The SELinux Notebook

The Foundations

(3rd Edition)

0. Notebook Information

0.1 Copyright Information

0.2 Revision History

Edition	Date	Changes
1.0	20th Nov '09	First released.
2.0	8th May '10	Second release.
3.0	2nd September '12	Third release. Many minor updates and new sections on SELinux userspace libraries and implementing SELinux-aware applications and object managers plus SEAndroid. Volume 2 has been discontinued as the Notebook source tarball contains enough information to experiment with the example policy modules and code.

0.3 Acknowledgements

Logo designed by Máirín Duffy

0.4 Abbreviations

Term	Definition
apol	Policy analysis tool
AV	Access Vector

Term	Definition
AVC	Access Vector Cache
BLP	Bell-La Padula
CC	Common Criteria
CIL	Common Intermediate Language
CMW	Compartmented Mode Workstation
DAC	Discretionary Access Control
F-17	Fedora 17
FLASK	Flux Advanced Security Kernel - A security-enhanced version of the Fluke kernel and OS developed by the Utah Flux team and the US Department of Defence.
Fluke	Flux µ-kernel Environment - A specification and implementation of a micro kernel and operating system architecture.
Flux	The Flux Research Group (http://www.cs.utah.edu/flux/)
ID	Identification
LSM	Linux Security Module
LAPP	Linux, Apache, PostgreSQL, PHP / Perl / Python
LSPP	Labeled Security Protection Profile
MAC	Mandatory Access Control
MCS	Multi-Category Security
MLS	Multi-Level Security
NSA	National Security Agency
OM	Object Manager
OTA	over the air
PAM	Pluggable Authentication Module
RBAC	Role-based Access Control
rpm	Red Hat Package Manager
SELinux	Security Enhanced Linux
SID	Security Identifier
SL	Security Level
SLIDE	SELinux Integrated Development Environment
SMACK	Simplified Mandatory Access Control Kernel
SUID	Super-user Identifier
TE	Type Enforcement
UID	User Identifier
XACE	X (windows) Access Control Extension

0.5 Index

1. The SELinux Notebook

1.1 Introduction

This Notebook should help with explaining:

a) SELinux and its purpose in life.

b) The LSM / SELinux architecture, its supporting services and how they are implemented within GNU / Linux.

c) SELinux Networking, Virtual Machine, X-Windows, PostgreSQL and Apache/SELinux-Plus SELinux-aware capabilities.

d) The core SELinux 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.

g) The integration of SELinux within Android - SEAndroid.

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

Most sections of this Notebook have been added to the SELinux Project web site as part of the SELinux documentation project.

While the majority of this Notebook is based on Fedora 16 and 17, all additional developments as seen on the SELinux mail list (selinux@tycho.nsa.gov) up to August '12 have been added (e.g. new policy language statements: `default_user` `default_role` ..., support for `ptrace_child` and SEAndroid updates ...).

1.2 The Foundations Overview

This volume has the following major sections:

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

SELinux Configuration Files - Describes all the known SELinux configuration file with samples. Also lists any specific SELinux 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. Also an introduction to the new CIL language (Common Intermediate Language).

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

SEAndroid - A brief overview of the SELinux services used to support Android.

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

libselinux Functions - Describes the SELinux library functions and a list of example sources available in the Notebook tarball.

1.2.1 Notebook Source Overview

To demonstrate some of the SELinux capabilities a supporting Notebook source tarball is available (`notebook-source-3.0.tar.gz`). The tarball contains directories and READMEs covering the following:

Building a Basic Policy - Describes how to build monolithic, base and loadable policy modules using core policy language statements and SELinux commands. Note that these policies should not to be used in a live environment, they are examples to show simple policy construction. Then using the basic policy with additional module:

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

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.

Example `libselinux` applications - This contains over 100 samples that use all `libselinux` 2.1.6 functions. To save typing long context strings it makes use of a configuration file. There are also some supporting policy modules for the F-16 / F-17 `targeted` policy to show how the functions work.

Experimenting with X-Windows - Builds a sample selection manager application to demonstrate polyinstantiated selections. Also has a simple test application for the XSELinux extension Get/Set functions (this also uses `python-xcb` that is not distributed with Fedora).

Experimenting with PostgreSQL 9.1 using sepgsql - This shows how to create a simple database that uses SELinux functionality. This is then expanded to demonstrate adding additional functions to support `libselinux`. There are also demos using Apache with threads (`mod_selinux`), PHP, Labeled IPSec and NetLabel. The policy modules supplied have been tested using F-16 / F-17 `targeted` policy.

2. SELinux Overview

2.1 Introduction

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

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

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

Description	Old Location	New Location
The SELinux filesystem that interfaces with the kernel based security server.	`/selinux`	`/sys/fs/selinux`
The SELinux configuration directory that holds the sub-system configuration files and policies.	`/etc/selinux`	No change
The SELinux policy store that holds policy modules and configuration details	`/etc/selinux`	`/var/lib/selinux`

2.1.1 Is SELinux useful

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

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

1. If SELinux is enabled, the policy defines what access to resources and operations on them (e.g. read, write) are allowed (i.e. SELinux stops all access unless allowed by policy). This is why SELinux is called a 'mandatory access control' (MAC) system.
2. The policy design, implementation and testing against a defined security policy or requirements is important, otherwise there could be 'a false sense of security'.

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

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

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

   However with careful design and known security goals the SELinux 'Apache / SELinux Plus' services could be used to build a more secure web service (also see http://code.google.com/p/sepgsql/wiki/Apache_SELinux_plus).
2. SELinux services have been added to Andriod, producing SEAndroid. The presentation "The Case for Security Enhanced (SE)Android" gives use-cases

and types of Android exploits that SELinux could have overcome. The presentation is available at:

https://events.linuxfoundation.org/images/stories/pdf/lf_abs12_smalley.pdf

2.2 Core SELinux Components

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

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

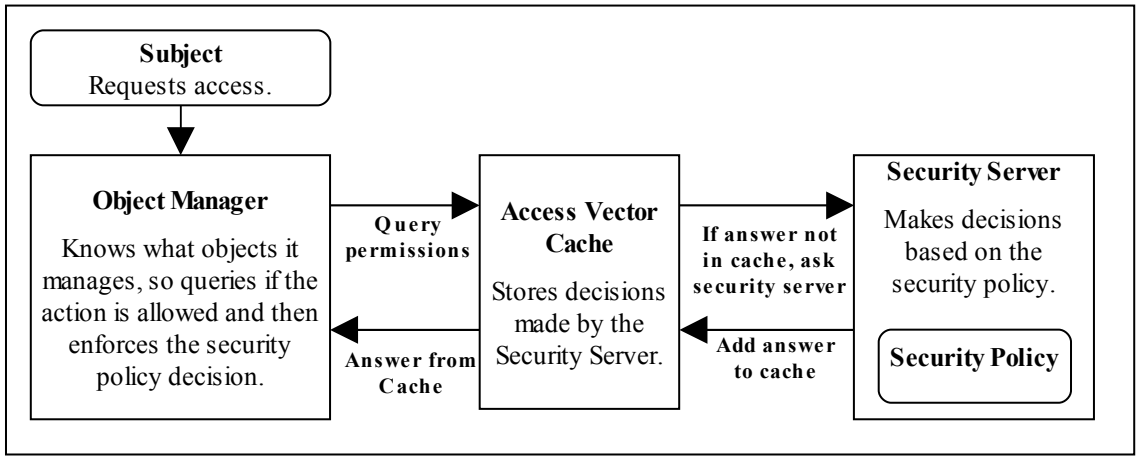


Figure 2.1: High Level Core SELinux Components - *Decisions by the Security Server are cached in the AVC to enhance performance of future requests.*

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

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

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

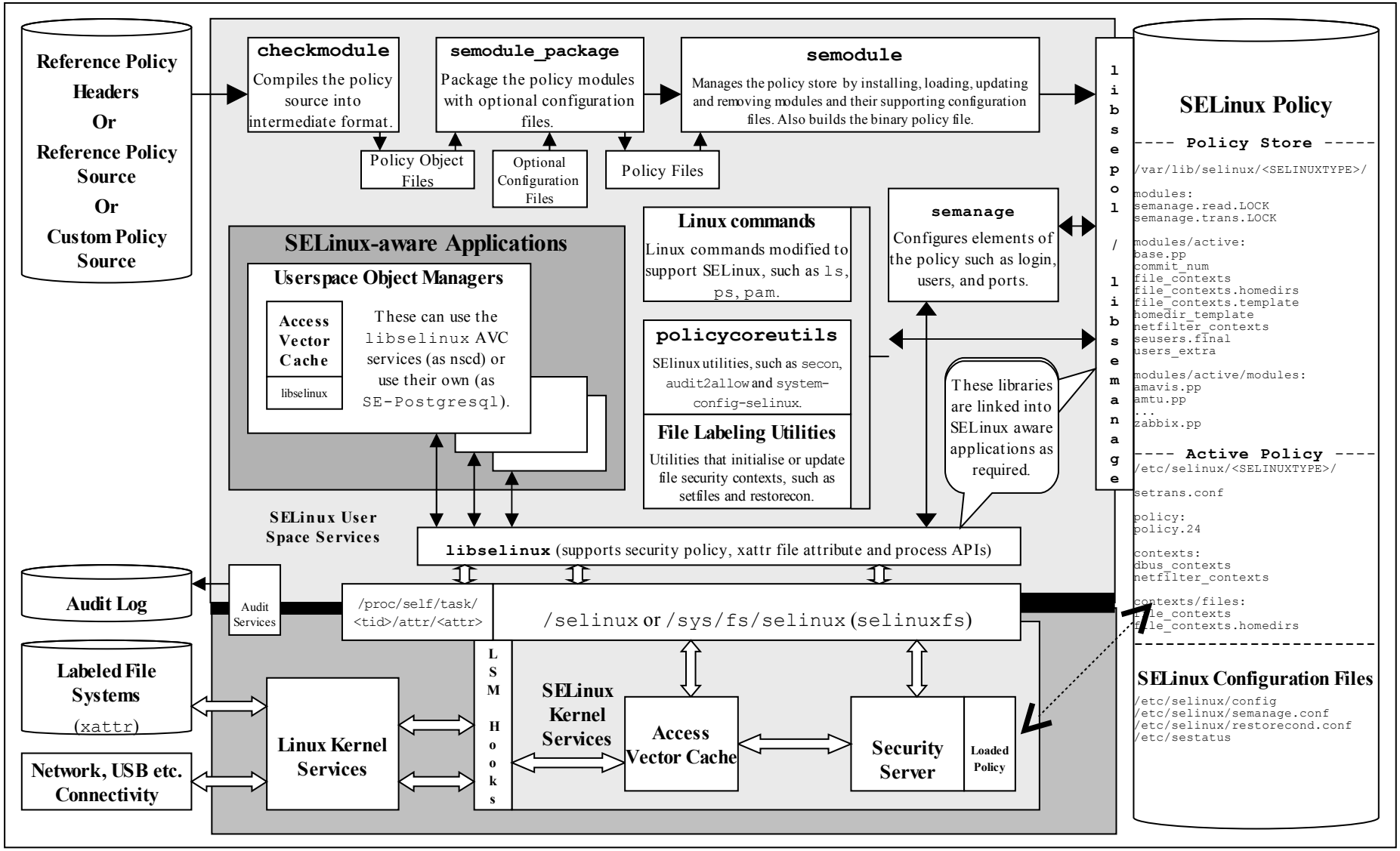


Figure 2.2: High Level SELinux Architecture – *Showing the major supporting services*

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 LSM section. The SELinux kernel AVC service is used to cache the security servers response to the kernel based object managers thus speeding up access decisions should the same request be asked in future.

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

b) The SELinux security policy (right hand side of Figure 2.2) and its supporting configuration files are contained in the `/etc/selinux` directory. This directory contains the main SELinux configuration file (`config`) that has the name of the policy to be loaded (via the `SELINUXTYPE` entry) and the initial enforcement mode[1] of the policy at load time (via the `SELINUX` entry). The `/etc/selinux/<SELINUXTYPE>` directories contain policies that can be activated along with their configuration files (e.g. '`SELINUXTYPE=targeted`' will have its policy and associated configuration files located at `/etc/selinux/targeted`). All known configuration files for are shown in the SELinux Configuration Files section.

c) SELinux supports a 'modular policy', this means that a policy does not have to be one large source policy but can be built from modules. A modular policy consists of a base policy that contains the mandatory information (such as object classes, permissions etc.), and zero or more policy modules where generally each supports a particular application or service. These modules are compiled, linked, and held in a 'policy store' where they can be built into a binary format that is then loaded into the security server (in the diagram the binary policy is located at `/etc/selinux/targeted/policy/policy.26`). The types of policy and their construction are covered in the Types of SELinux Policy section.

d) To be able to build the policy in the first place, policy source is required (top left hand side of Figure 2.2). This can be supplied in two basic ways (and soon a third when the CIL (Common Intermediate Language) development is complete):

   i) as source code written using the SELinux Policy Language. This is how the simple policies have been written to support the examples in

[1] When SELinux is enabled, the policy can be running in 'permissive mode' (`SELINUX=permissive`), where all accesses are allowed. The policy can also be run in 'enforcing mode' (`SELINUX=enforcing`), where any access that is not defined in the policy is denied and an entry placed in the audit log. SELinux can also be disabled (at boot time only) by setting `SELINUX=disabled`.

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. This is the standard way policies are now built for SELinux distributions such as Red Hat and Debian and is discussed in the Reference Policy section.

e) To be able to compile and link the source code then load it into the security server requires a number of tools (top of Figure 2.2). These are used to build the sample policy modules where their use is described.

f) To enable system administrators to manage the policy, the SELinux environment and label file systems requires tools and modified GNU / Linux commands. These are mentioned throughout the Notebook as needed and summarised in Appendix B – SELinux Commands. Note that there are many other applications to manage policy, however this Notebook only concentrates on the core services.

g) To ensure security events are logged, GNU / Linux has an audit service that captures policy violations. The Auditing SELinux Events section describes the format of these security events.

h) SELinux supports network services that are described in the SELinux Networking Support section.

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

2.3 Mandatory Access Control (MAC)

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

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

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

Note that the subject (and therefore the user) cannot decide to bypass the policy rules being enforced by the MAC policy with SELinux enabled. Contrast this to standard Linux Discretionary Access Control (DAC), which also governs the ability of subjects to access objects, however it allows users to make policy decisions. The steps in the decision making chain for DAC and MAC are shown in Figure 2.3.

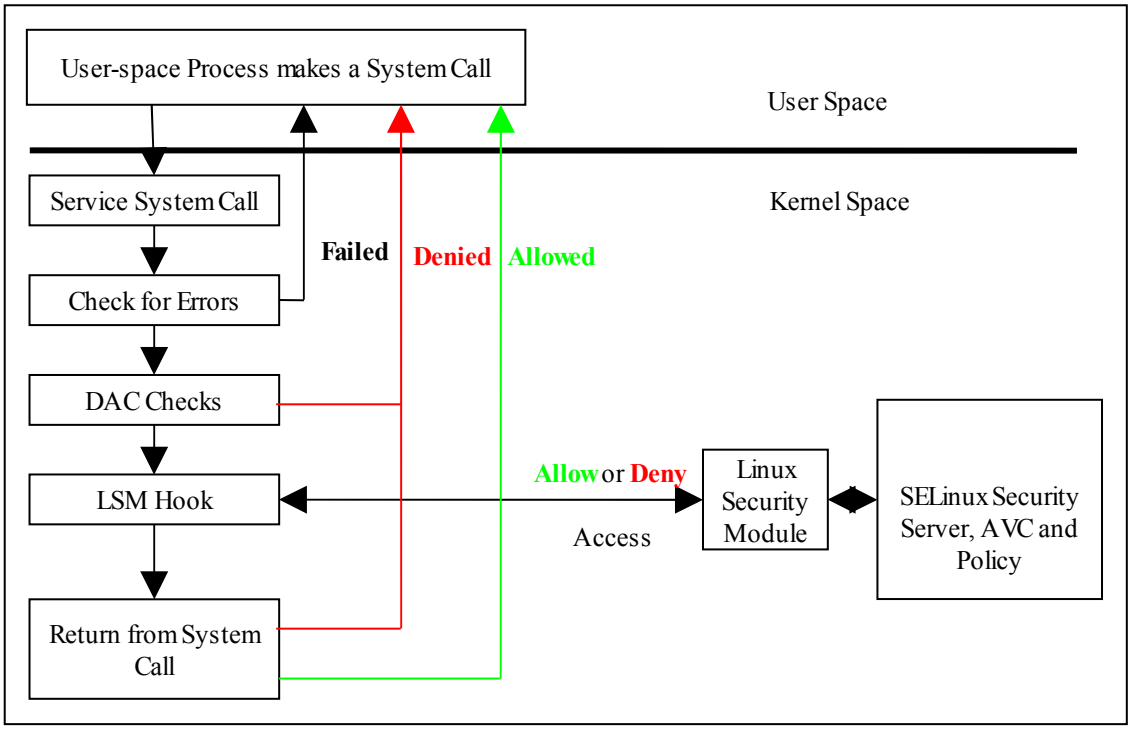


Figure 2.3: Processing a System Call – *The DAC checks are carried out first, if they pass then the Security Server is consulted for a decision.*

SELinux supports two forms of MAC:

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

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

2.4 SELinux Users

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

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

The SELinux user name is the first component of a 'security context' and by convention SELinux user names end in '_u', however this is not enforced by any SELinux service (i.e. it is only to identify the user component).

2.5 Role-Based Access Control (RBAC)

To further control access to TE domains SELinux makes use of role-based access control (RBAC). This feature allows SELinux users to be associated to one or more roles, where each role is then associated to one or more domain types as shown in Figure 2.4.

The SELinux role name is the second component of a 'security context' and by convention SELinux roles end in '_r', however this is not enforced by any SELinux service (i.e. it is only used to identify the role component).

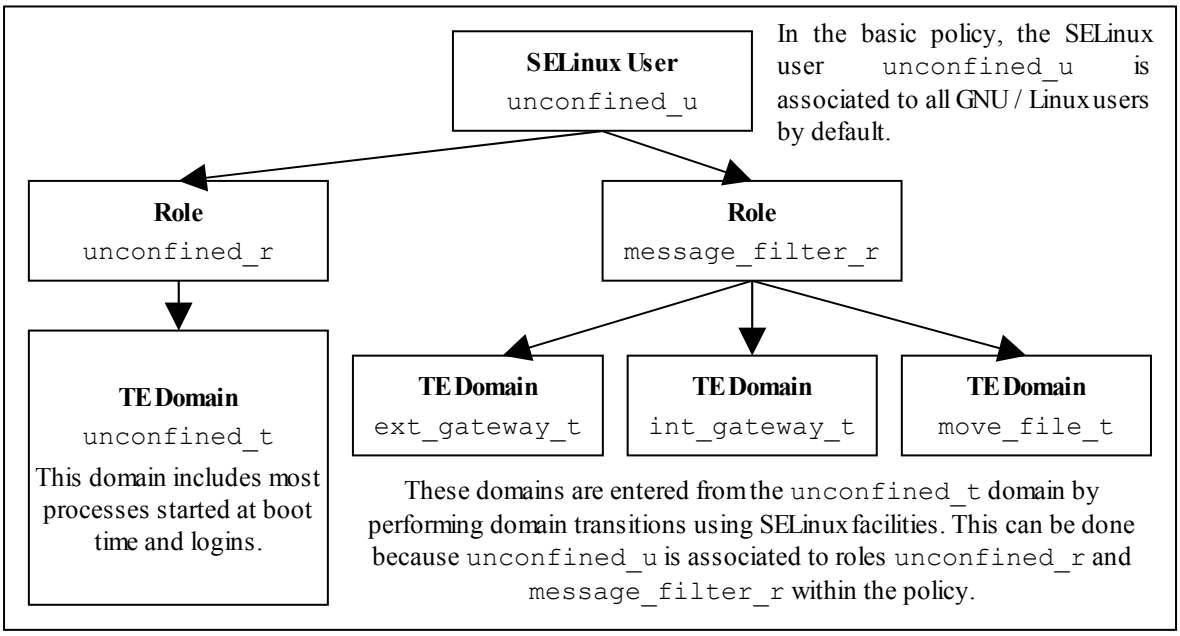


Figure 2.4: Role Based Access Control – *Showing how SELinux controls access via user, role and domain type association.*

2.6 Type Enforcement (TE)

SELinux makes use of a specific style of type enforcement[2] (TE) to enforce mandatory access control. For SELinux it means that all subjects and objects have a type identifier associated to them that can then be used to enforce rules laid down in a policy.

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

Because the type identifier (or just 'type') is associated to all subjects and objects, it can sometimes be difficult to distinguish what the type is actually associated with (it's not helped by the fact that by convention, type identifiers all end in '_t'). In the end

[2] There are various 'type enforcement' technologies.

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 Linux process or collection of processes (a domain or domain type). If the type identifier is used to reference an object then it is specifying its object type (i.e. file type).

While SELinux refers to a subject as being an active process that is associated to a domain type, the scope of an SELinux type enforcement domain can vary widely. For example in the simple policy built in the `basic-selinux-policy` directory of the source tarball, all the processes on the system run in the `unconfined_t` domain, 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 policy statements are added to the simple policy, that areas start to be confined. For example, an external gateway is run in its own isolated domain (`ext_gateway_t`) that cannot be 'interfered' with by any of the `unconfined_t` processes (except to run or transition the gateway process into its own domain). This scenario is similar to the 'targeted' policy delivered as standard in Red Hat Fedora where the majority of user space processes run under the `unconfined_t` domain (although don't think the simple policies implemented in source tarball are equivalent to the Reference Policy, they are not - so do not use them as live implementations).

The SELinux type is the third component of a 'security context' and by convention SELinux types end in '`_t`', however this is not enforced by any SELinux service (i.e. it is only used to identify the type component).

2.6.1 Constraints

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

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

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

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

This states that a process transition can only occur if the source role is the same as the target role, therefore a constraint is a condition that must be satisfied in order for one or more permissions to be granted (i.e. a constraint imposes additional restrictions on TE rules).

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

2.7 Security Context

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

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

Within SELinux, a security context is represented as variable-length strings that define the SELinux user[3], their role, a type identifier and an optional MCS / MLS security range or level as follows:

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

Where:

`user`	The SELinux user identity. This can be associated to one or more roles that the SELinux user is allowed to use.
`role`	The SELinux role. This can be associated to one or more types the SELinux user is allowed to access.
`type`	When a type is associated with a process, it defines what processes (or domains) the SELinux user (the subject) can access. When a type is associated with an object, it defines what access permissions the SELinux user has to that object.
`range`	This field can also be know as a `level` 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 Security Levels section.

However note that:

1. Access decisions regarding a subject make use of all the components of the security context.
2. Access decisions regarding an object make use of the components as follows:
    a) the `user` is either set to a special user called `system_u` or it is set to the SELinux user id of the creating process (as it serves no real purpose other than it can be used for audit purposes within logs).
    b) the `role` is generally set to a special SELinux internal role of `object_r`, although policy version 26 with kernel 2.6.39 and above do support role transitions on any object class.

[3] An SELinux user id is not the same as the GNU / Linux user id. The GNU / Linux user id is mapped to the SELinux user id by configuration files.

Therefore for an object the role, type and level/range 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 Computing Security Contexts section decribes how SELinux computes the security context components based on a source context, target context and an object class.

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

Example Process Security Context:

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

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

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

Example Object Security Context:

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

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

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

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

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

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

# Note that they are instantiated with unconfined_u as that was
# the SELinux user id of the process that created the files
```

```
# (see the process example above). The role remained as
object_r.
```

2.8 Subjects

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

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

1. A running process (i.e. an active entity) is a subject because it causes information to flow among objects or can change the system state.
2. The process can also be referred to as an object because each process has an associated object class[4] called '`process`'. This process 'object', defines what permissions the policy is allowed to grant or deny on the active process.

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

In SELinux subjects can be:

Trusted – Generally these are commands, applications etc. that have been written or modified to support specific SELinux functionality to enforce the security policy (e.g. the kernel, init, pam, xinetd and login). However, it can also cover any application that the organisation is willing to trust as a part of the overall system. Although (depending on your paranoia level), the best policy is to trust nothing until it has been verified that it conforms to the security policy. Generally these trusted applications would run in either their own domain (e.g. the audit daemon could run under `auditd_t`) or grouped together (e.g. the **`semanage`**`(8)` and **`semodule`**`(8)` commands could be grouped under `semanage_t`).

Untrusted – Everything else.

2.9 Objects

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

2.9.1 Object Classes and Permissions

Each object consists of a class identifier that defines its purpose (e.g. `file`, `socket`) along with a set of permissions[5] 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

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

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

context (e.g. `system_u:object_r:selinux_config_t`) as shown in Figure 2.5.

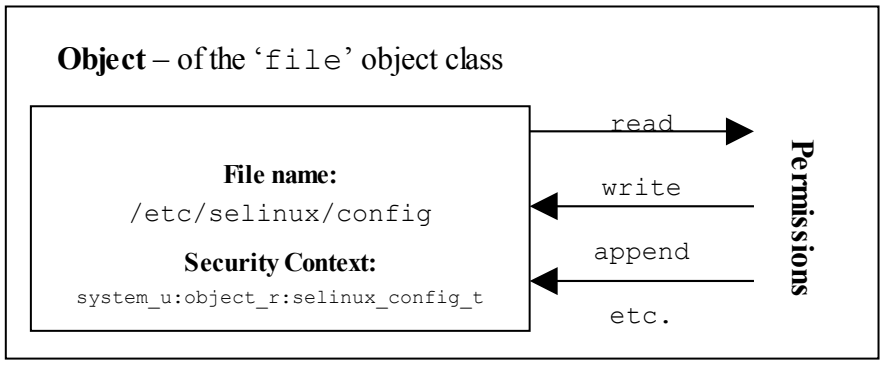


Figure 2.5: Object Class = 'file' and permissions – *the policy rules would define those permissions allowed for each process that needs access to the `/etc/selinux/config` file.*

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

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

The object classes consist of kernel object classes (for handling files, sockets etc.) plus userspace object classes for userspace object managers (for services such as X-Windows or dbus). The number of object classes and their permissions can vary depending on the features configured in the GNU / Linux release. All the known object classes and permissions are described in Appendix A - Object Classes and Permissions.

2.9.2 Allowing a Process Access to Resources

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

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

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

The following `allow` rule and Figure 2.6 illustrates 'a process can also be an object' as it allows processes running in the `unconfined_t` domain, permission to '`transition`' the external gateway application to the `ext_gateway_t` domain once it has been executed:

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

Where:

`allow`	The SELinux language `allow` rule.
`unconfined_t`	The source domain (or subject) identifier – in this case the `shell` that wants to exec the gateway application.
`ext_gateway_t`	The target object identifier – the object instance of the gateway application process.
`process`	The target object class - the '`process`' object class.
`transition`	The permission granted to the source domain on the targets object – in this case the `unconfined_t` domain has `transition` permission on the `ext_gateway_t` 'process' object.

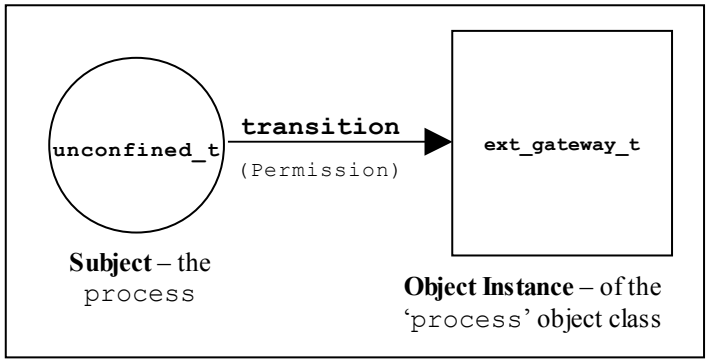


Figure 2.6: The `allow` rule – *Showing that the subject (the processes running in the `unconfined_t` domain) has been given the transition permission on the `ext_gateway_t` 'process' object.*

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

2.9.3 Labeling Objects

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

1. Inherit their labels from the parent process or object.
2. The policy type, role and range transition statements allow a different label to be assigned as discussed in the Domain and Object Transitions section.
3. SELinux-aware applications can enforce a new label (with the policies approval of course) using the `libselinux` API functions.

4. An object manager (OM) can enforce a default label that can either be built into the OM or obtained via a configuration file (such as X-Windows, NetLabel Labeled IPSec, and SECMARK (iptables).
5. Use an 'initial security identifier' (or initial SID). These are defined in all base and monolithic policies and are used to either set an initial context during the boot process, or if an object requires a default (i.e. the object does not already have a valid context).

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

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

2.9.3.1 Labeling Extended Attribute Filesystems

The labeling of file systems that implement extended attributes[6] is supported by SELinux using:

1. The `fs_use_xattr` statement within the policy to identify what file systems use extended attributes. This statement is used to inform the security server how the file system is labeled.
2. A 'file contexts' file that defines what the initial contexts should be for each file and directory within the file system. The format of this file is described in the `modules/active/file_contexts.template` file[7] section.
3. A method to initialise the filesystem with these extended attributes. This is achieved by SELinux utilities such as **`fixfiles`**`(8)` and **`setfiles`**`(8)`. There are also commands such as **`chcon`**`(1)`, **`restorecon`**`(8)` and **`restorecond`**`(8)` that can be used to relabel files.

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

```
ls -Z myfile
-rw-r--r-- 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:
```

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

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

```
# system_u:object_r:admin_home_t:s0
```

2.9.3.1.1 Copying and Moving Files

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

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

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

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

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


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

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

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


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

However, if the `restorecond` daemon is running:

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

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

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

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

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

2.9.3.2 Labeling Subjects

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

The 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 label components or labels to processes such as:

`user`, `role` and `type` statements.

and manage their scope:

`role allow` and `constrain`

and manage their transition:

`type_transition`, `role_transition` and `range_transition`

2.9.4 Object Reuse

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

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

2.10 Computing Security Contexts

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

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

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

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

`avc_compute_create``(3)` and **`security_compute_create`**`(3)`

avc_compute_member(3) and **security_compute_member**(3)

security_compute_relabel(3)

Note that the kernel has equivalent functions in the security server, however they are not covered here.

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

type_transition, role_transition, range_transition, type_member and type_change and also their corresponding CIL language statements: typetransition / filetransition, roletransition, rangetransition, typemember and typechange. There are also the default_user, default_role, default_type and default_range statements that will be available in later releases.

The sections that follow explain how security contexts are computed when using the libselinux functions and the policy statements that influence the outcome (note that the equivalent kernel services behave exactly the same).

2.10.1 avc_compute_create and security_compute_create

The table below[8] shows how the components from the source context scon, target context tcon and class tclass are used to compute the new context newcon (referenced by SIDs for **avc_compute_create**(3). The following notes also apply:

a) Any valid policy role_transition, type_transition and range_transition enforcement rules will influence the final outcome as shown.

b) For kernels less than 2.6.39 the context generated will depend on whether the class is process or any other class.

c) For kernels 2.6.39 and above the following also applies:

   i. Those classes suffixed by socket will also be included in the process class outcome.

   ii. If a valid role_transition rule for tclass, then use that instead of the default object_r. Also requires policy version 26 or greater - see **security_policyvers**(3).

   iii. If the type_transition rule is classed as the 'file name transition rule' (i.e. it has an object_name parameter), then provided the object name in the rule matches the last component of the objects name (in this case a file or directory name), then use the rules default_type (note CIL uses the filetransition rule). Also requires policy version 25 or greater.

d) For kernels 3.5 and above with policy version 27 or greater, the default_user, default_role, default_range statements will influence the user, role and range of the computed context for the specified class tclass. With policy version 28 or greater the

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

default_type statement can also influence the type in the computed context.

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

2.10.2 avc_compute_member and security_compute_member

The table below[9] shows how the components from the source context, scon target context, tcon and class, tclass are used to compute the new context newcon (referenced by SIDs for **avc_compute_member**(3). The following notes also apply:

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

a) Any valid policy `type_member` enforcement rules will influence the final outcome as shown.

b) For kernels less than 2.6.39 the context generated will depend on whether the class is `process` or any `other` class.

c) For kernels 2.6.39 and above, those classes suffixed by `socket` are also included in the `process` class outcome.

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

user	role	type	range
If kernel >= 3.5 with a **default_user** tclass **source** rule then use scon user **OR** If kernel >= 3.5 with a **default_user** tclass **target** rule then use tcon user **ELSE** Use tcon user	If kernel >= 3.5 with **default_role** tclass **source** rule then use scon role **OR** If kernel >= 3.5 with **default_role** tclass **target** rule then use tcon role **OR** If kernel >= 2.6.39 and tclass is **process** or *socket, then use scon role **OR** If kernel <= 2.6.38 and tclass is **process**, then use scon role **ELSE** Use **object_r**	If there is a valid **type_member** rule then use the rules member_type **OR** If kernel >= 3.5 with **default_type** tclass **source** rule then use scon type **OR** If kernel >= 3.5 with **default_type** tclass **target** rule then use tcon type **OR** If kernel >= 2.6.39 and tclass is **process** or *socket, then use scon type **OR** If kernel <= 2.6.38 and tclass is **process**, then use scon type **ELSE** Use tcon type	If kernel >= 3.5 with **default_range** tclass **source low** rule then use scon low **OR** If kernel >= 3.5 with **default_range** tclass **source high** rule then use scon high **OR** If kernel >= 3.5 with **default_range** tclass **source low_high** rule then use scon range **OR** If kernel >= 3.5 with **default_range** tclass **target low** rule then use tcon low **OR** If kernel >= 3.5 with **default_range** tclass **target high** rule then use tcon high **OR** If kernel >= 3.5 with **default_range** tclass **target low_high** rule then use tcon range **OR** If kernel >= 2.6.39 and tclass is **process** or *socket, then use scon range **OR** If kernel <= 2.6.38 and tclass is **process**, then use scon range **ELSE** Use scon low

2.10.3 security_compute_relabel

The table below[10] shows how the components from the source context, scon target context, tcon and class, tclass are used to compute the new context newcon for **security_compute_relabel**(3). The following notes also apply:

a) Any valid policy type_change enforcement rules will influence the final outcome shown in the table.

b) For kernels less than 2.6.39 the context generated will depend on whether the class is process or any other class.

c) For kernels 2.6.39 and above, those classes suffixed by **socket** are also included in the process class outcome.

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

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

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

2.11 Domain and Object Transitions

This section discusses the type_transition statement that is used to:

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

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

2.11.1 Domain Transition

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

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

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

2. SELinux-aware applications can specify the domain of the new process using the `libselinux` API call **`setexeccon`**`(3)`. To achieve this the SELinux-aware application must also have the `setexec` permission, for example:

```
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 the source tarball) that will be used to explain the transition process[11]:

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

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

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

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

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

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

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

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

[11] 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`.

```
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.

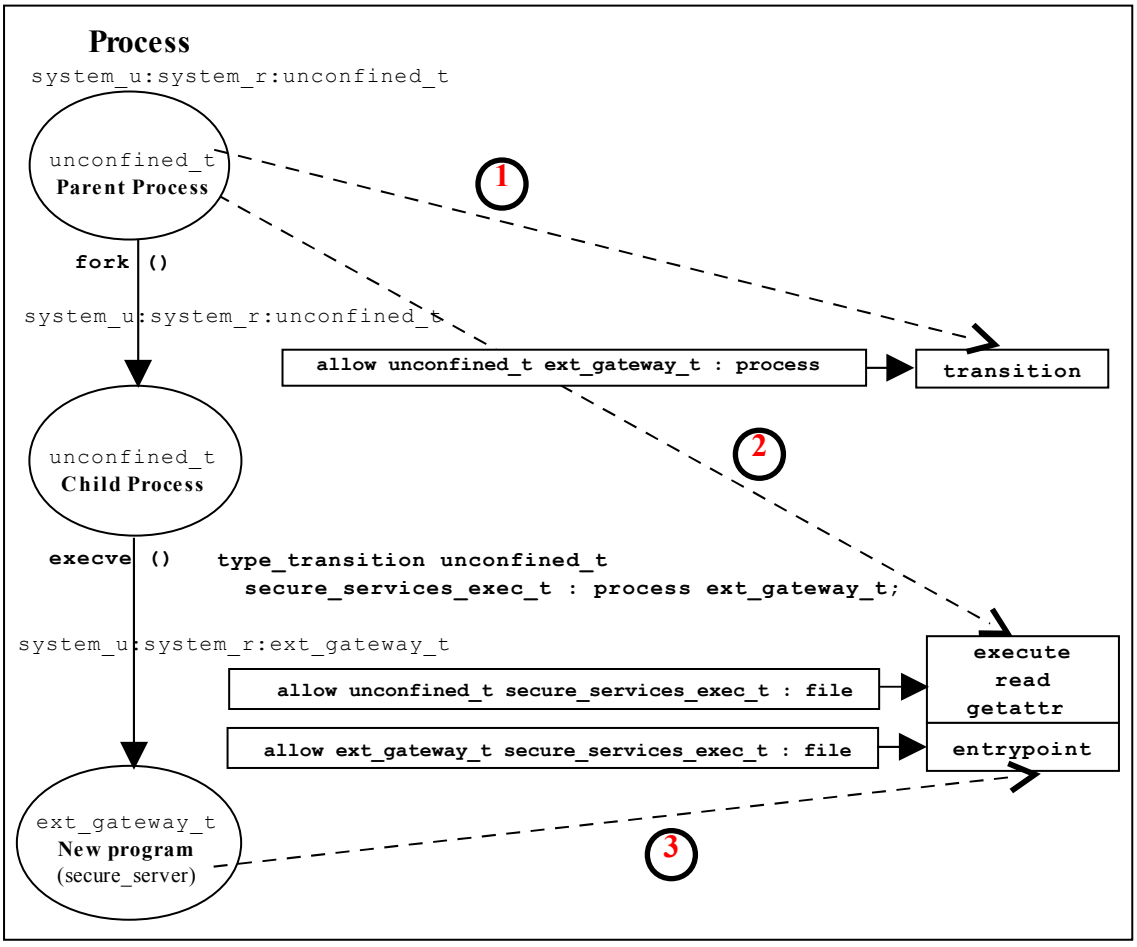


Figure 2.7: Domain Transition – *Where the secure_server is executed within the `unconfined_t` domain and then transitioned to the `ext_gateway_t` domain.*

2.11.1.1 Type Enforcement Rules

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

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

and the `int_gateway_t` statement would be:

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

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

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

This happened because the type enforcement rules will only allow a single 'default' type for a given source and target (see the Type Enforcement Rules section). In the above case there were two `type_transition` statements with the same source and target, but different default domains. The `ext_gateway.conf` module had the following statements:

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

And the `int_gateway.conf` module had the following statements:

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

While the allow rules are valid to enable the transitions to proceed, the two `type_transition` statements had different 'default' types (or target domains), that 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 that is defined in the
   # policy.
   ```

The `runcon` command makes use of a number of `libselinux` API functions to check the current context and set up the new context (for example **`getfilecon`**`(3)`

is used to get the executable files context and **setexeccon**(3) is used to set the new process context). If 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' into the int_gateway_t domain with the message_filter_r role. The runcon source can be found in the coreutils package.

Other ways to resolve this issue are:

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

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

2.11.2 Object Transition

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

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

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

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

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

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

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

However, as stated above to be able to create the file, the following minimum permissions need to be granted in the policy using `allow` rules, where:

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

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

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

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

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

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

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

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

2.12 Multi-Level Security and Multi-Category Security

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

Figure 2.8 shows a simple diagram where security levels represent the classification of files within a file server. The security levels are strictly hierarchical and conform to the Bell-La Padula model (BLP) in that (in the case of SELinux) a process (running at the 'Confidential' level) can read / write at their current level but only read down levels or write up levels (the assumption here is that the process is authorised).

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

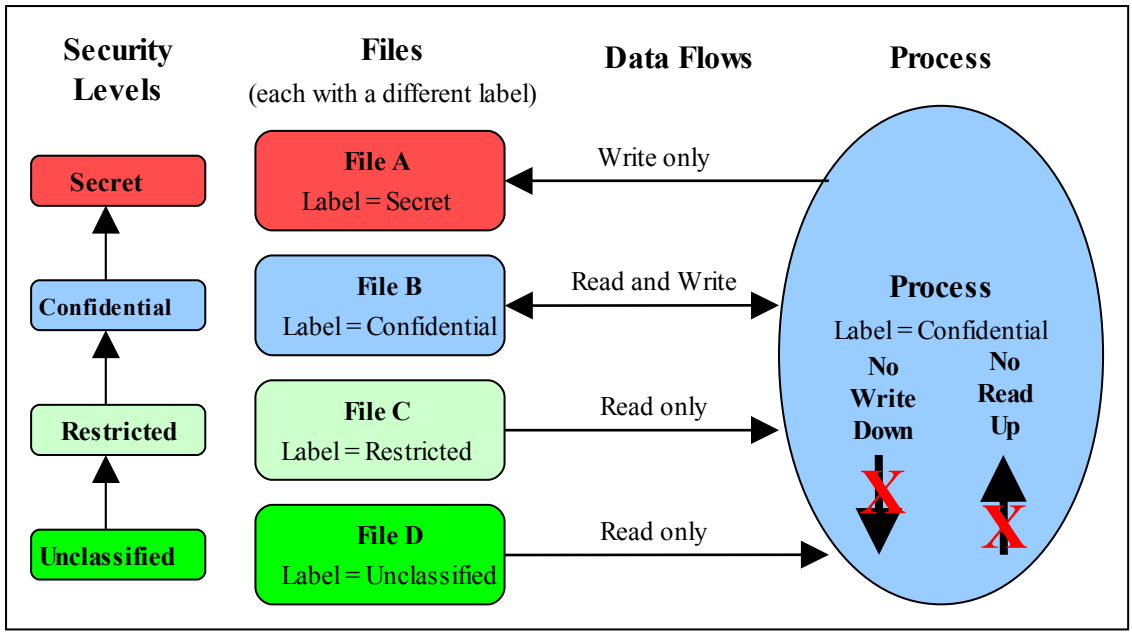


Figure 2.8: Security Levels and Data Flows – *This shows how the process can only 'Read Down' and 'Write Up' within an MLS enabled system.*

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

However, as always life is not so simple as:

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

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

2.12.1 Security Levels

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

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

<table>
<tr><td colspan="2">Security Level (or Level)
Consisting of a <code>sensitivity</code> and zero or more <code>category</code> entries:</td><td rowspan="2">Note that SELinux uses <code>level</code>, <code>sensitivity</code> and <code>category</code> in the language statements (see the MLS Language Statements section), however when discussing these the following terms can also be used: labels, classifications, and compartments.</td></tr>
<tr><td colspan="2"><code>sensitivity [: category, ...]</code>

also known as:
Sensitivity Label
Consisting of a <code>classification</code> and <code>compartment</code>.</td></tr>
<tr><td colspan="3">← Range →</td></tr>
<tr><td>Low</td><td rowspan="3">–</td><td>High</td></tr>
<tr><td><code>sensitivity [: category, ...]</code></td><td><code>sensitivity [: category, ...]</code></td></tr>
<tr><td>For a <code>process</code> or <code>subject</code> this is the current level or sensitivity</td><td>For a <code>process</code> or <code>subject</code> this is the Clearance</td></tr>
<tr><td>For an <code>object</code> this is the current level or sensitivity</td><td></td><td>For an <code>object</code> this is the maximum <code>range</code></td></tr>
<tr><td>SystemLow</td><td rowspan="2"></td><td>SystemHigh</td></tr>
<tr><td>This is the lowest level or classification for the system (for SELinux this is generally ‘<code>s0</code>’, note that there are no categories).</td><td>This is the highest level or classification for the system (for SELinux this is generally ‘<code>s15:c0,c255</code>’, although note that they will be the highest set by the policy).</td></tr>
</table>

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

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

2.12.1.1 MLS / MCS Range Format

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

```
user:role:type:sensitivity[:category,...]  -  sensitivity [:category,...]
---------------▼-----------------------▼-----▼-------------------------▼
               |         level         |  -  |          level          |
               |                                                       |
               |                          range                        |
```

Where:

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

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

2.12.1.2 Translating Levels

When writing policy for MLS / MCS security level components it is usual to use an abbreviated form such as `s0`, `s1` etc. to represent sensitivities and `c0`, `c1` etc. to represent categories. This is done simply to conserve space as they are held on files as extended attributes and also in memory. So that these labels can be represented in human readable form, a translation service is provided via the `setrans.conf` configuration file that is used by the **`mcstransd`**`(8)` daemon. For example `s0` = Unclassified, `s15` = Top Secret and `c0` = Finance, `c100` = Spy Stories. The **`semanage`**`(8)` command can be used to set up this translation and is shown in the `setrans.conf` configuration file section.

2.12.2 Managing Security Levels via Dominance Rules

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

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

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

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

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

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

As the MLS `dominance` statement is used to enforce the sensitivity hierarchy, the security levels now follow that sequence (lowest = `s0` to highest = `s3`) with the categories being unordered lists of 'compartments'. To allow the process access to files within its scope and within the dominance rules, the process will be constrained by using the `mlsconstrain` statement as illustrated in Figure 2.9.

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

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

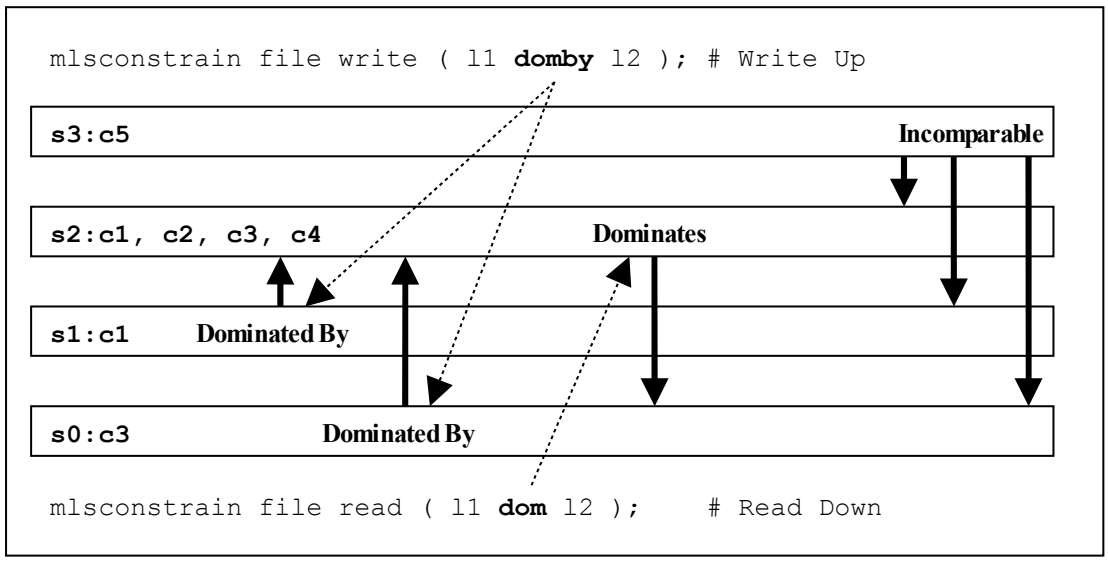


Figure 2.9: Showing the `mlsconstrain` Statements controlling Read Down & Write Up – *This ties in with Table 2 that shows a process running with a security range of `s0 - s3:c1.c5`.*

Using Figure 2.9:

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

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

   Target level = `s2:c1.c4`

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

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

   Source level = `s2:c1.c4`

   Target level = `s0:c3`

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

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

2.12.3 MLS Labeled Network and Database Support

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

PostgreSQL supports labeling for MLS database services as discussed in the SE-PostgreSQL section.

2.12.4 Common Criteria Certification

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

The Red Hat evaluation process cover the:

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

An interesting point:

- Both Red Hat Linux 5.1 and Microsoft Server 2003 (with XP) have both been certified to EAL4+ , however while the evaluation levels may be the same the Protection Profiles that they were evaluated under were: Microsoft CAPP

only, Red Hat CAPP and LSPP. Therefore always look at the protection profiles as they define what was actually evaluated.

2.13 Types of SELinux Policy

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

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

1. Source code – These can be described as: Example, Reference Policy or Custom
2. The source code descriptions or builds can also be sub-classified as: Monolithic, Base Module or Loadable Module.
3. Policies can also be described by the type of policy functionality they provide such as: `targeted`, `mls`, `mcs`, `standard`, `strict` or `minimum`.
4. Classified using language statements – These can be described as Modular, Optional or Conditional.
5. Binary policy (or kernel policy) – These can be described as Monolithic, Kernel Policy or Binary file.
6. Classification can also be on the 'policy version' used (examples are version 22, 23 and 24).

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

2.13.1 Example Policy

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

2.13.2 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 section.

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 Policy'. For example, the F-17 distribution is based on a specific build of the standard Reference Policy that is then modified and distributed by Red Hat as an RPM.

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

2.13.3 Policy Functionality Based on Name or Type

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

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

This is how the reference policies distributed with F-17 are named, where:

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

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

`mls` – supports server based MLS systems.

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

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

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

`mls` – supports MLS labels as discussed in the Multi-Level Security and Multi-Category Security section.

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

Note that at some stage in the future the Reference Policy may be replaced by the Common Intermediate Language (CIL) service that is under development (see http://userspace.selinuxproject.org/trac/wiki/CilDesign).

2.13.4 Custom Policy

This generally refers to a policy source that is either:

1. A customised version of the Example policy.
2. A customised version of the Reference Policy (i.e. not the standard distribution version).
3. A policy that has been built using policy language statements to build a specific policy such as the basic policy built in the Notebook source tarball.

2.13.5 Monolithic Policy

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

An example monolithic policy is the NSAs original Example Policy. A simple monolithic policy is shown in the Building the Monolithic Policy section and Table 14 shows the order of language statements that can be in a source file.

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

The Reference Policy supports the building of monolithic policies.

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

2.13.6 Loadable Module Policy

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

There are number of 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'.

igure 2.2 shows these components along the top of the diagram. The files contained in the policy store are detailed in the Policy Store Configuration Files section.

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

2.13.6.1 Optional Policy

The loadable module policy infrastructure supports an optional policy statement that allows policy rules to be defined but only enabled in the binary policy once the conditions have been satisfied.

Example loadable modules with `optional` statements are used in the message filter example contained in the Notebook source tarball.

2.13.7 Conditional Policy

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

flag. This is often used to enable or disable features within the policy (i.e. change the policy enforcement rules).

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

```
setsebool -P ext_gateway_audit false
```

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

2.13.8 Binary Policy

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

The binary policy can be built from source files supplied by the Example Policy, the Reference Policy or custom built source files as described in the in the "Sample Policy Source" Notebook.

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.26 (as version 26 is supported by F-16):

```
/etc/selinux/targeted/policy/policy.26
```

2.13.9 Policy Versions

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

The **sestatus**(8) command will show the maximum policy version number supported by the kernel in its output as follows:

```
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
Current mode:                   enforcing
```

```
Mode from config file:      permissive
Policy version:             26
Policy from config file:    modular-test
```

The F-16 kernel policy version is '26' with Table 3 describing the different versions. There is also another version that applies to the modular policy, however the main policy database version is the one that is generally quoted (some SELinux utilities 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 `validatetrans` Statement.
20	-	Reduced the size of the access vector table.
21	6	Added support for the MLS `range_transition` Statement.
22	7	Added policy capabilities that allows various kernel options to be enabled as described in the SELinux Filesystem section.
23	8	Added support for the `permissive` statement. This allows a domain to run in permissive mode while the others are still confined (instead of the all or nothing set by the `SELINUX` entry in the `/etc/selinux/config` 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 "A secure web application platform powered by SELinux" [Ref. 20].
25	11	Add support for file name transition in the `type_transition` rule. Requires kernel 2.6.39 minimum.
26	12/13	Add support for a class parameter in the `role_transition` rule. Add support for the `attribute_role` and `roleattribute` statements. These require kernel 2.6.39 minimum.
-	14	Separate tunables.
27	15	Support setting object defaults for the user, role and range components when computing a new context. Requires kernel 3.5 minimum.

policy db Version	*modular db Version*	*Description*
28	16	Support setting object defaults for the type component when computing a new context. Requires kernel 3.5 minimum.
29	16	Adds an IP address to the SELinux port statement via a SELinux node label. Note that the kernel and userspace versions containing this feature is not yet known.

Table 3: Policy version descriptions

2.14 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 Auditing SELinux Events section. The SELinux utilities such as **audit2allow**(1) and **audit2why**(8) can then be used to determine the cause and possible resolution by generating the appropriate allow rules.

Disabled – The SELinux infrastructure is not enabled, therefore no policy can be loaded.

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

There is another method for running specific domains in permissive mode using the `permissive` statement. This can be used directly in a user written 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/<SELINUXTYPE>/modules/active/modules/permissive_unconfined_t.pp
# and then reload the policy:

semanage permissive -a unconfined_t
```

It is also possible to set permissive mode on a userspace object manager using `libselinux` functions such as **avc_open**(3).

The **sestatus**(8) command will show the current SELinux enforcement mode in its output as follows:

```
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
Current mode:                   permissive
Mode from config file:          enforcing
Policy version:                 26
Policy from config file:        modular-test
```

2.15 Auditing SELinux Events

For SELinux there are two main types of audit event:

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

The audit and event messages are generally stored in one of the following logs (in F-17 anyway):

1. The SELinux kernel boot events are logged in the `/var/log/dmesg` log.
2. The system log `/var/log/messages` contains messages generated by SELinux before the audit daemon has been loaded, although some kernel messages continue to be logged here as well[13].
3. The audit log `/var/log/audit/audit.log` contains events that take place after the audit daemon has been loaded. The AVC audit messages of interest are described in the AVC Audit Events section with others described in the General SELinux Audit Events section. F-17 uses the audit framework **`auditd`**`(8)` as standard.

Notes:

a) It is not mandatory for SELinux-aware applications to audit events or even log them in the audit log. The decision is made by the application designer.

b) The format of audit messages do not need to conform to any format, however where possible applications should use the **`audit_log_user_avc_message`**`(3)` function with a suitably formatted message if using **`auditd`**`(8)`. The type of audit events possible are defined in the `include/libaudit.h` and `include/linux/audit.h` files.

c) Those libselinux library functions that output messages do so to `stderr` by default, however this can be changed by calling **`selinux_set_callback`**`(3)` and specifying an alternative log handler (the `notebook-source-3.0.tar.gz` tarball `libselinux/avc_has_perm_callbacks_example.c` shows a worked example).

2.15.1 AVC Audit Events

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

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

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` may 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: **type=AVC** msg=audit(1243332701.744:**101**): avc: denied { getattr } for pid=2714 comm="ls" path="/usr/lib/locale/locale-archive" dev=dm-0 ino=353593 scontext=system_u:object_r:unlabeled_t:s0 tcontext=system_u:object_r:locale_t:s0 tclass=file **type=SYSCALL** msg=audit(1243332701.744:**101**): arch=40000003 syscall=197 success=yes exit=0 a0=3 a1=553ac0 a2=552ff4 a3=bfc5eab0 items=0 ppid=2671 pid=2714 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts1 ses=1 comm="ls" *exe="/bin/ls*" subj=system_u:object_r:unlabeled_t:s0 key=(null)
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 `auditallow` rule 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` `comm`	If a task, then log the process id (`pid`) and the name of the executable file (`comm`).
`key`	If an IPC event then log the identifier.
`capability`	If a Capability event then log the identifier.

`path`
`name`
`dev`
`ino`
	If a File System event then log the relevant information. Note that the `name` field may not always be present.

`laddr`
`lport`
`faddr`
	If a Socket event then log the Source / Destination addresses and ports for IP4 or IP6 sockets (`AF_INET`).

Keyword	*Description*
fport	
path	If a File Socket event then log the path (AF_UNIX).
saddr	If a Network event then log the Source / Destination addresses and ports with the network interface for IP4 or IP6 networks (AF_INET).
src	
daddr	
dest	
netif	
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 4: AVC Audit Message Description – *The keywords in* **bold** *are in all AVC audit messages, the others depend on the type of event being audited.*

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

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

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

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

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

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

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

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

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

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

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

2.15.2 General SELinux Audit Events

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

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

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

```
type=MAC_POLICY_LOAD msg=audit(1336662937.117:394): policy loaded auid=0 ses=2
type=SYSCALL msg=audit(1336662937.117:394): arch=c000003e syscall=1 success=yes
exit=4345108 a0=4 a1=7f0a0c547000 a2=424d14 a3=7fffe3450f20 items=0 ppid=3845
pid=3848 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts2
ses=2 comm="load_policy" exe="/sbin/load_policy"
subj=unconfined_u:unconfined_r:load_policy_t:s0-s0:c0.c1023 key=(null)

type=USER_MAC_POLICY_LOAD msg=audit(1336662938.535:395): pid=0 uid=0
auid=4294967295 ses=4294967295 subj=system_u:system_r:xserver_t:s0-s0:c0.c1023
msg='avc:  received policyload notice (seqno=2) : exe="/usr/bin/Xorg" sauid=0
hostname=? addr=? terminal=?'
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

These events were generated by the X-Windows Selection Manager demo that is in the Notebook source tarball. Note that the event `type=TRUSTED_APP`, with the actual event embedded as additional text in the message (using **`audit_log_user_avc_message`**`(3)`).

```
type=TRUSTED_APP msg=audit(1336317006.208:327): pid=0 uid=0 auid=0 ses=2
subj=unconfined_u:unconfined_r:selmgr_t:s0-s0:c0.c1023 msg='X-Selection Manager:
The requested paste from scontext 'unconfined_u:unconfined_r:text_test1_t:s0-
s0:c20.c29' to tcontext 'unconfined_u:unconfined_r:text_test1_t:s0-s0:c20.c29'
has been accepted : exe="/usr/local/bin/selmgr" sauid=0 hostname=? addr=?
terminal=pts/0'
```

```
type=TRUSTED_APP msg=audit(1336317010.417:328): pid=0 uid=0 auid=0 ses=2
subj=unconfined_u:unconfined_r:selmgr_t:s0-s0:c0.c1023 msg='X-Selection Manager:
The requested paste from scontext 'unconfined_u:unconfined_r:unconfined_t:s0-
s0:c0.c1023' to tcontext 'unconfined_u:unconfined_r:text_test1_t:s0-s0:c20.c29'
has been denied : exe="/usr/local/bin/selmgr" sauid=0 hostname=? addr=?
terminal=pts/0'
```

```
type=TRUSTED_APP msg=audit(1336662938.523:393): pid=0 uid=0 auid=0 ses=2
subj=unconfined_u:unconfined_r:selmgr_t:s0-s0:c0.c1023 msg='X-Selection Manager:
Received policy reload notice (seqno = 2). Resetting context information :
exe="/usr/local/bin/selmgr" sauid=0 hostname=? addr=? terminal=pts/0'
```

```
type=TRUSTED_APP msg=audit(1336836093.837:407): pid=0 uid=0 auid=0 ses=2
subj=unconfined_u:unconfined_r:selmgr_t:s0-s0:c0.c1023 msg='X-Selection Manager:
Received setenforce  notice. SELinux set to ENFORCING mode :
exe="/usr/local/bin/selmgr" sauid=0 hostname=? addr=? terminal=pts/2'
```

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

2.16.1 Polyinstantiated Objects

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

2.16.2 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 a good reference). Note that PAM and Namespace services are SELinux-aware.

The default installation of F-17 does not enable polyinstantiated directories, therefore this section will show the configuration required to enable the feature and some examples.

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

1. A `pam_namespace` module entry added to the appropriate `/etc/pam.d/` login configuration file (e.g. `login`, `sshd`, `gdm` etc.). F-17 already has these entries configured, with an example `/etc/pam.d/gdm` file being:

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

2. Entries added to the `/etc/security/namespace.conf` file that defines the directories to be polyinstantiated by PAM (and other services that may need to use the namespace service). The entries are explained in the `namespace.conf` Configuration File section, with the default entries in F-17 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-17 configuration also includes an `/etc/security/namespace.init` script that is used to initialise the namespace every time a new directory instance is set up. This script receives four parameters: the polyinstantiated directory path, the instance directory path, a flag to indicate if a new instance, and the user name. If a new instance is being set up, the directory permissions are set and the **`restorecon`**`(8)` command is run to set the correct file contexts.

2.16.2.1 `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.

2.16.2.2 Example Configurations

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

Example 1 - method=user:

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

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

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

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

# /tmp
/tmp/tmp-inst/rch

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

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

Example 2 - method=context:

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

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

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

```
# The directories will contain the security context and
```

```
# user name as a part of the polyinstantiated directory
# name as follows:

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

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

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

2.16.3 Polyinstantiation support in X-Windows

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

2.16.4 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 there is a selection manager example in the Notebook source tarball.

2.17 PAM Login Process

Applications used to provide login services (such as `gdm` and `ssh`) in F-17 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-17 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-17 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.
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-17 uses the /etc/pam.d directory configuration (in this case gdm for the Gnome login service).

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

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

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

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

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

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

2.18 Linux Security Module and SELinux

This section gives a high level overview of the LSM and SELinux internal kernel structure and workings as enabled in kernel 3.3.2. 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 (and also using the SELinux kernel source code). The major areas covered are:

1. How the LSM and SELinux modules work together.
2. The major SELinux internal services.
3. The `fork` and `exec` system calls are followed through as an example to tie in with the transition process covered in the Domain Transition section.
4. The SELinux filesystem `/sys/fs/selinux`.
5. The `/proc` filesystem area most applicable to SELinux.
6. The boot sequences that are relevant to SELinux.

2.18.1 The LSM Module

The LSM is the Linux security framework that allows 3rd party access control mechanisms to be linked into the GNU / Linux kernel. Currently there are five 3rd party services that utilise the LSM:

1. SELinux - the subject of this Notebook.
2. AppArmor is a MAC service based on pathnames and does not require labelling or relabelling of filesystems. See http://wiki.apparmor.net for details.
3. Simplified Mandatory Access Control Kernel (SMACK). See http://www.schaufler-ca.com/ for details.
4. The Trusted Computing Group runtime Integrity Measurement Architecture (IMA). This maintains lists of hash values for sensitive system files at runtime. See http://domino.research.ibm.com/comm/research_people.nsf/pages/sailer.ima.html for details.
5. Tomoyo that is a name based MAC and details can be found at http://sourceforge.jp/projects/tomoyo/docs.
6. Yama extends the DAC support (currently) for ptrace. See `Documentation/security/Yama.txt` for further details.

The basic idea behind LSM is to:

- Insert security function hooks and security data structures in the various kernel services to allow access control to be applied over and above that already implemented via DAC. The type of service that have hooks inserted are shown in Table 5 with an example task and program execution shown in the Fork Walk-thorough and Process Transition Walk-thorough sections.
- Allow registration and initialisation services for the 3rd party security modules.
- Allow process security attributes to be available to userspace services by extending the `/proc` filesystem with a security namespace as shown in Table 6. These are located at:

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

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

  Where `<pid>` is the process id, `<tid>` is the thread id and `<attr>` is the entry described in Table 6.
- Support filesystems that use extended attributes (SELinux uses `security.selinux` as explained in the Labeling Extended Attribute Filesystems section).
- Consolidate the Linux capabilities into an optional module.

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

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

Table 5: LSM Hooks - *These are the kernel services that LSM has inserted security hooks and structures to allow access control to be managed by 3rd party modules (see `./linux-3.3/include/linux/security.h`).*

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

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

The major kernel source files (relative to `./linux-3.3/security`) that form the LSM are shown in Table 7. However there is one major header file (`include/linux/security.h`) that describes all the LSM security hooks and structures.

Name	Function
`capability.c`	Some capability functions were in various kernel modules have been consolidated into these source files. 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 3rd party security module to initialise a security filesystem. In the case of SELinux this would be `/sys/fs/selinux` that is defined in the `selinux/selinuxfs.c` source file.
`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 3rd party modules.
`lsm_audit.c`	Contains common LSM audit functions.
`min_addr.c`	Minimum VM address protection from userspace for DAC and LSM.

Table 7: The core LSM source modules.

2.18.2 The SELinux Module

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

The major kernel SELinux source files (relative to `./linux-3.3/security/selinux`) that form the SELinux security module are shown inTable 8. The diagrams shown in Figure 2.2 and Figure 2.12 can be used to see how some of these kernel source modules fit together.

Name	Function
`avc.c`	Access Vector Cache functions and structures. The function calls are for the kernel services, however they have been ported to form the `libselinux` userspace library.
`exports.c`	Exported SELinux services for SECMARK (as there is SELinux specific code in the netfilter source tree).
`hooks.c`	Contains all the SELinux functions that are called by the kernel resources via the `security_ops` function table (they form the kernel resource object managers). There are also support functions for managing process exec's, managing SID allocation and removal, interfacing into the AVC and Security Server.
`netif.c`	These manage the mapping between labels and SIDs for the `net*` language statements when they are declared in the active policy.
`netnode.c`	
`netport.c`	
`netlabel.c`	The interface between NetLabel services and SELinux.
`netlink.c`	Manages the notification of policy updates to resources including userspace applications via `libselinux`.
`nlmsgtab.c`	
`selinuxfs.c`	The `selinuxfs` pseudo filesystem (`/sys/fs/selinux`) that imports/exports security policy information to/from userspace services. The services exported are shown in the SELinux Filesystem section.
`xfrm.c`	Contains the IPSec XFRM (transform) hooks for SELinux.
`include/classmap.h`	`classmap.h` contains all the kernel security classes and permissions. `initial_sid_to_string.h` contains the initial SID contexts. These are used to build the `flask.h` and `av_permissions.h` kernel configuration files when the kernel is being built (using the `genheaders` script defined in the `selinux/Makefile`). These files are built this way now to support the new security class mapping structure to remove the need for fixed class to SID mapping.
`include/initial_sid_to_string.h`	
`ss/avtab.c`	AVC table functions for inserting / deleting entries.
`ss/conditional.c`	Support boolean statement functions and implements a conditional AV table to hold entries.
`ss/ebitmap.c`	Bitmaps to represent sets of values, such as types, roles, categories, and classes.
`ss/hashtab.c`	Hash table.
`ss/mls.c`	Functions to support MLS.

Name	Function
ss/policydb.c	Defines the structure of the policy database. See the "SELinux Policy Module Primer" [Ref. 4] article for details on the structure.
ss/services.c	This contains the supporting services for kernel hooks defined in 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/status.c	Interface for selinuxfs/status. Used by the libselinux selinux_status_*(3) functions.
ss/symtab.c	Maintains associations between symbol strings and their values.

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

2.18.2.1 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:

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

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

a) Initialise the AV structure so that it is clear.

b) Check the object class and permissions are correct. It also checks the status of the `allow_unknown` flag (see the SELinux Filesystem, `/etc/selinux/semanage.conf` file and Reference Policy Build Options - `build.conf` - `UNK_PERMS` sections).

c) Checks if there are any type enforcement rules (`ALLOW`, `AUDIT_ALLOW`, `AUDIT_DENY`).

d) Check whether any conditional statements are involved via the `cond_compute_av()` function in `conditional.c`.

e) Remove permissions that are defined in any constraint via the `constraint_expr_eval()` function call (in `services.c`). This function will also check any MLS constraints.

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

g) Finally check whether there are any constraints applied via the `typebounds` rule.

8. Once the result has been computed it is returned to the `kernel/fork.c` system call via the initial `selinux_task_create()` function. In this case the fork call is allowed.
9. The End.

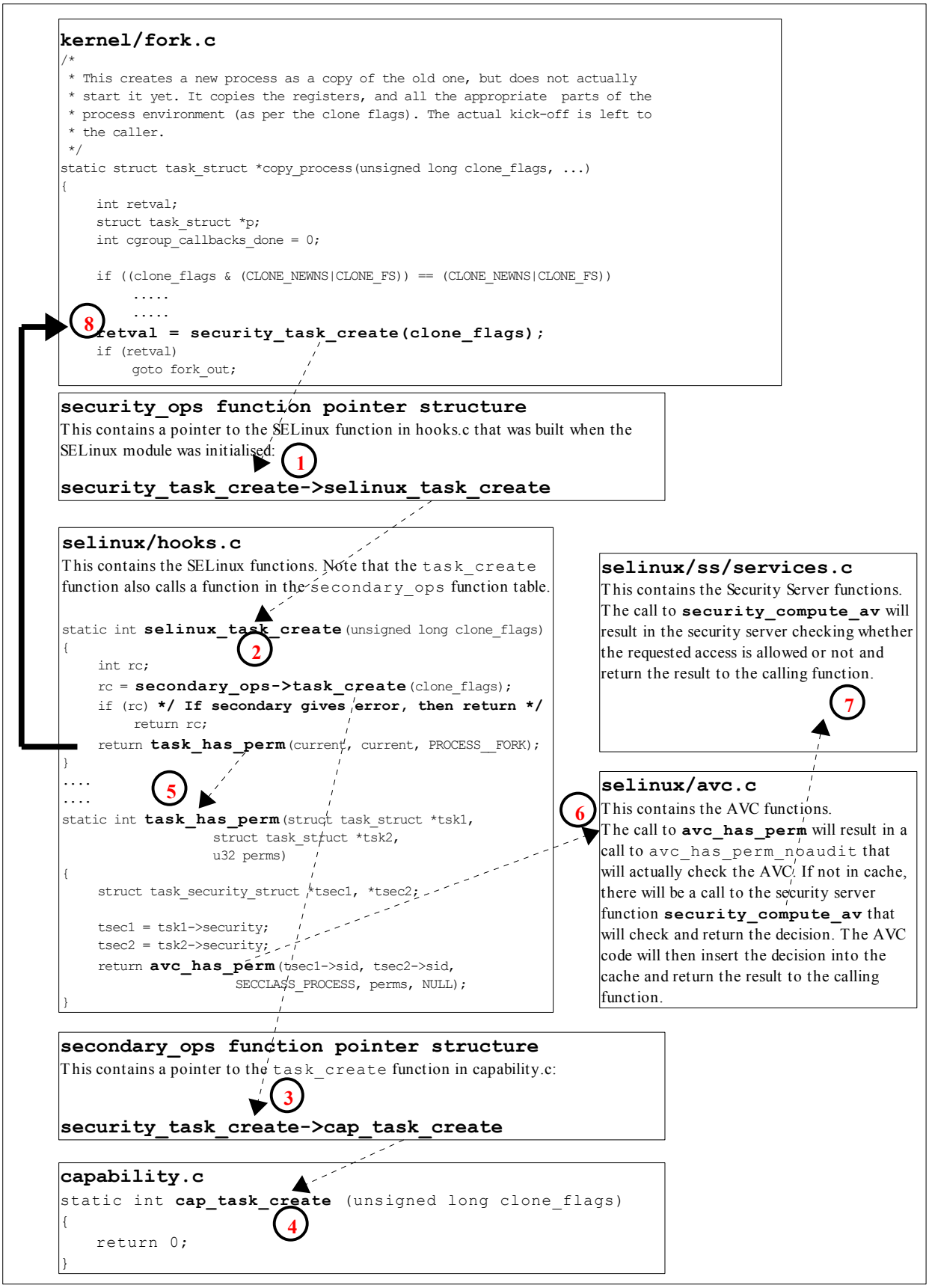


Figure 2.10: Hooks for the fork system call - *This describes the steps required to check access permissions for Object Class 'process' and permission 'fork'.*

2.18.2.2 Process Transition Walk-thorough

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

The process starts with the Linux operating system issuing a `do_execve()`[14] 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 9 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->` `selinux_bprm_alloc_security`	Allocates memory for the `bprm` structure.
`security_bprm_free->` `selinux_bprm_free_security`	Frees memory from the `bprm` structure.
`security_bprm_apply_creds->` `selinux_bprm_apply_creds`	Sets task lock and new security attributes for a transformed 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`.
`security_bprm_post_apply_creds->` `selinux_bprm_post_apply_creds`	Supports the `security_bprm_apply_creds` function for areas that must not be locked.
`security_bprm_secureexec->` `selinux_bprm_secureexec`	Called after the `selinux_bprm_post_apply_creds` function to check `AT_SECURE` flag for glibc secure mode support.
`security_bprm_set->` `selinux_bprm_set_security`	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->` `selinux_bprm_check_security`	This hook is not used by SELinux.

Table 9: The LSM / SELinux Program Loading Hooks

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

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

[14] 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.

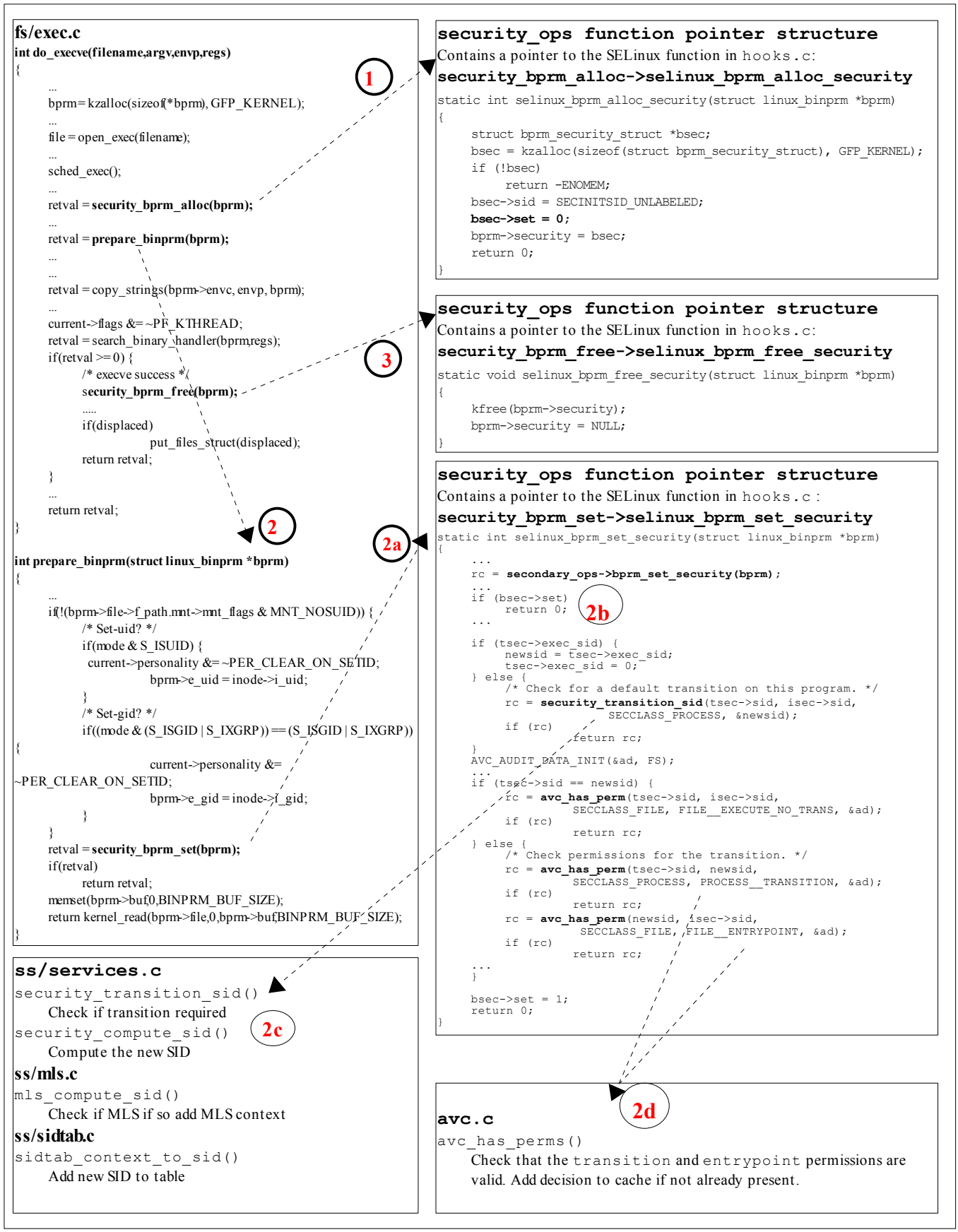


Figure 2.11: Process Transition - *This shows the major steps required to check if a transition is allowed from the* `unconfined_t` *domain to the* `ext_gateway_t` *domain.*

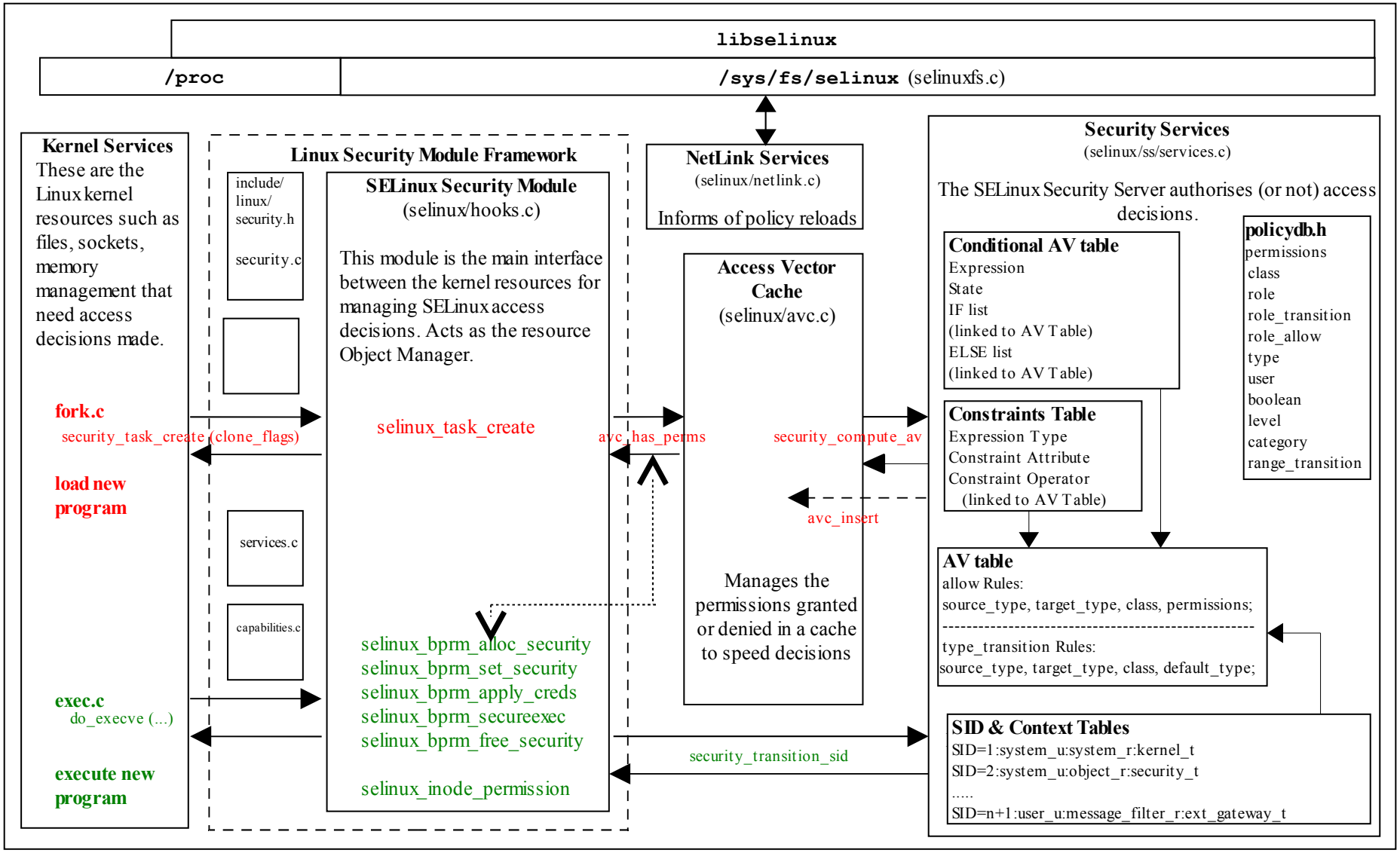


Figure 2.12: The Main LSM / SELinux Modules – *The fork and exec functions link to Figure 2.7 where the transition process is described.*

2.18.2.3 SELinux Filesystem

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

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

***selinuxfs* Directory and File Names**	***Permissions***	***Comments***
create	-rw-rw-rw-	Compute create labeling decision interface that is used by the **security_compute_create**(3) and **avc_compute_create**(3) functions. The kernel security server (see services.c) converts the contexts to SIDs and then calls the security_transition_sid_user function to compute the new SID that is then converted to a context string. Requires security {compute_create} permission.
deny_unknown	-r--r--r--	These two files export deny_unknown (read by **security_deny_unknown**(3) function) and reject_unknown status to user space. These are taken from the handle-unknown parameter set[15] in the /etc/selinux/semanage.conf file when policy is being built and are set as follows: deny:reject 0:0 = Allow unknown object class / permissions. This will set the returned AV with all 1's. 1:0 = Deny unknown object class / permissions (the default). This will set the returned AV with all 0's. 1:1 = Reject loading the policy if it does not contain all the object classes / permissions.
reject_unknown	-r--r--r--	
disable	--w-------	Disable SELinux until next reboot.
enforce	-rw-r--r--	Get or set enforcing status. Requires security {setenforce} permission.
load	-rw-------	Load policy interface. Requires security {load_policy} permission.
member	-rw-rw-rw-	Compute polyinstantiation membership decision interface that is used by the **security_compute_member**(3) and **avc_compute_member**(3) functions. The kernel security server (see services.c) converts the contexts to SIDs and then calls the security_member_sid function to compute the new SID that is then converted to a context string. Requires security {compute_member} permission.
mls	-r--r--r--	Returns 1 if MLS policy is enabled or 0 if not.

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

***selinuxfs* Directory and File Names**	***Permissions***	***Comments***
null	crw-rw-rw-	The SELinux equivalent of /dev/null for file descriptors that have been redirected by SELinux.
policyvers	-r--r--r--	Returns supported policy version for kernel. Read by **security_policyvers**(3) function.
relabel	-rw-rw-rw-	Compute relabeling decision interface that is used by the **security_compute_relabel**(3) function. The kernel security server (see services.c) converts the contexts to SIDs and then calls the security_change_sid function to compute the new SID that is then converted to a context string. Requires security {compute_relabel} permission.
status	-r--r--r--	This can be used to obtain enforcing mode and policy load changes with much less over-head than using the libselinux netlink / call backs. This was added for Object Managers that have high volumes of AVC requests so they can quickly check whether to invalidate their cache or not. The status structure indicates the following: version - Version number of the status structure. This will increase as other entries are added. sequence - This is incremented for each event with an even number meaning that the events are stable. An odd number indicates that one of the events is changing and therefore the userspace application should wait before reading the status of any event. enforcing - 0 = Permissive mode, 1 = enforcing mode. policyload - This contains the policy load sequence number and should be read and stored, then compared to detect a policy reload. deny_unknown - 0 = Allow and 1 = Deny unknown object classes / permissions. This is the same as the deny_unknown entry above.
user	-rw-rw-rw-	Compute reachable user contexts interface that is used by the **security_compute_user**(3) function. The kernel security server (see services.c) converts the contexts to SIDs and then calls the security_get_user_sids function to compute the user SIDs that are then converted to context strings. Requires security {compute_user} permission.

***selinuxfs* Directory and File Names**	***Permissions***	***Comments***
/sys/fs/selinux/avc	**Directory**	This directory contains information regarding the kernel AVC that can be displayed by the `avcstat` command.
cache_stats	-r--r--r--	Shows the kernel AVC lookups, hits, misses etc.
cache_threshold	-rw-r--r--	The default value is 512, however caching can be turned off (but performance suffers) by: `echo 0 > /selinux/avc/cache_threshold` Requires `security {setsecparam}` permission.
hash_stats	-r--r--r--	Shows the number of kernel AVC entries, longest chain etc.
/sys/fs/selinux/booleans	**Directory**	This directory contains one file for each boolean defined in the active policy.
secmark_audit	-rw-r--r--	Each file contains the current and pending status of the boolean (0 = false or 1 = true). The **`getsebool`**`(8)`, **`setsebool`**`(8)` and **`sestatus -b`** commands use this interface via the `libselinux` library functions.
/sys/fs/selinux/initial_contexts	**Directory**	This directory contains one file for each initial SID defined in the active policy.
any_socket devnull	-r--r--r--	Each file contains the initial context of the initial SID as defined in the active policy (e.g. `any_socket` was assigned `system_u:object_r:unconfined_t`).
/sys/fs/selinux/policy_capabilities	**Directory**	This directory contains the policy capabilities that have been configured by default in the kernel via the policycap Statement in the active policy. These are generally new features that can be enabled for testing by using the `policycap` Statement in policy.

network_peer_controls	-r--r--r--	For the F-17 Reference Policy this file contains '1' (true) which means that the following `network_peer_controls` are enabled by default:
		`node: sendto recvfrom`
		`netif: ingress egress`
		`peer: recv`

open_perms	-r--r--r--	For the F-17 Reference Policy this file contains '1' (true) which means that `open` permissions are enabled by default on the following objects: `dir`, `file`, `fifo_file`, `chr_file`, `blk_file`.
ptrace_child	-r--r--r--	This will be enabled kernel 3.4 to allow finer control of ptrace. Requires policy support and the `security` class permission `ptrace_child`.

***selinuxfs* Directory and File Names**	***Permissions***	***Comments***
/sys/fs/selinux/class	**Directory**	This directory contains a list of classes and their permissions as defined within the policy.
/sys/fs/selinux/class/appletalk_socket	**Directory**	Each class has its own directory where each one is named using the appropriate class statement from the policy (i.e. `class appletalk_socket`). Each directory contains the following:
index	-r--r--r--	This file contains the allocated `class` number (e.g. `appletalk_socket` is '56' in `flask.h`).
/sys/fs/selinux/class/appletalk_socket/perms	**Directory**	This directory contains one file for each permission defined in the policy.
accept append bind	-r--r--r--	Each file is named by the permission assigned in the policy and contains a number that represents its position in the list (e.g. `accept` is the 14th permission listed in `av_permission.h` for `appletalk_socket` and therefore contains '14'.

Table 10: `/selinux` File and Directory Information

Notes:

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

2.19 libselinux Library

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

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

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

The general category of functions available in `libselinux` are shown in Table 11, with Appendix B giving a complete list of functions and source code examples available in the Notebook tarball.

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

Socket Creation Labeling	Get and set socket creation contexts.
User Session Management	Retrieve default contexts for user sessions.

Table 11: `libselinux` function types

Other SELinux userspace libraries are:

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

`libsemanage` - To manage the policy infrastructure.

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

http://userspace.selinuxproject.org/trac

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

Writing kernel based object managers is a more specialised subject and is not covered within this man page.

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

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

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

2.20 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).

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

```
yum install netlabel_tools
```

```
yum install ipsec-tools
```

2.20.1 `compat_net` Controls

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

Figure 2.13 shows how these network statements are used and the type of allow rules that would be required.

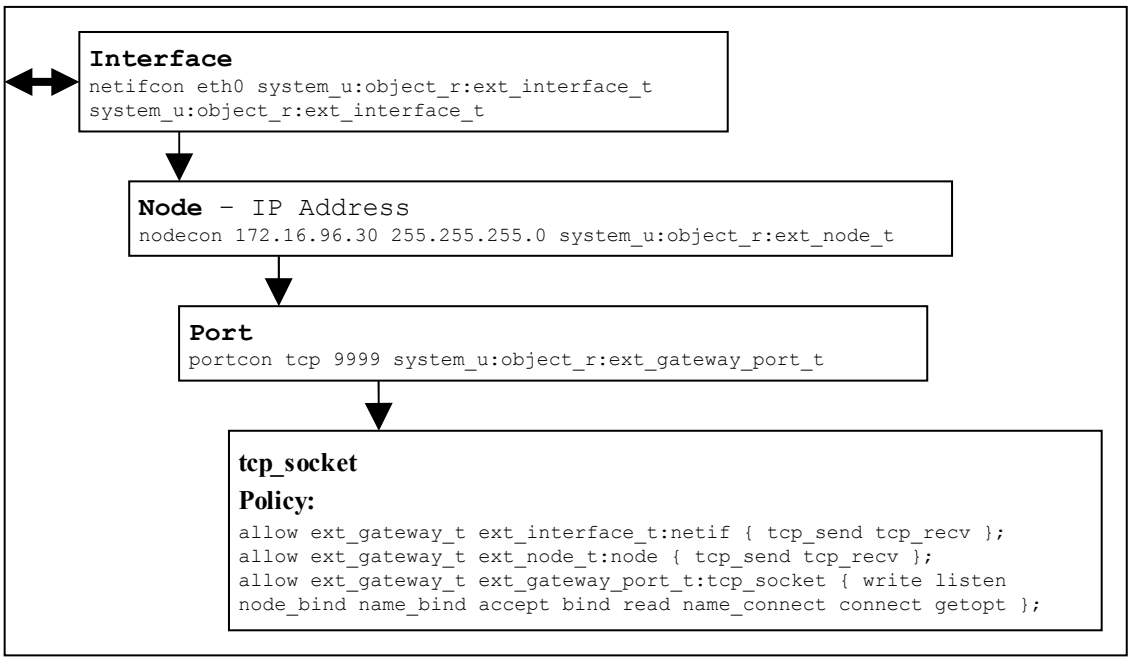


Figure 2.13: `compat_net` Controls – *Showing the policy statements and rules required to allow communications.*

The current SELinux port definition does not include an IP address which makes it difficult to restrict `connect()` and `bind()` operations using SELinux. Policy version 29 solves this problem by adding an IP address to the SELinux port definition via a SELinux node label (however, note that the kernel and userspace versions containing this feature are not yet known).

2.20.2 SECMARK

SECMARK makes use of the standard 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 available[16]. Figure 2.14 shows the basic structure with the process working as follows:

- A table called the 'security table' is used to define the parameters that identify and 'mark' packets that can then be tracked as the packet travels through the networking sub-system. These 'marks' are called SECMARK and CONNSECMARK.
- A SECMARK is placed against a packet if it matches an entry in the security table. 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 session[17] with the appropriate label that can then be used to enforce policy.

[16] There is a very good tutorial at http://www.frozentux.net/documents/iptables-tutorial/ [Ref.7], however it does not cover the security table that was introduced by: http://lwn.net/Articles/267140/. It is still possible to use the 'mangle table' to hold security labels as described in [Ref. 7].

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

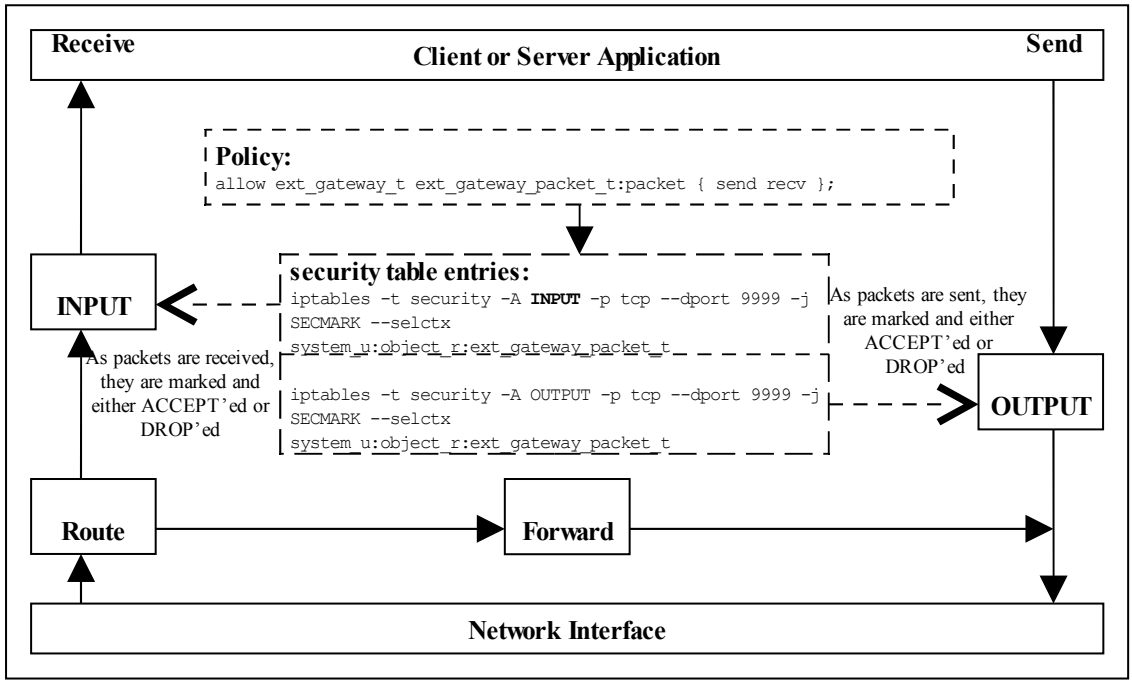


Figure 2.14: SECMARK Processing – *Received packets are processed by the INPUT chain where labels are added to the appropriate packets that will either be accepted or dropped by the SECMARK process. Packets being sent are treated the same way.*

An example `iptables`[18] 'security table' entry is as follows:

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

#-------------- INPUT IP Stream --------------------#
# This INPUT rule sets all packets to default_secmark_packet_t
iptables -t security -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 security -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 Notebook source tarball. There are also articles "Transitioning to Secmark" [Ref. 9] and "New secmark-based network controls for SELinux" [Ref. 8] that explain the transition and services.

2.20.3 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 their usage, the **`netlabelctl`**`(8)` man page is also a useful reference.

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

The example message filter has an optional module that makes use of fallback labels and can be found in the Notebook source tarball, where there are also examples shown for SE-PostgreSQL.

The network peer controls have been extended to support an additional object class of 'peer' that is enabled by default in the F-17 policy as the `network_peer_controls` in `/sys/fs/selinux/policy_capabilities` is set to '1'. Figure 2.15 shows the differences between the policy capability `network_peer_controls` being set to 0 and 1.

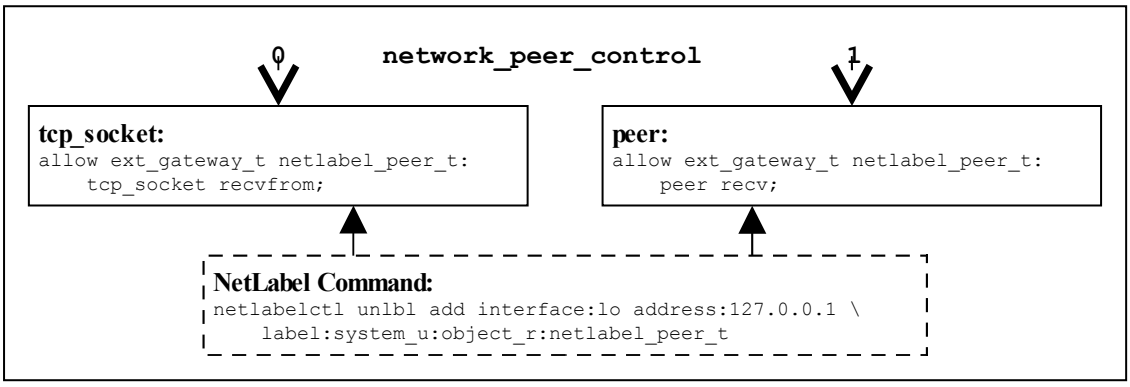


Figure 2.15: Fallback Labeling – *Showing the differences between the policy capability `network_peer_controls` set to 0 and 1.*

2.20.4 NetLabel - CIPSO

To allow security levels to be passed over a network between MLS systems[19], the CIPSO protocol is used that is defined in the CIPSO Internet Draft document (this is an obsolete document, however the protocol is still in use). The protocol defines how security levels are encoded in the IP packet header.

The protocol is implemented by the NetLabel service (see the **`netlabelctl`**`(8)` man page) 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 2.16).

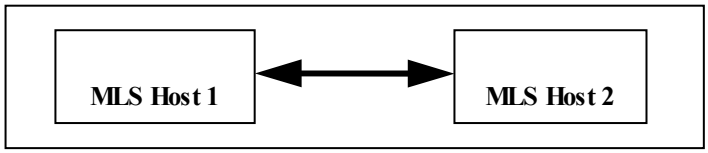


Figure 2.16: MLS Systems on the same network

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 2.17).

[19] 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).

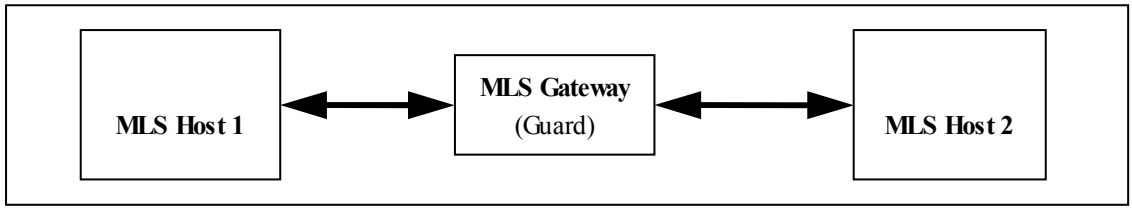


Figure 2.17: MLS Systems on different networks communicating via a gateway

4. Support for full SELinux labels over loopback as discussed in the article at http://paulmoore.livejournal.com/7234.html.

2.20.5 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 2.18 shows the basic components that form the IPSec service where it is generally used to set up either an encrypted tunnel between two machines[20] 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).

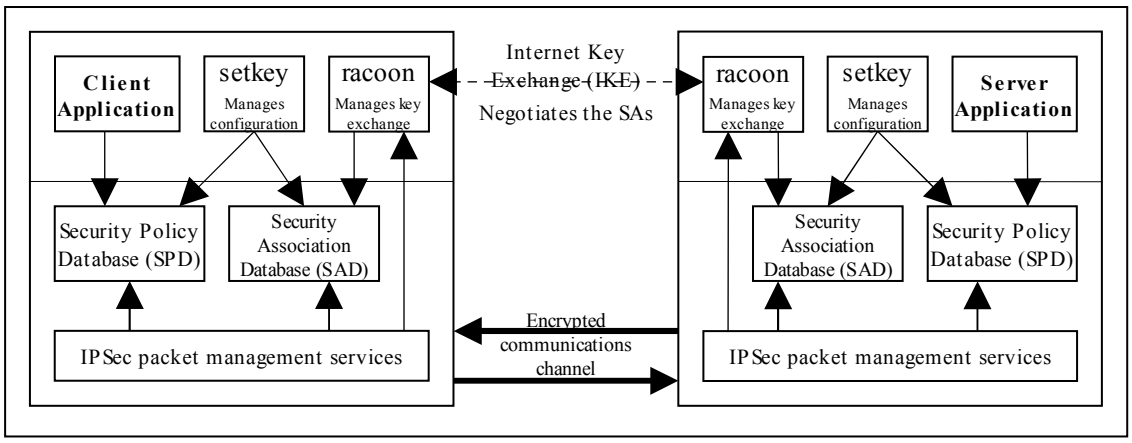


Figure 2.18: IPSec communications – *The SPD contains information regarding the security contexts to be used. These are exchanged between the two systems as part of the channel set-up.*

Basically what happens is as follows[21]:

1. The security policy database (SPD) defines the security communications characteristics to be used between the two systems. This is populated using the **`setkey`**`(8)` utility with an example shown in the Configuration Example section.
2. The SAs have their configuration parameters such as protocols used for securing packets, encryption algorithms and how long the keys are valid held in the Security Association database (SAD). For Labeled IPSec the security context (or labels) is also defined within the SAD. SAs can be negotiated

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

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

between the two systems using either **racoon**(8)[22] that will automatically populate the SAD or manually by the setkey utility (see the example below).

3. Once the SAs have been negotiated and agreed, the link should be active.

A point to note is that SAs are one way only, therefore 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 protocol[23] (e.g. ESP (encapsulated security payload) and AH (authentication header)).

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

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).

There are worked examples of Labeled IPSec sessions showing manual and racoon[24] configuration in the Notebook source tarball.

There is a further example in the "Secure Networking with SELinux" [Ref. 13] article.

2.20.5.1 Configuration Example

```
# 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 "unconfined_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
```

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

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

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

```
#    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 that populates the SPD and SAD[25] the following command would be used:

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

2.21 SELinux Virtual Machine Support

SELinux support is available in the KVM/QEMU and Xen virtual machine (VM) technologies[26] that are discussed in the sections that follow, however the package documentation should be read for how these products actually work and how they are configured.

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

The sections that follow give an introduction to KVM/QEMU, then `libvirt` support with some examples using the Virtual Machine Manager (the sections assume two VM images have been generated: VM1 & VM2[27]) to configure VMs, then an overview of the Xen implementation follows.

The examples shown were tested using F-16 with the following major packages:

```
libvirt-0.9.6-4.fc16.x86_64
qemu-0.15.1-3.fc16.x86_64
qemu-system-x86-0.15.1-3.fc16.x86_64
virt-manager-0.9.0-7.fc16.noarch
```

[25] 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.

[26] 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.

[27] These can be generated using the VVM by selecting the 'Create a new virtual machine' menu item. A simple Linux kernel was used to generate these and is available at: http://wiki.qemu.org/Testing (the `linux-0.2.img.bz2` disk image). This image was renamed to reflect each test, for example '`Dynamic_VM1.img`'.

```
selinux-policy-targeted-3.10.0-84.fc16.noarch
```

To ensure all dependencies are installed run:

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

2.21.1 KVM / QEMU Support

KVM is a kernel loadable module that uses the Linux kernel as a hypervisor and makes use of a modified QEMU emulator to support the hardware I/O emulation. The "Kernel-based Virtual Machine" [Ref. 21] document gives a good overview of how KVM and QEMU are implemented. It also provides an introduction to virtualisation in general. Note that KVM requires virtulisation support in the CPU (Intel-VT or AMD-V extensions).

The SELinux support for VMs is implemented by the `libvirt` sub-system that is used to manage the VM images using a Virtual Machine Manager, and as KVM is based on Linux it has SELinux support by default. There are also Reference Policy modules to support the overall infrastructure (KVM support is in various kernel and system modules with a `virt` module supporting the `libvirt` services). Figure 2.19 shows a high level overview with two VMs running in their own domains. The `libvirt` Support section shows how to configure these and their VM image files.

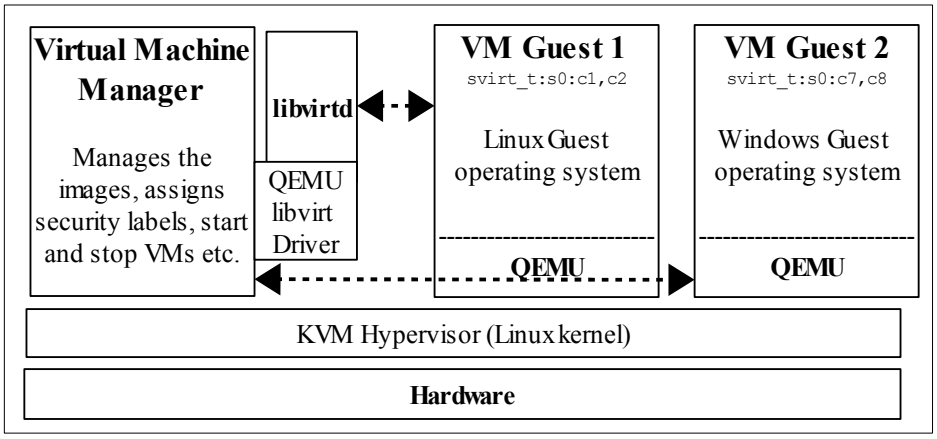


Figure 2.19: KVM Environment - *KVM provides the hypervisor while QEMU provides the hardware emulation services for the guest operating systems. Note that KVM requires CPU virtualisation support.*

2.21.2 `libvirt` Support

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

The VM supplier can implement any security mechanism they require using a product specific libvirt driver that will load and manage the images. The SELinux implementation supports four methods of labeling VM images, processes and their

resources with support from the Reference Policy `modules/services/virt.*` loadable module[28]. To support this labeling, `libvirt` requires an MCS or MLS enabled policy as the `level` entry of the security context is used (`user:role:type:level`).

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

2.21.3 VM Image Labeling

2.21.3.1 Dynamic Labeling

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

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

b) An initial context for the image file label is obtained from the `/etc/selinux/<SELINUXTYPE>/contexts/virtual_image_context` file. The default is `system_u:system_r:svirt_image_t:s0` that allows read/write of image files.

c) When the image is used to start the VM, a random MCS `level` is generated and added to the process context and the image file context. The process and image files are then transitioned to the context by the `libselinux` API calls `setfilecon` and `setexeccon` respectively (see `security_selinux.c` in the `libvirt` source). The following example shows two running VM sessions each having different labels:

VM Name	Object	Dynamically assigned security context
`Dynamic_VM1`	Process	`system_u:system_r:svirt_t:s0:c585,c813`
	File	`system_u:system_r:svirt_image_t:s0:c585,c813`
`Dynamic_VM2`	Process	`system_u:system_r:svirt_t:s0:c535,c601`
	File	`system_u:system_r:svirt_image_t:s0:c535,c601`

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

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

ps -eZ | grep qemu
system_u:system_r:svirt_t:s0:c585,c813  8707 ? 00:00:44 qemu-system-x86
system_u:system_r:svirt_t:s0:cc535,c601 8796 ? 00:00:37 qemu-system-x86
```

[28] 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.

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

2.21.3.2 Static Labeling

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

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, 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-17 `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 `Static` check box. The required security context can then be set; for this example `svirt_t` has been chosen as it is a valid context (however it will not run as explained in the text):

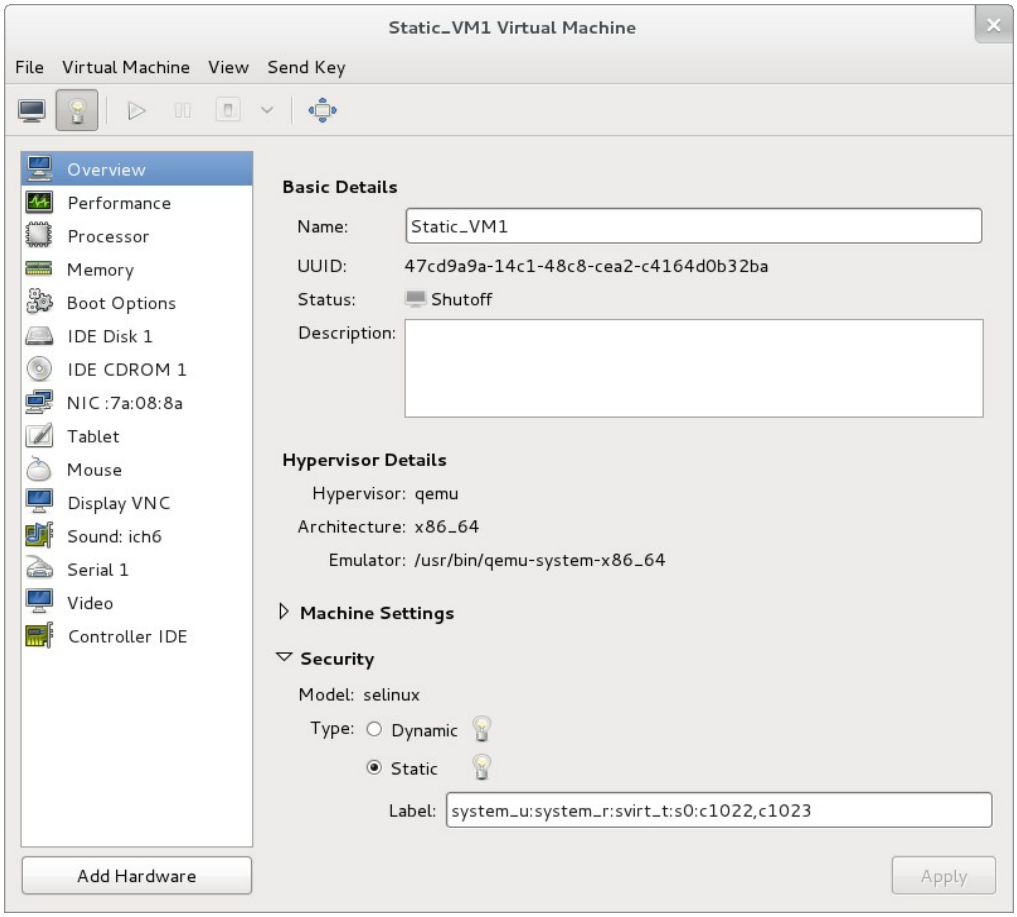


Figure 2.20: Static Configuration

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

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

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

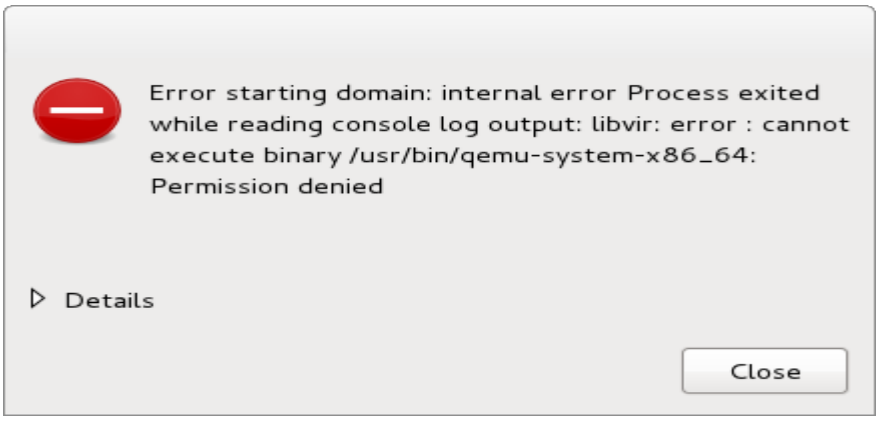


Figure 2.21: Image Start Error

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
```

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

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

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

The following example shows two static VMs (one is configured for `unconfined_t` that is allowed to run under the targeted policy):

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

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

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

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

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

2.21.3.3 Share Image

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

The Virtual Machine Manager can be used to set the image as shareable by checking the `Shareable` box as shown in Figure 2.22.

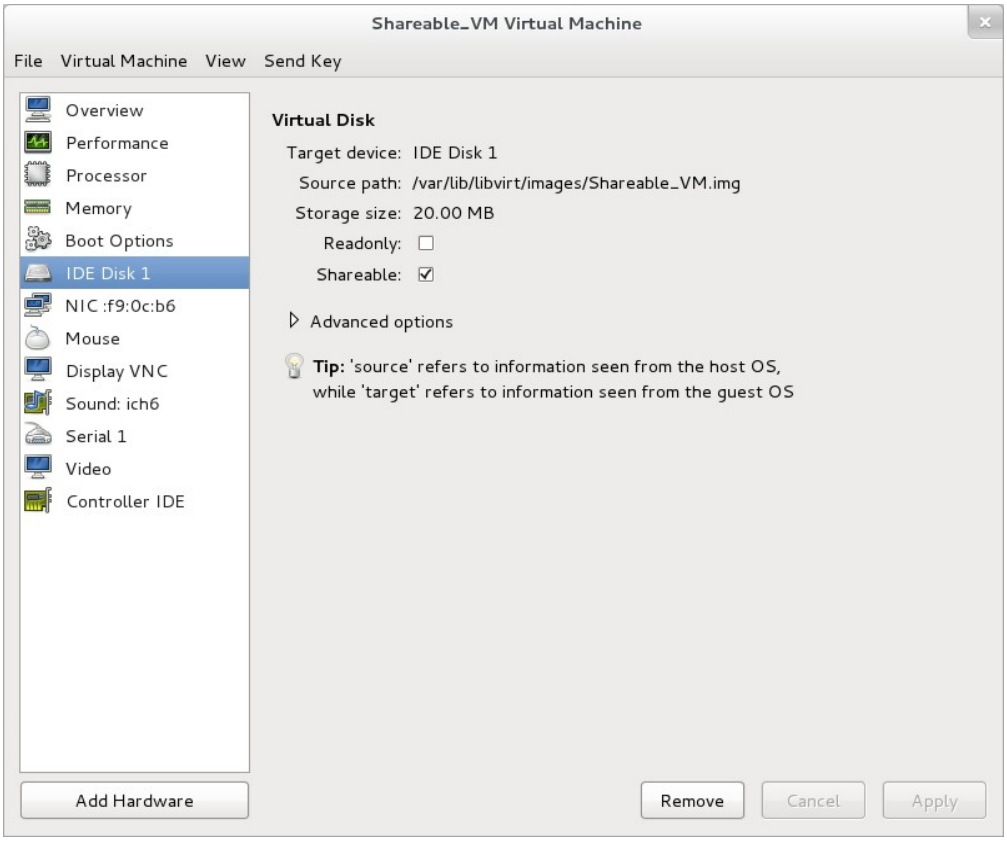


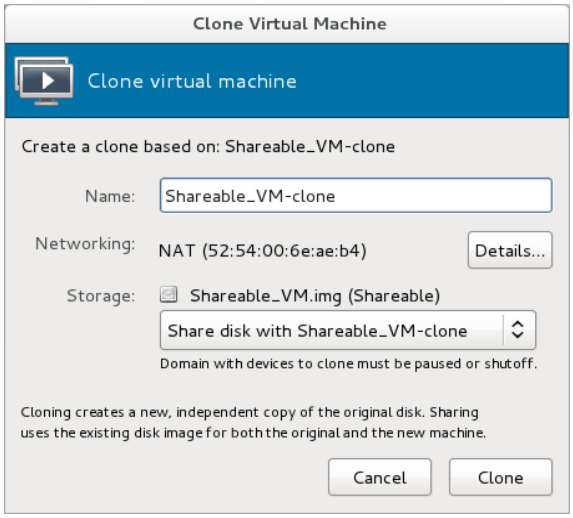
Figure 2.22: Setting the Virtual Disk as Shareable

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

```
# /etc/libvirt/qemu/Shareable_VM.xml:

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

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



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

```
# /etc/libvirt/qemu/Shareable_VM-clone.xml:

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

With the targeted policy on F-16 the shareable option gave a error when the VMs were run as follows:

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

The audit log contained the following AVC message:

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

To overcome this error, the following module was created and installed by:

```
cat /var/log/audit/audit.log | audit2allow -M qemu_execmem >
qemu_execmem.te

# Once generated, the module needs to be activated by:
semodule -i  qemu_execmem.pp
```

For reference, the module generated by `audit2allow` is as follows:

```
module qemu_execmem 1.0;

require {
  type svirt_t;
  class process execmem;
```

```
}
allow svirt_t self:process execmem;
```

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

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

b) An initial context for the image file label is obtained from the `/etc/selinux/<SELINUXTYPE>/contexts/virtual_image_context` file. The default is `system_u:system_r:svirt_image_t:s0` that allows read/write of image files.

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

VM Name	Object	Security context
`Shareable_VM`	Process	`system_u:system_r:svirt_t:s0:c231,c245`
`Shareable_VM -clone`	Process	`system_u:system_r:svirt_t:s0:c695,c894`
	File	`system_u:system_r:svirt_image_t:s0`

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

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

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

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

2.21.3.4 Readonly Image

Changes to `qemu` means that the `readonly` option on IDE drives has been dropped (see http://lists.gnu.org/archive/html/qemu-devel/2011-08/msg00799.html). The consequences of this is that while the Virtual Machine Manager will allow it to be set, when running the image a message is generated stating "Can't use a read-only device". A bug report has been generated asking that `libvirt` be changed to allow the security service (i.e. SELinux) to still manage the read-only option and not pass the `readonly=on` flag, if set to `qemu` (see https://bugzilla.redhat.com/show_bug.cgi?id=732461).

For reference, the `readonly` configuration sequence is similar to the `shared` option shown above with a dynamically allocated level generated for each VM process instance. The major differences are that the disk image will be read only by virtue of the policy 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`).

This section will be updated should `libvirt` be changed.

2.21.4 Xen Support

This is not supported by SELinux in the usual way as it is built into the actual Xen software as a 'Flask/TE' extension[29] 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 2.23.

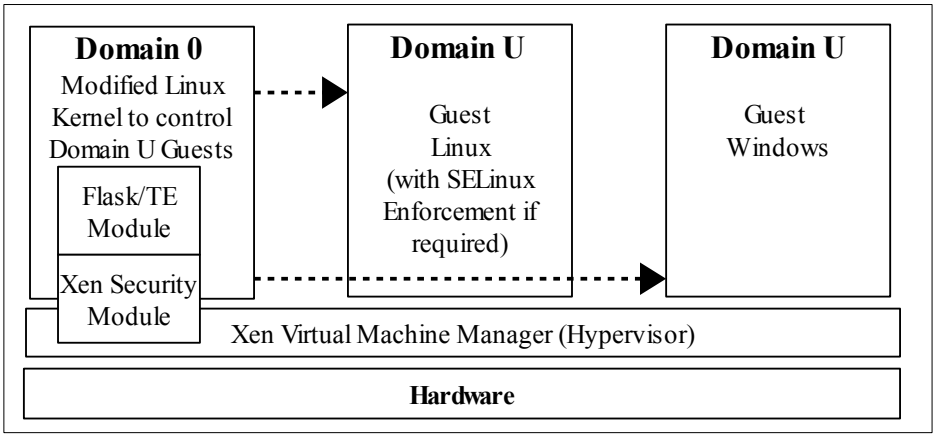


Figure 2.23: Xen Hypervisor - *Using XSM and Flask/TE to enforce policy on the physical I/O resources.*

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

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

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

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

2.22 X-Windows SELinux Support

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

This Notebook will only give a high level description of the infrastructure based on Figure 2.24, 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 has been implemented, although it does not cover areas such as polyinstantiation.

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

The XSELinux object manager source can be found in the `xorg-x11-server-1.12.2-2.fc17.src.rpm`. in the `Xext` directory. The Reference Policy modules have also been updated to enforce policy using the XSELinux object manager.

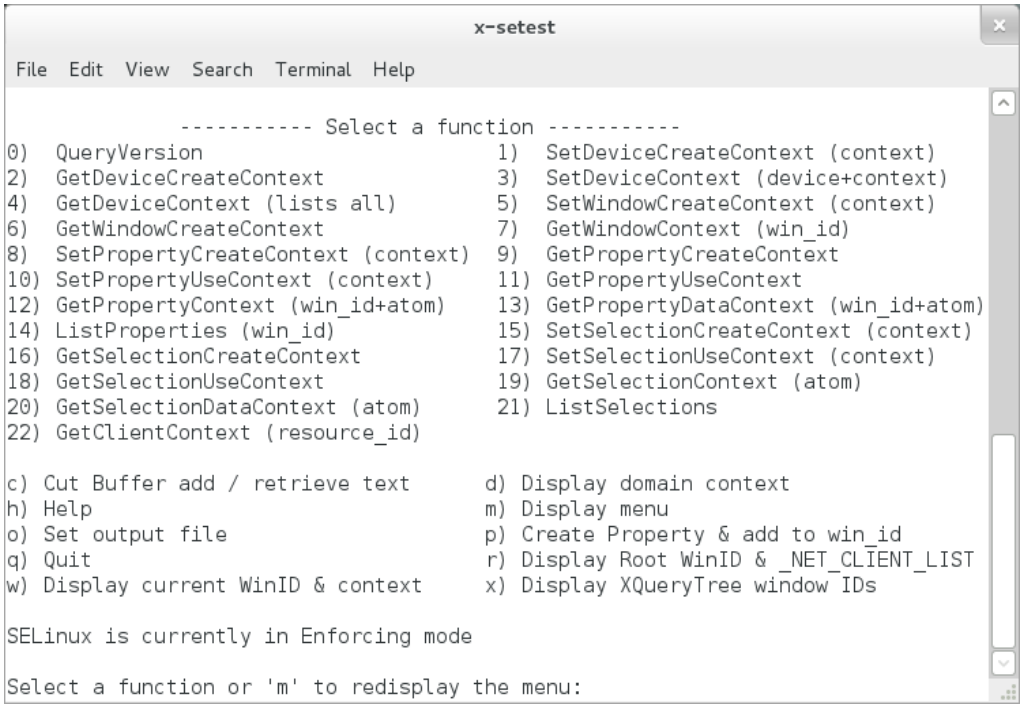
Note that if using Fedora 17 with `xorg-x11-server-1.12.2-2`, the X-server will not load due the following bug: https://bugs.freedesktop.org/show_bug.cgi?id=50641 (that also contains a patch to fix the problem).

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

2.22.1 Notebook Examples

There are three sample X-widows applications in the source code tarball in the `x-windows` directory that use the XSELinux features:

1. A test tool to set and retrieve context information using the XSELinux functions that form part of the object manager (these are listed in Table 12). This tool will also allow properties to be created and display window IDs. The following screen shot shows all the options available:



There is another version of this tool in the `../python-xcb` directory that uses the Python XCB bindings. As these are not distributed by Red Hat there is a `README` with instructions on how to build and install `xpyb`.

2. A simple application to retrieve all the `x_context` information (there is also another in `libselinux/examples/selabel_x_example.c` that is interactive).
3. A sample selection manager for polyinstantiated "`PRIMARY`" selections with a demo application and policy module suitable for the F-16 or F-17 targeted environment. This uses many of the functions listed in Table 12.

2.22.2 Infrastructure Overview

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

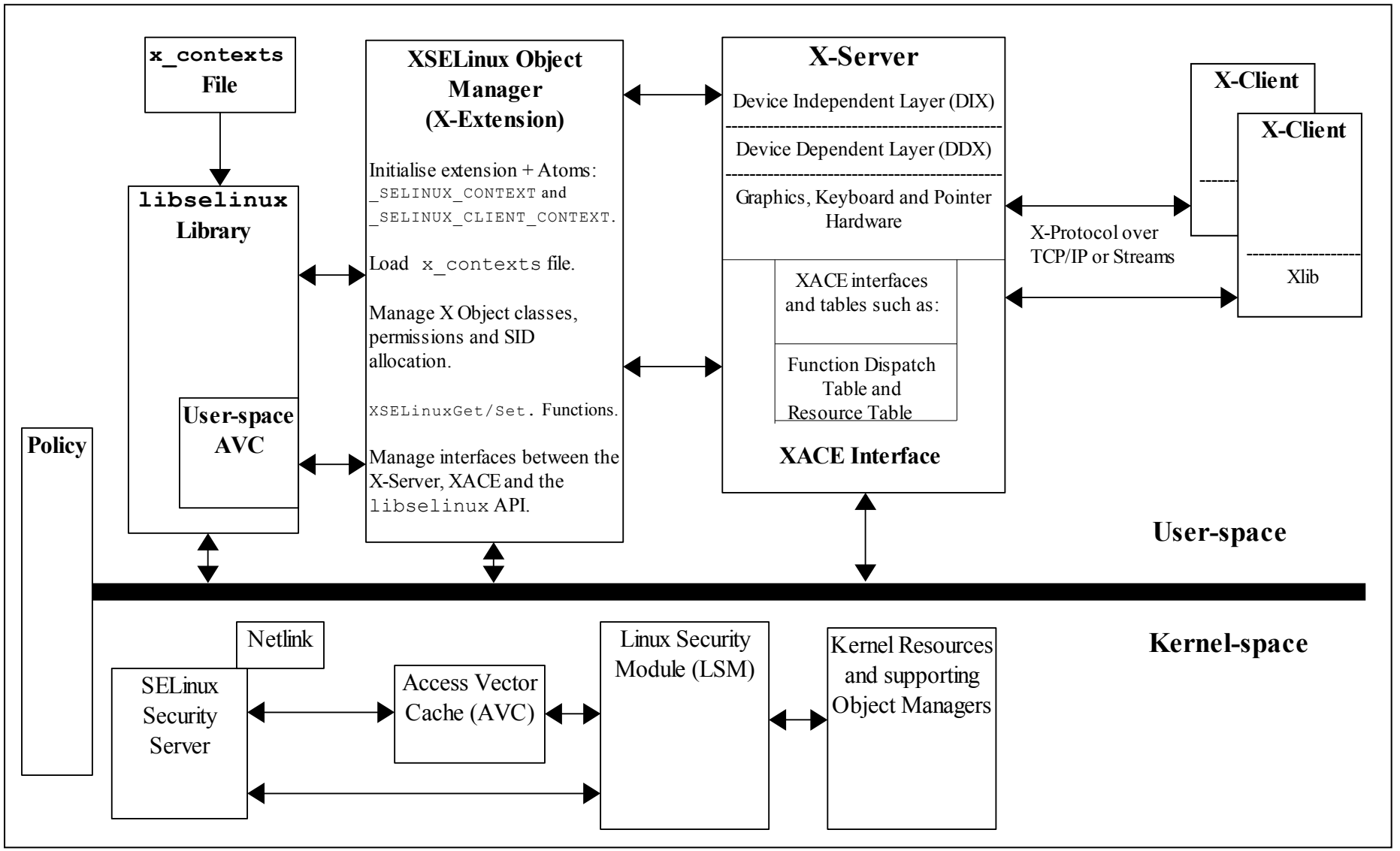


Figure 2.24: X-Server and XSELinux Object Manager – *Showing the supporting services. The kernel space services are discussed in the Linux Security Module and SELinux section.*

Using Figure 2.24, the major components that form the overall XSELinux OM are (top left to right):

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

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

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

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

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

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

This interface is defined in the "X Access Control Extension Specification" [Ref. 19]. The specification also defines the hooks available to OMs and how they should be used. The provision of polyinstantiation services for properties and selections is also discussed. The XACE interface is a similar service to the LSM that supports the kernel OMs.

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

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

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

2.22.2.1 Polyinstantiation

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

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

The source tarball has a simple 'Selection Manager' to show polyinstantiation using the X-Windows selection[30] mechanism.

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

2.22.3 Configuration Information

This section covers:

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

2.22.3.1 Enable/Disable the OM from Policy Decisions

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

```
setsebool -P xserver_object_manager true
```

If the boolean is set to false, the x-server log will indicate that "SELinux: Disabled by boolean". Important note - If the boolean is not present in a policy then the object manager will always be enabled (therefore if not required then either do not include the object manager in the X-server build or add the boolean to the policy and set it to false).

2.22.3.2 Determine OM X-extension Opcode

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

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

[30] That uses InterClient Communication (ICC) allowing X-clients to communicate and exchange information.

2.22.3.3 Configure OM Enforcement Mode

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

`"SELinux mode disabled"`

`"SELinux mode permissive"`

`"SELinux mode enforcing"`

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

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

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

2.22.3.4 The `x_contexts` File

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

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

or for polyinstantiation support:

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

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

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

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

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

Notes:

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

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

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

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

A full description of the x_contexts file format is given in the x_contexts File section.

2.22.4 SELinux Extension Functions

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

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

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

2.23 Sandbox Services

Fedora has support for three types of sandbox services in F-17:

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

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

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

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

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

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

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

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

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

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

3. Virtulisation sandboxing of applications using either KVM/qemu or LXC[31] (Linux Containers) (`virt-sandbox` - see http://people.redhat.com/berrange/fosdem-2012/libvirt-sandbox-fosdem-2012.pdf that contains a good overview).

   This service is available in the `libvirt-sandbox` package and provides an API and command line services to start sessions. There is currently limited

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

policy support for `virt-sandbox` as it primary aim is for developers to build services and provide the appropriate policy.

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

An LXC example:

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

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

```
module lxc_example 1.0.0;

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

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

that was built and installed as follows:

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

2.24 SE-PostgreSQL

This section gives an overview of PostgreSQL version 9.1 with the `sepgsql` extension to support SELinux labeling. It assumes some basic knowledge of PostgreSQL that can be found at: http://wiki.postgresql.org/wiki/Main_Page

It is important to note that PostgreSQL from version 9.1 has the necessary infrastructure to support labeling of database objects via external 'providers'. The `sepgsql` extension has been added as the SELinux labeling provider. This is not installed by default but as an option as outlined in the sections that follow. Because of these changes the original version 9.0 patches are no longer supported (i.e. the SE-PostgreSQL database engine is replaced by PostgreSQL database engine 9.1 plus the

sepgsql extension). A consequence of this change is that row level labeling is no longer supported.

The features of sepgsql 9.1 and its setup are covered in the following document:

http://www.postgresql.org/docs/devel/static/sepgsql.html

or if PostgreSQL is already installed:

file:///usr/share/doc/postgresql-9.1.4/html/sepgsql.html

2.24.1 Notebook Examples

The Notebook source tarball contains a number of examples using PostgreSQL / sepgsql (a.k.a. SE-PostgreSQL). These are located in the sepgsql-9.1 directory and covers the following:

a) Installing and initialising a simple database that has SELinux security labels attached to various objects.

b) Selecting data from columns with different labels locally and remotely (using NetLabel (fallback mode) and Labeled IPSec).

c) Using Apache with thread bounding (mod_selinux) and PostgreSQL authentication to select data from columns with different labels locally and remotely (using NetLabel).

d) Building additional sepgsql SELinux functions using libselinux. These are also tested when using the Apache/PHP web services.

2.24.2 sepgsql Overview

SE-PostgreSQL adds SELinux mandatory access controls (MAC) to database objects such as tables, columns, views, functions, schemas and sequences. Figure 2.25 shows a simple database with one table, two columns and three rows, each with their object class and associated security context (the Internal Tables section shows these entries for the testdb database from the Notebook tarball example). The database object classes and permissions are described in Appendix A - Object Classes and Permissions.

Figure 2.25: Database Security Context Information - *Showing the security contexts that can be associated to a schema, table and columns. Note that the row level labeling has been removed from version 9.1.*

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

If a client is connecting remotely and labeled networking is required, then it is possible to use IPSec or NetLabel as discussed in the SELinux Networking Support section (the "Security-Enhanced PostgreSQL Security Wiki" [Ref. 3] also covers these methods of connectivity with examples).

Using Figure 2.26, the database client application (that could be provided by an API for Perl/PHP or some other programming language) connects to a database and executes SQL commands. As the SQL commands are processed by PostgreSQL, each operation performed on an object is checked by the object manager (OM) to see if the opration is allowed by the security policy or not.

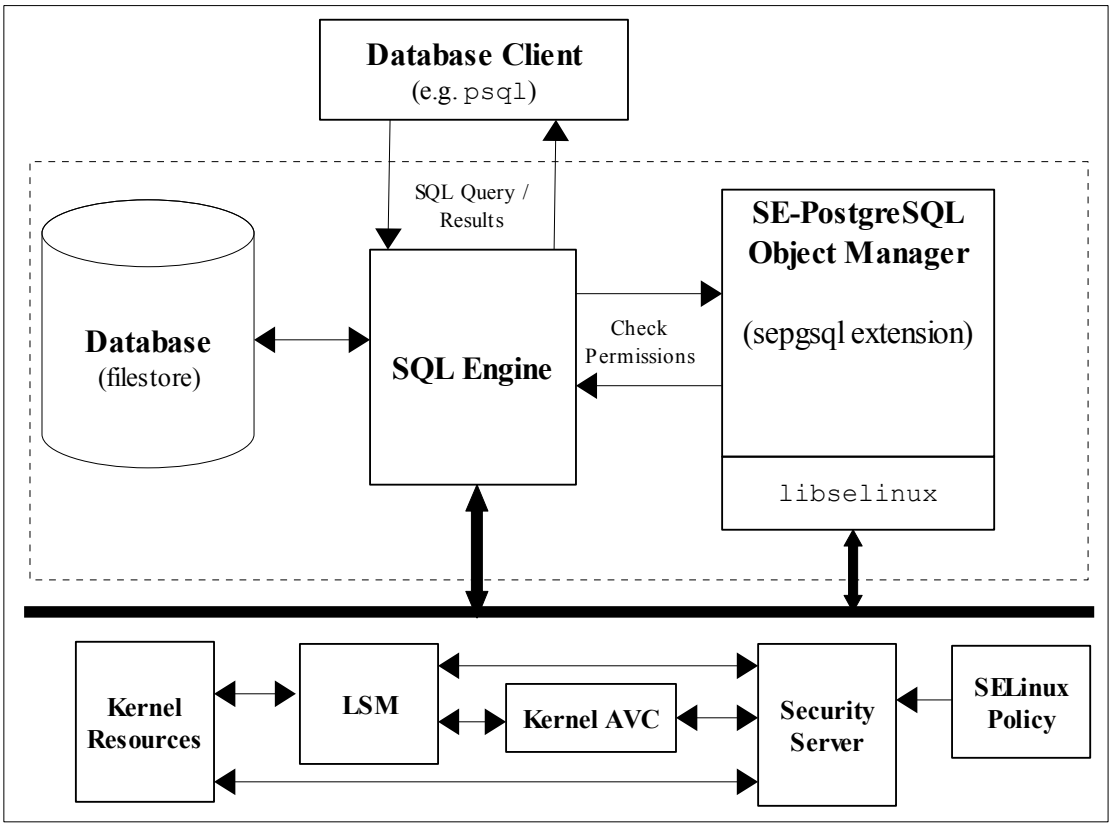


Figure 2.26: SE-PostgreSQL Services - *The Object Manager checks access permissions for all objects under its control.*

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

2.24.3 Installing SE-PostgreSQL

The http://www.postgresql.org/docs/devel/static/sepgsql.html page contains all the information required to install PostgreSQL and the `sepgsql` extension, however this section will give a short version of the installation on F-16 or F-17 from the Notebook tarball `sepgsql-9.1/README` file.

1. Install the following packages:

   `postgresql`

   `postgresql-libs`

   `postgresql-server`

   `postgresql-contrib` - Contains the `sepgsql` extension.

   `postgresql-devel` - Only required to build the additional functions.

   The default policy modules for SE-PostgreSQL are automatically installed with the `targeted` policy.

2. As `root` initialise PostgreSQL:

   ```
   postgresql-setup initdb
   ```

If an error states that the 'Data directory is not empty!' then you have already built a database service (`/var/lib/pgsql/data` exists).

3. To automatically load the SELinux `sepgsql` extension module, the `postgresql.conf` file needs to be updated as follows:

   ```
   vi /var/lib/pgsql/data/postgresql.conf
   ```

   and change the:

   ```
   #shared_preload_libraries = ''
   ```

   entry to read:

   ```
   shared_preload_libraries = 'sepgsql'
   ```

4. Start the PostgreSQL database by:

   ```
   service postgresql start
   ```

5. `su` to the `postgres` user and create roles for any database users giving them superuser rights:

   ```
   su - postgres
   createuser root
   createuser ....
   ```

6. Create a `testdb` database:

   ```
   createdb testdb
   ```

7. Now `Crtl\D` to exit the `postgres` user and as `root` stop the db:

   ```
   service postgresql stop
   ```

8. The database now needs to be initialised to support labeling. This involves running an SQL script that is supplied in the `postgresql-contrib` package (note: user must not be `root`):

   ```
   su - postgres
   ```

   Paste the following in to the terminal session:

   ```
   for testdb in template0 template1 postgres; do
   postgres --single -F -c exit_on_error=true testdb \
   </usr/share/pgsql/contrib/sepgsql.sql
   done
   ```

   The `/usr/share/pgsql/contrib/sepgsql.sql` script adds `sepgsql` functions and then runs `sepgsql_restorecon(NULL);` to label the database objects.

9. Now `Crtl\D` to exit the `postgres` user and as `root` start the db:

   ```
   service postgresql start
   ```

The `testdb` database should now be labeled using default contexts from the `/etc/selinux/targeted/contexts/sepgsql_contexts` file. The `README` file gives further information on initialising the database and reading columns according the labels set on them.

2.24.4 `SECURITY LABEL` SQL Command

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

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

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

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

2.24.5 Additional SQL Functions

The following functions have been added:

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

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

The Notebook tarball contains additional SQL functions for supporting `libselinux` functions (e.g. `sepgsql_compute_av`).

2.24.6 Additional `postgresql.conf` Entries

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

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

```
shared_preload_libraries = 'sepgsql'
```

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

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

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

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

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

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

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

2.24.7 Logging Security Events

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

2.24.8 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 `pg_seclabel` table that holds the security label and other references. The `pg_seclabel` is described in Table 13 that has been taken from http://www.postgresql.org/docs/9.1/static/catalog-pg-seclabel.html.

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

Table 13: `pg_seclabel` Table Columns

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

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

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

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

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

2.25 Apache SELinux Support

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

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

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

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

The objective of these Apache add-on services is to achieve a fully SELinux-aware web stack (although not there yet). For example, currently the LAPP[32] (Linux, Apache, PostgreSQL, PHP / Perl / Python) stack has the following support:

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

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

2.25.1 `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:

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

a) Without `mod_selinux` enabled the web applications security context is identical to the Apache web server process, it is therefore not possible to restrict 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[33]). It is also possible to restrict its privileges as described in the Bounds Overview section.

3. The web application exits, handing control back to the web server that replies with the HTTP response.

2.25.2 Bounds Overview

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

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

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

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

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

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

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

Should there be a valid `typebounds` policy statement and the child domain exercises a privilege greater that that of the parent domain, the operation will be denied and an `SELINUX_ERR` entry will be added to the audit log stating

[33] It is also possible to obtain the domain from a PostgreSQL table (this is how the demo in the Notebook tarball obtains the context).

'op=security_compute_av reason=bounds' with the context strings and the denied class and permissions.

2.25.2.1 Notebook Examples

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

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

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

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

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

3. SELinux Configuration Files

3.1 Introduction

This section explains each SELinux configuration file with its format, example content and where applicable, any supporting SELinux commands or `libselinux` library API function names.

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

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

1. Global Configuration files that affect the active policy and their supporting SELinux-aware applications, utilities or commands. 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. These are located at `/etc/selinux/<SELINUXTYPE>` for the run time configuration and `/var/lib/selinux/<SELINUXTYPE>` for the run time policy store (although on some systems the policy store files may still be located at `/etc/selinux/<SELINUXTYPE>/modules`).

   These two areas are described as follows:

   a. The Policy Store Configuration files are 'private'[34] and managed by the **`semanage`**`(8)` and **`semodule`**`(8)` commands[35]. These are used to build the majority of the Policy Configuration files.

   b. The Policy Configuration files that are used when the policy is activated.

   However note that there can be multiple named policy configuration areas on a system.
3. SELinux Kernel Configuration files located under the `/sys/fs/selinux` directory and reflect the current configuration of SELinux and the active policy. This area is used extensively by the `libselinux` library for userspace object managers and other SELinux-aware applications. These files and directories should not be updated by users (the majority are read only anyway), however they can be read to check various configuration parameters.

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

[34] They should NOT be edited as together they describe the 'policy'.

[35] The `system-config-selinux` GUI (supplied in the `polycoreutils-gui` rpm) can also be used to manage users, booleans and the general configuration of SELinux as it calls **`semanage`**`(8)`, however it does not manage all that the `semanage` command can (it also gets bitter & twisted if there are no MCS/MLS labels on some operations).

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 file has a man page called **`selinux_config`**`(5)`, this is because 'config' has already been taken. The `config` file controls the state of SELinux using the following parameters:

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

Where:

`SELINUX`	This entry can contain one of three values: **`enforcing`** SELinux security policy is enforced. **`permissive`** SELinux logs warnings (see the Auditing SELinux Events section) instead of enforcing the policy (i.e. the action is allowed to proceed). **`disabled`** No SELinux policy is loaded.
`SELINUXTYPE`	The `policy_name` is used as the directory name where the active policy and its configuration files will be located. The system will then use this information to locate and load the policy contained within this directory structure. The policy directory must be located at: `/etc/selinux/<policy_name>/`
`SETLOCALDEFS`	This optional field should be set to `0` (or the entry removed) as the policy store management infrastructure (**`semanage`**`(8)` / **`semodule`**`(8)`) 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 a login if there is no matching or default entry in the **seusers** file or if the file is missing. It is checked by the libselinux function **getseuserbyname**(3) that is used by SELinux-aware login applications such as **PAM**(8). If it is set to 0 or the entry missing: **getseuserbyname**(3) will return the GNU / Linux user name as the SELinux user. If it is set to 1: **getseuserbyname**(3) will fail.
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.

Example config file contents are:

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

3.2.2 /etc/selinux/semanage.conf File

The **semanage.config**(5) file controls the configuration and actions of the **semanage**(8) and **semodule**(8) set of commands using the following parameters:

```
module-store = method
policy-version = policy_version
expand-check = 0|1
```

```
file-mode = mode
save-previous = true|false
save-linked = true|false
disable-genhomedircon = true|false
handle-unknown = allow|deny|reject
bzip-blocksize = 0|1..9
bzip-small true|false
usepasswd = true|false
ignoredirs dir [;dir] ...
target-platform = selinux | xen
mls = true|false

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

Where:

`module-store`	The `method` can be one of four options: `direct` — `libsemanage` will write directly to a module store. This is the default value. `source` — `libsemanage` manipulates a source SELinux policy. `/foo/bar` — Write via a policy management server, whose named socket is at `/foo/bar`. The path must begin with a '`/`'. `foo.com:4242` — Establish a TCP connection to a remote policy management server at `foo.com`. If there is a colon then the remainder is interpreted as a port number; otherwise default to port 4242.
`policy-version`	This optional entry can contain a policy version number, however it is normally commented out as it then defaults to that supported by the system.
`expand-check`	This optional entry controls whether hierarchy checking on module expansion is enabled (`1`) or disabled (`0`). The default is `0`. It is also required to detect the presence of policy rules that are to be excluded with `neverallow` rules.
`file-mode`	This optional entry allows the file permissions to be set on runtime policy files. The format is the same as the `mode` parameter of the `chmod` command and

	defaults to `0644` if not present.
`save-previous`	This optional entry controls whether the previous module directory is saved (`TRUE`) after a successful commit to the policy store. The default is to delete the previous version (`FALSE`).
`save-linked`	This optional entry controls whether the previously linked module is saved (`TRUE`) after a successful commit to the policy store. Note that this option will create a `base.linked` file in the module policy store. The default is to delete the previous module (`FALSE`).
`disable-genhomedircon`	This optional entry controls whether the embedded `genhomedircon` function is run when using the **`semanage`**`(8)` command. The default is `FALSE`.
`handle-unknown`	This optional entry controls the kernel behaviour for handling permissions defined in the kernel but missing from the policy (that are declared at the start of the `base.conf` (loadable policy) or `policy.conf` (monolithic policy)). The options are: `allow` the permission, `reject` by not loading the policy or `deny` the permission. The default is `deny`. See the SELinux Filesystem section for how these are reported in `/selinux`. Note: to activate any change, the base policy needs to be rebuilt with the `semodule -B` command.
`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` or `--small` option). If `FALSE` or no entry present, then does not try to reduce memory requirements.
`usepasswd`	When this optional entry is set to `TRUE` `semanage` will scan all password records for home directories and set up their labels correctly. If set to `FALSE` (the default if no entry present), then only the `/home` directory will be automatically re-

	labeled.
`ignoredirs`	With a list of directories to ignore (separated by `';'`) when setting up users home directories. This is used by some distributions to stop labeling `/root` as a home directory.
`target-platform`	Build a platform for SELinux (`selinux`) or XEN (`xen`), as XEN has additional policy statements. The default is `selinux`. This was added for the CIL compiler.
`mls`	Build policy as MLS/MCS (`true`) or non-MLS (`false`). The default is `false`. This was added for the CIL compiler.

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

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` and `restorecond-user.conf` Files

The `restorecond.conf` file contains a list of files that may be created by applications with an incorrect security context. The **`restorecond`**`(8)` daemon will then watch for their creation and automatically correct their security context to that

specified by the active policy file context configuration files[36] (located in the /etc/selinux/<policy_name>/contexts/files directory).

Each line of the file contains the full path of a file or directory. Entries that start with a tilde (~) will be expanded to watch for files in users home directories (e.g. ~/public_html would cause the daemon to listen for changes to public_html in all logged on users home directories).

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

Example restorecond.conf file contents are:

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

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

Example restorecond-user.conf file contents are:

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

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

3.2.4 /etc/selinux/newrole_pam.conf

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

3.2.5 /etc/sestatus.conf File

The **sestatus.conf**(5) file is used by the **sestatus**(8) command to list files and processes whose security context should be displayed when the -v flag is used (sestatus -v).

The file has the following parameters:

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

[process]
```

[36] The daemon uses functions in libselinux such as **matchpathcon**(3) to manage the context updates.

```
List of processes to display context
```

Example `sestatus.conf` file contents are:

```
# /etc/sestatus.conf

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

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

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

The **`sepermit.conf`**(5) file is used by the `pam_sepermit.so` module to allow or deny a user login depending on whether SELinux is enforcing the policy or not. An example use of this facility is the Red Hat kiosk policy where a terminal can be set up with a guest user that does not require a password, but can only log in if SELinux is in enforcing mode.

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

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

The usage is described in **`pam_sepermit`**(5), with the following example that describes the configuration:

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

# Each line can contain 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
#
#   ignore - The module will never return PAM_SUCCESS status
#     for the user.

# An example entry for 'kiosk mode':
xguest:exclusive
```

3.3 Policy Store Configuration Files

Because there can be multiple policy stores on a system, each file discussed in this section is relative to the policy name as follows:

```
/var/lib/selinux/<policy_name>
```

The Policy Store files in the `/var/lib/selinux/<policy_name>/modules` area are either installed, updated or built by the **`semodule`**`(8)` and **`semanage`**`(8)` commands, and as a part of the process, relevant files are copied to the Policy Configuration files area.

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

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

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

3.3.1 `modules/` Files

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

```
semanage.read.LOCK
```

```
semanage.trans.LOCK
```

3.3.2 `modules/active/base.pp` File

This is the packaged base policy that contains the mandatory modules and policy components such as object classes, permission declarations and initial SIDs.

3.3.3 modules/active/base.linked File

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

3.3.4 modules/active/commit_num File

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

3.3.5 modules/active/file_contexts.template File

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

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

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

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

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) [37] to generate individual SELinux user entries in the file_contexts.homedirs file as discussed in the ./modules/active/file_contexts.homedirs section.

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

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

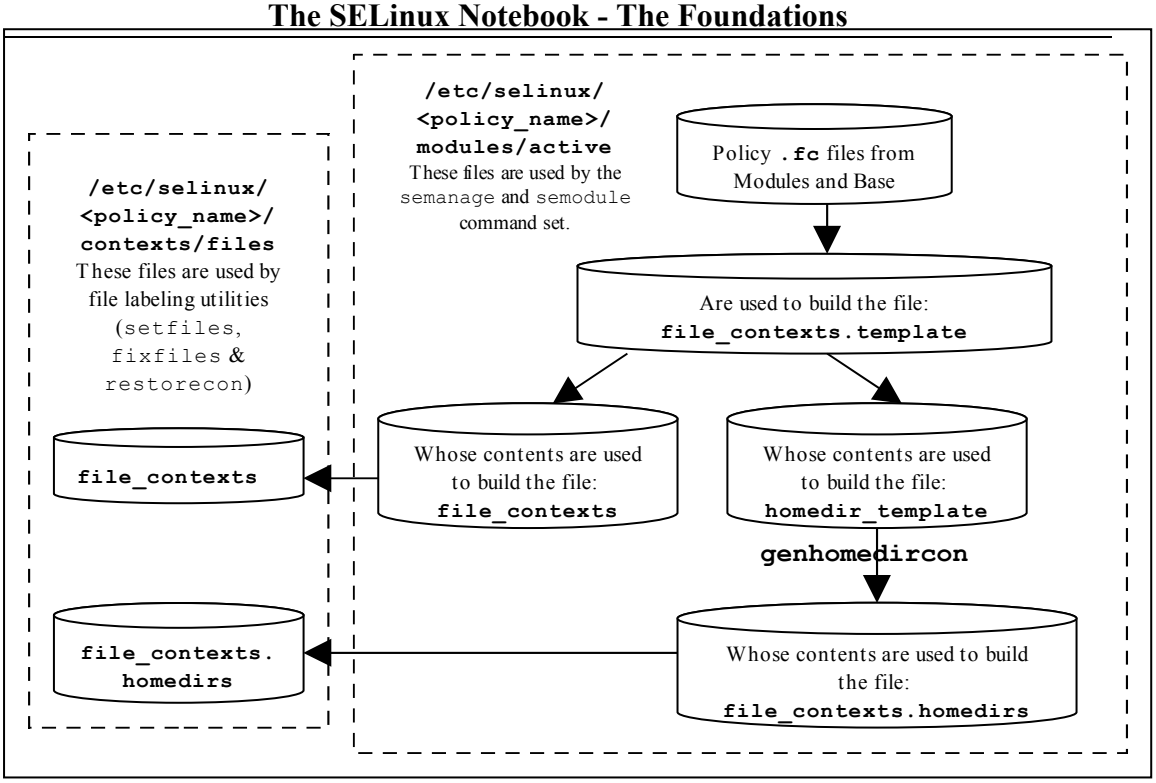


Figure 3.1: File Context Configuration Files - *The two files copied to the policy area will be used by the file labeling utilities to relabel files.*

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 that may be 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 keyword definitions below and the `./modules/active/homedir_template` file section for their usage).
`file_type`	The `file_type` options are:

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

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

HOME_ROOT	This keyword is replaced by the GNU / Linux users root home directory, normally '/home' 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:
	`user user_u prefix user;`
	`user staff_u prefix staff;`
	`user group1_u prefix group1;`
	It is used for files and directories within the users home directory area when relabeling takes place to allow the domain context to be based on a specific role (or any identifier !!) to allow easier identification in log files.
	It can be added by the semanage user command as follows:
	`# Add prefix for SELinux user:`
	`semanage user -a -R staff_r -P group1 group1_u`
	`# Add login user:`
	`semanage login -a -s group1_u rch`
	The usage is similar to the Reference Policy 'per_role_template' (<param name="userdomain_prefix">) that is an optional component of the external interface file (see the ftp.if or ssh.if files in the Reference Policy source).

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

3.3.6 `modules/active/file_contexts` File

This file becomes the policies `./contexts/files/file_contexts` file and is built from entries in the `./modules/active/file_contexts.template` file as explained above and shown in Figure 3.1. 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
# that will be replaced by the users GNU / Linux user id when
# the file labeling utilities are run.
# The other keywords HOME_DIR, HOME_ROOT and ROLE have been
# extracted and put in the homedir_template file.

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

3.3.7 modules/active/homedir_template File

This file is built from entries in the file_contexts.template file (as shown in Figure 3.1) and explained in the ./modules/active/file_contexts.template section.

The file is used by genhomedircon, semanage login or semanage user to generate individual user entries in the file_contexts.homedirs file.

The homedir_template file has the same per line format as the ./modules/active/file_contexts.template file.

Example file contents:

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

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

3.3.8 modules/active/file_contexts.homedirs File

This file becomes the policies ./contexts/files/file_contexts.homedirs file when building policy as shown in Figure 3.1. It is then used by the file labeling utilities to ensure that users home directory areas are labeled according to the policy.

The file can be built by the genhomedircon command (that just calls /usr/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 ./modules/active/file_contexts.template file, however the HOME_DIR, ROOT_DIR and ROLE keywords will be replaced as explained in the keyword definitions section above. Note that the ROLE keyword will only be replaced for those valid types within the policy (for example if staff_irc_home_t cannot be found in the policy it will be silently dropped from the file_context.homedirs when being built **True?**.

Example file_contexts.homedirs contents:

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

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

3.3.9 modules/active/netfilter_contexts & netfilter.local File

These files are not used at present. There is code to produce a netfilter_contexts file for use by the GNU/Linux iptables service[38] in the Reference Policy that would generate a file similar to the example below, however there seems much debate on how they should be managed (see bug 201573 – Secmark iptables integration for details).

[38] This uses SECMARK labeling that has been utilised by SELinux as described in the SELinux Networking Support section.

3.3.10 modules/active/policy.kern File

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

3.3.11 modules/active/seusers.final and seusers Files

The seusers.final file maps GNU / Linux users to SELinux users and becomes the policies seusers[39] file as discussed in the ./seusers section. The seusers.final file is built or modified when:

1. Building a policy where an optional seusers file has been included in the base package via the **semodule_package**(8) command (signified by the -s flag) as follows[40]:

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

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

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

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

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

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

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

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

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

Where:

user_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.

[39] 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.

[40] The Reference Policy Makefile 'Rules.modular' script uses this method to install the initial seusers file.

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:

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

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

   The users_extra file can optionally be included in the base package via the **semodule_package**(8) command (signified by the -u flag) as follows[41]:

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

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

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

The sections that follow will:

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

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

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

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.

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 **`semanage`**`(8)`).

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

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

semanage boolean -m -0 ext_gateway_audit
```

The resulting `booleans.local` file would be:

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

ext_gateway_audit=0
```

3.3.14 `modules/active/file_contexts.local` File

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

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

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

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

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

semanage fcontext -a -t user_t /usr/move_file

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

The resulting `file_contexts.local` file would be:

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

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

3.3.15 modules/active/interfaces.local File

This file is created and updated by the `semanage interface` command to hold network interface information that was not delivered by the core policy (i.e. they are not defined in `base.conf` file). The new interface information is then built into the policy by the **`semanage`**`(8)` command.

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

3.3.16 modules/active/nodes.local File

This file is created and updated by the `semanage node` command to hold network address information that was not delivered by the core policy (i.e. they are not defined in `base.conf` file). The new node information is then built into the policy by the **`semanage`**`(8)` command.

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

3.3.17 modules/active/ports.local File

This file is created and updated by the `semanage port` command to hold network port information that was not delivered by the core policy (i.e. they are not defined in `base.conf` file). The new port information is then built into the policy by the **`semanage`**`(8)` command.

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

3.3.18 modules/active/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 -l
ext_gateway     1.1.0
int_gateway     1.1.0
move_file       1.1.0
netlabel        1.0.0    Disabled
```

3.4 Policy Configuration Files

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

`/etc/selinux/<policy_name>`

The majority of files are installed by the Reference Policy, **`semanage`**`(8)` or **`semanage`**`(8)` commands. It is possible to build custom monolithic policies that only use the files installed in this area (i.e. do not use `semanage` or `semodule`). For example the simple monolithic policy described in the Notebook source tarball could run at `init 3` (i.e. no X-Windows) and only require the following configuration files:

`./policy/policy.26` – The binary policy loaded into the kernel.

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

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

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

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

3.4.1 `seusers` File

The **`seusers`**`(5)` file is used by login programs (normally via the `libselinux` library) and maps GNU / Linux users (as defined in the `user` / `passwd` files) to SELinux users (defined in the policy). A typical login sequence would be:

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

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

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

Example seusers file contents:

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

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

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

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

Supporting libselinux API functions are:

```
getseuser
getseuserbyname
```

3.4.2 booleans and booleans.local File

Generally these **booleans**(5) files are not present if **semanage**(8) is being used to manage booleans (see the modules/active/booleans.local File section). However if semanage is not being used or there is an SELinux-aware application that uses the libselinux functions listed below, then these files may be present (they could also be present in older Reference or Example policies):

security_set_boolean_list - Writes a boolean.local file if flag permanent = '1'.

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

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

```
boolean_name value
```

Where:

boolean_name	The name of the boolean.
value	The default setting for the boolean that can be

	one of the following: `true | false | 1 | 0`

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

3.4.3 `booleans.subs` File

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

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

```
policy_bool_name  new_name
```

Where:

`policy_bool_name`

The policy boolean name.

`new_name`

The new boolean name.

Example:

```
# ./booleans.subs

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

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

Supporting `libselinux` API functions are:

```
selinux_booleans_subs_path
security_get_boolean_names
security_set_boolean
```

3.4.4 `setrans.conf` File

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

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

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

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

```
disable = 1
```

The **`semanage`**`(8)` 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 `secolor.conf` File section.

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

```
# Syntax

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

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

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

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

# Aruba - bit 201
~c200,~c201=ABW
~c200,~c201=AA
# Afghanistan - bit 202
```

```
~c200,~c202=AFG
~c200,~c202=AF
# repeat as required...

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

# New domain
Domain=NAME2

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

Example file contents:

```
# ./setrans.conf - 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 -l

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

Supporting `libselinux` API functions are:

```
selinux_translations_path
selinux_raw_to_trans_context
selinux_trans_to_raw_context
```

3.4.5 `secolor.conf` File

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

```
color color_name = #color_mask

context_component string fg_color_name bg_color_name
```

Where:

`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 RGB colours with black being `#000000` and white being `#ffffff`.
`context_component`	The colour translation supports different colours on the context string components (`user`, `role`, `type` and `range`). Each component is on a separate line.
`string`	This is the `context_component` string that will be matched with the `raw` context component passed by **`selinux_raw_context_to_color`**`(3)` A wildcard '**`*`**' may be used to match any undefined `string` for the `user`, `role` and `type` `context_component` entries only A wildcard '**`*`**' may be used to match any undefined `string` for the `user`, `role` and `type` `context_component` entries only.
`fg_color_name`	The `color_name` string that will be used as the

	foreground colour. A `color_mask` may also be used.
`bg_color_name`	The `color_name` string that will be used as the background colour. A `color_mask` may also be used.

Example file contents:

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

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

Supporting `libselinux` API functions are:

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

3.4.6 `policy/policy.<ver>` File

This is the binary policy file that is loaded into the kernel to enforce policy and is built by either `checkpolicy` or `semodule`. Life is too short to describe the format but the `libsepol` source could be used as a reference or for an overview the "SELinux Policy Module Primer" [Ref. 4] notes.

The file name extension is the policy database version supported by the GNU / Linux release and can be found by executing the following command:

```
cat /selinux/policyvers
26
```

The different versions are discussed in the Policy Versions section.

3.4.7 contexts/customizable_types File

The **customizable_types**(5) file contains a list of types that will not be relabeled by the **setfiles**(8) or **restorecon**(8) commands. The commands check this file before relabeling and excludes those in the list unless the -F flag is used (see the man pages).

The file format is as follows:

```
type
```

Where:

type	The type defined in the policy that needs to 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
```

3.4.8 contexts/default_contexts File

The **default_contexts**(5) file is used by SELinux-aware applications that need to set a security context for user processes (generally the login applications) where:

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

The file format is as follows:

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

Or:

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

Where:

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

Example file contents:

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

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
#
system_r:local_login_t   user_r:user_t staff_r:staff_t sysadm_r:sysadm_t
unconfined_r:unconfined_t
#
system_r:remote_login_t  user_r:user_t staff_r:staff_t unconfined_r:unconfined_t
#
system_r:sshd_t          user_r:user_t staff_r:staff_t sysadm_r:sysadm_t
unconfined_r:unconfined_t
```

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

system_r:crond_t:s0         system_r:system_crond_t:s0
system_r:local_login_t:s0   user_r:user_t:s0
system_r:remote_login_t:s0  user_r:user_t:s0
system_r:sshd_t:s0          user_r:user_t:s0
system_r:sulogin_t:s0       sysadm_r:sysadm_t:s0
system_r:xdm_t:s0           user_r:user_t:s0
```

Supporting `libselinux` API functions are:

```
# Note that the ./contexts/users/[seuser_id] file is also read
# by some of these functions.

selinux_contexts_path
selinux_default_context_path
get_default_context
get_ordered_context_list
get_ordered_context_list_with_level
get_default_context_with_level
get_default_context_with_role
```

```
get_default_context_with_rolelevel
query_user_context
manual_user_enter_context
```

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

However when adding the loadable module that used another type (`ext_gateway_t`) but with the same role and user (e.g. `unconfined_u:unconfined_r:ext_gateway_t`), then it was found that the login process would always set the logged in user context to `unconfined_u:unconfined_r:ext_gateway_t` (i.e. the login application now had a choice and choose the wrong one, probably because the types are sorted and 'e' comes before 'u').

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

```
unconfined_r:unconfined_t  unconfined_r:unconfined_t
```

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

3.4.9 `contexts/dbus_contexts` File

This file is for the dbus messaging service daemon (a form of IPC) that is used by a number of GNU / Linux applications such as GNOME and KDE desktops. If SELinux is enabled, then this file needs to exist in order for these applications to work. The **`dbus-daemon`**`(1) man` page details the contents and the Free Desktop web site has detailed information at:

http://dbus.freedesktop.org

Example file contents:

```
# ./contexts/dbus_contexts - 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>
  <selinux>
  </selinux>
</busconfig>
```

Supporting `libselinux` API function is:

```
selinux_context_path
```

3.4.10 contexts/default_type File

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

The file format is as follows:

```
role:type
```

Where:

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_default_type_path
get_default_type
```

3.4.11 contexts/failsafe_context File

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

The file format is as follows:

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

Where:

role:type[:range]	A single line that has a valid context to allow an administrator access to the system.

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
get_default_context
get_default_context_with_role
get_default_context_with_level
get_default_context_with_rolelevel
get_ordered_context_list
get_ordered_context_list_with_level
```

3.4.12 `contexts/initrc_context` File

This is used by the **`run_init`**`(8)` command to allow system services to be started in the same security context as `init`. This file could also be used by other SELinux-aware applications for the same purpose.

The file format is as follows:

```
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
```

3.4.13 contexts/netfilter_contexts File

This file will support the Secmark labeling for Netfilter / `iptable` rule matching of network packets, however it is currently unused (see the `./modules/active/netfilter_contexts & netfilter.local` file section for further information).

Supporting `libselinux` API functions are:

```
selinux_context_path
selinux_netfilter_context_path
```

3.4.14 contexts/removable_context File

The **`removable_context`**`(5)` file contains a single default label that should be used for removable devices that are not defined in the `contexts/files/media` file.

The file format is as follows:

```
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
```

3.4.15 contexts/securetty_types File

The **`securetty_types`**`(5)` file is used by the **`newrole`**`(1)` command to find the `type` to use with `tty` devices when changing roles or levels.

The file format is as follows:

```
type
```

Where:

`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
```

3.4.16 contexts/sepgsql_contexts File

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

The file format is as follows:

Each line within the database contexts file is as follows:

```
object_type  object_name  context
```

Where:

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

	should be the last entry in the object_type block.
context	The security context that will be applied to the object.

Example file contents:

```
# ./contexts/sepgsql_contexts file

# object_type object_name  context
db_database   my_database  system_u:object_r:my_sepgsql_db_t:s0
db_database   *            system_u:object_r:sepgsql_db_t:s0
db_schema     *.*          system_u:object_r:sepgsql_schema_t:s0
```

3.4.17 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
```

3.4.18 contexts/virtual_domain_context File

The **virtual_domain_context**(5) file is used by the virtulization API (libvirt) and provides the domain contexts available 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
```

3.4.19 `contexts/virtual_image_context` File

The **`virtual_image_context`**`(5)` file is used by the virtulization API (`libvirt`) and provides the image contexts that are available in the policy..

Example file contents:

```
# ./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
```

3.4.20 `contexts/x_contexts` File

The **`x_contexts`**`(5)` file provides the default security contexts for the X-Windows SELinux security extension. The usage is discussed in the X-windows SELinux Support section and examples of how to add additional entries is shown in the X-Windows section of the Notebook source tarball. The MCS / MLS version of the file has the appropriate `level` or `range` information added.

A typical entry is as follows:

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

Where:

`object_type`	These are types of object supported and valid entries are: `client`, `property`, `poly_property`, `extension`, `selection`, `poly_selection` and `events`.

object_name	These are the object names of the specific X-server resource such as PRIMARY, CUT_BUFFER0 etc. They are generally defined in the X-server source code (protocol.txt and 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.

Example file contents:

```
#
# Config file for XSELinux extension
#

### Rules for X Clients
# The default client rule defines a context to be used for all clients
# connecting to the server from a remote host.
#
client *              system_u:object_r:remote_t

#
### Rules for X Properties
# Property rules map a property name to a context.  A default property
# rule indicated by an asterisk should follow all other property rules.
#
# Properties that normal clients may only read
property _SELINUX_*      system_u:object_r:seclabel_xproperty_t

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

# Default fallback type
property *            system_u:object_r:xproperty_t

#
### Rules for X Extensions
# Extension rules map an extension name to a context.  A default extension
# rule indicated by an asterisk should follow all other extension rules.
#
# Restricted extensions
extension SELinux     system_u:object_r:security_xextension_t

# Standard extensions
extension *           system_u:object_r:xextension_t

#
### Rules for X Selections
# Selection rules map a selection name to a context.  A default selection
# rule indicated by an asterisk should follow all other selection rules.
#
# Standard selections
selection PRIMARY         system_u:object_r:clipboard_xselection_t
selection CLIPBOARD       system_u:object_r:clipboard_xselection_t

# Default fallback type
selection *           system_u:object_r:xselection_t

#
### Rules for X Events
```

```
# Event rules map an event protocol name to a context.  A default event
# rule indicated by an asterisk should follow all other event rules.
#
# Input events
event X11:KeyPress                          system_u:object_r:input_xevent_t
event X11:KeyRelease                        system_u:object_r:input_xevent_t
event X11:ButtonPress                       system_u:object_r:input_xevent_t
event X11:ButtonRelease                     system_u:object_r:input_xevent_t
event X11:MotionNotify                      system_u:object_r:input_xevent_t
event XInputExtension:DeviceKeyPress        system_u:object_r:input_xevent_t
event XInputExtension:DeviceKeyRelease      system_u:object_r:input_xevent_t
event XInputExtension:DeviceButtonPress     system_u:object_r:input_xevent_t
event XInputExtension:DeviceButtonRelease   system_u:object_r:input_xevent_t
event XInputExtension:DeviceMotionNotify    system_u:object_r:input_xevent_t
event XInputExtension:DeviceValuator        system_u:object_r:input_xevent_t
event XInputExtension:ProximityIn           system_u:object_r:input_xevent_t
event XInputExtension:ProximityOut          system_u:object_r:input_xevent_t

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

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

Supporting `libselinux` API functions are:

```
selinux_x_context_path
selabel_open
selabel_close
selabel_lookup
selabel_stats
```

3.4.21 `contexts/files/file_contexts` File

The **`file_contexts`**`(5)` file is managed by the **`semodule`**`(8)` and **`semanage`**`(8)` commands[42] as the policy is updated (adding or removing modules or updating the base), and therefore should not be edited.

The file is used by a number of SELinux-aware commands (**`setfiles`**`(8)`, **`fixfiles`**`(8)`, **`matchpathcon`**`(8)`, **`restorecon`**`(8)`) to relabel either part or all of the file system.

Note that users home directory file contexts are not present in this file as they are managed by the `file_contexts.homedirs` file as explained below.

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

Supporting `libselinux` API functions are:

```
selinux_file_context_path
```

[42] 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.

3.4.22 contexts/files/file_contexts.local File

This file is added by the semanage fcontext command as described in the ./modules/active/file_contexts.local file section to allow locally defined files to be labeled correctly. The **file_contexts**(5) man page also decribes this file.

Supporting libselinux API functions are:

```
selinux_file_context_local_path
```

3.4.23 contexts/files/file_contexts.homedirs File

This file is managed by the **semodule**(8) and **semanage**(8) commands as the policy is updated (adding or removing users and modules or updating the base), and therefore should not be edited.

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

It is fully described in the ./modules/active/file_contexts.homedirs file section. The **file_contexts**(5) man page also decribes this file.

Supporting libselinux API functions are:

```
selinux_file_context_homedir_path
selinux_homedir_context_path
```

3.4.24 contexts/files/file_contexts.subs & file_contexts.subs_dist File

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

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

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

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

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

Supporting libselinux API functions are:

```
selinux_file_context_subs_path
selinux_file_context_subs_dist_path
selabel_lookup
matchpathcon
matchpathcon_index
```

3.4.25 contexts/files/media File

The **media**(5) file is used to map media types to a file context. If the media_id cannot be found in this file, then the default context in the ./contexts/removable_contexts is used instead.

The file format is as follows:

```
media_id file_context
```

Where:

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 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
```

3.4.26 contexts/users/[seuser_id] File

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

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
system_r:xdm_t:s0           user_r:user_t:s0
user_r:user_su_t:s0         user_r:user_t:s0
user_r:user_sudo_t:s0       user_r:user_t:s0
```

Supporting `libselinux` API functions are:

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

3.4.27 `logins/<linuxuser_id>` File

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

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

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

The file format is as follows:

```
service_name:seuser_id:level
```

Where:

`service_name`	The name of the service.
`seuser_id`	The SELinux user name.
`level`	The run level

Example file contents:

```
# ./logins/ipa example entries

ipa_service:user_u:s0
another_service:unconfined_u:s0
```

Supporting `libselinux` API functions are:

```
getseuser
```

3.4.28 `users/local.users` File

Generally the **`local.users`**`(5)` file is not present if **`semanage`**`(8)` is being used to manage users, however if **`semanage`** is not being used then this file may be present (it could also be present in older Reference or Example policies).

The file would contain local user definitions in the form of `user` statements as defined in the `modules/active/users.local` section.

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

4. SELinux Policy 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 sources[43] (where possible). Also all of the language updates to Policy DB version 28 should have been captured. For a more detailed explanation of the policy language the "SELinux by Example" [Ref. 14] book is recommended.

There is currently a project underway called the Common Intermediate Language (CIL) project that will define a new policy definition language that could eventually replace the current policy language described in this section. Reference to CIL can be found at: http://userspace.selinuxproject.org/trac/wiki/CilDesign, however it is not yet complete although a simple compiler is available by:

```
git clone http://oss.tresys.com/git/cil.git
```

The Notebook tarball contains CIL language examples that will compile as explained in the Notebook Example Policy section.

4.1.1 CIL Overview

While the CIL design web pages give the main objectives of CIL, from a language perspective it will:

a) Apply consistancy to the current language statements. For example the `type_transition` statement has been extended to allow an additional parameter to support object names. With CIL this becomes two separate statements `typetransition` and `nametypetransition`.

Examples:

CIL	Current
`allow`	`allow`
`roleallow`	`allow` (role)
`dominance`	`dominance`
`roledominance`	`dominance` (role)
`roletransition`	`role_transition`

b) Additional CIL statements have been defined to allow clarity, for example:

`typeattributeset`

`classpermissionset`

`classmap`

`classmapping`

[43] 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.

c) Allow named an anonymous context definitions to be defined. This example shows the both context declarations and the supporting statements required to build them:

```
; Declare range info:
(category c0)
(categoryorder (c0))
(sensitivity s0)
(dominance (s0))
(sensitivitycategory s0 (c0))
(levelrange default ((s0 (c0)) (s0 (c0))))
(level low (s0 (c0)))

; Declare type info:
(type  unconfined_t)

; Declare role info
(role  unconfined_r)
(roletype unconfined_r unconfined_t)
(role  object_r)
(roletype object_r unconfined_t)

; Declare user info:
(user  unconfined_u)
(userrole  unconfined_u  unconfined_r)
(userrange unconfined_u (low low))
(userlevel unconfined_u low)

(user  system_u)
(userrole  system_u  unconfined_r)
(userrole  system_u  object_r)
(userrange system_u (low low))
(userlevel system_u low)

; Declare sids:
(sid kernel)
(sid security)
(sid unlabeled)

; Declare context ids (named) and info:
(context default_context (unconfined_u unconfined_r unconfined_t default))
(context system_context (system_u unconfined_r unconfined_t default))
(context object_context (system_u object_r unconfined_t default))

; Use context ids in various statements:
(sidcontext kernel system_context)
(sidcontext security object_context)
(sidcontext unlabeled object_context)

(fsuse xattr ext2 object_context)
(fsuse task eventpollfs object_context)
(fsuse trans mqueue object_context)
(genfscon selinuxfs / object_context)

(filecon "/" "" any object_context)

; Declare this as an anonymous context:
(filecon "/" ".*" any (unconfined_u unconfined_r unconfined_t (s0 (c0)) (s0 (c0))))
```

d) Support namespace features allowing policy modules to be defined within blocks with inheritance and template features. An example with blocks and inheritance:

```
; blockinherit.cil
;
; Need to include the cil-base.cil module when building this example to declare
; the user, role, type and range.
;
; This example defines three nested blocks each declaring a different class
; with two permissions and an allow rule. Three new namespaces are then created
; and the respective levels (1, 2, 3) are then inherited by each new namespace.
;
; Using APOL, sedispol or sesearch 9 allow rules will be found:
```

```
; Three allow rules from the three nested blocks:
; allow level1.level1_t level1.level1_t : level1.file { open execute } ;
; allow level1.level2.level2_t level1.level2.level2_t : level1.level2.socket { recv_msg send_msg } ;
; allow level1.level2.level3.level3_t level1.level2.level3.level3_t : level1.level2.level3.ipc { create destroy } ;

; Three allow rules from the new_namespace1 that inherited the three nested blocks:
; allow new_namespace1.level1_t new_namespace1.level1_t : new_namespace1.file { open execute } ;
; allow new_namespace1.level2.level2_t new_namespace1.level2.level2_t : new_namespace1.level2.socket { recv_msg send_msg }
;
; allow new_namespace1.level2.level3.level3_t new_namespace1.level2.level3.level3_t : new_namespace1.level2.level3.ipc
{ create destroy } ;

; Two allow rules from the new_namespace2 that inherited the two lower nested blocks:
; allow new_namespace2.level2_t new_namespace2.level2_t : new_namespace2.socket { recv_msg send_msg } ;
; allow new_namespace2.level3.level3_t new_namespace2.level3.level3_t : new_namespace2.level3.ipc { create destroy } ;

; One allow rule from the new_namespace3 that inherited the last nested block:
; allow new_namespace3.level3_t new_namespace3.level3_t : new_namespace3.ipc { create destroy } ;

(block level1
    (type level1_t)
    (class file (open execute))
    (allow level1_t self (file (open execute)))

    (block level2
         (type level2_t)
         (class socket (recv_msg send_msg))
         (allow level2_t self (socket (recv_msg send_msg)))

         (block level3
              (type level3_t)
              (class ipc (create destroy))
              (allow level3_t self (ipc (create destroy)))

         ) ; End level3 block
    ) ; End level2 block
) ; End level1 block

(block new_namespace1
    (blockinherit level1)
) ; End new_namespace1 namespace

(block new_namespace2
    (blockinherit level1.level2)
) ; End new_namespace2 namespace

(block new_namespace3
    (blockinherit level1.level2.level3)
) ; End new_namespace3 namespace
```

e) Remove the order dependancy in that policy statements can be anywhere within the source (i.e. remove dependancy of class, sid etc. being within a base module).

f) Able to define macros and calls that will remove any dependancy on M4 macro support. This is a block that calls two different macros:

```
; SEAndroid dbus daemon - dbusd.cil
(block dbusd
     (type dbusd)
     (typeattributeset domain dbusd)

     (type dbusd_exec)
     (typeattributeset exec_type dbusd_exec)
     (typeattributeset file_type dbusd_exec)

     (call init_daemon_domain (dbusd dbusd_exec))
     ; Reads /proc/pid/cmdline of clients
     (call r_dir_file (dbusd system.system))
     (call r_dir_file (dbusd bluetoothd.bluetoothd))
)
```

These are the macros that are called:

```
;
; SEAndroid macros - te_macros.cil
;
; Macros are instantiated within the namespace that has the 'call'
; statement.
; These macros defined in the global namespace:

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
; r_dir_file(domain, type)
; Allow the specified domain to read directories, files
; and symbolic links of the specified type.
(macro r_dir_file ((type type_id1) (type type_id2))
   (allow type_id1 type_id2 (kernel.dir r_dir_perms))
   (allow type_id1 type_id2 (kernel.file r_file_perms))
   (allow type_id1 type_id2 (kernel.lnk_file r_file_perms))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; init_daemon_domain(domain)
; Set up a transition from init to the daemon domain
; upon executing its binary.
(macro init_daemon_domain ((type type_id1) (type type_id2))
    (call domain_auto_trans (init.init type_id2 type_id1))
    (call tmpfs_domain (type_id1))
)
```

g) Directly generate the binary policy file and other configuration files - currently the `file_contexts` file.

h) Support transformation services such as delete, transform and inherit with except.

4.1.2 Notebook Example Policy

The Notebook tarball has a number of simple policies in the `basic-selinux-policy` directory. The sub-directories enable a monolithic and modular policy to be built, also the modular policy can be extended to build a simple message filter (this also has a CIL equivalent version in the CIL directory). The CIL version should build with the compiler from the git repository (although there is a 64 bit binary version with debug enabled included). There is a Python script that will build a simple base policy module in either CIL or the standard policy language using the Reference Policy `security_classes`, `access_vectors` and `initial_sids` files (see `notebook-tools/build-policy.py`).

4.2 Policy Statements and Rules

4.2.1 Policy Source Files

There are three basic types of policy source file[44] that can contain language statements and rules (examples of these can be found in the Building a Basic Policy section of the Notebook source tarball). The three types of policy source file[45] 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`**`(8)` command that produces the binary policy file.

Base Policy – This is the mandatory base policy source file that supports the loadable module infrastructure. The whole system policy could be fully contained

[44] 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.

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

within this file, however it is more usual for the base policy to hold the mandatory components of a policy, with the optional components contained in loadable module source files. By convention this file is called `base.conf` and is compiled using the **`checkpolicy`**`(8)` or **`checkmodule`**`(8)` command.

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

Table 14 shows the order in which the statements should appear in source files with the minimum (and therefore mandatory) statements that must be defined.

Base Entries	***M/O***	***Module Entries***	***M/O***
Security Classes (`class`)	m	`module` Statement	o
Initial SIDs	m		
Access Vectors (permissions)	m	`require` Statement	o
MLS sensitivity, category and level Statements	o		
MLS Constraints	o		
Policy Capability Statements	o		
Attributes	o	Attributes	o
Booleans	o	Booleans	o
Default user, role, type, range rules	o		
Type / Type Alias	m	Type / Type Alias	o
Roles	m	Roles	o
Policy Rules	o	Policy Rules	o
Users	m	Users	o
Constraints	o		
Default SID labeling	m		
`fs_use_xattr` Statements	o		
`fs_use_task` and `fs_use_trans` Statements	o		
`genfscon` Statements	o		
`portcon`, `netifcon` and `nodecon` Statements	o		

Table 14: Base and Module Policy Statements – *A Monolithic source file would contain the same statements as the Base Module. The Mandatory policy entries are noted (the type, role and user require at least one entry each).*

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

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

Where:

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

Table 16 shows a cross reference matrix of statements and rules allowed in each type of policy source file.

4.2.2 Conditional, Optional and Require Statement Rules

The language grammar specifies what statements and rules can be included within Conditional Policy, Optional Policy statements and the `require` statement. To highlight these rules the following table is included in each statement and rule section to show what circumstances each one is valid within a policy source file:

Conditional Policy (`if`) Statement	`optional` Statement	`require` Statement
Yes/No	**Yes/No**	**Yes/No**

Where:

Conditional Policy (`if`) Statement	Whether the statement is allowed within a conditional statement (`IF` / `ELSE` construct) as described in the `if` Statement section. Conditional statements can be in all types of policy source file.
`optional` Statement	Whether the statement is allowed within the `optional { rule_list }` construct as described in the `optional` Statement section.
`require` Statement	Whether the statement keyword is allowed within the `require { rule_list }` construct as described in the `require` Statement section.

Table 16 shows a cross reference matrix of statements and rules allowed in each of the above policy statements.

4.2.3 MLS Statements and Optional MLS Components

The MLS Statements section defines statements specifically for MLS support. However when MLS is enabled, there are other statements that require the MLS Security Context component as an argument, therefore these statements show an example taken from the Reference Policy MLS build.

4.2.4 General Statement Information

1. Identifiers can generally be any length but should be restricted to the following characters: `a-z`, `A-Z`, `0-9` and _ (underscore).
2. A '`#`' indicates the start of a comment in policy source files.
3. 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`**`(8)` or **`checkmodule`**`(8)`) will expand these to individual statements or rules as shown in the following example:

```
# This allow rule has two target entries console_device_t and
# tty_device_t:
allow apm_t { console_device_t tty_device_t }:chr_file
   { getattr read write append ioctl lock };

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

   Therefore when comparing the actual source code with a compiled binary using (for example) **`apol`**`(8)`, **`sedispol`** or **`sedismod`**, the results will differ (however the resulting policy rules will be the same).
5. Some statements can be added to a policy (via the policy store) using the **`semanage`**`(8)` command. Examples of these are shown where applicable, however the **`semanage`** man page should be consulted for all the possible command line options.
6. Table 15 lists words reserved for the SELinux policy language.

alias	allow	and
attribute	attribute_role	auditallow
auditdeny	bool	category
cfalse	class	clone
common	constrain	ctrue
dom	domby	dominance
dontaudit	else	equals
false	filename	filesystem
fscon	fs_use_task	fs_use_trans
fs_use_xattr	genfscon	h1
h2	identifier	if
incomp	inherits	iomemcon

ioportcon	ipv4_addr	ipv6_addr
l1	l2	level
mlsconstrain	mlsvalidatetrans	module
netifcon	neverallow	nodecon
not	notequal	number
object_r	optional	or
path	pcidevicecon	permissive
pirqcon	policycap	portcon
r1	r2	r3
range	range_transition	require
role	roleattribute	roles
role_transition	sameuser	sensitivity
sid	source	t1
t2	t3	target
true	type	typealias
typeattribute	typebounds	type_change
type_member	types	type_transition
u1	u2	u3
user	validatetrans	version_identifier
xor	default_user	default_role
default_type	default_range	low
high	low_high	

Table 15: Policy language reserved words.

7. Table 16 shows what policy language statements and rules are allowed within each type of policy source file, and whether the statement is valid within an `if / else` construct, `optional {rule_list}`, or `require {rule_list}` statement.

Statement / Rule	Monolithic Policy	Base Policy	Module Policy	Conditional Statements	optional Statement	require Statement[46]
allow	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**No**
allow - Role	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**No**
attribute	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**
attribute_role	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**
auditallow	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**No**
auditdeny (Deprecated)	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**No**
bool	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**
category	**Yes**	**Yes**	**No**	**No**	**No**	**Yes**
class	**Yes**	**Yes**	**No**	**No**	**No**	**Yes**
common	**Yes**	**Yes**	**No**	**No**	**No**	**No**
constrain	**Yes**	**Yes**	**No**	**No**	**No**	**No**

[46] Only the statement keyword is allowed.

Statement / Rule	Monolithic Policy	Base Policy	Module Policy	Conditional Statements	optional Statement	require Statement
default_user	Yes	Yes	No	No	No	No
default_role	Yes	Yes	No	No	No	No
default_type	Yes	Yes	No	No	No	No
default_range	Yes	Yes	No	No	No	No
dominance - MLS	Yes	Yes	No	No	No	No
dominance - Role (Deprecated)	Yes	Yes	Yes	No	Yes	No
dontaudit	Yes	Yes	Yes	Yes	Yes	No
fs_use_task	Yes	Yes	No	No	No	No
fs_use_trans	Yes	Yes	No	No	No	No
fs_use_xattr	Yes	Yes	No	No	No	No
genfscon	Yes	Yes	No	No	No	No
if	Yes	Yes	Yes	No	Yes	No
level	Yes	Yes	No	No	No	No
mlsconstrain	Yes	Yes	No	No	No	No
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[47]	No	Yes	No
nodecon	Yes	Yes	No	No	No	No
optional	No	Yes	Yes	Yes	Yes	Yes
permissive	Yes	Yes	Yes	Yes	Yes	No
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[48]	Yes	Yes[49]	Yes	No
role	Yes	Yes	Yes	No	Yes	Yes
roleattribute	Yes	Yes	Yes	No	Yes	No
role_transition	Yes	Yes	Yes	No	Yes	No
sensitivity	Yes	Yes	No	No	No	Yes
sid	Yes	Yes	No	No	No	No
type	Yes	Yes	Yes	No	No	Yes
type_change	Yes	Yes	Yes	Yes	Yes	No
type_member	Yes	Yes	Yes	Yes	Yes	No
type_transition	Yes	Yes	Yes	Yes[50]	Yes	No
typealias	Yes	Yes	Yes	No	Yes	No
typeattribute	Yes	Yes	Yes	No	Yes	No

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

[48] Only if preceded by the `optional` statement.

[49] Only if preceded by the `optional` statement.

[50] Excluding the 'file name transition' rule.

Statement / Rule	Monolithic Policy	Base Policy	Module Policy	Conditional Statements	optional Statement	require Statement
typebounds	Yes	Yes	Yes	No	Yes	No
user	Yes	Yes	Yes	No	Yes	Yes
validatetrans	Yes	Yes	No	No	No	No

Table 16: The policy language statements and rules that are allowed within each type of policy source file - *The left hand side of the table shows what Policy Language Statements and Rules are allowed within each type of policy source file. The right hand side of the table shows whether the statement is valid within the* `if / else` *construct,* `optional {rule_list}`*, or* `require {rule_list}` *statement.*

4.2.5 Section Contents

The policy language statement and rule sections are as follows:

a) Type and Attribute Statements
b) Default Rules
c) Type Enforcement Rules
d) Access Vector Rules
e) User Statement
f) Role Statement
g) Role Rules
h) Conditional Policy Statements
i) Constraint Statements
j) File System Labeling Statements
k) Network Labeling Statements
l) MLS Statements
m) Policy Support Statements
n) Object Class and Permission Statements
o) Security ID (SID) Statement

4.3 Type 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:

```
# Statement Identifier  Comment
#-----------------------------------------
type        bin_t;      # A type identifier ends with _t
attribute   file_type;  # An attribute identifier ends with
                        # anything else
```

4.3.1 `type` Statement

The `type` statement declares the `type` identifier and any optional associated `alias` or `attribute` identifiers. Type identifiers are a component of the Security Context.

The statement definition is:

```
type type_id;
```

Or

```
type type_id, attribute_id;
```

Or

```
type type_id alias alias_id;
```

Or

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

Where:

`type`	The `type` keyword.
`type_id`	The `type` identifier.
`alias`	Optional `alias` keyword that signifies alternate identifiers for the `type_id` that are declared in the `alias_id` list.
`alias_id`	One or more `alias` identifiers that have been previously declared by the `typealias` Statement. Multiple entries consist of a space separated list enclosed in braces (`{}`).
`attribute_id`	One or more optional `attribute` identifiers that have been previously declared by the `attribute` Statement. Multiple entries consist of a comma (`,`) separated list, also note the lead comma.

The statement is valid in:

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

Examples:

```
# Using the type statement to declare a type of shell_exec_t,
# where exec_t is used to identify a file as an executable type.

type shell_exec_t;
```

```
# Using the type statement to declare a type of bin_t, where
# bin_t is used to identify a file as an ordinary program type.

type bin_t;
```

```
# Using the type statement to declare a type of bin_t with two
# alias names. The sbin_t is used to identify the file as a
# system admin program type.

type bin_t alias { ls_exec_t sbin_t };
```

```
# Using the type statement to declare a type of boolean_t that
# also associates it to a previously declared attribute
# booleans_type (see the attribute Statement).

attribute booleans_type;      # declare the attribute

type boolean_t, booleans_type;   # and associate with the type
```

```
# Using the type statement to declare a type of setfiles_t that
# also has an alias of restorecon_t and one previously declared
# attribute of can_relabelto_binary_policy associated with it.

attribute can_relabelto_binary_policy;

type setfiles_t alias restorecon_t, can_relabelto_binary_policy;
```

```
# Using the type statement to declare a type of
# ssh_server_packet_t that also associates it to two previously
# declared attributes packet_type and server_packet_type.

attribute packet_type;     # declare attribute 1
attribute server_packet_type;# declare attribute 2

# Associate the type identifier with the two attributes:

type ssh_server_packet_t, packet_type, server_packet_type;
```

4.3.2 `attribute` Statement

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

The statement definition is:

```
attribute attribute_id;
```

Where:

attribute	The attribute keyword.
attribute_id	The attribute identifier.

The statement is valid in:

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

Examples:

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

attribute domain;
attribute daemon;
attribute file_type;
attribute non_security_file_type;
```

4.3.3 typeattribute Statement

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

The statement definition is:

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

Where:

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

The statement is valid in:

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

Examples:

```
# Using the typeattribute statement to associate a previously
```

```
# declared type of setroubleshootd_t to a previously declared
# domain attribute.

# The previously declared attribute:
attribute domain;

# The previously declared type:
type setroubleshootd_t;

# The association using the typeattribute statement:
typeattribute setroubleshootd_t domain;
```

```
# Using the typeattribute statement to associate a type of
# setroubleshootd_exec_t to two attributes file_type and
# non_security_file_type.

# These are the previously declared attributes:
attribute file_type;
attribute non_security_file_type;

# The previously declared type:
type setroubleshootd_exec_t;

# These are the associations using the typeattribute statement:
typeattribute setroubleshootd_exec_t file_type, non_security_file_type;
```

4.3.4 `typealias` Statement

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

The statement definition is:

```
typealias type_id alias alias_id;
```

Where:

`typealias`	The `typealias` keyword.
`type_id`	The identifier of a previously declared `type`.
`alias`	The `alias` keyword.
`alias_id`	One or more `alias` identifiers. Multiple entries consist of a space separated list enclosed in braces (`{}`).

The statement is valid in:

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

Examples:

```
# Using the typealias statement to associate the previously
# declared type mount_t with an alias of mount_ntfs_t.

# Declare the type:
type mount_t;

# Then alias the identifier:
typealias mount_t alias mount_ntfs_t;
```

```
# Using the typealias statement to associate the previously
# declared type netif_t with two alias, lo_netif_t and
# netif_lo_t.

# Declare the type:
type netif_t;

# Then assign two alias identifiers lo_netif_t and netif_lo_t:
typealias netif_t alias { lo_netif_t netif_lo_t };
```

4.4 Default Rules

These rules allow a default user, role, type and/or range to be used when computing a context for a new object. These require policy version 27 or 28 with kernels 3.5 or greater.

4.4.1 `default_user` Rule

Allows the default user to be taken from the source or target context when computing a new context for an object of the defined class. Requires policy version 27.

The statement definition is:

```
default_user class default;
```

Where:

`default_user`	The `default_user` rule keyword.
`class`	One or more `class` identifiers. Multiple entries consist of a space separated list enclosed in braces (`{ }`). Entries can be excluded from the list by using the negative operator (`-`).
`default`	A single keyword consisting of either `source` or `target` that will state whether the default user should be obtained from the source or target context.

The statement is valid in:

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

Example:

```
# When computing the context for a new file object, the user
# will be obtained from the target context.
default_user file target;

# When computing the context for a new x_selection or x_property
# object, the user will be obtained from the source context.
default_user { x_selection x_property } source;
```

4.4.2 `default_role` Rule

Allows the default role to be taken from the source or target context when computing a new context for an object of the defined class. Requires policy version 27.

The statement definition is:

```
default_role class default;
```

Where:

`default_role`	The `default_role` rule keyword.
`class`	One or more `class` identifiers. Multiple entries consist of a space separated list enclosed in braces (`{ }`). Entries can be excluded from the list by using the negative operator (`-`).
`default`	A single keyword consisting of either `source` or `target` that will state whether the default role should be obtained from the source or target context.

The statement is valid in:

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

Example:

```
# When computing the context for a new file object, the role
# will be obtained from the target context.
default_role file target;

# When computing the context for a new x_selection or x_property
# object, the role will be obtained from the source context.
default_role { x_selection x_property } source;
```

4.4.3 `default_type` Rule

Allows the default type to be taken from the source or target context when computing a new context for an object of the defined class. Requires policy version 28.

The statement definition is:

```
default_type class default;
```

Where:

`default_type`	The `default_type` rule keyword.
`class`	One or more `class` identifiers. Multiple entries consist of a space separated list enclosed in braces (`{ }`). Entries can be excluded from the list by using the negative operator (`-`).
`default`	A single keyword consisting of either `source` or `target` that will state whether the default type should be obtained from the source or target context.

The statement is valid in:

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

Example:

```
# When computing the context for a new file object, the type
# will be obtained from the target context.
default_type file target;

# When computing the context for a new x_selection or x_property
# object, the type will be obtained from the source context.
default_type { x_selection x_property } source;
```

4.4.4 `default_range` Rule

Allows the default range or level to be taken from the source or target context when computing a new context for an object of the defined class. Requires policy version 27.

The statement definition is:

```
default_range class default entry;
```

Where:

`default_range`	The `default_range` rule keyword.
`class`	One or more `class` identifiers. Multiple entries consist of a space separated list enclosed in braces (`{ }`). Entries can be excluded from the list by using the negative operator (`-`).
`default`	A single keyword consisting of either `source` or `target` that will state whether the default level or range should be obtained from the source or target context.
`entry`	A single keyword consisting of either: `low`, `high` or `low_high` that will state whether the default level or range should be obtained from the source or target context.

The statement is valid in:

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

Example:

```
# When computing the context for a new file object, the lower
# level will be taken from the target context range.
default_range file target low;

# When computing the context for a new x_selection or x_property
# object, the range will be obtained from the source context.
default_type { x_selection x_property } source low_high;
```

4.5 Type Enforcement Rules

There are three types of enforcement rule: `type_transition`, `type_change`, and `type_member` that are explained below.

Important note: type enforcement rules only specify the rule and labeling required, it is the `allow` rules that will finally determine if the enforcement rule is actually allowed (or not).

4.5.1 `type_transition` Rule

The `type_transition` rule specifies the labeling and object creation allowed between the `source_type` and `target_type` when a Domain Transition is requested. Kernels from 2.6.39 with Policy versions from 25 also support the 'name transition rule' extension.

The statement definition is:

```
type_transition source_type target_type : class default_type;
```

Policy versions 25 and above also support a 'name transition' rule, however, this is not allowed inside conditionals and currently only supports the `file` class:

```
type_transition source_type target_type : class default_type object_name;
```

Where:

`type_transition`	The `type_transition` 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 (but see `object_name`).
`object_name`	For the 'name transition' rule this is matched against the objects name (i.e. the last component of a path). If `object_name` exactly matches the object name, then use `default_type` for the type.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	**Yes**	**Yes**
Conditional Policy (`if`) Statement	`optional` Statement	`require` Statement
Yes (except the 'file name transition' rule)	**Yes**	**No**

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
# allow Rule section).

# A minimum of: add_name, write and search on the var_log_t
# directory. The actual policy has:
#
allow acct_t var_log_t:dir { read getattr lock search ioctl
   add_name remove_name write };

# A minimum of: create and write on the wtmp_t file. The actual
# policy has:
#
allow acct_t wtmp_t:file { create open getattr setattr read
   write append rename link unlink ioctl lock };
```

Example – Name Transition:

```
# type_transition to allow using the last path component as
# part of the information in making labeling decisions for
# new objects. An example rule:
#
type_transition unconfined_t etc_t : file system_conf_t eric;

# This rule says if unconfined_t creates a file in a directory
# labeled etc_t and the last path component is "eric" (must be
# an exact strcmp) it should be labeled system_conf_t.
# Note: This rule in not supported in conditionals.
```

4.5.2 type_change Rule

The type_change rule is used to define a different label of an object for userspace SELinux-aware applications. These applications would use **security_compute_relabel**(3) and type_change rules in the policy to determine the new context to be applied.

The statement definition is:

```
type_change source_type target_type : class change_type;
```

Where:

type_change	The type_change 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 ({}).

change_type	A single type identifier that will become the new type.

The statement is valid in:

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

Examples:

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

type_change auditadm_t sysadm_devpts_t:chr_file auditadm_devpts_t;
```

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

type_change staff_t server_ptynode:chr_file staff_devpts_t;
```

4.5.3 type_member Rule

The type_member rule is used to define a new polyinstantiated label of an object for SELinux-aware applications. These applications would use **avc_compute_member**(3) or **security_compute_member**(3) with the type_member rules in the policy to determine the context to be applied. The application would then manage any required polyinstantiation.

The statement definition is:

```
member_type source_type target_type : class member_type;
```

Where:

type_member	The 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 ({}).

member_type	A single type identifier that will become the polyinstantiated type.

The statement is valid in:

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

Example:

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

type_member sysadm_t user_home_dir_t:dir user_home_dir_t;
```

4.6 bounds Statements

4.6.1 typebounds Rule

The typebounds rule was added in version 24 of the policy. This defines a hierarchical relationship between domains where the bounded domain cannot have more permissions than its bounding domain (the parent). It requires kernel 2.6.28 and above to control the security context associated to threads in multi-threaded applications.

The statement definition is:

```
typebounds bounding_domain bounded_domain, [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**	**require Statement**
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 not be allowed by the kernel SELinux security server.
allow  httpd_child_t etc_t : file { read write };
```

4.7 Access Vector Rules

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

The general format of an AV rule is that the `source_type` is the identifier of a process that is attempting to access an object identifier `target_type`, that has an object class of `class`, and `perm_set` defines the access permissions `source_type` is allowed.

The common format of the Access Vector Rule is:

```
rule_name source_type target_type : class perm_set;
```

Where:

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

The statements are valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	**Yes**	**Yes**
Conditional Policy (`if`) Statement	**`optional` Statement**	**`require` Statement**
allow = Yes **auditallow = Yes** **dontaudit = Yes** **neverallow = No**	**allow = Yes** **auditallow = Yes** **dontaudit = Yes** **neverallow = Yes**	**allow = No** **auditallow = No** **dontaudit = No** **neverallow = No**

4.7.1 `allow` Rule

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

Examples:

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

allow initrc_t acct_exec_t:file { getattr read execute };
```

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

allow kernel_t filesystem_type:filesystem mount;
```

```
# This rule includes the self keyword in the target_type that
```

```
# states that staff_t is allowed setgid, chown and fowner
# permissions on the capability object:

allow staff_t self:capability { setgid chown fowner };

# This would be the same as the above:
allow staff_t staff_t:capability { setgid chown fowner };
```

```
# This rule includes the wildcard operator (*) on the perm_set
# and states that bootloader_t is allowed to use all permissions
# available on the dbus object that are type system_dbusd_t:

allow bootloader_t system_dbusd_t:dbus *;

# This would be the same as the above:
allow bootloader_t system_dbusd_t:dbus { acquire_svc send_msg };
```

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

allow files_unconfined_type file_type:{ file chr_file }
~execmod;
```

4.7.2 dontaudit Rule

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

Example:

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

dontaudit traceroute_t { port_type -port_t }:tcp_socket
name_bind;
```

4.7.3 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;
```

4.7.4 neverallow Rule

This rule specifies that an allow Rule must not be generated for the operation, even if it has been previously allowed. The neverallow statement is a compiler enforced action, where the checkpolicy or checkmodule[51] 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;
```

4.8 User Statement

4.8.1 user Statement

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

The statement definition is:

```
user seuser_id roles role_id;
```

Or for MCS/MLS Policy:

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

Where:

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

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 level Statement. 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 MLS range Definition section.

The statement is valid in:

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

Example:

```
# Using the user statement to define an SELinux user user_u that
# has been assigned the role of user_r. The SELinux user_u is a
# generic user identity for Linux users who have no specific
# SELinux user identity defined.
#
user user_u roles { user_r };
```

MLS Examples:

```
# Using the user statement to define an MLS SELinux user user_u
# that has been assigned the role of user_r and has a default
# login security level of s0 assigned, and is only allowed
# access to the s0 range of security levels (See the
# MLS Statements section for details):

user user_u roles { user_r } level s0 range s0;
```

```
# Using the user statement to define an MLS SELinux user
# sysadm_u that has been assigned the role of sysadm_r and has
# a default login security level of s0 assigned, and is
```

```
# allowed access to the range of security levels (low - high)
# between s0 and s15:c0.c255 (See the MLS Statements section
# for details):

user sysadm_u roles { sysadm_r } level s0 range s0-s15:c0.c255;
```

`semanage(8)` Command example:

```
# Add user mque_u to SELinux and associate to the unconfined_r
# role:
semanage user -a -R unconfined_r mque_u
```

This command will produce the following files in the default <policy_name> policy store and then activate the policy:

/etc/selinux/<policy_name>/modules/active/users.local:

```
# This file is auto-generated by libsemanage
# Do not edit directly.

user mque_u roles { unconfined_r } ;
```

/etc/selinux/<policy_name>/modules/active/users_extra:

```
# This file is auto-generated by libsemanage
# Do not edit directly.

user mque_u prefix user;
```

/etc/selinux/<policy_name>/modules/active/users_extra.local:

```
# This file is auto-generated by libsemanage
# Do not edit directly.

user mque_u prefix user;
```

4.9 Role Statements

Policy version 26 introduced two new role statements aimed at replacing the role `dominance` rule by making role relationships easier to understand. These new statements: `attribute_role` and `roleattribute`, are similar in operation to the `attribute` and `typeattribute` statements used for types and are defined in this section with examples.

4.9.1 `role` Statement

The `role` statement either declares a role identifier or associates a role identifier to one or more `types` (i.e. authorise the role to access the domain or domains). Where there are multiple `role` statements declaring the same role, the compiler will associate the additional `types` with the `role`.

The statement definition to declare a role is:

```
role role_id;
```

The statement definition to associate a role to one or more types is:

```
role role_id types type_id;
```

Where:

`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.

The statement is valid in:

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

Examples:

```
# Using the role statement to define standard roles in the
# Reference Policy.

role system_r;
role sysadm_r;
role staff_r;
role user_r;
role secadm_r;
role auditadm_r;

# Within the policy the roles are then associated to the
# required types with this example showing the user_r role
# being associated to two domains:

role user_r types user_t;
role user_r types chfn_t;
```

4.9.2 `attribute_role` Statement

The `attribute_role` statement declares a role attribute identifier that can then be used to refer to a group of `roles`.

The statement definition is:

```
attribute_role attribute_id;
```

Where:

`attribute_role`	The `attribute_role` keyword.
`attribute_id`	The attribute identifier.

The statement is valid in:

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

Examples:

```
# Using the attribute_role statement to declare attributes that
# can then refers to a list of roles. Note that there are no
# roles associated with them yet.

attribute_role role_list_1;
attribute_role srole_list_2;
```

4.9.3 `roleattribute` Statement

The `roleattribute` statement allows the association of previously declared `roles` to one or more previously declared `attribute_roles`.

The statement definition is:

```
roleattribute role_id attribute_id [ ,attribute_id ];
```

Where:

roleattribute	The roleattribute keyword.
role_id	The identifier of a previously declared role.
attribute_id	One or more previously declared attribute_role identifiers. Multiple entries consist of a comma (,) separated list.

The statement is valid in:

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

Examples:

```
# Using the roleattribute statement to associate a previously
# declared role of service_r to a previously declared
# role_list_1 attribute_role.

attribute_role role_list_1;
role service_r;

# The association using the roleattribute statement:
roleattribute service_r  role_list_1;
```

4.10 Role Rules

4.10.1 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.

Note that the role allow rule has the same keyword as the allow AV rule.

The statement definition is:

```
allow from_role_id to_role_id;
```

Where:

allow	The role allow rule keyword.
from_role_id	One or more role identifiers that identify the current role. Multiple entries consist of a space separated list enclosed in braces ({}).
to_role_id	One or more role identifiers that identify the new role to be granted on the transition. Multiple entries consist of a space separated list enclosed

	in braces ({}).

The statement is valid in:

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

Example:

```
# Using the role allow rule to define authorised role
# transitions in the Reference Policy. The current role
# sysadm_r is granted permission to transition to the secadm_r
# role in the MLS policy.

allow sysadm_r secadm_r;
```

4.10.2 `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. From policy version 25, the class can now be defined.

The statement definition is:

```
role_transition current_role_id type_id new_role_id;
```

Or from Policy version 25:

```
role_transition current_role_id type_id : class new_role_id;
```

Where:

`role_transition`	The `role_transition` keyword.
`current_role_id`	One or more `role` 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 (-).
`class`	For policy versions > 24, an object class that applies to the role transition. If omitted, defaults to the process object class as used by policy versions =< 24.
`new_role_id`	The new role to be granted on transition.

The statement is valid in:

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

Example:

```
# This is a role_transition used in the ext_gateway.conf
# loadable module to allow the secure client / server process to
# run under the message_filter_r role. The role needs to be
# declared, allowed to transition from its current role of
# unconfined_r and it then transitions when the process
# transitions via the type_transition statement (not shown).
# Note that the role needs to be associated to a user by either:
# 1) An embedded user statement in the policy. This is not
#    recommended as it makes the policy fixed to either
#    standard, MCS or MLS.
# 2) Using the semanage(8) command to add the role. This will
#    allow the module to be used by MCS/MLS policies as well.
#

# The secure client / server will run in this domain:
type ext_gateway_t;
# The binaries will be labeled:
type secure_services_exec_t;

# Use message_filter_r role and then transition
role message_filter_r types ext_gatway_t;
allow unconfined_r message_filter_r;
role_transition unconfined_r secure_services_exec_t message_filter_r;
```

4.10.3 Role `dominance` Rule

This rule has been deprecated and therefore should not be used. The role `dominance` rule allows the `dom_role_id` to dominate the `role_id` (consisting of one or more roles). The dominant role will automatically inherit all the `type` associations of the other roles.

Notes:

1. There is another `dominance` rule for MLS (see the MLS `dominance` Statement).
2. The role `dominance` rule is not used by the Reference Policy as the policy manages role dominance using the `constrain` Statement.
3. Note the usage of braces '`{}`' and the '`;`' in the statement.

The statement definition is:

```
dominance { role dom_role_id { role role_id; } }
```

Where:

dominance	The dominance keyword.
role	The role keyword.
dom_role_id	The dominant role identifier.
role_id	For the simple case each { role role_id; } pair defines the role_id that will be dominated by the dom_role_id. More complex rules can be defined the statement is deprecated.

The statement is valid in:

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

Example:

```
# This shows the dominance role rule, note however that it
# has been deprecated and should not be used.

dominance { role message_filter_r { role unconfined_r };}
```

4.11 Conditional Policy Statements

Conditional policies consist of a bool statement that defines a condition as true or false, with a supporting if / else construct that specifies what rules are valid under the condition as shown in the example below:

```
bool allow_daemons_use_tty true;

  if (allow_daemons_use_tty) {
     # Rules if condition is true;

  } else {
        # Rules if condition is false;
  }
```

Table 16 shows what policy statements or rules are valid within the if / else construct under the "Conditional Statements" column.

The bool statement default value can be changed when a policy is active by using the setsebool command as follows:

```
# This command will set the allow_daemons_use_tty bool to false,
# however it will only remain false until the next system
# re-boot where it will then revert back to its default state
# (in the above case, this would be true).

setsebool allow_daemons_use_tty false
```

```
# This command will set the allow_daemons_use_tty bool to false,
# and because the -P option is used (for persistent), the value
# will remain across system re-boots. Note however that all
# other pending bool values will become persistent across
# re-boots as well (see 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
```

4.11.1 `bool` Statement

The `bool` statement is used to specify a boolean identifier and its initial state (`true` or `false`) that can then be used with the `if` Statement to form a 'conditional policy' as described in the Conditional Policy section.

The statement definition is:

```
bool bool_id default_value;
```

Where:

`bool`	The `bool` keyword.
`bool_id`	The boolean identifier.
`default_value`	Either `true` or `false`.

The statement is valid in:

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

Examples:

```
# Using the bool statement to allow unconfined executables to
# make their memory heap executable or not. As the value is
# false, then by default they cannot make their heap executable.

bool allow_execheap false;
```

```
# Using the bool statement to allow unconfined executables to
# make their stack executable or not. As the value is true,
# then by default their stacks are executable.

bool allow_execstack true;
```

4.11.2 `if` Statement

The `if` statement is used to form a 'conditional block' of statements and rules that are enforced depending on whether one or more boolean identifiers (defined by the `bool` Statement) evaluate to `TRUE` or `FALSE`. An `if / else` construct is also supported.

The only statements and rules allowed within the `if / else` construct are:

`allow`, `auditallow`, `auditdeny`, `dontaudit`, `type_member`, `type_transition` (except 'file_name_transition'), `type_change` and `require`.

The statement definition is:

```
if (conditional_expression) { true_list } [ else
{ false_list } ]
```

Where:

`if`	The `if` keyword.
`conditional_expression`	One or more `bool_name` identifiers that have been previously defined by the `bool` Statement. Multiple identifiers must be separated by the following logical operators: `&&`, `||`, `^`, `!`, `==`, `!=`. The `conditional_expression` is enclosed in brackets `()`.
`true_list`	A list of rules enclosed within braces '`{}`' that will be executed when the `conditional_expression` is '`true`'. Valid statements and rules are highlighted 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.

The statement is valid in:

Monolithic Policy	Base Policy	Module Policy
Yes	**Yes**	**Yes**
Conditional Policy (`if`) Statement	`optional` Statement	`require` Statement
No – As this is a Conditional Statement and cannot be nested.	**Yes**	**No**

Examples:

```
# An example showing a boolean and supporting if statement.

bool allow_execmem false;

# The bool allow_execmem is FALSE therefore the allow statement
# is not executed:

if (allow_execmem) {
      allow sysadm_t self:process execmem;
}
```

```
# An example showing two booleans and a supporting if statement.

bool allow_execmem false;
bool allow_execstack true;

# The bool allow_execmem is FALSE and allow_execstack is TRUE
# therefore the allow statement is not executed:

if (allow_execmem && allow_execstack) {
      allow sysadm_t self:process execstack;
}
```

```
# An example of an IF - ELSE statement where the bool statement
# is FALSE, therefore the ELSE statements will be executed.
#
bool read_untrusted_content false;

if (read_untrusted_content) {
      allow sysadm_t { sysadm_untrusted_content_t
       sysadm_untrusted_content_tmp_t }:dir { getattr search
read lock ioctl };
      .....

} else {
      dontaudit sysadm_t { sysadm_untrusted_content_t
       sysadm_untrusted_content_tmp_t }:dir { getattr search
read lock ioctl };
      ...
}
```

4.12 Constraint Statements

4.12.1 `constrain` Statement

The `constrain` statement allows further restriction on permissions for the specified object classes by using boolean expressions covering: source and target types, roles and users as described in the examples.

The statement definition is:

```
constrain class perm_set expression;
```

Where:

`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:

	`( expression : expression )`
	`| not expression`
	`| expression and expression`
	`| expression or expression`
	`| u1 op u2`
	`| r1 role_op r2`
	`| t1 op t2`
	`| u1 op names`
	`| u2 op names`
	`| r1 op names`
	`| r2 op names`
	`| t1 op names`
	`| t2 op names`

	Where:
	`u1, r1, t1 = Source user, role, type`
	`u2, r2, t2 = Target user, role, type`
	and:
	`op : == | !=`
	`role_op : == | != | eq | dom | domby | incomp`
	`names : name | { name_list }`
	`name_list : name | name_list name`

The statement is valid in:

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

Examples:

These examples have been taken from the Reference Policy source `./policy/constraints` file.

```
# This constrain statement is the "SELinux process identity
# change constraint" taken from the Reference Policy source and
# contains multiple expressions.
#
# The overall constraint is on the process object class with the
# transition permission, and is stating that a domain transition
# is being constrained by the rules listed (u1 == u2 etc.),
# however only the first two expressions are explained.
#
# The first expression u1 == u2 states that the source (u1) and
# target (u2) user identifiers must be equal for a process
# transition to be allowed.
#
# However note that there are a number of or operators that can
# override this first constraint.
#
# The second expression:
#  ( t1 == can_change_process_identity and t2 == process_user_target )
#
# states that if the source type (t1) is equal to any type
# associated to the can_change_process_identity attribute, and
# the target type (t2) is equal to any type associated to the
# process_user_target attribute, then a process transition is
# allowed.

# What this expression means in the 'standard' build Reference
# Policy is that if the source domain is either cron_t,
# firstboot_t, local_login_t, su_login_t, sshd_t or xdm_t (as
# the can_change_process_identity attribute has these types
# associated to it) and the target domain is sysadm_t (as that
# is the only type associated to the can_change_process_identity
# attribute), then a domain transition is allowed.
#
# SELinux process identity change constraint:
constrain process transition (
   u1 == u2
or
   ( t1 == can_change_process_identity and t2 == process_user_target )
or
   ( t1 == cron_source_domain and ( t2 == cron_job_domain or u2 == system_u ))
or
   ( t1 == can_system_change and u2 == system_u )
or
   ( t1 == process_uncond_exempt ) );
```

```
# This constrain statement is the "SELinux file related object
```

```
# identity change constraint" taken from the Reference Policy
# source and contains two expressions.
#
# The overall constraint is on the listed file related object
# classes (dir, file etc.), covering the create, relabelto, and
# relabelfrom permissions. It is stating that when any of the
# object class listed are being created or relabeled, then they
# are subject to the constraint rules listed (u1 == u2 etc.).
#
# The first expression u1 == u2 states that the source (u1) and
# target (u2) user identifiers (within the security context)
# must be equal when creating or relabeling any of the file
# related objects listed.
#
# The second expression:
#  or t1 == can_change_object_identity
#
# states or if the source type (t1) is equal to any type
# associated to the can_change_object_identity attribute, then
# any of the object class listed can be created or relabeled.
#
# What this expression means in the 'standard' build
# Reference Policy is that if the source domain (t1) matches a
# type entry in the can_change_object_identity attribute, then
# any of the object class listed can be created or relabeled.
#
# SELinux file related object identity change constraint:
constrain { dir file lnk_file sock_file fifo_file chr_file
            blk_file } { create relabelto relabelfrom }
(
  u1 == u2
  or t1 == can_change_object_identity
);
```

4.12.2 `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:

```
( expression : expression )
   | not expression
   | expression and expression
   | expression or expression
   | u1 op u2
   | r1 role_op r2
   | t1 op t2
   | u1 op names
   | u2 op names
   | r1 op names
   | r2 op names
   | t1 op names
   | t2 op names
   | u3 op names
   | r3 op names
   | t3 op names
```

Where:

```
u1, r1, t1 = Old user, role, type
u2, r2, t2 = New user, role, type
u3, r3, t3 = Process user, role, type
```

and:

```
op : == | !=
role_op : == | != | eq | dom | domby | incomp
names : name | { name_list }
name_list : name | name_list name
```

The statement is valid in:

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

Examples:

4.13 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.

4.13.1 `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**	**`require` Statement**
No	**No**	**No**

Example:

```
# These statements define file systems that support extended
# attributes (security.selinux).

fs_use_xattr encfs system_u:object_r:fs_t;
fs_use_xattr ext2 system_u:object_r:fs_t;
fs_use_xattr ext3 system_u:object_r:fs_t;
```

MLS Examples:

```
# These statements define file systems that support extended
# attributes (security.selinux).

fs_use_xattr encfs system_u:object_r:fs_t:s0;
fs_use_xattr ext2 system_u:object_r:fs_t:s0;
fs_use_xattr ext3 system_u:object_r:fs_t:s0;
```

4.13.2 `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**	**`require` Statement**
No	**No**	**No**

Example:

```
# These statements define the file systems that support pseudo
# filesystems that represent objects like pipes and sockets, so
# that these objects are labeled with the same type as the
# creating task.
#
fs_use_task eventpollfs system_u:object_r:fs_t;
fs_use_task pipefs system_u:object_r:fs_t;
fs_use_task sockfs system_u:object_r:fs_t;
```

MLS Example:

```
# These statements define the file systems that support pseudo
# filesystems that represent objects like pipes and sockets, so
# that these objects are labeled with the same type as the
# creating task.
#
fs_use_task eventpollfs system_u:object_r:fs_t:s0;
fs_use_task pipefs system_u:object_r:fs_t:s0;
fs_use_task sockfs system_u:object_r:fs_t:s0;
```

4.13.3 `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 shm system_u:object_r:tmpfs_t:s0;
fs_use_trans tmpfs system_u:object_r:tmpfs_t:s0;
fs_use_trans devpts system_u:object_r:devpts_t:s0;
```

4.13.4 `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 iso9660 / system_u:object_r:iso9660_t
genfscon usbfs / system_u:object_r:usbfs_t
genfscon selinuxfs / system_u:object_r:security_t

# The following show some example /proc entries that can have
# directories added to the path.

genfscon proc / system_u:object_r:proc_t
genfscon proc /sysvipc system_u:object_r:proc_t
genfscon proc /fs/openafs system_u:object_r:proc_afs_t
genfscon proc /kmsg system_u:object_r:proc_kmsg_t
```

MLS Examples:

```
# The following examples show those filesystems that only
# support a single security context across the filesystem
# with the MLS levels added.

genfscon msdos / system_u:object_r:dosfs_t:s0
genfscon iso9660 / system_u:object_r:iso9660_t:s0
genfscon usbfs / system_u:object_r:usbfs_t:s0
genfscon selinuxfs / system_u:object_r:security_t:s0

# The following show some example /proc entries. Note that the
# /kmsg has the highest sensitivity level assigned (s15) because
# it is a trusted process.

genfscon proc / system_u:object_r:proc_t:s0
genfscon proc /sysvipc system_u:object_r:proc_t:s0
```

```
genfscon proc /fs/openafs system_u:object_r:proc_afs_t:s0
genfscon proc /kmsg system_u:object_r:proc_kmsg_t:s15:c0.c255
```

4.14 Network Labeling Statements

The network labeling statements are used to label the following objects:

Network interfaces – This covers those interfaces managed by the **`ifconfig`**`(8)` command.

Network nodes – These are generally used to specify host systems using either IPv4 or IPv6 addresses.

Network ports – These can be either `udp` or `tcp` port numbers.

A security context is defined by these network labeling statements, therefore if the policy supports MCS / MLS, then an `mls_range` is required as described in the MLS range Definition section. Note that there are no terminating semi-colons (`;`) on these statements.

If any of the network objects do not have a specific security context assigned by the policy, then the value given in the policies initial SID is used (`netif`, `node` or `port` respectively), as shown below:

```
# Network Initial SIDs from the Standard 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
```

4.14.1 IP Address Formats

4.14.1.1 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
```

4.14.1.2 IPv6 Address Formats

IPv6 addresses are written as eight groups of four hexadecimal digits, where each group is separated by a colon (`:`) as follows:

```
2001:0db8:85a3:0000:0000:8a2e:0370:7334
```

To shorten the writing and presentation of addresses, the following rules apply:

a) Any leading zeros in a group may be replaced with a single '`0`' as shown:

```
2001:db8:85a3:0:0:8a2e:370:7334
```

b) Any leading zeros in a group may be omitted and be replaced with two colons (`::`), however this is only allowed once in an address as follows:

```
2001:db8:85a3::8a2e:370:7334
```

c) The localhost (loopback) address can be written as:

```
0000:0000:0000:0000:0000:0000:0000:0001
```

Or

```
::1
```

d) An undetermined IPv6 address i.e. all bits are zero is written as:

```
::
```

4.14.2 `netifcon` Statement

The `netifcon` statement is used to label network interface objects (e.g. `eth0`).

It is also possible to use the '`semanage interface`' command to associate the interface to a security context.

The statement definition is:

```
netifcon netif_id netif_context packet_context
```

Where:

`netifcon`	The `netifcon` keyword.
`netif_id`	The network interface name (e.g. `eth0`).
`netif_context`	The security context allocated to the network interface.
`packet_context`	The security context allocated packets. Note that these are defined but currently unused. The iptable SECMARK services should be used to label packets.

The statement is valid in:

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

Examples:

```
# The following netifcon statement has been taken from the
# MLS policy that shows an interface name of lo with the same
# security context assigned to both the interface and packets.

netifcon lo system_u:object_r:lo_netif_t:s0 - s15:c0.c255
   system_u:object_r:unlabeled_t:s0 - s15:c0.c255
```

`semanage(8)` Command example:

```
semanage interface -a -t 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
```

4.14.3 `nodecon` Statement

The `nodecon` statement is used to label network address objects that represent IPv4 or IPv6 IP addresses and network masks.

It is also possible to add SELinux these outside the policy using the '`semanage node`' command that will associate the node to a security context.

The statement definition is:

```
nodecon subnet netmask node_context
```

Where:

`nodecon`	The `nodecon` keyword.
`subnet`	The subnet or specific IP address in IPv4 or IPv6 format. Note that the `subnet` and `netmask` values are used to ensure that the `node_context` is assigned to all IP addresses within the `subnet` range.
`netmask`	The `subnet` mask in IPv4 or IPv6 format.

node_context	The security context for the node.

The statement is valid in:

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

Examples:

```
# The 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
```

4.14.4 portcon Statement

The portcon statement is used to label udp or tcp ports.

It is also possible to add a security context to ports outside the policy using the 'semanage port' command that will associate the port (or range of ports) to a security context.

The statement definition is:

```
portcon protocol port_number port_context
```

Where:

`portcon`	The `portcon` keyword.
`protocol`	The protocol type. Valid entries are `udp` or `tcp`.
`port_number`	The port number or range of ports. The ranges are separated by a hyphen (`-`).
`port_context`	The security context for the port or range of ports.

The statement is valid in:

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

Examples:

```
# The Standard Reference Policy portcon statements:
portcon tcp 20 system_u:object_r:ftp_data_port_t
portcon tcp 21 system_u:object_r:ftp_port_t
portcon tcp 600-1023 system_u:object_r:hi_reserved_port_t
portcon udp 600-1023 system_u:object_r:hi_reserved_port_t
portcon tcp 1-599 system_u:object_r:reserved_port_t
portcon udp 1-599 system_u:object_r:reserved_port_t

# The equivalent MLS Reference Policy portcon statements:
portcon tcp 20 system_u:object_r:ftp_data_port_t:s0
portcon tcp 21 system_u:object_r:ftp_port_t:s0
portcon tcp 600-1023 system_u:object_r:hi_reserved_port_t:s0
portcon udp 600-1023 system_u:object_r:hi_reserved_port_t:s0
portcon tcp 1-599 system_u:object_r:reserved_port_t:s0
portcon udp 1-599 system_u:object_r:reserved_port_t:s0
```

`semanage(8)` Command example:

```
semanage port -a -t unconfined_t -p udp 1234
```

This command will produce the following file in the default `<policy_name>` policy store and then activate the policy:

`/etc/selinux/<policy_name>/modules/active/ports.local:`

```
# This file is auto-generated by libsemanage
# Do not edit directly.

portcon udp 1234 system_u:object_r:unconfined_t
```

4.15 MLS Statements

The optional MLS policy extension adds an additional security context component that consists of the following highlighted entries:

```
user:role:type:sensitivity[:category,...]- sensitivity [:category,...]
```

These consist of a mandatory hierarchical `sensitivity` and optional non-hierarchical `category`'s. The combination of the two comprise a `level` or security level as shown in Table 17. Depending on the circumstances, there can be one `level` defined or a `range` as shown in Table 17.

<table>
<tr><td>Security Level (or Level)
Consisting of a <code>sensitivity</code> and zero or more <code>category</code> entries:</td><td rowspan="2" colspan="3">Note that SELinux uses <code>level</code>, <code>sensitivity</code> and <code>category</code> in the language statements, however when discussing these the following terms can also be used: labels, classifications, and compartments.</td></tr>
<tr><td><code>sensitivity [: category, ...]</code></td></tr>
<tr><td colspan="4">← Range →</td></tr>
<tr><td>Low</td><td rowspan="4">–</td><td colspan="2">High</td></tr>
<tr><td><code>sensitivity [: category, ...]</code></td><td colspan="2"><code>sensitivity [: category, ...]</code></td></tr>
<tr><td>For a <code>process</code> or <code>subject</code> this is the current level or sensitivity</td><td colspan="2">For a <code>process</code> or <code>subject</code> this is the Clearance</td></tr>
<tr><td>For an <code>object</code> this is the current level or sensitivity</td><td colspan="2">For an <code>object</code> this is the maximum <code>range</code></td></tr>
<tr><td>SystemLow</td><td rowspan="2"></td><td colspan="2">SystemHigh</td></tr>
<tr><td>This is the lowest level or classification for the system (for SELinux this is generally '<code>s0</code>', note that there are no categories).</td><td colspan="2">This is the highest level or classification for the system (for SELinux this is generally '<code>s15:c0,c255</code>', although note that they will be the highest set by the policy).</td></tr>
</table>

Table 17: Sensitivity and Category = Security Level – *this table shows the meanings depending on the context being discussed.*

To make the security levels more meaningful, it is possible to use the `setransd` daemon to translate these to human readable formats. The **`semanage`**`(8)` command will allow this mapping to be defined as discussed in the `./setrans.conf` file section.

4.15.1 `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**	**`require` Statement**
No	**No**	**Yes**

Examples:

```
# The MLS Reference Policy default is to assign 16 sensitivity
# identifiers (s0 to s15):
sensitivity s0;
....
sensitivity s15;

# The policy does not specify any alias entries, however a valid
# example would be:
sensitivity s0 alias secret wellmaybe ornot;
```

4.15.2 MLS `dominance` Statement

When more than one `sensitivity` Statement is defined within a policy, then a `dominance` statement is required to define the actual hierarchy between all sensitivities.

The statement definition is:

```
dominance { sens_id ... }
```

Where:

`dominance`	The `dominance` keyword.
`sens_id`	A space separated list of previously declared `sensitivity` identifiers (or alias) in the order lowest to highest. They are enclosed in braces

	({ }), and note that there is no terminating semi-colon (;).

The statement is valid in:

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

Example:

```
# The MLS Reference Policy dominance statement defines s0 as the
# lowest and s15 as the highest sensitivity level:

dominance { s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15  }
```

4.15.3 `category` Statement

The `category` statement defines the MLS policy category identifiers[52] and optional `alias` identifiers.

The statement definition is:

```
category cat_id;
```

Or

```
category cat_id alias alias_id;
```

Where:

`category`	The `category` keyword.
`cat_id`	The `category` identifier.
`alias`	The optional `alias` keyword.
`alias_id`	One or more alias identifiers in a space separated list.

The statement is valid in:

[52] SELinux use the term ‘category’ or ‘categories’ while some MLS systems and documentation use the term ‘compartment’ or ‘compartments’, however they have the same meaning.

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;
```

4.15.4 `level` Statement

The `level` statement enables the previously declared sensitivity and category identifiers to be combined into a Security Level.

Note there must only be one `level` statement for each `sensitivity` Statement.

The statement definition is:

```
level sens_id [ :category_id ];
```

Where:

`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`).

The statement is valid in:

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

Examples:

```
# The MLS Reference Policy default is to assign each Security
# Level with the complete set of categories (i.e. the inclusive
# set from c0 to c255):

level s0:c0.c255;
...
level s15:c0.c255;
```

4.15.5 `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_type target_type new_range;
```

or (for policy version 21 and greater):

```
range_transition source_type target_type : class new_range;
```

Where:

`range_transition`	The `range_transition` keyword.
`source_type` `target_type`	One or more source / target `type` or `attribute` identifiers. Multiple entries consist of a space separated list enclosed in braces (`{ }`). Entries can be excluded from the list by using the negative operator (`-`).
`class`	The optional object `class` keyword (this allows policy versions 21 and greater to specify a class other than the default of `process`).
`new_range`	The new MLS range for the object class. The format of this field is described in the MLS `range_` Definition section.

The statement is valid in:

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

Examples:

```
# A range_transition statement from the MLS Reference Policy
# showing that a process anaconda_t can transition between
# systemLow and systemHigh depending on calling applications
# level.

range_transition anaconda_t init_script_file_type:process s0 -
s15:c0.c255;

# Two range_transition statements from the MLS Reference Policy
# showing that init will transition the audit and cups daemon
# to systemHigh (that is the lowest level they can run at).

range_transition initrc_t auditd_exec_t:process s15:c0.c255;
range_transition initrc_t cupsd_exec_t:process s15:c0.c255;
```

4.15.5.1 MLS `range` Definition

The MLS `range` is appended to a number of statements and defines the lowest and highest security levels. The range can also consist of a single `level` as discussed at the start of the MLS section.

The definition is:

```
low_level
```

Or

```
low_level - high_level
```

Where:

`low_level`	The processes lowest `level` identifier that has been previously declared by a `level` Statement. If a `high_level` is not defined, then it is taken as the same as the `low_level`.
`-`	The optional hyphen (`-`) separator if a `high_level` is also being defined.
`high_level`	The processes highest `level` identifier that has been previously declared by a `level` Statement.

4.15.6 `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:

	`( expression : expression )`
	`| not expression`
	`| expression and expression`
	`| expression or expression`
	`| u1 op u2`
	`| r1 role_mls_op r2`
	`| t1 op t2`
	`| l1 role_mls_op l2`
	`| l1 role_mls_op h2`
	`| h1 role_mls_op l2`
	`| h1 role_mls_op h2`
	`| l1 role_mls_op h1`
	`| l2 role_mls_op h2`
	`| u1 op names`
	`| u2 op names`
	`| r1 op names`
	`| r2 op names`
	`| t1 op names`
	`| t2 op names`

Where:

`u1, r1, t1, l1, h1 = Source user, role, type, low level, high level`
`u2, r2, t2, l2, h2 = Target user, role, type, low level, high level`

and:

```
op : == | !=
role_mls_op : == | != | eq | dom | domby | incomp
names : name | { name_list }
name_list : name | name_list name
```

The statement is valid in:

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

Examples:

These examples have been taken from the Reference Policy source `./policy/mls` constraints file (the `mcs` file supports the MCS constraints).

These are built into the policy at build time and add constraints to many of the object classes.

```
# The MLS Reference Policy mlsconstrain statement for searching
# directories that comprises of multiple expressions. Only the
# first two expressions are explained.
#
# Expression 1 ( l1 dom l2 ) reads as follows:
# The dir object class search permission is allowed if the
# source lowest security level is dominated by the targets
# lowest security level.
# OR
# Expression 2 (( t1 == mlsfilereadtoclr ) and ( h1 dom l2 ))
# reads as follows:
# If the source type is equal to a type associated to the
# mlsfilereadtoclr attribute and the source highest security
# level is dominated by the targets lowest security level,
# then search permission is allowed on the dir object class.

mlsconstrain dir search
   (( l1 dom l2 ) or
    (( t1 == mlsfilereadtoclr ) and ( h1 dom l2 )) or
    ( t1 == mlsfileread ) or
    ( t2 == mlstrustedobject ));
```

4.15.7 `mlsvalidatetrans` Statement

The `mlsvalidatetrans` is the MLS equivalent of the `validatetrans` statement and is only used for file related object classes where it is used to control the ability to change the objects security context.

The statement definition is:

```
mlsvalidatetrans class expression;
```

Where:

`mlsvalidatetrans`	The `mlsvalidatetrans` keyword.
`class`	One or more file type object classes. Multiple entries consist of a space separated list enclosed in braces `{}`.
`expression`	The boolean `expression` of the constraint that is defined as follows:

	`( expression : expression )`
	`  | not expression`
	`  | and (expression and expression`
	`  | or expression or expression`
	`  | u1 op u2`
	`  | r1 role_mls_op r2`
	`  | t1 op t2`
	`  | l1 role_mls_op l2`
	`  | l1 role_mls_op h2`
	`  | h1 role_mls_op l2`
	`  | h1 role_mls_op h2`
	`  | l1 role_mls_op h1`
	`  | l2 role_mls_op h2`
	`  | u1 op names`
	`  | u2 op names`
	`  | r1 op names`
	`  | r2 op names`
	`  | t1 op names`
	`  | t2 op names`
	`  | u3 op names`
	`  | r3 op names`
	`  | t3 op names`

Where:

`u1, r1, t1, l1, h1 = Old user, role, type, low level, high level`
`u2, r2, t2, l2, h2 = New user, role, type, low level, high level`
`u3, r3, t3, l3, h3 = Process user, role, type, low level, high level`

and:

```
op : == | !=
role_mls_op : == | != | eq | dom | domby | incomp
names : name | { name_list }
name_list : name | name_list name
```

The statement is valid in:

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

Examples:

This example has been taken from the Reference Policy source `./policy/mls` file.

```
# The MLS Reference Policy mlsvalidatetrans statement for
# managing the file upgrade/downgrade rules that comprises of
# multiple expressions. Only the first two expressions are
# explained.
#
# Expression 1: ( l1 eq l2 ) reads as follows:
# For a file related object to change security context, its
# current (old) low security level must be equal to the new
# objects low security level.
#
# The second part of the expression:
# or (( t3 == mlsfileupgrade ) and ( l1 domby l2 )) reads as
# follows:
# or the process type must equal a type associated to the
# mlsfileupgrade attribute and its current (old) low security
# level must be dominated by the new objects low security level.
#
mlsvalidatetrans { dir file lnk_file chr_file blk_file sock_file
fifo_file }
  ((( l1 eq l2 ) or
  (( t3 == mlsfileupgrade ) and ( l1 domby l2 )) or
  (( t3 == mlsfiledowngrade ) and ( l1 dom l2 )) or
  (( t3 == mlsfiledowngrade ) and ( l1 incomp l2 ))) and (( h1 eq h2 )
or
  (( t3 == mlsfileupgrade ) and ( h1 domby h2 )) or
  (( t3 == mlsfiledowngrade ) and ( h1 dom h2 )) or
  (( t3 == mlsfiledowngrade ) and ( h1 incomp h2 ))));
```

4.16 Policy Support Statements

This section contains language statements used to support policy.

4.16.1 `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;
```

4.16.2 `require` Statement

The `require` statement is used for two reasons:

1. Within loadable module policy source files to indicate what policy components are required from an external source file (i.e. they are not explicitly defined in this module but elsewhere). The examples below show the usage.
2. Within a base policy source file, but only if preceded by the `optional` Statement to indicate what policy components are required from an external source file (i.e. they are not explicitly defined in the base policy but elsewhere). The examples below show the usage.

The statement definition is:

```
require { rule_list }
```

Where:

`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 `optional` Statement.**	**Yes**
Conditional Policy (`if`) Statement	**`optional` Statement**	**`require` Statement**
Yes – But only if proceeded by the `optional` Statement.	**Yes**	**No**

Examples:

```
# A series of require statements showing various entries:

require {
   role system_r;
   class security { compute_av compute_create compute_member
      check_context load_policy compute_relabel compute_user
      setenforce setbool 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 };
}
```

4.16.3 `optional` Statement

The `optional` statement is used to indicate what policy statements may or may not be present in the final compiled policy. The statements will be included in the policy only if all statements within the `optional { rule list }` can be expanded successfully, this is generally achieved by using a `require` Statement at the start of the list.

The statement definition is:

```
optional { rule_list }
```

Or

```
optional { rule_list } else { rule_list }
```

Where:

`optional`	The `optional` keyword.
`rule_list`	One or more statements enclosed within braces ({ }). The list of valid statements is given in Table 16.
`else`	An optional `else` keyword.
`rule_list`	As the `rule_list` above.

The statement is valid in:

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

Examples:

```
# Use of optional block in a base policy source file.

optional {
     require {
           type unconfined_t;
     } # end require

     allow acct_t unconfined_t:fd use;
} # end optional
```

```
# Use of optional / else blocks in a base policy source file.

optional {
      require {
         type ping_t, ping_exec_t;
      } # end require

   allow dhcpc_t ping_exec_t:file { getattr read execute };
   .....
      require {
         type netutils_t, netutils_exec_t;
      } # end require
  allow dhcpc_t netutils_exec_t:file { getattr read execute };
  .....
  type_transition dhcpc_t netutils_exec_t:process netutils_t;
  ...
   } else {
      allow dhcpc_t self:capability setuid;
      .....
} # end optional
```

4.16.4 `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 SELinux Filesystem 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;
```

4.16.5 `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`**`(8)` command as it supports a permissive option as follows:

```
# semanage supports enabling and disabling of permissive
# mode using the following command:
# semanage permissive -a|d type

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

semanage permissive -a unconfined_t
```

3. Can be built into a loadable policy module so that permissive mode can be easily enabled or disabled by adding or removing the module. An example module is as follows:

```
# This is an example loadable module that would allow the
# domain to be set to permissive mode.
#
module permissive_unconfined_t 1.0.0;
require {
    type unconfined_t;
}
permissive unconfined_t;
```

The statement definition is:

```
permissive type_id;
```

Where:

`permissive`	The `permissive` keyword.
`type_id`	The `type` identifier of the domain that will be run in `permissive` mode.

The statement is valid in:

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

Example:

```
# This is the simple statement that would allow permissive mode
# to be set on the httpd_t domain, however this statement is
# generally built into a loadable policy module so that the
# permissive mode can be easily removed by removing the module.
#
permissive httpd_t;
```

`semanage(8)` Command example:

```
semanage permissive -a unconfined_t
```

This command will produce the following module in the default `<policy_name>` policy store and then activate the policy:

```
/etc/selinux/<policy_name>/modules/active/modules/permissive_unconfined_t.pp
```

4.17 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.

4.17.1 Object Classes

A list of object classes used by Fedora can be found in the Reference Policy source in the `./policy/flask/security_classes` file.

Object classes are defined within a policy as follows:

The statement definition is:

```
class class_id
```

Where:

`class`	The `class` keyword.
`class_id`	The `class` identifier.

The statement is valid in:

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

Example:

```
# Define the PostgreSQL db_tuple object class
#
class db_tuple
```

4.17.2 Permissions

A list of permissions used by the Reference Policy are listed in the `./policy/flask/access_vectors` file.

Permissions can be defined within policy in two ways:

1. Define class specific permissions. This is where permissions are declared for a specific object class only (i.e. the permission is not inherited by any other object class).
2. Define a set of `common` permissions that can then be inherited by one or more object classes. The statement for creating a set of common permissions is shown in the Defining `common` Permissions section.

The permission (or AV) statement definition is:

```
class class_id [ inherits common_set ] [ { perm_set } ]
```

Where:

`class`	The `class` keyword.
`class_id`	The previously declared `class` identifier.
`inherits`	The optional `inherits` keyword that allows a set of `common` permissions to be inherited.
`common_set`	A previously declared `common` identifier as described in the Defining `common` Permissions 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**	**`require` Statement**
No	**No**	**Yes**

Examples:

```
# The following example shows the db_tuple object class being
# allocated two permissions:

class db_tuple { relabelfrom relabelto }
```

```
# The following example shows the db_blob object class
# inheriting permissions from the database set of common
# permissions (as described in the Defining common Permissions
# section):

class db_blob inherits database
```

```
# The following example (from the access_vector file) shows the
# db_blob object class inheriting permissions from the database
# set of common permissions and adding a further four
# permissions:

class db_blob inherits database { read write import export }
```

4.17.2.1 Defining common Permissions

A list of common permissions used by the Reference Policy are listed 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	require Statement
No	**No**	**No**

Example:

```
# Define the common PostgreSQL permissions
#
common database { create drop getattr setattr relabelfrom
relabelto }
```

4.18 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.

4.18.1 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
```

4.18.2 sid context Statement

The sid context statement is used to add an initial security context to the SID that is used when SELinux initialises, or as a default if an object is not labeled correctly.

```
sid sid_id context
```

Where:

sid	The sid keyword.
sid_id	The previously declared sid identifier.
context	The initial security context associated with the SID. Note that there is no terminating ';'.

The statements are valid in:

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

Examples:

```
# These statements add an initial security context to an object
# that is used when SELinux initialises or as a default if a
```

```
# context is not available or labeled incorrectly.
#
# This one is from a targeted policy:

sid unlabeled system_u:object_r:unlabeled_t

# This one is from an MLS policy. Note that the security level
# is set to SystemHigh as it may need to label any object in
# the system.

sid unlabeled system_u:object_r:unlabeled_t:s15:c0.c255
```

4.19 Xen Statements

Xen policy supports additional policy language statements: `iomemcon`, `ioportcon`, `pcidevicecon` and `pirqcon` that are discussed in the sections that follow.

To compile these additional statements using **`semodule`**`(8)`, ensure that the **`semanage.conf`**`(5)` file has the `policy-target=xen` entry.

4.19.1 `iomemcon` Statement

The `sid` statement declares the actual SID identifier and is defined at the start of a policy source file.

The statement definition is:

```
iomemcon addr context;
```

Where:

`iomemcon`	The `iomemcon` keyword.
`addr`	The memory address to apply the context. This may also be a range that consists of a start and end address separated by a hypen ('-').
`context`	The security context to be applied.

The statement is valid in:

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

Example:

```
iomemcon 0xfebd9 system_u:object_r:nicP_t;
```

```
iomemcon 0xfebe0-0xfebff system_u:object_r:nicP_t;
```

4.19.2 `ioportcon` Statement

The `sid` statement declares the actual SID identifier and is defined at the start of a policy source file.

The statement definition is:

```
ioportcon port context;
```

Where:

`ioportcon`	The `ioportcon` keyword.
`port`	The port to apply the context. This may also be a range that consists of a start and end port number separated by a hypen ('-').
`context`	The security context to be applied.

The statement is valid in:

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

Example:

```
ioportcon 0xeac0 system_u:object_r:nicP_t;

ioportcon 0xecc0-0xecdf system_u:object_r:nicP_t;
```

4.19.3 `pcidevicecon` Statement

The `sid` statement declares the actual SID identifier and is defined at the start of a policy source file.

The statement definition is:

```
pcidevicecon pci_id context;
```

Where:

`pcidevicecon`	The `pcidevicecon` keyword.
`pci_id`	The PCI indentifer.
`context`	The security context to be applied.

The statement is valid in:

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

Example:

```
pcidevicecon 0xc800 system_u:object_r:nicP_t;
```

4.19.4 `pirqcon` Statement

The `sid` statement declares the actual SID identifier and is defined at the start of a policy source file.

The statement definition is:

```
pirqcon irq context;
```

Where:

`pirqcon`	The `pirqcon` keyword.
`irq`	The interrupt request number.
`context`	The security context to be applied.

The statement is valid in:

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

Example:

```
pirqcon 33 system_u:object_r:nicP_t;
```

5. The Reference Policy

5.1 Introduction

The Reference Policy is now the standard policy source used to build 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-17 distribution that is built from the standard Reference Policy that is modified and distributed by Red Hat as the following RPM:

`selinux-policy-3.10.0-86.fc16.src.rpm`[53]

This core source code is then used to build various policy RPMs that are distributed by Red Hat as:

`selinux-policy-3.10.0-86.fc16.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-3.10.0-86.fc16.noarch` - Contains the `html` policy documentation that is located at `/usr/share/doc/selinux-policy-3.10.0/html`

`selinux-policy-minimum-3.10.0-86.fc16.noarch`

`selinux-policy-mls-3.10.0-86.fc16.noarch`

`selinux-policy-targeted-3.10.0-86.fc16.noarch`

These three rpms contain policy configuration files and the packaged policy modules (`*.pp`). They will be used to build the particular policy type in `/usr/share/selinux/<policy_name>`, the install process will then install the policy in the appropriate `/etc/selinux/<policy_name>` directory.

[53] This RPM can be obtained from the http://koji.fedoraproject.org web site.

5.2 Reference Policy Overview

Important notes - Strictly speaking the 'Reference Policy' should refer to the policy taken from the master repository (see http://oss.tresys.com/projects/refpolicy/wiki/RepositoryCheckout) or the latest released version (see http://oss.tresys.com/projects/refpolicy/wiki/DownloadRelease). However most Linux distributors take a released version and then tailor it to their specific requirements. Therefore as this Notebook is based on Fedora 16, then this is the Reference Policy version that will be discussed (if refering to the master Reference Policy, then the word 'master' will be used).

The Reference Policy[54] can be used to build two different formats of a policy:

1. Loadable Module Policy – A policy that has a base module for core services and has the ability to load / unload modules to support applications as required [55]. This is now the standard used by GNU / Linux distributions.
2. Monolithic Policy – A policy that has all the required policy information in a single base policy.

Each of the policy types are built using module files that define the specific rules required by the policy as detailed in the Reference Policy Module Files section. Note that the monolithic policy is built using the the same module files, however they are all assembled into a single 'base' source file.

There are tools such as SLIDE (SELinux integrated development environment) that can be used to make the task of policy development and testing easier when using the Reference Policy source or headers. SLIDE is an Eclipse plugin and details can be found at:

http://oss.tresys.com/projects/slide

5.2.1 Distributing Policies

It is possible to distribute the Reference Policy in two forms:

1. As source code that is then used to build policies. This is not the general way policies are distributed as it contains the complete source that most administrators do not need. The Reference Policy Source section describes the source and the Installing and Building the Reference Policy Source section describes how to install the source and build a policy.
2. As 'Policy Headers'. This is the most common way to distribute the Reference Policy. Basically, the modules that make up 'the distribution' are pre-built and then linked to form a base and optional modules. The 'headers' that make-up the policy are then distributed along with makefiles and documentation. A policy writer can then build policy using the core modules supported by the distribution, and using development tools they can add their own policy modules. The Reference Policy Headers section describes how these are installed and used to build modules.

---

[54] The full source code and details are at the following site: http://oss.tresys.com/projects/refpolicy.

[55] These can be installed by system administrators as required. The dynamic loading / unloading of policies as applications are loaded is not yet supported.

The policy header files for F-16 are distributed in a number of rpms as follows:

selinux-policy-3.10.0-86.fc16.noarch – This package contains the SELinux /etc/selinux/config file, man pages and the 'Policy Header' development environment that is located at /usr/share/selinux/devel

selinux-policy-doc-3.10.0-86.fc16.noarch – This package contains the html policy documentation that is located at /usr/share/doc/selinux-policy-3.10.0/html

selinux-policy-minimum-3.10.0-86.fc16.noarch

selinux-policy-mls-3.10.0-86.fc16.noarch

selinux-policy-targeted-3.10.0-86.fc16.noarch

These three packages contain policy configuration files and policy modules (*.pp files) for the particular policy type to be installed. The 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-16 above, they can be classified by the name of the functionality they support (taken from the NAME entry of the build.conf as shown in Table 19), for example the Red Hat policies support[56]:

minimum – supports a minimal set of confined daemons within their own domains. The remainder run in the unconfined_t space.

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).

mls – supports server based MLS systems.

For information, the master Reference Policy supports the following types (taken from the TYPE entry of the build.conf as shown in Table 19):

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

mcs – As standard but supports MCS labels.

mls – supports MLS labels and confines server processes.

[56] Note that Red Hat pre-configure MCS support within all their policies.

5.2.3 Reference Policy Module Files

The reference policy modules are constructed using a mixture of policy language statements, support macros and access interface calls using three principle types of source file:

1. A private policy file that contains statements required to enforce policy on the specific GNU / Linux service being defined within the module. These files are named `<module_name>.te`.

   For example the `ada.te` file shown below has two statements:

   a) one to state that the `ada_t` process has permission to write to the stack and memory allocated to a file.

   b) one that states that if the `unconfined` module is loaded, then allow the `ada_t` domain unconfined access. Note that if the flow of this statement is followed it will be seen that many more interfaces and macros are called to build the final raw SELinux language statements. An expanded module source is shown in the Module Expansion Process section.

2. An external interface file that defines the services available to other modules. These files are named `<module_name>.if`.

   For example the `ada.if` file shown below has two interfaces defined for other modules to call (see also Figure 5.1 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 `interface` Macro section.

   Template Interfaces – These are required whenever a module is required in different domains and allows the type(s) to be redefined by adding a prefix supplied by the calling module. The basic idea is to set up an application in a domain that is suitable for the defined SELinux user and role to access but not others. These are generated by the `template` macro as detailed in the `template` Macro section that also explains the `openoffice.if` template.

3. A file labeling file that defines the labels to be added to files for the specified module. These files are named `<module_name>.fc`. The build process will

amalgamate all the `.fc` files and finally form the `file_contexts` file that will be used to label the filesystem.

For example the `ada.fc` file shown below requires that the specified files are all labeled `system_u:object_r:ada_exec_t:s0`.

The `<module_name>` must be unique within the reference policy source tree and should reflect the specific GNU / Linux service being enforced by the policy.

The module files are constructed using a mixture of:

1. Policy language statements as defined in the SELinux Policy Language section.
2. Reference Policy macros that are defined in the Reference Policy Macros section.
3. External interface calls defined within other modules (`.te` and `.if` only).

An example of each file taken from the `ada` module is as follows:

`ada.te` file contents:

```
policy_module(ada, 1.4.0)

########################################
#
# 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-16 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-3.10.0/html`

Distributed as Policy Source - `<location>/src/policy/doc/html`. Where `<location>` is the location of the installed source after `make install-src` has been executed as described in the Installing The Reference Policy Source section. The documentation can then be generated using `make html`, where for the build used in this Notebook the location is:

`/etc/selinux/targeted-86/src/policy/doc/html`

Figure 5.1 shows an example screen shot of the documentation produced for the ada module interfaces.

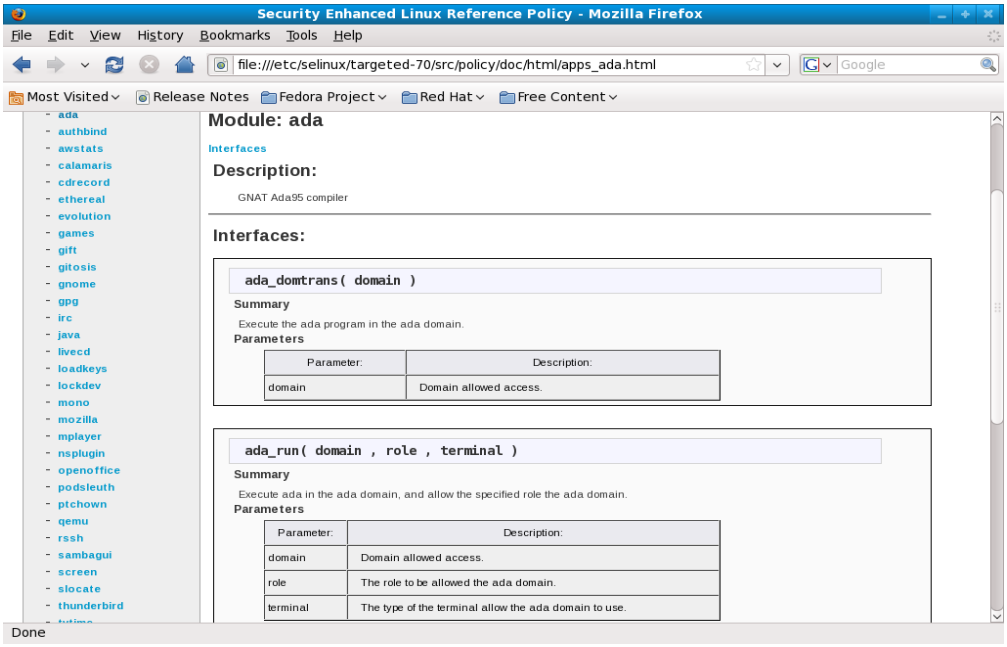


Figure 5.1: Example Documentation Screen Shot

5.3 Reference Policy Source

This section will explain the source layout and configuration files, with the actual installation and building covered in the Installing and Building the Reference Policy Source section.

The source has a `README` file containing information on the configuration and installation processes that has been used within this section (and updated with the authors comments as necessary). There is also a `VERSION` file that contains the Reference Policy release date which can be used to obtain the original source from the repository located at:

`http://oss.tresys.com/projects/refpolicy`

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:

- Reference Policy Files and Directories – Describes the files and their location.
- Source Configuration Files – Details the contents of the `build.conf` and `modules.conf` configuration files.
- Source Installation and Build Make Options – Describes the `make` targets.

- Modular Policy Build Process – Describes how the various source files are linked together to form a base policy module (`base.conf`) during the build process.

The Installing and Building the Reference Policy Source section then describes how the initial source is installed and configured to allow a version of the F-16 targeted policy to be built.

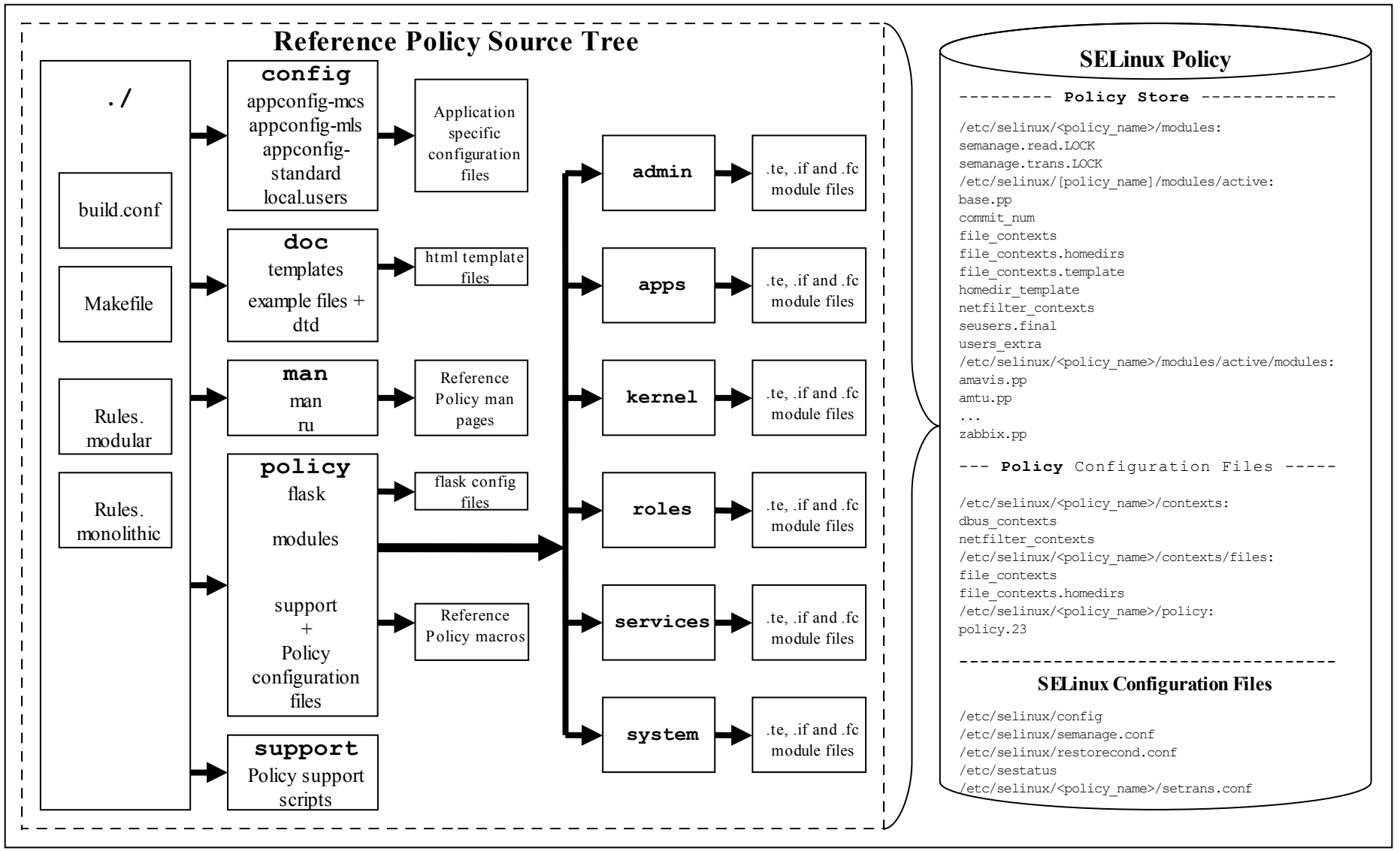


Figure 5.2: The Reference Policy Source Tree – *When building a modular policy, files are added to the policy store. For monolithic builds the policy store is not used.*

5.3.2 Reference Policy Files and Directories

Table 18 shows the major files and their directories with a description of each taken from the `README` file. All directories are relative to the root of the Reference Policy source directory `./policy`.

Two of these configuration files (`build.conf` and `modules.conf`) are further detailed in the Source Configuration Files section as they define how the policy will be built.

During the build process, a file is generated in the `./policy` directory called either `policy.conf` or `base.conf` depending whether a monolithic or modular policy is being built. This file is explained in the Modular Policy Build Structure section.

File / Directory Name	Comments
`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 Reference Policy Build Options - `build.conf` section.
`config/appconfig-<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 SELinux Configuration Files section.
`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 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 Modular Policy Build Structure section.
`policy/flask/security_classes`	This file has declarations for each security `class`. The file usage in policy generation is described in the Modular Policy Build Structure section.
`policy/flask/access_vectors`	This file defines the `access vectors`. `Common`

File / Directory Name	Comments
	prefixes for `access vectors` may be defined at the 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 Modular Policy Build Structure 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 Reference Policy Module Files section. 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 any system that implements the modular policy - see the Booleans, Global Booleans and Tunable Booleans 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 Booleans, Global Booleans and Tunable Booleans section.
`policy/global_tunables`	This file defines all tunables that have a global scope, their default value, and documentation. See the Booleans, Global Booleans and Tunable Booleans section.
`policy/mcs`	This contains information used to generate the `sensitivity`, `category`, `level` and `mlsconstraint` statements used to define the `MCS` configuration. The file usage in policy generation is described in the Modular Policy Build Structure section.
`policy/mls`	This contains information used to generate the `sensitivity`, `category`, `level` and `mlsconstraint` statements used to define the `MLS`

File / Directory Name	Comments
	configuration. The file usage in policy generation is described in the Modular Policy Build Structure section.
policy/modules.conf	This file contains a listing of available modules, and how they will be used when building Reference Policy. This file is described in the Reference Policy Build Options - modules.conf section, it is also updated by the F-16 source updates as described in the Installing and Building the Reference Policy Source section.
policy/policy_capabilities	This file defines the policy capabilities that can be enabled in the policy. The file usage in policy generation is described in the Modular Policy Build Structure section.
policy/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 Modular Policy Build Structure section.
policy/support/*	Reference Policy support macros. These are described in the Reference Policy Macros section.
securetty_types **setrans.conf**	These files are not part of the standard distribution but is added by the F-16 source updates as described in the Installing and Building the Reference Policy Source section.

Table 18: The Reference Policy Files and Directories

5.3.3 Source Configuration Files

There are two major configuration files (`build.conf` and `modules.conf`) that define the policy to be built and are detailed in this section.

5.3.3.1 Reference Policy Build Options - `build.conf`

This file defines the policy type to be built that will influence its name and where the source will be located once it is finally installed. It also configures the MCS / MLS sensitivity and category maximum values. An example file content is shown in the Installing and Building the Reference Policy Source section where it is used to install and then build the policy.

Table 19 explains the fields that can be defined within this file, however there are a number of `m4` macro parameters that are set up when this file is read by the build process makefiles. These definitions are shown in Table 20 and are also used within the module source files to control how the policy is built with examples shown in the `ifdef` / `ifndef` Parameters section.

Option	Type	Comments
OUTPUT_POLICY	Integer	Set the version of the policy created when building a

Option	Type	Comments
		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`. The `mls` and `mcs` options control the `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 Reference Policy Build Options – `policy/modules.conf` 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 SELinux Filesystem for more details. If not set, then it will be taken from the `semanage.conf` file.
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.
CUSTOM_BUILDOPT	String	Space separated list of custom build options.
MLS_SENS	Integer	Set the number of `sensitivities` in the `MLS` policy. Ignored on `standard` and `MCS` policies.
MLS_CATS	Integer	Set the number of `categories` in the `MLS` policy. Ignored on `standard` and `MCS` policies.
MCS_CATS	Integer	Set the number of `categories` in the `MCS` policy. Ignored on `standard` and `MLS` policies.
QUIET	Boolean (`y|n`)	If 'y' the build system will only display status messages and error messages. This option has no effect on policy.

Table 19: `build.conf` Entries

m4 Parameter Name in Makefile	From build.conf entry	Comments
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).

Table 20: m4 parameters set at build time - *These have been extracted from the Reference Policy Makefile file.*

5.3.3.2 Reference Policy Build Options – policy/modules.conf

This file controls what modules are built within the policy with example entries as follows:

```
# Layer: kernel
# Module: kernel
# Required in base
#
# Policy for kernel threads, proc filesystem,and unlabeled processes and
# objects.
#
kernel = base

# Module: amanda
#
# Automated backup program.
#
amanda = module

# Layer: admin
# Module: ddcprobe
#
# ddcprobe retrieves monitor and graphics card information
#
ddcprobe = off
```

As can be seen the only active line (those without comments[57]) is:

```
<module_name> = base | module | off
```

Where:

module_name	The name of the module to be included within the build.
base	The module will be in the base module for a modular policy build (build.conf entry MONOLITHIC = n).

[57] 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 distributor to decide which modules are in the base and those that are loadable, however there are some modules that MUST be in the base module. To highlight this there is a special entry at the start of the modules interface file (.if) that has the entry <required val="true"> as shown below (taken from the kernel.if file):

```
## <summary>
##Policy for kernel threads, proc filesystem,
##and unlabeled processes and objects.
## </summary>
## <required val="true">
##This module has initial SIDs.
## </required>
```

The modules.conf file will also reflect that a module is required in the base by adding a comment 'Required in base' when the make conf target is executed (as all the .if files are checked during this process and the modules.conf file updated).

```
# Layer: kernel
# Module: kernel
# Required in base
#
# Policy for kernel threads, proc filesystem,and unlabeled processes and
objects.
#
kernel = base
```

There are 12 modules in the F-16 reference policy source marked as required and are shown in Table 21.

Layer	Module Name	Comments
kernel	corecommands	Core policy for shells, and generic programs in: /bin, /sbin, /usr/bin, and /usr/sbin. The .fc file sets up the labels for these items. All the interface calls start with 'corecmd_'.
kernel	corenetwork	Policy controlling access to network objects and also contains the initial SIDs for these. The .if file is large and automatically generated. All the interface calls start with 'corenet_'.
kernel	devices	This module creates the device node concept and provides the policy for many of the device files. Notable exceptions are the mass storage and terminal devices that are covered by other modules (that is a char or block device file, usually in /dev). All types that are used to label device nodes should use the dev_node macro. Additionally this module controls access to three things:

Layer	Module Name	Comments
		1. the device directories containing device nodes. 2. device nodes as a group 3. individual access to specific device nodes covered by this module. All the interface calls start with 'dev_'.
kernel	domain	Contains the core policy for forming and managing domains. All the interface calls start with 'domain_'.
kernel	files	This module contains basic filesystem types and interfaces and includes: 1. The concept of different file types including basic files, mount points, tmp files, etc. 2. Access to groups of files and all files. 3. Types and interfaces for the basic filesystem layout (/, /etc, /tmp, /usr, etc.). 4. Contains the file initial SID. All the interface calls start with 'files_'.
kernel	filesystem	Contains the policy for filesystems and the initial SID. All the interface calls start with 'fs_'.
kernel	kernel	Contains the policy for kernel threads, proc filesystem, and unlabeled processes and objects. This module has initial SIDs. All the interface calls start with 'kernel_'.
kernel	mcs	Policy for Multicategory security. The .te file only contains attributes used in MCS policy. All the interface calls start with 'mcs_'.
kernel	mls	Policy for Multilevel security. The .te file only contains attributes used in MLS policy. All the interface calls start with 'mls_'.
kernel	selinux	Contains the policy for the kernel SELinux security interface (selinuxfs). All the interface calls start with 'selinux_'.
kernel	terminal	Contains the policy for terminals. All the interface calls start with 'term_'.
kernel	ubac	To support user-based access control - if enabled.

Table 21: Mandatory `modules.conf` Entries

5.3.3.2.1 Building the `modules.conf` File

The file can be created by an editor, however it is generally built initially by `make conf` that will add any additional modules to the file. The file can then be edited to configure the required modules as base, module or off.

As will be seen in the Installing and Building the Reference Policy Source section, the Red Hat reference policy source comes with a number of pre-configured files that are used to produce the required policy including multiple versions of the `modules.conf` file.

5.3.4 Source Installation and Build Make Options

This section explains the various make options available that have been taken from the `README` file. Table 22 describes the general make targets, Table 23 describes the modular policy make targets and Table 24 describes the monolithic policy make targets.

Make Target	Comments
`install-src`	Install the policy sources into `/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 (not in README text but still used)

Table 22: 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 23: Modular Policy Build Make Targets

Make Target	Comments
policy	Compile a policy locally for development and testing. This is the default target for monolithic policies.
install	Compile and install the policy and file contexts.
load	Compile and install the policy and file contexts, then load the policy.
enableaudit	Remove all `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 24: Monolithic Policy Build Make Targets

5.3.5 Booleans, Global Booleans and Tunable Booleans

The three files `booleans.conf`, `global_booleans` and `global_tunables` are built and used as follows:

`booleans.conf`	This file is generated / updated by `make conf`, and contains all the booleans in the policy with their default values. If tunable and global booleans are implemented then these are also included. This file can also be delivered as a part of the reference policy source as shown in the Installing and Building the Reference Policy Source section. This is generally because other default values are used for booleans and not those defined within the modules themselves (i.e. distribution specific booleans). When the make install is executed, this file will be used to set the default values. Note that if booleans are updated locally the policy store will contain a `booleans.local` file. In SELinux enabled systems that support the policy store features (modular policies) this file is not installed as `/etc/selinux/NAME/booleans`.
`global_booleans`	These are booleans that have been defined in the `global_tunables` file using the `gen_bool` macro. They are normally booleans for managing the overall policy and currently consist of the following (where the default values are `false`): `secure_mode`
`global_tunables`	These are booleans that have been defined in module

	files using the gen_tunable macro and added to the global_tunables file by make conf. The tunable_policy macros are defined in each module where policy statements or interface calls are required. They are booleans for managing specific areas of policy that are global in scope. An example is allow_execstack that will allow all processes running in unconfined_t to make their stacks executable.

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 macros[58] in the source modules will be expanded to form actual policy language statements as described in the SELinux Policy Language section. Figure 5.3 shows these temporary files that are used to form the base.conf[59] file during policy generation.

The base.conf file will consist of language statements taken from the module defined as base in the modules.conf file along with the constraints, users etc. that are required to build a complete policy.

The individual loadable modules are built in much the same way as shown in Figure 5.4.

Base Policy Component Description	Policy Source File Name (relative to ./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 statements extracted from the *.te and	modules/*/*.te modules/*/*.if	all_attrs_types.conf

[58] These are explained in the Reference Policy Macros section.

[59] 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
*.if files that form the base module.		
Contains the global and tunable bools extracted from the conf files.	global_bools.conf global_tunables.conf	global_bools.conf
Contains the rules extracted from each of the modules .te and .if files defined in the modules.conf file as 'base'.	base modules	only_te_rules.conf
Contains the expanded users from the users file.	users	all_post.conf
Contains the expanded constraints from the constraints file.	constraints	
Contains the default SID labeling extracted from the *.te files.	modules/*/*.te	
Contains the fs_use_xattr, fs_use_task, fs_use_trans and genfscon statements extracted from each of the modules .te and .if files defined in the modules.conf file as 'base'.	modules/*/*.te modules/*/*.if	
Contains the netifcon, nodecon and portcon statements extracted from each of the modules .te and .if files defined in the modules.conf file as 'base'.	modules/*/*.te modules/*/*.if	
Contains the expanded file context file entries extracted from the *.fc files defined in the modules.conf file as 'base'.	modules/*/*.fc	base.fc.tmp
Expanded seusers file.	seusers	seusers
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.	modules/*/<module_name>.te modules/*/<module_name>.if	<module_name>.tmp
For each module defined as 'module' in the modules.conf configuration file, an object module is produced from executing the checkmodule command shown below.	tmp/<module_name>.tmp	<module_name>.mod

For each module defined as 'module' in the modules.conf configuration file, an expanded file context file is built from the <module_name>.fc file.	modules/*/<module_name>.fc	base.fc.tmp

This command is used to compile each module:
`checkmodule tmp/<module_name>.tmp -o tmp/<module_name>.mod`
 	
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-16 reference policy source that is distributed by Red Hat (however the same principle is followed for the source taken directly from the Tresys repository, except that it will not build a compatible policy to that discussed in this section).

Any F-16 policy source rpm will suffice and can be obtained from the http://koji.fedoraproject.org web site, however it is assumed that the source rpm is:

selinux-policy-3.10.0-86.fc16.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-16 targeted rpm:

selinux-policy-targeted-3.10.0-86.fc16.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-3.10.0-86.fc16.src.rpm
```

The source will be installed in the users home directory under `./rpmbuild/SOURCES` where the `serefpolicy-3.10.0.tgz` will need to be unpacked:

```
cd $HOME/rpmbuild/SOURCES
tar -xzf serefpolicy-3.10.0.tgz
```

The `SOURCES` directory contents will then look like this:

```
booleans-minimum.conf        booleans-mls.conf         booleans-targeted.conf
config.tgz                   customizable_types        file_contexts.subs_dist
Makefile.devel               modules-minimum.conf      modules-mls.conf
modules-targeted.conf        policy-F16.patch          securetty_types-minimum
securetty_types-mls          securetty_types-targeted  serefpolicy-3.10.0
serefpolicy-3.10.0.tgz       setrans-minimum.conf       setrans-mls.conf
setrans-targeted.conf        users-minimum             users-mls
users-targeted
```

The files with `minimum`, `targeted`, and `mls` 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:

```
cd serefpolicy-3.10.0

patch -p1 <../policy-F16.patch
```

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:
cd ..
tar -xzf config.tgz

# move to source directory:
cd serefpolicy-3.10.0

# save the old files:
mv config config.org

# and copy over the new Red Hat files
cp -r ../config config

# But also need two files from original location:
cp config.org/appconfig-mcs/sepgsql_contexts config/appconfig-mcs
cp config.org/appconfig-mcs/x_contexts config/appconfig-mcs
```

As the 'targeted' policy is being built, the files shown in Table 25 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-3.10.0/policy/booleans.conf
customizable_types	./serefpolicy-3.10.0/config/appconfig-mcs/customizable_types
file_contexts.subs_dist	./serefpolicy-3.10.0/config/appconfig-mcs/file_contexts.subs_dist
modules-targeted.conf	./serefpolicy-3.10.0/policy/modules.conf
securetty_types-targeted	./serefpolicy-3.10.0/config/appconfig-mcs/securetty_types
setrans-targeted.conf	./serefpolicy-3.10.0/config/appconfig-mcs/setrans.conf
users-targeted.conf	./serefpolicy-3.10.0/policy/users

Table 25: Red Hat specific policy configuration files – *This example builds a 'targeted' policy.*

The serefpolicy-3.10.0 directory will now contain the source code with the latest patches for this release (3.10.0-86) of the Red Hat Reference Policy and the correct configuration files for a targeted policy.

The ./serefpolicy-3.10.0/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 below[60]:

```
#
# 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-86/src/policy directory.
NAME = targeted-86

# 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.
```

[60] The README file in this directory contains helpful information on installation of the source, headers, documentation etc. The only point the README will not cover are the Red Hat specific configuration files that need to be copied over as shown in Table 25.

```
DISTRO = redhat

# Unknown Permissions Handling
# The behaviour for handling permissions defined in the kernel but missing from
# the policy.  The permissions can either be allowed, denied, or the policy
# loading can be rejected.
# allow, deny, and reject are current options. 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

CUSTOM BUILD OPTS

# Number of MLS Sensitivities
# The sensitivities will be s0 to s(MLS_SENS-1). Dominance will be in increasing
# numerical order with s0 being lowest.
MLS_SENS = 16

# Number of MLS Categories. Note Red Hat use 1024 categories for MLS and MCS.
# The categories will be c0 to c(MLS_CATS-1).
MLS_CATS = 1024

# Number of MCS Categories
# The categories will be c0 to c(MLS_CATS-1).
MCS_CATS = 1024

# Set this to y to only display status messages during build.
QUIET = n
```

The policy source is now in a position to be installed at its default location that will be derived from the `NAME = targeted-86` entry and will therefore be located at:

`/etc/selinux/targeted-86/src/policy`

5.4.2 Building the `targeted` Policy Type

From the `./serefpolicy-3.10.0` 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 store[61] by running the following commands:

```
# Go to the source location:
cd /etc/selinux/targeted-86/src/policy
```

```
# To ensure a clean source build:
make clean
```

[61] 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

```
# Build the policy modules and load into the policy store:
make load
```

The policy will now be built as a `targeted` policy that will be an exact copy of the policy distributed in the following rpm:

`selinux-policy-targeted-3.10.0-86.fc16.noarch.rpm`

Finally copy over 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-86
cp  config/appconfig-mcs/file_contexts.subs_dist
  /etc/selinux/targeted-86/contexts/files
```

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-3.10.0-86.fc16.noarch`

The binary files sizes of both policies should be 4,398,420 bytes.

```
ls -l /etc/selinux/targeted/policy
-rw-r--r-- root root 4398420 <date+time> policy.26

ls -l /etc/selinux/targeted-86/policy
-rw-r--r-- root root 4398420 <date+time> policy.26
```

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 policies[62] which should show that they are the same and give the results shown in Figure 5.5.

```
Policy:
/etc/selinux/targeted/policy/policy.26
Policy Version & Type: v.26 (binary, mls)

Number of Classes and Permissions:
    Object Classes: 82
    Common Classes: 5
    Permissions: 241

Number of Types and Attributes:
    Types: 3651
    Attributes: 297

Number of Rules:
    allow: 91628
    auditallow: 103
    dontaudit 7002
    neverallow: not calculated
    type_change: 62
    type_member: 46
    type_transition: 14467

Number of Roles: 13

Number of RBAC Rules:
```

```
Policy:
/etc/selinux/targeted-86/policy/policy.26
Policy Version & Type: v.26 (binary, mls)

Number of Classes and Permissions:
    Object Classes: 82
    Common Classes: 5
    Permissions: 241

Number of Types and Attributes:
    Types: 3651
    Attributes: 297

Number of Rules:
    allow: 91628
    auditallow: 103
    dontaudit 7002
    neverallow: not calculated
    type_change: 62
    type_member: 46
    type_transition: 14467

Number of Roles: 13

Number of RBAC Rules:
```

[62] 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.

```
    allow: 290                                    allow: 290
    role_transition 0                             role_transition 0

Number of Users: 9                            Number of Users: 9

Number of Booleans: 218                       Number of Booleans: 218

                          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-86` build edit the `/etc/selinux/config` file entry to read `SELINUXTYPE=targeted-86`, 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 Red Hat F-16 policy header installations are slightly different as described below.

5.5.1 Building and Installing the Header Files

To be able to fully build the policy headers from the reference policy source two steps are required:

1. Ensure the source is installed and configured as described in the Installing and Building the Reference Policy Source section. This is because the `make load` (or `make install`) command will package all the modules as defined in the `modules.conf` file, producing a `base.pp` and the relevant `.pp` packages. The build process will then install these 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-86` build generated above (edited for readability) would be:

```
# The policy packages:
targeted-86/abrt.pp
....
targeted-86/base.pp

# Build / Configuration files:
targeted-86/include/build.conf
targeted-86/include/Makefile
targeted-86/include/rolemap  # Note this file is not used by F-16

# XML Documentation:
targeted-86/include/global_tunables.xml
targeted-86/include/global_booleans.xml
targeted-86/include/apps.xml
targeted-86/include/roles.xml
targeted-86/include/system.xml
targeted-86/include/kernel.xml
targeted-86/include/services.xml
targeted-86/include/admin.xml

# Support Macros:
targeted-86/include/support/ipc_patterns.spt
...
...
# The module interface files in their relevant directories:
targeted-86/include/admin/acct.if
..
targeted-86/include/apps/ada.if
..
targeted-86/include/kernel/corecommands.if
..
targeted-86/include/roles/auditadm.if
..
targeted-86/include/services/abrt.if
...
targeted-86/include/system/application.if
...
```

5.5.2 Using the Standard Ref Policy Headers

Note that this section describes the standard Reference Policy headers, the F-16 installation is slightly different and described in the Using F-16 Supplied Headers section.

Once the headers are installed as defined above, new modules can be built in any local directory. An example set of module files are located in the reference policy source at `/etc/selinux/targeted-86/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-86/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
```

Table 26 shows the make targets for modules built from headers.

Make Target	Comments
MODULENAME.pp	Compile and package the MODULENAME local module.
all	Compile and package the modules in the current directory.
load	Compile and package the modules in the current directory, then insert them into the module store.
refresh	Attempts to reinsert all modules that are currently in the module store from the local and system module packages.
xml	Build a policy.xml from the XML included with the base policy headers and any XML in the modules in the current directory.

Table 26: Header Policy Build Make Targets

5.5.3 Using F-16 Supplied Headers

The F-16 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>-3.10.0-86.fc16.noarch type rpms.
- The development header files are installed in the /usr/share/selinux/devel directory by the selinux-policy-3.10.0-86.fc16.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-3.10.0-86.fc16.noarch type rpms and would be installed (for this version), in the /usr/share/doc/selinux-policy-3.10.0/html directory.

5.6 Reference Policy Support Macros

This section explains some of the support macros used to build reference policy source modules (see Table 27 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 27: Support Macros described in this section

Notes:

1. The macro calls can be in any configuration file read by the build process and can be found in (for example) the `users`, `mls`, `mcs` and `constraints` files.
2. There are four main m4 `ifdef` parameters used within modules:
   a) `enable_mcs` - this is used to test if the MCS policy is being built.
   b) `enable_mls` - this is used to test if the MLS policy is being built.
   c) `enable_ubac` - this enables the user based access control within the `constraints` file.
   d) `hide_broken_symptoms` - this is used to hide errors in modules with `dontaudit` rules.

   These are also mentioned in Table 20 as they are set by the initial build process with examples shown in the `ifdef` / `ifndef` Parameters section.
3. The macro examples in this section have been taken from the reference policy module files and shown in each relevant "**Example Macro**" section. The macros are then expanded by the build process to form modules containing the policy language statements and rules in the `tmp` directory. These files have been extracted and modified for readability, then shown in each relevant "**Expanded Macro**" section.
4. An example policy that has had macros expanded is shown in the Module Expansion Process section.
5. Be aware that spaces between macro names and their parameters are not allowed:

   Correct:

   ```
   policy_module(ftp, 1.7.0)
   ```

   Incorrect:

   ```
   policy_module (ftp, 1.7.0)
   ```

5.6.1 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 `module` statement to a loadable module, and automatically add a `require` Statement with pre-defined information for all loadable modules

such as the system_r role, kernel classes and permissions, and optionally MCS / MLS information (sensitivity and category statements).

The macro definition is:

```
policy_module(module_name,version)
```

Where:

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 `require` Statement.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	**Yes**	**No**

Example Macro:

```
# This example is from the modules/services/ftp.te module:
#
gen_require(`type ftp_script_exec_t;')
```

Expanded Macro:

```
# This is the expanded macro from the tmp/ftp.tmp file:
#
require {
  type ftp_script_exec_t;
}
```

5.6.1.3 `optional_policy` Macro

For use within module files to insert an `optional` block that will be expanded by the build process only if the modules containing the access or template interface calls that follow are present. If one module is present and the other is not, then the optional statements are not included (need to check).

The macro definition is:

```
optional_policy(`optional_statements`)
```

Where:

`optional_policy`	The `optional_policy` macro keyword.
`optional_statements`	These statements consist of those allowed in the policy language `optional` Statement. However they can also be `interface`, `template` or support macro calls.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	**Yes**	**No**

Example Macro:

```
# This example is from the modules/services/ftp.te module and
# shows the optional_policy macro with two levels.
#
optional_policy(`
   corecmd_exec_shell(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.
#
##### Start optional_policy - Level 1 ############
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)

##### Start optional_policy - Level 2 ##########
   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 `tunable_policy` macro contains the supporting statements allowed or not depending on the value of the boolean. These entries are extracted as a part of the build process (by the `make conf` target) and added to the `global_tunables` file where they can then be used to alter the default values for the `make load` or `make install` targets.

Note that the comments shown in the example MUST be present as they are used to describe the function and are extracted for the documentation.

The macro definition is:

```
gen_tunable(boolean_name,boolean_value)
```

Where:

`gen_tunable`	The `gen_tunable` macro keyword.
`boolean_name`	The `boolean` identifier.
`boolean_value`	The `boolean` value that can be either `true` or `false`.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	**Yes**	**No**

Example Macro:

```
# This example is from the modules/services/ftp.te module:
#

## <desc>
## <p>
## Allow ftp servers to use nfs
## for public file transfer services.
## </p>
## </desc>
gen_tunable(allow_ftpd_use_nfs, false)
```

Expanded Macro:

```
# This is the expanded macro from the tmp/ftp.tmp file:
```

```
#
bool allow_ftpd_use_nfs false;
```

5.6.1.5 `tunable_policy` Macro

This macro contains the statements allowed or not depending on the value of the boolean defined by the `gen_tunable` macro.

The macro definition is:

```
tunable_policy(`gen_tunable_id',`tunable_policy_rules`)
```

Where:

`tunable_policy`	The `tunable_policy` macro keyword.
`gen_tunable_id`	This is the boolean identifier defined by the `gen_tunable` macro. It is possible to have multiple entries separated by `&&` or `||` as shown in the example.
`tunable_policy_rules`	These are the policy rules and statements as defined in the `if` statement policy language section.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	**Yes**	**No**

Example Macro:

```
# This example is from the modules/services/ftp.te module
# showing the use of the boolean with the && operator.
#
tunable_policy(`allow_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.

The macro definition is:

```
interface(`name`,`interface_rules`)
```

Where:

`interface`	The `interface` macro keyword.
`name`	The `interface` identifier that should be named to reflect the module identifier and its purpose.
`interface_rules`	This can consist of the support macros, policy language statements or other `interface` calls as required to provide the service.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
No	**Yes**	**No**

Example Interface Definition:

Note that the comments shown in the example MUST be present as they are used to describe the function and are extracted for the documentation.

```
# This example is from the modules/services/ftp.if module
# showing the 'ftp_read_config' interface.
#

########################################
## <summary>
##      Read ftpd etc files
## </summary>
## <param name="domain">
##<summary>
##      Domain allowed access.
##</summary>
## </param>
#
interface(`ftp_read_config',`
  gen_require(`
     type ftpd_etc_t;
  ')
```

```
   files_search_etc($1)
   allow $1 ftpd_etc_t:file { getattr read };
')
```

Expanded Macro: (taken from the `base.conf` file)**:**

```
# Access Interfaces are only expanded at policy compile time
# if they are called by a module that requires their services.
#
# In this example the ftp_read_config interface is called from
# the init.te module via the optional_policy macro as shown
# below with the expanded code shown afterwards.
#
######## From ./policy/policy/modules/system/init.te ########
#
# optional_policy(`
#         ftp_read_config(initrc_t)
# ')
#
#
############## Expanded policy statements taken ###############
############## from the base.conf file that ##################
############## forms the base policy. ########################
#
optional { # Start optional_policy segment for ftp interface
#
# This is the resulting output contained the base.conf file
# where init calls the ftp_read_config ($1) interface from
# init.te with the parameter initrc_t:
#
   require {
       type ftpd_etc_t;
   }

#
# Call the files_search_etc ($1) interface contained in the
# ftp.if file with  the parameter initrc_t:
#
   require {
       type etc_t;
   }
   allow initrc_t etc_t:dir { getattr search };
#
# end files_search_etc(initrc_t)
#
# This is the allow $1 ftpd_etc_t:file { getattr read };
# statement with the initrc_t parameter resolved:
#
   allow initrc_t ftpd_etc_t:file { getattr read };
#
# end ftp_read_config(initrc_t)

} # End optional_policy segment for this ftp interface
```

5.6.1.7 `template` Macro

A template interface is used to help create a domain and set up the appropriate rules and statements to run an application / process. The basic idea is to set up an application in a domain that is suitable for the defined SELinux user and role to access but not others. Should a different user / role need to access the same application, another domain would be allocated (these are known as ‘derived domains’ as the domain name is derived from caller information).

The application template shown in the example below is for `openoffice.org` where the domain being set up to run the application is based on the SELinux user `xguest` (parameter `$1`) therefore a domain type is initialised called `xguest_openoffice_t`, this is then added to the user domain attribute `xguest_usertype` (parameter `$2`). Finally the role `xguest_r` (parameter `$3`) is allowed access to the domain type `xguest_openoffice_t`. If a different user / role required access to openoffice.org, then by passing different parameters (i.e. `user_u`), a different domain would be set up.

The main differences between an application interface and a template interface are:

- An access interface is called by other modules to perform a service.
- A template interface allows an application to be run in a domain based on user / role information to isolate different instances.

Note that the comments shown in the example MUST be present as they are used to describe the function and are extracted for the documentation.

The macro definition is:

```
template(`name`,`template_rules`)
```

Where:

`template`	The `template` macro keyword.
`name`	The `template` identifier that should be named to reflect the module identifier and its purpose. By convention the last component is `_template` (e.g. `ftp_per_role_template`).
`template_rules`	This can consist of the support macros, policy language statements or `interface` calls as required to provide the service.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
No	**Yes**	**No**

Example Macro:

```
# This example is from the modules/apps/openoffice.if module
# showing the 'openoffice_per_role_template' template interface.
#
########################################
## <summary>
## The per role template for the openoffice module.
## </summary>
## <desc>
## <p>
## This template creates a derived domains which are used
## for openoffice applications.
## </p>
## </desc>
## <param name="userdomain_prefix">
## <summary>
```

```
## The prefix of the user domain (e.g., user
## is the prefix for user_t).
## </summary>
## </param>
## <param name="user_domain">
## <summary>
## The type of the user domain.
## </summary>
## </param>
## <param name="user_role">
## <summary>
## The role associated with the user domain.
## </summary>
## </param>
#
template(`openoffice_per_role_template',`
   gen_require(`
       type openoffice_exec_t;
   ')

   type $1_openoffice_t;
   domain_type($1_openoffice_t)
   domain_entry_file($1_openoffice_t, openoffice_exec_t)
   role $3 types $1_openoffice_t;

   domain_interactive_fd($1_openoffice_t)

   userdom_unpriv_usertype($1, $1_openoffice_t)
   userdom_exec_user_home_content_files($1, $1_openoffice_t)

   allow $1_openoffice_t self:process { getsched sigkill execheap execmem
execstack };

   allow $2 $1_openoffice_t:process { getattr ptrace signal_perms noatsecure
siginh rlimitinh };
   allow $1_openoffice_t $2:tcp_socket { read write };

   domtrans_pattern($2, openoffice_exec_t, $1_openoffice_t)

   dev_read_urand($1_openoffice_t)
   dev_read_rand($1_openoffice_t)

   fs_dontaudit_rw_tmpfs_files($1_openoffice_t)

   allow $2 $1_openoffice_t:process { signal sigkill };
   allow $1_openoffice_t $2:unix_stream_socket connectto;
')
```

Expanded Macro:

```
# Template Interfaces are only expanded at policy compile time
# if they are called by a module that requires their services.
# This has been expanded as a part of the roles/xguest.te
# module and extracted from tmp/xguest.tmp.
#
################# START Expanded code segment ###########
#
optional {

##### begin openoffice_per_role_template(xguest,xguest_usertype,xguest_r)
   require {
       type openoffice_exec_t;
   } # end require
   type xguest_openoffice_t; # 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:

```
# This is from the modules/apps/gnome.fc file. Note that the
# HOME_DIR and USER parameters will be entered during
# the file_contexts.homedirs file build as described in the
# modules/active/file_contexts.template File section.
#

HOME_DIR/.gnome2(/.*)?
  gen_context(system_u:object_r:gnome_home_t,s0)
HOME_DIR/\.config/gtk-.*
  gen_context(system_u:object_r:gnome_home_t,s0)
HOME_DIR/\.gconf(d)?(/.*)?
  gen_context(system_u:object_r:gconf_home_t,s0)
HOME_DIR/\.local.*
  gen_context(system_u:object_r:gconf_home_t,s0)

/tmp/gconfd-USER/.* --
  gen_context(system_u:object_r:gconf_tmp_t,s0)

HOME_DIR/.pulse(/.*)?
  gen_context(system_u:object_r:gnome_home_t,s0)
```

Expanded File Context `.fc` file:

```
# The resulting expanded tmp/gnome.mod.fc file. This will be
# concatenated with the main file_contexts file during the
# policy build process.
#

HOME_DIR/.gnome2(/.*)?     system_u:object_r:gnome_home_t:s0
HOME_DIR/\.config/gtk-.*   system_u:object_r:gnome_home_t:s0
HOME_DIR/\.gconf(d)?(/.*)?system_u:object_r:gconf_home_t:s0
HOME_DIR/\.local.*      system_u:object_r:gconf_home_t:s0

/tmp/gconfd-USER/.* -- system_u:object_r:gconf_tmp_t:s0

HOME_DIR/.pulse(/.*)?      system_u:object_r:gnome_home_t:s0
```

5.6.2.2 `gen_user` Macro

This macro is used to generate a valid user statement and add an entry in the users_extra configuration file if it exists.

The macro definition is:

```
gen_user(username, prefix, role_set, mls_defaultlevel,
mls_range, [mcs_categories])
```

Where:

gen_user	The gen_user macro keyword.
username	The SELinux user id.
prefix	SELinux users without the prefix will not be in the users_extra file. This is added to user

	directories by the genhomedircon as discussed in the modules/active/file_contexts.template File section.
role_set	The user roles.
mls_defaultlevel	The default level if MLS / MCS policy.
mls_range	The range if MLS / MCS policy.
mcs_categories	The categories if MLS / MCS policy.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	**No**	**No**

Example Macro:

```
# This example has been taken from the policy/policy/users file:
#

gen_user(root, user, unconfined_r sysadm_r staff_r
ifdef(`enable_mls',`secadm_r auditadm_r') system_r, s0, s0 -
mls_systemhigh, mcs_allcats)
```

Expanded Macro:

```
# The expanded gen_user macro from the base.conf for an MLS
# build. Note that the prefix is not present. This is added to
# the users_extra file as shown below.
#

user root roles { unconfined_r sysadm_r staff_r secadm_r
auditadm_r system_r } level s0 range s0 - s15:c0.c1023;
```

```
# users_extra file entry:
#
user root prefix user;
```

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 if / else statement as shown in the example.

Note that the comments shown in the example MUST be present as they are used to describe the function and are extracted for the documentation.

The macro definition is:

```
gen_bool(name,default_value)
```

Where:

`gen_bool`	The `gen_bool` macro keyword.
`name`	The boolean identifier.
`default_value`	The value `true` or `false`.

The macro is only valid in the **`global_booleans`** file but the boolean declared can be used in the following module types:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
Yes	**Yes**	**No**

Example Macro (in `global_booleans`):

```
# This example is from the global_booleans file where the bool
# is declared. The comments must be present as it is used to
# generate the documentation.
#

## <desc>
## <p>
## Disable transitions to insmod.
## </p>
## </desc>
gen_bool(secure_mode_insmod,false)
```

```
# Example usage from the system/modutils.te module:
#
if( ! secure_mode_insmod ) {
  kernel_domtrans_to(insmod_t,insmod_exec_t)
}
```

Expanded Macro:

```
# This has been taken from the base.conf source file after
# expansion by the build process of the modutils.te module.
#

if( ! secure_mode_insmod ) {
##### begin kernel_domtrans_to(insmod_t,insmod_exec_t)
  allow kernel_t insmod_exec_t:file { getattr read execute };
  allow kernel_t insmod_t:process transition;
  dontaudit kernel_t insmod_t:process { noatsecure siginh
rlimitinh };
  type_transition kernel_t insmod_exec_t:process insmod_t;
  allow insmod_t kernel_t:fd use;
  allow insmod_t kernel_t:fifo_file { getattr read write append
ioctl lock };
  allow insmod_t kernel_t:process sigchld;
##### end kernel_domtrans_to(insmod_t,insmod_exec_t)
}
```

5.6.3 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 `category` statement for each category defined. These are then used in the `base.conf` / `policy.conf` source file and also inserted into each module by the `policy_module` Macro. The `policy/policy/mcs` and `mls` configuration files are the only files that contain this macro in the current reference policy.

The macro definition is:

```
gen_cats(mcs_num_cats | mls_num_cats)
```

Where:

`gen_cats`	The `gen_cats` macro keyword.
`mcs_num_cats` `mls_num_cats`	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 `sensitivity` statement for each sensitivity defined. These are then used in the `base.conf` / `policy.conf` source file and also inserted into each module by the `policy_module` Macro. The

policy/policy/mcs and mls configuration files are the only files that contain this macro in the current reference policy (note that the mcs file has gen_sens(1) as only one sensitivity is required).

The macro definition is:

```
gen_sens(mls_num_sens)
```

Where:

gen_sens	The gen_sens macro keyword.
mls_num_sens	These are the maximum number of sensitivities that have been extracted from the build.conf file MLS_SENS entries and set as an m4 parameter.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
na	**na**	**na**

Example Macro:

```
# This example is from the policy/policy/mls configuration file.
#

gen_cats(mls_num_sens)
```

Expanded Macro:

```
# This example has been extracted from the base.conf source
# file.

sensitivity s0;
sensitivity s1;
...
sensitivity s15;
```

5.6.3.3 gen_levels Macro

This macro will generate a level statement for each level defined. These are then used in the base.conf / policy.conf source file. The policy/policy/mcs and mls configuration files are the only files that contain this macro in the current reference policy.

The macro definition is:

```
gen_levels(mls_num_sens,mls_num_cats)
```

Where:

gen_levels	The gen_levels macro keyword.
mls_num_sens	This is the parameter that defines the number of sensitivities to generate. The MCS policy is set to '1'.
mls_num_cats mcs_num_cats	This is the parameter that defines the number of categories to generate.

The macro is valid in:

Private Policy File (.te)	External Interface File (.if)	File Labeling Policy File (.fc)
na	**na**	**na**

Example Macro:

```
# This example is from the policy/policy/mls configuration file.
#
gen_levels(mls_num_sens,mls_num_cats)
```

Expanded Macro:

```
# This example has been extracted from the base.conf source
# file. Note that the all categories are allocated to each
# sensitivity.

level s0:c0.c1023;
level s1:c0.c1023;
...
level s15:c0.c1023;
```

5.6.3.4 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.

Example Macro:

```
# This example is from the modules/kernel/domain.te module.
#
ifdef(`hide_broken_symptoms',`
  cron_dontaudit_rw_tcp_sockets(domain)
  allow domain domain:key { link search };
')
```

5.6.4.2 `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-16 (`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 | ubac = n`).

Example Macro:

```
# This example is from the ./policy/constraints file.
# Note that the ubac_constrained_type attribute is defined in
# modules/kernel/ubac.te module.

define(`basic_ubac_conditions',`
  ifdef(`enable_ubac',`
     u1 == u2
     or u1 == system_u
     or u2 == system_u
     or t1 != ubac_constrained_type
     or t2 != ubac_constrained_type
  ')
')
```

5.6.4.4 `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)
  ')
')
```

```
# This example is from the modules/system/userdomain.te module.
#
ifdef(`direct_sysadm_daemon',`
     domain_system_change_exemption($1_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 Reference Policy Module Files section.

The initial build process will build the source text files in the `policy/tmp` directory as `ada.tmp` and `ada.mod.fc` (that are basically build equivalent `ada.conf` and `ada.fc` formatted files). The basic steps are shown in Figure 5.6, and the resulting expanded code shown in Figure 5.7 and then described in the Module Expansion section.

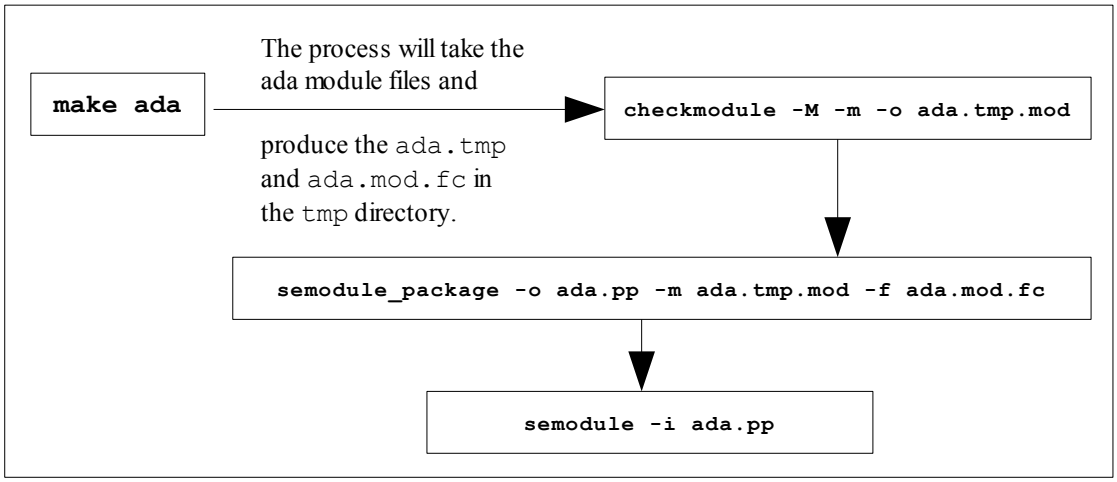


Figure 5.6: The `make ada` sequence of events

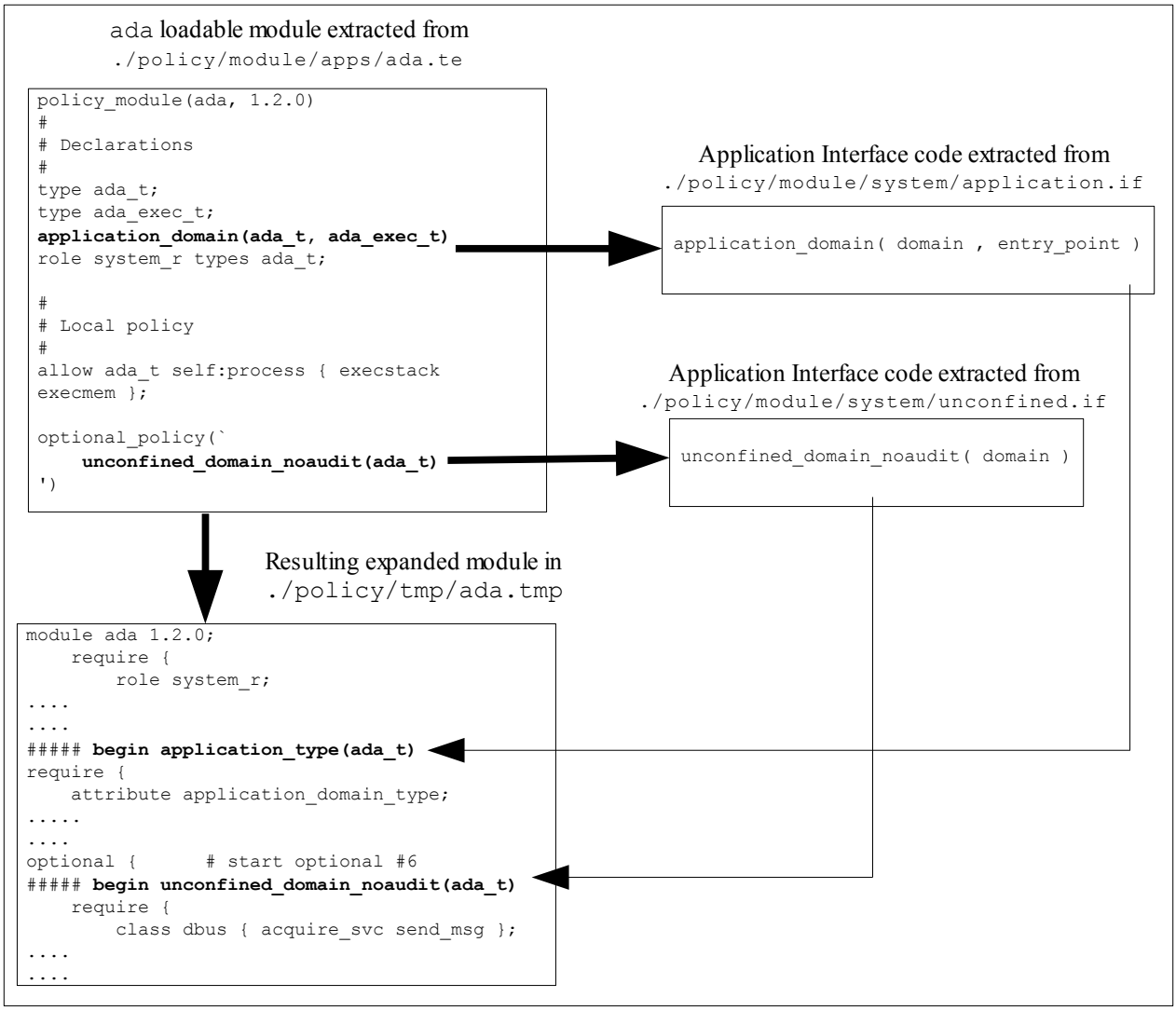


Figure 5.7: The Resulting Code - *The expanded code in the Module Expansion section.*

5.7.1 Module Expansion

The `ada.te` module is expanded as shown below. Note that 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.
#
######## START application_domain(ada_t,ada_exec_t) SEQUENCE ##############
#
##### begin application_type(ada_t)
require {
   attribute application_domain_type;
} # end require

typeattribute ada_t application_domain_type;

##### begin domain_type(ada_t)
##### begin domain_base_type(ada_t)
require {
   attribute domain;
} # end require

typeattribute ada_t domain;
##### end domain_base_type(ada_t)

optional { # start optional #1
   ##### begin unconfined_use_fds(ada_t)
   require {
       type unconfined_t;
   } # end require

   allow ada_t unconfined_t:fd use;
   ##### end unconfined_use_fds(ada_t)
```

```
} # end optional #1


# send init a sigchld and signull
optional { # start optional #2
##### begin init_sigchld(ada_t)
   require {
       type init_t;
   } # end require
   allow ada_t init_t:process sigchld;
##### end init_sigchld(ada_t)

##### begin init_signull(ada_t)
   require {
       type init_t;
   } # end require

   allow ada_t init_t:process signull;
#### end init_signull(ada_t)
} # end optional #2

# these seem questionable:
optional { # start optional #3
##### begin rpm_use_fds(ada_t)
   require {
       type rpm_t;
   } # end require

   allow ada_t rpm_t:fd use;
##### end rpm_use_fds(ada_t)

##### begin rpm_read_pipes(ada_t)
   require {
       type rpm_t;
   } # end require

   allow ada_t rpm_t:fifo_file { getattr read lock ioctl };
##### end rpm_read_pipes(ada_t)
} # end optional #3


optional { # start optional #4
##### begin selinux_dontaudit_getattr_fs(ada_t)
   require {
       type security_t;
   } # end require

   dontaudit ada_t security_t:filesystem getattr;
##### end selinux_dontaudit_getattr_fs(ada_t)

##### begin selinux_dontaudit_read_fs(ada_t)
   require {
       type security_t;
   } # end require

##### begin selinux_dontaudit_getattr_fs(ada_t)
   require {
       type security_t;
   } # end require

   dontaudit ada_t security_t:filesystem getattr;
##### end selinux_dontaudit_getattr_fs(ada_t)
   dontaudit ada_t security_t:dir { getattr search };
   dontaudit ada_t security_t:file { getattr read };
##### end selinux_dontaudit_read_fs(ada_t)
} # end optional #4


optional { # start optional #5
##### begin seutil_dontaudit_read_config(ada_t)
   require {
       type selinux_config_t;
   } # end require
   dontaudit ada_t selinux_config_t:dir { getattr search };
   dontaudit ada_t selinux_config_t:file { getattr read };
```

```
##### end seutil_dontaudit_read_config(ada_t)
} # end optional #5

##### end domain_type(ada_t)
##### end application_type(ada_t)

##### begin application_executable_file(ada_exec_t)
require {
   attribute application_exec_type;
} # end require

typeattribute ada_exec_t application_exec_type;

##### begin corecmd_executable_file(ada_exec_t)
require {
   attribute exec_type;
} # end require

typeattribute ada_exec_t exec_type;

##### begin files_type(ada_exec_t)
require {
   attribute file_type, non_security_file_type;
} # end require

typeattribute ada_exec_t file_type, non_security_file_type;
##### end files_type(ada_exec_t)
##### end corecmd_executable_file(ada_exec_t)
##### end application_executable_file(ada_exec_t)

##### begin domain_entry_file(ada_t,ada_exec_t)
require {
   attribute entry_type;
} # end require
allow ada_t ada_exec_t:file entrypoint;
allow ada_t ada_exec_t:file { { getattr read execute ioctl } ioctl lock };
typeattribute ada_exec_t entry_type;

##### begin corecmd_executable_file(ada_exec_t)
require {
   attribute exec_type;
} # end require
typeattribute ada_exec_t exec_type;

##### begin files_type(ada_exec_t)
   require {
   attribute file_type, non_security_file_type;
} # end require
typeattribute ada_exec_t file_type, non_security_file_type;
##### end files_type(ada_exec_t) depth: 3
##### end corecmd_executable_file(ada_exec_t)
##### end domain_entry_file(ada_t,ada_exec_t)
##### end application_domain(ada_t,ada_exec_t)

#
########## END application_domain(ada_t,ada_exec_t) INSERT ########
#

role system_r types ada_t;

########################################
#
# Local policy
#

# This is the only allow statement in the ada.te file:
allow ada_t self:process { execstack execmem };

#
# The "optional_policy(`unconfined_domain_noaudit(ada_t)')" is the next
# line that expands into the lines between START and END OPTIONAL comments.
# NOTE: If the unconfined module was NOT part of the build then this optional
#       policy section would not be present.
#
################## START OPTIONAL_POLICY STATEMENT ######################
```

```
optional {    # start optional #6
##### begin unconfined_domain_noaudit(ada_t)
   require {
       class dbus { acquire_svc send_msg };
       class nscd { getpwd getgrp gethost getstat admin shmempwd shmemgrp
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) {
##### begin domain_mmap_low(ada_t)
       allow ada_t self:memprotect mmap_zero;
##### end domain_mmap_low(ada_t)
   }

   require {
       bool allow_execheap;
   } # end require

   if (allow_execheap) {
   # Allow making the stack executable via mprotect.
       allow ada_t self:process execheap;
   }

   require {
       bool allow_execmem;
   } # end require

   if (allow_execmem) {
       # Allow making anonymous memory executable, e.g.
       # for runtime-code generation or executable stack.
       allow ada_t self:process execmem;
   }
```

```
   require {
       bool allow_execstack;
   } # end require

   if (allow_execstack) {
       # Allow making the stack executable via mprotect;
       # execstack implies execmem;
       allow ada_t self:process { execstack execmem };
       #
       auditallow ada_t self:process execstack;}


   optional {  # start optional #7
##### begin auth_unconfined(ada_t)
       require {
           attribute can_read_shadow_passwords;
           attribute can_write_shadow_passwords;
           attribute can_relabelto_shadow_passwords;
       } # end require

typeattribute ada_t can_read_shadow_passwords;typeattribute ada_t
can_write_shadow_passwords;typeattribute ada_t can_relabelto_shadow_passwords;
##### end auth_unconfined(ada_t) depth: 1
} # end optional #7

   optional { # start optional #8
# Communicate via 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 }:lnk_file { getattr read };
   allow ada_t lib_t:dir { getattr search };allow ada_t { lib_t
textrel_shlib_t }:file { getattr read execute ioctl };
   allow ada_t textrel_shlib_t:file execmod;
##### end libs_use_shared_libs(ada_t)
   } # end optional #10

   optional {  # start optional #11
##### begin nscd_unconfined(ada_t)
       require {
           type nscd_t;
           class nscd { getpwd getgrp gethost getstat admin shmempwd shmemgrp
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

################## END OPTIONAL_POLICY STATEMENT ######################
#
```

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. Implementing SELinux-aware Applications

6.1 Introduction

The following definitions attempt to explain the difference between the two types of userspace SELinux application (however the distinction can get 'blurred'):

SELinux-aware - Any application that provides support for SELinux. This generally means that the application makes use of SELinux libraries and/or other SELinux applications. Example SELinux-aware applications are the Pluggable Authentication Manager (**PAM**(8)) and SELinux commands such as **runcon**(1). It is of course possible to class an object manager as an SELinux-aware application.

Object Manager - Object Managers are a specialised form of SELinux-aware application that are responsible for the labeling, management and enforcement[63] of the objects under their control.

Generally the userspace Object Manager forms part of an application that can be configured out should the base Linux OS not support SELinux.

Example userspace Object Managers are:

- X-SELinux is an optional X-Windows extension responsible for labeling and enforcement of X-Windows objects.
- Dbus has an optional Object Manager built if SELinux is defined in the Linux build. This is responsible for the labeling and enforcement of Dbus objects.
- SE-PostgreSQL is an optional extension for PostgreSQL that is responsible for the labeling and enforcement of PostgreSQL database and supporting objects.

Therefore the basic distinction is that Object Managers manage their defined objects on behalf of an application, whereas general SELinux-aware applications do not (they rely on 'Object Managers' to do this e.g. the kernel based Object Managers such as those that manage filesystem, IPC and network labeling).

6.2 Types of Object Manager

There are three basic forms of userspace object manager:

1. Those that do not cache access decisions (i.e. they do not use the `libselinux` AVC services). These require a call to the kernel for every decision using **security_compute_av**(3) or **security_compute_av_flags**(3). The avc_netlink_*(3) functions can be used to detect policy change events. Auditing would need to be implemented if required.

[63] The SELinux policy / security server do not themselves enforce a decision, they merely state whether the operation is allowed or not according to the policy. It is the object manager that enforces the decision of the policy / security server, therefore an object manager must be trusted. This is also true of labeling - the object manager ensures that the labels are applied to their objects as defined by the policy.

2. Those that utilise the libselinux userspace AVC services that are initialised with **avc_open**(3). These can be built in various configurations such as:
   a) Using the default single threaded mode where **avc_has_perm**(3) will automatically cache entries, audit the decision and manage the handling of policy change events.
   b) Implementing threads or a similar service that will handle policy change events and auditing in real time with **avc_has_perm**(3) or **avc_has_perm_noaudit**(3) handling decisions and caching. This has the advantage of better performance, which can be further increased by caching the entry reference.
3. Those that build their own caching service and use **security_compute_av**(3) or **security_compute_av_flags**(3) for computing access decisions. The avc_netlink_*(3) functions can then be used to detect policy change events. Auditing would need to be implemented if required.

6.2.1 Implementing SELinux-aware Applications

This section puts forward various points that may be useful when developing SELinux-aware applications and object managers using libselinux.

1. Determine the security objectives and requirements.
2. Because these applications manage labeling and access control, they need to be trusted.
3. Where possible use the libselinux *_raw functions as they avoid the overhead of translating the context to/from the readable format (unless of course there is a requirement for a readable context - see **mcstransd**(8)).
4. Use **selinux_set_mapping**(3) to limit the classes and permissions to only those required by the application.
5. The standard output for messages generated by libselinux functions is stderr. Use **selinux_set_callback**(3) with **SELINUX_CB_LOG** type to redirect these to a log handler.
6. Do not directly reference SELinux configuration files, always use the libselinux path functions to return the location. This will help portability as SELinux has some changes in the pipe-line for the location of the policy configuration files and the SELinux filesystem.
7. Where possible use the selabel_*(3) functions to determine a files default context as they effectively replace the matchpathcon*(3) series of functions - see **selabel_file**(5).
8. Do not use class IDs directly, use **string_to_security_class**(3) that will take the class string defined in the policy and return the class ID/value. Always check the value is > 0. If 0, then signifies that the class is unknown and the **deny_unknown** flag setting in the policy will determine the outcome of any decision - see **security_deny_unknown**(3).

9. Do not use permission bits directly, use **string_to_av_perm**(3) that will take the permission string defined in the policy and return the permission bit mask.
10. Where performance is important (see the Types of Object Manager section) when making policy decisions (i.e. using **security_compute_av**(3), **security_compute_av_flags**(3), **avc_has_perm**(3) or **avc_has_perm_noaudit**(3)), then use the selinux_status_*(3) functions to detect policy updates etc. as these do not require system call overheads once set up. Note that the selinux_status_* functions are only available from libselinux 2.0.99, with Linux kernel 2.6.37 and above.
11. Be aware that applications being built for 32 bit systems need to specify the CFLAG -D_FILE_OFFSET_BITS=64 as libselinux is built with this flag. This is particularly important if **matchpathcon_filespec_add**(3) is used as it passes over **ino_t** ino that is too small otherwise (i.e. needs to be 64 bits).
12. There are changes to the way contexts are computed for sockets in kernels 2.6.39 and above as described in the Computing Contexts section. The functions affected by this are: **avc_compute_create**(3), **avc_compute_member**(3), **security_compute_create**(3), **security_compute_member**(3) and **security_compute_relabel**(3).
13. It is possible to set an undefined file context if the process has **capability**(7) **CAP_MAC_ADMIN** and class **capability2** with **mac_admin** permission in the policy. This is called 'deferred mapping of security contexts' and is explained in **setfilecon**(3) and at:

    http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commit;h=12b29f34558b9b45a2c6eabd4f3c6be939a3980f

6.2.2 Implementing Object Managers

To implement object managers for applications, an understanding of the application is essential, because as a minimum:

- What object types and their permissions are required.
- Where in the code object instances are created.
- Where access controls need to be applied.

While this section cannot help with those points, here are some notes to help during the design phase (also see the Implementing SELinux-aware Applications section):

1. Determine what objects are required and the access controls (permissions) that need to be applied.
2. Does SELinux already have some of these object classes and permissions defined. For standard Linux OS objects such as files, then these would be available. If so, the object manager should remap them with **selinux_set_mapping**(3) so only those required are available.

However, do not try to reuse a current object that may be similar to the requirements, it will cause confusion at some stage. Always generate new classes/permissions.

3. If the application has APIs or functions that integrate with other applications or scripts, then as part of the object manager implementation these may need to support the use of security contexts (examples are X-Windows and SE-PostgreSQL that provide functions for other applications to use). Therefore if required, provide common functions that can be used to label the objects.
4. Determine how the initial objects will be labeled. For example will a configuration file be required for default labels, if so how will this be introduced into the SELinux userspace build. Examples of these are the X-Windows (**selabel_x**(5)), SE-PostgreSQL (**selabel_db**(3)), and file context series of files (**selabel_file**(5)).
5. Will the labeling need to be persistent across policy and system reloads or not. X-Windows is an example of a non-persistent, and SE-PostgreSQL is an example of a persistent object manager.
6. Will support for the standard audit log or its own be required (the libselinux functions default to stderr). Use **selinux_set_callback**(3) to manage logging services.
7. Decide whether an AVC cache is required or not. If the object manager handles high volumes of requests then an AVC will be required. See the Types of Object Manager section for details.
8. Will the object manager need to do additional processing when policy or enforcement changes are detected. This could be clearing any caches or resetting variables etc.. If so, then **selinux_set_callback**(3) will be used to set up these functions. These events are detected via the **netlink**(7) services, see **avc_open**(3) and **avc_netlink_open**(3) for the various options available.
9. If possible implement a service like XACE for the application, and use it to interface with the applications SELinux object manager. The XACE interface acts like the LSM which supports SELinux as well as other providers such as SMACK. The XACE interface is defined in the "X Access Control Extension Specification", and for reference, the SE-PostgreSQL service also implements a similar interface.

   The XACE specification is available from:

   http://www.x.org/releases/X11R7.5/doc/security/XACE-Spec.pdf

6.2.3 Reference Policy Changes

When adding a new object manager to SELinux, it will require at least a new policy module to be added. This section assumes that the SELinux Reference Policy is in use and gives some pointers, however any detail is beyond the scope of this man page. Further information can be found at the follwing:

http://oss.tresys.com/projects/refpolicy

http://selinuxproject.org.

The latest Reference Policy source can be obtained as follows:

```
git clone http://oss.tresys.com/git/refpolicy.git
```

The main points to note when adding to the Reference Policy are:

1. Create sample Reference Policy policy modules (*.te, *.if and *.fc module files) that provide rules for managing the new objects as described in:

   http://selinuxproject.org/page/NB_RefPolicy#Reference_Policy_Module_Files

   The SE-PostgreSQL modules provide an example, see the ./refpolicy/policy/modules/services/postgresql.* files in the Reference Policy source.

2. Create any new policy classes and permissions for the Reference Policy, these will need to be built into the base module as described in the Adding New Object Classes and Permissions section.

   Note, that if no new object classes, permissions or constraints are being added to the policy, then the Reference Policy source code does not require modification, and supplying the module files (*.te, *.if and *.fc) should suffice.

3. Create any constraints required as these need to be built into the base module of the Reference Policy. They are added to the ./refpolicy/policy/constraints, mcs and mls files. Again the SE-PostgreSQL entries in these files give examples (find the db_* class entries).

4. Create any SELinux configuration files (context, user etc.) that need to be added to the policy at build time.

5. Either produce an updated Reference Policy source or module patch, depending on whether new classes/constraints have been added. Note that by default a new module will be generated as a 'module', if it is required that the module is in the base (unusual), then add an entry **<required val='true'>** to the start of the interface file as shown below:

   ```
   ## <summary>
   ##Comment regarding interface file
   ## </summary>
   ## <required val="true">
   ##Comment on reason why required in base
   ## </required>
   ```

6.2.4 Adding New Object Classes and Permissions

Because userspace object managers do not require their new classes and permissions to be built into the kernel, the configuration is limited to the actual policy (generally the Reference Policy) and the application object manager code. New classes are added to the Reference Policy security_classes file and permissions to the access_vectors file.

The class configuration file is at:

./refpolicy/policy/flask/security_classes

and each entry must be added to the end of the file in the following format:

```
class object_name    # userspace
```

Where **class** is the class keyword and object_name is the name of the object. The **# userspace** is used by build scripts to detect userspace objects (or more specifically 'non-mandatory' objects).

The permissions configuration file is at:

./refpolicy/policy/flask/access_vectors

and each entry must be added to the end of the file in the following format:

```
class object_name
{
  perm_name
  [........]
}
```

Where **class** is the class keyword, object_name is the name of the object and perm_name is the name given to each permission in the class (there is a limit of 32 permissions within a class). It is possible to have a common permission section within this file, see the file object entry in the access_vectors file for an example.

For reference, http://selinuxproject.org/page/Adding_New_Permissions describes how new kernel object classes and permissions are added to the system.

7. SEAndroid

This section gives a brief overview of SEAndroid as it stood in August '12 as this is a new project and continually being enhanced (Android 4.1.1 with SEAndroid installs on F-17 with no issues). It is recommended that the project web site is checked for the latest enhancements: http://selinuxproject.org/page/SEAndroid.

There is a presentation 'The Case for Security Enhanced (SE)Android' [Ref 22] that gives the rationale behind the enhancements.

The http://en.wikipedia.org/wiki/Android_%28operating_system%29 site gives a good introduction to Android and http://source.android.com gives details on installation of the source.

7.1.1 Overview

SEAndroid enhances the Android system by adding SELinux support to the kernel and userspace with the main objectives being to (taken from http://selinuxproject.org/page/SEAndroid):

1. Confine privileged daemons to protect them from misuse and limit the damage that can be done via them.
2. Sandbox and isolate apps from each other and from the system, prevent privilege escalation by apps.
3. Allow application privileges to be controlled at installation and run-time (the Middleware-MAC)
4. Provide a centralized, analyzable policy.

These objectives are achieved by:

- Per-file security labeling support for yaffs2
- Filesystem images (yaffs2 and ext4) labeled at build time
- Labeling support in the recovery console and updater program
- Kernel permission checks controlling Binder IPC
- Labeling of service sockets and socket files created by init
- Labeling of device nodes created by ueventd
- Flexible, configurable labeling of apps and app data directories
- Minimal port of SELinux userspace
- SELinux support for the Android toolbox
- JNI bindings for SELinux APIs
- Userspace permission checks controlling use of the Zygote socket commands
- Userspace permission checks controlling setting of Android properties
- Small TE policy written from scratch for Android
- Confined domains for system services and apps

- Use of MLS categories to isolate apps

The Android git repositories can be found at https://android.googlesource.com and the SEAndroid enhancements at https://bitbucket.org/seandroid.

But do read the installation information at http://selinuxproject.org/page/SEAndroid first.

7.1.2 SEAndroid Project Updates

This gives a high level view of the new and updated projects to support the SEAndroid build:

`external/libselinux`

Provides the SELinux userspace function library that is installed on the device. It is based on a version of the Linux library but has additional functions to support Android. Currently these changes have not been added to the Linux version. The additional functions are:

`selinux_android_setcontext`

Sets the correct domain context for applications using **`setcon`**`(3)`. Information contained in the `seapp_context` file is used to compute the correct context (see `selinux_android_setcontext`). It is called by `dalvik/vm/native/dalvik-system-Zygote.cpp`

`selinux_android_setfilecon`

Used to set the correct context on application directory/files using **`setfilecon`**`(3)`. Information contained in the `seapp_context` file is used to compute the correct context (see `selinux_android_setfilecon`).

Used by `frameworks/base/cmds/installd/commands.c` for package `install`, `make_user_data` and `protect` functions.

`selinux_android_seapp_context_reload`

Reloads the `seapp_context` file and sorts the entries in order of precedence as discussed in the `seapps_context` File section.

`selinux_android_restorecon`

Set file contexts to match entries defined in the `file_contexts` file using **`lsetfilecon`**`(3)`. Used by various commands to reset contexts during initialisation, installation etc.

There is also a new labeling service for **`selabel_lookup`**`(3)` to query the `property_contexts` file (see `label_android_property.c`). This file is loaded at init time (see `system/core/init/init.c`) and used by `system/core/init/property_service.c`. that checks property MAC permissions at system initialisation time (class: `property`, permission: `set`).

`external/libsepol`

Provides the policy userspace library. There are no specific updates to support SEAndroid, also this library is not available on the device.

`external/checkpolicy`

Provides the policy build tool. There are no specific updates to support SEAndroid, also this is not available on the device (therefore policy rebuilds must be done in the development environment.

`external/sepolicy`

This is a new policy specifically for SEAndroid. It looks much like the reference policy but is contained in one directory that has the policy modules (`*.te` files), class / permission files (`security_classes` etc.), and other configuration files and macros. It is built by the `Android.mk` file and the resulting policy is installed on the target device (as `sepolicy.24`).

There are three new object classes defined for the policy that are described in the SEAndroid Classes & Permissions section.

The configurable policy files are discussed in the Policy Configuration Files section.

`external/mac-policy`

Contains the middle-ware MAC policy files and tools. The http://selinuxproject.org/page/SEAndroid#Middleware_MAC section contains details.

`packages/apps/SEAndroidManager`

This is an Android application to manage the SEAndroid environment. It allows control of the enforcement modes, booleans, view logs etc.

`bionic`

Bionic is the Android `libc` that is a derived from the BSD standard C library code. It contains enhancements to support security providers such as SELinux.

`bootable/recovery`

Changes to manage file labeling on recovery.

`build`

Changes to manage file labeling on images and OTA (over the air) target files.

`dalvik`

Set the context (using **`selinux_android_setcontext`**`(3)`) on the dalvik process being forked.

`external/yaffs2`

`mkyaffs2image` support for labeling and extended attributes (`xattr`)

`libcore`

Add additional parameters `seInfo` and `niceName` to `Zygote.java`

`frameworks/base`

JNI - Add SELinux support functions such as `isSELinuxEnabled` and `setFSCreateCon`.

SELinux Java class and method definitions.

Checking Zygote connection contexts.

Managing file permissions for the package manager and wallpaper services.

`system/core`

SELinux support services for toolbox (e.g. `load_policy`, `runcon`).

SELinux support for system initialisation (e.g. `init`, `init.rc`).

`system/extras`

SELinux support for the `ext4` file system.

`packages/apps/Settings`

SELinux settings for the settings manager application.

7.1.2.1 Kernels

The following Android kernels have been enhanced to configure security/SELinux services and extend LSM/SELinux to support the Android Binder IPC service.

`kernel/goldfish` - Emulator platforms

`kernel/samsung` - Nexus S and Samsung Hummingbird chipsets

`kernel/tegra` - Xoom and NVIDIA Tegra chipsets

`kernel/omap` - PandaBoard, Galaxy Nexus and TI OMAP chipsets.

Note that as the Android kernels are based on various versions - 2.6.29 for the emulator and 3.0 for omap and Samsung, therefore any additional SELinux enhancements will not have taken place and will need to be added.

7.1.2.2 Devices

These contain updated board configurations to enable SELinux for the following devices:

`device/samsung/crespo` - Google Nexus S

`device/samsung/crespo4g` - Google Nexus S 4G

`device/samsung/tuna` - Common Samsung Galaxy Nexus files

`device/samsung/maguro` - Galaxy Nexus (GSM/HSPA+) specific files

`device/samsung/toro` - Galaxy Nexus (CDMA/LTE) specific files

`device/moto/wingray` - Motorola Xoom (Wifi)

There are more details at http://selinuxproject.org/page/SEAndroid#Building_for_a_Device regarding configuration information.

7.2 Policy Configuration Files

These are contained in the `external/sepolicy` directory, with some being copied over to the device as part of the build process.

The `access_vectors` and `security_classes` files have been modified to support the new SEAndroid classes / permissions. The `initial_sids` file is standard. To allow the policy to be built easier the contexts for initial SIDs, ports, genfs etc. are defined in separate files and then linked together during the build. It is also possible to locate policy files within the device directories as discussed below, to allow device specific configuration. The mls and macro files are specific to SEAndroid. The remaining files are as follows:

`*.te` and `*.fc` - The policy module definition files. These are the same format as the standard reference policy. The `Android.mk` file with the **`checkpolicy`**`(8)` command are used to build the resulting kernel policy file that will be copied to the root directory as `sepolicy.24`, but may also be in `/data/system`. The policy file may be examined by the standard setools.

`file_contexts` - Contains default contexts for setting the filesystem as standard SELinux. This file will be copied to the root directory, but may also be in `/data/system`.

`property_contexts` - Contains default contexts to be applied to Android property services as discussed in the `property_context` File section. This file will be copied to the root directory, but may also be in `/data/system`.

`seapps_context` - Contains information to allow domain or file contexts to be computed based on parameters as discussed in the `seapps_context` File section. This file will be copied to the root directory, but may also be in `/data/system`.

`selinux-network.sh` - If using **`iptables`**`(8)` then SECMARK information may be configured in this file as part of the build. It is installed in `system/bin` and executed at system initialisation time.

`mac_permissions.xml` - This is installed in `/system/etc/security` and is discussed in the `mac_permissions.xml` File section.

Certain policy files may also be present in device directories where they would contain device specific information. The `Andoid.mk` file would add these to the main policy files. The device specific files currently allowed are:

`sepolicy.te`, `sepolicy.fc`, `sepolicy.pc`, `sepolicy.fsuse`, `sepolicy.port_contexts`, `sepolicy.genfs`, `sepolicy.init_sids`, `system.prop`

If the policy needs to be updated, the following process can be used (assumes that the Android build environment is set-up correctly):

1. Modify the required policy source files, then regenerate the kernel policy file by:

   ```
   make sepolicy
   ```

2. Copy the policy file to the device:

```
adb push out/target/product/<device>/root/sepolicy.24 /data/system/
```

3. Then load the new policy by[64]:

```
adb shell su 0 setprop selinux.reloadpolicy 1
```

7.2.1 seapps_context File

This file is loaded and sorted into memory by **selinux_seapp_context_reload**(3) using the following precedence rules:

1. **isSystemServer=** true before **isSystemServer=** false.
2. Specified **user=** string before unspecified **user=** string.
3. Fixed **user=** equal string before **user=** prefix (i.e. ending in *****).
4. Longer **user=** prefix before shorter **user=** prefix.
5. Specified **seinfo=** string before unspecified **seinfo=** string.
6. Specified **name=** string before unspecified **name=** string.
7. Specified **sebool=** string before unspecified sebool= string.

The entries can now be used by **selinux_android_setcontext**(3) and **selinux_android_setfilecon**(3) to compute process and file contexts as described in the sections that follow.

7.2.1.1 selinux_android_setcontext

Set the security context for an Android application domain. This function is used by the Dalvik / Zygote services to start applications in their specified domain.

Synopsis

```
#include <selinux/android.h>

int selinux_android_setcontext(uid_t uid, int isSystemServer,
    const char *seinfo, const char *name);
```

Description

selinux_android_setcontext sets the security context using **setcon**(3). The computed context is validated against policy and is based on the applications username (obtained from uid), seinfo, and the (package) name information.

The username is validated against rules, and then used with the other parameters to compute a context as described in the **Context Computation** section.

The isSystemServer value must match that set in the running system and be true or false.

seinfo and name may be **NULL**.

[64] The 'setprop selinux.reloadpolicy 1' forces a policy reload (see system/core/init/property_service.c - property_set()). The policy load function will then check for a policy file in the /data/system directory first (the root directory will be checked next if a policy file is not found).

If a context cannot be computed an error is returned and an entry added to the error log.

See the **Examples** section for a selection of computed contexts based on various parameters.

Context Computation

Using the 'in memory' **seapp_context** file entries sorted in order of precedence, the username is checked according to the following rules to determine whether the application is named (e.g radio), is a system app, or a standard app, and what level may need to be applied:

- IF the first four chars of the username is '**app_**', and remainder numeric, calculate the category based on these numerics.

  Example username: **app_123**

- ELSE IF the first char of the username is '**u**' and the second char is numeric, and it ends in numerics, calculate the category using these numerics until an '**a**' is found. Finally reset the username to '**app_**'.

  Example username: **u0_a36**

- ELSE keep username intact. The category is set to '**0**'.

- Using the isSystemServer, username, seinfo and name parameters, find a matchingentry (logical AND) within the 'in memory' sorted version of the **seapp_context**(5) file using the following rules:

  1. isSystemServer != current value
  2. Does username match the **user=** entry
  3. Does seinfo match the **seinfo=** entry (may also be NULL)
  4. Does name match the **name=** entry (may also be NULL)
  5. Is there a **domain=** entry, if so save as the type component of the context.
  6. Is there an **sebool=** entry, if so the specified boolean must be 'active' (enabled/true).
  7. IF **levelFromUid=true** use category from username check to form the level, ELSE use the specified level.

- IF all parameters match, build context and check if valid. If valid call **setcon**(3) and return, ELSE raise error, log and return.

The **Examples** section shows a selection of computed contexts.

Return Value

On success, zero is returned.

On failure, -1 is returned and errno is set appropriately.

Error Log Entries

Errors will be logged in the error log when:

i. No match can be found in the **seapp_context**(5) file.

ii. If isSystemServer is true, a message stating that setting the system server context failed.

iii. If isSystemServer is false, then a message stating that setting an application context failed.

iv. If there is an **sebool=** entry defined, but no boolean of that name is present.

Files

seinfo is defined in the **mac_permissions.xml**(5) file.

name is defined in the packages AndroidManifest.xml file.

The context is computed using information from the **seapp_context**(5) file.

Examples

The **seapp_context** file used is:

```
isSystemServer=true domain=system
user=system domain=system_app type=system_data_file
user=nfc domain=nfc type=nfc_data_file
user=radio domain=radio type=radio_data_file
user=app_* domain=untrusted_app type=app_data_file levelFromUid=true
user=app_* seinfo=platform domain=platform_app levelFromUid=true
user=app_* seinfo=shared domain=shared_app levelFromUid=true
user=app_* seinfo=media domain=media_app levelFromUid=true
user=app_* seinfo=release domain=release_app levelFromUid=true
user=app_* seinfo=release name=com.android.browser domain=browser_app levelFromUid=true
```

The following is an example taken as the system server is loaded:

```
uid             1000
isSystemServer 1
seinfo          (null)
name            (null)
sebool          (null)

username from uid system
Computed username system

Computed context u:r:system:s0
```

This is the'radio' application that is part of the platform:

```
uid             1001
isSystemServer 0
seinfo          platform
name            com.android.phone
sebool          (null)

username from uid radio
Computed username radio

Computed context u:r:radio:s0
```

This is the'SEAndroid Manager' application that is part of the platform:

```
uid             1000
isSystemServer 0
seinfo          platform
name            com.android.seandroid_manager
sebool          (null)

username from uid system
Computed username system

Computed context u:r:system_app:s0
```

This is the'Desk Clock' application that is part of the release:

```
uid             10036
isSystemServer 0
seinfo          release
name            com.android.deskclock
sebool          (null)

username from uid u0_a36
Computed username app_

Computed context u:r:release_app:s0:c36
```

These are example process contexts taken from the '`ps -Z`' command:

```
# ps -Z
LABEL                           USER     PID   PPID  NAME
u:r:init:s0                     root      1     0     /init
..
u:r:ueventd:s0                  root      28    1     /sbin/ueventd
..
u:r:kernel:s0                   root      31    2     yaffs-bg-1
..
u:r:servicemanager:s0           system    33    1     /system/bin/servicemanager
u:r:vold:s0                     root      34    1     /system/bin/vold
u:r:netd:s0                     root      36    1     /system/bin/netd
u:r:debuggerd:s0                root      37    1     /system/bin/debuggerd
u:r:rild:s0                     radio     38    1     /system/bin/rild
u:r:surfaceflinger:s0           system    39    1     /system/bin/surfaceflinger
u:r:zygote:s0                   root      40    1     zygote
u:r:drmserver:s0                drm       41    1     /system/bin/drmserver
u:r:mediaserver:s0              media     42    1     /system/bin/mediaserver
u:r:dbusd:s0                    bluetooth 43    1     /system/bin/dbus-daemon
u:r:installd:s0                 root      44    1     /system/bin/installd
u:r:keystore:s0                 keystore  45    1     /system/bin/keystore
u:r:qemud:s0                    root      48    1     /system/bin/qemud
u:r:system:s0                   system    89    40    system_server
..
u:r:radio:s0                    radio     179   40    com.android.phone
..
u:r:system_app:s0               system    220   40    com.android.settings
u:r:shared_app:s0:c3            app_3     242   40    android.process.acore
u:r:release_app:s0:c28          app_28    302   40    com.android.deskclock
u:r:shared_app:s0:c3            app_3     325   40    com.android.contacts
u:r:system_app:s0               system    362   40    com.android.seandroid_manager
```

7.2.1.2 `selinux_android_setfilecon`

Set the security context on an Android application package directory. This function is primarily used by the package installer to label application directories.

Synopsis

```
#include <selinux/android.h>

int selinux_android_setfilecon(const char *pkgdir,
    const char *pkgname, uid_t uid);
```

Description

selinux_android_setfilecon sets the security context on the file object using **setfilecon**(3). The computed context is validated against policy and is based on the applications username (obtained from uid) and pkgname. The pkgdir will be labeled with the computed context.

The username is validated against rules, and then used with the other parameters to compute a context as described in the **Context Computation** section.

pkgname may be **NULL**.

If a context cannot be computed an error is returned and an entry added to the error log.

Context Computation

Using the 'in memory' **seapp_context** file entries sorted in order of precedence, the username is checked according to the following rules to determine whether the application is named (e.g radio), is a system app, or a standard app, and what level may need to be applied:

- IF the first four chars of the username is '**app_**', and remainder numeric, calculate the category based on these numerics.

  Example username: **app_123**

- ELSE IF the first char of the username is '**u**' and the second char is numeric, and it ends in numerics, calculate the category using these numerics until an '**a**' is found. Finally reset the username to '**app_**'.

  Example username: **u0_a36**

- ELSE keep username intact. The category is set to '**0**'.

- Using the username and pkgname parameters, find a matchingentry (logical AND) within the 'in memory' sorted version of the **seapp_context**(5) file using the following rules:

  1. isSystemServer false
  2. Does username match the **user=** entry
  3. Does pkgname match the **name=** entry (may also be NULL)
  4. Is there a **type=** entry, if so save as the type component of the context.
  5. Is there an **sebool=** entry, if so the specified boolean must be 'active' (enabled/true).
  6. IF **levelFromUid=true** use category from username check to form the level, ELSE use the specified level.

- IF all parameters match, build context and check if valid. If valid call **setfilecon**(3) with pkgdir and return, ELSE raise error, log and return.

Return Value

On success, zero is returned.

On failure, -1 is returned and errno is set appropriately.

Error Log Entries

Errors will be logged in the error log when:

i. No match can be found in the **seapp_context**(5) file.

ii. If isSystemServer is true, a message stating that setting the system server context failed.

iii. If isSystemServer is false, then a message stating that setting an application context failed.

iv. If there is an **sebool=** entry defined, but no boolean of that name is present.

Files

pkgname is defined in the packages AndroidManifest.xml file.

The context is computed using information from the **seapp_context**(5) file.

Examples

These are sample application directory (data) contexts taken from /data/data using the 'ls -Z' command:

```
# ls -Z /data/data
drwxr-x--x u0_a14   u0_a14   u:object_r:platform_app_data_file:s0 com.android.bluetooth
drwxr-x--x u0_a3    u0_a3    u:object_r:platform_app_data_file:s0 com.android.contacts
drwxr-x--x u0_a35   u0_a35   u:object_r:platform_app_data_file:s0 com.android.defcontainer
drwxr-x--x u0_a36   u0_a36   u:object_r:platform_app_data_file:s0 com.android.deskclock
drwxr-x--x system   system   u:object_r:system_data_file:s0 com.android.inputdevices
drwxr-x--x u0_a39   u0_a39   u:object_r:platform_app_data_file:s0 com.android.packageinstaller
drwxr-x--x radio    radio    u:object_r:radio_data_file:s0 com.android.phone
drwxr-x--x u0_a3    u0_a3    u:object_r:platform_app_data_file:s0 com.android.providers.applications
drwxr-x--x u0_a26   u0_a26   u:object_r:platform_app_data_file:s0 com.android.providers.calendar
drwxr-x--x u0_a3    u0_a3    u:object_r:platform_app_data_file:s0 com.android.providers.contacts
drwxr-x--x u0_a1    u0_a1    u:object_r:platform_app_data_file:s0 com.android.providers.downloads
drwxr-x--x u0_a1    u0_a1    u:object_r:platform_app_data_file:s0 com.android.providers.downloads.ui
drwxr-x--x u0_a1    u0_a1    u:object_r:platform_app_data_file:s0 com.android.providers.drm
drwxr-x--x u0_a1    u0_a1    u:object_r:platform_app_data_file:s0 com.android.providers.media
drwxr-x--x system   system   u:object_r:system_data_file:s0 com.android.providers.settings
drwxr-x--x radio    radio    u:object_r:radio_data_file:s0 com.android.providers.telephony
drwxr-x--x system   system   u:object_r:system_data_file:s0 com.android.seandroid_manager
drwxr-x--x system   system   u:object_r:system_data_file:s0 com.android.settings
drwxr-x--x u0_a4    u0_a4    u:object_r:platform_app_data_file:s0 com.android.systemui
drwxr-x--x u0_a11   u0_a11   u:object_r:platform_app_data_file:s0 com.android.videoeditor
```

7.2.2 property_context File

This file holds property names and their contexts that will be applied by SELinux when applications are loaded. The property names reflect the 'white list' of Android property entries that are also built into the system (see system/core/init/property_service.c), however there are also additional property entries for applications that require specific contexts to be set.

When **selabel_open**(3) is called specifying this file it will be read into memory and sorted using **qsort**(3), subsequent calls using **selabel_lookup**(3) will then retrieve the appropriate context.

Each line within the property contexts file is as follows:

```
property_key  context
```

Where:

property_key

The key used to obtain the context and may contain '*' for wildcard matching or '?' for substitution.

context

The security context that will be applied to the object.

Example:

```
##########################
# property service keys
#
#
# property_key                    context
net.rmnet0                       u:object_r:radio_prop:s0
net.gprs                         u:object_r:radio_prop:s0
sys.usb.config                   u:object_r:radio_prop:s0
ril.                             u:object_r:rild_prop:s0
net.                             u:object_r:system_prop:s0
service.                         u:object_r:system_prop:s0
debug.                           u:object_r:shell_prop:s0
log.                             u:object_r:shell_prop:s0
service.adb.tcp.port             u:object_r:shell_prop:s0
persist.sys.                     u:object_r:system_prop:s0
persist.service.                 u:object_r:system_prop:s0
persist.security.                u:object_r:system_prop:s0
selinux.                         u:object_r:system_prop:s0

# mac middleware property
persist.mac_enforcing_mode       u:object_r:system_prop:s0
persist.tagprop_enforcing_mode   u:object_r:system_prop:s0

# default property context
*                                u:object_r:default_prop:s0
```

7.2.3 mac_permissions.xml File

This file provides two main functions:

1. x.509 certificate to seinfo string mapping so that (using the seapp_contexts file), zygote spawns the application in the correct domain. See the **selinux_android_setcontext** section for how this is achieved using information from the seapp_contexts file.
2. Install time Android permission checking.

The following rules have been extracted from the source `mac_permissions.xml` file, it is recommended that this file is consulted for a working example that is based on the AOSP developer keys.

The `mac_permissions.xml` rules are as follows:

1. A signature is a hex encoded X.509 certificate and is required for each signer tag.
2. A `<signer signature="" >` element may have multiple child elements:

   `allow-permission` : produces a set of maximal allowed permissions (whitelist).

   `deny-permission` : produces a blacklist of permissions to deny.

   `allow-all` : a wildcard tag that will allow every permission requested.

   `package` : a complex tag which itself defines allow, deny, and wildcard sub elements for a specific package name protected by the signature.
3. Zero or more global `<package name="">` tags are allowed. These tags allow a policy to be set outside any signature for specific package names.
4. Unknown tags at any level are skipped.
5. Zero or more signer tags are allowed.
6. Zero or more package tags are allowed per signer tag.
7. A `<package name="">` tag may not contain another `<package name="">` tag. If found, it's skipped.
8. A `<default>` tag is allowed that can contain install policy for all apps not signed with a previously listed cert and not having a per package global policy.
9. When multiple sub elements appear for a tag the following logic is used to ultimately determine the type of enforcement:
   a) A blacklist is used if at least one `deny-permission` tag is found
   b) A whitelist is used if not a blacklist and at least one `allow-permission` tag is found
   c) A wildcard (accept all permission) policy is used if not a blacklist and not a whitelist and at least one `allow-all` tag is present.
   d) If a `<package name="">` sub element is found then that sub element's policy is used according to the above logic and overrides any signature global policy type.
   e) In order for a policy stanza to be enforced at least one of the above situations must apply. Meaning, empty `signer`, `default` or `package` tags will not be accepted.
10. Each `signer` / `default` / global `package` tag is allowed to contain one `<seinfo value=""/>` tag. This tag represents additional information that each appplication can use in setting a SELinux security context on the

eventual process. Any `<seinfo value=""/>` tag found as a child of a `<package name="">` tag which is protected (sub element of `signer` or the `default` tag) is ignored. It's possible that multiple `seinfo` tags are relevant for one application. In the event that this happens, the `seinfo` tag that will be applied is the one for which the corresponding policy stanza is used in the policy decision.

11. Strict enforcing of any xml stanza is not enforced in most cases. This mainly applies to duplicate tags which are allowed. In the event that a tag already exists, the original tag is replaced.
12. There are also no checks on the validity of permission names. Although valid android permissions are expected, nothing prevents unknowns.
13. Enforcement decisions:
    a) All signatures used to sign an application are checked for policy according to signer tags. Only one of the signature policies has to pass however.
    b) In the event that none of the signature policies pass, or none even match, then a global `package` policy is sought. If found, this policy mediates the install.
    c) The `default` tag is consulted last if needed.
    d) A local package policy always overrides any parent policy.
    e) If none of the cases apply then the application is denied.

An example global package policy is as follows:

```
<package name="com.foo.com">
<allow-permission name="android.permission.INTERNET" />
<allow-permission
name="android.permission.WRITE_EXTERNAL_STORAGE" />
<allow-permission name="android.permission.ACCESS_NETWORK_STATE" />
</package>
```

7.3 SEAndroid Classes & Permissions

These are the additional class / permissions for SEAndroid, the policy

Class	**`binder`** – This is a kernel object to manage the Binder IPC service.
Permission	**Description** (5 unique permissions)
`impersonate`	Not currently used in policy but kernel (`selinux/hooks.c`) checks permission in `selinux_binder_transaction` call.
`call`	Get binder references.
`set_context_mgr`	Allow setting of contexts. This is used by the SEAndroid service manager that acts as a context manager for Binder. See the `servicemanager.te` file.
`transfer`	Transfer binder references
`receive`	Receive binder references

Class	**zygote** – This is a userspace object to manage the Android application loader. See Java `SELinux.checkSELinuxAccess`. In `ZygoteConnection.java`
Permission	**Description** (5 unique permissions)
`specifyids`	Peer may specify uid's or gid's
`specifyrlimits`	Peer may specify rlimits.
`specifycapabilities`	Peer may specify capabilities
`specifyinvokewith`	Peer may specify `--invoke-with` to launch Zygote with a wrapper command.
`specifyseinfo`	Specify that `seinfo` may be used

Class	**property_service** – This is a userspace object to manage the Android Property Service.
Permission	**Description** (1 unique permission)
`set`	Set a property

8. Appendix A - Object Classes and Permissions

8.1 Introduction

This section contains a list of object classes and their associated permissions that have been taken from the Fedora F-17 policy sources. There are also additional entries for Xen, however the SEAndroid entries are shown in the SEAndroid Classes & Permissions section.

All objects are kernel objects unless marked as user space objects.

In most cases the permissions are self explanatory as they are those used in the standard Linux function calls (such as 'create a socket' or 'write to a file'). Some SELinux specific permissions are:

`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 location and would probably be more up-to-date:

http://selinuxproject.org/page/ObjectClassesPerms

8.2 Defining Object Classes and Permissions

The Reference Policy already contains the default object classes and permissions required to manage the system and supporting services.

For those who write or manager SELinux policy, there is no need to define new objects and their associated permissions as these would be done by those who actually design and/or write object managers.

The Object Classes and Permissions sections explain how these are defined within the SELinux Policy Language.

8.3 Common Permissions

8.3.1 Common File Permissions

Table 28 describes the common file permissions that are inherited by a number of object classes.

Permissions	**Description** (19 permissions)
`append`	Append to file.
`audit_access`	The rules for this permission work as follows: If a process calls `access()` or `faccessat()` and SELinux denies their request there will be a check for a `dontaudit` rule on the `audit_access` permission. If there is a `dontaudit` rule on `audit_access` an AVC event will not be written. If there is no `dontaudit` rule an AVC event will be written for the permissions requested (`read`, `write`, or `exec`). Notes: 1) There will never be a denial message with the `audit_access` permission as this permission does not control security decisions. 2) `allow` and `auditallow` rules with this permission are therfore meaningless, however the kernel will accept a policy with such rules, but they will do nothing.
`create`	Create new file.
`execute`	Execute the file with domain transition.
`execmod`	Make executable a file that has been modified by copy-on-write.
`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 28: Common File Permissions

8.3.2 Common Socket Permissions

Table 29 describes the common socket permissions that are inherited by a number of object classes.

Permissions	**Description** (22 Permissions)
`accept`	Accept a connection.
`append`	Write or append socket contents

bind	Bind to a name.
connect	Initiate a connection.
create	Create new socket.
getattr	Get socket information.
getopt	Get socket options.
ioctl	Get and set attributes via 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 29: Common Socket Permissions

8.3.3 Common IPC Permissions

Table 30 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 30: Common IPC Permissions

8.3.4 Common Database Permissions

Table 31 describes the common database permissions that are inherited by a number of object classes. The "Security-Enhanced PostgreSQL Security Wiki" [Ref. 3] explains the objects, their permissions and how they should be used in detail.

Permissions	**Description** (6 Permissions)
`create`	Create a database object such as a '`TABLE`'.
`drop`	Delete (`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 31: Common PostgreSQL Database Permissions

8.3.5 Common X_Device Permissions

Table 32 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 32: Common X_Device Permissions

8.4 File Object Classes

Class	**filesystem** – A mounted filesystem
Permissions	**Description** (10 unique permissions)
associate	Use type as label for file.
getattr	Get file attributes.
mount	Mount filesystem.
quotaget	Get quota information.
quotamod	Modify quota information.
relabelfrom	Change the security context based on existing type.
relabelto	Change the security context based on the new type.
remount	Remount existing mount.
transition	Transition to a new SID (change security context).
unmount	Unmount filesystem.

Class	**dir** - Directory
Permissions	**Description** (Inherit 19 common file permissions + 6 unique)
Inherit Common File Permissions	append, audit_access, create, execute, execmod, 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 19 common file permissions + 3 unique)
Inherit Common File Permissions	append, audit_access, create, execute, execmod, getattr, ioctl, link, lock, mounton, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
entrypoint	Entry point permission for a domain transition.
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 19 common file permissions + 1 unique)
Inherit Common File Permissions	append, audit_access, create, execute, execmod, 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	**chr_file** – Character files
Permissions	**Description** (Inherit 19 common file permissions + 3 unique)
Inherit Common File Permissions	append, audit_access, create, execute, execmod, getattr, ioctl, link, lock, mounton, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write
entrypoint	Entry point permission for a domain transition.
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 19 common file permissions + 1 unique)
Inherit Common File Permissions	append, audit_access, create, execute, execmod, 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 19 common file permissions + 1 unique)
Inherit Common File Permissions	append, audit_access, create, execute, execmod, 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	**fifo_file** – Named pipes
Permissions	**Description** (Inherit 19 common file permissions + 1 unique)
Inherit Common File Permissions	append, audit_access, create, execute, execmod, 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.

8.5 Network Object Classes

Class	**node** – IP address or range of IP addresses
Permissions	**Description** (11 unique permissions)

dccp_recv	Allow Datagram Congestion Control Protocol receive packets.
dccp_send	Allow Datagram Congestion Control Protocol send packets.
enforce_dest	Ensure that destination node can enforce restrictions on the destination socket.
rawip_recv	Receive raw IP packet.
rawip_send	Send raw IP packet.
recvfrom	Network interface and address check permission for use with the ingress permission.
sendto	Network interface and address check permission for use with the egress permission.
tcp_recv	Receive TCP packet.
tcp_send	Send TCP packet.
udp_recv	Receive UDP packet.
udp_send	Send UDP packet.

Class	**netif** – Network Interface (e.g. eth0)
Permissions	**Description** (10 unique permissions)
dccp_recv	Allow Datagram Congestion Control Protocol receive packets.
dccp_send	Allow Datagram Congestion Control Protocol send packets.
egress	Each packet leaving the system must pass an egress access control. Also requires the node sendto permission.
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)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	**tcp_socket** – Protocol: PF_INET, PF_INET6 Family Type: SOCK_STREAM
Permissions	**Description** (Inherit 22 common socket permissions + 5 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown,

	write
acceptfrom	Accept connection from client socket.
connectto	Connect to server socket.
name_connect	Connect to a specific port type.
newconn	Create new connection.
node_bind	Bind to a node.

Class	**udp_socket** – Protocol: PF_INET, PF_INET6 Family Type: SOCK_DGRAM
Permissions	**Description** (Inherit 22 common socket permissions + 1 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
node_bind	Bind to a node.

Class	**rawip_socket** – Protocol: PF_INET, PF_INET6 Family Type: SOCK_RAW
Permissions	**Description** (Inherit 22 common socket permissions + 1 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
node_bind	Bind to a node.

Class	**packet_socket** – Protocol: PF_PACKET Family Type: All.
Permissions	**Description** (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	**unix_stream_socket** – Communicate with processes on same machine. Protocol: PF_STREAM Family Type: SOCK_STREAM
Permissions	**Description** (Inherit 22 common socket permissions + 3 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
acceptfrom	Accept connection from client socket.
connectto	Connect to server socket.
newconn	Create new socket for connection.

Class	**unix_dgram_socket** – Communicate with processes on same

	machine. Protocol: PF_STREAM Family Type: SOCK_DGRAM
Permissions	**Description** (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	**tun_socket** – TUN is Virtual Point-to-Point network device driver to support IP tunneling.
Permissions	**Description** (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

8.5.1 IPSec Network Object Classes

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)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	**netlink_xfrm_socket** - Netlink socket to maintain IPSec parameters.
Permissions	**Description** (Inherit 22 common socket permissions + 2 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
nlmsg_read	Get IPSec configuration information.
nlmsg_write	Set IPSec configuration information.

8.5.2 Netlink Object Classes

Netlink sockets communicate between userspace and the kernel.

Class	**netlink_socket** - Netlink socket that is not part of any specific SELinux Netlink socket class. Protocol: PF_NETLINK Family Type: All other types that are not part of any other specific netlink object class.
Permissions	**Description** (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	**netlink_route_socket** – Netlink socket to manage and control network resources.
Permissions	**Description** (Inherit 22 common socket permissions + 2 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
nlmsg_read	Read kernel routing table.
nlmsg_write	Write to kernel routing table.

Class	**netlink_firewall_socket** – Netlink socket for firewall filters.
Permissions	**Description** (Inherit 22 common socket permissions + 2 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
nlmsg_read	Read netlink message.
nlmsg_write	Write netlink message.

Class	**netlink_tcpdiag_socket** - Netlink socket to monitor TCP connections.
Permissions	**Description** (Inherit 22 common socket permissions + 2 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
nlmsg_read	Request information about a protocol.
nlmsg_write	Write netlink message.

Class	**netlink_nflog_socket** - Netlink socket for Netfilter logging
Permissions	**Description** (Inherit 22 common socket permissions)

Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	**netlink_selinux_socket** - Netlink socket to receive SELinux events such as a policy or boolean change.
Permissions	**Description** (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	**netlink_audit_socket** - Netlink socket for audit service.
Permissions	**Description** (Inherit 22 common socket permissions + 5 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
nlmsg_read	Query status of audit service.
nlmsg_readpriv	List auditing configuration rules.
nlmsg_relay	Send userspace audit messages to theaudit service.
nlmsg_tty_audit	Control TTY auditing.
nlmsg_write	Update audit service configuration.

Class	**netlink_ip6fw_socket** - Netlink socket for IPv6 firewall filters.
Permissions	**Description** (Inherit 22 common socket permissions + 2 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
nlmsg_read	Read netlink message.
nlmsg_write	Write netlink message.

Class	**netlink_dnrt_socket** - Netlink socket for DECnet routing
Permissions	**Description** (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	**netlink_kobject_uevent_socket** - Netlink socket to

	send kernel events to userspace.
Permissions	**Description** (Inherit 22 common socket permissions)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

8.5.3 Miscellaneous Network Object Classes

Class	**peer** - NetLabel and Labeled IPsec have separate access controls, the network peer label consolidates these two access controls into a single one (see http://paulmoore.livejournal.com/1863.html for details).
Permissions	**Description** (1 unique permission)
recv	Receive packets from a labeled networking peer.

Class	**packet** – Supports 'secmark' services where packets are labeled using iptables to select and label packets, SELinux 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)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write

Class	**dccp_socket** - Datagram Congestion Control Protocol (DCCP)
Permissions	**Description** (Inherit 22 common socket permissions + 2 unique)
Inherit Common Socket Permissions	accept, append, bind, connect, create, getattr, getopt, ioctl, listen, lock, name_bind, read, recv_msg, recvfrom, relabelfrom, relabelto, send_msg, sendto, setattr, setopt, shutdown, write
name_connect	Allow DCCP name connect().
node_bind	Allow DCCP bind().

8.6 IPC Object Classes

Class	**ipc** – No longer used
Permissions	**Description** (Inherit 9 common IPC permissions)
Inherit Common IPC Permissions	associate, create, destroy, getattr, read, setattr, unix_read, unix_write, write

Class	**sem** - Semaphores
Permissions	**Description** (Inherit 9 common IPC permissions)
Inherit Common IPC Permissions	associate, create, destroy, getattr, read, setattr, unix_read, unix_write, write

Class	**msgq** – IPC Message queues
Permissions	**Description** (Inherit 9 common IPC permissions + 1 unique)
Inherit Common IPC Permissions	associate, create, destroy, getattr, read, setattr, unix_read, unix_write, write
enqueue	Send message to message queue.

Class	**msg** – Message in a queue
Permissions	**Description** (2 unique permissions)
receive	Read (and remove) message from queue.
send	Add message to queue.

Class	**shm** – Shared memory segment
Permissions	**Description** (Inherit 9 common IPC permissions + 1 unique)
Inherit Common IPC Permissions	associate, create, destroy, getattr, read, setattr, unix_read, unix_write, write
lock	Lock or unlock shared memory.

8.7 Process Object Class

Class	**process** – An object is instantiated for each process created by the system.
Permissions	**Description** (30 unique permissions)
dyntransition	Dynamically transition to a new context using **setcon**(3).
execheap	Make the heap executable.
execmem	Make executable an anonymous mapping or private file mapping that is writable.
execstack	Make the main process stack executable.
fork	Create new process using fork(2).
getattr	Get process security information.
getcap	Get Linux capabilities of process.
getpgid	Get group Process ID of another process.
getsched	Get scheduling information of another process.
getsession	Get session ID of another process.

noatsecure	Disable secure mode environment cleansing.
ptrace	Trace program execution of parent 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**(3).
setfscreate	Set security context by **setfscreatecon**(3).
setkeycreate	Set security context by **setkeycreatecon**(3).
setpgid	Set group Process ID of another process.
setrlimit	Change process rlimit information.
setsched	Modify scheduling information of another process.
setsockcreate	Set security context by **setsockcreatecon**(3).
share	Allow state sharing with cloned or forked process.
sigchld	Send SIGCHLD signal.
siginh	Inherit signal state from parent process.
sigkill	Send SIGKILL signal.
signal	Send a signal other than SIGKILL, SIGSTOP, or SIGCHLD.
signull	Test for exisitence of another process without sending a signal
sigstop	Send SIGSTOP signal
transition	Transition to a new context on exec().

8.8 Security Object Class

Class	**security** - This is the security server object and there is only one instance of this object (for the SELinux security server).
Permissions	**Description** (11 unique permissions)
check_context	Determine whether the context is valid by querying the security server.
compute_av	Compute an access vector given a source/target/class.
compute_create	Determine context to use when querying the security server about a transition rule (type_transition).
compute_member	Determine context to use when querying the security server about a membership decision (type_member for a polyinstantiated object).
compute_relabel	Determines the context to use when querying the security server about a relabeling decision (type_change).
compute_user	Determines the context to use when querying the security server about a user decision (user).
load_policy	Load the security policy into the kernel (the security server).
setbool	Change a boolean value within the active policy.
setcheckreqprot	Set if SELinux will check original protection mode or modified protection mode (read-implies-exec) for mmap / mprotect.
setenforce	Change the enforcement state of SELinux (permissive or enforcing).
setsecparam	Set kernel access vector cache tuning parameters.

8.9 System Operation Object Class

Class	**system** - This is the overall system object and there is only one instance of this object.
Permissions	**Description** (4 unique permissions)
ipc_info	Get info about an IPC object.
syslog_console	Control output of kernel messages to the console with syslog(2).
syslog_mod	Clear kernel message buffer with syslog(2).
syslog_read	Read kernel message with syslog(2).

8.10 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.

8.11 Capability Object Classes

Class	**capability** – Used to manage the Linux capabilities granted to root processes. Taken from the header file: /usr/include/linux/capability.h
Permissions	**Description** (32 unique permissions)
audit_control	Change auditing rules. Set login UID.
audit_write	Send audit messsages from user space.
chown	Allow changing file and group ownership.
dac_override	Overrides all DAC including ACL execute access.
dac_read_search	Overrides DAC for read and directory search.
fowner	Grant all file operations otherwise restricted due to different ownership except where FSETID capability is applicable. DAC and MAC accesses are not overridden.
fsetid	Overrides the restriction that the real or effective user ID of a process sending a signal must match the real or effective user ID of the process receiving the signal.
ipc_lock	Grants the capability to lock non-shared and shared memory segments.
ipc_owner	Grant the ability to ignore IPC ownership checks.
kill	Allow signal raising for any process.
lease	Grants ability to take leases on a file.
linux_immutable	Grant privilege to modify S_IMMUTABLE and S_APPEND file attributes on supporting filesystems.
mknod	Grants permission to creation of character and block device nodes.

net_admin	Allow the following: interface configuration; administration of IP firewall; masquerading and accounting; setting debug option on sockets; modification of routing tables; setting arbitrary process / group ownership on sockets; binding to any address for transparent proxying; setting TOS (type of service); setting promiscuous mode; clearing driver statistics; multicasting; read/write of device-specific registers; activation of ATM control sockets.
net_bind_service	Allow low port binding. Port < 1024 for TCP/UDP. VCI < 32 for ATM.
net_raw	Allows opening of raw sockets and packet sockets.
netbroadcast	Grant network broadcasting and listening to incoming multicasts.
setfcap	Allow the assignment of file capabilities.
setgid	Allow setgid(2) allow setgroups(2) allow fake gids on credentials passed over a socket.
setpcap	Transfer capability maps from current process to any process.
setuid	Allow all setsuid(2) type calls including fsuid. Allow passing of forged pids on credentials passed over a socket.
sys_admin	Allow the following: configuration of the secure attention key; administration of the random device; examination and configuration of disk quotas; configuring the kernel's syslog; setting the domainname; setting the hostname; calling bdflush(); mount() and umount(), setting up new smb connection; some autofs root ioctls; 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
Permissions	**Description** (2 unique permissions)
mac_admin	Allow contexts not defined in the policy to be assigned. This is called 'deferred mapping of security contexts' and is explained at: http://www.nsa.gov/research/selinux/list-archive/0805/26046.shtml
mac_override	
wake_alarm	
epollwakeup	

8.12 X Windows Object Classes

These are userspace objects managed by XSELinux.

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.
write	Draw within a window. Note that this will also give write permission to all child windows, therefore (for MLS), only SystemHigh processes should have 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)
Inherit Common X_Device Permissions	add, bell, create, destroy, force_cursor, freeze, get_property, getattr, getfocus, grab, list_property, manage, read, remove, set_property, setattr, setfocus, use, write

Class	**x_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)
copy	Copy the data
paste	Paste the data
paste_after_confirm	Need to confirm that the paste is allowed.

Class	x_pointer - The mouse or other pointing device managed by the X-server.
Permissions	**Description** (Inherit 19 common x_device permissions)
Inherit Common X_Device Permissions	add, bell, create, destroy, force_cursor, freeze, get_property, getattr, getfocus, grab, list_property, manage, read, remove, set_property, setattr, setfocus, use, write

Class	x_keyboard - The keyboard managed by the X-server.
Permissions	**Description** (Inherit 19 common x_device permissions)
Inherit Common X_Device Permissions	add, bell, create, destroy, force_cursor, freeze, get_property, getattr, getfocus, grab, list_property, manage, read, remove, set_property, setattr, setfocus, use, write

8.13 Database Object Classes

These are userspace objects – The PostgreSQL database supports these with their SE-PostgreSQL database extension. The "Security-Enhanced PostgreSQL Security Wiki" [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 + 3 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
access	Required to connect to the database – this is the minimum permission required by an SE-PostgreSQL client.

install_module	Required to install a dynmic link library.
load_module	Required to load a dynmic link library.

Class	db_table
Permission	**Description** (Inherit 6 common database permissions + 5 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
delete	Required to delete from a table with a DELETE statement, or when removing the table contents with a TRUNCATE statement.
insert	Required to insert into a table with an INSERT statement, or when restoring it with a COPY FROM statement.
lock	Required to get a table lock with a LOCK statement.
select	Required to refer to a table with a SELECT statement or to dump the table contents with a COPY TO statement.
update	Required to update a table with an UPDATE statement.

Class	db_schema
Permission	**Description** (Inherit 6 common database permissions + 3 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
search	Search for an object in the schema.
add_name	Add an object to the schema.
remove_name	Remove an object from the schema.

Class	db_procedure
Permission	**Description** (Inherit 6 common database permissions + 3 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
entrypoint	Required for any functions defined as Trusted Procedures (see [Ref. 3]).
execute	Required for functions executed with SQL queries.
install	

Class	db_column
Permission	**Description** (Inherit 6 common database permissions + 3 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
insert	Required to insert a new entry using the INSERT statement.
select	Required to reference columns.
update	Required to update a table with an UPDATE statement.

Class	db_tuple

Permission	**Description** (7 unique)
`delete`	Required to delete entries with a `DELETE` or `TRUNCATE` statement.
`insert`	Required when inserting a entry with an `INSERT` statement, or restoring tables with a `COPY FROM` statement.
`relabelfrom`	The security context of an entry can be changed with an `UPDATE` to the `security_context` column at which time `relabelfrom` and `relabelto` permission is evaluated. The client must have `relabelfrom` permission to the security context before the entry is changed, and `relabelto` permission to the security context after the entry is changed.
`relabelto`	
`select`	Required when: reading entries with a `SELECT` statement, returning entries that are subjects for updating queries with a `RETURNING` clause, or dumping tables with a `COPY TO` statement. Entries that the client does not have `select` permission on will be filtered from the result set.
`update`	Required when updating an entry with an `UPDATE` statement. Entries that the client does not have update permission on will not be updated.
`use`	Controls usage of system objects that require permission to "use" objects such as data types, tablespaces and operators.

Class	**`db_blob`**
Permission	**Description** (Inherit 6 common database permissions + 4 unique)
`Inherit Common Database Permissions`	`create, drop, getattr, relabelfrom, relabelto, setattr`
`export`	Export a binary large object by calling the `lo_export()` function.
`import`	Import a file as a binary large object by calling the `lo_import()` function.
`read`	Read a binary large object the `loread()` function.
`write`	Write a binary large objecty with the `lowrite()` function.

Class	**`db_view`**
Permission	**Description** (Inherit 6 common database permissions + 1 unique)
`Inherit Common Database Permissions`	`create, drop, getattr, relabelfrom, relabelto, setattr`
`expand`	Allows the expansion of a 'view'.

Class	**`db_sequence` -** A sequential number generator
Permission	**Description** (Inherit 6 common database permissions + 3 unique)
`Inherit Common Database Permissions`	`create, drop, getattr, relabelfrom, relabelto, setattr`
`get_value`	Get a value from the sequence generator object.
`next_value`	Get and increment value.
`set_value`	Set an arbitrary value.

Class	**db_language** - Support for script languages such as Perl and Tcl for SQL Procedures
Permission	**Description** (Inherit 6 common database permissions + 2 unique)
Inherit Common Database Permissions	create, drop, getattr, relabelfrom, relabelto, setattr
implement	Whether the language can be implemented or not for the SQL procedure.
execute	Allow the execution of a code block using a 'DO' statement.

8.14 Miscellaneous Object Classes

Class	**passwd** - This is a userspace object for controlling changes to passwd information.
Permissions	**Description** (5 unique permissions)
chfn	Change another users finger info.
chsh	Change another users shell.
crontab	crontab another user.
passwd	Change another users passwd.
rootok	pam_rootok check – skip authentication.

Class	**nscd** - This is a userspace object for the Name Service Cache Daemon.
Permission	**Description** (10 unique permissions)
admin	Allow the nscd daemon to be shut down.
getgrp	Get group information.
gethost	Get host information.
getpwd	Get password information.
getserv	Get ?? information.
getstat	Get the AVC stats from the nscd daemon.
shmemgrp	Get shmem group file descriptor.
shmemhost	Get shmem host descriptor. ??
shmempwd	
shmemserv	

Class	**dbus** - This is a userspace object for the D-BUS Messaging service that is required to run various services.
Permission	**Description** (2 unique permissions)
acquire_svc	Open a virtual circuit (communications channel).
send_msg	Send a message.

Class	**context** - This is a userspace object for the translation daemon mcstransd. These permissions are required to allow translation and querying of level and ranges for MCS and MLS systems.

Permission	Description (2 unique permissions)
contains	Calculate a MLS/MCS subset - Required to check what the configuration file contains.
translate	Translate a raw MLS/MCS label - Required to allow a domain to translate contexts.

Class	key – This is a kernel object to manage Keyrings.
Permission	**Description** (7 unique permissions)
create	Create a keyring.
link	Link a key into the keyring.
read	Read a keyring.
search	Search a keyring.
setattr	Change permissions on a keyring.
view	View a keyring.
write	Add a key to the keyring.

Class	memprotect – This is a kernel object to protect lower memory blocks.
Permission	**Description** (1 unique permission)
mmap_zero	Security check on mmap operations to see if the user is attempting to mmap to low area of the address space. The amount of space protected is indicated by a proc tunable (/proc/sys/vm/mmap_min_addr). Setting this value to 0 will disable the checks. The "SELinux hardening for mmap_min_addr protections" [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 / mmap_zero permission instead of only one or the other).

Class	service – This is a userspace object to manage systemd services.
Permission	**Description** (5 unique permissions)
start	Start systemd services.
stop	Stop systemd services.
status	Read service status.
reload	Restart systemd services.
kill	Kill services.

9. Appendix B - libselinux Library Functions

This appendix contains:

1. Information regarding the example source that use the libselinux functions - these are available in the Notebook source tarball (these are for libselinux version 2.1.6).
2. A summary of all libselinux functions available in version 2.1.11.

9.1 Source Code Examples

The Notebook source tarball contains a number of simple examples that show all the F-17 (libselinux 2.1.6) functions. These are located in the notebook-source/libselinux/examples directory along with a README file and simple Makefile. These examples make use of a simple shared library that must be installed first (see notebook-source/notebook-tools/library/README) , this allows users to select contexts strings and other variables from a configuration file.

Some of the examples require simple policy modules to show different results, these are located in the notebook-source/libselinux/policy-modules directory.

Table 33 shows the example name and the functions they use, the policy module name is shown where applicable.

Notes:

1. The 'raw' version of a function is always used if available as this will avoid any confusion if the mcstrans daemon is running, the exception is for the translation example.
2. Examples that use the binary policy file also make use of libsepol functions:

   sepol_set_policydb_from_file(3)

   sepol_check_context(3)
3. There are other examples (basic_...) that use the selinuxfs interface to obtain information. These are just to show how this interface is used and should not be implemented in production software.

.

	General Context Functions
1.	**context_get_components_example.c** context_new, context_str, context_user_get, context_role_get, context_type_get, context_range_get, security_check_context, context_free
2.	**context_set_components_example.c** context_new, context_str, context_user_set, context_role_set, context_type_set, context_range_set, security_check_context, context_free, is_selinux_mls_enabled
3.	**security_canonicalize_context_example.c** (security_canonicalize_context_example.conf) security_canonicalize_context, freecon
4.	**security_check_context_example.c** security_check_context

5.	**security_get_initial_context_example.c** security_get_initial_context_example, freecon
6.	**selinux_file_context_cmp_example.c** selinux_file_context_cmp
7.	**selinux_file_context_verify_example.c** selinux_file_context_verify
	User Session Functions
8.	**get_default_type_example.c** get_default_type
9.	**get_default_context_example.c** get_default_context, freecon
10.	**get_default_context_with_level_example.c** get_default_context_with_level, freecon
11.	**get_default_context_with_role_example.c** get_default_context_with_role, freecon
12.	**get_default_context_with_rolelevel_example.c** get_default_context_with_rolelevel, freecon
13.	**get_ordered_context_list_example.c** get_ordered_context_list, freecon, freeconary
14.	**get_ordered_context_list_with_level_example.c** get_ordered_context_list, freecon, freeconary
15.	**manual_user_enter_context_example.c** manual_user_enter_context, freecon
16.	**query_user_context_example.c** query_user_context, freecon
17.	**getseuser_example.c** getseuser
18.	**getseuserbyname_example.c** getseuserbyname
	Default File Labeling Functions (get/set default file contexts)
19.	**matchpathcon_example.c** matchpathcon, matchpathcon_fini, freecon
20.	**matchpathcon_init_prefix_example.c** matchpathcon, matchpathcon_fini, freecon, matchpathcon_init_prefix
21.	**matchpathcon_file_example.c** matchpathcon_init_prefix, matchpathcon, matchpathcon_checkmatches, matchpathcon_fini, freecon
22.	**matchpathcon_filespec_example1.c** set_matchpathcon_flags, matchpathcon_init_prefix, matchpathcon_index, matchpathcon_filespec_add, matchpathcon_filespec_eval, matchpathcon_checkmatches, matchpathcon_filespec_destroy, matchpathcon_fini, freecon
23.	**matchpathcon_filespec_example2.c** set_matchpathcon_printf, set_matchpathcon_invalidcon, set_matchpathcon_flags, matchpathcon_init_prefix, matchpathcon_index, matchpathcon_filespec_add, matchpathcon_filespec_eval, matchpathcon_checkmatches, matchpathcon_filespec_destroy, matchpathcon_fini, freecon
24.	**matchpathcon_policy_file_example.c** sepol_set_policydb_from_file, sepol_check_context, set_matchpathcon_invalidcon, set_matchpathcon_flags, matchpathcon_init, matchpathcon, matchpathcon_checkmatches, matchpathcon_fini, freecon
25.	**selabel_file_example.c** selabel_open, selabel_lookup, selabel_stats, selabel_close, freecon
26.	**selabel_policy_file_example.c** sepol_set_policydb_from_file, sepol_check_context, selinux_set_callback,

	selabel_open, selabel_lookup, selabel_stats, selabel_close
27.	**selabel_policy_file_log_example.c** sepol_set_policydb_from_file, sepol_check_context, selinux_set_callback, selabel_open, selabel_lookup, selabel_stats, selabel_close
28.	**selinux_lsetfilecon_default_example.c** lgetfilecon, selinux_lsetfilecon_default, freecon
	File Labeling Functions (get/set file contexts in extended attributes - `xattr`)
29.	**getfilecon_example.c** getfilecon, freecon
30.	**setfilecon_example.c** (setfilecon_example.conf) getfilecon, setfilecon, freecon
31.	**fgetfilecon_example.c** fgetfilecon, freecon
32.	**fsetfilecon_example.c** fgetfilecon, fsetfilecon, freecon
33.	**lgetfilecon_example.c** lgetfilecon, freecon
34.	**lsetfilecon_example.c** lgetfilecon, lsetfilecon, freecon
	SELinux-aware Application Labeling Functions
35.	**selabel_db_example.c** selabel_open, selabel_lookup, selabel_stats, selabel_close, freecon
36.	**selabel_media_example.c** selabel_open, selabel_lookup, selabel_stats, selabel_close, freecon
37.	**selabel_x_example.c** selabel_open, selabel_lookup, selabel_stats, selabel_close, freecon
38.	**selinux_path_functions_example.c** selinux_path, selinux_policy_root, selinux_binary_policy_path, selinux_booleans_path, selinux_colors_path, selinux_contexts_path, selinux_customizable_types_path, selinux_default_context_path, selinux_default_type_path, selinux_failsafe_context_path, selinux_file_context_homedir_path, selinux_file_context_local_path, selinux_file_context_path, selinux_file_context_subs_path, selinux_homedir_context_path, selinux_media_context_path, selinux_netfilter_context_path, selinux_removable_context_path, selinux_securetty_types_path, selinux_translations_path, selinux_user_contexts_path, selinux_users_path, selinux_usersconf_path, selinux_virtual_domain_context_path, selinux_virtual_image_context_path, selinux_x_context_path, selinux_sepgsql_context_path
39.	**matchmediacon_example.c** matchmediacon, freecon
40.	**is_context_customizable_example.c** is_context_customizable
41.	**selinux_check_securetty_context_example.c** selinux_check_securetty_context
	Label Translation Management Functions
42.	**selinux_raw_to_trans_context_example.c** selinux_raw_to_trans_context, selinux_translation_path, freecon
43.	**selinux_trans_to_raw_context_example.c** selinux_trans_to_raw_context, selinux_translation_path, freecon
44.	**selinux_raw_context_to_color_example1.c** selinux_raw_context_to_color, selinux_colors_path, selinux_translation_path, freecon
45.	**selinux_raw_context_to_color_example2.c** selinux_raw_context_to_color, selinux_colors_path, selinux_translation_path, context_new, context_user_get, context_role_get, context_type_get,

	context_range_get, context_free, freecon
	Process Labeling Functions
46.	**getcon_example.c** getcon, freecon
47.	**setcon_example.c** (setcon_example.conf) setcon, getcon, freecon, *execvp*
48.	**setcon_thread1_example.c** (setcon_example.conf, setcon_thread_example.conf) setcon, getcon, freecon, *execvp*, *pthread_create*
49.	**setcon_thread2_example.c** (setcon_example.conf, setcon_thread_example.conf) setcon, getcon, freecon, *execvp*, *pthread_create*
50.	**getexeccon_example.c** getexeccon, freecon
51.	**setexeccon_example.c** (setexeccon_example.conf) setexeccon, getexeccon, freecon, *execvp*
52.	**getpidcon_example.c** getpidcon, freecon
53.	**getprevcon_example.c** getprevcon, freecon
54.	**rpm_execcon_example.c** rpm_execcon, getcon
	File Creation Labeling Functions
55.	**getfscreatecon_example.c** getfscreatecon, freecon
56.	**setfscreatecon_example.c** setfscreatecon, getfscreatecon, *open (file)*, fgetfilecon, freecon
	Key Creation Labeling Functions
57.	**getkeycreatecon_example.c** getkeycreatecon, freecon
58.	**setkeycreatecon_example.c** (setkeycreatecon_example.conf) setkeycreatecon, getkeycreatecon, freecon, *add_key*, *request_key*, *keyctl*, *keyctl_revoke*
	Socket Creation Labeling Functions
59.	**getsockcreatecon_example.c** getsockcreatecon, freecon
60.	**setsockcreatecon_example.c** setsockcreatecon, getsockcreatecon, freecon, *socket*, fgetfilecon, getpeercon
	Peer Socket Labeling Functions - These work as client/server
61.	**getpeercon_server_example.c** (getpeercon_example.conf) getpeercon, getcon, fgetfilecon, freecon + socket services
62.	**getpeercon_client_example.c** (getpeercon_example.conf) getpeercon, freecon + socket services
	Class and Permission Functions
63.	**print_access_vector_example.c** print_access_vector, string_to_security_class
64.	**security_class_to_string_example.c** security_class_to_string
65.	**security_av_perm_to_string_example.c** security_av_perm_to_string, string_to_security_class

66.	**security_av_string_example.c** security_av_string, string_to_security_class
67.	**selinux_set_mapping_example.c** selinux_set_mapping, string_to_security_class, string_to_av_perm
68.	**selinux_set_mapping_with_errors_example.c** selinux_set_mapping, string_to_security_class, string_to_av_perm, security_class_to_string, security_av_perm_to_string
69.	**security_class_to_string_with_class_mapping_example.c** selinux_set_mapping, security_class_to_string
70.	**string_to_security_class_example.c** string_to_security_class
71.	**string_to_av_perm_example.c** string_to_av_perm, sring_to_security_class, security_class_to_string
72.	**selinux_check_passwd_access_example.c** selinux_check_passwd_access, security_getenforce
73.	**selinux_check_access_example.c** selinux_check_access, security_getenforce
74.	**selinux_check_access_audit_example.c** selinux_check_access, security_getenforce, selinux_set_callback, *audit_open*, *audit_log_user_avc_message*
75.	**security_deny_unknown_example.c** security_deny_unknown
	Compute Access Decision Functions
76.	**security_compute_av_example.c** security_compute_av, string_to_security_class, security_class_to_string, print_access_vector
77.	**security_compute_av_flags_example.c** security_compute_av_flags,string_to_security_class, print_access_vector
78.	**security_compute_av_mapping_example.c** security_compute_av_flags,string_to_security_class, print_access_vector, selinux_set_mapping, string_to_av_perm, security_av_string
	Compute Labeling Functions
79.	**security_compute_create_example.c** (security_compute_create_example.conf) security_compute_create, string_to_security_class
80.	**security_compute_member_example.c** (security_compute_member_example.conf) security_compute_member, string_to_security_class
81.	**security_compute_relabel_example.c** (security_compute_relabel_example.conf) security_compute_relabel, string_to_security_class
82.	**security_compute_user_example.c** security_compute_user, freecon, freeconary
	Access Vector Cache Functions
83.	**avc_has_perm_example.c** avc_has_perm, avc_has_perm_noaudit, security_getenforce, avc_open, avc_entry_ref_init, getcon, avc_context_to_sid, avc_sid_to_context, string_to_security_class, string_to_av_perm, getfilecon, matchpathcon, avc_audit, avc_av_stats, avc_sid_stats, avc_cache_stats, avc_reset, avc_destroy
84.	**avc_compute_create_example.c** avc_compute_create, string_to_security_class, avc_context_to_sid, avc_sid_to_context, avc_destroy
85.	**avc_compute_member_example.c** avc_compute_member, string_to_security_class, avc_context_to_sid, avc_sid_to_context, avc_destroy

86.	**avc_has_perm_callbacks_example.c** avc_has_perm, security_getenforce, avc_open, avc_entry_ref_init, getcon, avc_context_to_sid, avc_sid_to_context, string_to_security_class, string_to_av_perm, getfilecon, matchpathcon, avc_audit, avc_av_stats, avc_sid_stats, avc_cache_stats, avc_reset, avc_destroy, selinux_set_callback, avc_add_callback, *audit_open*, *audit_user_avc_message*
87.	**avc_has_perm_deny_unknown_example.c** avc_has_perm, deny_unknown, avc_open, avc_entry_ref_init, avc_destory, avc_context_to_sid, string_to_security_class
	Netlink Functions
88.	**avc_netlink_check_nb_example.c** avc_netlink_check_nb, avc_acquire_netlink_fd, selinux_set_callback, selinux_get_callback, avc_add_callback, security_getenforce, avc_open, avc_entry_ref_init, getcon, avc_context_to_sid, avc_sid_to_context, string_to_security_class, string_to_av_perm, getfilecon, matchpathcon, avc_has_perm, avc_av_stats, avc_sid_stats, avc_cache_stats, avc_netlink_release_fd, avc_netlink_close, audit_close, avc_destroy, *sigaction*, *audit_open*, *audit_close*, *audit_log_user_avc_message*
89.	**avc_netlink_loop_example.c** avc_netlink_loop, avc_acquire_netlink_fd, selinux_set_callback, selinux_get_callback, avc_add_callback, security_getenforce, avc_open, avc_entry_ref_init, getcon, avc_context_to_sid, avc_sid_to_context, string_to_security_class, string_to_av_perm, getfilecon, matchpathcon, avc_has_perm, avc_av_stats, avc_sid_stats, avc_cache_stats, avc_netlink_release_fd, avc_netlink_close, audit_close, avc_destroy, *pthread_create*, *audit_open*, *audit_close*, *audit_log_user_avc_message*
90.	**avc_netlink_open_example.c** avc_netlink_loop, avc_acquire_netlink_fd, selinux_set_callback, selinux_get_callback, security_getenforce, getcon, string_to_security_class, getfilecon, matchpathcon, security_compute_av, print_access_vector, avc_netlink_release_fd, avc_netlink_close, audit_close, avc_destroy, *pthread_create*, *audit_open*, *audit_close*, *audit_log_user_avc_message*
91.	**netlink_security_compute_av_example.c** security_getenforce, getcon, string_to_security_class, security_class_to_string, string_to_av_perm, security_av_string, getfilecon, matchpathcon, security_compute_av, print_access_vector, *pthread_create*, *socket*, *bind*, *recvfrom*, *audit_open*, *audit_log_user_avc_message*, *audit_close*
	Boolean Functions
92.	**security_get_boolean_active_example.c** security_get_boolean_active
93.	**security_get_boolean_names_example.c** security_get_boolean_names
94.	**security_get_boolean_pending_example.c** security_get_boolean_pending
95.	**security_set_boolean_example.c** security_set_boolean
96.	**security_set_boolean_list_example.c** security_set_boolean_list, selinux_booleans_path
97.	**security_commit_booleans_example.c** security_commit_booleans
98.	**security_load_booleans_example.c** security_load_booleans, selinux_booleans_path
	SELinux Management Functions
99.	**is_selinux_mls_enabled_example.c** is_selinux_mls_enabled
100.	**selinux_getenforcemode_example.c** selinux_getenforcemode, selinux_path
101.	**selinux_getpolicytype_example.c** selinux_getpolicytype

102.	**selinux_reset_config_example.c** selinux_reset_config, selinux_path
103.	**security_load_policy_example.c** security_load_policy
104.	**selinux_init_load_policy_example.c** selinux_init_load_policy
105.	**selinux_mkload_policy_example.c** selinux_mkload_policy, selinux_path, selinux_users_path
106.	**is_selinux_enabled_example.c** is_selinux_enabled
107.	**security_getenforce_example.c** security_getenforce
108.	**security_setenforce_example.c** security_setenforce, security_getenforce
109.	**security_policyvers_example.c** security_policyvers
110.	**set_selinuxmnt_example.c** set_selinuxmnt
111.	**security_disable_example.c** security_disable

Table 33: Grouping of the `libselinux` library functions

9.2 API Summary for `libselinux 2.1.11`

These functions have been taken from the following header files of libselinux version 2.1.11:

```
/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.

Num.	Function Name	Description	Header File
1.	`avc_add_callback`	Register a callback for security events.	`avc.h`
2.	`avc_audit`	Audit the granting or denial of permissions in accordance with the policy. This function is typically called by `avc_has_perm()` 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. Logs a message with information about the size and distribution of the access vector table. The audit callback is used to print the message.	`avc.h`
4.	`avc_cache_stats`	Get cache access statistics. Fill the supplied structure with information about AVC activity since the last call to `avc_init()` or `avc_reset()`.	`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`

Num.	Function Name	Description	Header File
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.	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.	avc.h
8.	avc_context_to_sid avc_context_to_sid_raw	Get SID for context. Look up security context ctx in SID table, making a new entry if ctx is not found. 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
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	avc.h

Num.	Function Name	Description	Header File
		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.	
14.	`avc_init (deprecated)`	Use `avc_open` 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`
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	`avc.h`

Num.	Function Name	Description	Header File
		distribution of the SID table. The audit callback is used to print the message.	
24.	avc_sid_to_context avc_sid_to_context_raw	Get copy of context corresponding to SID. Return a copy of the security context corresponding to the input 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 (deprecated)	Use selinux_check_passwd_access or preferably selinux_check_access 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
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.	fini_selinuxmnt	Clear selinuxmnt variable and free allocated memory.	selinux.h
39.	freecon	Free the memory allocated for a context by any of the get* calls.	selinux.h
40.	freeconary	Free the memory allocated for a context array by security_compute_user.	selinux.h
41.	fsetfilecon	Wrapper for the xattr API - Set file context.	selinux.h

Num.	Function Name	Description	Header File
	fsetfilecon_raw		
42.	get_default_context	Get the default security context for a user session for 'user' spawned by 'fromcon' and set *newcon to refer to it. The context will be one of those authorized by the policy, but the selection of a default is subject to user customizable preferences. If 'fromcon' is NULL, defaults to current context. Returns 0 on success or -1 otherwise. Caller must free via freecon.	get_context_list.h
43.	get_default_context_with_level	Same as get_default_context, but use the provided MLS level rather than the default level for the user.	get_context_list.h
44.	get_default_context_with_role	Same as get_default_context, but only return a context that has the specified role.	get_context_list.h
45.	get_default_context_with_rolelevel	Same as get_default_context, but only return a context that has the specified role and level.	get_context_list.h
46.	get_default_type	Get the default type (domain) for 'role' and set 'type' to refer to it. Caller must free via free(). Return 0 on success or -1 otherwise.	get_default_type.h
47.	get_ordered_context_list	Get an ordered list of authorized security contexts for a user session for 'user' spawned by 'fromcon' and set *conary to refer to the NULL-terminated array of contexts. Every entry in the list will be authorized by the policy, but the ordering is subject to user customizable preferences. Returns number of entries in *conary. If 'fromcon' is NULL, defaults to current context. Caller must free via freeconary.	get_context_list.h
48.	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
49.	getcon getcon_raw	Get current context, and set *con to refer to it. Caller must free via freecon.	selinux.h
50.	getexeccon getexeccon_raw	Get exec context, and set *con to refer to it. Sets *con to NULL if no exec context has been set, i.e. using default. If non-NULL, caller must free via freecon.	selinux.h
51.	getfilecon getfilecon_raw	Wrapper for the xattr API - Get file context, and set *con to refer to it. Caller must free via freecon.	selinux.h
52.	getfscreatecon getfscreatecon_raw	Get fscreate context, and set *con to refer to it. Sets *con to NULL if no fs create context has been set, i.e. using default.If non-NULL, caller must free	selinux.h

Num.	Function Name	Description	Header File
		via `freecon`.	
53.	`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`
54.	`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`
55.	`getpidcon` `getpidcon_raw`	Get context of process identified by `pid`, and set `*con` to refer to it. Caller must free via `freecon`.	`selinux.h`
56.	`getprevcon` `getprevcon_raw`	Get previous context (prior to last `exec`), and set `*con` to refer to it. Caller must free via `freecon`.	`selinux.h`
57.	`getseuser`	Get the SELinux `username` and `level` to use for a given Linux `username` and service. These values may then be passed into the `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`
58.	`getseuserbyname`	Get the SELinux `username` and `level` to use for a given Linux `username`. These values may then be passed into the `get_ordered_context_list*` and `get_default_context*` functions to obtain a context for the user. Returns 0 on success or -1 otherwise. Caller must free the returned strings via `free()`.	`selinux.h`
59.	`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`
60.	`init_selinuxmnt`	There is a man page for this, however it is not a user accessable function (internal use only - although the `fini_selinuxmnt` is reachable).	-
61.	`is_context_customizable`	Returns whether a file context is customizable, and should not be relabeled.	`selinux.h`
62.	`is_selinux_enabled`	Return 1 if running on a SELinux kernel, or 0 if not or -1 for error.	`selinux.h`
63.	`is_selinux_mls_enabled`	Return 1 if we are running on a SELinux MLS kernel, or 0 otherwise.	`selinux.h`
64.	`lgetfilecon` `lgetfilecon_raw`	Wrapper for the `xattr` API - Get file context, and set `*con` to refer to it. Caller must free via `freecon`.	`selinux.h`

Num.	Function Name	Description	Header File
65.	`lsetfilecon` `lsetfilecon_raw`	Wrapper for the `xattr` API- Set file context for symbolic link.	`selinux.h`
66.	`manual_user_enter_context`	Allow the user to manually enter a context as a fallback if a list of authorized contexts could not be obtained. Caller must free via `freecon`. Returns 0 on success or -1 otherwise.	`get_context_list.h`
67.	`matchmediacon`	Match the specified media and against the media contexts configuration and set `*con` to refer to the resulting context. Caller must free con via `freecon`.	`selinux.h`
68.	`matchpathcon`	Match the specified pathname and mode against the file context sconfiguration and set `*con` to refer to the resulting `context.'mode'` can be 0 to disable mode matching. Caller must free via `freecon`. If `matchpathcon_init` has not already been called, then this function will call it upon its first invocation with a NULL path.	`selinux.h`
69.	`matchpathcon_checkmatches`	Check to see whether any specifications had no matches and report them. The '`str`' is used as a prefix for any warning messages.	`selinux.h`
70.	`matchpathcon_filespec_add`	Maintain an association between an inode and a specification index, and check whether a conflicting specification is already associated with the same inode (e.g. due to multiple hard links). If so, then use the latter of the two specifications based on their order in the file contexts configuration. Return the used specification index.	`selinux.h`
71.	`matchpathcon_filespec_destroy`	Destroy any inode associations that have been added, e.g. to restart for a new filesystem.	`selinux.h`
72.	`matchpathcon_filespec_eval`	Display statistics on the hash table usage for the associations.	`selinux.h`
73.	`matchpathcon_fini`	Free the memory allocated by `matchpathcon_init`.	`selinux.h`
74.	`matchpathcon_index`	Same as ‘`matchpathcon`’, but return a specification index for later use in a `matchpathcon_filespec_add()` call.	`selinux.h`
75.	`matchpathcon_init`	Load the file contexts configuration specified by '`path`' into memory for use by subsequent `matchpathcon` calls. If '`path`' is NULL, then load the active file contexts configuration, i.e. the path returned by `selinux_file_context_path()`. 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`

Num.	Function Name	Description	Header File
76.	matchpathcon_init_prefix	Same as matchpathcon_init, but only load entries with regexes that have stems that are prefixes of 'prefix'.	selinux.h
77.	print_access_vector	Display an access vector in a string representation.	selinux.h
78.	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 sucess or -1 otherwise.	get_context_list.h
79.	rpm_execcon	Execute a helper for rpm in an appropriate security context.	selinux.h
80.	security_av_perm_to_string	Convert access vector permissions to string names.	selinux.h
81.	security_av_string	Returns an access vector in a string representation. User must free the returned string via free().	selinux.h
82.	security_canonicalize_context security_canonicalize_context_raw	Canonicalize a security context. Returns a pointer to the canonical (primary) form of a security context in canoncon that the kernel is using rather than what is provided by the userspace application in con.	selinux.h
83.	security_check_context security_check_context_raw	Check the validity of a security context.	selinux.h
84.	security_class_to_string	Convert security class values to string names.	selinux.h
85.	security_commit_booleans	Commit the pending values for the booleans.	selinux.h
86.	security_compute_av security_compute_av_raw	Compute an access decision. Queries whether the policy permits the source context scon to access the target context tcon via class tclass with the requested access vector. The decision is returned in avd.	selinux.h
87.	security_compute_av_flags security_compute_av__flags_raw	Compute an access decision and return the flags. Queries whether the policy permits the source context scon to access the target context tcon via class tclass with the requested access vector. The decision is returned in avd. that has an additional **flags** entry. Currently the only flag defined is SELINUX_AVD_FLAGS_PERMISSIVE that indicates the decision was computed on a permissive domain (i.e. the **permissive** policy language statement has been used in policy or **semanage**(8) has been used to set the domain in permissive mode). Note this does not indicate that SELinux is running in permissive mode, only the scon domain.	selinux.h

Num.	Function Name	Description	Header File
88.	`security_compute_create` `security_compute_create_raw`	Compute a labeling decision and set `*newcon` to refer to it. Caller must free via `freecon`.	`selinux.h`
89.	`security_compute_create_name` `security_compute_create_name_raw`	This is identical to **`security_compute_create`**`(3)` but also takes the name of the new object in creation as an argument. When a `type_transtion` rule (see the `type_transition` Rule section) on the given class and the `scon` / `tcon` pair has an object name extension, `newcon` will be returned according to the policy. Note that this interface is only supported on the kernels 2.6.40 or later. For older kernels the object name is ignored.	`selinux.h`
90.	`security_compute_member` `security_compute_member_raw`	Compute a polyinstantiation member decision and set `*newcon` to refer to it. Caller must free via `freecon`.	`selinux.h`
91.	`security_compute_relabel` `security_compute_relabel_raw`	Compute a relabeling decision and set `*newcon` to refer to it. Caller must free via `freecon`.	`selinux.h`
92.	`security_compute_user` `security_compute_user_raw`	Compute the set of reachable user contexts and set `*con` to refer to the NULL-terminated array of contexts. Caller must free via `freeconary`.	`selinux.h`
93.	`security_deny_unknown`	Get the behavior for undefined classes / permissions.	`selinux.h`
94.	`security_disable`	Disable SELinux at runtime (must be done prior to initial policy load).	`selinux.h`
95.	`security_get_boolean_active`	Get the active value for the boolean.	`selinux.h`
96.	`security_get_boolean_names`	Get the boolean names	`selinux.h`
97.	`security_get_boolean_pending`	Get the pending value for the boolean.	`selinux.h`
98.	`security_get_initial_context` `security_get_initial_context_raw`	Get the context of an initial kernel security identifier by name. Caller must free via `freecon`.	`selinux.h`
99.	`security_getenforce`	Get the enforce flag value.	`selinux.h`
100.	`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`
101.	`security_load_policy`	Load a policy configuration.	`selinux.h`
102.	`security_policyvers`	Get the policy version number.	`selinux.h`
103.	`security_set_boolean`	Set the pending value for the boolean.	`selinux.h`
104.	`security_set_boolean_list`	Save a list of booleans in a single transaction.	`selinux.h`

Num.	Function Name	Description	Header File
105.	security_setenforce	Set the enforce flag value.	selinux.h
106.	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
107.	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 freecon.	label.h
108.	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. SELABEL_CTX_ANDROID_PROP – property_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
109.	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
110.	selinux_binary_policy_path	Return path to the binary policy file under the policy root directory.	selinux.h
111.	selinux_booleans_path	Return path to the booleans file under the policy root directory.	selinux.h

Num.	Function Name	Description	Header File
112.	selinux_check_access	Used to check if the source context has the access permission for the specified class on the target context. Note that the permission and class are reference strings. The aux parameter may reference supplemental auditing information. Auditing is handled as described in **avc_audit**(3). See **security_deny_unknown**(3) for how the deny_unknown flag can influence policy decisions.	selinux.h
113.	selinux_check_passwd_access	Check a permission in the passwd class. Return 0 if granted or -1 otherwise. Replaced by selinux_check_access	selinux.h
114.	selinux_check_securetty_context	Check if the tty_context is defined as a securetty. Return 0 if secure, < 0 otherwise.	selinux.h
115.	selinux_colors_path	Return path to file under the policy root directory.	selinux.h
116.	selinux_contexts_path	Return path to contexts directory under the policy root directory.	selinux.h
117.	selinux_customizable_types_path	Return path to customizable_types file under the policy root directory.	selinux.h
118.	selinux_default_context_path	Return path to default_context file under the policy root directory.	selinux.h
119.	selinux_default_type_path	Return path to default_type file.	get_default_type.h
120.	selinux_failsafe_context_path	Return path to failsafe_context file under the policy root directory.	selinux.h
121.	selinux_file_context_cmp	Compare two file contexts, return 0 if equivalent.	selinux.h
122.	selinux_file_context_homedir_path	Return path to file_context.homedir file under the policy root directory.	selinux.h
123.	selinux_file_context_local_path	Return path to file_context.local file under the policy root directory.	selinux.h
124.	selinux_file_context_path	Return path to file_context file under the policy root directory.	selinux.h
125.	selinux_file_context_subs_path	Return path to file_context.subs file under the policy root directory.	selinux.h
126.	selinux_file_context_subsdist_path	Return path to file_context.subs_dist file under the policy root directory.	selinux.h
127.	selinux_file_context_verify	Verify the context of the file 'path' against policy. Return 0 if correct.	selinux.h
128.	selinux_get_callback	Used to get a pointer to the callback function of the given type. Callback functions are set using **selinux_set_callback**(3).	selinux.h
129.	selinux_getenforcemode	Reads the /etc/selinux/config file and determines whether the	selinux.h

Num.	Function Name	Description	Header File
		machine should be started in `enforcing (1)`, `permissive (0)` or `disabled (-1)` mode.	
130.	`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`
131.	`selinux_homedir_context_path`	Return path to file under the policy root directory. Note that this file will only appear in older versions of policy at this location. On systems that are managed using **`semanage`**`(8)` this is now in the policy store.	`selinux.h`
132.	`selinux_init_load_policy`	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`
133.	`selinux_lsetfilecon_default`	This function sets the file context on to the system defaults returns 0 on success.	`selinux.h`
134.	`selinux_media_context_path`	Return path to file under the policy root directory.	`selinux.h`

Num.	Function Name	Description	Header File
135.	`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`
136.	`selinux_netfilter_context_path`	Returns path to the `netfilter_context` file under the policy root directory.	`selinux.h`
137.	`selinux_path`	Returns path to the policy root directory.	`selinux.h`
138.	`selinux_policy_root`	Reads the `/etc/selinux/config` file and returns the top level directory.	`selinux.h`
139.	`selinux_raw_context_to_color`	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`
140.	`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`
141.	`selinux_removable_context_path`	Return path to `removable_context` file under the policy root directory.	`selinux.h`
142.	`selinux_securetty_types_path`	Return path to the `securetty_types` file under the policy root directory.	`selinux.h`
143.	`selinux_sepgsql_context_path`	Return path to `sepgsql_context` file under the policy root directory.	
144.	`selinux_set_mapping`	Userspace class mapping support that establishes a mapping from a user-provided ordering of object classes and permissions to the numbers actually used by the loaded system policy.	`selinux.h`
145.	`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	`selinux.h`

Num.	Function Name	Description	Header File
		passed NULL, sets the returned context to NULL and returns 0.	
146.	`selinux_translations_path`	Return path to `setrans.conf` file under the policy root directory.	`selinux.h`
147.	`selinux_user_contexts_path`	Return path to file under the policy root directory.	`selinux.h`
148.	`selinux_users_path`	Return path to file under the policy root directory.	`selinux.h`
149.	`selinux_usersconf_path`	Return path to file under the policy root directory.	`selinux.h`
150.	`selinux_virtual_domain_context_path`	Return path to file under the policy root directory.	`selinux.h`
151.	`selinux_virtual_image_context_path`	Return path to file under the policy root directory.	`selinux.h`
152.	`selinux_x_context_path`	Return path to `x_context` file under the policy root directory.	`selinux.h`
153.	`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`
154.	`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`
155.	`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`
156.	`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`
157.	`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	`selinux.h`

Num.	Function Name	Description	Header File
		selinuxfs.	
158.	setcon setcon_raw	Set the current security context to con. Note that use of this function requires that the entire application be trusted to maintain any desired separation between the old and new security contexts, unlike exec-based transitions performed via setexeccon. 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
159.	setexeccon setexeccon_raw	Set exec security context for the next execve. Call with NULL if you want to reset to the default.	selinux.h
160.	setfilecon setfilecon_raw	Wrapper for the xattr API - Set file context.	selinux.h
161.	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
162.	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
163.	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
164.	sidget (deprecated)	From 2.0.86 this is a no-op.	avc.h
165.	sidput (deprecated)	From 2.0.86 this is a no-op.	avc.h
166.	string_to_av_perm	Convert string names to access vector permissions.	selinux.h
167.	string_to_security_class	Convert string names to security class values.	selinux.h

10. Appendix C – SELinux Commands

This section gives a brief explanation of the SELinux specific commands. Some of these have been used within this Notebook, however the appropriate man pages do give more detail and the SELinux project site has a page that details all the available tools and commands at:

http://userspace.selinuxproject.org/trac/wiki/SelinuxTools

Command	Man Page	Purpose
audit2allow	1	Generates policy allow rules from the audit.log file.
audit2why	8	Describes audit.log messages and why access was denied.
avcstat	8	Displays the AVC statistics.
chcat	8	Change or remove a 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**(8), therefore when using the policy store / loadable modules this does not need to be used.
getenforce	1	Shows the current enforcement state.
getsebool	8	Shows the state of the booleans.
load_policy	8	Loads a new policy into the kernel. Not required when using **semanage**(8) / **semodule**(8) commands.
matchpathcon	8	Show a files path and security context.
newrole	1	Allows users to change roles - runs a new shell with the new security context.
restorecon	8	Sets the security context on one or more files.
run_init	8	Runs an init script under the correct context.
runcon	1	Runs a command with the specified context.
selinuxenabled	1	Shows whether SELinux is enabled or not.
semanage	8	Used to configure various areas of a policy within a policy store.
semodule	8	Used to manage the installation, upgrading etc. of policy modules.
semodule_expand	8	Manually expand a base policy package into a kernel binary policy file.
semodule_link	8	Manually link a set of module packages.
semodule_package	8	Create a module package with various configuration files (file context etc.)
sestatus	8	Show the current status of SELinux and the loaded policy.
setenforce	1	Sets / unsets enforcement mode.
setfiles	8	Initialise the extended attributes of filesystems.
setsebool	8	Sets the state of a boolean to on or off persistently across reboots or for this session only.

11. Appendix D – Document References

Ref.	*Title*	*Author*
1.	Security-Enhanced PostgreSQL Security Wiki	K. Kohei
2.	SELinux Policy Module Primer	J. Brindle
3.	Polyinstantiation of directories in an SELinux system	R. Coker
4.	Implementing SELinux as a Linux Security Module	S. Smalley, C. Vance, W. Salamon
5.	Iptables Tutorial	O. Andreasson
6.	New secmark-based network controls for SELinux	J. Morris
7.	Transitioning to Secmark	Paul Moore
8.	Fallback Label Configuration Example	Paul Moore
9.	Leveraging IPSec for Distributed Authorization	Trent Jaeger
10.	IPSec HOWTO	Ralf Spenneberg
11.	Secure Networking with SELinux	J. Brindle
12.	SELinux by Example	F. Mayer K Macmillan D Caplan
13.	SELinux From Scratch	S. Hallyn
14.	SELinux hardening for mmap_min_addr protections	E. Paris
15.		
16.	Application of the Flask Architecture to the X Window System Server	E. Walsh
17.	X Access Control Extension Specification	E. Walsh
18.	A secure web application platform powered by SELinux	K. Kohei
19.	Kernel-based Virtual Machine	Red Hat
20.	How Does Xen Work	Xen Project
21.	Xen Security Modules	G. Coker
22.	The Case for Security Enhanced (SE)Android	S. Smalley

12. Appendix E - GNU Free Documentation License

Version 1.3, 3 November 2008

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, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. Copying In Quantity

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. Collections Of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. Future Revisions Of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. Relicensing

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.