classes and their permissions as defined within the policy.
/selinux/class/appletalk_socket	Directory	Each class has its own directory that contains the following:
index	-rrr	This file contains the allocated class number (e.g. appletalk_socket is '56' in flask.h (linux-2.6.27/security/selinux/include/flask.h)).
/selinux/class/appletalk_socket/ perms	Directory	This directory contains one file for each permission defined in the policy.
accept append bind 	-rr	Each file contains a number but not sure what it represents.

Table 2-9: /selinux File and Directory Information

Notes:

- 1. SIDs are not passed to userspace, only the security context string. The context can be read via the <u>libselinux API</u> where a userspace object manager normally manages the relationship (see "SELinux Support for Userspace Object Managers" [Ref. 17]).
- 2. The /proc filesystem exports the process security context string to userspace via /proc/<pid>/attr interface (where <pid> is the process ID). These can also be managed via the <u>libselinux API</u>.

2.16.2.4 SELinux Boot Process

Figure 2.13 shows the boot process that has been limited to what is considered relevant for initialising SELinux¹⁸, loading the policy and checking whether re-labeling is required. The SELinux kernel initialisation areas are in red. The kernel, mkinitrd and upstart source code rpms were used to find the sequence of events (all corrections welcome).

```
Start Kernel Boot Process
                ./init/main.c start kernel()
Load the initial RAM Disk (this is a temporary root filesystem). The source
       code for this and nash(8) is in the mkinitrd source code.
Kernel calls security init() to initialise the LSM security framework.
For SELinux this results in a call to selinux init() that is in hooks.c
       Set the kernel context to the initial SID value '1' taken from
            include/flask.h(SECINITSID KERNEL)
            The AVC is initialised by a call to avc init()
            Other areas of SELinux get initialised such as the
   selinuxfs (/selinux) pseudo filesystem and netlink with their
             objects set with the initial SIDs from flask.h
             /sbin/nash is run by the kernel.
 /sbin/nash initialises services such as drivers, loads the root filesystem
read-only and loads the SELinux policy using the loadPolicyCommand.
  This function will check various directories, then call the SELinux API
       selinux init load policy function to load the policy.
     Loading the policy will now complete the SELinux initialisation
      with a call to selinux complete init() in hooks.c.
    SELinux will now start enforcing policy or allow permissive access
  depending on the value set in /etc/selinux/config SELINUX=
      The kernel is now loaded, the RAM disk removed, SELinux is
        initialised, the policy loaded and /sbin/init is running
               with the root filesystem in read only mode.
```

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

End Kernel Load and Initialisation
End Kerner Load and Initiansation
/etc/rc.d/sysinit is run by init that will:
mount /proc and sysfs filesystems
Check that the selinuxfs (/selinux) pseudo filesystem is present and whether the current process is labeled kernel_t
If the current SELinux state can be read (/selinux/enforce), then set to value. If cannot read, set to '1'.
Run restorecon – R on /dev if needed.
Kill off /sbin/nash if it is still running.
Run restorecon on /dev/pts if needed.
Set contexts on the files in /etc/rwtab and /etc/statetab
by running restorecon -R path.
Check if relabeling required:
if./autorelabel file+AUTORELABEL=0(in/etc/selinux/config):
then drop to shell for manual relabel, or
if only ./autorelabel file, then run fixfiles -F restore.
remove ./autorelabel file.
If /sbin/init has changed context after the relabel,
then ensure a reboot. ELSE carry on.
/etc/rc.d/sysinit will do other initialisation tasks, then exit.
Note: Some SELinux notes state that /sbin/init is re-exec'ed
to allow it to run in the correct context. Could not find where this
happened as policy seems to be active before init daemon is run??
Initialisation Complete

Figure 2.13: The Boot Sequence – *This shows how SELinux is initialised and the policy loaded during the boot process.*

2.17 SELinux Networking Support

SELinux supports the following types of network labeling:

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

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

Note that F-12 does not have NetLabel or IPSec tools installed as standard, therefore yum can be used to install them as shown below:

```
yum install netlabel_tools
```

```
yum install ipsec-tools
```

2.17.1 compat net Controls

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

Figure 2.14 shows how these network statements are used and the type of allow rules that would be required.

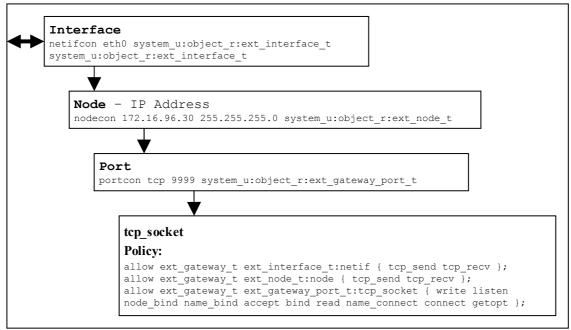


Figure 2.14: compat_net Controls – Showing the policy statements and rules required to allow communications.

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

2.17.2 SECMARK

SECMARK makes use of standard the kernel NetFilter framework that underpins the GNU / Linux IP networking sub-system. NetFilter automatically inspects all incoming and outgoing packets and can place controls on interfaces, IP addresses (nodes) and ports with the added advantage of connection tracking. The SECMARK and CONNSECMARK are security extensions to the Netfilter iptables that allow security contexts to be added to packets (SECMARK) or sessions (CONNSECMARK) such as those used by ftp (as some applications within a single session can use a number of different ports, some fixed and others dynamically allocated).

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

While the implementation of iptables / NetFilter is beyond the scope of this Notebook, there are tutorials available¹⁹. Figure 2.15 shows the basic structure and the process works as follows:

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

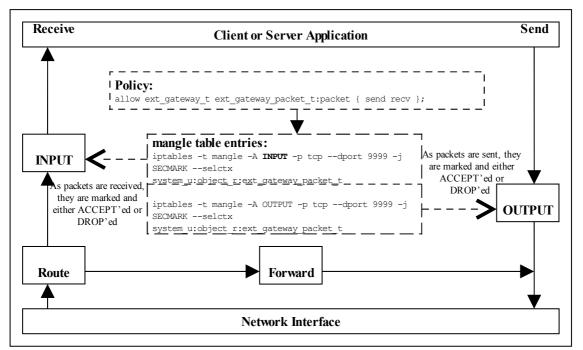


Figure 2.15: SECMARK Processing – *Received packets are processed by the INPUT chain where labels are added to the appropriate packets that will either be accepted or dropped by the SECMARK process. Packets being sent are treated the same way.*

An example iptables²¹ entry is as follows:

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

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

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

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

An example loadable module that makes use of SECMARK services is described in the Building the SECMARK Test Loadable Module section of volume 2, there is also an article "<u>New secmark-based network controls for SELinux</u>" [Ref. 8] that explains the services.

As stated in the <u>compat_net Controls</u> section, SECMARK will be replacing these and there is an article "<u>Transitioning to Secmark</u>" [Ref. 9] that explains the transition.

2.17.3 NetLabel - Fallback Peer Labeling

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

The example message filter has an optional module that makes use of fallback labels as explained in the Overview of modules section of volume 2.

The network peer controls has been extended to support an additional object class of 'peer', although by default this is not enabled in F-12. To enabled this functionality the policy capability needs to be set as explained in the NetLabel Module Support for network_peer_controls section of the Sample Policy Source volume where an example loadable module is given.

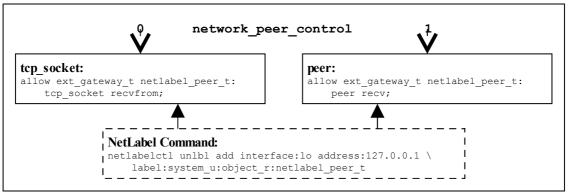


Figure 2.16: Fallback Labeling – Showing the differences between the policy capability network_peer_controls set to 0 and 1.

2.17.4 Labeled IPSec

Labeled IPSec has been built into the standard GNU / Linux IPSec services as described in the "Leveraging IPSec for Distributed Authorization" [Ref. 11] document. Figure 2.17 shows the basic components that form the IPSec service where it is generally used to set up either an encrypted tunnel between two machines²² or an encrypted transport session. The extensions defined in [Ref. 11] describe how the security context is used and negotiated between the two systems (called security associations (SAs) in IPSec terminology).

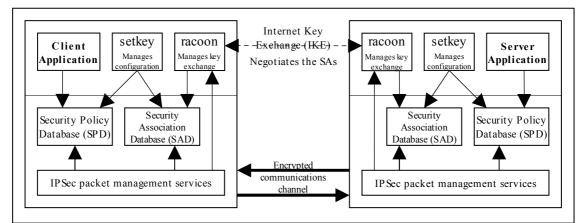


Figure 2.17: IPSec communications – The SPD contains information regarding the security contexts to be used. These are exchanged between the two systems as part of the channel set-up.

Basically what happens is as follows²³:

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

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

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

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

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

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

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

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

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

A worked example of a Labeled IPSec session showing manual and racoon²⁶ to configure the SAD is described in the Labeled IPSec Module Example section of the Sample Policy Source volume.

There is a further example is shown in the "<u>Secure Networking with SELinux</u>" [Ref. 13] article.

```
# setkey -f configuration file entries
# Flush the SAD and SPD
flush;
spdflush;
# Security Association Database entries.
# 1) There would be another SAD entry on the other system (the
     client), where the IP addresses would be reversed.
#
# 2) The security context must be that of the running application.
add 172.16.96.30 172.16.96.31 esp 0x201
-ctx 1 1 "user u:message filter r:ext gateway t"
-E 3des-cbc 0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831;
# Security Policy Database entries.
# 1) there would be another SPD entry on the other system (the
     client), where the IP addresses would be reversed.
\# 2) The security context must be valid (i.e. defined in the active policy as
     it will be used by the polmatch permission process to find a matching
    domain. (note only the 'type' field is used unlike the SAD, where
    the context is the active process).
#
# SAin
spdadd 172.16.96.30 172.16.96.31 any
-ctx 1 1 "system u:object r:ext gateway t"
-P in ipsec esp/transport//require;
# SAout
spdadd 172.16.96.31 172.16.96.30 any
-ctx 1 1 "system_u:object_r:ext_gateway_t"
-P out ipsec esp/transport//require;
```

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

setkey -f <SPD_configuration_file>

²⁶ Unfortunately racoon core dumps using the example base module decribed in volume 2 but does work using the standard Red Hat targeted policy.

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

2.17.5 NetLabel - CIPSO

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

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

- 1. Tag Type 1 bit mapped format that allows a maximum of 256 sensitivity levels and 240 categories to be mapped.
- 2. A non-translation option where labels are passed to / from systems unchanged (for host to host communications as show in Figure 2.18).



Figure 2.18: MLS Systems on the same network

3. A translation option where both the sensitivity and category components can be mapped for systems that have either different definitions for labels or information can be exchanged over different networks (for example using an SELinux enabled gateway as a guard as shown in Figure 2.19).

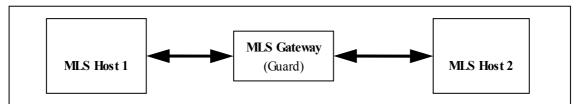


Figure 2.19: MLS Systems on different networks communicating via a gateway

2.18 SELinux Virtual Machine Support

SELinux support is available in the KVM/QEMU and Xen virtual machine (VM) technologies²⁹ 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 opensource virtualisation API that can be used to dynamically load guest VMs on behalf of the virtualisation product. Security extensions were added as a part of the <u>Svirt</u> 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

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

²⁹ 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.

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 and then the libvirt support with some examples that make use of the Virtual Machine Manager (virt-manager rpm) to configure VMs, an overview of the Xen implementation then follows.

2.18.1 KVM / QEMU Support

KVM is a kernel loadable module that uses the Linux kernel as a hypervisorand makes use of a modified QEMU emulator to support the hardware I/O emulation. The "Kernelbased Virtual Machine" [Ref. 21] document gives a good overview of how KVM and QEMU are implemented. It also provides an introduction the virtualisation in general.

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 2.20 shows a high level overview with two VMs running in their own domains. The <u>libvirt</u> Support section shows how to configure these and their VM image files.

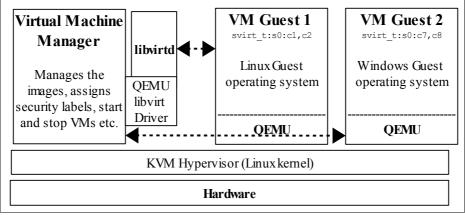


Figure 2.20: KVM Environment - *KVM provides the hypervisor while QEMU provides the hardware emulation services for the guest operating systems.*

2.18.2 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 security hooks can be utilised by any security mechanism by the VM supplier providing a product specific libvirt <u>driver</u> that loads and manages the images. The SELinux implementation (when SELinux is enabled on the host system) supports four methods of labeling VM images, processes and their resources with support from the

Reference Policy modules/services/virt.* loadable module³⁰. To support this labeling, libvirt requires an MCS or MLS enabled policy as the <u>level</u> entry of the security context is used (user:role:type:level).

2.18.2.1 Default Mode

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 system is rebooted 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/<policy_name>/contexts/virtual_domain_context file (the default is system_u:system_r:svirt_t:s0).
- b) An initial context for the image file label is obtained from the /etc/selinux/<policy_name>/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 object context. The process and image objects 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 VM sessions having different label as they were launched during the same boot session:

VM	Object	Dynamically assigned security context
VM1	Process	<pre>system_u:system_r:svirt_t:s0:c585,c813</pre>
VM1	File	<pre>system_u:system_r:svirt_image_t:s0:c585,c813</pre>
VM2	Process	<pre>system_u:system_r:svirt_t:s0:c535,c601</pre>
VM2	File	<pre>system_u:system_r:svirt_image_t:s0:c535,c601</pre>

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 Test_VM1.img
system_u:object_r:svirt_image_t:s0:c535,c601 Test_VM2.img
ps -eZ | grep qemu
system_u:system_r:svirt_t:s0:c585,c813 8707 ? 00:00:44 qemu
system_u:system_r:svirt_t:s0:c1022,c535 8796 ? 00:00:37 qemu
# 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 Test_VM1.img
system_u:object_r:virt_image_t:s0 Test_VM2.img
```

³⁰ 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.

2.18.2.2 Shared Image Mode

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 2.21.

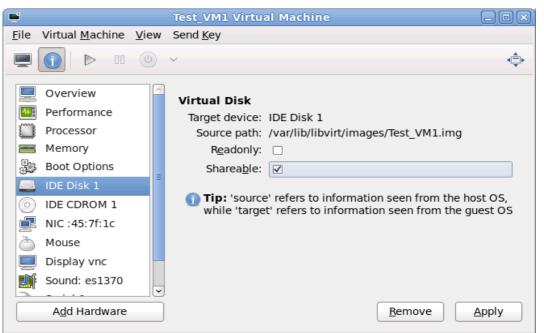


Figure 2.21: Setting the image as Shareable or Readonly

This will set the image (Test_VM1.xml) resource XML configuration file located in the /etc/libvirt/qemu directory <disk> contents as follows:



As the two VMs will share the same image, the Test_VM1 image needs to be cloned and its XML file <disk> contents are as follows (note that it has the same shared image source file name):

<pre># /etc/libvirt/qemu/Test_VM1-clone.xml:</pre>	
<disk device="disk" type="file"></disk>	
<driver name="qemu" type="raw"></driver>	
<source file="/var/lib/libvirt/images/Test_VM1.img"/>	
<target bus="ide" dev="hda"></target>	
<shareable></shareable>	

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/<policy_name>/contexts/virtual_domain_context file (the default is system_u:system_r:svirt_t:s0).
- b) An initial context for the image file label is obtained from the /etc/selinux/<policy_name>/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 object). The process and image objects are 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	Object	Default security context
VM1	Process	<pre>system_u:system_r:svirt_t:s0:c231,c245</pre>
VM1	File	<pre>system_u:system_r:svirt_image_t:s0</pre>
VM2	Process	system_u:system_r:svirt_t:s0:c695,c894
VM1	File	<pre>system_u:system_r:svirt_image_t:s0</pre>

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 as Test_VM1 was cloned:
ls -Z /var/lib/libvirt/images
system_u:object_r:svirt_image_t:s0 Test_VM1.img
ps -eZ | grep qemu
system_u:system_r:svirt_t:s0:c231,c254 6748 ? 00:01:17 qemu
system_u:system_r:svirt_t:s0:c695,c894 7664 ? 00:00:03 qemu
# 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 Test_VM1.img
```

2.18.2.3 Readonly Image Mode

The readonly configuration sequence is similar to the shared option shown above with a dynamically allocated level generated for each VM process instance and the disk image can be shared. The major differences are that the disk image will be read only by setting the image context to virt_content_t (that enforces read only - see the virt.if module interface file - read_blk_files_pattern) instead of svirt_image_t (that allows read/write - rw_blk_files_pattern).

The Virtual Machine Manager can be used to set the image as read only by checking the Readonly box as shown in Figure 2.21. This will set the image (Test_VM1.xml) resource XML configuration file located in the /etc/libvirt/qemu directory <disk> contents as follows:

```
<target dev='hda' bus='ide'/>
<readonly/>
</disk>
```

As the two VMs will share the same image the $Test_VM1$ image needs to be cloned and its XML file <disk> contents will be as follows:

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

- a) An initial context for the process is obtained from the /etc/selinux/<policy_name>/contexts/virtual_domain_context file (the default is system_u:system_r:svirt_t:s0).
- b) An initial context for the image file label is obtained from the /etc/selinux/<policy_name>/contexts/virtual_image_context file. The default for read only images is system_u:system_r:virt_content_t:s0 as discussed above.
- 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 object). The process and image objects are 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	Object	Default security context	
VM1	Process	ystem_u:system_r:svirt_t:s0:c103,c950	
VM1	File	<pre>system_u:system_r:virt_content_t:s0</pre>	
VM2	Process	<pre>system_u:system_r:svirt_t:s0:c312,c820</pre>	
VM1	File	<pre>system_u:system_r:virt_content_t:s0</pre>	

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 as Test_VM1 was cloned:
ls -Z /var/lib/libvirt/images
system_u:object_r:virt_content_t:s0 Test_VM1.img
ps -eZ | grep qemu
system_u:system_r:svirt_t:s0:c103,c950 8756 ? 00:01:08 qemu
system_u:system_r:svirt_t:s0:c312,c820 9246 ? 00:01:03 qemu
# Both VMs stopped (note that the type remained as virt_content_t),
# however if the disk type was reset to <shared/>, then it would be
# reset back to virt_image_t:s0 once the VM was running again.
ls -Z /var/lib/libvirt/images
system_u:object_r:virt_content_t:s0 Test_VM1.img
```

2.18.2.4 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 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.

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 then configured for static labels and 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-12 targeted policy in enforcing mode (just so all errors are flagged during the build):

a) Once the VM has been built, it will need to be stopped from the Static_VM1 Virtual Machine screen. Display the Security menu and select selinux as the Model and check the <u>Static check box</u>. The required security context can then be set - for this example svirt_t has been chosen as it is a valid context:

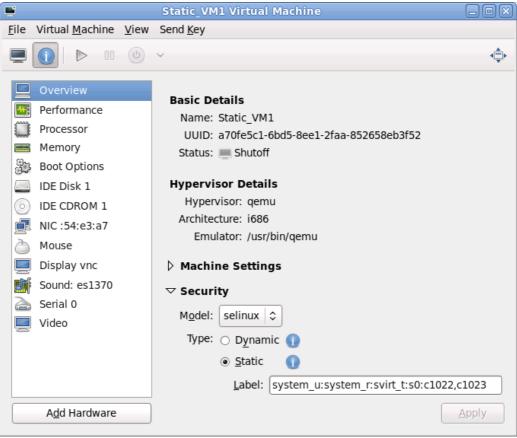


Figure 2.22: Static Configuration

This context will be written to the Static_VM1.xml file in the /etc/libvirt/qemu directory as follows:

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

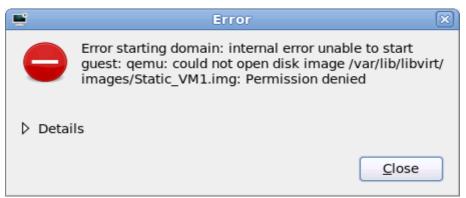


Figure 2.23: Image Start Error

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 as follows:

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

If required, 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 VMs (the unconfined_t configuration is discussed below):

VM	Object	Static security context
VM1	Process	<pre>system_u:system_r:svirt_t:s0:c1022,c1023</pre>
VM1	File	<pre>system_u:system_r:svirt_image_t:s0:c1022,c1023</pre>
VM2	Process	<pre>system_u:system_r:unconfined_t:s0:c11,c22</pre>
VM2	File	<pre>system_u:system_r:virt_image_t:s0</pre>

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:c1022,c1023 Static_VM1.img
system_u:object_r:virt_image_t:s0:c11,c22 Static_VM2.img
ps -eZ | grep qemu
system_u:system_r:svirt_t:s0:c585,c813 6707 ? 00:00:45 qemu
system_u:system_r:unconfined_t:s0:c11,c22 6796 ? 00:00:26 qemu
# 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
```

2.18.2.4.1 Configuring the unconfined_timage

The objective of this section is to configure a VM domain that the targeted policy does not currently support. The domain chosen is unconfined_t as that is the default for general users and requires a minimal additional policy module. The steps required to enable the VM are:

- 1. Using the Virtual Machine Manager, generate a VM (this has been called Static VM2).
- 2. Stop the VM and set a static context of system_u:system_r:unconfined_t:s0:c11,c22. This context will be written to the Static_VM2.xml file in the /etc/libvirt/qemu directory as follows:

3. Before attempting to start the VM clear the audit log first so that a module can be generated with audit2allow to allow the VM to start:

> /var/log/audit/audit.log

4. Now if the VM is started the following error will be shown:



Figure 2.24: Image Execution Error

This is because the libvirt daemon does not have permission to transition the VM process to the unconfined_t domain. The audit log AVC entry would be:

```
type=AVC msg=audit(1271080140.988:30): avc: denied { transition } for
pid=2000 comm="libvirtd" path="/usr/bin/qemu" dev=dm-0 ino=71778
scontext=system_u:system_r:virtd_t:s0-s0:c0.c1023
tcontext=system_u:system_r:unconfined_t:s0:c11,c22 tclass=process
type=SYSCALL msg=audit(1271080140.988:30): arch=40000003 syscall=11
success=no exit=-13 a0=b425c470 a1=b427f610 a2=b42a4a68 a3=0 items=0
ppid=1999 pid=2000 auid=4294967295 uid=107 gid=107 euid=107 suid=107
fsuid=107 egid=107 sgid=107 fsgid=107 tty=(none) ses=4294967295
comm="libvirtd" exe="/usr/sbin/libvirtd" subj=system_u:system_r:virtd_t:s0-
s0:c0.c1023 key=(null)
```

5. To generate a loadable module that will allow the transition use the following commands:

```
# These cmds will generate an unconfinedvm.pp module package:
cat /var/log/audit/audit.log | audit2allow -M unconfinedvm > unconfinedvm.te
# Once the package has been generated, it needs to be activated:
semodule -i unconfinedvm.pp
```

6. Once the module has been loaded and the policy rebuilt, the VM can now be started. For reference the module file generated by audit2allow consists of the following:

2.18.3 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' extension³¹ 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 2.25.

The "<u>How Does Xen Work</u>" [Ref. 22] document describes the basic operation of Xen, the "<u>Xen Security Modules</u>" [Ref. 23] 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.

³¹ This is a version of the SELinux security server, avc etc. that has been specifically ported for the Xen implementation.

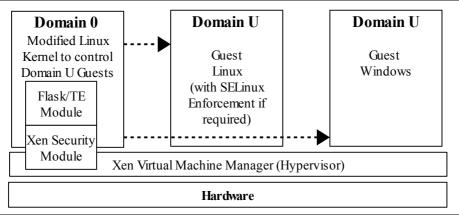


Figure 2.25: Xen Hypervisor - Using XSM and Flask/TE to enforce policy on the physical I/O resources.

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.

2.19 SELinux X-Windows Support

The SELinux X-Windows (XSELinux) implementation provides fine grained access control over the majority of the X-server objects (known as resources). The Reference Policy modules have also been updated to enforce policy using the XSELinux object manager (OM).

This Notebook will only give a high level description of the infrastructure based on Figure 2.26, however the "Application of the Flask Architecture to the X Window System Server" [Ref. 18] paper has a good overview of how the object manager (OM) has been implemented, although it does not cover areas such as polyinstantiation. There are also some sample X-widows applications for experimenting with policy in the Experimenting with X-Windows section of volume 2.

The object classes and permissions are listed in the X Windows Object Classes section.

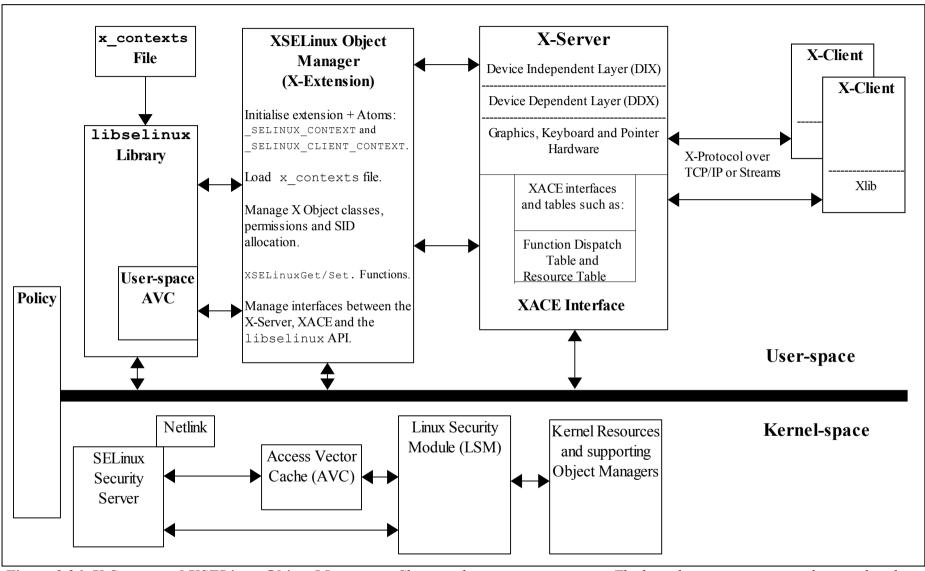


Figure 2.26: X-Server and XSELinux Object Manager – Showing the supporting services. The kernel space services are discussed in the Linux Security Module and SELinux section.

2.19.1 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, just letting the application keep aborting, or modifying the application.

Using Figure 2.26, the major components that form the overall XSELinux OM are (top left to right):

The Policy - The Reference Policy has been updated to support the XSELinux OM and F-12 is now operational from policy version selinux-policy-3.6.32-100.fc12 for targeted and mls versions (Note that in F-12 the OM is enabled for mls and disabled for targeted policies via the xserver-object-manager 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 context configuration information that is required by the OM for labeling certain objects. The OM reads its contents using the selabel_lookup 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 added X-protocol extensions to allow contexts to be set and retrieved by userspace SELinux-aware applications. These are shown in <u>Table 2-10</u> and used in the Experimenting with X-Windows section of volume 2.

XACE Interface - This is a standards based '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.

The interface is defined in the "<u>X Access Control Extension Specification</u>" [Ref. 19]. This 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 are typically windows managers such as Gnome, twm or KDE. The default for F-12 is the Gnome desktop manager.

Kernel-Space Services - These are discussed in the <u>Linux Security Module and</u> <u>SELinux</u> section.

2.19.2 Polyinstantiation

The OM / XACE services support polyinstantiation of properties and selections as these form the InterClient Communication (ICC) that allows X-clients to communicate and exchange information. This allows properties and selections to be grouped into different membership areas so that one group does not know of the exsistance of the others. To implement polyinstantiation the poly_ keyword is required in the <u>x_contexts file</u> for the required selections and/or properties, there is then a corresponding <u>type_member rule</u> in the policy to enforce the separation.

The Experimenting with X-Windows section in volume 2 has examples of using polyinstantiation for selections and then comparing the results to non-polyinstantiated cases.

Note that the current Reference Policy (build 20091117) does not implement polyinstantiation, instead the MLS policy version uses <u>mlsconstrain rules</u> to limit the scope of these.

2.19.3 Configuration Information

This section covers:

- How to determine the OM X-extension opcode.
- How to configure the OM in permissive mode.
- How to disable the OM when using the Reference policy.
- The x-contexts configuration file.
- The OMs SELinuxGet/Set.. functions (shown in <u>Table 2-10</u>).

2.19.3.1 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
        Events: %d Error: %d \n", opcode, event, error);
// Have XSELinux Object Manager
```

2.19.3.2 Configure OM in Permissive Mode

If the X-server object manager needs to be run in permissive mode the following entry can be added to the xorg.conf file (normally in /etc):

```
Section "Module"
SubSection "extmod"
```

```
Option "SELinux mode permissive"
EndSubSection
EndSection
```

2.19.3.3 Disable the OM

The Reference Policy has a boolean that can be used to disable the x-server object manager if is not required by:

setsebool -P xserver_object_manager false

2.19.3.4 The x_contexts File

The $x_contexts$ file contains labels and initial context information that is required by the OM to initialise the service and then to 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 to a new context. A typical entry is as follows:

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

or for polyinstantiation support:

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

The object name can contain '*' for 'any' or '?' for 'substitute'.

The OM uses the selabel functions (such as selabel_lookup) that are a part of libselinux (see <u>Appendix C - API Summary for libselinux</u>) 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). Note that if an object_name has both poly and non-poly entries in the file, the non-poly entry takes precedence (i.e. the poly entry is ignored by the OM).

Note that 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

The <u>Experimenting with X-Windows</u> section has examples of adding additional entries to the $x_contexts$ file.

A full description of the x_contexts file format is given in the <u>contexts/x_contexts</u> <u>File</u> section.

Function Name	Minor Opcode	Parameters	Comments
SELinuxQueryVersion	0	None	Returns the XSELinux version. F-12 returns 1.0
SELinuxSetDeviceCreateContext	1	Context+Len	This is used by SELinux-aware applications for setting the context on device data.
SELinuxGetDeviceCreateContext	2	None	Get the context set on the device data. This is for use by SELinux-aware applications.
SELinuxSetDeviceContext	3	DeviceID + Context+Len	This is used by SELinux-aware applications for setting the context on selected x_device object.
SELinuxGetDeviceContext	4	DeviceID	Get context of the selected x_device object.
SELinuxSetWindowCreateContext	5	Context+Len	This is used by SELinux-aware applications for setting the context on windows data.
SELinuxGetWindowCreateContext	6	None	Get the context set on the window data. This is for use by SELinux-aware applications.
SELinuxGetWindowContext	7	WindowID	Get the process context that this window is running under (x_drawable object ??)
SELinuxSetPropertyCreateContext	8	Context+Len	This is used by SELinux-aware applications for setting the context on property data.
SELinuxGetPropertyCreateContext	9	None	Get the context set on the property data. This is for use by SELinux-aware applications.
SELinuxSetPropertyUseContext	10	Context+Len	This is for use by SELinux-aware applications for setting the context on the property object itself.
SELinuxGetPropertyUseContext	11	None	Get the property object context. This is for use by SELinux-aware applications.
SELinuxGetPropertyContext	12	WindowID + AtomID	Get context of the x_property object.
SELinuxGetPropertyDataContext	13	WindowID + AtomID	Get the context of the property data. This could be the policy default or that set by the SELinuxSetPropertyCreateContext (9) function.
SELinuxListProperties	14	WindowID	List the contexts of properties associated with the selected WindowID.

Function Name	Minor Opcode	Parameters	Comments
SELinuxSetSelectionCreateContext	15	Context+Len	This is used by SELinux-aware applications for setting the context on selected data.
SELinuxGetSelectionCreateContext	16	None	Get the context set on the selected data. This is for use by SELinux-aware applications.
SELinuxSetSelectionUseContext	17	Context+Len	This is for use by SELinux-aware applications for setting the context on the selection object itself.
SELinuxGetSelectionUseContext	18	None	Get the selection object context. This is for use by SELinux-aware applications.
SELinuxGetSelectionContext	19	AtomID	Get context of the x_selection object.
SELinuxGetSelectionDataContext	20	AtomID	Get the context of the selected data. This could be the policy default or that set by the SELinuxSetSelectionCreateContext (15) function.
SELinuxListSelections	21	None	List the selection atoms for this display. The main difference in the listings is that when 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: 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_pastel_t Data Context: system_u:object_r:x_select_pastel_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
SELinuxGetClientContext 22 ResourceID		ResourceID	This function will return the process context any valid X resource ID is running under (or the x_client object ?).

Table 2-10: The XSELinux 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.

2.20 SELinux PostgreSQL Support

This section gives an overview of the SE-PostgreSQL (version 8.4) extensions to support SELinux in F-12 and how the database context information is managed. It assumes some basic knowledge of PostrgreSQL that can be found at the following web site:

http://wiki.postgresql.org/wiki/Main_Page

For a more in-depth overview of SE-PostgreSQL the "<u>Security-Enhanced PostgreSQL</u> <u>Security Wiki</u>" [Ref. 3] is recommended.

2.20.1 SE-PostgreSQL Overview

SE-PostgreSQL adds SELinux mandatory access controls (MAC) to database objects such as databases, tables, columns, rows (tuples), procedures and blobs (binary large objects)³². Figure 2.27 shows a simple database with one table, two columns and three rows, each with their object class and associated security context. The database object classes and permissions are described in <u>Appendix A - Object Classes and Permissions</u>.

<pre>database (db_database) security_context = 'unconfined_u:object_r:sepgsql_db_t:s0:c999'</pre>				
security_context = 'u	<pre>table (db_table) security_context = 'unconfined_u:object_r:sepgsql_table_t:s0:c10'</pre>			
column 1 (db_column)column 2 (db_column)security_context =security_context ='unconfined_u:object_r:sep'unconfined_u:object_r:sepgsql_table_t:s0:c20'pgsql_table_t:s0:c30'				
<pre>row 1 (db_tuple) security_context = 'unconfined_u:object_r:sepg sql_table_t:s0:c100'</pre>	1:1 Information	1:2 Information		
<pre>row 2 (db_tuple) security_context = 'unconfined_u:object_r:sepg sql_table_t:s0:c200'</pre>	2:1 Information	2:2 Information		
<pre>row 3 (db_tuple) security_context = 'unconfined_u:object_r:sepg sql_table_t:s0:c300'</pre>	3:1 Information	3:2 Information		

Figure 2.27: Database Security Context Information - Showing the security contexts that can be associated to a database, table, columns and rows. It is also possible to associate security contexts to procedures and blobs.

<u>Appendix D - SE-PostgreSQL Database Example</u> has a walk-through on how to install SE-PostgreSQL on F-12 with setting up a database, adding tables etc. to show how the security context is used to enforce access control.

³² Version 8.5 will support additional database objects.

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 shown in <u>Appendix D - SE-PostgreSQL Database Example</u> automatically adds a user called sepgsql 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 <u>SELinux Networking Support</u> section (the "<u>Security-Enhanced PostgreSQL Security Wiki</u>" [Ref. 3] also covers these methods of connectivity with examples).

Using Figure 2.28, 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 managed by the object manager (OM) is checked to see if this is allowed by the security policy or not. If the internal AVC does not hold the cached decision then the SELinux kernel Security Server is asked to resolve the query, with the result being cached internally by the OM.

Because PostgreSQL (and therefore SE-PostgreSQL) handles processes, files and directories as part of database operations, the OM also handles permissions for these objects where needed (see the sepostgresql-8.4.2-2583.fcl2.src rpm - perms.c source code) by re-mapping these permissions internally.

SE-PostgreSQL supports SELinux services via the libselinux library, however it does not use the libselinux AVC API functions as it provides its own services. The AVC audits are logged into the sepostgresql.log file as described in the Logging Security Events section.

The SE-PostgreSQL extensions to support MAC access control are described in the <u>SE-PostgreSQL Extensions</u> section.

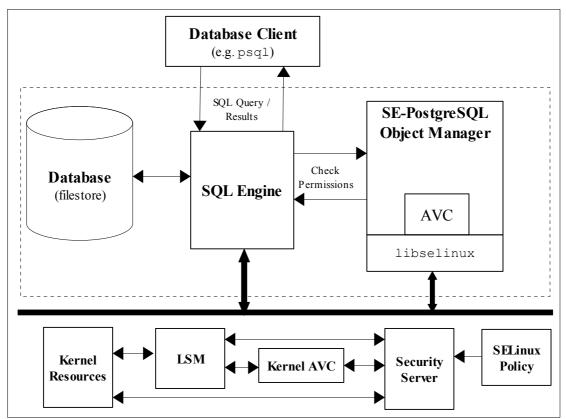


Figure 2.28: SE-PostgreSQL Services - The Object Manager checks access permissions for all objects under its control. If the internal AVC does not hold the cached decision then the SELinux kernel Security Server is asked to resolve, with the result being cached internally by the OM.

2.20.2 SE-PostgreSQL Extensions

The following sections describe the areas that have been extended to manage the security context information and enforce access control. There are a number of examples shown in <u>Appendix D - SE-PostgreSQL Database Example</u> that contains a walk-through of the installation, set-up and using SE-PostgreSQL to build a simple database with a single table, two columns and then adding a number of rows.

The main areas expanded are:

1. Adding an object manager that utilises SELinux support for policy enforcement via libselinux as shown in <u>Figure 2.28</u>. This runs as the sepostgresql server (replacing the postgresql server).

The PostgreSQL internal tables (the system catalog) have also been enhanced to support security context information and are described in the <u>Internal</u><u>Tables</u> section.

- 2. Extending SQL statements to support a security context field.
- 3. Adding additional SQL functions to support viewing and updating security context information.
- 4. Modifying utilities to support security context information.

The sections that follow give a brief overview of the extensions added to support SE-PostgreSQL.

2.20.2.1 Extended SQL Statements

The following SQL Statements have been extended to add a SECURITY_CONTEXT = 'security context' field to support SE-PostgreSQL:

CREATE DATABASE	ALTER DATABASE
CREATE TABLE	ALTER TABLE
CREATE FUNCTION	ALTER FUNCTION

For example to create a table with a specific security context, execute:

```
testdb=# CREATE TABLE info () SECURITY_CONTEXT =
'unconfined_u:object_r:sepgsql_table_t:s0:c10';
CREATE TABLE
```

2.20.2.2 Additional SQL Functions

The following functions have been added to manage the additional security context entries (examples are shown in <u>Appendix D - SE-PostgreSQL Database Example</u>):

sepgsql_getcon	Returns the client security context.
sepgsql_server_getcon	Returns security context of the sepostgresql server.
sepgsql_get_user sepgsql_set_user	Returns or sets the user portion of the security context.
sepgsql_get_role sepgsql_set_role	Returns or sets the role portion of the security context.
sepgsql_get_type sepgsql_set_type	Returns or sets the type portion of the security context.
sepgsql_get_range sepgsql_set_range	Returns or sets the range portion of the security context.
lo_get_security lo_set_security	Returns or sets the security context of a binary large object.
security_reclaim_label	Reclaims orphaned security context entries from the internal pg_security table.

```
security_label_to_secid
Returns the secid column entry from the
pg_security table for the requested
security context.
```

2.20.2.3 Additional Utilities

The pg_dump and pg_dumpall backup and restore utilities have been made SELinux-aware so that the security context is maintained.

An additional utility called sepg_ctl is also supplied that can be used to start, stop, restart, reload configuration files and report the status of a postgresql or sepostgresql server. sepg_ctl --help will list all the options.

2.20.2.4 Additional postgresql.conf Entries

The postgresql.conf file has the following additional entries added to manage the sepostgresql process³³:

sepostgresql	SE-PostgreSQL activation option on or off. The default is on.
sepostgresql_mcstrans	If on (the default) SE-PostgreSQL uses mcstrans to translate the raw security context to a readable text field. If off the context is not translated.
sepostgresql_row_level	If on (the default), then row-level access controls (the db_tuple object class) will be enforced. If off row-level access control is not enforced.

2.20.2.5 Internal Tables

To support the overall database operation PostgreSQL has internal tables in the system catalog that hold information relating to user databases, tables etc. This section will only highlight the internal tables and their columns used by SE-PostgreSQL to support the object classes and security context entries using examples taken from <u>Appendix D - SE-PostgreSQL Database Example</u>.

<u>Table 2-11</u> describes each of the tables used by SE-PostgreSQL to support security context relationships with example SELECT statements to retrieve the relevant information. The only internal table to actually hold security context strings is the pg_security table as all others reference these strings using identifiers as described in <u>Table 2-12</u>.

³³ For the default installation described in <u>Appendix D - SE-PostgreSQL Database Example</u>, the configuration file is located at /var/lib/sepgsql/data/postgresql.conf.

Internal Table Name	Object	Object Class	Comments
pg_database	Database	db_database	The datname column holds the database name.
			<pre>SELECT datname, security_context FROM pg_database WHERE datname = 'testdb'; datname security_context</pre>
			testdb unconfined_u:object_r:sepgsql_db_t:s0
pg_class	Table	db_table	The relname column holds the table name.
			The relnatts column holds the number of columns in this table.
			The relfilenode column value is that contained in the pg_security.relid entry for each row of the table (as they are related).
			<pre>SELECT relname, security_context, relnatts, relfilenode FROM pg_class WHERE relname = 'info'; relname security_context relnatts relfilenode</pre>
			info unconfined_u:object_r:sepgsql_table_t:s0:c10 2 16389
pg_attribute	Column	db_column	The attname column holds the column name.
			The attnum column holds the column number.
			The attrelid column value is that contained in the pg_security.relid entry for each row of the table (as they are related).
			<pre>SELECT attname, security_context, attnum, attrelid FROM pg_attribute WHERE attrelid = 'info'::regclass AND attnum > 0; attname security_context attnum attrelid</pre>
			++
			<pre>user_name unconfined_u:object_r:sepgsql_table_t:s0:c20 1 16389 email_addr unconfined_u:object_r:sepgsql_table_t:s0:c30 2 16389</pre>
pg_security	Row	db_tuple	The pg_security table holds the security context strings and pointers for all objects including the rows (or tuples) as described in <u>Table 2-12</u> .

Table 2-11: PostgreSQL Internal Tables - Note that each table has other columns containing information, however only that relevant to the overview are described.

Table 2-12 describes each of the co	olumns defined in	the pg_	security table with
example entries after the table.			

pg_security Column	Comment	
secid	The unique identifier for this security context. The context is unique for this database (the datid column) and related OID (the relid column for the table, procedure, row etc.).	
datid	The OID of the database to which this entry refers. This can be obtained from the pg_stat_database table as shown in the following example (that will list all contexts used by this instance of the database):	
	<pre>SELECT datname, secid, relid, secattr FROM pg_security, pg_stat_database WHERE pg_security.datid = pg_stat_database.datid AND datname='testdb';</pre>	
relid	The OID of an object (table, column etc.) or the related ID of a row. This section will only describe the table, column and row entries for relid. There are many others that relate to internal OIDs used by PostgreSQL that are beyond the scope of this Notebook ³⁴ . For tables an OID of '1259' is assigned. These relate to table names in the pg_class internal table. For columns an OID of '1249' is assigned. These relate to column names in the pg_attribute internal table. For rows inserted into a table this is the related pg_class.relfilenode and	
seckind	pg_attribute.attrelid entry for that table / column. This is currently 'l' for label.	
secattr	Text string of the security context for the object (database, table etc.).	

Table 2-12: pg_security Table Columns

The following are example entries with comments taken from the pg_security table columns that were displayed using SELECT * FROM pg security;:

datid '0' is assigned to an internal database.
relid '1262' is the pg_database (database) OID

³⁴ Note that the database context (OID = 1262 in the relid column) is listed as being under the datid of database '0'. The best way to retrieve the actual database context is by: SELECT security context FROM pg database WHERE datname = '...';

```
# Therefore this context entry is assigned to database objects.
# Note that datid = 0 and relid = 1262 entries define contexts assigned to
# database instances including 'testdb' (but see next example).
secid | datid | relid | seckind |
                                                 secattr
                               _____
          0 | 1262 | 1 | unconfined_u:object_r:sepgsql_db_t:s0
 3399 |
# This example is for the 'testdb' database after altering its context from the
# above default to 'unconfined u:object r:sepgsql db t:s0:c888' using:
ALTER DATABASE testdb SECURITY CONTEXT =
          'unconfined_u:object_r:sepgsql_db_t:s0:c888'
# This will insert an additional entry into the pg security table as follows:
secid | datid | relid | seckind |
                                                 secattr
                        3400 | 0 | 1262 | 1 | unconfined_u:object_r:sepgsql_db_t:s0:c888
```

2.20.2.6 Logging Security Events

SE-PostgreSQL manages its own AVC audit entries in the /var/log/sepostgresql.log file and by default only errors are logged (i.e. it does not add AVC entries into the standard audit.log). To be able to see greater detail then the boolean sepgsql_enable_audit_allow can be enabled (although this does show much gory detail). A pre-requisite is that the sepostgresql-devel policy module is installed. If the SE-PostgreSQL package has been installed as shown in <u>Appendix D - SE-PostgreSQL Database Example</u>, then the policy module would have been installed but not activated. To activate the module:

```
semodule -i /usr/share/selinux/packages/sepostgresql-devel.pp
```

Once installed, the boolean can be enabled by:

```
setsebool -P sepgsql_enable_audit_allow on
```

The following examples show an sepostgresql.log sequence when the sepgsql_enable_audit_allow boolean has been enabled and a user connects to a database and then performs a SELECT statement.

The following commands are executed:

```
# Connect to a database with the psql client:
psql testdb
# Issue a SELECT statement to retrieve information:
testdb=# SELECT user_name, email_addr, security_context FROM info;
user_name | email_addr | security_context
fred | fred@yahoo.com | unconfined_u:object_r:sepgsql_table_t:s0:c100
derf | derf@hotmail.com | unconfined_u:object_r:sepgsql_table_t:s0:c110
george | george@hotmail.com | unconfined_u:object_r:sepgsql_table_t:s0:c120
(3 rows)
```

And the resulting sepostgresql.log entries would be:

```
# This is the 'psql testdb' sequence:
LOG: SELinux: granted { access }
scontext=unconfined u:unconfined r:unconfined t:s0-s0:c0.c1023
tcontext=unconfined u:object r:sepgsql db t:s0 tclass=db database name=testdb
# This is the 'SELECT user name, email addr, security context FROM info;'
# sequence of events.
LOG: SELinux: granted { select }
scontext=unconfined u:unconfined r:unconfined t:s0-s0:c0.c1023
tcontext=unconfined_u:object_r:sepgsql_table_t:s0:c10 tclass=db_table name=info
STATEMENT: SELECT user name, email addr, security context FROM info;
LOG: SELinux: granted { select }
scontext=unconfined u:unconfined r:unconfined t:s0-s0:c0.c1023
tcontext=unconfined_u:object_r:sepgsql_table_t:s0 tclass=db_column
name=info.security_context
STATEMENT: SELECT user name, email addr, security context FROM info;
LOG: SELinux: granted { select }
scontext=unconfined u:unconfined r:unconfined t:s0-s0:c0.c1023
tcontext=unconfined_u:object_r:sepgsql_table_t:s0:c20 tclass=db_column
name=info.user_name
STATEMENT: SELECT user_name, email_addr, security_context FROM info;
LOG: SELinux: granted { select }
scontext=unconfined_u:unconfined_r:unconfined t:s0-s0:c0.c1023
tcontext=unconfined u:object r:sepgsql table t:s0:c30 tclass=db column
name=info.email addr
STATEMENT: SELECT user name, email addr, security context FROM info;
```

2.21 Apache SELinux Support

Apache web servers are generally managed under SELinux by using the Apache policy modules from the Reference Policy, however an SELinux-aware shared library is available that will allow finer grained access control as described in this section. The additional Apache module is called mod_selinux.so and has a supporting policy module called mod_selinux.pp.

The mod_selinux.pp policy module also makes use of the <u>typebounds</u> <u>Statement</u> that was introduced into version 24 of the policy (that requires a minimum kernel of 2.6.28). This was introduced to allow threads in a multi-threaded application (such as Apache) to be bound within a defined set of permissions (i.e. 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 the following web site:

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 LAPP³⁵ (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.
Р	PostgreSQL has SELinux support using SE-PostgreSQL.
Р	PHP / Perl / Python is not currently SELinux-aware.

The "<u>A secure web application platform powered by SELinux</u>" [Ref. 20] document gives a good overview of the LAPP architecture.

2.21.1 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:
 - 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 it 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 therefore possible to restrict its privileges as described in the <u>Bounds</u> <u>Overview</u> section.
- 3. The web application exits, handing control back to the web server that replies with the HTTP response.

³⁵ This is similar to the LAMP (Linux, Apache, MySQL, PHP/Perl/Python) stack, however MySQL is not SELinux-aware.

2.21.1.1 Bounds Overview

Because multiple threads share the same memory segment, SELinux is unable to check the information flows between these different threads. This means that if a thread (the parent) should launch another thread (a child) with a different security context, SELinux cannot enforce the different permissions (this is why pre 2.6.28 kernels did not allow a different security context to be set on a thread).

To resolve this issue the typebound statement was introduced 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 };
```

States that the parent domain (httpd_t) has file:{getattr read} permissions. However the child domain (httpd_child_t) has been given file: {read write}. This would not be allowed by the compiler because the parent does not have write permission, thus ensuring the child domain will always have equal or less privileges than the parent.

3. SELinux Configuration Files

3.1 Introduction

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

Note: Configuration file names and content have changed over the various releases of SELinux, this section defines those seen in the F-12 release when building custom and Reference Policy policies.

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

- 1. <u>Global Configuration files</u> that affect the active policy and their supporting SELinux-aware applications, utilities or commands. These can be located in /etc/selinux or other places depending on the application. This Notebook will only refer to the commonly used configuration files.
- 2. Files specific to a named policy configuration that are located at /etc/selinux/<policy_name>, where <policy_name> is the name given in the SELINUXTYPE= entry of the /etc/selinux/config file. The files in this area are split into two main sections:
 - a. The <u>Policy Store Configuration files</u> that are 'private'³⁶ and managed by the semanage(8) and semodule(8) commands³⁷. These are located in the /etc/selinux/<policy_name>/module set of directories. These are used to build the majority of the <u>Policy</u> <u>Configuration files</u>.
 - b. The <u>Policy Configuration files</u> that are used when the policy is activated³⁸. The majority of these files are now managed via the Policy Store and should not be edited directly, however others are specific to SELinux-aware applications and have no configuration utilities (e.g. debus and X-Windows context files).
- 3. <u>SELinux Kernel Configuration files</u> that are located under the /selinux directory and reflect the current configuration of SELinux and the active policy. This area is used extensively by the libselinux library for user space object managers and other SELinux-aware applications. These files and directories should not be updated by users (the majority are read only anyway), however they can be read to check various configuration parameters.

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

³⁶ They should NOT be edited as together they describe the 'policy'.

³⁷ The system-config-selinux GUI (supplied in the polycoreutils-gui rpm) can also be used to manage users, booleans and the general configuration of SELinux as it calls semanage, however it does not manage all that the semanage command can (it also gets bitter & twisted if there are no MCS/MLS labels on some operations).

³⁸ The 'active policy' is pointed to by an entry in the /etc/selinux/config file discussed in the <u>/etc/selinux/config</u> file section.

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

3.2 Global Configuration Files

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

3.2.1 /etc/selinux/config File

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

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

Where:

SELINUX	<pre>This can contain one of three values: enforcing SELinux security policy is enforced. permissive SELinux logs warnings (see the <u>Audit Logs section</u>) instead of enforcing the policy (i.e. the action is allowed to proceed). disabled No SELinux policy is loaded.</pre>
SELINUXTYPE	Where policy_name is the policy type or name that will be loaded at system boot time. With F-12 there are three possible values (targeted, mls or minimum) as these are the policies available with the release. However it can be any custom policy with examples being given in the Building a Basic Policy section of volume 2. The policy MUST be located at:
	<pre>/etc/selinux/<policy_name>/</policy_name></pre>
SETLOCALDEFS	This optional field should be set to 0 (or the entry removed) as the policy store management infrastructure (semanage / semodule) is now used.
	If set to 1, then init(8) and load_policy(8) will read the local customisation for booleans and users.

REQUIRESEUSERS	This optional field can be used to fail the login when there is no <u>seusers</u> file if it is set to 1.
	The default action (if 0 or the entry is not present) the libselinux function getseuserbyname will use the GNU / Linux user name.
AUTORELABEL	This is an optional field. If set to '0' and there is a file called .autorelabel in the root directory, then on a reboot, the loader will drop to a shell where a root logon is required. An administrator can then manually relabel the file system.
	If set to '1' or the parameter name is not used (the default) there is no login for manual relabeling, however should the /.autorelabel file exist, then the file system will be automatically relabeled using fixfiles -F restore.
	In both cases the /.autorelabel file will be removed so the relabel is not done again. This option is also discussed in the <u>System Boot Process</u> section.

Example config file contents are:

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

3.2.2 /etc/selinux/semanage.conf File

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

```
module-store = method
policy-version = policy_version
expand-check = 0|1
file-mode = mode
save-previous = true|false
save-linked = true|false
disable-genhomedircon = true|false
handle-unknown = allow|deny|reject
bzip-blocksize = 0|1..9
bzip-small true|false
[verify kernel] # a section to verify policy
```

Where:

module-store	The method can be one of four options:		
	direct	libsemanage will write directly to a module store. This is the default value.	
	source	libsemanage manipulates a source SELinux policy.	
	/foo/bar	Write via a policy management server, whose named socket is at /foo/bar. The path must begin with a '/'.	
	foo.com:4242	Establish a TCP connection to a remote policy management server at foo.com. If there is a colon then the remainder is interpreted as a port number; otherwise default to port 4242.	
policy-version	This optional entry can contain a <u>policy version</u> <u>number</u> , however it is normally commented out as it then defaults to that supported by the system.		
expand-check	This optional entry controls whether hierarchy checking on module expansion is enabled (1) or disabled (0). The default is 0.		
	1	detect the presence of policy acluded with <u>neverallow</u>	
file-mode	set on runtime policy	llows the file permissions to be files. The format is the same as of the chmod command and ot present.	
save-previous	module directory is s	ontrols whether the previous aved (TRUE) after a successful store. The default is to delete the LSE).	
save-linked	linked module is save commit to the policy create a base.link store.	ontrols whether the previously ed (TRUE) after a successful store. Note that this option will ked file in the module policy	
	The default is to delete the previous module (FALSE).		
disable- genhomedircon	genhomedircon f	ontrols whether the embedded function is run when using the d. The default is FALSE.	

handle-unknown	This optional entry controls the kernel behaviour for handling permissions defined in the kernel but missing from the policy (<u>that are declared at the start</u> of the base.conf (loadable policy) or policy.conf (monolithic policy).
	The options are: allow the permission, reject by not loading the policy or deny the permission. The default is deny. See the <u>SELinux Filesystem</u> section for how these are reported in /selinux.
	Note: to activate any change, the base policy needs to be reloaded with the semodule -b command (as semodule -R does not change them).
bzip-blocksize	This optional entry determines whether the modules are compressed or not with bzip. If the entry is 0, then no compression will be used (this is required with tools such as sechecker and apol). This can also be set to a value between 1 and 9 that will set the block size used for compression (bzip will multiply this by 100,000, so '9' is faster but uses more memory).
bzip-small	When this optional entry is set to TRUE the memory usage is reduced for compression and decompression (the bzip -s orsmall option). If FALSE or no entry present, then does not try to reduce memory requirements.
[verify kernel]	This starts an additional set of entries that can be used to validate a policy with an external application during the build process. The validation process takes place before the policy is allowed to be inserted into the store with the SELinux Project web site showing a worked example at:
	http://selinuxproject.org/page/PolicyValidate
	The entries required for this option are as follows:
	<pre>[verify kernel] path = <application_to_run> args = <arguments> [end]</arguments></application_to_run></pre>

Example semanage.config 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 File

The restorecond.conf file contains a list of files that may be created by applications with an incorrect security context. The restorecond daemon will then watch for their creation or modification and automatically correct their security context to that specified by the active policy file context configuration files³⁹ (located in the /etc/selinux/<policy name>/contexts/files directory).

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

Example restorecond.conf file contents are:

```
# /etc/selinux/restorecond.conf
/etc/services
/etc/resolv.conf
/etc/samba/secrets.tdb
/etc/mtab
/var/run/utmp
/var/log/wtmp
# This entry expands to listen for all files created for all
# logged in users within their home directories:
~/*
```

3.2.4 /etc/sestatus.conf File

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

The sestatus.conf file has the following parameters:

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

Example sestatus.conf file contents are:

/etc/sestatus.conf

³⁹ The daemon uses functions in libselinux such as matchpathcon(3) to manage the context updates.

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

3.2.5 /etc/security/sepermit.conf File

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

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

#0.D314 1 (`	
#%PAM-1.0		
auth	[success=done	<pre>ignore=ignore default=bad] pam_selinux_permit.so</pre>
# auth	required	pam_succeed_if.so user != root quiet
auth	required	pam_env.so
auth	substack	system-auth
auth	optional	pam gnome keyring.so
account	required	pam nologin.so
account	include	system-auth
password	include	system-auth
session	required	pam selinux.so close
session	required	pam_loginuid.so
session	optional	pam_console.so
session	required	pam selinux.so open
session	optional	pam keyinit.so force revoke
session	required	pam namespace.so
session	optional	pam gnome keyring.so auto start
session	include	system-auth

The usage is described in the pam_sepermit man page, but the example taken from F-12 describes the configuration:

```
# /etc/security/sepermit.conf
#
# Each line contains either:
# - an user name
# - a group name, with @group syntax
# - a SELinux user name, with %seuser syntax
```

```
# Each line can contain optional arguments separated by :
# The possible arguments are:
# - exclusive - only single login session will be allowed for
# the user and the user's processes will be killed on logout
# An example entry for 'kiosk mode':
xguest:exclusive
```

3.3 Policy Store Configuration Files

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

```
/etc/selinux/<policy name>
```

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

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

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

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

3.3.1 modules/ Files

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

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

3.3.2 modules/active/base.pp File

This is the packaged base policy that contains the mandatory modules and policy components such as object classes, 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 <u>/etc/selinux/semanage.conf</u> 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 <u>base.pp</u> and the loadable modules in the <u>modules/active/modules</u> directory.

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

- <u>homedir_template</u> file that will be used to produce the <u>file_contexts.homedirs</u> file which will then become the policies ./contexts/files/file_contexts.homedirs file.
- 2. <u>file_contexts</u> file that will become the policies ./contexts/files/file contexts file.

The way these two files are built is as follows (and shown in Figure 3.1):

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

The homedir_template file will then be used by genhomedircon (8)⁴⁰ to generate individual SELinux user entries in the file_contexts.homedirs file as discussed in the <u>./modules/active/file_contexts.homedirs</u> section.

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

⁴⁰ The genhomedircon command has now been built into the libsepol library as a function to build the file_contexts.homedirs file via semanage.

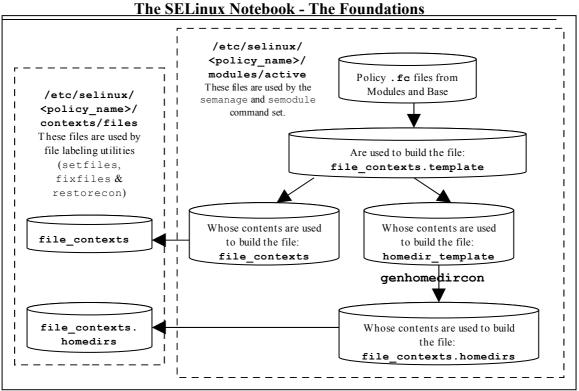


Figure 3.1: File Context Configuration Files - *The two files copied to the policy area will be used by the file labeling utilities to relabel files.*

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

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

```
pathname_regexp opt_security_context
```

Or

```
pathname_regexp file_type opt_security_context
```

Where:

pathname_regexp	An entry that defines the pathname in the form of a regular expression. The metacharacters '^' (match beginning of line) and '\$' (match end of line) are automatically added to the expression by the routines that process this file, however they can be over- ridden by using '.*' at either the beginning or end of the expression (see the example
	file_contexts files below).
	There are also keywords of HOME_ROOT, HOME_DIR, ROLE and USER that are used by file labeling commands (see the <u>keyword</u> <u>definitions below</u> and the <u>./modules/active/homedir_template</u> file section for their usage).
file_type	The file_type options are:

	 '-b' - Block Device '-c' - Character Device '-d' - Directory '-p' - Named Pipe '-1' - Symbolic Link '-s' - Socket '' - Ordinary file
opt_security_context	This entry can be either: a. The security context, including the MLS / MCS level or range if applicable that will be assigned to the file.
	 A value of <<none>> can be used to indicate that the matching files should not be re-labeled.</none>

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.	
ROLE	This keyword is replaced by the 'prefix' entry from the users_extra configuration file that corresponds to the SELinux users user id. Example users_extra configuration file entries are:	
	<pre>user user_u prefix user; user staff_u prefix staff; user group1_u prefix group1;</pre>	
	It is used for files and directories within the users home directory area when relabeling takes place to allow the domain context to be based on a specific role (or any identifier !!) to allow easier identification in log files.	
	It can be added by the semanage user command as follows:	
	<pre># Add prefix for SELinux user: semanage user -a -R staff_r -P group1 group1_u # Add login user: semanage login -a -s group1_u rch</pre>	
	The usage is similar to the Reference Policy 'per_role_template' (<pre>sparse name="userdomain_prefix">) that is an optional component of the external interface file (see the ftp.if or ssh.if files in the Reference Policy source). This feature will probably be removed as the semanage user -P option is more flexible !!!.</pre>	

USER

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

Example file contexts.template contents:

```
# ./modules/active/file contexts.template - These sample entries
# have been taken from the Reference Policy and show the
# HOME DIR, HOME ROOT keywords whose lines will be extracted and
# added to the homedir template file that is used to manage
# user home directory entries. The USER keyword will be replaced
# by the file labeling utilities with the corresponding GNU /
# Linux user id. The ROLE keyword will be replaced by the prefix
# assigned to the SELinux seuser id taken from the users extra
# file.
/.*
                             system u:object r:default t
/a?quota\.(user|group) -- system u:object r:quota db t
/xen(/.*)?
                            system u:object r:xen image t
                          -b system_u:object_r:removable device t
/dev/mcdx?
HOME DIR/.+
                           system u:object r:user home t
/var/log/.*
                             system u:object r:var log t
/tmp/gconfd-USER/.* -- system_u:object_r:gconf_tmp_t
/var/log/sxid\.log.* -- system_u:object_r:sxid_log_t
/var/log/messages[^/]* system_u:object_r:var_log_t
/var/run/wnn-unix(/.*) system_u:object_r:canna_var_run_t
HOME_DIR/\.ircmotd -- system_u:object_r:ROLE_irc_home_t
HOME_ROOT/lost\+found/.* <<none>>
HOME_DIR/\.config/gtk-.* system_u:object_r:gnome_home_t
```

3.3.6 modules/active/file_contexts File

This file becomes the policies <u>./contexts/files/file_contexts</u> file and is built from entries in the <u>./modules/active/file_contexts.template</u> 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 Reference Policy and show the USER keyword
\ensuremath{\texttt{\#}} that will be replaced by the users GNU / Linux user id when
# the file labeling utilities are run.
# The other keywords HOME DIR, HOME ROOT and ROLE have been
# extracted and put in the homedir template file.
                           system u:object r:default t
/.*
/a?quota\.(user|group) -- system_u:object_r:quota_db_t
/xen(/.*)?
                           system_u:object_r:xen_image_t
/dev/mcdx?
                         -b system_u:object_r:removable_device_t
/var/log/.*
                          system_u:object_r:var_log_t
/var/log/.* system_u:object_r:var_log_t
/tmp/gconfd-USER/.* -- system_u:object_r:gconf_tmp_t
```

```
/var/log/sxid\.log.*
                      -- system u:object r:sxid log t
/var/log/messages[^/]* system_u:object_r:var_log_t
/var/run/wnn-unix(/.*)
                         system u:object r:canna var run t
```

```
./contexts/files/file contexts - Sample entries taken 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(/[^/]*) -1 system_u:object_r:mnt_t:s0
/mnt/[^/]*/.* <<none>>
/mnt/[^/]*/.* <<<none>>
/dev/.*mouse.* -c system_u:object_r:mouse_device_t:s0
/dev/.*tty[^/]* -c system_u:object_r:tty_device_t:s0
/dev/[shmx]d[^/]* -b system_u:object_r:fixed_disk_device_t:s15:c0.c255
/var/[xgk]dm(/.*)? system_u:object_r:server_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
/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
                              -d system_u:object_r:var_run_t:s0-s15:c0.c255
-d system_u:object_r:tmp_t:s0-s15:c0.c255
/var/run
/usr/tmp
```

3.3.7 modules/active/homedir template File

This file is built from entries in the <u>file contexts.template</u> 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 <u>file contexts.homedirs</u> 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 Reference Policy and show the
# HOME DIR, HOME ROOT and ROLE keywords that are used to manage
# users home directories:
HOME DIR/.+
                           system_u:object_r:user_home_t
HOME_DIR/.ircmotd -- system_u:object_r:ROLE_irc_home_t
HOME_ROOT/lost\+found/.* <<none>>
HOME_DIR/\.config/gtk-.* system u:object r:gnome home t
```

3.3.8 modules/active/file_contexts.homedirs File

This file becomes the policies

<u>./contexts/files/file_contexts.homedirs</u> file when building policy as shown in <u>Figure 3.1</u>. 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/sin/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 <u>./modules/active/file_contexts.template</u> file, however the HOME_DIR, ROOT_DIR and ROLE keywords will be replaced as explained in the keyword definitions section above. Note that the ROLE keyword will only be replaced for those valid types within the policy (for example if staff_irc_home_t cannot be found in the policy it will be silently dropped from the file_context.homedirs when being built **True?**.

Example file_contexts.homedirs contents:

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

3.3.9 modules/active/netfilter_contexts & netfilter.local File

These files do not seem to be used at present. There is code to produce a netfilter_contexts file for use by the GNU/Linux iptables service⁴¹ 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 <u>bug 201573 – Secmark</u> iptables integration for details).

Example netfilter_contexts contents:

⁴¹ This uses SECMARK labeling that has been utilised by SELinux as described in the <u>SELinux</u> <u>Networking Support</u> section.

```
# This is an example that would be generated by the Reference
# Policy, however seems on hold.
# This is the standard iptables header:
*mangle
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
:selinux input - [0:0]
:selinux_output - [0:0]
:selinux_new_input - [0:0]
:selinux_new_output - [0:0]
-A INPUT -j selinux_input
-A OUTPUT -j selinux output
-A selinux_input -m state --state NEW -j selinux_new_input
-A selinux_input -m state --state RELATED,ESTABLISHED -j CONNSECMARK --restore
-A selinux_output -m state --state NEW -j selinux_new_output
-A selinux output -m state --state RELATED, ESTABLISHED -j CONNSECMARK --restore
-A selinux new input -j SECMARK --selctx system_u:object_r:server_packet_t:s0
# These entries are built from the ports defined in the policy:
# INESE Entries are built from the points defined in the points defined in the points;
-A selinux_new_input -p tcp --dport 7007 -j SECMARK --selctx system_u:object_r:afs_fs_server_packet_t:s0
-A selinux_new_input -p tcp --dport 2040 -j SECMARK --selctx system_u:object_r:afs_fs_server_packet_t:s0
-A selinux_new_input -p udp --dport 7000 -j SECMARK --selctx system_u:object_r:afs_fs_server_packet_t:s0
-A selinux_new_input -p udp --dport 7005 -j SECMARK --selctx system_u:object_r:afs_fs_server_packet_t:s0
. . . . .
# This is the standard iptables trailer:
-A selinux_new_input -j CONNSECMARK --save
-A selinux_new_input -j RETURN
-A selinux_new_output -j CONNSECMARK --save
-A selinux new output -j RETURN
COMMIT
```

3.3.10 modules/active/policy.kern File

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

3.3.11 modules/active/seusers.final and seusers Files

The seusers.final file maps GNU / Linux users to SELinux users and becomes the policies seusers⁴² file as discussed in the <u>./seusers</u> section. The seusers.final file is built or modified when:

 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 follows⁴³:

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.

⁴² 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.

⁴³ The Reference Policy Makefile 'Rules.modular' script uses this method to install the initial seusers 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]
```

or

```
%group_id:seuser_id [:range]
```

Where:

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

Example seusers.final file contents:

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

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

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

The resulting seusers file would be:

```
# ./modules/active/seusers
```

```
rch:user u
```

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

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

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:

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

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

The users_extra file can optionally be included in the base package via the semodule_package(8) command (signified by the -u flag) as follows⁴⁴:

```
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 <u>user Statement</u>). This is useful when only the Reference Policy headers are installed and additional users need to added. The semanage user command will allow a new SELinux user to be added that would generate the user.local file and if a -P flag has been specified, then a users_extra.local file is also generated (note: if this is a new SELinux user and a prefix is not specified a default prefix of user is generated).

The sections that follow will:

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

Note that each line of the users.local file contains a user statement that is defined in the policy language <u>user Statement</u> section, and will be built into the policy via the semanage command.

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

```
user seuser_id prefix prefix_id;
```

Where:

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 ./modules/active/homedir_template file when building the ./modules/active/file_contexts.homedirs file for the relabeling utilities to set the security context on users home directories.

⁴⁴ The Reference Policy Makefile 'Rules.modular' script uses this method to install the initial users_extra file.

Example users_extra file contents:

```
# ./modules/active/users_extra entries, note that the
# users_extra.local file contents are similar and generated by
# the semanage user command.
user user_u prefix user;
user staff_u prefix user;
user sysadm_u prefix user;
user root prefix user;
```

Example semanage user command to add a new SELinux user:

```
# This command will add the user test_u prefix staff entry in
# the users_extra.local file:
semanage user -a -R staff r -P staff test u
```

The resulting users extra.local file is as follows:

```
# ./modules/active/users_extra.local
user test u prefix staff;
```

The resulting users extra file is as follows:

```
# ./modules/active/users_extra
user user_u prefix user;
user staff_u prefix user;
user sysadm_u prefix user;
user root prefix user;
user test_u prefix staff;
```

The resulting users.local file is as follows:

```
# ./modules/active/users.local file entry:
user test_u roles { staff_r } level s0 range s0;
```

3.3.13 modules/active/booleans.local File

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

Example semanage boolean command to modify a boolean value:

```
# This command will add an entry in the booleans.local
# file and set the boolean value to 'off':
semanage boolean -m -0 ext_gateway_audit
```

The resulting booleans.local file would be:

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

```
ext gateway audit=0
```

3.3.14 modules/active/file_contexts.local File

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

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

The format of the file_contexts.local file is the same as the ./modules/active/file_contexts.template file.

Example semanage fcontext command to add a new entry:

```
# This command will add an entry in the file_contexts.local
# file:
semanage fcontext -a -t user_t /usr/move_file
# Note that the type (-t flag) must exist in the policy
# otherwise the command will fail.
```

The resulting file contexts.local file would be:

```
# ./modules/active/file_contexts.local
/usr/move_file system_u:object_r:user_t
```

3.3.15 modules/active/interfaces.local File

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

Each line of the file contains a netifcon statement that is defined along with examples in the <u>netifcon Statement</u> section.

3.3.16 modules/active/nodes.local File

This file is created and updated by the semanage node command to hold network address information that was not delivered by the core policy (i.e. they are not defined

in base.conf file). The new node information is then built into the policy by the semanage process.

Each line of the file contains a nodecon statement that is defined along with examples in the policy language <u>nodecon Statement</u> section.

3.3.17 modules/active/ports.local File

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

Each line of the file contains a portcon statement that is defined along with examples in the policy language <u>portcon Statement</u> section.

3.3.18 modules/active/modules Directory Contents

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

```
# Package the module move_file_c:
semodule_package -o move_file_c.pp -m move_file_c.mod -f
move_file.fc
# Then to install it in the store (at /etc/selinux/modular-test/
# modules/active/modules/move_file_c.pp) and build the binary
# policy file, run the semodule command:
semodule -v -s modular-test -i move_file_c.pp
```

The modules within the policy store can be listed using the semanage -l command as shown below. Note that this will also list the modules that have been disabled by the semanage -d <module_name> command.

```
semanage -1
ext_gateway 1.1.0
int_gateway 1.1.0
move_file 1.1.0
netlabel 1.00 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 or semodule processes. It is possible to build custom monolithic policies that only use the files installed in this area (i.e. do not use semanage or semodule). For

example the simple monolithic policy described in the Building a Basic Policy section of volume 2 could run at init 3 (i.e. no X-Windows) and only require the following configuration files:

./policy/policy.24 – The binary policy loaded into the kernel.

./context/files/file_contexts - To allow the filesystem to be relabeled.

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

 $./{\tt context/dbus_contexts}$ – To allow the debus messaging service to run under SELinux.

3.4.1 seusers File

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

- Using the GNU / Linux user_id, lookup the seuser_id from this file. If an entry cannot be found, then use the __default__ entry.
- To determine the remaining context to be used as the security context, read the <u>./contexts/users/[seuser_id]</u> file. If this file is not present, then:
 - Check for a default context in the <u>./contexts/default_contexts</u> file. If no default context is found, then:
 - Read the <u>./contexts/failsafe_context</u> 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 <u>./modules/active/seusers.final and seusers</u> section, where an example semanage user command is also shown.

Example seusers file contents:

```
# ./seusers file for non-MCS/MLS systems.
system_u:system_u
root:root
fred:user_u
__default__:user_u
```

```
# ./seusers file for an MLS system. Note that the system_u user
# has access to all security levels and therefore should not be
# configured as a valid GNU / Linux user.
system_u:system_u:s0-s15:c0.c255
root:root:s0-s15:c0.c255
```

```
fred:user_u:s0
___default__:user_u:s0
```

Supporting libselinux API functions are:

getseuser getseuserbyname

3.4.2 setrans.conf File

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

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

The semanage command can be used to update this file.

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 <u>secolor.conf</u> File section.

The file format is as follows:

There are a number of configuration options available that are described in a README file within the source code package (but not included with the installed package). The contents of this file (from mcstrans-0.3.1-3 source package) is as follows:

```
# This file is from the mcstrans-0.3.1/conf/README file within the
# source package.
# 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 - Taken from the reference policy.
#
# Multi-Level Security translation table for SELinux
# Uncomment the following to disable translation library
# disable=1
# SystemLow and SystemHigh
s0=SystemLow
s15:c0.c1023=SystemHigh
s0-s15:c0.c1023=SystemLow-SystemHigh
# Unclassified level
s1=Unclassified
# Secret level with compartments
s2=Secret
s2:c0=A
s2:c1=B
# ranges for Unclassified
```

```
s0-s1=SystemLow-Unclassified
s1-s2=Unclassified-Secret
s1-s15:c0.c1023=Unclassified-SystemHigh
# ranges for Secret with compartments
s0-s2=SystemLow-Secret
s2:c1-s15:c0.c1023=Secret:B-SystemHigh
s2:c0,c1-s15:c0.c1023=Secret:AB-SystemHigh
```

Example semanage command:

```
# Add a new entry to the file. Note that the -T flag component
# (the user friendly name for the level) must not have spaces.
semanage translation -a -T Top-Level s15:c1023
# List the setrans.conf file contents
semanage translation -1
...
```

s15:c1023=Top-Level

Supporting libselinux API functions are:

selinux_translations_path
selinux_raw_to_trans_context
selinux_trans_to_raw_context

3.4.3 secolor.conf File

This optional file controls the colour to be associated to the various fields within the mctrans.conf file when the information is displayed by a SELinux colour-aware application (currently none!). This has not been fully documented, however the file format is as follows:

```
color color_name = color_mask
context field string foreground color name background color name
```

Where:

color	The color keyword.
color_name	A descriptive name for the colour (e.g. red).
color_mask	A colour mask starting with a hash (#) that describes the colour with black being #ffffff and white being #000000.
context_field	The colour translation supports different colours on the context string components

	(user, role, type and range or level). Each component is on a separate line.
string	This is a defined string within the setrans.conf file that will be displayed in the colour required. The colour initialisation code will check that the field exists in the file. An * can be used to define all entries of the defined context_field entry.
foreground_color_name	The color_name string that will be used as the foreground colour.
background_color_name	The color_name string that will be used as the background colour.

Example file contents:

```
color black = \#000000
color green = #008000
color yellow = #ffff00
color blue = #0000 \text{ff}
color white = #ffffff
color red = #ff0000
color orange = #ffa500
color tan = \#D2B48C
user * = black black
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
'color' hex digits as follows:
    user : role : type : range
#000000 #000000 #ffffff #000000 #d2b48c #ffa500 #000000 #008000
black black white black tan orange black green
```

3.4.4 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 "<u>SELinux Policy Module Primer</u>" [Ref. 4] notes.

The file name extension ('.24' on F-12), is the policy database version supported by the GNU / Linux release and can be found by executing the following command:

```
cat /selinux/policyvers
24
```

The different versions are discussed in the Policy Versions section.

3.4.5 contexts/customizable types File

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

The file format is as follows:

type

Where:

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

Example file contents:

```
# ./contexts/customizable_types - Taken from the reference
# policy.
mount_loopback_t
public_content_rw_t
public_content_t
swapfile_t
sysadm_untrusted_content_t
sysadm_untrusted_content_tmp_t
```

Supporting libselinux API functions are:

```
is_context_customizable
selinux_customizable_types_path
selinux_context_path
```

3.4.6 contexts/default_contexts File

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

1. The GNU / Linux user identity should be known by the application.

- 2. If a login application, then the SELinux user (seuser), would have been determined as described in the <u>seusers</u> file section.
- 3. The login applications will check the <u>./contexts/users/</u><u>[seuser_id]</u> file first and if no valid entry, will then look in the [seuser_id] file for a default context to use.

The file format is as follows:

role:type role:type ...

Or:

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

Where:

role:type	The file contains one or more lines that consist of role:type pairs.
	The entry at the start of a new line corresponds to the partial role:type context of (generally) the login application.
	The other role:type entries on that line represent an ordered list of valid contexts that could be used to set the users context.
range	The range as defined in the <u>MLS range definition</u> section.

Example file contents:

<pre># ./contexts/default_contexts - Taken from the reference # policy. The highlighted entry at the start of each line # corresponds to the login applications role:type context.</pre>
system r:crond t user r:user crond t staff r:staff crond t
sysadm r:sysadm crond t system r:system crond t unconfined r:unconfined crond t
"" "" "" "" "" "" "" "" "" "" "" "" ""
Ŧ
<pre>system_r:local_login_t user_r:user_t staff_r:staff_t sysadm_r:sysadm_t</pre>
unconfined r:unconfined t
#
system r:remote login t user r:user t staff r:staff t unconfined r:unconfined t
<pre>system_r:sshd_t user_r:user_t staff_r:staff_t sysadm_r:sysadm_t</pre>
unconfined_r:unconfined_t

```
# ./contexts/default_contexts - Taken from the MLS
# reference policy.
system_r:crond_t:s0 system_r:system_crond_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
get_default_context_path
get_ordered_context_list
get_ordered_context_list
get_ordered_context_list_with_level
get_default_context_with_level
get_default_context_with_role
get_default_context_with_rolelevel
query_user_context
manual_user_enter_context
get_default_role
```

An example use in this Notebook (to get over a small feature) is that when the initial basic policy was built (see volume 2), no default_contexts file entries were required as only one role:type of unconfined_r:unconfined_t had been defined, therefore the login process did not need to decide anything (as the only user context was user_u:unconfined_r:unconfined_t).

However when adding the loadable module that used another type (ext gateway t) but with the same role and user (e.g. user u:unconfined r:ext gateway t), then it was found that the login always set the logged process would in user context to user u:unconfined r:ext gateway t (i.e. the login application now had a choice and choose the wrong one, probably because the types are sorted and 'e' comes before 'u').

The end result was that as soon as enforcing mode was set, the system got bitter and twisted. To resolve this the default_contexts file entries were set to:

unconfined_r:unconfined_t unconfined_r:unconfined_t

The login process could now set the context correctly to unconfined_r:unconfined_t. Note that adding the same entry to the contexts/users/user_u configuration file instead could also have achieved this.

3.4.7 contexts/debus contexts File

This file is for the debus 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 <u>dbus-daemon</u> man page details the contents, however it is not recommended that this file is changed. The Free Desktop web site has detailed information at:

http://dbus.freedesktop.org

Example file contents:

```
# ./contexts/debus_contexts - Taken from the reference policy.
<!DOCTYPE busconfig PUBLIC "-//freedesktop//DTD D-BUS Bus
Configuration 1.0//EN"
"http://www.freedesktop.org/standards/dbus/
1.0/busconfig.dtd">
<busconfig.dtd">
</busconfig.dtd">
</busconfig.dtd"
</busconfig.dtd">
</busconfig.dtd">
</busconfig.dtd">
</busconfig.dtd">
</busconfig.dtd">
</busconfig.dtd"
</busconfig.dtd">
</busconfig.dtd"
</busconfig.dtd"
</busconfig.dtd">
</busconfig.dtd"
</busconfig.dtd"
</busconfig.dtd">
</busconfig.dtd"
</p>
```

Supporting libselinux API function is:

selinux context path

3.4.8 contexts/default_type File

This file allows SELinux-aware applications such as newrole (1) to select a default type for a role if one is not supplied. An example use is by newrole when it is called to change a users role, with no type specified, this file would then be consulted to determine the default type to use for the requested role.

The file format is as follows:

role:type

Where:

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

Example file contents:

```
# ./contexts/default_type - Taken from the reference policy.
auditadm_r:auditadm_t
secadm_r:secadm_t
sysadm_r:sysadm_t
staff_r:staff_t
unconfined_r:unconfined_t
user_r:user_t
```

Supporting libselinux API functions are:

```
selinux_context_path
get default type
```

3.4.9 contexts/failsafe_context File

If the login process cannot determine a default context to use, then this can be set to allow an administrator access to the system by setting a known valid context.

The file format is as follows:

role:type

```
Or:
```

role:type:range

Where:

role:type	The file contains a single line that consist of a role:type for a known valid context to allow an administrator access to the system
range	The range or level as defined in the <u>MLS range</u> <u>definition</u> section.

Example file contents:

```
# ./contexts/failsafe_context - Taken from the standard
# reference policy.
```

sysadm_r:sysadm_t

```
# ./contexts/failsafe_context - Taken from the MLS/MCS reference
# policy.
```

sysadm_r:sysadm_t:s0

Supporting libselinux API functions are:

```
selinux_context_path
selinux_failsafe_context_path
```

3.4.10 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 SELinuxaware applications for the same purpose.

The file format is as follows:

security_context

Where:

```
security_context The file contains one line that consists of a full
```

security context, including the MLS / MCS level or range if applicable.

Example file contents:

```
# ./contexts/initrc_context - Taken from the standard reference
# policy.
```

```
system u:system r:initrc t
```

```
# ./contexts/initrc_context - Taken from the MLS reference
# policy. Note that the init process has full access via the
# range s0-s15:c0.c255.
```

```
system_u:system_r:initrc_t:s0-s15:c0.c255
```

Supporting libselinux API functions are:

selinux_context_path

3.4.11 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 <u>./modules/active/netfilter_contexts & netfilter.local</u> file section for further information).

Supporting libselinux API functions are:

```
selinux_context_path
selinux_netfilter_context_path
```

3.4.12 contexts/removable contexts File

This file contains the default label that should be used for removable devices that are not defined in the <u>contexts/files/media</u> file.

The file format is as follows:

```
security_context
```

Where:

security_context	The file contains one line that consists of a full
	security context, including the MLS / MCS level or
	range if applicable.

Example file contents:

./contexts/removable_contexts - Taken from the standard

```
# reference policy.
system_u:object_r:removable_t
# ./contexts/removable_contexts - Taken from the MLS/MCS
# reference policy.
```

system_u:object_r:removable_t:s0

Supporting libselinux API functions are:

selinux_removable_context_path

3.4.13 contexts/securetty_types File

This file is used by the newrole (1) command to find the type to use with tty devices when changing roles or levels.

The file format is as follows:

type

Where:

type	Zero or more type entries that are defined in the policy for		
	tty devices.		

Example file contents:

```
# ./contexts/securetty_types - Taken from the standard reference
# policy.
sysadm_tty_device_t
user_tty_device_t
staff_tty_device_t
```

```
# ./contexts/securetty_types - Taken from the MLS/MCS reference
# policy.
sysadm_tty_device_t
user_tty_device_t
staff_tty_device_t
auditadm_tty_device_t
secureadm_tty_device_t
```

Supporting libselinux API functions are:

```
selinux_securetty_types_path
```

3.4.14 contexts/userhelper_context File

This file contains the default security context used by the system-config-* applications when running from root.

The file format is as follows:

security_context

Where:

security_context	The file contains one line that consists of a full
	security context, including the MLS / MCS level or
	range if applicable.

Example file contents:

```
# ./contexts/userhelper_context - Taken from the 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

3.4.15 contexts/virtual_domain_context File

This is used by the virtulization API (libvirt) and provides the domain contexts that can be used (these are defined in the policy).

Example file contents:

```
# ./contexts/virtual_domain_context - Taken from the standard
# reference policy.
system_u:system_r:svirt_t
```

```
# ./contexts/virtual_domain_context - Taken from the MLS/MCS
# reference policy.
```

```
system_u:system_r:svirt_t:s0
```

Supporting libselinux API functions are:

selinux_virtual_domain_context_path

3.4.16 contexts/virtual_image_context File

This is used by the virtulization API (libvirt) and provides the image contexts that can be used (these are defined in the policy).

Example file contents:

```
# ./contexts/virtual_image_context - Taken from the standard
# reference policy.
system_u:system_r:svirt_image_t:s0
system_u:system_r:svirtcontent_t:s0
```

```
# ./contexts/virtual_image_context - Taken from the MLS/MCS
# reference 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

3.4.17 contexts/x_contexts File

This file is provides the security contexts (and other configuration information) for the X-Windows SELinux security extension. The useage is discussed in the <u>X-windows</u> <u>SELinux Support</u> section and examples of how to add additional entries is shown in the Experimenting with X-Windows section in volume 2. The MCS / MLS version of the file has the appropriate level or range information added.

The selabel_* set of libselinux API functions allow information to be retrieved from the x_contexts file. These are described in the relevant man pages and also in selabel x(5).

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 resourd such as PRIMARY, CUT_BUFFER0 etc. They are general defined in the X-server source code (protocol.txt an BuiltInAtoms in the dix directory of the xorg server source package).	

	This can contain '*' for 'any' or '?' for 'substitute' (see the CUT_BUFFER? 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 (:s0 as standard).

Example file contents (taken from the 20091117 Reference Policy):

```
# 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
                    system_u:object_r:xproperty_t
property *
### 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
                    system u:object r:xextension t
extension *
### 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:Device event XInputExtension:Device event XInputExtension:Device event XInputExtension:Device event XInputExtension:Device event XInputExtension:Device event XInputExtension:Proxim	KeyRelease ButtonPress ButtonRelease MotionNotify Valuator ityIn	<pre>system_u:object_r:input_xevent_t system_u:object_r:input_xevent_t system_u:object_r:input_xevent_t system_u:object_r:input_xevent_t system_u:object_r:input_xevent_t system_u:object_r:input_xevent_t</pre>
<pre>event XInputExtension:Proxim # Client message events event X11:ClientMessage event X11:SelectionNotify event X11:UnmapNotify event X11:ConfigureNotify</pre>	system_u:objec system_u:objec system_u:objec	<pre>system_u:object_r:input_xevent_t t_r:client_xevent_t t_r:client_xevent_t t_r:client_xevent_t t_r:client_xevent_t</pre>
<pre># Default fallback type event *</pre>	system_u:objec	t_r:xevent_t

Supporting libselinux API functions are:

```
selinux_x_context_path
selabel_open
selabel_close
selabel_lookup
selabel_stats
```

3.4.18 contexts/files/file_contexts File

This file is managed by the semodule and semanage commands⁴⁵ 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 <u>file contexts.homedirs</u> file as explained below.

The format of the file_contexts file is the same as the files described in the ./modules/active/file_contexts file section.

Supporting libselinux API functions are:

```
selinux_file_context_path
selinux_file_context_verify
selinux_file_context_local_path
```

3.4.19 contexts/files/file_contexts.local File

This file is added by the semanage fcontext command as described in the <u>./modules/active/file_contexts.local</u> file section to allow locally defined files to be labeled correctly.

⁴⁵ 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.

3.4.20 contexts/files/file_contexts.homedirs File

This file is managed by the semodule and semanage commands as the policy is updated (adding or removing users and modules or updating the base), and therefore should not be edited.

It is generated by the genhomedircon(8) command (in fact by semodule -Bn that rebuilds the policy) and used to set the correct contexts on the users home directory and files.

It is fully described in the <u>./modules/active/file_contexts.homedirs</u> file section.

Supporting libselinux API functions are:

```
selinux_file_context_homedir_path
selinux_homedir_context_path
```

3.4.21 contexts/files/media File

Used to map media types to a file context. If the media_id cannot be found in this file, then the default context in the <u>./contexts/removable_contexts</u> is used instead.

The file format is as follows:

```
media_id file_context
```

Where:

media_id	The media identifier (those known are: cdrom, floppy, disk and usb).
file_context	The context to be used for the device. Note that it does not seem to have the MLS / MCS level).

Example file contents:

```
# contexts/files/media - Taken from the reference policy
# (note that the same file is generated for all types of
# policy).
cdrom system_u:object_r:removable_device_t
floppy system_u:object_r:removable_device_t
disk system_u:object_r:fixed_disk_device_t
```

Supporting libselinux API functions are:

selinux_media_context_path

3.4.22 contexts/users/[seuser_id] File

These optional files are named after the SELinux user they represent (e.g. seuser_id = user_u). Each file has the same format as the <u>contexts/default_contexts</u> file and is used to assign the correct context to the SELinux user.

Example file contents:

```
# ./contexts/users/user_u - Taken from the standard reference
# policy.
system_r:local_login_t user_r:user_t
system_r:remote_login_t user_r:user_t
system_r:sshd_t user_r:user_t
system_r:crond_t user_r:user_t
```

```
# ./contexts/users/user_u - Taken from the MLS/MCS reference
# policy.
system_r:local_login_t:s0 user_r:user_t:s0
system_r:remote_login_t:s0 user_r:user_t:s0
system_r:sshd_t:s0 user_r:user_t:s0
system_r:crond_t:s0 user_r:user_t:s0
user_r:user_su_t:s0 user_r:user_t:s0
user_r:user_su_t:s0 user_r:user_t:s0
user_r:user_sud_t:s0 user_r:user_t:s0
```

Supporting libselinux API functions are:

```
selinux_user_contexts_path
selinux_users_path
selinux_usersconf_path
```

4. SELinux Policy Language

4.1 Introduction

This section is intended as a reference to give a basic understanding of the policy language statements and rules with supporting examples taken from the Reference Policy sources⁴⁶ (where possible). Also all of the language updates to Policy DB version 24 should have been captured. For a more detailed explanation of the policy language the "SELinux by Example" [Ref. 14] book is recommended.

4.2 Policy Statements and Rules

4.2.1 Policy Source Files

There are three basic types of policy source file⁴⁷ that can contain language statements and rules (examples of these can be found in the Building a Basic Policy section in volume 2). The three types of policy source file⁴⁸ are:

Monolithic Policy – This is a single policy source file that contains all statements. By convention this file is called policy.conf and is compiled using the checkpolicy command that produces the binary policy file.

Base Policy – This is the mandatory base policy source file that supports the loadable module infrastructure. The whole system policy could be fully contained within this file, however it is more usual for the base policy to hold the mandatory components of a policy, with the optional components contained in loadable module source files. By convention this file is called base.conf and is compiled using the checkpolicy or checkmodule command.

Module (or Non-base) Policy – These are optional policy source files that when compiled, can be dynamically loaded or unloaded within the policy store. By convention these files are named after the module or application they represent, with the compiled binary having a '.pp' extension. These files are compiled using the checkmodule command.

<u>Table 4-1</u> shows the order in which the statements should appear in source files with the minimum (and therefore mandatory) statements that must be defined.

⁴⁶ From the resulting base.conf and loadable module sources in the ./tmp directory after a make load had been executed for a reference policy 'standard' build and where applicable an 'mls' build.

⁴⁷ It is important to note that the Reference Policy, builds policy using makefiles and support macros within its own source file structure. However, the end result of the make process is that there can be three possible types of source file built (depending on the MONOLITHIC=Y/N build option). These files contain the policy language statements and rules that are finally complied into a binary policy.

⁴⁸ This does not include the 'file_contexts' file as it does not contain policy statements, only persistent security contexts (labels) that will be used by files and directories.

Base Entries	M/ 0	Module Entries	<i>M/O</i>
Security Classes (class)	m	module Statement	0
Initial SIDs	m		
Access Vectors (permissions)	m	require Statement	0
MLS sensitivity, category and level Statements	0		
MLS Constraints	0]	
Policy Capability Statements	0		
Attributes	0	Attributes	0
Booleans	0	Booleans	0
Type / Type Alias	m	Type / Type Alias	0
Roles	m	Roles	0
Policy Rules	0	Policy Rules	0
Users	m	Users	0
Constraints	0		
Default SID labeling	m		
fs_use_xattr Statements	0		
fs_use_task and fs_use_trans Statements	0		
genfscon Statements	0]	
portcon, netifcon and nodecon Statements	0		

Table 4-1: Base and Module Policy Statements – A Monolithic source file would contain the same statements as the Base Module. The Mandatory policy entries are noted (the type, role and user require at least one entry each).

The language grammar defines what statements and rules can be used within the different types of source file. To highlight these rules, the following table is included in each statement and rule section to show what circumstances each one is valid within a policy source file:

Monolithic Policy	Base Policy	Module Policy
Yes/No	Yes/No	Yes/No

Where:

Monolithic Policy	Whether the statement is allowed within a monolithic policy source file or not.
Base Policy	Whether the statement is allowed within a base (for loadable module support) policy source file or not.
Module Policy	Whether the statement is allowed within the optional loadable module policy source file or not.

<u>Table 4-3</u> 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 <u>Conditional Policy</u>, <u>Optional Policy</u> statements and the <u>require statement</u>. 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	<u>require Statement</u>
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 <u>if Statement</u> 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 <u>optional Statement</u> section.
require Statement	Whether the statement keyword is allowed within the require { rule_list } construct as described in the <u>require Statement</u> section.

<u>Table 4-3</u> 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 <u>MLS Statements</u> section defines statements specifically for MLS support. However when MLS is enabled, there are other statements that require the MLS <u>Security Context</u> 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. Statements that were defined in the older NSA documentation have been updated to capture changes such as to prohibit the use of * and ~ in type and role sets (other than in the neverallow statement). Note that some of these changes are not captured by the language grammar, but are managed within the policy_parse.y source code).

4. When multiple source and target entries are shown in a single statement or rule, the compiler (checkpolicy or checkmodule) will expand these to individual statements or rules as shown in the following example:

```
# This allow rule has two target entries console_device_t and
# tty_device_t:
allow apm_t { console_device_t tty_device_t }:chr_file
    { getattr read write append ioctl lock };
# The compiler will expand this to become:
allow apm_t console_device_t:chr_file { getattr read write
    append ioctl lock };
# and:
allow apm_t tty_device_t:chr_file { getattr read write append
    ioctl lock };
```

Therefore when comparing the actual source code with a compiled binary using (for example) apol, sedispol or sedismod, the results will differ (however the resulting policy rules will be the same).

5. Some statements can be added to a policy (via the policy store) using the semanage(8) command. Examples of these are shown where applicable, however the semanage man page should be consulted for all the possible command line options.

alias	allow	and
attribute	auditallow	auditdeny
bool	category	cfalse
class	clone	common
constrain	ctrue	dom
domby	dominance	dontaudit
else	eq	false
fs use task	fs use trans	fs use xattr
 fscon	 genfscon	 h1
h2	if	incomp
inherits	ipv4 addr	ipv6 addr
11	12	level
mlsconstrain	mlsvalidatetrans	module
netifcon	neverallow	nodecon
not	object r	optional
or	permissive	policycap
portcon	rl	r2
r3	range	range transition
require	role	role transition
roles	sameuser	self
sensitivity	sid	source
 t1	t2	t3
target	true	type

6. <u>Table 4-2</u> lists words reserved for the SELinux policy language.

type_change	type_member	type_transition
typealias	typeattribute	types
u1	u2	u3
user	validatetrans	version
version_identifier	xor	typebounds

Table 4-2: Policy language reserved words.

7. <u>Table 4-3</u> 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	<u>Monolithic</u> <u>Policy</u>	<u>Base</u> Policy	<u>Module</u> <u>Policy</u>	<u>Conditional</u> <u>Statements</u>	<u>optional</u> <u>Statement</u>	<u>require</u> <u>Statement</u> 49
allow	Yes	Yes	Yes	Yes	Yes	No
allow - Role	Yes	Yes	Yes	No	Yes	No
attribute	Yes	Yes	Yes	No	Yes	Yes
auditallow	Yes	Yes	Yes	Yes	Yes	No
auditdeny (Depreciated)	Yes	Yes	Yes	Yes	Yes	No
bool	Yes	Yes	Yes	No	Yes	Yes
category	Yes	Yes	No	No	No	Yes
class	Yes	Yes	No	No	No	Yes
common	Yes	Yes	No	No	No	No
constrain	Yes	Yes	No	No	No	No
dominance - MLS	Yes	Yes	No	No	No	No
dominance – Role (Depreciated)	Yes	Yes	Yes	No	Yes	No
dontaudit	Yes	Yes	Yes	Yes	Yes	No
fs_use_task	Yes	Yes	No	No	No	No
fs_use_trans	Yes	Yes	No	No	No	No
fs_use_xattr	Yes	Yes	No	No	No	No
genfscon	Yes	Yes	No	No	No	No
if	Yes	Yes	Yes	No	Yes	No
level	Yes	Yes	No	No	No	No
mlsconstrain	Yes	Yes	No	No	No	No
mlsvalidatetrans	Yes	Yes	No	No	No	No
module	No	No	Yes	No	No	No
netifcon	Yes	Yes	No	No	No	No
neverallow	Yes	Yes	Yes ⁵⁰	No	Yes	No
nodecon	Yes	Yes	No	No	No	No
optional	No	Yes	Yes	Yes	Yes	Yes
permissive	Yes	Yes	Yes	Yes	Yes	No

⁴⁹ Only the statement keyword is allowed.

⁵⁰ neverallow statements are allowed in modules, however to detect these the semanage.conf file must have the expand-check=1 entry present.

Statement / Rule	<u>Monolithic</u> <u>Policy</u>	<u>Base</u> Policy	<u>Module</u> <u>Policy</u>	<u>Conditional</u> <u>Statements</u>	<u>optional</u> <u>Statement</u>	<u>require</u> <u>Statement</u>
policycap	Yes	Yes	No	No	No	No
portcon	Yes	Yes	No	No	No	No
range_transition	Yes	Yes	Yes	No	Yes	No
require	No	Yes ⁵¹	Yes	Yes ⁵²	Yes	No
role	Yes	Yes	Yes	No	Yes	Yes
role_transition	Yes	Yes	Yes	No	Yes	No
sensitivity	Yes	Yes	No	No	No	Yes
sid	Yes	Yes	No	No	No	No
type	Yes	Yes	Yes	No	No	Yes
type_change	Yes	Yes	Yes	Yes	Yes	No
type_member	Yes	Yes	Yes	Yes	Yes	No
type_transition	Yes	Yes	Yes	Yes	Yes	No
typealias	Yes	Yes	Yes	No	Yes	No
typeattribute	Yes	Yes	Yes	No	Yes	No
typebounds	Yes	Yes	Yes	No	Yes	No
user	Yes	Yes	Yes	No	Yes	Yes
validatetrans	Yes	Yes	No	No	No	No

Table 4-3: The policy language statements and rules that are allowed within each type of policy source file - The left hand side of the table shows what Policy Language Statements and Rules are allowed within each type of policy source file. The right hand side of the table shows whether the statement is valid within the if / else construct, optional {rule_list}, or require {rule list} statement.

4.2.5 SELinux Indentifier Naming Conventions

<u>Table 4-4</u> shows some of the general naming conventions used when developing policy in SELinux, these need not be obeyed but does make reading policy much easier.

<u>Table 4-2</u> in the <u>SELinux Policy Language</u> section lists all the reserved words for the SELinux policy language.

⁵¹ Only if preceded by the optional statement.

⁵² Only if preceded by the optional statement.

	type identifier naming conventions
_t	type identifier (Note that attributes by convention do not end in _t by are normally domain_types, file_types etc.)
bin_t	Ordinary program (a binary file)
sbin_t	System administration program
shbin_t	System shared library
ld_so_t	Dynamic linked library
etc_t	Files in /etc directory
var_log_t	Files in /var/log directory
exec_t	Executable files (allowed the file object execute permission)
wtmp_t	System log
boolean_t	A boolean
	Miscellaneous naming conventions
_r	SELinux role identifier
_u	SELinux user identifier
	File naming conventions
.conf	General policy source files:
	policy.conf – monolithic policy
	base.conf – base policy for loadable policy infrastructure
	<module_name>.conf - name of the loadable module</module_name>
.pp	Policy packages (built by semodule (8))
.te	Private policy files in the Reference Policy
.fc	File labeling (context) files in the Reference Policy
.if	Interface files in the Reference Policy

Table 4-4: SELinux identifier naming conventions

4.2.6 Section Contents

The policy language statement and rule sections are as follows:

- a) Type Enforcement and Attribute Statements
- b) Type Enforcement Rules
- c) <u>Access Vector Rules</u>
- d) User Statement
- e) <u>Role Statement</u>
- f) <u>Role Rules</u>
- g) Conditional Policy Statements
- h) Constraint Statements
- i) File System Labeling Statements
- j) <u>Network Labeling Statements</u>
- k) MLS Statements

- 1) Policy Support Statements
- m) Object Class and Permission Statements
- n) Security ID (SID) Statement

4.3 Type Enforcement and Attribute Statements

These statements share the same namespace, therefore the general convention is to use '_t' as the final two characters of a type identifier to differentiate it from an attribute identifier as shown in the following examples:

4.3.1 type Statement

The type statement declares the type identifier and any optional associated alias or attribute identifiers. Type identifiers are the main component of a <u>Security</u> <u>Context</u>.

The statement definition is:

type type_id;

Or

```
type type id ,attribute id;
```

```
Or
```

```
type type_id alias alias_id;
```

Or

```
type type_id alias alias_id ,attribute_id;
```

type	The type keyword.
type_id	The type identifier.
alias	Optional alias keyword that signifies alternate identifiers for the type_id that are declared in the alias_id list.
alias_id	One or more alias identifiers. Multiple entries consist of a space separated list enclosed in braces ({}).
attribute_id	One or more optional attribute identifiers that

have been previously declared by the <u>attribute</u>
Statement. Multiple entries consist of a comma (,)
separated list, also note the lead comma.

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;

4.3.2 attribute Statement

An attribute statement declares an identifier that can then be used to refer to a group of types.

The statement definition is:

```
attribute attribute id;
```

Where:

attribute	The attribute keyword.
attribute_id	The attribute identifier.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	Yes	Yes

Examples:

```
# Using the attribute statement to declare attributes domain,
# daemon, file_type and non_security_file_type:
attribute domain;
attribute daemon;
attribute file_type;
attribute non_security_file_type;
```

4.3.3 typeattribute Statement

The typeattribute statement allows the association of previously declared types to one or more previously declared attributes.

The statement definition is:

```
typeattribute type_id attribute_id [ ,attribute_id ];
```

typeattribute	The typeattribute keyword.
type_id	The identifier of a previously declared type.
attribute_id	One or more previously declared attribute identifiers. Multiple entries consist of a comma (,) separated list.

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	Yes	No

Examples:

Using the typeattribute statement to associate a previously # declared type of setroubleshootd_t to a previously declared # domain attribute. # The previously declared attribute: attribute domain; # The previously declared type: type setroubleshootd_t; # The association using the typeattribute statement: typeattribute setroubleshootd_t domain;

```
# Using the typeattribute statement to associate a type of
# setroubleshootd_exec_t to two attributes file_type and
# non_security_file_type.
# These are the previously declared attributes:
attribute file_type;
attribute non_security_file_type;
# The previously declared type:
type setroubleshootd_exec_t;
# These are the associations using the typeattribute statement:
typeattribute setroubleshootd_exec_t file_type, non_security_file_type;
```

4.3.4 typealias Statement

The typealias statement allows the association of a previously declared type to one or more alias identifiers (an alternative way is to use the type Statement).

The statement definition is:

```
typealias type_id alias alias_id;
```

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 ({}).

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	No

Examples:

```
# Using the typealias statement to associate the previously
# declared type mount_t with an alias of mount_ntfs_t.
# Declare the type:
type mount_t;
# Then alias the identifier:
typealias mount_t alias mount_ntfs_t;
```

```
# Using the typealias statement to associate the previously
# declared type netif_t with two alias, lo_netif_t and
netif_lo_t.
# Declare the type:
type netif_t;
# Then assign two alias identifiers lo_netif_t and netif_lo_t:
typealias netif t alias { lo netif t netif lo t };
```

4.4 Type Enforcement Rules

The TE rules define what access control privileges are allowed for processes. There are three types of enforcement rule: type_transition, type_change, and type member that are explained below.

The typebounds statement was added in version 24 of the policy and has a different format that is explained in the <u>typebounds Statement</u> section.

The common format of the Type Enforcement Rule is:

type_rule source_type target_type : class default_type;

type_rule	The applicable type_transition, type_change, or type_member rule 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	One or more object classes. Multiple entries consist of a space separated list enclosed in braces ({ }).
default_type	A single type identifier that will become the default process type for a domain transition or the type for object transitions.

The statements are valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
Ves	Yes	No

4.4.1 type_transition Statement

The type_transition statement specifies the labeling and object creation allowed between the source_type and target_type when a <u>Domain Transition</u> is requested.

Example – Domain Transition:

```
# Using the type transition statement to show a domain
# transition (as the statement has the process object class
# in the class).
# The rule states that when a process of type initrc t executes
# a file of type acct exec t, the process type should be changed
# to acct t if allowed by the policy (i.e. Transition from the
# initrc t domain to the acc t domain).
type transition initrc t acct exec t:process acct t;
# Note that to be able to transition to the acc 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
# <u>allow Rule</u> 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 };
```

4.4.2 type_change Statement

The type_change statement is used to determine any re-labeling of default types for user space SELinux-aware applications that would then manage any required relabeling via the libselinux API.

Examples:

```
# Using the type_change statement to show that when relabeling a
# character file with type sysadm_devpts_t on behalf of
# auditadm_t, the type auditadm_devpts_t should be used:
type_change auditadm_t sysadm_devpts_t:chr_file
auditadm_devpts_t;
```

```
# Using the type_change statement to show that when relabeling a
# character file with any type associated to the attribute
# server_ptynode on behalf of staff_t, the type staff_devpts_t
# should be used:
type_change staff_t server_ptynode:chr_file staff_devpts_t;
```

4.4.3 type member Statement

The type_member statement determines whether an object can be polyinstantiated. It is used by SELinux-aware applications that would then manage any required polyinstantiation requirements via the libselinux API (see the <u>Polyinstantiation</u> section). Currently only directories are managed by SELinux-aware applications, although the actual statement is not limited to specific object classes.

Example:

```
# Using the type_member statement to show that if the source
# type is sysadm_t, and the target type is user_home_dir_t,
# then use user_home_dir_t as the type on the newly created
# directory object.
type_member sysadm_t user_home_dir_t:dir user_home_dir_t;
```

4.4.4 typebounds Statement

The typebounds statement defines a hierarchical relationship between domains where the bounded domain cannot have any permissions when its bounding domain (the parent) does not have them. 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, [bounded domain];

Where:

typebounds	The typebounds keyword.
bounding_domain	The type identifier of the parent domain.
bounded_domain	One or more $type$ identifiers of the child domains.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	Yes	No

Example:

```
# This example has been taken from [Ref. 20] and 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 flag an error during policy build.
allow httpd_child_t etc_t : file { read write };
```

4.5 Access Vector Rules

The AV rules define what access control privileges are allowed for processes. There are four types of AV rule: allow, dontaudit, auditallow, and neverallow as explained in the sections that follow with a number of examples to cover all the scenarios. There is also an auditdeny rule, however it is no longer used in the Reference Policy and has been replaced by the dontaudit rule.

The general format of an AV rule is that the <code>source_type</code> is the identifier of a process that is attempting to access an object identifier <code>target_type</code>, that has an object class of <code>class</code>, and <code>perm_set</code> 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;

rule_name	The applicable allow, dontaudit, auditallow, and neverallow rule 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 (-).
	The target_type can have the self keyword instead of type or attribute identifiers. This means that the target_type is the same as the source_type.
	The neverallow rule also supports the wildcard operator (*) to specify that all types are to be included and the complement operator (~) to specify all types are to be included except those explicitly listed.
class	One or more object classes. Multiple entries consist of a space separated list enclosed in braces ({ }).
perm_set	The access permissions the source is allowed to access for the target object (also known as the <u>Access</u> <u>Vector</u>). Multiple entries consist of a space separated

list enclosed in braces ({ }).
The optional wildcard operator (*) specifies that all permissions for the object class can be used.
The complement operator (~) is used to specify all permissions except those explicitly listed (although the compiler issues a warning if the dontaudit rule has '~').

The statements are valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
allow = Yes auditallow = Yes dontaudit = Yes neverallow = No	allow = Yes auditallow = Yes dontaudit = Yes neverallow = Yes	allow = No auditallow = No dontaudit = No neverallow = No

4.5.1 allow Rule

The allow rule checks whether the operations between the source_type and target_type are allowed. It is the most common statement that many of the Reference Policy helper macros and interface definitions expand into multiple allow rules.

Examples:

```
# Using the allow rule to show that initrc_t is allowed access
# to files of type acct_exec_t that have the getattr, read and
# execute file permissions:
allow initrc_t acct_exec_t:file { getattr read execute };
```

This rule includes an attribute filesystem_type and states # that kernel_t is allowed mount permissions on the filesystem # object for all types associated to the filesystem_type # attribute:

allow kernel_t filesystem_type:filesystem mount;

This rule includes the self keyword in the target_type that # states that staff_t is allowed setgid, chown and fowner # permissions on the capability object: allow staff_t self:capability { setgid chown fowner }; # This would be the same as the above: allow staff_t staff_t:capability { setgid chown fowner };

This rule includes the wildcard operator (*) on the perm_set

and states that bootloader_t is allowed to use all permissions
available on the dbus object that are type system_dbusd_t:
allow bootloader_t system_dbusd_t:dbus *;
This would be the same as the above:
allow bootloader t system dbusd t:dbus { acquire svc send msg };

```
# This rule includes the complement operator (~) on the perm_set
# and two class entries file and chr_file.
#
# The allow rule states that all types associated with the
# attribute files_unconfined_type are allowed to use all
# permissions available on the file and chr_file objects except
# the execmod permission when they are associated to the types
# listed within the attribute file_type:
allow files_unconfined_type file_type:{ file chr_file }
~execmod;
```

4.5.2 dontaudit Rule

The dontaudit rule stops the auditing of denial messages as it is know that this event always happens and does not cause any real issues. This also helps to manage the audit log by excluding known events.

Example:

```
# Using the dontaudit rule to stop auditing events that are
# known to happen. The rule states that when the traceroute_t
# process is denied access to the name_bind permission on a
# tcp_socket for all types associated to the port_type
# attribute (except port_t), then do not audit the event:
dontaudit traceroute_t { port_type -port_t }:tcp_socket
name_bind;
```

4.5.3 auditallow Rule

Audit the event as a record as it is useful for auditing purposes. Note that this rule only audits the event, it still requires the allow rule to grant permission.

Example:

```
# Using the auditallow rule to force an audit event to be
# logged. The rule states that when the ada_t process has
# permission to execstack, then that event must be audited:
auditallow ada_t self:process execstack;
```

4.5.4 neverallow Rule

This rule specifies that an <u>allow Rule</u> must not be generated for the operation, even if it has been previously allowed. The neverallow statement is a compiler enforced action, where the checkpolicy or checkmodule compiler checks if any allow rules have been generated in the policy source, if so it will issue a warning and stop.

Examples:

```
# Using the neverallow rule to state that no allow rule may ever
# grant any file read access to type shadow_t except those
# associated with the can_read_shadow_passwords attribute:
```

```
neverallow ~can_read_shadow_passwords shadow_t:file read;
```

```
# Using the neverallow rule to state that no allow rule may ever
# grant mmap_zero permissions any type associated to the domain
# attribute except those associated to the mmap_low_domain_type
# attribute (as these have been excluded by the negative
# operator (-)):
neverallow { domain -mmap_low_domain_type } self:memprotect
    mmap zero;
```

4.6 User Statement

4.6.1 user Statement

The user statement is used to declare an SELinux user identifier within the policy and associate that to one or more roles. The statement also allows an optional MLS level and range to control a users security level. It is also possible to add SELinux user id's outside the policy using the 'semanage user' command that will associate the user with roles previously declared within the policy.

The statement definition is:

```
user seuser id roles role id;
```

Or for MCS/MLS Policy:

```
user seuser_id roles role_id level mls_level range mls range;
```

user	The user keyword.
seuser_id	The SELinux user identifier.
roles	The roles keyword.
role_id	One or more previously declared role identifiers. Multiple role identifiers consist of a space separated list enclosed in braces ({}).
level	If MLS is configured, the MLS level keyword.
mls_level	The users default MLS security level that has been previously declared with a <u>level Statement</u> .

	Note that the compiler only accepts the sensitivity component of the level (e.g. s0).
range	If MLS is configured, the MLS range keyword.
mls_range	The range of security levels that the user can run. The format is described in the <u>MLS range</u> <u>Definition</u> section.

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	Yes	Yes

Example:

```
# Using the user statement to define an SELinux user user_u that
# has been assigned the role of user_r. The SELinux user_u is a
# generic user identity for Linux users who have no specific
# SELinux user identity defined.
#
user user_u roles { user_r };
```

MLS Examples:

```
# Using the user statement to define an MLS SELinux user user_u
# that has been assigned the role of user_r and has a default
# login security level of s0 assigned, and is only allowed
# access to the s0 range of security levels (See the
# MLS Statements section for details):
user user_u roles { user_r } level s0 range s0;
```

Using the user statement to define an MLS SELinux user # sysadm_u that has been assigned the role of sysadm_r and has # a default login security level of s0 assigned, and is # allowed access to the range of security levels (low - high) # between s0 and s15:c0.c255 (See the MLS Statements section # for details): user sysadm_u roles { sysadm_r } level s0 range s0 s15:c0.c255;

semanage (8) Command example:

```
# Add user mque_u to SELinux and associate to the unconfined_r
# role:
semanage user -a -R unconfined_r mque_u
```

This command will produce the following files in the default <policy_name> policy store and then activate the policy:

/etc/selinux/<policy name>/modules/active/users.local:

```
# This file is auto-generated by libsemanage
# Do not edit directly.
user mque u roles { unconfined r } ;
```

/etc/selinux/<policy_name>/modules/active/users_extra:

```
# This file is auto-generated by libsemanage
# Do not edit directly.
user mque u prefix user;
```

/etc/selinux/<policy_name>/modules/active/users_extra.local:

```
# This file is auto-generated by libsemanage
# Do not edit directly.
user mque u prefix user;
```

4.7 Role Statement

4.7.1 role Statement

The role statement associates a role identifier to one or more types (i.e. authorise the role to access the domain or domains). Where there are multiple role statements declaring the same role, the compiler will associate the additional types with the role.

The statement definition is:

role role_id;

Or

```
role role_id types type_id;
```

role	The role keyword.
role_id	The identifier of the role being declared. The same role identifier can be declared more than once in a policy, in which case the type_id entries will be amalgamated by the compiler.
types	The optional types keyword.
type_id	When used with the types keyword, one or more

type or attribute identifiers associated with the role_id. Multiple entries consist of a space separated list enclosed in braces ({}). Entries can be excluded from the list by using the negative operator (-).
For role statements, only type or attribute identifiers associated to domains have any meaning within SELinux.

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	Yes	Yes

Examples:

```
# Using the role statement to define standard roles in the
# Reference Policy. Note that there are no domains associated
# with them yet.
role system_r;
role system_r;
role user_r;
role user_r;
role auditadm_r;
# Within the policy the roles are then associated to the
# required domains with this example showing the user_r role
# being associated to two domains:
role user_r types user_t;
role user_r types chfn_t;
```

4.8 Role Rules

4.8.1 Role allow Rule

The role allow rule checks whether a request to change roles is allowed, if it is, then there may be a further request for a role_transition so that the process runs with the new role or role set.

Important Notes:

- 1. The role allow rule has the same keyword as the allow AV rule.
- 2. The role allow rule is used in the Reference Policy sources, however there are no corresponding role transition rules. This is because the policy

expects users to either keep the same role as when they logged onto the system, or use the newrole (1) command to change roles.

3. The Reference Policy uses the <u>constrain Statement</u> to manage role relationships.

The statement definition is:

allow from_role_id to_role_id;

Where:

allow	The role allow rule keyword.
from_role_id	One or more role identifiers that identify the current role. Multiple entries consist of a space separated list enclosed in braces ({ }).
to_role_id	One or more role identifiers that identify the new role to be granted on the transition. Multiple entries consist of a space separated list enclosed in braces ({}).

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	Yes	No

Example:

```
# Using the role allow rule to define authorised role
# transitions in the Reference Policy. The current role
# sysadm_r is granted permission to transition to the secadm_r
# role in the MLS policy.
allow sysadm_r secadm_r;
```

4.8.2 role_transition Rule

The role_transition rule specifies that a role transition is required, and if allowed, the process will run under the new role.

Note that the role_transition rule is not used by the Reference Policy as the policy only allows roles to change at login or by executing the newrole command.

The SECMARK Loadable Module shown in volume 2 however does use a role transition that is described as the example.

The statement definition is:

```
role_transition current_role_id type_id new_role_id;
```

Where:

role_transition	The role_transition keyword.
current_role_id	One or more role identifiers that identify the current role. Multiple entries consist of a space separated list enclosed in braces ({ }).
type_id	One or more type or attribute identifiers. Multiple entries consist of a space separated list enclosed in braces ({}). Entries can be excluded from the list by using the negative operator (-). Only 'domain' types make sense within the policy.
new_role_id	The new role to be granted on transition.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	No

Example:

<pre># 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. #</pre>
<pre># The secure client / server will run in this domain: type ext_gateway_t; # The binaries will be labeled: type secure_services_exec_t;</pre>
<pre># Use message_filter_r role and then transition role message_filter_r types ext_gatway_t; allow unconfined_r message_filter_r; role_transition unconfined_r secure_services_exec_t message_filter_r;</pre>

4.8.3 Role dominance Rule

This rule has been deprecated and therefore should not be used. The role dominance rule allows the dom_role_id to dominate the role_id (consisting of one or more roles). The dominant role will automatically inherit all the type associations of the other roles.

Notes:

- 1. There is another dominance rule for MLS (see the <u>MLS dominance</u> <u>Statement</u>).
- 2. The role dominance rule is not used by the Reference Policy as the policy manages role dominance using the <u>constrain Statement</u>.
- 3. Note the usage of braces '{}' and the ';' in the statement.

The statement definition is:

dominance { role dom role id { role role id; } }

Where:

dominance	The dominance keyword.
role	The role keyword.
dom_role_id	The dominant role identifier.
role_id	For the simple case each { role role_id; } pair defines the role_id that will be dominated by the dom_role_id. More complex rules can be defined but as the statement is depreciated !!!.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	Yes	No

Example:

```
# This shows the dominance role rule, note however that it
# has been depreciated and should not be used.
dominance { role message_filter_r { role unconfined_r };}
```

4.9 **Conditional Policy Statements**

Conditional policies consist of a bool statement that defines a condition as true or false, with a supporting if / else construct that specifies what rules are valid under the condition as shown in the example below:

```
bool allow_daemons_use_tty true;
if (allow_daemons_use_tty) {
    # Rules if condition is true;
} else {
    # Rules if condition is false;
}
```

<u>Table 4-3</u> shows what policy statements or rules are valid within the if / else construct under the "Conditional Statements" column.

The bool statement default value can be changed when a policy is active by using the setsebool command as follows:

```
# This command will set the allow_daemons_use_tty bool to false,
# however it will only remain false until the next system
# re-boot where it will then revert back to its default state
# (in the above case, this would be true).
```

```
setsebool allow_daemons_use_tty false
```

```
# This command will set the allow_daemons_use_tty bool to false,
# and because the -P option is used (for persistent), the value
# will remain across system re-boots. Note however that all
# other pending bool values will become persistent across
# re-boots as well (see the setsebool (8) man page).
```

setsebool -P allow_daemons_use_tty false

The getsebool command can be used to query the current bool statement value as follows:

```
\# This command will list all bool values in the active policy:
```

getsebool -a

```
# This command will show the current allow_daemons_use_tty bool
# value in the active policy:
getsebool allow daemons use tty
```

4.9.1 bool Statement

The bool statement is used to specify a boolean identifier and its initial state (true or false) that can then be used with the <u>if Statement</u> to form a 'conditional policy' as described in the <u>Conditional Policy</u> section.

The statement definition is:

```
bool bool_id default_value;
```

bool	The bool keyword.
bool_id	The boolean identifier.
default_value	Either true or false.

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	Yes	Yes

Examples:

Using the bool statement to allow unconfined executables to # make their memory heap executable or not. As the value is # false, then by default they cannot make their heap executable.

bool allow_execheap false;

```
# Using the bool statement to allow unconfined executables to
# make their stack executable or not. As the value is true,
# then by default their stacks are executable.
```

bool allow execstack true;

4.9.2 if Statement

The if statement is used to form a 'conditional block' of statements and rules that are enforced depending on whether one or more boolean identifiers (defined by the <u>bool</u> <u>Statement</u>) evaluate to TRUE or FALSE. An if / else construct is also supported.

The only statements and rules allowed within the if / else construct are:

```
allow, auditallow, auditdeny, dontaudit, type_member, type transition, type change and require.
```

The statement definition is:

```
if (conditional_expression) { true_list } [ else
{ false_list } ]
```

if	The if keyword.
conditional_expression	One or more bool_name identifiers that have been previously defined by the <u>bool</u> <u>Statement</u> . Multiple identifiers must be separated by the following logical operators: $\&\&, + , \uparrow, !, ==, !=.$

	The conditional_expression is enclosed in brackets ().
true_list	A list of rules enclosed within braces '{ }' that will be executed when the conditional_expression is 'true'.
	Valid statements and rules are highlighted within each language definition statement.
else	Optional else keyword.
false_list	A list of rules enclosed within braces '{}' that will be executed when the optional 'else' keyword is present and the conditional_expression is 'false'. Valid statements and rules are highlighted within each language definition statement.

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No – As this is a Conditional Statement and cannot be nested.	Yes	No

Examples:

```
# An example showing a boolean and supporting if statement.
bool allow_execmem false;
# The bool allow_execmem is FALSE therefore the allow statement
# is not executed:
if (allow_execmem) {
    allow sysadm_t self:process execmem;
}
```

```
# An example of an IF - ELSE statement where the bool statement
# is FALSE, therefore the ELSE statements will be executed.
#
bool read untrusted content false;
if (read_untrusted_content) {
      allow sysadm_t { sysadm_untrusted_content_t
       sysadm untrusted content tmp t }:dir { getattr search
read
       lock ioctl };
      . . . . .
} else {
      dontaudit sysadm_t { sysadm_untrusted_content_t
       sysadm untrusted content tmp t }:dir { getattr search
read
       lock ioctl };
      . . .
}
```

4.10 Constraint Statements

4.10.1 constrain Statement

The constrain statement allows further restriction on permissions for the specified object classes by using boolean expressions covering: source and target types, roles and users as described in the examples.

The statement definition is:

constrain class perm_set expression;

W	here:	
W	here:	

constrain	The constrain keyword.	
class	One or more object classes. Multiple entries consist of a space separated list enclosed in braces ({ }).	
perm_set	One or more permissions. 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</pre>	

r2 op names
t1 op names
t2 op names
Where:
ul, rl, tl = Source user, role, type
u2, r2, t2 = Target user, role, type
and:
op : == !=
role_op : == != eq dom domby incomp
<pre>names : name { name_list }</pre>
<pre>name_list : name name_list name</pre>

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Examples:

These examples have been taken from the Reference Policy source ./policy/constraints file.

```
# This constrain statement is the "SELinux process identity
# change constraint" taken from the Reference Policy source and
# contains multiple expressions.
# The overall constraint is on the process object class with the
# transition permission, and is stating that a domain transition
# is being constrained by the rules listed (u1 == u2 etc.),
# however only the first two expressions are explained.
# The first expression u1 == u2 states that the source (u1) and
# target (u2) user identifiers must be equal for a process
 transition to be allowed.
# However note that there are a number of or operators that can
 override this first constraint.
#
# The second expression:
   ( t1 = can change process identity and t2 = process user target )
#
# states that if the source type (t1) is equal to any type
# associated to the can_change_process_identity attribute, and
# the target type (t2) is equal to any type associated to the
# process_user_target attribute, then a process transition is
# allowed.
# What this expression means in the 'standard' build Reference
# Policy is that if the source domain is either cron t,
# firstboot_t, local_login_t, su_login_t, sshd_t or xdm_t (as
# the can_change_process_identity attribute has these types
```

```
# associated to it) and the target domain is sysadm_t (as that
# is the only type associated to the can_change_process_identity
# attribute), then a domain transition is allowed.
#
# SELinux process identity change constraint:
constrain process transition (
    u1 == u2
or
    ( t1 == can_change_process_identity and t2 == process_user_target )
or
    ( t1 == cron_source_domain and ( t2 == cron_job_domain or u2 == system_u ))
or
    ( t1 == can_system_change and u2 == system_u )
or
    ( t1 == process uncond exempt ) );
```

```
# This constrain statement is the "SELinux file related object
# identity change constraint" taken from the Reference Policy
# source and contains two expressions.
# The overall constraint is on the listed file related object
# classes (dir, file etc.), covering the create, relabelto, and
# relabelfrom permissions. It is stating that when any of the
 object class listed are being created or relabeled, then they
 are subject to the constraint rules listed (u1 == u2 etc.).
# The first expression u1 == u2 states that the source (u1) and
 target (u2) user identifiers (within the security context)
#
 must be equal when creating or relabeling any of the file
#
 related objects listed.
#
#
 The second expression:
  or t1 == can change object identity
#
# states or if the source type (t1) is equal to any type
# associated to the can change object identity attribute, then
# any of the object class listed can be created or relabeled.
# What this expression means in the 'standard' build
# Reference Policy is that if the source domain (t1) matches a
# type entry in the can change object identity attribute, then
# any of the object class listed can be created or relabeled.
# SELinux file related object identity change constraint:
constrain { dir file lnk file sock file fifo file chr file
            blk file } { create relabelto relabelfrom }
(
  u1 == u2
  or t1 == can change object identity
);
```

4.10.2 validatetrans Statement

Only file related object classes are currently supported by this statement and it is used to control the ability to change the objects security context.

Note there are no validatetrans statements specified within the 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)</pre>	
u^{2} , r^{2} , $t^{2} = Ne^{-1}$	d user, role, type w user, role, type ocess user, role, type	
names : name	!= eq dom domby incomp { name_list } e 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:

4.11 File System Labeling Statements

There are four types of file labeling statements: fs_use_xattr, fs_use_task, fs_use_trans and genfscon that are explained below.

The filesystem identifiers (fs_name) used by these statements are defined by the SELinux teams who are responsible for their development, the policy writer then uses those needed to be supported by the policy.

A security context is defined by these filesystem labeling statements, therefore if the policy supports MCS / MLS, then an mls_range is required as described in the MLS range Definition section.

4.11.1 fs use xattr Statements

The fs_use_xattr statement is used to allocate a security context to filesystems that support the extended attribute security.selinux. The labeling is persistent for filesystems that support these extended attributes, and the security context is added to these files (and directories) by the SELinux commands such as setfiles as explained in the Labeling Extended Attribute Filesystems section.

The statement definition is:

fs use xattr fs name fs context;

Where:

fs_use_xattr	The fs_use_xattr keyword.
fs_name	The filesystem name that supports extended attributes. The known valid names are: encfs, ext2, ext3, ext4, ext4dev, gfs, gfs2, jffs2, jfs, lustre and xfs.
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	<u>require Statement</u>
No	No	No

Example:

```
# These statements define file systems that support extended
# attributes (security.selinux).
fs_use_xattr encfs system_u:object_r:fs_t;
fs_use_xattr ext2 system_u:object_r:fs_t;
fs_use_xattr ext3 system_u:object_r:fs_t;
```

MLS Examples:

```
# These statements define file systems that support extended
# attributes (security.selinux).
fs_use_xattr encfs system_u:object_r:fs_t:s0;
fs_use_xattr ext2 system_u:object_r:fs_t:s0;
fs_use_xattr ext3 system u:object r:fs_t:s0;
```

4.11.2 fs_use_task Statement

The fs_use_task statement is used to allocate a security context to pseudo filesystems that support task related services such as pipes and sockets.

The statement definition is:

```
fs_use_task fs_name fs_context;
```

Where:

fs_use_task	The fs_use_task keyword.
fs_name	Filesystem name that supports task related services. The known valid names are: eventpollfs, pipefs and sockfs.
fs_context	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	<u>require Statement</u>
No	No	No

Example:

```
# These statements define the file systems that support pseudo
# filesystems that represent objects like pipes and sockets, so
# that these objects are labeled with the same type as the
# creating task.
#
fs_use_task eventpollfs system_u:object_r:fs_t;
fs_use_task pipefs system_u:object_r:fs_t;
fs_use_task sockfs system_u:object_r:fs_t;
```

MLS Example:

```
# These statements define the file systems that support pseudo
# filesystems that represent objects like pipes and sockets, so
# that these objects are labeled with the same type as the
# creating task.
```

```
fs_use_task eventpollfs system_u:object_r:fs_t:s0;
fs_use_task pipefs system_u:object_r:fs_t:s0;
fs_use_task sockfs system_u:object_r:fs_t:s0;
```

4.11.3 fs use trans Statement

The fs_use_trans statement is used to allocate a security context to pseudo filesystems such as pseudo terminals and temporary objects. The assigned context is derived from the creating process and that of the filesystem type based on transition rules.

The statement definition is:

fs use trans fs name fs context;

Where:

fs_use_trans	The fs_use_trans keyword.
fs_name	Filesystem name that supports transition rules. The known valid 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;
fs_use_trans shm system_u:object_r:tmpfs_t;
fs_use_trans tmpfs system_u:object_r:tmpfs_t;
fs_use_trans devpts system_u:object_r:devpts_t;
```

MLS Example:

```
# These statements define pseudo filesystems such as devpts
# and tmpfs where objects are labeled with a derived context.
#
fs_use_trans mqueue system_u:object_r:tmpfs_t:s0;
fs_use_trans tmpfs system_u:object_r:tmpfs_t:s0;
fs_use_trans tmpfs system_u:object_r:tmpfs_t:s0;
```

```
fs use trans devpts system u:object r:devpts t:s0;
```

4.11.4 genfscon Statements

The genfscon statement is used to allocate a security context to filesystems that cannot support any of the other file labeling statements (fs_use_xattr, fs_use_task or fs_use_trans). Generally a filesystem would have a single default security context assigned by genfscon from the root (/) that would then be inherited by all files and directories on that filesystem. The exception to this is the /proc filesystem, where directories can be labeled with a specific security context (as shown in the examples). Note that there is no terminating semi-colon (;) on this statement.

The statement definition is:

genfscon fs_name partial_path fs_context

Where:

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

Examples:

```
# The following examples show those filesystems that only
# support a single security context across the filesystem.
genfscon msdos / system_u:object_r:dosfs_t
genfscon usbfs / system_u:object_r:usbfs_t
genfscon selinuxfs / system_u:object_r:security_t
# The following show some example /proc entries that can have
# directories added to the path.
genfscon proc / system_u:object_r:proc_t
genfscon proc /fs/openafs system_u:object_r:proc_afs_t
genfscon proc /kmsg system_u:object_r:proc_kmsg_t
```

MLS Examples:

```
# The following examples show those filesystems that only
# support a single security context across the filesystem
# with the MLS levels added.
genfscon msdos / system_u:object_r: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 /fs/openafs system_u:object_r:proc_afs_t:s0
genfscon proc /kmsg system_u:object_r:proc_kmsg_t:s15:c0.c255
```

4.12 Network Labeling Statements

The network labeling statements are used to label the following objects:

Network interfaces – This covers those interfaces managed by the ifconfig(8) command.

Network nodes – These are generally used to specify host systems using either IPv4 or IPv6 addresses.

Network ports – These can be either udp or tcp port numbers.

A security context is defined by these network labeling statements, therefore if the policy supports MCS / MLS, then an mls_range is required as described in the <u>MLS range Definition</u> section. Note that there are no terminating semi-colons (;) on these statements.

If any of the network objects do not have a specific security context assigned by the policy, then the value given in the policies initial SID is used (netif, node or port respectively), as shown below:

```
# Network Initial SIDs from the Standard Reference Policy:
sid netif system_u:object_r:netif_t
sid node system_u:object_r:node_t
sid port system_u:object_r:port_t
# Network Initial SIDs from the MLS Reference Policy:
sid netif system_u:object_r:netif_t:s0 - s15:c0.c255
sid node system_u:object_r:node_t:s0 - s15:c0.c255
sid port system_u:object_r:port_t:s0
```

4.12.1 IP Address Formats

4.12.1.1 IPv4 Address Format

IPv4 addresses are represented in dotted-decimal notation (four numbers, each ranging from 0 to 255, separated by dots as shown:

```
192.77.188.166
```

4.12.1.2 IPv6 Address Formats

IPv6 addresses are written as eight groups of four hexadecimal digits, where each group is separated by a colon (:) as follows:

```
2001:0db8:85a3:0000:0000:8a2e:0370:7334
```

To shorten the writing and presentation of addresses, the following rules apply:

a) Any leading zeros in a group may be replaced with a single '0' as shown:

```
2001:db8:85a3:0:0:8a2e:370:7334
```

b) Any leading zeros in a group may be omitted and be replaced with two colons (::), however this is only allowed once in an address as follows:

2001:db8:85a3::8a2e:370:7334

c) The localhost (loopback) address can be written as:

0000:0000:0000:0000:0000:0000:0000

Or

::1

d) An undetermined IPv6 address i.e. all bits are zero is written as:

::

4.12.2 netifcon Statement

The netifcon statement is used to label network interface objects (e.g. eth0).

It is also possible to add SELinux user id's outside the policy using the 'semanage interface' command that will associate the interface to a security context.

The statement definition is:

netifcon netif_id netif_context packet_context

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.

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
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 unconfined t eth0

This command will produce the following file in the default <policy_name> policy store and then activate the policy:

/etc/selinux/<policy name>/modules/active/interfaces.local:

```
# This file is auto-generated by libsemanage
# Do not edit directly.
netifcon eth0 system_u:object_r:unconfined_t system_u:object_r:unconfined_t
```

4.12.3 nodecon Statement

The nodecon statement is used to label network address objects that represent IPv4 or IPv6 IP addresses and network masks.

It is also possible to add SELinux these outside the policy using the 'semanage node' command that will associate the node to a security context.

The statement definition is:

nodecon subnet netmask node context

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.	

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	No	No

Examples:

```
# The Standard Reference Policy nodecon statement for the IPv4
# Local Host:
nodecon 127.0.0.1 255.255.255.255 system_u:object_r:lo_node_t
# The equivalent MLS Reference Policy nodecon statement for the
# IPv4 Local Host:
nodecon 127.0.0.1 255.255.255.255 system_u:object_r:lo_node_t:
    s0 - s15:c0.c255
```

```
# The Standard Reference Policy nodecon statement for the IPv4
# multicast address:
nodecon 127.0.0.1 255.255.255.255 system_u:object_r:lo_node_t:
    s0 - s15:c0.c255
# The equivalent MLS Reference Policy nodecon statement for the
# multicast address, however using an IPv6 address:
nodecon ff00:: ff00:: system_u:object_r:multicast_node_t:
    s0 - s15:c0.c255
```

semanage (8) Command example:

semanage node -a -t unconfined_t -p ipv4 -M 255.255.255.255 127.0.0.2

This command will produce the following file in the default <policy_name> policy store and then activate the policy:

/etc/selinux/<policy_name>/modules/active/nodes.local:

```
# This file is auto-generated by libsemanage
# Do not edit directly.
```

nodecon ipv4 127.0.0.2 255.255.255.255 system_u:object_r:unconfined_t

4.12.4 portcon Statement

The portcon statement is used to label udp or tcp ports.

It is also possible to add a security context to ports outside the policy using the 'semanage port' command that will associate the port (or range of ports) to a security context.

The statement definition is:

portcon protocol port number port context

Where:

portcon	The portcon keyword.
protocol	The protocol type. Valid entries are udp or tcp.
port_number	The port number or range of ports. The ranges are separated by a hyphen (–).
port_context	The security context for the port or range of ports.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Examples:

```
# The Standard Reference Policy portcon statements:
portcon tcp 20 system_u:object_r:ftp_data_port_t
portcon tcp 600-1023 system_u:object_r:hi_reserved_port_t
portcon udp 600-1023 system_u:object_r:hi_reserved_port_t
portcon tcp 1-599 system_u:object_r:reserved_port_t
portcon udp 1-599 system_u:object_r:reserved_port_t
# The equivalent MLS Reference Policy portcon statements:
portcon tcp 20 system_u:object_r:ftp_data_port_t:s0
portcon tcp 21 system_u:object_r:ftp_port_t:s0
portcon tcp 600-1023 system_u:object_r:hi_reserved_port_t:s0
portcon udp 600-1023 system_u:object_r:hi_reserved_port_t:s0
portcon tcp 1-599 system_u:object_r: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
```

semanage (8) Command example:

semanage port -a -t unconfined_t -p udp 1234

This command will produce the following file in the default <policy_name> policy store and then activate the policy:

/etc/selinux/<policy_name>/modules/active/ports.local:

```
# This file is auto-generated by libsemanage
# Do not edit directly.
portcon udp 1234 system_u:object_r:unconfined_t
```

4.13 MLS Statements

The optional MLS policy extension adds an additional security context component that consists of the following highlighted entries:

```
user:role:type:sensitivity[:category,...]- sensitivity [:category,...]
```

These consist of a mandatory hierarchical <u>sensitivity</u> and optional nonhierarchical <u>category</u>'s. The combination of the two comprise a <u>level</u> or security level as shown in <u>Table 4-5</u>. Depending on the circumstances, there can be one level defined or a <u>range</u> as shown in <u>Table 4-5</u>.

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

 Table 4-5: Sensitivity and Category = Security Level – this table shows the meanings depending on the context being discussed.

To make the security levels more meaningful, it is possible to use the setransd daemon to translate these to human readable formats. The semanage command will allow this mapping to be defined as discussed in the <u>./setrans.conf file</u> section.

4.13.1 sensitivity Statement

The sensitivity statement defines the MLS policy sensitivity identifies and optional alias identifiers.

The statement definition is:

sensitivity identifier;

Or

```
sensitivity sens_id alias alias_id [ alias_id ];
```

Where:

sensitivity	The sensitivity keyword.
sens_id	The sensitivity identifier.
alias	The optional alias keyword.
alias_id	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	<u>require Statement</u>
No	No	Yes

Examples:

```
# The MLS Reference Policy default is to assign 16 sensitivity
# identifiers (s0 to s15):
sensitivity s0;
....
sensitivity s15;
# The policy does not specify any alias entries, however a valid
# example would be:
sensitivity s0 alias secret wellmaybe ornot;
```

4.13.2 MLS dominance Statement

When more than one <u>sensitivity Statement</u> is defined within a policy, then a dominance statement is required to define the actual hierarchy between all sensitivities.

The statement definition is:

```
dominance { sens_id ... }
```

Where:

dominance	The dominance keyword.
sens_id	A space separated list of previously declared sensitivity identifiers (or alias) in the order lowest to highest. They are enclosed in braces ({ }), and note that there is no terminating semi- colon (;).

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	No	No

Example:

```
# The MLS Reference Policy dominance statement defines s0 as the
# lowest and s15 as the highest sensitivity level:
dominance { s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 }
```

4.13.3 category Statement

The category statement defines the MLS policy category identifiers⁵³ and optional alias identifiers.

The statement definition is:

category cat_id;

```
Or
```

```
category cat_id alias alias_id;
```

category	The category keyword.
cat_id	The category identifier.
alias	The optional alias keyword.
alias_id	One or more alias identifiers in a space separated

⁵³ SELinux use the term 'category' or 'categories' while some MLS systems and documentation use the term 'compartment', however they have the same meaning.

list.

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	Yes

Examples:

```
# The MLS Reference Policy default is to assign 256 category
# identifiers (c0 to c255):
category c0;
...
category c255;
# The policy does not specify any alias entries, however a valid
# example would be:
category c0 alias planning development benefits;
```

4.13.4 level Statement

The level statement enables the previously declared sensitivity and category identifiers to be combined into a Security Level.

Note there must only be one level statement for each sensitivity Statement.

The statement definition is:

```
level sens_id [ :category_id ];
```

level	The level keyword.	
sens_id	A previously declared sensitivity identifier.	
category_id	An optional set of zero or more previously declared category identifiers that are preceded by a colon (:), that can be written as follows:	
	• The stop sign (.) 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).	

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	No	No

Examples:

```
# The MLS Reference Policy default is to assign each Security
# Level with the complete set of categories (i.e. the inclusive
# set from c0 to c255):
level s0:c0.c255;
...
level s15:c0.c255;
```

4.13.5 range_transition Statement

The range_transition statement is primarily used by the init process or administration commands to ensure processes run with their correct MLS range (for example init would run at SystemHigh and needs to initialise / run other processes at their correct MLS range). The statement was enhanced in Policy version 21 to accept other object classes.

The statement definition is (for pre-policy version 21):

range_transition source_domain target_exectype new_mls_range;

or (for policy version 21 and greater):

range_transition source_domain target_exectype : class new_mls_range;

range_transition	The range_transition keyword.
source_domain	A source process domain (as only the process object class is supported).
target_exectype	A target executable type or attribute. (i.e. an identifier for a file that has the execute permission set.
class	The optional object class keyword (this allows policy versions 21 and greater to specify a class other than the default of process).
new_mls_range	The new MLS range for the object class. The format of this field is described in the MLS range Definition section.

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 <u>MLS section</u>.

The definition is:

```
low level
```

Or

```
low level - high level
```

low_level	The processes lowest level identifier that has been previously declared by a <u>level Statement</u> .	
	If a high_level is not defined, then it is taken as the same as the low_level.	
-	The optional hyphen (-) separator if a high_level is also being defined.	
high_level	The processes highest $level$ identifier that has been previously declared by a <u>level Statement</u> .	

4.13.6 mlsconstrain Statement

The mlsconstrain statement allows further restriction on permissions for the specified object classes by using boolean expressions covering: source and target types, roles, users and security levels as described in the examples.

The statement definition is:

mlsconstrain class perm_set expression;

Where:

mlsconstrain	The mlsconstrain keyword.	
class	One or more object classes. Multiple entries consist of a space separated list enclosed in braces { }.	
perm_set	One or more permissions. 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)</pre>	
	urce user,role,type,low level,high level rget user,role,type,low level,high level	
and: op : == !=	!= eq dom domby incomp ame_list }	

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	No	No

Examples:

These examples have been taken from the Reference Policy source ./policy/mls constraints file (the mcs file supports the MCS constraints).

These are built into the policy at build time and add constraints to many of the object classes.

```
# The MLS Reference Policy mlsconstrain statement for searching
# directories that comprises of multiple expressions. Only the
# first two expressions are explained.
# Expression 1 ( 11 dom 12 ) reads as follows:
# The dir object class search permission is allowed if the
# source lowest security level is dominated by the targets
# lowest security level.
# OR
# Expression 2 (( t1 == mlsfilereadtoclr ) and ( h1 dom 12 ))
# reads as follows:
# If the source type is equal to a type associated to the
# mlsfilereadtoclr attribute and the source highest security
# level is dominated by the targets lowest security level,
# then search permission is allowed on the dir object class.
mlsconstrain dir search
  (( 11 dom 12 ) or
   (( t1 == mlsfilereadtoclr ) and ( h1 dom 12 )) or
   ( t1 == mlsfileread ) or
   ( t2 == mlstrustedobject ));
```

4.13.7 mlsvalidatetrans Statement

The mlsvalidatetrans is the MLS equivalent of the validatetrans statement and is only used for file related object classes where it is used to control the ability to change the objects security context.

The statement definition is:

```
mlsvalidatetrans class expression;
```

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)</pre>	
Where: u1, r1, t1, l1, h1 = Old user, role, type, low level, high 1 u2, r2, t2, l2, h2 = New user, role, type, low level, high 1	
<pre>u3,r3,t3,l3,h3 = Process user,role,type,lowlevel,hi and: op : == != role_mls_op : == != eq dom domby incomp</pre>	gh level
<pre>names : name { name_list } name_list : name name_list name</pre>	

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
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: ( 11 eq 12 ) 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 }
  ((( 11 eq 12 ) or
  ((t3 == mlsfileupgrade) and (l1 domby l2)) or
  (( t3 == mlsfiledowngrade ) and ( l1 dom l2 )) or
  (( t3 == mlsfiledowngrade ) and ( l1 incomp l2 ))) and (( h1 eq h2 )
or
  (( t3 == mlsfileupgrade ) and ( h1 domby h2 )) or
  (( t3 == mlsfiledowngrade ) and ( h1 dom h2 )) or
  (( t3 == mlsfiledowngrade ) and ( h1 incomp h2 ))));
```

4.14 Policy Support Statements

This section contains language statements used to support policy.

4.14.1 module Statement

This statement is mandatory for loadable modules (non-base) and must be the first line of any module policy source file. The identifier should not conflict with other module names within the overall policy, otherwise it will over-write an existing module when loaded via the semodule command. The semodule -l command can be used to list all active modules within the policy.

The statement definition is:

module module_name version_number;

Where:

module	The module keyword.
module_name	The module name.
version_number	The module version number in $M.m.m$ format (where M = major version number and m = minor version numbers).

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
No	No	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Example:

```
# Using the module statement to define a loadable module called
# bind with a version 1.0.0:
module bind 1.8.0;
```

4.14.2 require Statement

The require statement is used for two reasons:

- 1. Within loadable module policy source files to indicate what policy components are required from an external source file (i.e. they are not explicitly defined in this module but elsewhere). The examples below show the usage.
- 2. Within a base policy source file, but only if preceded by the <u>optional</u> <u>Statement</u> 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 require keyword.
require_list	One or more specific statement keywords with their required identifiers in a semi-colon (;) separated list enclosed within braces ({ }).
	The valid statement keywords are:
	 role, type, attribute, user, bool, sensitivity and category. The keyword is followed by one or more identifiers in a comma (,) separated list, with the last entry being terminated with a semi-colon (;).
	 class. The class keyword is followed by a single object class identifier and one or more permissions. Multiple permissions consist of a space separated list enclosed within braces ({ }). The list is then terminated with a semi-colon (;).
	The examples below show these in detail.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
No	Yes – But only if proceeded by the <u>optional Statement</u> .	Yes

Conditional Policy (if) Statement	optional Statement	require Statement
Yes – But only if proceeded by the optional Statement.	Yes	No

Examples:

```
# A series of require statements showing various entries:
require {
  role system r;
   class security { compute av compute create compute member
      check context load policy compute relabel compute user
      setenforce setbool setsecparam setcheckreqprot };
   class capability2 { mac override mac admin };
}
#
require {
   attribute direct run init, direct init, direct init entry;
   type initrc t;
   role system r;
   attribute daemon;
#
require {
   type nscd_t, nscd_var_run_t;
   class nscd { getserv getpwd getgrp gethost shmempwd shmemgrp
      shmemhost shmemserv };
}
```

4.14.3 optional Statement

The optional statement is used to indicate what policy statements may or may not be present in the final compiled policy. The statements will be included in the policy only if all statements within the optional { rule list } can be expanded successfully, this is generally achieved by using a <u>require Statement</u> at the start of the list.

The statement definition is:

```
optional { rule_list }
```

Or

optional { rule_list } else { rule_list }

optional	The optional keyword.
rule_list	One or more statements enclosed within braces $(\{ \})$. The list of valid statements is given in Table 4-3.

else	An optional else keyword.
rule_list	As the rule_list above.

Monolithic Policy	Base Policy	Module Policy
No	Yes	Yes
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
Yes	Yes	Yes

Examples:

```
# Use of optional block in a base policy source file.
optional {
    require {
        type unconfined_t;
    } # end require
    allow acct_t unconfined_t:fd use;
} # end optional
```

```
# Use of optional / else blocks in a base policy source file.
optional {
    require {
        type ping_t, ping_exec_t;
     } # end require
  allow dhcpc_t ping_exec_t:file { getattr read execute };
   . . . . .
    require {
        type netutils_t, netutils_exec_t;
    } # end require
  allow dhcpc t netutils exec t:file { getattr read execute };
  type transition dhcpc t netutils exec t:process netutils t;
  } else {
    allow dhcpc t self:capability setuid;
     . . . . .
 # end optional
```

4.14.4 policycap Statement

Policy database version 22 introduced the policycap statement to allow new capabilities to be enabled or disabled via the policy. In the Reference Policy there are three policy capabilities configured as shown in the <u>SELinux Filesystem</u> section.

The statement definition is:

```
policycap capability;
```

Where:

policycap	The policycap keyword.
capability	The capability identifier that needs to be enabled for this policy.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Example:

```
# This statement enables the network_peer_controls to be enabled
# for use by the policy.
#
policycap network_peer_controls;
```

4.14.5 permissive Statement

Policy database version 23 introduced the permissive statement to allow the named domain to run in permissive mode instead of running all SELinux domains in permissive mode (that was the only option prior to version 23). Note that the permissive statement:

- 1. Only tests the source context for any policy denial.
- 2. Can be set by the semanage command as it supports a permissive option as follows:

```
# semanage supports enabling and disabling of permissive
# mode using the following command:
# semanage permissive -a|d type
# This example will add a new module in /etc/selinux/
# <policy_name>/modules/active/modules/ called
# permissive_unconfined_t.pp and then reload the policy:
semanage permissive -a unconfined t
```

3. Can be built into a loadable policy module so that permissive mode can be easily enabled or disabled by adding or removing the module. An example module is as follows:

```
# This is an example loadable module that would allow the
# domain to be set to permissive mode.
#
module permissive_unconfined_t 1.0.0;
require {
   type unconfined_t;
```

```
permissive unconfined_t;
```

The statement definition is:

```
permissive type_id;
```

Where:

permissive	The permissive keyword.
type_id	The type identifier of the domain that will be run in permissive mode.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	Yes
Conditional Policy (if) Statement	optional Statement	require Statement
No	Yes	No

Example:

```
# This is the simple statement that would allow permissive mode
# to be set on the httpd_t domain, however this statement is
# generally built into a loadable policy module so that the
# permissive mode can be easily removed by removing the module.
#
permissive httpd t;
```

semanage (8) Command example:

semanage permissive -a unconfined_t

This command will produce the following module in the default <policy_name> policy store and then activate the policy:

/etc/selinux/<policy_name>/modules/active/modules/permissive_unconfined_t.pp

4.15 Object Class and Permission Statements

For those who write or manager SELinux policy, there is no need to define new objects and their associated permissions as these would be done by those who actually design and/or write object managers.

4.15.1 Object Classes

A list of object classes used by Fedora can be found in the Reference Policy source in the ./policy/flask/security_classes file.

Object classes are defined within a policy as follows:

The statement definition is:

class class_id

Where:

class	The class keyword.
class_id	The class identifier.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement

Example:

```
# Define the PostgreSQL db_tuple object class
#
class db_tuple
```

4.15.2 Permissions

A list of permissions used by Fedora can be found in the Reference Policy source in the ./policy/flask/access vectors file.

Permissions can be defined within policy in two ways:

- 1. Define class specific permissions. This is where permissions are declared for a specific object class only (i.e. the permission is not inherited by any other object class).
- 2. Define a set of common permissions that can then be inherited by one or more object classes. The statement for creating a set of common permissions is shown in the <u>Defining common Permissions</u> section.

The permission (or AVC) statement definition is:

```
class class id [ inherits common set ] [ { perm set } ]
```

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 as described in the <u>Defining common Permissions</u> section.
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	<u>require Statement</u>
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 <u>Defining common Permissions</u>
# section):
```

class db_blob inherits database

The following example (from the access_vector file) shows the # db_blob object class inheriting permissions from the database # set of common permissions and adding a further four # permissions:

class db_blob inherits database { read write import export }

4.15.2.1 Defining common Permissions

A list of common permissions used by Fedora can be found in the Reference Policy source in the ./policy/flask/access vectors file.

New or updated common permissions would only be updated by those who produce kernel or user space object managers.

The statement definition is:

```
common common_id { perm_set }
```

Where:

common	The common keyword.
common_id	The common identifier.
perm_set	One or more permission identifiers in a space separated list enclosed within braces ({ }).

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	No	No

Example:

```
# Define the common PostgreSQL permissions
#
common database { create drop getattr setattr relabelfrom
relabelto }
```

4.16 Security ID (SID) Statement

There are two SID statements, the first one declares the actual SID identifier and is defined at the start of a policy source file. The second statement is used to add an initial security context to the SID that is used when SELinux initialises or as a default if an object is not labeled correctly.

4.16.1 sid Statement

The sid statement declares the actual SID identifier and is defined at the start of a policy source file.

The statement definition is:

sid	sid	id	

Where:

sid	The sid keyword.
sid_id	The sid identifier. Note that there is no terminating ';'.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	require Statement
No	No	No

Example:

This example has been taken from the Reference Policy source ../policy/flask/initial sids file.

```
# This example was taken from the
# ./policy/flask/initial_sids file and declares some
# of the initial SIDs:
#
sid kernel
sid security
sid unlabeled
sid fs
```

4.16.2 sid context Statement

The sid context statement is used to add an initial security context to the SID that is used when SELinux initialises, or as a default if an object is not labeled correctly.

sid sid_id context

Where:

sid	The sid keyword.
sid_id	The previously declared sid identifier.
context	The initial security context associated with the SID. Note that there is no terminating ';'.

The statements are valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	Yes	No
Conditional Policy (if) Statement	optional Statement	<u>require Statement</u>
No	No	No

Examples:

```
# These statements add an initial security context to an object
# that is used when SELinux initialises or as a default if a
# context is not available or labeled incorrectly.
#
# This one is from a targeted policy:
sid unlabeled system_u:object_r:unlabeled_t
```

This one is from an MLS policy. Note that the security level # is set to SystemHigh as it may need to label any object in # the system.

sid unlabeled system_u:object_r:unlabeled_t:s15:c0.c255

5. The Reference Policy

5.1 Introduction

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

This section details how the Reference Policy is:

- 1. Constructed and types of policy builds supported.
- 2. Installation as a full Reference Policy source or as Header files.
- 3. Modifying the configuration files to build new policies.
- 4. Adding new modules to the build.

5.1.1 Notebook Reference Policy Information

This section makes use of the F-12 distribution that is built from the standard <u>Reference Policy VERSION=20090730</u>⁵⁴. This is modified and distributed by Red Hat as the following RPM:

```
selinux-policy-3.6.32-103.fc12.src.rpm<sup>55</sup>
```

This core source code is then used to build various policy RPMs that are distributed by Red Hat as:

selinux-policy-3.6.32-103.fc12.noarch - Contains the SELinux
/etc/selinux/config file, man pages and the 'Policy Header' development
environment that is located at /usr/share/selinux/devel

selinux-policy-doc-3.6.32-103.fc12.noarch - Contains the
html policy documentation that is located at /usr/share/doc/selinuxpolicy-3.6.32/html

selinux-policy-minimum-3.6.32-103.fc12.noarch

selinux-policy-mls-3.6.32-103.fc12.noarch

selinux-policy-targeted-3.6.32-103.fc12.noarch

These three rpms contain policy configuration files and the packaged policy modules (*.pp). These will be used to build the particular policy type in /usr/share/selinux/<policy_name> and the install process will then install the policy in the /etc/selinux/<policy_name> directory.

⁵⁴ The full source code and details are at the following site: <u>http://oss.tresys.com/projects/refpolicy</u>.

⁵⁵ This RPM can be obtained from the <u>http://koji.fedoraproject.org</u> web site.

5.2 Reference Policy Overview

The Reference Policy can be used to build two different formats of a policy:

- 1. <u>Loadable Module Policy</u> A policy that has a base module for core services and has the ability to load / unload modules to support applications as required⁵⁶. This is now the standard used by GNU / Linux distributions.
- 2. <u>Monolithic Policy</u> A policy that has all the required policy information in a single base policy.

Each of the policy types are built using module files that define the specific modules policy as detailed in the <u>Reference Policy Module Files</u>. Note that the monolithic policy is built using the the same module files, however they are all assembled into a single 'base' source file.

The Reference Policy is now used by all major distributions of SELinux, however each distribution makes its own specific changes to support their 'version of the Reference Policy' (as this section should show as the Red Hat F-12 policy distribution has a slightly different build to the standard <u>Reference Policy</u> <u>VERSION=2.20090730</u>).

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 <u>Eclipse</u> 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 <u>Reference Policy Source</u> section describes the source and the <u>Installing and Building the Reference Policy Source</u> 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 <u>Reference Policy Headers</u> section describes how these are installed and used to build modules.

The policy header files for F-12 are distributed in a number of rpms as follows:

selinux-policy-3.6.32-103.fc12.noarch – This package contains the SELinux /etc/selinux/config file, man pages and the

⁵⁶ These can be installed by system administrators as required. The dynamic loading / unloading of policies as applications are loaded is not yet supported.

'Policy Header' development environment that is located at /usr/share/selinux/devel

selinux-policy-doc-3.6.32-103.fc12.noarch - This
package contains the html policy documentation that is located at
/usr/share/doc/selinux-policy-3.6.32/html

selinux-policy-minimum-3.6.32-103.fc12.noarch

selinux-policy-mls-3.6.32-103.fc12.noarch

selinux-policy-targeted-3.6.32-103.fc12.noarch

These three packages contain policy configuration files and policy modules (*.pp files) for the particular policy type to be installed.

These files are used to build the policy type in /usr/share/selinux/<policy_name> and then install the policy in the /etc/selinux/<policy_name> directory.

Normally only one policy would be installed and active, however for development purposes all three can be installed.

5.2.2 Policy Functionality

As can be seen from the policies distributed with F-12 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 5-2), for example the Red Hat policies support⁵⁷:

minimum – supports a minimal set of confined daemons within their own domains. The remainder run in the unconfined_t space.

mls – supports server based MLS systems.

targeted – supports a greater number of confined daemons and can also confine other areas and users (this targeted version also supports the older 'strict' version).

For information, the Reference Policy supports the following types (taken from the TYPE entry of the build.conf as shown in <u>Table 5-2</u>):

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.

 ${\tt mls}-{\tt supports}$ MLS labels and confines server processes.

5.2.3 Reference Policy Module Files

The reference policy modules are constructed using a mixture of <u>policy language</u> <u>statements</u>, <u>support macros</u> and <u>access interface calls</u> using three principle types of source file (note that all three must exist even if empty):

1. A private policy file that contains statements required to enforce policy on the specific GNU / Linux service being defined within the module. These files are named <module_name>.te.

⁵⁷ Note that Red Hat pre-configure MCS support within all their policies.

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 <u>Module Expansion</u> <u>Process</u> 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 5.1 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 the <u>interface</u> 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 <u>template Macro</u> 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 <u>file_contexts</u> 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 <u>SELinux Policy Language</u> section.
- 2. Reference Policy macros that are defined in the <u>Reference Policy Macros</u> 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, 1.4.0)
*****
# Declarations
#
type ada_t;
type ada exec t;
application_domain(ada_t, ada_exec_t)
role system r types ada t;
*****
# Local policy
#
allow ada_t self:process { execstack execmem };
userdom use 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 access.
## </summary>
## </param>
#
interface(`ada domtrans',`
  gen_require(
     type ada_t, ada_exec_t;
  ')
  corecmd search bin($1)
  domtrans_pattern($1, ada_exec_t, ada_t)
•)
*****
## <summary>
## Execute ada in the ada domain, and
## allow the specified role the ada domain.
## </summary>
## <param name="domain">
## <summary>
## Domain allowed access.
## </summary>
## </param>
## <param name="role">
```

```
## <summary>
## The role to be allowed the ada domain.
## </summary>
## </param>
## <param name="terminal">
## <summary>
## The type of the terminal allow the ada domain to use.
## </summary>
## </param>
#
interface(`ada run',`
  gen require (
     type ada t;
   •)
  ada domtrans($1)
  role $2 types ada_t;
•)
```

ada.fc file contents:

```
#
#
# /usr
#
/usr/bin/gnatbind -- gen_context(system_u:object_r:ada_exec_t,s0)
/usr/bin/gnatls -- gen_context(system_u:object_r:ada_exec_t,s0)
/usr/bin/gnatmake -- gen_context(system_u:object_r:ada_exec_t,s0)
/usr/libexec/gcc(/.*)?/gnat1 -- gen_context(system_u:object_r:ada_exec_t,s0)
```

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 F-12 can be found in the following locations:

Distributed as Policy Headers - /usr/share/doc/selinux-policy-<version>/html. Where <version> is the version number of the Red Hat release, for the build used in this Notebook the location is:

```
/usr/share/doc/selinux-policy-3.6.32/html
```

Distributed as Policy Source - <location>/src/policy/doc/html. Where <location> is the location of the installed source after make install-src has been executed as described in the <u>Installing The Reference</u> <u>Policy Source</u> section. The documentation can then be generated using make html, where for the build used in this Notebook the location is:

/etc/selinux/targeted-103/src/policy/doc/html

Figure 5.1 shows an example screen shot of the documentation produced for the ada module interfaces.

e	Security Enhanced	Linux Reference Policy - Mozilla Firefox		_ + ×
<u>F</u> ile <u>E</u> dit <u>V</u> iew Hi <u>s</u> tory <u>B</u> ookmar	ks <u>T</u> ools <u>H</u> elp			2.5
🜪 🔶 ~ 😂 🔕 🖀 💽 file	:///etc/selinux/targete	d-70/src/policy/doc/html/apps_ada.html	☆ 🗸 🕻 🗸 Google	٩
📷 Most Visited∽ 🛛 💿 Release Notes	💼 Fedora Project 🗸	💼 Red Hat 🗸 🛛 🛅 Free Content 🗸		
- ada Modu	le: ada			^
- awstats Interface	S			
- calamaris Descu	ription:			
- cdrecord	iption.			
- ethereal GNA	T Ada95 compiler			
- evolution				
- games Interf	aces:			
- gift				1
- gitosis	a_domtrans(domain	1		
gitterine				
and the second	Summary			
Exe	cute the ada program in the	ada domain.		
- livecd Para	Parameters			
- loadkeys	Parameter:	Description:		
- lockdev	domain	Domain allowed access.		
- mono			1	
- mozilla				
- mplayer				
- nsplugin ad	a_run(domain , ro	le , terminal)		
- openoffice Sum	- openoffice Summary			
- podsleuth Exe				
- ptchown Para	- ptchown Parameters Parameters			
- qemu	Parameter:	Description:	1	
- rssh				
- sambagui	domain Domain	allowed access.		
- screen - slocate	role The role to be allowed the ada domain.			
- thunderbird	terminal The type	e of the terminal allow the ada domain to use.		
- tutimo			1	~

Figure 5.1: Example Documentation Screen Shot

5.3 Reference Policy Source

This section will explain the source layout and configuration files, with the actual installation and building covered in the <u>Installing and Building the Reference Policy</u> <u>Source</u> section.

The source has a README file containing information on the configuration and installation processes that has been used within this section (and updated with the authors comments as necessary). There is also a VERSION file that contains the Reference Policy release date which can be used to obtain the original source from the repository located at:

http://oss.tresys.com/projects/refpolicy

5.3.1 Source Layout

Figure 5.2 shows the layout of the reference policy source tree, that once installed would be located at:

/etc/selinux/<policy name>/src/policy

The following sections detail the source contents:

- <u>Reference Policy Files and Directories</u> Describes the files and their location.
- <u>Source Configuration Files</u> Details the contents of the build.conf and modules.conf configuration files.
- Source Installation and Build Make Options Describes the make targets.
- <u>Modular Policy Build Process</u> Describes how the various source files are linked together to form a base policy module (base.conf) during the build process.

The <u>Installing and Building the Reference Policy Source</u> section then describes how the initial source is installed and configured to allow a version of the F-12 targeted policy to be built.

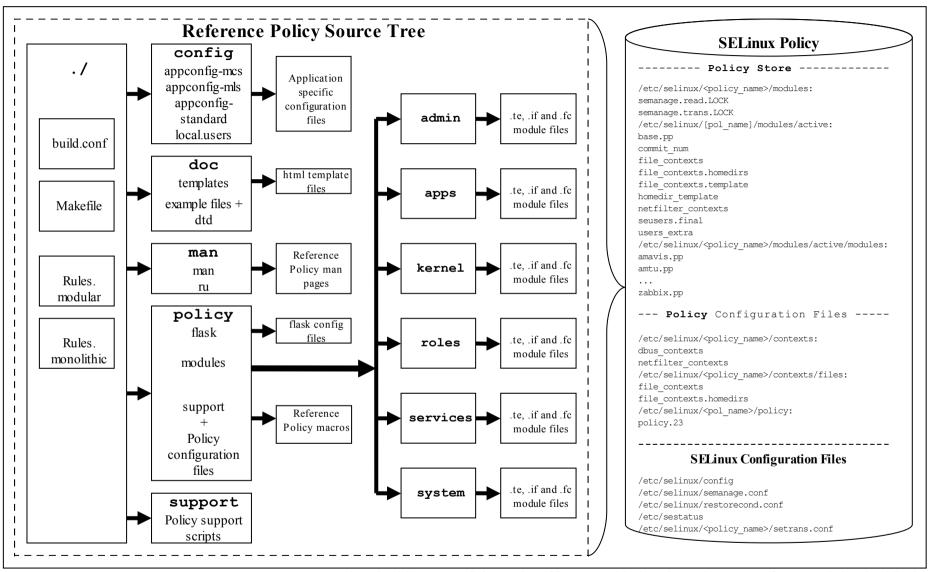


Figure 5.2: The Reference Policy Source Tree – *When building a modular policy files are added to the policy store, for monolithic builds the policy store is not used.*

5.3.2 Reference Policy Files and Directories

<u>Table 5-1</u> 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 <u>Source Configuration Files</u> 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 <u>Modular Policy Build Structure</u> section.

File / Directory Name	Comments
Makefile	General rules for building the policy.
Rules.modular	Makefile rules specific to building loadable module policies.
Rules.monolithic	Makefile rules specific to building monolithic policies.
build.conf	Options which influence the building of the policy, such as the policy type and distribution. This file is described in the <u>Reference Policy Build Options - build.conf</u> section.
config/appconfig- <type></type>	Application configuration files for all configurations of the Reference Policy where <type> is taken from the build.conf TYPE entry that are currently: standard, MLS and MCS). These files are used by SELinux-aware programs and described in the <u>SELinux Configuration</u> <u>Files</u> section.</type>
config/local.users	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 F-12 modular policy build.
doc/html/*	When make html has been executed, contains the in-policy XML documentation, presented in web page form
doc/policy.dtd	The doc/policy.xml file is validated against this DTD.
doc/policy.xml	This file is generated/updated by the conf and html make targets. It contains the complete XML documentation included in the policy.
doc/templates/*	Templates used for documentation web pages.
support/*	Tools used in the build process.
policy/flask/initial_sids	This file has declarations for each initial SID.
	The file usage in policy generation is described in the <u>Modular Policy Build Structure</u> section.
policy/flask/security_classes	This file has declarations for each security class.
	The file usage in policy generation is described in the <u>Modular Policy Build Structure</u> section.
policy/flask/access_vectors	This file defines the access vectors. Common prefixes for access vectors may be defined at the

File / Directory Name	Comments
	beginning of the file. After the common prefixes are defined, an access vector may be defined for each security class.
	The file usage in policy generation is described in the <u>Modular Policy Build Structure</u> section.
policy/modules/*	Each directory represents a layer in Reference Policy all of the modules are contained in one of these layers. The files present are:
	metadata.xml - describes the layer.
	<module_name>.te, .if & .fc - contains policy source as described in the <u>Reference Policy</u> <u>Module Files</u> section.</module_name>
	The file usage in policy generation is described in the Modular Policy Build Structure section.
policy/booleans.conf	This file is generated/updated by the conf make target. It contains the booleans in the policy, and their default values. If tunables are implemented as booleans, tunables will also be included. This file will be installed as the /etc/selinux/NAME/booleans file (note that this is not true for F-12 or any system that implements the modular policy - see the <u>Booleans</u> , <u>Global Booleans and Tunable Booleans</u> section).
	This file is also included in the F-12 source updates as described in the <u>Installing and Building the Reference</u> <u>Policy Source</u> section.
policy/constraints	This file defines additional constraints on permissions in the form of boolean expressions that must be satisfied in order for specified permissions to be granted. These constraints are used to further refine the type enforcement rules and the role allow rules. Typically, these constraints are used to restrict changes in user identity or role to certain domains. (Note that this file does not contain the MLS / MCS constraints as they are in the mls and mcs files
	described below). The file usage in policy generation is described in the
	Modular Policy Build Structure section.
policy/global_booleans	This file defines all booleans that have a global scope, their default value, and documentation. See the <u>Booleans, Global Booleans and Tunable Booleans</u> section.
policy/global_tunables	This file defines all tunables that have a global scope, their default value, and documentation. See the <u>Booleans, Global Booleans and Tunable Booleans</u> section.
policy/mcs	This contains information used to generate the sensitivity, category, level and mlsconstraint statements used to define the MCS configuration. The file usage in policy generation is described in the Modular Policy Puild Structure section
policy/mls	Modular Policy Build Structure section. This contains information used to generate the
Porrey/mrs	

File / Directory Name	Comments
	sensitivity, category, level and mlsconstraint statements used to define the MLS configuration.
	The file usage in policy generation is described in the <u>Modular Policy Build Structure</u> section.
policy/modules.conf	This file contains a listing of available modules, and how they will be used when building Reference Policy.
	This file is described in the <u>Reference Policy Build</u> <u>Options - modules.conf</u> section, it is also updated by the F-12 source updates as described in the <u>Installing</u> <u>and Building the Reference Policy Source</u> 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 <u>Modular Policy Build Structure</u> section.
policy/rolemap	This file contains prefix and user domain type that corresponds to each user role. The contents of this file will be used to expand the per-user domain templates for each module.
	Note this is not used in the Reference Policy (commented out in makefiles).
policy/users	This file defines the users included in the policy. The file usage in policy generation is described in the <u>Modular Policy Build Structure</u> section.
policy/support/*	Reference Policy support macros. These are described in the <u>Reference Policy Macros</u> section.
securetty_types	These files are not part of the standard distribution but
setrans.conf	is added by the F-12 source updates as described in the <u>Installing and Building the Reference Policy Source</u> section.

 Table 5-1: 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

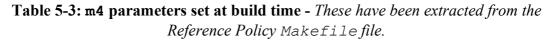
This file defines the policy type to be built that will influence its name and where the source will be located once it is finally installed. It also configures the MCS / MLS sensitivity and category maximum values. An example file content is shown in the Installing and Building the Reference Policy Source section where it is used to install and then build the policy.

<u>Table 5-2</u> explains the fields that can be defined within this file, however there are a number of m4 macro parameters that are set up when this file is read by the build process makefiles. These definitions are shown in <u>Table 5-3</u> and are also used within the module source files to control how the policy is built with examples shown in the <u>ifdef/ifndefParameters</u> section.

Option	Туре	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 standard, mls, and mcs. For a type enforcement only system, set standard. This optionally enables multi-level security (MLS) or multi-category security (MCS) features. This option controls enable_mls, and enable_mcs policy blocks.
NAME	String (optional)	Sets the name of the policy; the NAME is used when installing files to e.g., /etc/selinux/NAME and /usr/share/selinux/NAME. If not set, the policy type field (TYPE) is used. The policy built under this directory is also controlled by
		the modules.conf that is described in the <u>Reference</u> <u>Policy Build Options – policy/modules.conf</u> section.
DISTRO	String (optional)	Enable distribution-specific policy. Available options are redhat, rhel4, gentoo, debian, and suse. This option controls distro_redhat, distro_rhel4, distro_suse 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 allowed, denied, or the policy loading can be rejected. See the <u>SELinux Filesystem</u> for more details.
DIRECT_INITRC	Boolean (y n)	If 'y' sysadm will be allowed to directly run init scripts, instead of requiring the run_init tool. This is a build option instead of a tunable since role transitions do not work in conditional policy. This option controls direct_sysadm_daemon policy blocks.
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 policy/constraints file. Note Version 1 of the Reference Policy did not have this entry and defaulted to Role Based Access Control.
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 5-2: build.	conf Entries
-------------------	--------------

m4 Parameter Name in Makefile	From build.conf entry	Comments
enable_mls	TYPE	Set if MLS policy build enabled.
enable_mcs	TYPE	Set if MCS policy build enabled.
enable_ubac	UBAC	Set if UBAC set to 'y'.
mls_num_sens	MLS_SENS	The number of MLS sensitivities configured.
mls_num_cats	MLS_CATS	The number of MLS categories configured.
mcs_num_cats	MCS_CATS	The number of MCS categories configured.
distro_\$(DISTRO)	DISTRO	The distro name or blank.
direct_sysadm_daemon	DIRECT_INITRC	If DIRECT_INITRC entry set to 'y'.
hide_broken_symtoms		This is set up in the Makefile and can be used in modules to hide errors with dontaudit rules (or even allow rules).



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 comments⁵⁸) is:

<module_name> = base | module | off

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).

⁵⁸ The comments are also important as they form part of the documentation when it is generated by the make html target.

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 writer to decide which modules are in the base and those that are loadable, however there are some modules that MUST be in the base module. To highlight this there is a special entry at the start of the modules interface file (.if) that has the entry <required val="true"> as shown below (taken from the kernel.if file):

```
## <summary>
## Policy for kernel threads, proc filesystem,
## and unlabeled processes and objects.
## </summary>
## <required val="true">
## This module has initial SIDs.
## </required>
```

The modules.conf file will also reflect that a module is required in the base by adding a comment 'Required in base' when the make conf target is executed (as all the .if files are checked during this process and the modules.conf file updated).

```
# Layer: kernel
# Module: kernel
# Required in base
#
# Policy for kernel threads, proc filesystem, and unlabeled processes and
objects.
#
kernel = base
```

There are 13 modules in the F-12 reference policy source marked as required and are shown in <u>Table 5-4</u>.

Layer	Module Name	Comments	
system	application	Policy for user executable applications. All the interface calls start with 'application_'.	
system	setrans	Policy for the SELinux MLS/MCS label translation service. All the interface calls start with 'setrans_'.	
kernel	corecommands	Core policy for shells, and generic programs in: /bin,/sbin,/usr/bin, and/usr/sbin. The .fc file sets up the labels for these items. All the interface calls start with 'corecmd_'.	
kernel	corenetwork	Policy controlling access to network objects and also contains the initial SIDs for these. The .if file is large and automatically generated. All the interface calls start with 'corenet_'.	
kernel	devices	This module creates the device node concept and provides the	

Layer	Module Name	Comments	
		 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: the device directories containing device nodes. device nodes as a group 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: The concept of different file types including basic files, mount points, tmp files, etc. Access to groups of files and all files. Types and interfaces for the basic filesystem layout (/, /etc, /tmp, /usr, etc.). Contains the file initial SID. All the interface calls start with 'files_'. 	
kernel	filesystem	Contains the policy for filesystems and the initial SID. All the interface calls start with 'fs_'.	
kernel	kernel	Contains the policy for kernel threads, proc filesystem, and unlabeled processes and objects. This module has initial SIDs. All the interface calls start with 'kernel_'.	
kernel	mcs	Policy for Multicategory security. The .te file only contains attributes used in MCS policy. All the interface calls start with 'mcs_'.	
kernel	mls	Policy for Multilevel security. The .te file only contains attributes used in MLS policy. All the interface calls start with 'mls_'.	
kernel	selinux	Contains the policy for the kernel SELinux security interface (selinuxfs). All the interface calls start with 'selinux_'.	
kernel	terminal	Contains the policy for terminals. All the interface calls start with 'term_'.	

Table 5-4: 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 <u>Installing and Building the Reference Policy Source</u> section, the 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 5-5 describes the general make targets, Table 5-6 describes the modular policy make targets and Table 5-7 describes the monolithic policy make targets.

Make Target	Comments	
install-src	Install the policy sources into /etc/selinux/NAME/src/policy, where NAME is defined in the build.conf file. If it is not defined, then TYPE is used instead. If a build.conf does not have the information, then the Makefile will default to the current entry in the /etc/selinux/config file or default to refpolicy. A pre- existing source policy will be moved to /etc/selinux/NAME/src/policy.bak.	
conf	Regenerate policy.xml, and update/create modules.conf and booleans.conf. 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.	
clean	Delete all temporary files, compiled policies, and file_contexts. Configuration files are left intact.	
bare	Do the clean make target and also delete configuration files, web page documentation, and policy.xml.	
html	Regenerate policy.xml and create web page documentation in the doc/html directory.	
install-appconfig	Installs the appropriate SELinux-aware configuration files.	

Table 5-5: General Build Make Targets

Make Target	Comments	
base	Compile and package the base 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 MODULENAME Reference Policy module.	
all	Compile and package the base module and all Reference Policy modules configured to be built as loadable modules.	
install	Compile, package, and install the base module and Reference Policy modules configured to be built as loadable modules.	
load	Compile, package, and install the base 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 /usr/share/selinux/NAME. The headers are sufficient for building a policy module locally, without requiring the complete Reference Policy sources. The build.conf settings for this policy configuration should be set before using this target.	

Table 5-6: Modular Policy Build Make Targets

Make Target	Comments	
policy	Compile a policy locally for development and testing. This is the default target for monolithic policies.	
install	Compile and install the policy and file contexts.	
load	Compile and install the policy and file contexts, then load the policy.	
enableaudit	Remove all dontaudit rules from policy.conf.	
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 5-7: 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:

booleans.conf	This file is generated / updated by make conf, and contains all the booleans in the policy with their default values. If tunable and global booleans are implemented then these are also included. This file can also be delivered as a part of the reference policy source as shown in the <u>Installing and Building the Reference Policy Source</u> section. This is generally because other default values are used for booleans and not those defined within the modules themselves (i.e. distribution specific booleans). When the make install is executed, then this file will be used to set the default values. Note that if booleans are updated locally then the policy store will contain a <u>booleans.local</u> file. In SELinux enabled systems that support the policy store features (modular policies) this file is not installed as /etc/selinux/NAME/booleans.	
global_booleans	These are booleans that have been defined in the global_tunables file using the <u>gen_bool</u> macro. They are normally booleans for managing the	
	overall policy and currently consist of the following (where the default values are false):	
	secure_mode	
	secure_mode_insmod	
	secure_mode_policyload	

global_tunables	These are booleans that have been defined in module files using the <u>gen_tunable</u> macro and added to the global_tunables file by make conf. The <u>tunable_policy</u> macros are defined in each module where policy statements or interface calls are required. They are booleans for managing specific areas of policy that are global in scope. An example is allow_execstack that will allow all processes running in unconfined t to make their stacks
	executable.

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 macros⁵⁹ in the source modules will be expanded to form actual policy language statements as described in the <u>SELinux</u> Policy Language section. Figure 5.3 shows these temporary files that are used to form the base.conf⁶⁰ 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.

Base Policy Component Description	Policy Source File Name (relative to ./policy/policy)	./policy/tmp File Name
The object classes supported by the kernel.	flask/security_classes	pre_te_files.conf
The initial SIDs supported by the kernel.	flask/initial_sids	
The object class permissions supported by the kernel.	flask/access_vectors	
This is either the expanded mls or mcs file depending on the type of policy being built.	mls or mcs	
These are the policy capabilities that can be configured / enabled to support the policy.	policy_capabilities	
This area contains all the attribute, bool, type and typealias	<pre>modules/*/*.te modules/*/*.if</pre>	all_attrs_types.conf

The individual loadable modules are built in much the same way as shown in <u>Figure 5.4</u>.

⁵⁹ These are explained in the <u>Reference Policy Macros</u> section.

⁶⁰ The base.conf gets built for modular policies and a policy.conf file gets built for a monolithic policy.

Base Policy Component Description	Policy Source File Name (relative to ./policy/policy)	./policy/tmp File Name
statements extracted from the *.te and *.if files that form the base module.		
Contains the global and tunable bools extracted from the conf files.	<pre>global_bools.conf global_tunables.conf</pre>	global_bools.conf
Contains the rules extracted from each of the modules .te and .if files defined in the modules.conf file as 'base'.	base modules	only_te_rules.conf
Contains the expanded users from the users file.	users	all_post.conf
Contains the expanded constraints from the constraints file.	constraints	
Contains the default SID labeling extracted from the *.te files.	modules/*/*.te	
Contains the fs_use_xattr, fs_use_task, fs_use_trans and genfscon statements extracted from each of the modules .te and .if files defined in the modules .conf file as 'base'.	<pre>modules/*/*.te modules/*/*.if</pre>	
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'.	<pre>modules/*/*.te modules/*/*.if</pre>	
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
These are the commands used to compile, link and load the base policy module: 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 The 'NAME' is that defined in the build.conf file.		

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.	<pre>modules/*/<module_name>.te modules/*/<module_name>.if</module_name></module_name></pre>	<module_name>.tmp</module_name>
For each module defined as 'module' in the modules.conf configuration file, an object module is produced from executing the checkmodule command	tmp/ <module_name>.tmp</module_name>	<module_name>.mod</module_name>

shown below.		
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.</module_name>	<pre>modules/*/<module_name>.fc</module_name></pre>	base.fc.tmp
This command is used to compile each module: checkmodule tmp/ <module_name>.tmp -o tmp/<module_name>.mod</module_name></module_name>		
Each module is packaged and loaded with the base module using the following commands: 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 The 'NAME' is that defined in the build.conf file.		

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 explains how to install the F-12 reference policy source that is distributed by Red Hat (however the same principle is followed for the source taken directly from the <u>Tresys repository</u>, except that it will not build a compatible policy to that discussed in this section).

Any F-12 policy source rpm will suffice and can be obtained from the <u>http://koji.fedoraproject.org</u> web site, however it is assumed that the source rpm is:

```
selinux-policy-3.6.32-103.fc12.src.rpm
```

The objective of this exercise is to show that the policy built from the above source rpm is an exact replica of the targeted policy distributed as header files in the F-12 targeted rpm:

selinux-policy-targeted-3.6.32-103.fc12.noarch.rpm

Note that there is a good overview of rebuilding the source policy at Dan Walsh's site:

http://danwalsh.livejournal.com/2009/02/16/

5.4.1 Installation and Configuration

Install the source by:

rpm -Uvh selinux-policy-3.6.32-103.fc12.src.rpm

The source will be installed in the users home directory under ./rpmbuild/SOURCES where the serefpolicy-3.6.32.tgz will need to be unpacked:

```
cd $HOME/rpmbuild/SOURCES
tar -xzf serefpolicy.tgz
```

The SOURCES directory contents will then look like this:

booleans-minimum.conf	booleans-mls.conf	booleans-olpc.conf
booleans-targeted.conf	config.tgz	customizable_types
Makefile.devel	modules-minimum.conf	modules-mls.conf
modules-olpc.conf	modules-targeted.conf	policy-20100106.patch
policy-F12.patch	policygentool	securetty_types-minimum
securetty_types-mls	securetty_types-olpc	securetty_types-targeted
serefpolicy-3.6.32	serefpolicy-3.6.32.tgz	setrans-minimum.conf
setrans-mls.conf	setrans-olpc.conf	setrans-targeted.conf
users-minimum	users-mls	users-olpc
users-targeted		

The files with minimum, targeted, mls and olpc within their names are the specific configuration files used within the Reference Policy for that particular Red Hat policy type.

The latest patches now need to be applied to the source tree as follows:

```
patch -p0 <policy-F12.patch
patch -p0 <policy-20100106.patch</pre>
```

The config.tgz is Red Hat's updated configuration files this will need to be unpacked and replace the original set of files:

```
# Unpack the archive:
tar -xzf config.tgz
# move to source directory:
cd serefpolicy-3.6.32
# save the old files:
mv config config.org
# and copy over the new Red Hat files
cp -r ../config config
```

As the 'targeted' policy is being built, the files shown in <u>Table 5-8</u> left hand column need to be copied to the location and named as shown in the right hand column.

Configuration File:	Installed as Reference Policy Configuration File:	
booleans-targeted.conf	./serefpolicy-3.6.32/policy/booleans.conf	
customizable_types	<pre>./serefpolicy-3.6.32/config/appconfig-mcs/ customizable_types</pre>	
modules-targeted.conf	./serefpolicy-3.6.32/policy/modules.conf	
securetty_types-targeted	<pre>./serefpolicy-3.6.32/config/appconfig-mcs/ securetty_types</pre>	
setrans-targeted.conf	<pre>./serefpolicy-3.6.32/config/appconfig-mcs/ setrans.conf</pre>	
users-targeted.conf	./serefpolicy-3.6.32/policy/users	

Table 5-8: Red Hat specific policy configuration files – This example builds a 'targeted' policy.

The serefpolicy-3.6.32 directory will now contain the source code with the latest patches for this release (3.6.32-103) of the Red Hat Reference Policy and the correct configuration files for a targeted policy.

The ./serefpolicy-3.6.32/build.conf must now be modified to allow the source to be installed in its final location and have the correct parameters set for the build. The entries that need to be updated in the build.conf file are highlighted below⁶¹:

```
# Policy build options
# Policy version
# By default, checkpolicy will create the highest version policy it supports.
# Setting this will override the version. This only has an effect for
# monolithic policies.
#OUTPUT POLICY = 18
# Policy Type
# standard, mls, mcs. Note Red Hat always build the MCS Policy Type
# as their 'targeted' version.
TYPE = mcs
# Policy Name
# If set, this will be used as the policy name. Otherwise the policy type
# will be used for the name. This entry is also used by the
# 'make install-src' process
# to copy the source to the /etc/selinux/targeted-103/src/policy directory.
NAME = targeted-103
# 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
```

⁶¹ The README file in this directory contains helpful information on installation of the source, headers, documentation etc. The only point the README will not cover are the Red Hat specific configuration files that need to be copied over as shown in <u>Table 5-8</u>.

```
# the policy. The permissions can either be allowed, denied, or the policy
# loading can be rejected.
# allow, deny, and reject are current options. Red Hat 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 n here will build a loadable module policy.
MONOLITHIC = n
# User-based access control (UBAC)
# Enable UBAC for role separations. Note Red Hat disable UBAC.
UBAC = n
# Number of MLS Sensitivities
 The sensitivities will be s0 to s(MLS SENS-1). Dominance will be in increasing
# numerical order with s0 being lowest.
MLS SENS = 16
# Number of MLS Categories. Note Red Hat use 1024 categories for MLS and MCS.
# The categories will be c0 to c(MLS CATS-1).
MLS CATS = 1024
# Number of MCS Categories
# The categories will be c0 to c(MLS CATS-1).
MCS CATS = 1024
# Set this to y to only display status messages during build.
OUIET = n
```

The policy source is now in a position to be installed at its default location that will be derived from the NAME = targeted-103 entry and will therefore be located at:

```
/etc/selinux/targeted-103/src/policy
```

5.4.2 Building the targeted Policy Type

From the ./serefpolicy-3.6.32 directory run the following command:

```
make install-src
```

This will copy the source code to its final location making any directories required.

Once the copy process is complete the policy can be built and the modules loaded into the policy store⁶² by running the following commands:

```
# Go to the source location:
cd /etc/selinux/targeted-103/src/policy
```

```
# To ensure a clean source build:
make clean
```

```
# Build the policy modules and load into the policy store:
make load
```

⁶² Note that the term 'load' is not loading the policy as the active policy, but just building the base policy + the modules and installing them ready to be activated if required

```
# Finally copy over two files that are not automatically
# managed by the build process. These are held in the
# config/appconfig-mcs directory.
#
cp config/appconfig-mcs/setrans.conf /etc/selinux/targeted-103
# Need to over-write the old version of this file with this one:
cp config/appconfig-mcs/customizable_types
   /etc/selinux/targeted-103/contexts
```

The policy will now be built as a targeted policy that will be an exact copy of the policy distributed in the following rpm:

```
selinux-policy-targeted-3.6.32-103.fc12.noarch.rpm
```

5.4.3 Checking the Build

Now that the targeted policy has been built, the policy binary file can be compared to the one that is distributed and built by the following rpm:

```
selinux-policy-targeted-3.6.32-103.fc12.noarch
```

The binary files sizes of both policies should be 4,776,004 bytes.

```
ls -l /etc/selinux/targeted/policy
-rw-r--r- root root 4776004 <date+time> policy.24
ls -l /etc/selinux/targeted-103/policy
-rw-r--r-- root root 4776004 <date+time> policy.24
```

Note that the binaries would not be an exact comparison due to time stamps etc., therefore the SETools sediffx utility should be run against the two binary policies⁶³ which should show that they are the same and give the results shown in Figure 5.5.

⁶³ Be aware that comparing these two policies on a low specification machine will take hours. It is best to select a few items for comparison first.

🖚 sediffx - [Policy file: /etc/selinux/t 🗕 🗆 🗙	🛤 sediffx - [Policy file: /etc/selinux/t 🗕 🗆 🗙	
<u>File E</u> dit <u>T</u> ools <u>H</u> elp	<u>F</u> ile <u>E</u> dit <u>T</u> ools <u>H</u> elp	
Open Policies Run Diff Remap Types	Open Policies Run Diff Remap Types	
Differences Original Policy Modified Policy	Differences Original Policy Modified Policy	
Policy Statistics Source	Policy Statistics Source	
Policy: /etc/selinux/targeted/policy/policy.24 Policy Version & Type: v.24 (binary, mls)	Policy: /etc/selinux/targeted-103/policy/policy.24 Policy Version & Type: v.24 (binary, mls)	
Number of Classes and Permissions: Object Classes: 77 Common Classes: 5 Permissions: 229	Number of Classes and Permissions: Object Classes: 77 Common Classes: 5 Permissions: 229	
Number of Types and Attributes: Types: 2800 Attributes: 234	Number of Types and Attributes: Types: 2800 Attributes: 234	
Number of Rules: allow: 236875 auditallow: 39 dontaudit 112769 neverallow: not calculated type_change: 38 type_member: 57 type_transition: 9942	Number of Rules: allow: 236875 auditallow: 39 dontaudit 112769 neverallow: not calculated type_change: 38 type_member: 57 type_transition: 9942	
Number of Roles: 11	Number of Roles: 11	
Number of RBAC Rules: allow: 197 role_transition 0	Number of RBAC Rules: allow: 197 role_transition 0	
Number of Users: 8	Number of Users: 8	
Number of Booleans: 150	Number of Booleans: 150	
Total Differences: 0	Total Differences: 0	

Figure 5.5: The two 'targeted' policies should be the same using sediffx

5.4.4 Running with the new Policy

To run the system using the new targeted-103 build edit the /etc/selinux/config file entry to read SELINUXTYPE=targeted-103, and then run the following commands:

```
touch /.autorelabel
reboot
```

During reboot, the system will be relabeled and the policy loaded (hopefully with no errors).

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 <u>Red Hat F-12 policy header</u> 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:

- Ensure the source is installed and configured as described in the <u>Installing and</u> <u>Building the Reference Policy Source</u> section. This is because the make load (or make install) command will package all the modules as defined in the modules.conf file, producing a base.pp and the relevant .pp packages. The build process will then install these files in the /usr/share/selinux/<policy name> directory.
- 2. Execute the make install-headers command that will:
 - a) Produce a build.conf file that represents the contents of the master build.conf file and place it in the /usr/share/selinux/<policy name>/include directory.
 - b) Produce the XML documentation set that reflects the source and place it in the /usr/share/selinux/<policy_name>/include directory.
 - c) Copy a development Makefile for building from policy headers to the /usr/share/selinux/<policy_name>/include directory.
 - d) Copy the support macros .spt files to the
 /usr/share/selinux/<policy_name>/include/support
 directory.
 - e) Copy the module interface files (.if) to the relevant module directories at:

/usr/share/selinux/<policy_name>/include/modules.

The directory structure for the targeted-103 build generated above (edited for readability) would be:

```
# The policy packages:
targeted-103/abrt.pp
targeted-103/base.pp
# Build / Configuration files:
targeted-103/include/build.conf
targeted-103/include/Makefile
targeted-103/include/rolemap # Note this file is not used by F-12
# XML Documentation:
targeted-103/include/global_tunables.xml
targeted-103/include/global booleans.xml
targeted-103/include/apps.xml
targeted-103/include/roles.xml
targeted-103/include/system.xml
targeted-103/include/kernel.xml
targeted-103/include/services.xml
targeted-103/include/admin.xml
# Support Macros:
```

```
targeted-103/include/support/ipc_patterns.spt
...
# The module interface files in their relevant directories:
targeted-103/include/admin/acct.if
...
targeted-103/include/apps/ada.if
...
targeted-103/include/kernel/corecommands.if
...
targeted-103/include/roles/auditadm.if
...
targeted-103/include/services/abrt.if
...
```

5.5.2 Using the Header Files

Note that this section describes the standard Reference Policy headers, the F-12 installation is slightly different and described in the <u>Using F-12 Supplied Headers</u> 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/targeted-103/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 makes -f option as follows (assuming that the example module files are in the local directory):

make -f /usr/share/selinux/<policy name>/include/Makefile

However there is another Makefile that can be installed in the users home directory (\$HOME) that will call the master Makefile. This is located at /etc/selinux/targeted-103/src/policy/doc in the reference policy source and is called Makefile.example. This is shown below (note that it extracts the <policy name> from the SELinux config file):

```
AWK ?= gawk
NAME ?= $(shell $(AWK) -F= '/^SELINUXTYPE/{ print $$2 }'
/etc/selinux/config)
SHAREDIR ?= /usr/share/selinux
HEADERDIR := $(SHAREDIR)/$(NAME)/include
include $(HEADERDIR)/Makefile
```

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	

Make Target	Comments	
	the local and system module packages.	
xml	Build a policy.xml from the XML included with the base policy headers and any XML in the modules in the current directory.	

Table 5-9: Header Policy Build Make Targets

5.5.3 Using F-12 Supplied Headers

The F-12 distribution installs the headers in a slightly different manner as Red Hat installs:

- The packaged files under the /usr/share/selinux/<policy_name>, these files may be .pp files or .pp.bz2 depending on the version of rpm installed (later ones compressed the packages). They are installed by the selinux-policy-<policy_name>-3.6.32-103.fc12.noarch type rpms.
- The development header files installed are in the /usr/share/selinux/devel directory by the selinux-policy-3.6.32-103.fc12.noarch rpm. Red Hat also include an additional application called policygentool that allows users to generate policy by answering various questions. This tool is described in the Fedora 12 SELinux User Guide [Ref. 1]. The example modules are also in this directory and the Makefile is also slightly different to that used by the Reference Policy source.
- The documentation is supplied in the selinux-policy-doc-3.6.32-103.fc12.noarch type rpms and would be installed (for this version), in the /usr/share/doc/selinux-policy-3.6.32/html directory.

5.6 Reference Policy Support Macros

This section explains some of the support macros used to build reference policy source modules (see <u>Table 5-10</u> for the list). These macros are located at:

- ./policy/policy/support for the reference policy source.
- /usr/share/selinux/<policy_name>/include/support for reference policy installed header files.
- /usr/share/selinux/devel/support for Red Hat installed header files.

They consist of the following files:

loadable module.spt - Loadable module support.

misc macros.spt - Generate users, bools and security contexts.

mls mcs macros.spt - MLS / MCS support.

file_patterns.spt - Sets up allow rules via parameters for files and directories.

ipc_patterns.spt - Sets up allow rules via parameters for Unix domain sockets.

misc patterns.spt - Domain and process transitions.

obj perm sets.spt - Object classes and permissions.

Macro Name	Function	Macro file name	
policy_module	For adding the module statement and mandatory require block entries.	/ loadable_module.spt	
gen_require	For use in interfaces to optionally insert a require block	-	
template	Generate template interface block		
interface	Generate the access interface block		
optional_policy	Optional policy handling		
gen_tunable	Tunable declaration		
tunable_policy	Tunable policy handling		
gen_user	Generate an SELinux user	misc_macros.spt	
gen_context	Generate a security context		
gen_bool	Generate a boolean		
gen_cats	Declares categories c0 to c (N-1)	mls_mcs_macros.spt	
gen_sens	Declares sensitivities $s0$ to s (N-1) with dominance in increasing numeric order with $s0$ lowest, s (N-1) highest.		
gen_levels	Generate levels from $s0$ to $(N-1)$ with categories $c0$ to $(M-1)$	-	
mls_systemlow	Basic level names for system low and high		
mls_systemhigh			
mcs_systemlow			
mcs_systemhigh			
mcs_allcats	Allocates all categories		

Table 5-10: Support Macros described in this section

Notes:

- 1. The macro calls can be in any configuration file read by the build process and are contained in (for example) the users, mls, mcs and constraints files.
- 2. There are 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 <u>Table 5-3</u> as they are set by the initial build process with examples shown in the <u>ifdef/ifndefParameters</u> 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 <u>Module</u> <u>Expansion Process</u> section.
- 5. Be aware that spaces between macro names and their parameters are not allowed:

Allowed:

```
policy_module(ftp, 1.7.0)
```

Not allowed:

policy_module (ftp, 1.7.0)

5.6.1 Loadable Policy Macros

The loadable policy module support macros are located in the loadable_module.spt file.

5.6.1.1 policy_module Macro

This macro will add the <u>module statement</u> to a loadable module, and automatically add a <u>require Statement</u> 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:

policy_module	The policy_module macro keyword.
module_name	The module identifier that must be unique in the module layers.
version_number	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;
    ....
}
```

5.6.1.2 gen_require Macro

For use within module files to insert a require block.

The macro definition is:

```
gen_require(`require_statements`)
```

Where:

gen_require	The gen_require macro keyword.
require_statements	These statements consist of those allowed in the policy language <u>require Statement</u> .

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	Yes	No

```
# 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;
}
```

5.6.1.3 optional_policy Macro

For use within module files to insert an optional block that will be expanded by the build process only if the modules containing the access or template interface calls that follow are present. If one module is present and the other is not, then the optional statements are not included (need to check).

The macro definition is:

optional policy(`optional statements`)

Where:

optional_policy	The optional_policy macro keyword.
optional_statements	These statements consist of those allowed in the policy language <u>optional Statement</u> . However they can also be <u>interface</u> , <u>template</u> 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(ftpd_t)
    files_read_usr_files(ftpd_t)
    cron_system_entry(ftpd_t, ftpd_exec_t)
    optional_policy(`
        logrotate_exec(ftpd_t)
    ')
')
```

Expanded Macro:

This is the expanded macro from the tmp/ftp.tmp file showing # the policy language statements with both optional levels # expanded.

```
optional
##### begin corecmd exec shell(ftpd 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(ftpd_t)
##### begin files read usr files(ftpd 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(ftpd t)
###### begin cron system entry(ftpd t,ftpd exec t)
  require {
     type crond t, system crond t;
  } # end require
  allow system crond t ftpd exec t:file { getattr read execute };
  allow system_crond_t ftpd_t:process transition;
  dontaudit system_crond_t ftpd_t:process { noatsecure siginh rlimitinh };
  type_transition system_crond_t ftpd_exec_t:process ftpd_t;
   # cjp: perhaps these four rules from the old
   # domain auto trans are not needed?
  allow ftpd_t system_crond_t:fd use;
  allow ftpd_t system_crond_t:fifo file { getattr read write append ioctl
lock };
  allow ftpd t system_crond_t:process sigchld;
  allow ftpd t crond t:fifo file { getattr read write append ioctl lock };
  allow ftpd t crond t:fd use;
  allow ftpd t crond t:process sigchld;
  role system r types ftpd t;
##### end cron_system_entry(ftpd_t,ftpd_exec_t)
optional
##### begin logrotate exec(ftpd t)
     require {
         type logrotate_exec_t;
      } # end require
     allow ftpd t logrotate exec t:file { { getattr read execute ioctl } ioctl
lock execute no trans };
##### end logrotate_exec(ftpd_t)
  } # end optional 2nd level
} # end optional 1st level
```

5.6.1.4 gen tunable Macro

This macro defines booleans that are global in scope. The corresponding <u>tunable_policy</u> 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 <u>documentation</u>.

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>
## 
## Allow ftp servers to use nfs
## for public file transfer services.
## 
## </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;
```

5.6.1.5 tunable policy Macro

This macro contains the statements allowed or not depending on the value of the boolean defined by the <u>gen_tunable</u> 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 \underline{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_ftpd_use_nfs && allow_ftpd_anon_write',`
    fs_manage_nfs_files(ftpd_t)
')
```

Expanded Macro:

```
# This is the expanded macro from the tmp/ftp.tmp file.
#
if (allow_ftpd_use_nfs && allow_ftpd_anon_write) {
###### begin fs_manage_nfs_files(ftpd_t)
   require {
     type nfs_t;
   } # end require
   allow ftpd_t nfs_t:dir { read getattr lock search ioctl
   add_name remove_name write };
   allow ftpd_t nfs_t:file { create open getattr setattr read
   write append rename link unlink ioctl lock };
   ##### end fs_manage_nfs_files(ftpd_t)
   } # end if
```

5.6.1.6 interface Macro

Access interface macros are defined in the interface module file (.if) and form the interface through which other modules can call on the modules services (as shown in Figure 5.7 and described in the Module Expansion section.

Note that the comments shown in the example MUST be present as they are used to describe the function and are extracted for the <u>documentation</u>.

The macro definition is:

interface(`name`,`interface rules`)

Where:

interface	The interface macro keyword.
name	The interface identifier that should be named to reflect the module identifier and its purpose.
interface_rules	This can consist of the support macros, policy language statements or other interface calls as required to provide the service.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
No	Yes	No

Example Interface Definition:

```
# This example is from the modules/services/ftp.if module
# showing the `ftp read config' interface.
#
## <summary>
##
     Read ftpd etc files
## </summary>
## <param name="domain">
##<summary>
##
    Domain allowed access.
##</summary>
## </param>
#
interface(`ftp_read_config',`
 gen require(
    type ftpd etc t;
  1)
 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)
# ')
#
#
```

```
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 initro_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
```

5.6.1.7 template Macro

A template interface is used to help create a domain and set up the appropriate rules and statements to run an application / process. The basic idea is to set up an application in a domain that is suitable for the defined SELinux user and role to access but not others. Should a different user / role need to access the same application, another domain would be allocated (these are known as 'derived domains' as the domain name is derived from caller information).

The application template shown in the example below is for openoffice.org where the domain being set up to run the application is based on the SELinux user xguest (parameter \$1) therefore a domain type is initialised called xguest_openoffice_t, this is then added to the user domain attribute xguest_usertype (parameter \$2). Finally the role xguest_r (parameter \$3) is allowed access to the domain type xguest_openoffice_t. If a different user / role required access to openoffice.org, then by passing different parameters (i.e. user_u), a different domain would be set up.

The main differences between an application interface and a template interface are:

- An access interface is called by other modules to perform a service.
- A template interface allows an application to be run in a domain based on user / role information to isolate different instances.

Note that the comments shown in the example MUST be present as they are used to describe the function and are extracted for the <u>documentation</u>.

The macro definition is:

```
template(`name`,`template rules`)
```

Where:

template	The template macro keyword. The template identifier that should be named to reflect the module identifier and its purpose. By convention the last component is _template (e.g. ftp_per_role_template).	
name		
template_rules	This can consist of the support macros, policy language statements or interface calls as required to provide the service.	

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
No	Yes	No

```
# 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>
## 
## This template creates a derived domains which are used
## for openoffice applications.
## 
## </desc>
## <param name="userdomain prefix">
## <summary>
## The prefix of the user domain (e.g., user
## is the prefix for user_t).
## </summary>
## </param>
## <param name="user domain">
## <summary>
## The type of the user domain.
## </summary>
## </param>
## <param name="user_role">
## <summary>
## The role associated with the user domain.
## </summary>
## </param>
#
template(`openoffice_per_role_template',`
  gen_require(
     type openoffice_exec t;
   •)
  type $1_openoffice_t;
  domain type ($1 openoffice t)
  domain_entry_file($1_openoffice_t, openoffice_exec_t)
  role $3 types $1_openoffice_t;
  domain interactive fd($1 openoffice t)
```

```
userdom_unpriv_usertype($1, $1_openoffice_t)
userdom_exec_user_home_content_files($1, $1_openoffice_t)
allow $1_openoffice_t self:process { getsched sigkill execheap execmem
execstack };
allow $2 $1_openoffice_t:process { getattr ptrace signal_perms noatsecure
siginh rlimitinh };
allow $1_openoffice_t $2:tcp_socket { read write };
domtrans_pattern($2, openoffice_exec_t, $1_openoffice_t)
dev_read_urand($1_openoffice_t)
dev_read_rand($1_openoffice_t)
fs_dontaudit_rw_tmpfs_files($1_openoffice_t)
allow $2 $1_openoffice_t:process { signal sigkill };
allow $1_openoffice_t $2:unix_stream_socket connectto;
')
```

Expanded Macro:

```
# Template Interfaces are only expanded at policy compile time
# if they are called by a module that requires their services.
# This has been expanded as a part of the roles/xguest.te
# module and extracted from tmp/xguest.tmp.
optional {
###### begin openoffice_per_role_template(xguest,xguest_usertype,xguest_r)
  require {
     type openoffice_exec_t;
  } # end require
  type xguest_openoffice t; # Paremeter $1
# This is a long set of rules, therefore has been cut down.
. . . . . .
. . . .
  typeattribute xguest openoffice t xguest usertype; # Paremeter $2
  type transition xguest usertype openoffice exec t:process xguest openoffice t;
  role xguest_r types xguest_openoffice_t; # Paremeter $3
  allow xguest usertype xguest openoffice t:process { signal sigkill };
  allow xguest openoffice t xguest usertype:unix stream socket connectto;
###### end openoffice_per_role_template(xguest,xguest_usertype,xguest_r)
} # end optional
```

5.6.2 Miscellaneous Macros

These macros are in the misc macros.spt file.

5.6.2.1 gen_context Macro

This macro is used to generate a valid security context and can be used in any of the module files. Its most general use is in the .fc file where it is used to set the files security context.

The macro definition is:

```
gen context(context[,mls | mcs])
```

Where:

gen_context	The gen_context macro keyword.
context	The security context to be generated. This can include macros that are relevant to a context as shown in the example below.
mls mcs	MLS or MCS labels if enabled in the policy.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	Yes	Yes

Example Macro:

```
# This example shows gen_context being used to generate a
# security context for the security initial sid in the
# selinux.te module:
sid security gen_context(system_u:object_r:security_t:mls_systemhigh)
```

Expanded Macro:

```
# This is the expanded entry built into the base.conf source
# file for an MLS policy:
sid security system_u:object_r:security_t:s15:c0.c255
```

Example File Context .fc file:

<pre># This is from the modules/apps/gnome.fc file. Note that the # HOME_DIR and USER parameters will be entered during # the <u>file_contexts.homedirs</u> file build as described in the # modules/active/file_contexts.template File section. #</pre>
HOME DIR/.gnome2(/.*)?
<pre>gen_context(system_u:object_r:gnome_home_t,s0)</pre>
HOME_DIR/\.config/gtk*
<pre>gen_context(system_u:object_r:gnome_home_t,s0) HOME DIR/\.gconf(d)?(/.*)?</pre>
gen context(system u:object r:gconf home t,s0)
HOME DIR/\.local.*
<pre>gen_context(system_u:object_r:gconf_home_t,s0)</pre>
<pre>/tmp/gconfd-USER/.* gen_context(system_u:object_r:gconf_tmp_t,s0)</pre>

```
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
```

5.6.2.2 gen_user Macro

This macro is used to generate a valid <u>user statement</u> and add an entry in the <u>users extra</u> 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.temp late 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;
```

5.6.2.3 gen_bool Macro

This macro defines a boolean and requires the following steps:

- 1. Declare the boolean in the global booleans file.
- 2. Use the boolean in the module files with an <u>if / else statement</u> 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 <u>documentation</u>.

The macro definition is:

```
gen_bool(name,default_value)
```

Where:

gen_bool	The gen_bool macro keyword.
name	The boolean identifier.
default_value	The value true or false.

The macro is only valid in in the global_booleans file but the boolean declared can be used in the following module types:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	Yes	No

Example Macro (in global_booleans):

```
# This example is from the global_booleans file where the bool
# is declared. The comments must be present as it is used to
# generate the documentation.
#
## <desc>
## 
## Disable transitions to insmod.
## 
## </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)
  }
```

5.6.3 MLS and MCS Macros

These macros are in the mls_mcs_macros.spt file.

5.6.3.1 gen_cats Macro

This macro will generate a <u>category statement</u> for each category defined. These are then used in the base.conf / policy.conf source file and also inserted into each module by the <u>policy module Macro</u>. 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)
```

gen_cats	The gen_cats macro keyword.
<pre>mcs_num_cats mls_num_cats</pre>	These are the maximum number of categories that have been extracted from the build.conf file MCS_CATS or MLS_CATS entries and set as m4 parameters.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
na	na	na

Example Macro:

```
# This example is from the policy/policy/mls configuration file.
#
gen_cats(mls_num_cats)
```

Expanded Macro:

```
# This example has been extracted from the base.conf source
# file.
category c0;
category c1;
...
category c1023;
```

5.6.3.2 gen_sens Macro

This macro will generate a <u>sensitivity statement</u> for each sensitivity defined. These are then used in the base.conf / policy.conf source file and also inserted into each module by the <u>policy_module Macro</u>. 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;
```

5.6.3.3 gen_levels Macro

This macro will generate a <u>level statement</u> 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

```
# 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;
```

5.6.3.4 System High/Low Parameters

These macros define system high etc. as shown.

```
mls_systemlow
# gives:
s0
mls_systemhigh
# gives:
s15:c0.c1023
mcs_systemlow
# gives:
s0
mcs_systemhigh
# gives:
s0:c0.c1023
mcs_allcats
# gives:
c0.c1023
```

5.6.4 ifdef/ifndef Parameters

This section contains examples of the common ifdef/ifndef parameters that can be used in module source files.

5.6.4.1 hide broken symptoms

This is used within modules as shown in the example. The parameter is set up by the Makefile at the start of the build process.

```
# 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 };
')
```

5.6.4.2 enable_mls and enable_mcs

These are used within modules as shown in the example. The parameters are set up by the Makefile with information taken from the build.conf file at the start of the build process.

Example Macros:

```
# This example is from the modules/kernel/kernel.te module.
#
ifdef(`enable_mls',`
   role secadm_r;
   role auditadm_r;
')
```

```
# This example is from the modules/kernel/kernel.if module.
#
ifdef(`enable_mcs',`
    range_transition kernel_t $2:process $3;
    ')
ifdef(`enable_mls',`
    range_transition kernel_t $2:process $3;
    mls_rangetrans_target($1)
    ')
```

5.6.4.3 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 Referefence Policy.

The orginal method (role based access control, or RBAC) is the default for F-12 (ubac = n). The parameter is set up by the Makefile with information taken from the build.conf file at the start of the build process (ubac = $y \mid ubac = n$).

```
# 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
')
```

5.6.4.4 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:

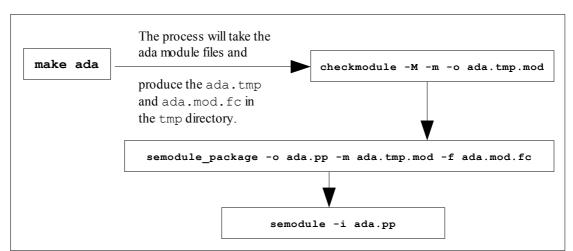
```
# This example is from the modules/system/selinuxutil.te module.
#
ifndef(`direct_sysadm_daemon',`
    ifdef(`distro_gentoo',`
        # Gentoo integrated run_init:
        init_script_file_entry_type(run_init_t)
    ')
')
```

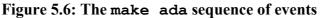
5.7 Module Expansion Process

The objective of this section is to show how the modules are expanded by the reference policy build process to form files that can then be compiled and then loaded into the policy store by using the make MODULENAME.pp target.

The files shown are those produced by the build process using the ada policy modules from the Reference Policy source tree (ada.te, ada.if and ada.fc) that are shown in the <u>Reference Policy Module Files</u> 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 5.6, and the resulting expanded code shown in Figure 5.7 and then described in the Module Expansion section.





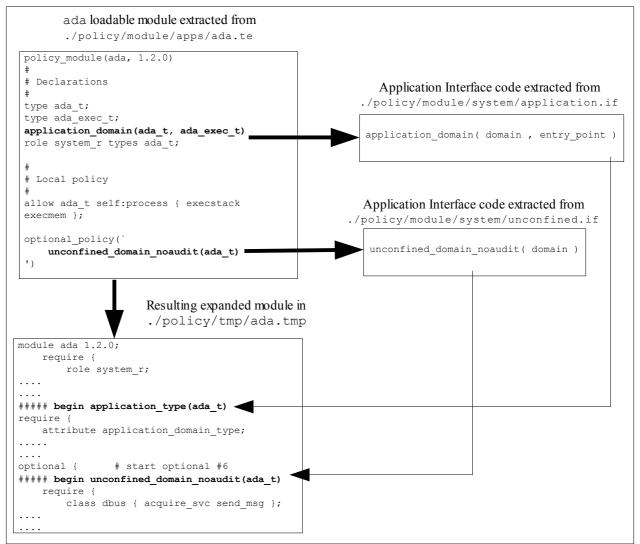


Figure 5.7: The Resulting Code - *The expanded code in the <u>Module Expansion</u> section.*

5.7.1 Module Expansion

The ada.te module is expanded as shown below. Note that the ada.if module is not expanded here. The ada.if module would only be expanded if another module calls these interfaces, where they would then be expanded into the calling module.

```
# This is the start of the ada.te file that is expanded by the build
# process. Note the following:
# 1) The macros have been expanded to transform the `ada.te' file into an
     'ada.conf' file.
# 2) The 'ada.if' file Application Interface calls have NOT been expanded
    into this file simply because the ada.te does not call them (but
    they would be expanded into any other policy module that called them).
# 3) The module calls the "application_domain(ada_t,ada_exec_t)" that is
    one of the mandatory modules that MUST be included in the base.
    Note that this then calls other Application Interface macros.
# 4) All the build information that was in the original ada.tmp build
    file have been removed for readability.
*************
# The "policy module(ada, 1.2.0)" macro has been expanded by the build process
 and the predefined 'require { }' entries are added (system r role, all
# kernel classes and the sensitivity and category statements).
module ada 1.2.0;
 require {
     role system r;
# These are the kernel class statements. There are many of them, therefore most
# have been removed for readability.
     class security { compute av compute create compute member check context
load policy compute relabel compute user setenforce setbool setsecparam
setcheckreqprot };
     class peer { recv };
     class capability2 { mac override mac admin };
# End of classes
# As this is built as an MCS policy, there is only one sensitivity.
     sensitivity s0;
     category c0;
# This would contain many lines, one for each of the 1024 category statements
# defined in the policy. These have been removed for clarity.
     category c1023;
  } # END REQUIRE
   ****
####
# Declarations
type ada t;
type ada_exec_t;
##################
              ****************
# This is the ``application_domain(ada_t, ada_exec_t)" Application Interface
# call that is expanded to policy language statements by the build process.
# Note that there are many `optional { }' statements, these have been
# marked numbered for easy reference.
```

```
#
##### begin application_type(ada_t)
require {
  attribute application domain type;
} # end require
typeattribute ada_t application_domain_type;
##### begin domain_type(ada_t)
##### begin domain_base_type(ada_t)
require {
 attribute domain;
} # end require
typeattribute ada_t domain;
##### end domain base type(ada t)
optional { # start optional #1
  ###### begin unconfined_use_fds(ada_t)
  require {
      type unconfined t;
  } # end require
  allow ada_t unconfined_t:fd use;
  ##### end unconfined use fds(ada t)
} # end optional #1
# send init a sigchld and signull
optional { # start optional #2
##### begin init_sigchld(ada_t)
  require {
     type init_t;
  } # end require
  allow ada t init t:process sigchld;
##### end init_sigchld(ada_t)
##### begin init_signull(ada_t)
  require {
     type init t;
  } # end require
  allow ada t init t:process signull;
#### end init_signull(ada_t)
} # end optional #2
# these seem questionable:
optional { # start optional #3
##### begin rpm_use_fds(ada_t)
 require {
      type rpm_t;
  } # end require
  allow ada_t rpm_t:fd use;
##### end rpm_use_fds(ada_t)
##### begin rpm read pipes(ada t)
  require {
      type rpm t;
  } # end require
  allow ada_t rpm_t:fifo_file { getattr read lock ioctl };
##### end rpm_read_pipes(ada_t)
} # end optional #3
optional { # start optional #4
##### begin selinux_dontaudit_getattr_fs(ada_t)
  require {
      type security_t;
  } # end require
  dontaudit ada_t security_t:filesystem getattr;
###### end selinux dontaudit getattr fs(ada t)
```

```
##### begin selinux_dontaudit_read_fs(ada_t)
  require {
     type security t;
  } # end require
##### begin selinux dontaudit getattr fs(ada t)
  require {
      type security t;
  } # end require
  dontaudit ada t security t:filesystem getattr;
##### end selinux dontaudit getattr fs(ada t)
  dontaudit ada_t security_t:dir { getattr search };
  dontaudit ada_t security_t:file { getattr read };
##### end selinux dontaudit read fs(ada t)
} # end optional \frac{1}{4}
optional { # start optional #5
##### begin seutil_dontaudit_read_config(ada_t)
  require {
     type selinux config t;
  } # end require
  dontaudit ada t selinux config t:dir { getattr search };
  dontaudit ada t selinux config t:file { getattr read };
###### end seutil_dontaudit_read_config(ada_t)
} # end optional #5
##### end domain type(ada t)
##### end application type (ada t)
##### begin application_executable_file(ada_exec_t)
require {
  attribute application exec type;
} # end require
typeattribute ada_exec_t application_exec_type;
###### begin corecmd executable file(ada exec t)
require {
  attribute exec_type;
} # end require
typeattribute ada_exec_t exec_type;
##### begin files_type(ada_exec_t)
require {
 attribute file type, non security file type;
} # end require
typeattribute ada_exec_t file_type, non_security_file_type;
##### end files type(ada exec t)
##### end corecmd executable file(ada exec t)
###### end application_executable_file(ada_exec_t)
##### begin domain_entry_file(ada_t,ada_exec_t)
require {
 attribute entry_type;
} # end require
allow ada_t ada_exec_t:file entrypoint;
allow ada t ada exec t:file { { getattr read execute ioctl } ioctl lock };
typeattribute ada exec_t entry_type;
##### begin corecmd_executable_file(ada_exec_t)
require {
 attribute exec type;
} # end require
typeattribute ada exec t exec type;
##### begin files type(ada exec t)
 require {
  attribute file_type, non_security_file_type;
} # end require
typeattribute ada exec t file type, non security file type;
##### end files type(ada exec t) depth: 3
```

```
##### end corecmd executable file(ada exec t)
##### end domain_entry_file(ada_t,ada_exec_t)
##### end application domain(ada t,ada exec t)
########## END application domain(ada t,ada exec t) INSERT ########
role system r types ada_t;
*****
# Local policy
# This is the only allow statement in the ada.te file:
allow ada t self:process { execstack execmem };
# The "optional policy(`unconfined domain noaudit(ada t)')" is the next
# line that expands into the lines between START and END OPTIONAL comments.
# NOTE: If the unconfined module was NOT part of the build then this optional
       policy section would not be present.
# start optional #6
optional {
##### begin unconfined domain noaudit(ada t)
  require {
     class dbus { acquire svc send msg };
      class nscd { getpwd getgrp gethost getstat admin shmempwd shmemgrp
shmemhost getserv shmemserv };
     class passwd { passwd chfn chsh rootok crontab };
  } # end require
# Use any Linux capability.
  allow ada_t self:capability { chown dac_override dac_read_search fowner fsetid
kill setgid setuid setpcap linux immutable net bind service net broadcast
net admin net raw ipc lock ipc owner sys module sys rawio sys chroot sys ptrace
sys_pacct sys_admin sys_boot sys_nice sys_resource sys_time sys_tty_config mknod
lease audit write audit control setfcap };
  allow ada t self:fifo file { create open getattr setattr read write append
rename link unlink ioctl lock };
# Transition to myself, to make get ordered context list happy.
  allow ada t self:process transition;
# Write access is for setting attributes under /proc/self/attr.
  allow ada_t self:file { getattr read write append ioctl lock };
  allow ada_t self:dir { read getattr lock search ioctl add_name remove_name
write };
# Userland object managers
  allow ada t self:nscd { getpwd getgrp gethost getstat admin shmempwd shmemgrp
shmemhost getserv shmemserv };
  allow ada t self:dbus { acquire svc send msg };
  allow ada t self:passwd { passwd chfn chsh rootok crontab };
  allow ada t self:association { sendto recvfrom setcontext polmatch };
##### begin kernel_unconfined(ada_t)
  require {
     attribute kern unconfined;
  } # end require
  typeattribute ada_t kern unconfined;
##### end kernel_unconfined(ada_t)
##### begin corenet unconfined(ada t)
  require { attribute corenet_unconfined_type;
  } # end require
  typeattribute ada t corenet unconfined type;
##### end corenet unconfined(ada t)
##### begin dev unconfined(ada t)
  require {
      attribute devices unconfined type;
```

```
} # end require
  typeattribute ada t devices unconfined type;
##### end dev unconfined(ada t)
##### begin domain unconfined(ada t)
  require {
     attribute set curr context;
      attribute can change object identity;
      attribute unconfined_domain_type;
      attribute process uncond exempt;
  } # end require
  typeattribute ada_t unconfined_domain_type;
# pass constraints
  typeattribute ada t can change object identity;
  typeattribute ada t set curr context;
  typeattribute ada_t process_uncond_exempt;
##### end domain_unconfined(ada_t)
##### begin domain dontaudit read all domains state(ada t)
  require {
     attribute domain;
  } # end require
  dontaudit ada t domain:dir { getattr search read lock ioctl };
  dontaudit ada t domain:lnk file { getattr read };
  dontaudit ada_t domain:file { getattr read lock ioctl };
# cjp: these should be removed:
  dontaudit ada t domain:sock_file { getattr read };dontaudit ada_t
domain:fifo_file { getattr read lock ioctl };
##### end domain_dontaudit_read_all_domains_state(ada_t)
##### begin domain dontaudit ptrace all domains(ada t)
  require {
      attribute domain;
  } # end require
  dontaudit ada t domain:process ptrace;
###### end domain_dontaudit_ptrace_all_domains(ada_t)
##### begin files_unconfined(ada_t)
  require {
      attribute files unconfined type;
  } # end require
  typeattribute ada t files unconfined type;
##### end files unconfined(ada t)
##### begin fs_unconfined(ada_t)
  require {
      attribute filesystem unconfined type;
  } # end require
  typeattribute ada_t filesystem_unconfined_type;
##### end fs unconfined(ada t)
##### begin selinux unconfined(ada t)
  require {
     attribute selinux_unconfined_type;
  } # end require
  typeattribute ada_t selinux_unconfined_type;
##### end selinux_unconfined(ada_t)
##### begin domain mmap low type(ada t)
  require {
      attribute mmap_low_domain_type;
  } # end require
  typeattribute ada t mmap low domain type;
##### end domain_mmap_low_type(ada_t)
  require {
      bool allow unconfined mmap low;
```

```
} # end require
  if (allow_unconfined_mmap_low) {
##### end domain_mmap_low(ada_t)
  }
  require {
     bool allow execheap;
  } # end require
  if (allow execheap) {
  # Allow making the stack executable via mprotect.
      allow ada_t self:process execheap;
   }
  require {
     bool allow execmem;
  } # end require
  if (allow execmem) {
      # Allow making anonymous memory executable, e.g.
      # for runtime-code generation or executable stack.
      allow ada t self:process execmem;
  }
  require {
     bool allow execstack;
  } # end require
  if (allow execstack) {
      # Allow making the stack executable via mprotect;
      # execstack implies execmem;
      allow ada t self:process { execstack execmem };
      auditallow ada_t self:process execstack;}
  optional { # start optional #7
##### begin auth_unconfined(ada_t)
      require {
          attribute can_read_shadow_passwords;
          attribute can write shadow passwords;
          attribute can relabelto shadow passwords;
      } # end require
typeattribute ada t can read shadow passwords; typeattribute ada t
can write shadow passwords; typeattribute ada t can relabelto shadow passwords;
##### end auth_unconfined(ada_t) depth: 1
} # end optional #7
  optional { # start optional #8
# Communicate via dbusd.
##### begin dbus_system_bus_unconfined(ada_t)
      require {
         type system dbusd t;
          class dbus { acquire svc send msg };
      } # end require
      allow ada_t system_dbusd_t:dbus *;
##### end dbus system bus unconfined(ada t)
##### begin dbus unconfined(ada t) depth: 2
      require {
         attribute dbusd unconfined;
      } # end require
      typeattribute ada_t dbusd_unconfined;
##### end dbus_unconfined(ada_t)
  } # end optional #8
  optional {# start optional #9
##### begin ipsec_setcontext_default_spd(ada_t)
      require {
         type ipsec spd t;
      } # end require
```

```
allow ada_t ipsec_spd_t:association setcontext;
##### end ipsec setcontext default spd(ada t)
###### begin ipsec_match_default_spd(ada_t)
    require { type ipsec_spd_t;
      } # end require
      allow ada t ipsec spd t:association polmatch;
      allow ada_t self:association sendto;
##### end ipsec match default spd(ada t)
  } # end optional #9
  optional { # start optional #10
  # this is to handle execmod on shared
  # libs with text relocations
##### begin libs_use_shared_libs(ada_t)
      require { type lib_t, textrel_shlib_t;
      } # end require
##### begin files_list_usr(ada_t)
      require {
         type usr t;
      } # end require
      allow ada t usr t:dir { getattr search read lock ioctl };
##### end files list usr(ada t)
  allow ada_t lib_t:dir { getattr search read lock ioctl };
  allow ada t lib t:dir { getattr search };allow ada t { lib t
textrel shlib t }: Ink file { getattr read };
  allow ada t lib t:dir { getattr search };allow ada t { lib t
textrel shlib t }:file { getattr read execute ioctl };
  allow ada_t textrel_shlib_t:file execmod;
##### end libs use shared libs(ada t)
  } # end optional #10
  optional { # start optional #11
##### begin nscd_unconfined(ada_t)
      require {
         type nscd t;
          class nscd { getpwd getgrp gethost getstat admin shmempwd shmemgrp
shmemhost getserv shmemserv };
     } # end require
      allow ada t nscd t:nscd *;
##### end nscd_unconfined(ada_t)
  } # end optional #11
  optional { # start optional #12
##### begin postgresql_unconfined(ada_t)
  require {
      attribute sepgsql_unconfined_type;
  } # end require
  typeattribute ada_t sepgsql_unconfined_type;
##### end postgresql unconfined(ada t)
  } # end optional #12
  optional { # start optional #13
##### begin seutil_create_bin_policy(ada_t)
      require {
#
          attribute can_write_binary_policy;
          type selinux_config_t, policy_config_t;
      } # end require
##### begin files_search_etc(ada_t)
      require {
          type etc t;
      } # end require
      allow ada_t etc_t:dir { getattr search };
##### end files search etc(ada t)
      allow ada t selinux config t:dir { getattr search };
```

```
allow ada t policy config t:dir { getattr search lock ioctl write add name
};
     allow ada t policy config t:file { getattr create open };
     allow ada_t policy_config_t:dir { getattr search };
     allow ada_t policy_config_t:file { getattr write append lock ioctl };
      # typeattribute ada_t can_write_binary_policy;
##### end seutil create bin policy (ada t)
##### begin seutil_relabelto_bin_policy(ada_t)
     require {
     attribute can_relabelto_binary_policy; type policy_config_t;
     } # end require
     allow ada_t policy_config_t:file relabelto;typeattribute ada_t
can relabelto binary policy;
##### end seutil relabelto bin_policy(ada_t)
  # end optional #13
  optional { # start optional #14
##### begin storage_unconfined(ada_t)
     require {
         attribute storage_unconfined_type;
     } # end require
     typeattribute ada t storage unconfined type;
##### end storage_unconfined(ada_t)
  } # end optional #14
  optional { # start optional #15
##### begin xserver_unconfined(ada_t)
     require {
         attribute xserver_unconfined_type, x_domain;
      } # end require
     typeattribute ada t xserver unconfined type, x domain;
##### end xserver_unconfined(ada_t)
 } # end optional #15
##### end unconfined_domain_noaudit(ada_t)
} # end optional #6
#
```

5.7.2 File Context Expansion

As can be seen the gen_context macro has been expanded to build the security context:

```
#
#
/usr
#
/usr/bin/gnatbind -- system_u:object_r:ada_exec_t:s0
/usr/bin/gnatls -- system_u:object_r:ada_exec_t:s0
/usr/bin/gnatmake -- system_u:object_r:ada_exec_t:s0
/usr/libexec/gcc(/.*)?/gnat1 -- system_u:object_r:ada_exec_t:s0
```

6. Appendix A - Object Classes and Permissions

6.1 Introduction

This section contains a list of object classes and their associated permissions that have been taken from the Fedora F-12 policy sources.

All objects are kernel objects unless marked as user space objects.

Depreciated permissions and objects still in the policy sources are shown in the tables as italic text.

The depreciated permissions are:

```
transition, swapon, {\tt enforce\_dest}, {\tt connectto}, {\tt acceptfrom} and {\tt newconn}
```

The depreciated object is:

ipc

In most cases the permissions are self explanatory as they are those used in the standard Linux function calls (such as 'create a socket' or 'write to a file'). The SELinux specific permissions are:

relabelfrom	Used on most objects to allow the objects security context to be changed from the current type.
relabelto	Used on most objects to allow the objects security context to be changed to the new type.
entrypoint	Used for files to indicate that they can be used as an entry point into a domain via a domain transition.
execute_no_trans	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).
execmod	Generally used for files to indicate that they can execute the modified file in memory.

Where possible the specific object class permissions are explained, however for some permissions it is difficult to determine what they are used for (or if used at all) so a '?' has been added when doubt exists. There are lists of object classes and permissions at the following locations and would probably be more up-to-date:

http://oss.tresys.com/projects/refpolicy/wiki/ObjectClassesPerms http://selinuxproject.org/page/ObjectClassesPerms

6.2 Defining Object Classes and Permissions

The Fedora 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 <u>Object Classes</u> and <u>Permissions</u> sections explain how these are defined within the <u>SELinux Policy Language</u>.

The sections that follow contain the entries defined in the F-12 ./policy/flask/access_vectors file of the <u>Reference Policy</u> source.

6.3 Common Permissions

6.3.1 Common File Permissions

<u>Table 6-1</u> 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 6-1: Common File Permissions

6.3.2 Common Socket Permissions

<u>Table 6-2</u> 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.

1 . 1	
ioctl	Get and set attributes via ioctl 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 6-2: Common Socket Permissions

6.3.3 Common IPC Permissions

<u>Table 6-3</u> 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 6-3: Common IPC Permissions

6.3.4 Common Database Permissions

<u>Table 6-4</u> describes the common database permissions that are inherited by a number of object classes. The "<u>Security-Enhanced PostgreSQL Security Wiki</u>" [Ref. 3] explains the objects, their permissions and how they should be used in detail.

Permissions	Description (6 Permissions)
create	Create a database object such as a 'TABLE'.
drop	Delete (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 6-4: Common PostgreSQL Database Permissions

6.3.5 Common X_Device Permissions

<u>Table 6-5</u> 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 6-5: Common X_Device Permissions

Class	filesystem – A mounted filesystem
Permissions	Description (10 unique permissions)
associate	Use type as label for file.
getattr	Get file attributes.
mount	Mount filesystem.
quotaget	Get quota information.
quotamod	Modify quota information.
relabelfrom	Change the security context based on existing type.
relabelto	Change the security context based on the new type.
remount	Remount existing mount.
transition	Transition to a new SID (change security context).
unmount	Unmount filesystem.

6.4 File Object Classes

Class	dir - Directory
Permissions	Description (Inherit 17 common file permissions + 6 unique)
<u>Inherit Common</u> <u>File Permissions</u>	append, create, execute, getattr, ioctl, link, lock, mounton, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
add_name	Add entry to the directory.
open	Added in 2.6.26 Kernel to control the open permission.
remove_name	Remove an entry from the directory.
reparent	Change parent directory.
rmdir	Remove directory.
search	Search directory.

Class	file – Ordinary file
Permissions	Description (Inherit 17 common file permissions + 4 unique)
<u>Inherit Common</u> <u>File Permissions</u>	append, create, execute, getattr, ioctl, link, lock, mounton, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
entrypoint	Entry point permission for a domain transition.
execmod	Make executable a file that has been modified by copy-on-write.
execute_no_trans	Execute in the caller's domain (i.e. no domain transition).
open	Added in 2.6.26 Kernel to control the open permission.

Class	lnk_file - Symbolic links
Permissions	Description (Inherit 17 common file permissions)
<u>Inherit Common</u> <u>File Permissions</u>	append, create, execute, getattr, ioctl, link, lock, mounton, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write

Class	chr_file – Character files
Permissions	Description (Inherit 17 common file permissions + 4 unique)

Inherit Common File Permissions	append, create, execute, getattr, ioctl, link, lock, mounton, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
entrypoint	Entry point permission for a domain transition.
execmod	Make executable a file that has been modified by copy-on-write.
execute_no_trans	Execute in the caller's domain (i.e. no domain transition).
open	Added in 2.6.26 Kernel to open a character device.

Class	blk_file - Block files
Permissions	Description (Inherit 17 common file permissions + 1 unique)
<u>Inherit Common</u> <u>File Permissions</u>	append, create, execute, getattr, ioctl, link, lock, mounton, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, 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)
<u>Inherit Common</u> <u>File Permissions</u>	append, create, execute, getattr, ioctl, link, lock, mounton, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write

Class	fifo_file - Named pipes
Permissions	Description (Inherit 17 common file permissions + 1 unique)
<u>Inherit Common</u> <u>File Permissions</u>	append, create, execute, getattr, ioctl, link, lock, mounton, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
open	Added in 2.6.26 Kernel to control the open permission.

Class	fd – File descriptors
Permissions	Description (1 unique permission)
use	1) Inherit fd when process is executed and domain has been changed.
	2) Receive fd from another process by Unix domain socket.
	3) Get and set attribute of fd.

6.5 Network Object Classes

Class	node – IP address or range of IP addresses
Permissions	Description (11 unique permissions)
dccp_recv	Allow Datagram Congestion Control Protocol receive packets.
dccp_send	Allow Datagram Congestion Control Protocol send packets.
enforce_dest	Ensure that destination node can enforce restrictions on the destination socket.
rawip_recv	Receive raw IP packet.
rawip_send	Send raw IP packet.
recvfrom	Network interface and address check permission for use with the ingress permission.

sendto	Network interface and address check permission for use with the egress permission.
tcp_recv	Receive TCP packet.
tcp_send	Send TCP packet.
udp_recv	Receive UDP packet.
udp_send	Send UDP packet.

Class	<pre>netif - Network Interface (e.g. eth0)</pre>
Permissions	Description (10 unique permissions)
dccp_recv	Allow Datagram Congestion Control Protocol receive packets.
dccp_send	Allow Datagram Congestion Control Protocol send packets.
egress	Each packet leaving the system must pass an egress access control. Also requires the node sendto permission.
ingress	Each packet entering the system must pass an ingress access control. Also requires the node recvfrom permission.
rawip_recv	Receive raw IP packet.
rawip_send	Send raw IP packet.
tcp_recv	Receive TCP packet.
tcp_send	Send TCP packet.
udp_recv	Receive UDP packet.
udp_send	Send UDP packet.

Class	socket – Socket that is not part of any other specific SELinux socket object class.
Permissions	Description (Inherit 22 common socket permissions)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>

Class	tcp_socket - Protocol: PF_INET, PF_INET6 Family Type: SOCK_STREAM
Permissions	Description (Inherit 22 common socket permissions + 5 unique)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>
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)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>
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)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>
node_bind	Bind to a node.

Class	packet_socket - Protocol: PF_PACKET Family Type: All.
Permissions	Description (Inherit 22 common socket permissions)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>

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)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	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)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>

Class	tun_socket – TUN is Virtual Point-to-Point network device driver to support IP tunneling.
Permissions	Description (Inherit 22 common socket permissions)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>

6.5.1 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 assocation.
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)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>

Class	netlink_xfrm_socket - Netlink socket to maintain IPSec parameters.
Permissions	Description (Inherit 22 common socket permissions + 2 unique)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>
nlmsg_read	Get IPSec configuration information.
nlmsg_write	Set IPSec configuration information.

6.5.2 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	accept, append, bind, connect, create, getattr,

<u>Socket</u>	getopt, ioctl, listen, lock, name_bind, read,
<u>Permissions</u>	<pre>recv_msg, recvfrom, relabelfrom, relabelto,</pre>
	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)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>
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)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>
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)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>
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)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>

Class	netlink_selinux_socket - Netlink socket to receive
	SELinux events such as a policy or boolean change.
Permissions	Description (Inherit 22 common socket permissions)

<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	netlink_audit_socket - Netlink socket for audit service.
Permissions	Description (Inherit 22 common socket permissions + 5 unique)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>
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)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>
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)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>

Class	netlink_kobject_uevent_socket - Netlink socket to send kernel events to userspace.
Permissions	Description (Inherit 22 common socket permissions)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>

6.5.3 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 <u>http://paulmoore.livejournal.com/1863.html</u> 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 thent 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 forwaded 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)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>

Class	dccp_socket - Datagram Congestion Control Protocol (DCCP)
Permissions	Description (Inherit 22 common socket permissions + 2 unique)
<u>Inherit Common</u> <u>Socket</u> <u>Permissions</u>	<pre>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</pre>
name_connect	Allow DCCP name connect().
node_bind	Allow DCCP bind().

6.6 IPC Object Classes

Class	<i>ipc</i> – No longer used
Permissions	Description (Inherit 9 common IPC permissions)
<u>Inherit Common</u> <u>IPC Permissions</u>	associate, create, destroy, getattr, read, setattr, unix_read, unix_write, write

Class	sem - Semaphores
Permissions	Description (Inherit 9 common IPC permissions)

	associate, create, destroy, getattr, read,
IPC Permissions	setattr, unix read, unix write, write

Class	msgq – IPC Message queues
Permissions	Description (Inherit 9 common IPC permissions + 1 unique)
<u>Inherit Common</u> <u>IPC Permissions</u>	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.

6.7 Process Object Class

Class	process – An object is instantiated for each process created by the system.
Permissions	Description (30 unique permissions)
dyntransition	Dynamically transition to a new context (domain).
execheap	Make the heap executable.
execmem	Make executable an anonymous mapping or private file mapping that is writable.
execstack	Make the main process stack executable.
fork	Create new process using fork (2).
getattr	Get process security information.
getcap	Get Linux capabilities of process.
getpgid	Get group Process ID of another process.
getsched	Get scheduling information of another process.
getsession	Get session ID of another process.
noatsecure	Disable secure mode environment cleansing.
ptrace	Trace program execution of parent or child (ptrace (2)).
rlimitinh	Inherit rlimit information from parent process.
setcap	Set Linux capabilities of process.
setcurrent	Set the current process context.
setexec	Set security context of executed process by setexecon() call.
setfscreate	Set security context by setfscreatecon() call.
setkeycreate	Set security context by setkeycreatecon() call.

setpgid	Set group Process ID of another process.
setrlimit	Change process rlimit information.
setsched	Modify scheduling information of another process.
setsockcreate	Set security context by setsockcreatecon() call.
share	Allow state sharing with cloned or forked process.
sigchld	Send SIGCHLD signal.
siginh	Inherit signal state from parent process.
sigkill	Send SIGKILL signal.
signal	Send a signal other than SIGKILL, SIGSTOP, or SIGCHLD.
signull	Test for exisitence of another process without sending a signal
sigstop	Send SIGSTOP signal
transition	Transition to a new context on exec().

6.8 Security Object Class

Class	security - This is the security server object and there is only one instance of this object (for the SELinux security server).
Permissions	Description (11 unique permissions)
check_context	Determine whether the context is valid by querying the security server.
compute_av	Compute an access vector given a source/target/class.
compute_create	Determine context to use when querying the security server about a transition rule (type_transition).
compute_member	Determine context to use when querying the security server about a membership decision (type_member for a polyinstantiated object).
compute_relabel	Determines the context to use when querying the security server about a relabeling decision (type_change).
compute_user	Determines the context to use when querying the security server about a user decision (user).
load_policy	Load the security policy into the kernel (the security server).
setbool	Change a boolean value within the active policy.
setcheckreqprot	Set if SELinux will check original protection mode or modified protection mode (read-implies-exec) for mmap/mprotect.
setenforce	Change the enforcement state of SELinux (permissive or enforcing).
setsecparam	Set kernel access vector cache tuning parameters.

6.9 System Operation Object Class

Class	system - This is the overall system object and there is only one instance of this object.
Permissions	Description (4 unique permissions)
ipc_info	Get info about an IPC object.
syslog_console	Control output of kernel messages to the console with syslog(2).
syslog_mod	Clear kernel message buffer with syslog(2).
syslog_read	Read kernel message with syslog(2).

6.10 Kernel Service Object Class

Class	kernel_service - Used to add kernel services.
Permissions	Description (2 unique permissions)
use_as_override	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 cachefilesd daemon will nominate the security ID to be used.
create_files_as	Grant a process the right to nominate a file creation label for a kernel service to use.

6.11 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 messsages from user space.
chown	Allow changing file and group ownership.
dac_override	Overrides all DAC including ACL execute access.
dac_read_search	OverridesDAC 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; nfsservctl; 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 sysctl.
sys_nice	Grants privilage 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 /proc/bus/usb.
sys_resource	Override the following: resource limits; quota limits; reserved space on ext2 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 ext3 filesystem, Allow more than 64hz interrupts from the real-time clock.
sys_time	Grant permission to set system time and to set the real-time lock.
sys_tty_config	Grant permission to configure tty devices.

Class	capability2 – This is currently unused by SELinux
Permissions	Description (2 unique permissions)
mac_admin	Unused
mac_override	Unused

6.12 X Windows Object Classes

These are userspace objects.

Class	x drawable - The drawable 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.
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 override-redirect 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 nthe window, therefore only SystemHigh processes should have the setattr permission.
show	Show window. Not applicable to the root windows as it cannot be hidden.

Draw within a window. Note that this will also give write permission to all child windows, therefore (for MLS), only SystemHigh processes
should have write permission on the root window.

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
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.
Permissions	Description (Inherit 19 common x_device permissions)
<u>Inherit Common</u> X_Device Permissions	<pre>add, bell, create, destroy, force_cursor, freeze, get_property, getattr, getfocus, grab, list_property, manage, read, remove, set property, setattr, setfocus, use, write</pre>

Class	x_server - The X-server that manages the display, keyboard and pointer.
Permissions	Description (6 unique permissions)
debug	
getattr	
grab	
manage	Required to create a context. (source code)
record	
setattr	

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)
query	Query for an extension.
use	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)
read	Allow reading a resource.
write	Allow writing to a resource.

Class	x_event - Manage X-server events.
Permissions	Description (2 unique permissions)
receive	Receive an event
send	Send an event

Class	x_synthetic_event - Manage some X-server events (e.g. confignotify). Note the x_event permissions will still be required (its magic).
Permissions	Description (2 unique permissions)
receive	Receive an event
send	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 CUT_BUFFERS).
Permission	Description (3 unique permissions)
сору	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)
<u>Inherit Common</u> X_Device Permissions	<pre>add, bell, create, destroy, force_cursor, freeze, get_property, getattr, getfocus, grab, list_property, manage, read, remove, set_property, setattr, setfocus, use, write</pre>

Class	x_keyboard - The keyboard managed by the X-server.
Permissions	Description (Inherit 19 common x_device permissions)
Inherit Common X_Device Permissions	<pre>add, bell, create, destroy, force_cursor, freeze, get_property, getattr, getfocus, grab, list_property, manage, read, remove, set_property, setattr, setfocus, use, write</pre>

6.13 Database Object Classes

These are userspace objects – The PostgreSQL database supports these with their SE-PostgreSQL database extension. The "<u>Security-Enhanced PostgreSQL Security Wiki</u>" [Ref. 3] explains the objects, their permissions and how they should be used in detail.

Class	db_database
Permission	Description (Inherit 6 common database permissions + 4 unique)
<u>Inherit Common</u> <u>Database</u> <u>Permissions</u>	create, drop, getattr, relabelfrom, relabelto, setattr
access	Required to connect to the database – this is the minimum permission required by an PostgreSQL client.
install_module	Required to install a dynmic link library.
load_module	Required to load a dynmic link library.
superuser	Required if user is allowed DBA access.

Class	db_table
Permission	Description (Inherit 6 common database permissions + 6 unique)
<u>Inherit Common</u> <u>Database</u> <u>Permissions</u>	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.
reference	

Class	db_sequence
Permission	Description (Inherit 6 common database permissions + 3 unique)
<u>Inherit Common</u> <u>Database</u> <u>Permissions</u>	create, drop, getattr, relabelfrom, relabelto, setattr
get_value	
next_value	
set_value	

Class	db_schema
Permission	Description (Inherit 6 common database permissions + 2 unique)
<u>Inherit Common</u> <u>Database</u> <u>Permissions</u>	create, drop, getattr, relabelfrom, relabelto, setattr
add_name	Add (CREATE) a schema name
remove_name	Remove (DROP) a schema name.

Class	db_procedure
Permission	Description (Inherit 6 common database permissions + 4 unique)
<u>Inherit Common</u> <u>Database</u> <u>Permissions</u>	create, drop, getattr, relabelfrom, relabelto, setattr
entrypoint	Required for any functions defined as Trusted Procedures (see [Ref. 3]).
execute	Required for functions executed with SQL queries.
install	
untrusted	

Class	db_column
Permission	Description (Inherit 6 common database permissions + 4 unique)
<u>Inherit Common</u> <u>Database</u> <u>Permissions</u>	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.
reference	

Class	db_tuple
Permission	Description (6 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
relabelto	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.
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.

Class	db_blob
Permission	Description (Inherit 6 common database permissions + 4 unique)
<u>Inherit Common</u> <u>Database</u> <u>Permissions</u>	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 objecty with the lowrite() function.

6.14 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.
crontab	crontab another user.
passwd	Change another users passwd.
rootok	pam_rootok check - skip authentication.

Class	nscd - This is a userspace object for the Name Service Cache Daemon.
Permission	Description (10 unique permissions)
admin	Allow the nscd daemon to be shut down.

The SELinux Notebook - The Foundations

getgrp	Get group information.
gethost	Get host information.
getpwd	Get password information.
getserv	Get ?? information.
getstat	Get the AVC stats from the nscd daemon.
shmemgrp	Get shmem group file descriptor.
shmemhost	Get shmem host descriptor. ??
shmempwd	
shmemserv	

Class	dbus - This is a userspace object for the D-BUS Messaging service that is required to run X-Windows (and other services).
Permission	Description (2 unique permissions)
acquire_svc	Open a virtual circuit (communications channel).
send_msg	Send a message.

Class	context - This is a userspace object for the translation daemon mcstransd. These permissions are required to allow translation and querying of level and ranges for MCS and MLS systems.
Permission	Description (2 unique permissions)
contains	Calculate a MLS/MCS subset - Required to check what the configuration file contains.
translate	Translate a raw MLS/MCS label - Required to allow a domain to translate contexts.

Class	key – This is a kernel object to manage Keyrings.
Permission	Description (7 unique permissions)
create	Create a keyring.
link	Link a key into the keyring.
read	Read a keyring.
search	Search a keyring.
setattr	Change permissions on a keyring.
view	View a keyring.
write	Add a key to the keyring.

Class	memprotect – This is a kernel object to protect lower memory blocks.	
Permission	Description (1 unique permission)	
mmap_zero	Security check on mmap operations to see if the user is attempting to mmap to low area of the address space. The amount of space protected is indicated by a proc tunable (/proc/sys/vm/mmap_min_addr). Setting this value to 0 will disable the checks. The " <u>SELinux hardening for</u> <u>mmap_min_addr protections</u> " [Ref. 16] 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 /
<pre>mmap_zero permission instead of only one or the other).</pre>

7. Appendix B – SELinux Commands

This section gives a brief explanation of the SELinux specific commands. Some of these have been used within this Notebook, however the appropriate man pages do give more detail and the SELinux project site has a page that details all the available tools and commands at:

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 catergory 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, 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 / semodule 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.		
selinux-relabel	-	This is a script that is called using 'service selinux- relabel cmd', where cmd is help, status, relabel, switch <policy_name> or cancel.</policy_name>		

http://userspace.selinuxproject.org/trac/wiki/SelinuxTools

8. Appendix C – API Summary for libselinux

These functions have been taken from the following header files delivered in the "libselinux-devel-2.0.90-5.fc12.i686" rpm, and sorted in alphabetical order:

```
/usr/include/selinux/avc.h
/usr/include/selinux/context.h
/usr/include/selinux/get_context_list.h
/usr/include/selinux/get_default_type.h
/usr/include/selinux/label.h
/usr/include/selinux/selinux.h
```

The appropriate man (3) pages should consulted for detailed usage, also the libselinux source code (see the "libselinux-2.0.90-5.fc12.src" rpm) has a number of sample applications showing the API usage, and the "SELinux Support for Userspace Object Managers" [Ref. 17] illustrates the use of a userspace AVC.

Num.	Function Name	Description	Header File
1.	avc_add_callback	Register a callback for security events. Register a callback function for events in the set @events related to the SID pair (@ssid, @tsid) and and the permissions @perms, interpreting @perms based on @tclass. Returns %0 on success or -%1 if insufficient memory exists to add the callback.	avc.h
2.	avc_audit	Audit the granting or denial of permissions in accordance with the policy. This function is typically called by avc_has_perm() after a permission check, but can also be called directly by callers who use avc_has_perm_noaudit() in order to separate the permission check from the auditing. For example, this separation is useful when the permission check must be performed under a lock, to allow the lock to be released before calling the auditing code.	avc.h
3.	avc_av_stats	Log AV table statistics. Log a message with information about the size and distribution of the access vector table. The audit callback is used to print the message.	avc.h

Num.	Function Name	Description	Header File
4.	avc_cache_stats	Get cache access statistics. Fill the supplied structure with information about AVC activity since the last call to avc_init() or avc_reset(). See the structure definition for details.	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 context for labeling a new object. Look up the context in the SID table, making a new entry if not found. Increment the reference counter for the SID. Store a pointer to the SID structure into the memory referenced by @newsid, returning %0 on success or -%1 on error with @errno set.	avc.h
7.	avc_compute_member	Compute SID for polyinstantation. Call the security server to obtain a context for labeling an object instance. Look up the context in the SID table, making a new entry if not found. Increment the reference counter for the SID. Store a pointer to the SID structure into the memory referenced by @newsid, returning %0 on success or -%1 on error with @errno set.	avc.h
8.	<pre>avc_context_to_sid avc_context_to_sid_raw</pre>	Get SID for context. Look up security context @ctx in SID table, making a new entry if @ctx is not found. Increment the reference counter for the SID. Store a pointer to the SID structure into the memory referenced by @sid,returning %0 on success or -%1 on error with @errno set.	avc.h
9.	avc_destroy	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 avc_init() if further use of AVC is desired.	avc.h
10.	avc_entry_ref_init	Initialize an AVC entry reference. Use this macro to initialize an avc entry reference structure before first use. These structures are passed to avc_has_perm(), which stores cache entry references in them. They can increase performance on repeated queries.	avc.h

Num.	Function Name	Description	Header File
11.	avc_get_initial_sid	Get SID for an initial kernel security identifier. Get the context for an initial kernel security identifier specified by @name using security_get_initial_context() and then call avc_context_to_sid() to get the corresponding SID.	avc.h
12.	avc_has_perm	Check permissions and perform any appropriate auditing. Check the AVC to determine whether the @requested permissions are granted for the SID pair (@ssid, @tsid), interpreting the permissions based on @tclass, and call the security server on a cache miss to obtain a new decision and add it to the cache. Update @aeref to refer to an AVC entry with the resulting decisions. Audit the granting or denial of permissions in accordance with the policy. Return %0 if all @requested permissions are granted, -%1 with @errno set to %EACCES if any permissions are denied or to another value upon other errors.	avc.h
13.	avc_has_perm_noaudit	Check permissions but perform no auditing. Check the AVC to determine whether the @requested permissions are granted for the SID pair (@ssid, @tsid), interpreting the permissions based on @tclass, and call the security server on a cache miss to obtain a new decision and add it to the cache. Update @aeref to refer to an AVC entry with the resulting decisions, and return a copy of the decisions in @avd. Return %0 if all @requested permissions are granted, -%1 with @errno set to %EACCES if any permissions are denied, or to another value upon other errors. This function is typically called by avc_has_perm(), 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.	avc.h
14.	avc_init	Initialize the AVC. Initialize the access vector cache. Return %0 on success or -%1 with @errno set on failure. If @msgprefix is NULL, use "uavc". If any callback structure references are NULL, use default methods for those callbacks (see the definition of the callback structures).	avc.h

Num.	Function Name	Description	Header File
15.	avc_netlink_acquire_fd	Create a netlink socket and connect to the kernel.	avc.h
16.	avc_netlink_check_nb	Wait for netlink messages from the kernel.	avc.h
17.	avc_netlink_close	Close the netlink socket.	avc.h
18.	avc_netlink_loop	Acquire netlink socket fd. Allows the application to manage messages from the netlink socket in its own main loop.	avc.h
19.	avc_netlink_open	Release netlink socket fd. Returns ownership of the netlink socket to the library.	avc.h
20.	avc_netlink_release_fd	Check netlink socket for new messages. Called by the application when using avc_netlink_acquire_fd() to process kernel netlink events.	avc.h
21.	avc_open	Initialize the AVC. This function is identical to avc_init() except the message prefix is set to "avc" and any callbacks desired should be specified via selinux_set_callback().	avc.h
22.	avc_reset	Flush the cache and reset statistics. Remove all entries from the cache and reset all access statistics (as returned by avc_cache_stats()) to zero. The SID mapping is not affected. Return %0 on success, -%1 with @errno set on error.	avc.h
23.	avc_sid_stats	Log SID table statistics. Log a message with information about the size and distribution of the SID table. The audit callback is used to print the message.	avc.h
24.	<pre>avc_sid_to_context avc_sid_to_context_raw</pre>	Get copy of context corresponding to SID. Return a copy of the security context corresponding to the input @sid in the memory referenced by @ctx. The caller is expected to free the context with freecon(). Return %0 on success, -%1 on failure, with @errno set to %ENOMEM if insufficient memory was available to make the copy, or %EINVAL if the input SID is invalid.	avc.h
25.	checkPasswdAccess	Check a permission in the passwd 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 range.	context.h
29.	context_range_set	Set the range component. Returns nonzero if unsuccessful.	context.h
30.	context_role_get	Get a pointer to the role.	context.h
31.	context_role_set	Set the role component. Returns nonzero if unsuccessful.	context.h

Num.	Function Name	Description	Header File
32.	context_str	Return a pointer to the string value of context_t. Valid until the next call to context_str or context_free for the same context_t*.	context.h
33.	context_type_get	Get a pointer to the type.	context.h
34.	context_type_set	Set the type component. Returns nonzero if unsuccessful.	context.h
35.	context_user_get	Get a pointer to the user.	context.h
36.	context_user_set	Set the user component. Returns nonzero if unsuccessful.	context.h
37.	fgetfilecon fgetfilecon_raw	Wrapper for the xattr API - Get file context, and set *con to refer to it. Caller must free via freecon.	selinux.h
38.	freecon	Free the memory allocated for a context by any of the get * calls.	selinux.h
39.	freeconary	Free the memory allocated for a context array by security_compute_user.	selinux.h
40.	fsetfilecon fsetfilecon_raw	Wrapper for the xattr API- Set file context.	selinux.h
41.	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
42.	get_default_context_with_level	Same as get_default_context, but use the provided MLS level rather than the default level for the user.	get_context_list.h
43.	get_default_context_with_role	Same as get_default_context, but only return a context that has the specified role.	get_context_list.h
44.	get_default_context_with_rolelevel	Same as get_default_context, but only return a context that has the specified role and level.	get_context_list.h
45.	get_default_type	Get the default type (domain) for 'role' and set 'type' to refer to it. Caller must free via free (). Return 0 on success or -1 otherwise.	get_default_type.h
46.	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	get_context_list.h

Num.	Function Name	Description	Header File
		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.	
47.	get_ordered_context_list_with_level	Same as get_ordered_context_list, but use the provided MLS level rather than the default level for the user.	get_context_list.h
48.	getcon getcon_raw	Get current context, and set *con to refer to it. Caller must free via freecon.	selinux.h
49.	getexeccon getexeccon_raw	Get exec context, and set *con to refer to it. Sets *con to NULL if no exec context has been set, i.e. using default. If non-NULL, caller must free via freecon.	selinux.h
50.	getfilecon getfilecon_raw	Wrapper for the xattr API - Get file context, and set *con to refer to it. Caller must free via freecon.	selinux.h
51.	getfscreatecon getfscreatecon_raw	Get fscreate context, and set *con to refer to it. Sets *con to NULL if no fs create context has been set, i.e. using default. If non-NULL, caller must free via freecon.	selinux.h
52.	getkeycreatecon getkeycreatecon_raw	Get keycreate context, and set *con to refer to it. Sets *con to NULL if no key create context has been set, i.e. using default. If non-NULL, caller must free via freecon.	selinux.h
53.	getpeercon getpeercon_raw	Wrapper for the socket API - Get context of peer socket, and set *con to refer to it. Caller must free via freecon.	selinux.h
54.	getpidcon getpidcon_raw	Get context of process identified by pid, and set *con to refer to it. Caller must free via freecon.	selinux.h
55.	getprevcon getprevcon_raw	Get previous context (prior to last exec), and set *con to refer to it. Caller must free via freecon.	selinux.h
56.	getseuser	Get the SELinux username and level to use for a given Linux username and service. These values may then be passed into the get_ordered_context_list* and get_default_context* functions to obtain a context for the user. Returns 0 on success or -1 otherwise. Caller must free the returned strings via free().	selinux.h
57.	getseuserbyname	Get the SELinux username and level to use for a given Linux	selinux.h

Num.	Function Name	Description	Header File
		<pre>username. These values may then be passed into the get_ordered_context_list* and get_default_context* functions to obtain a context for the user. Returns 0 on success or -1 otherwise. Caller must free the returned strings via free().</pre>	
58.	getsockcreatecon getsockcreatecon_raw	Get sockcreate context, and set *con to refer to it. Sets *con to NULL if no socket create context has been set, i.e. using default. If non-NULL, caller must free via freecon.	selinux.h
59.	is_context_customizable	Returns whether a file context is customizable, and should not be relabeled.	selinux.h
60.	is_selinux_enabled	Return 1 if running on a SELinux kernel, or 0 if not or -1 for error.	selinux.h
61.	is_selinux_mls_enabled	Return 1 if we are running on a SELinux MLS kernel, or 0 otherwise.	selinux.h
62.	lgetfilecon lgetfilecon_raw	Wrapper for the xattr API - Get file context, and set *con to refer to it. Caller must free via freecon.	selinux.h
63.	lsetfilecon lsetfilecon_raw	Wrapper for the xattr API- Set file context for symbolic link.	selinux.h
64.	manual_user_enter_context	Allow the user to manually enter a context as a fallback if a list of authorized contexts could not be obtained. Caller must free via freecon. Returns 0 on success or -1 otherwise.	get_context_list.h
65.	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
66.	matchpathcon	Match the specified pathname and mode against the file context sconfiguration and set *con to refer to the resulting context.'mode' can be 0 to disable mode matching. Caller must free via freecon. If matchpathcon_init has not already been called, then this function will call it upon its first invocation with a NULL path.	selinux.h
67.	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
68.	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

Num.	Function Name	Description	Header File
69.	matchpathcon_filespec_destroy	Destroy any inode associations that have been added, e.g. to restart for a new filesystem.	selinux.h
70.	matchpathcon_filespec_eval	Display statistics on the hash table usage for the associations.	selinux.h
71.	matchpathcon_fini	Free the memory allocated by matchpathcon_init.	selinux.h
72.	matchpathcon_index	Same as 'matchpathcon', but return a specification index for later use in a matchpathcon_filespec_add() call.	selinux.h
73.	matchpathcon_init	Load the file contexts configuration specified by 'path' into memory for use by subsequent matchpathcon calls. If 'path' is NULL, then load the active file contexts configuration, i.e. the path returned by selinux_file_context_path(). Unless the MATCHPATHCON_BASEONLY flag has been set, this function also checks for a 'path'.homedirs file and a 'path'.local file and loads additional specifications from them if present.	selinux.h
74.	<pre>matchpathcon_init_prefix</pre>	Same as matchpathcon_init, but only load entries with regexes that have stems that are prefixes of 'prefix'.	selinux.h
75.	print_access_vector	Display an access vector in a string representation.	selinux.h
76.	query_user_context	Given a list of authorized security contexts for the user, query the user to select one and set *newcon to refer to it. Caller must free via freecon. Returns 0 on success or -1 otherwise.	get_context_list.h
77.	rpm_execcon	Execute a helper for rpm in an appropriate security context.	selinux.h
78.	security_av_perm_to_string	Convert access vector permissions to string names.	selinux.h
79.	security_av_string	Returns an access vector in a string representation. User must free the returned string via free().	selinux.h
80.	<pre>security_canonicalize_context security_canonicalize_context_raw</pre>	Canonicalize a security context.	selinux.h
81.	<pre>security_check_context security_check_context_raw</pre>	Check the validity of a security context.	selinux.h
82.	security_class_to_string	Convert security class values to string names.	selinux.h
83.	security_commit_booleans	Commit the pending values for the booleans.	selinux.h
84.	security_compute_av	Compute an access decision.	selinux.h

Num.	Function Name	Description	Header File
	security_compute_av_raw		
85.	<pre>security_compute_create security_compute_create_raw</pre>	Compute a labeling decision and set *newcon to refer to it. Caller must free via freecon.	selinux.h
86.	<pre>security_compute_member security_compute_member_raw</pre>	Compute a polyinstantiation member decision and set *newcon to refer to it. Caller must free via freecon.	selinux.h
87.	<pre>security_compute_relabel security_compute_relabel_raw</pre>	Compute a relabeling decision and set *newcon to refer to it. Caller must free via freecon.	selinux.h
88.	<pre>security_compute_user security_compute_user_raw</pre>	Compute the set of reachable user contexts and set *con to refer to the NULL-terminated array of contexts. Caller must free via freeconary.	selinux.h
89.	security_disable	Disable SELinux at runtime (must be done prior to initial policy load).	selinux.h
90.	security_get_boolean_active	Get the active value for the boolean.	selinux.h
91.	<pre>security_get_boolean_names</pre>	Get the boolean names	selinux.h
92.	<pre>security_get_boolean_pending</pre>	Get the pending value for the boolean.	selinux.h
93.	<pre>security_get_initial_context security_get_initial_context_raw</pre>	Get the context of an initial kernel security identifier by name. Caller must free via freecon.	selinux.h
94.	security_getenforce	Get the enforce flag value.	selinux.h
95.	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
96.	security_load_policy	Load a policy configuration.	selinux.h
97.	security_policyvers	Get the policy version number.	selinux.h
98.	security_set_boolean	Set the pending value for the boolean.	selinux.h
99.	<pre>security_set_boolean_list</pre>	Save a list of booleans in a single transaction.	selinux.h
100.	security_setenforce	Set the enforce flag value.	selinux.h
101.	selabel_close	Destroy the specified handle, closing files, freeing allocated memory, etc. The handle may not be further used after it has been closed.	label.h
102.	selabel_lookup selabel_lookup_raw	Perform a labeling lookup operation. Return %0 on success, -%1 with @errno set on failure. The key and type arguments are the inputs to the lookup operation; appropriate values are dictated by the backend in use. The result is returned in the memory pointed to by @con and must be freed by the	label.h

Num.	Function Name	Description	Header File
		user with freecon().	
103.	selabel_open	 Create a labeling handle. Open a labeling backend for use. The available backend identifiers are: SELABEL_CTX_FILE - file_contexts. SELABEL_CTX_MEDIA - media contexts. SELABEL_CTX_X - x_contexts. Options may be provided via the opts parameter; available options are: SELABEL_OPT_UNUSED - no-op option, useful for unused slots in an array of options. SELABEL_OPT_VALIDATE - validate contexts before returning them (boolean value). SELABEL_OPT_BASEONLY - don't use local customizations to backend data (boolean value). SELABEL_OPT_PATH - specify an alternate path to use when loading backend data. SELABEL_OPT_SUBSET - select a subset of the search space as an optimization (file backend). Not all options may be supported by every backend. Return value is the created handle on success or NULL with @errno set on failure. 	label.h
104.	selabel_stats	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.	label.h
105.	selinux_binary_policy_path	Return path to the binary policy file under the policy root directory.	selinux.h
106.	selinux_booleans_path	Return path to the booleans file under the policy root directory.	selinux.h
107.	selinux_check_passwd_access	Check a permission in the passwd class. Return 0 if granted or -1 otherwise.	selinux.h
108.	selinux_check_securetty_context	Check if the tty_context is defined as a securetty. Return 0 if secure, <0 otherwise.	selinux.h
109.	selinux_colors_path	Return path to file under the policy root directory.	selinux.h
110.	selinux_contexts_path	Return path to contexts directory under the policy root directory.	selinux.h
111.	<pre>selinux_customizable_types_path</pre>	Return path to customizable types file under the policy root directory.	selinux.h

Num.	Function Name	Description	Header File
112.	<pre>selinux_default_context_path</pre>	Return path to default_context file under the policy root directory.	selinux.h
113.	selinux_failsafe_context_path	Return path to failsafe_context file under the policy root directory.	selinux.h
114.	selinux_file_context_cmp	Compare two file contexts, return 0 if equivalent.	selinux.h
115.	<pre>selinux_file_context_homedir_path</pre>	Return path to file under the policy root directory.	selinux.h
116.	<pre>selinux_file_context_local_path</pre>	Return path to file under the policy root directory.	selinux.h
117.	<pre>selinux_file_context_path</pre>	Return path to file under the policy root directory.	selinux.h
118.	<pre>selinux_file_context_verify</pre>	Verify the context of the file 'path' against policy. Return 0 if correct.	selinux.h
119.	selinux_getenforcemode	Reads the /etc/selinux/config file and determines whether the machine should be started in enforcing (1), permissive (0) or disabled (-1) mode.	selinux.h
120.	selinux_getpolicytype	Reads the /etc/selinux/config file and determines what the default policy for the machine is. Calling application must free policytype.	selinux.h
121.	selinux_homedir_context_path	Return path to file under the policy root directory.	selinux.h
122.	<pre>selinux_init_load_policy</pre>	Perform the initial policy load. This function determines the desired enforcing mode, sets the the *enforce 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 selinuxfs mount required to perform these actions. The function returns 0 if everything including the policy load succeeds. In this case, init is expected to re-exec itself in order to transition to the proper security context. Otherwise, the function returns -1, and init must check *enforce to determine how to proceed. If enforcing (*enforce > 0), then init should halt the system. Otherwise, init may proceed normally without a re-exec.	selinux.h
123.	selinux_lsetfilecon_default	This function sets the file context on to the system defaults returns 0 on success.	selinux.h
124.	selinux_media_context_path	Return path to file under the policy root directory.	selinux.h

Num.	Function Name	Description	Header File
125.	selinux_mkload_policy	Make a policy image and load it. This function provides a higher level interface for loading policy than security_load_policy, 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 security_load_policy to load it. 'preservebools' 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.	selinux.h
126.	<pre>selinux_netfilter_context_path</pre>	Returns path to the netfilter_context file under the policy root directory.	selinux.h
127.	selinux_path	Returns path to the policy root directory.	selinux.h
128.	selinux_policy_root	Reads the /etc/selinux/config file and returns the top level directory.	selinux.h
129.	<pre>selinux_raw_context_to_color</pre>	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 free(). Returns -1 upon an error or 0 otherwise.	selinux.h
130.	selinux_raw_to_trans_context	Perform context translation between the human-readable format ("translated") and the internal system format ("raw"). Caller must free the resulting context via freecon. Returns -1 upon an error or 0 otherwise. If passed NULL, sets the returned context to NULL and returns 0.	selinux.h
131.	selinux_removable_context_path	Return path to file under the policy root directory.	selinux.h
132.	selinux_securetty_types_path	Return path to the securetty_types file under the policy root directory.	selinux.h
133.	selinux_set_mapping	Userspace class mapping support.	selinux.h
134.	selinux_trans_to_raw_context	Perform context translation between the human-readable format ("translated") and the internal system format ("raw"). Caller must free the resulting context via freecon. Returns -1 upon an error or 0 otherwise. If passed NULL, sets the returned context to NULL and returns 0.	selinux.h
135.	selinux_translations_path	Return path to setrans.conf file under the policy root directory.	selinux.h

Num.	Function Name	Description	Header File
136.	selinux_user_contexts_path	Return path to file under the policy root directory.	selinux.h
137.	selinux_users_path	Return path to file under the policy root directory.	selinux.h
138.	selinux_usersconf_path	Return path to file under the policy root directory.	selinux.h
139.	<pre>selinux_virtual_domain_context_path</pre>	Return path to file under the policy root directory.	selinux.h
140.	<pre>selinux_virtual_image_context_path</pre>	Return path to file under the policy root directory.	selinux.h
141.	selinux_x_context_path	Return path to x_context file under the policy root directory.	selinux.h
142.	set_matchpathcon_canoncon	Same as 'set_matchpathcon_invalidcon', but also allows canonicalization of the context, by changing *context to refer to the canonical form. If not set, and invalidcon is also not set, then this defaults to calling security_canonicalize_context().	selinux.h
143.	set_matchpathcon_flags	Set flags controlling operation of matchpathcon_init or matchpathcon: MATCHPATHCON_BASEONLY - Only process the base file_contexts file. MATCHPATHCON_NOTRANS - Do not perform any context translation. MATCHPATHCON_VALIDATE - Validate/canonicalize contexts at init time.	selinux.h
144.	set_matchpathcon_invalidcon	Set the function used by matchpathcon_init 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(). The function is also responsible for reporting any such error, and may include the 'path' and 'lineno' in such error messages.	selinux.h
145.	set_matchpathcon_printf	Set the function used by matchpathcon_init when displaying errors about the file_contexts configuration. If not set, then this defaults to fprintf(stderr, fmt,).	selinux.h
146.	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

Num.	Function Name	Description	Header File
147.	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. When possible, decompose your application and use setexeccon()+execve() instead. Note that the application may lose access to its open descriptors as a result of a setcon() unless policy allows it to use descriptors opened by the old context.	selinux.h
148.	setexeccon setexeccon_raw	Set exec security context for the next execve. Call with NULL if you want to reset to the default.	selinux.h
149.	setfilecon setfilecon_raw	Wrapper for the xattr API - Set file context.	selinux.h
150.	setfscreatecon setfscreatecon_raw	Set the fscreate security context for subsequent file creations. Call with NULL if you want to reset to the default.	selinux.h
151.	setkeycreatecon setkeycreatecon_raw	Set the keycreate security context for subsequent key creations. Call with NULL if you want to reset to the default.	selinux.h
152.	setsockcreatecon setsockcreatecon_raw	Set the sockcreate security context for subsequent socket creations. Call with NULL if you want to reset to the default.	selinux.h
153.	sidget	Increment SID reference counter. Increment the reference counter for @sid, indicating that @sid is in use by an (additional) object. Return the new reference count, or zero if @sid is invalid (has zero reference count). Note that avc_context_to_sid() also increments reference counts.	avc.h
154.	sidput	Decrement SID reference counter. Decrement the reference counter for @sid, indicating that a reference to @sid is no longer in use. Return the new reference count. When the reference count reaches zero, the SID is invalid, and avc_context_to_sid() must be called to obtain a new SID for the security context.	avc.h
155.	string_to_av_perm	Convert string names to access vector permissions.	selinux.h
156.	string_to_security_class	Convert string names to security class values.	selinux.h

9. Appendix D - SE-PostgreSQL Database Example

9.1 Introduction

This section gives a run through installing and running a very simple database to show some of the SE-PostgreSQL features. The "Security-Enhanced PostgreSQL Security Wiki" [Ref. 3] contains a more complete coverage of the principles, however it does not have a simple walk-through.

The areas covered are:

- 1. Install sepostgresql using yum. It assumes that postgresql or sepostgresql are not installed.
- 2. Initialise a database cluster so that sepostgresql can be started.
- 3. Create a database called testdb.
- 4. Using the PostgreSQL terminal client psql create a simple table with two columns and insert 4 rows (or tuples) of data demonstrating how to add and show the security context information associated with these objects. To enable the security context information to be distinguished between the various objects the following will be used:

Name	Object	Context used
Database (testdb)	db_database	unconfined_u:object_r:sepgsql_db_t:s0
Table (info)	db_table	<pre>unconfined_u:object_r:sepgsql_table_t:s0:c10</pre>
Column 1 (user_name)	db_column	<pre>unconfined_u:object_r:sepgsql_table_t:s0:c20</pre>
Column 2 (email_addr)	db_column	<pre>unconfined_u:object_r:sepgsql_table_t:s0:c30</pre>
Row 1	db_tuple	<pre>unconfined_u:object_r:sepgsql_table_t:s0:c100</pre>
Row 2	db_tuple	<pre>unconfined_u:object_r:sepgsql_table_t:s0:c110</pre>
Row 3	db_tuple	<pre>unconfined_u:object_r:sepgsql_table_t:s0:c120</pre>
Row 4	db_tuple	unconfined_u:object_r::unconfined_t:s0:c130

5. Finally run some sepostgresql specific functions and explain their results.

The following assumptions have been made:

- 1. The user has a basic knowledge of databases and the SQL language.
- 2. SE-PostgreSQL or PostgreSQL are not installed.
- 3. The system used is Fedora 12 with the targeted policy (selinux-policy-targeted-3.6.32-103.fc12.noarch). This would have installed the postgresql policy modules by default.
- 4. Generally when adding entries to a database SE-PostgreSQL will use a default security context, however in this walk-through all entries will have specific security context defined for them (except the database (testdb) that will use the SE-PostgreSQL default).

9.2 SE-PostgreSQL Walk-through

Install sepostgresql using yum. This will install all the required components including postgresql:

```
yum install sepostgresql
```

On the authors machine, the following were installed:

```
rpm -qa | grep postgresql
sepostgresql-8.4.2-2583.fc12.i686
postgresql-libs-8.4.3-1.fc12.i686
postgresql-8.4.3-1.fc12.i686
postgresql-server-8.4.3-1.fc12.i686
```

Ensure SELinux is in enforcing mode:

setenforce 1

Once sepostgresql is installed a database cluster needs to be initialised. As part of the sepostgresql installation an init script (/etc/init.d/sepostgresql) was added that will manage this process:

```
service sepostgresql initdb
Initializing database:
```

For information, the database cluster will be built by the above process in /var/lib/sepgsql/data. Note that an sepgsql user and group were also added as a part of the installation process:

```
ls -lZ /var/lib/sepgsql
drwx-----. 2 sepgsql sepgsql system_u:object_r:postgresql_db_t:s0 backups
drwx-----. 2 sepgsql sepgsql unconfined_u:object_r:postgresql_db_t:s0 data
```

Once the database cluster has been initialised it can be started by:

```
service sepostgresql start
Starting sepostgresql service: [ OK ]
```

This demo will create the test database and tables etc. as the sepgsql user:

su - sepgsql

Optionally, once logged on as the sepgsql user, the PostgreSQL createuser command can be used to allow other GNU / Linux users to access PostgreSQL by:

```
createuser [login_name]
# for example:
createuser root
Shall the new role be a superuser? (y/n) y
# This would allow root to use the PostgreSQL commands to manage
# the database as a superuser.
```

Now the testdb database itself needs to be created by the PostgreSQL createdb command:

createdb testdb

Once created, the PostgreSQL interactive terminal (psql) needs to be loaded so that SQL statements can be run against the database:

```
# This command will load psql and connect it to the testdb
# database:
psql testdb
```

Now that psql is active and connected to the testdb database SQL statements can be run. The first one is to display the security context of the database that requires some knowledge of how SE-PostgreSQL holds its internal parameters. As explained in the <u>SELinux PostgreSQL Support</u> section the main internal tables of interest are pg_database, pg_class, pg_attribute and pg_security, with pg_database holding the database name. Therefore if the following SQL statement is executed, the security context of the testdb database will be returned:

Now a table (info) will be created that will have two columns (user_name and email_addr). A unique security context will be given to each object created as follows:

```
testdb=# CREATE TABLE info
(user_name CHAR(10) SECURITY_CONTEXT =
'unconfined_u:object_r:sepgsql_table_t:s0:c20',
email_addr CHAR(20) SECURITY_CONTEXT =
'unconfined_u:object_r:sepgsql_table_t:s0:c30'
) SECURITY_CONTEXT =
'unconfined_u:object_r:sepgsql_table_t:s0:c10';
CREATE TABLE
```

Now that the table has been created, the security context of each object can be displayed by querying SE-PostgreSQL internal tables.

The SQL statement to retrieve the table object info security context is as follows, note that the pg_class internal table holds the table name:

The SQL statement to retrieve the column object user_name is as follows, note that the pg attribute internal table holds the column name:

```
testdb=# SELECT attname, security_context FROM pg_attribute
WHERE attname = 'user_name';
    attname | security_context
    user_name | unconfined_u:object_r:sepgsql_table_t:s0:c20
(1 row)
```

And the SQL statement to retrieve the column object email addr is as follows:

```
testdb=# SELECT attname, security_context FROM pg_attribute
WHERE attname = 'email_addr';
    attname | security_context
    email_addr | unconfined_u:object_r:sepgsql_table_t:s0:c30
  (1 row)
```

Now that the table and its columns have been created, it is now possible to insert information into the database. Each row (or tuple) will now be added with its own unique security context.

Insert Row 1:

```
testdb=# INSERT INTO info (security_context, user_name,
email_addr) VALUES
('unconfined_u:object_r:sepgsql_table_t:s0:c100', 'fred',
'fred@yahoo.com');
INSERT 0 1
```

Show Row 1 security context, note that only the table name info is specified (i.e. no internal table name required):

```
testdb=# SELECT user_name, email_addr, security_context FROM info;
user_name | email_addr | security_context
fred | fred@yahoo.com | unconfined_u:object_r:sepgsql_table_t:s0:c100
(1 row)
```

Insert Rows 2 and 3 each with a unique security context:

```
testdb=# INSERT INTO info (security_context, user_name,
email_addr) VALUES
('unconfined_u:object_r:sepgsql_table_t:s0:c110', 'derf',
'derf@hotmail.com');
```

```
INSERT 0 1
testdb=# INSERT INTO info (security_context, user_name,
email_addr) VALUES
('unconfined_u:object_r:sepgsql_table_t:s0:c120', 'george',
'george@hotmail.com');
INSERT 0 1
```

Show Rows 1, 2 and 3 security context:

testdb=#	SELECT	user_name,	em	ail_addr,	security	_context	FROM	info;
user_name	ema	ail_addr			security_	_context		
fred derf george (3 rows)	derf@l	yahoo.com notmail.com e@hotmail.com		unconfined unconfined unconfined	u:object	r:sepgsql	table	t:s0:c110

To demonstrate that SE-PostgreSQL will not allow entries to be entered unless the security context is valid, an entry will be made with a type of unconfined_t as this is not valid for the standard targeted policy. It is assumed that SELinux is in enforcing mode:

```
testdb=# INSERT INTO info (security_context, user_name,
email_addr) VALUES
('unconfined_u:object_r:unconfined_t:s0:c130', 'hidden',
'hidden@hotmail.com');
ERROR: SELinux: security policy violation
```

Now to demonstrate that SE-PostgreSQL will not display information that is not allowed by the policy, set SELinux to permissive mode:

setenforce 0

Now insert the row again:

```
testdb=# INSERT INTO info (security_context, user_name,
email_addr) VALUES
('unconfined_u:object_r:unconfined_t:s0:c130', 'hidden',
'hidden@hotmail.com');
INSERT 0 1
```

And then display the information:

```
testdb=# SELECT user_name, email_addr, security_context FROM info;
user_name | email_addr | security_context
fred | fred@yahoo.com | unconfined_u:object_r:sepgsql_table_t:s0:c100
derf | derf@hotmail.com | unconfined_u:object_r:sepgsql_table_t:s0:c110
george | george@hotmail.com | unconfined_u:object_r:sepgsql_table_t:s0:c120
hidden | hidden@hotmail.com | unconfined_u:object_r:unconfined_t:s0:c130
(4 rows)
```

Now set SELinux to enforcing mode:

setenforce 1

And then display the information, note that the 4th row is not displayed:

```
testdb=# SELECT user_name, email_addr, security_context FROM info;
user_name | email_addr | security_context
fred | fred@yahoo.com | unconfined_u:object_r:sepgsql_table_t:s0:c100
derf | derf@hotmail.com | unconfined_u:object_r:sepgsql_table_t:s0:c110
george | george@hotmail.com | unconfined_u:object_r:sepgsql_table_t:s0:c120
(3 rows)
```

9.2.1 SE-PostgreSQL Functions

This section will demonstrate some of the additional SE-PostgreSQL functions using the example database 'testdb'.

9.2.1.1 Get / Set Security Context Components

There are functions to get/set the security context components and this example shows the sepgsql get/set range function.

Show the range component:

Now change the range for the row that contains the user name of 'fred':

```
testdb=# UPDATE info SET security_context =
   sepgsql_set_range(security_context, 's0:c999') WHERE user_name
   = 'fred';
UPDATE 1
```

Now show the updated range component:

9.2.1.2 Get Connection Information

There are two functions to retrieve client and server context information and are shown below.

Get client context:

Get SE-PostgreSQL server context:

9.2.1.3 Reclaiming Unused Labels

When security contexts (labels) are no longer used, they are left in the pg_security table until they are reused or the space reclaimed. The following example runs the security_reclaim_label function that first returns 0, a row is then deleted (remember each row in the example database has a unique label) and the unused label space is then reclaimed.

Run function to check there whether there are labels to reclaim:

```
SELECT security_reclaim_label();
security_reclaim_label
1
(1 row)
```

The function claimed one label as the entry:

Was changed to:

10. Appendix E - General Information

10.1 Bugs and Features

This section lists the bugs and / or features found while writing this Notebook.

10.1.1 semanage - roles get deleted

The semanage user command deletes roles already assigned. To add a role of message filter r it seems logical to do the following:

semanage user -m -R "message filter r" user u

However need to know the other roles and add these as well, otherwise semanage will remove them from the current policy:

semanage user -m -R "message_filter_r unconfined_r" user_u

10.1.2 apol Not Loading

In the earlier F-12 release apol would not load from the Gnome menu even though installed correctly. The following command will load force a load from a virtual terminal session:

wish /usr/bin/apol

10.1.3 apol not showing all screen in window

On the authors system the policy analysis tool apol never displayed all the information in the window. Various configuration items were tried (editing the .apol file, changing screen formats etc.), but nothing worked. After further investigation (recompiling source etc.) it was found that the easiest way was to edit the /usr/bin/apol file (that is a text file) to use smaller font sizes. To achieve this find all 'helvetica' entries in the /usr/bin/apol file (there should be 30 of them -3 'helvetica' and 27 'Helvetica') and change all the font sizes to '8'.

10.1.4 racoon coredumps

As detailed in the Labeled IPSec Module Example section of the Sample Policy Source volume, racoon core dumps when using loopback with policies that do not support MCS or MLS services. Some of the tests listed in the <u>http://sourceforge.net/projects/audit-test</u> audit-test-2090-1.src.rpm were carried out and those using ports without MCS / MLS being configured do not work on loopback either. Running racoon -Fd as described in the Labeled IPSec Module Example section of the Sample Policy Source volume will provide a trace and running the core dump with gdb gives:

```
#0 0x0078616b in pk_recvacquire (mhp=0xbfc7a774) at pfkey.c:2048
2048 newpr->reqid_in = sp_in->req->saidx.reqid;
```

10.1.5 Red Hat SELinux Config Utility

The system-config-selinux utility does not like non MCS / MLS policies.

10.2 Root Login from Gnome

To be able to build and test the policies described in this Notebook users need to be logged in as 'root'. If there is a requirement to login as root from the Gnome login screen, then the following files can be modified to achieve this:

1. Login as root by:

su -

 Edit the /etc/pam.d/gdm and /etc/pam.d/gdm-password files with vi and comment out the following line in each file:

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

auth required pam succeed if.so user != root quiet

3. Save the files and when the system is re-booted it will be possible to login as root.

10.3 Building a Source rpm

To create the source rpm for the Notebook source (assuming the source is in \$HOME/notebook-source):

1. Create the notebook-source compressed archive using the tar command:

```
tar -cvzf notebook-source-1.1.0-1.tar.gz notebook-source/
```

2. Move the archive to the \$HOME/rpmbuild/SOURCES directory:

```
mv notebook-source-1.1.0-1.tar.gz rpmbuild/SOURCES
```

3. Create the notebook.spec file using an editor in the \$HOME/rpmbuild/SPECS directory with the following content:

```
Summary: Software for The SELinux Notebook - The Foundations
```

```
Name: notebook-source
Version: 1.1.0
Release: 1
License: GPLv2
Group: Experimental
Source0: %{name}-%{version}-%{release}.tar.gz
%description
This contains the sample source code and modules that are
detailed in The SELinux Notebook - The Foundations
%prep
%files
%defattr(-,root,root,-)
```

4. Build the rpm file from the \$HOME/rpmbuild/SPECS directory by running the rpmbuild command as follows, the results are also shown:

```
rpmbuild -ba notebook.spec
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.41L8I5
+ umask 022
+ cd /root/rpmbuild/BUILD
+ LANG=C
+ export LANG
+ unset DISPLAY
+ exit 0
Processing files: notebook-source-1.1.0-1
Checking for unpackaged file(s): /usr/lib/rpm/check-
files /root/rpmbuild/BUILDROOT/notebook-source-1.1.0-1.i386
Wrote: /root/rpmbuild/SRPMS/notebook-source-1.1.0-1.src.rpm
Wrote: /root/rpmbuild/RPMS/i386/notebook-source-1.1.0-
1.i386.rpm
```

The only file of interest is the /root/rpmbuild/SRPMS/notebooksource-1.1.0-1.src.rpm as that contains the source files, the i386 rpm will be empty as no code was built for distribution.

The rpm can be checked, installed and uncompressed on a different system by executing the following commands:

```
# This command will show the contents of the rpm:
rpm -qlp notebook-source-1.1.0-1.src.rpm
notebook-source-1.1.0-1.tar.gz
notebook.spec
```

```
# This command will extract the rpm contents into:
# $HOME/rpmbuild/SOURCES:
```

rpm -Uvh notebook-source-1.1.0-1.src.rpm

```
# This command will uncompress the archive and install the
# contents:
tar -xzf notebook-source-1.1.0-1.tar.gz
```

11. Appendix F – Document References

Ref.	Title	Author
1.	Fedora 12 – SELinux User Guide	Red Hat
2.	Gentoo SELinux Handbook	Gentoo
3.	Security-Enhanced PostgreSQL Security Wiki	SE-PostgreSQL
4.	SELinux Policy Module Primer	J. Brindle
5.	Polyinstantiation of directories in an SELinux system	R. Coker
6.	Implementing SELinux as a Linux Security Module	S. Smalley, C. Vance, W. Salamon
7.	Iptables Tutorial	O. Andreasson
8.	New secmark-based network controls for SELinux	J. Morris
9.	Transitioning to Secmark	Paul Moore
10.	Fallback Label Configuration Example	Paul Moore
11.	Leveraging IPSec for Distributed Authorization	Trent Jaeger
12.	IPSec HOWTO	Ralf Spenneberg
13.	Secure Networking with SELinux	J. Brindle
14.	SELinux by Example	F. Mayer, K Macmillan, D. Caplan
15.	SELinux From Scratch	S. Hallyn
16.	SELinux hardening for mmap_min_addr protections	E. Paris
17.	SELinux Support for Userspace Object Managers (the example userspace AVC code is in the Notebook source).	E. Walsh
18.	Application of the Flask Architecture to the X Window System Server	E. Walsh
19.	X Access Control Extension Specification	E. Walsh
20.	A secure web application platform powered by <u>SELinux</u>	K. Kohei
21.	Kernel-based Virtual Machine	Red Hat
22.	How Does Xen Work	Xen Project
23.	Xen Security Modules	G. Coker

12. Appendix G - 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. Secondarily, 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, 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.

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 Tenglish 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 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.