The SELinux Notebook

Building The Sample Policy

0. Notebook Information

0.1 Copyright Information

0.2 Revision History

Version	Date	Changes
1.0	30th September '14	Added to tarball.

0.3 Acknowledgements

Logo designed by Máirín Duffy

0.4 Index

1. Building a Basic Policy

1.1 Introduction

The objective of this section is to show how policy files are constructed, compiled and loaded using the SELinux command line tools to produce a usable policy for instructional use only.

A monolithic and modular (with loadable modules) policy are built without the use of any support macros or make files from the Reference Policy source.

It is recommended that the `notebook-source-4.0.tar.gz` file is installed in `$HOME` as this contains all the configuration files and source code required to produce the required modules. It also contains README and a simple Makefile for each section.

Note: These examples have been tested on Fedora 20.

1.1.1 Overall Objectives

The main objectives of the sections that follow are to:

1. Show how to construct and build a simple monolithic and base policy.
2. Show how to construct and build a series of loadable modules for use with the base module. This builds into a very simple message filter using a network client / server application and file moving (filter) application.

1.1.2 Build Requirements

To be able to build the policy files only standard SELinux utilities are required. However to build the test 'C' programs development tools will be required:

- gcc tools to compile and link the test applications (`gcc` and `libgcc` packages).
- The `libselinux` library and `libselinux-devel` packages.

If the NetLabel module is being built, the NetLabel tools will need to be installed (`netlabel_tools`).

If the Tresys utilities (`setools`) are used (apol, sechecker etc.), then it is recommended that the policies are built uncompressed by adding the following entry in the `semanage.conf` file:

```
bzip-blocksize=0
```

1.1.3 The Test Policies

Normally SELinux policies are built to deny everything by default, and then enable access as required, however the example policies in this section grant access to everything and then run the test applications in their own domains to isolate them.

The policies built in this section have been tested using the follow sequence:

1. Will the system load, allow users to logon and run applications in permissive mode – If yes then:
2. Set the system to enforcing mode by `setenforce 1`, if still okay then:
3. Log out users and log in again (as now in enforcing mode, the login may fail), if okay then:
4. Edit the `config` file and set `SELINUX=enforcing`, then reboot the system, if okay then:
5. Log in users and run applications, if okay then:
6. Test that the policy meets the security requirements.

If at any stage the load fails, then the repair kernel/CD/DVD may have to be used to investigate the cause. Setting the `config` file `SELINUX` entry to `permissive` and investigating the messages and audit logs can be helpful (but not always).

1.2 Building The Policy Source Files

The policies built in this section make use of a common `policy.conf` source file to demonstrate a monolithic build and a base loadable policy build (traditionally called `base.conf`). Table 1-1 describes the core policy components.

Entry	***Comments***

Security Classes (`class`)	These are from the Reference Policy files:
	`./policy/flask/security_classes`
	`./policy/flask/access_vectors`
	`./policy/flask/initial_sids`

Access Vectors (permissions)	
Initial SIDs	
MLS Sensitivity, category and level Statements	There are no MLS security level information in the sample policy.
MLS Constraints	There are no MLS constraints in the sample policy.
Policy Capability Statements	There are no `policycap` statements in the sample policy, however one is added later for a NetLabel exercise using `network_peer_controls`.
Attributes	There are no `attributes` in the sample policy.
Booleans	There are no `bool` statements in the sample policy.
Type / Type Alias	There is only one `type`: `unconfined_t`. There are no `typealias` statements.
Roles	There is only one `role`: `unconfined_r`.
Policy Rules	There is one `allow` rule for each object class (taken from the `security_classes` file) in the policy that allows unrestricted access to all its permissions as follows: `allow unconfined_t self : class_name *;`
Users	There is one user: `unconfined_u`, for logging on. The `system_u` user is there to allow objects to be labeled `system_u:object_r` as in the standard

Entry	*Comments*
	Reference Policy. The `system_u` user is also required by `semanage(8)` to add network objects.
Constraints	These are no constraints in the sample policy, however one is added later to show role constraints.
Default SID labeling	These have been taken from the standard Reference Policy build with the security contexts updated. Note that the kernel is labeled `unconfined_r` and not `object_r`.
`fs_use_xattr` Statement	Only the `ext3` and `ext4` filesystems have been added. If the system being built supports other filesystems then these will need to be added.
`fs_use_task` and `fs_use_trans` Statements	These have been taken from the standard Reference Policy build.
`genfscon` Statements	Only a selection have been taken from the standard Reference Policy build.
`portcon`, `netifcon` and `nodecon` Statements	There are none of these statements in the policy.

Table 1-1: Policy Components - *for the `policy.conf` and `base.conf` source file.*

1.2.1.1 Problem Resolution

The following may help with resolving issues when building the examples:

1. Once the policies etc. have been built and all goes well, the filesystem will relabeled and the new policy loaded during the reboot process, however any errors encountered will probably result in either:
    a. GNU / Linux hanging, in which case the repair disk will be required. To allow GNU / Linux to load, the `/etc/selinux/config` file should be edited to set either `SELINUX=disabled` or the `SELINUXTYPE=` to a known working policy. The reason for the hang can then be investigated (such as correcting the policy source files and/or re-running the build commands).
    b. The policy will be rejected by the kernel and not loaded, GNU / Linux will then load with no policy enabled, giving another chance at fixing the problem (the screen messages will generally give the reason for the rejection).

1.2.1.2 Monolithic and Base Policy Source File

The policy source file for the monolithic and base loadable module are in the `basic-selinux-policy` directory with simple build scripts.

1.2.1.3 `file_contexts` File

The `file_contexts` file for the build is as follows:

```
/ system_u:object_r:unconfined_t
/.* system_u:object_r:unconfined_t
```

1.2.1.4 `default_contexts` File

The `default_contexts` file is to ensure that the initial logon process uses the `unconfined_r:unconfined_t` role / type pair and is as follows:

```
unconfined_r:unconfined_t unconfined_r:unconfined_t
```

Note that this file will only be required when the additional loadable modules are built as they contain multiple types associated to a single role (therefore the logon process needs to know which of the types to use for the users `user:role:type` security context).

1.2.1.5 `seusers` File

The `seusers` file is not mandatory, however one is added as all policies tend to have one, also when adding additional users via `semanage`, one will be required.

```
system_u:system_u
unconfined_u:unconfined_u
__default__:unconfined_u
```

1.2.1.6 `dbus_contexts` File

The `dbus_contexts` file is required to allow X-Windows to run and is as follows:

```
<!DOCTYPE busconfig PUBLIC "-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN"
 "http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
<busconfig>
  <selinux>
  </selinux>
</busconfig>
```

1.2.1.7 `x_contexts` File

The `x_contexts` file is mandatory to allow X-Windows to run if XSELinux is enabled. If the X-server has the XSELinux object manager enabled in the build (as does Fedora), then it will always be enabled unless there is an `xserver_object_manager` boolean that is set to `false`).

The `x_contexts` file contents is as follows (this is a modified version taken from the Reference Policy):

```
client *                system_u:object_r:unconfined_t
### Rules for X Properties
property _SELINUX_*       system_u:object_r:unconfined_t
property CUT_BUFFER?      system_u:object_r:unconfined_t
property *                system_u:object_r:unconfined_t
```

```
### Rules for X Extensions
extension SELinux         system_u:object_r:unconfined_t
extension *               system_u:object_runconfined_t
### Rules for X Selections
selection PRIMARY         system_u:object_r:unconfined_t
selection CLIPBOARD       system_u:object_r:unconfined_t
selection *               system_u:object_r:unconfined_t
### Rules for X Events
event X11:KeyPress            system_u:object_r:unconfined_t
event X11:KeyRelease          system_u:object_r:unconfined_t
event X11:ButtonPress         system_u:object_r:unconfined_t
event X11:ButtonRelease       system_u:object_r:unconfined_t
event X11:MotionNotify        system_u:object_r:unconfined_t
event XInputExtension:DeviceKeyPress         system_u:object_r:unconfined_t
event XInputExtension:DeviceKeyRelease       system_u:object_r:unconfined_t
event XInputExtension:DeviceButtonPress      system_u:object_r:unconfined_t
event XInputExtension:DeviceButtonRelease    system_u:object_r:unconfined_t
event XInputExtension:DeviceMotionNotify     system_u:object_r:unconfined_t
event XInputExtension:DeviceValuator         system_u:object_r:unconfined_t
event XInputExtension:ProximityIn            system_u:object_r:unconfined_t
event XInputExtension:ProximityOut           system_u:object_r:unconfined_t
event X11:ClientMessage       system_u:object_r:unconfined_t
event X11:SelectionNotify     system_u:object_r:unconfined_t
event X11:UnmapNotify         system_u:object_r:unconfined_t
event X11:ConfigureNotify     system_u:object_r:unconfined_t
event *                       system_u:object_r:unconfined_t
```

1.3 Building the Monolithic Policy

There is a simple Makefile that will build the policy, however the basic steps to build manually are:

1) Ensure you are logged on as 'root' or have access to `su` and that SELinux is running in permissive mode (`setenforce 0`) to perform the build process. It is assumed that the files are built in the `notebook-source/basic-selinux-policy/kernel-language/monolithic-policy` directory.

2) Produce a `policy.conf` file using the build tool:

```
../../../notebook-tools/build-sepolicy -o policy.conf \
-r ../../flask-files
```

3) Produce a `file_contexts` file with the contents shown in the `file_contexts` file section. This will be used to relabel the file system.

4) Produce a `dbus_contexts` file with the contents shown in the `dbus_contexts` file section. This is required for X-Windows to load as it uses the dbus messaging service that has a SELinux user space object manager.

5) Produce a `x_contexts` file with the contents shown in the `x_contexts` File section. This is required for the X-Windows object manager.

6) Find the maximum policy version the SELinux kernel will support by executing the following command:

```
cat /selinux/policyvers
26
```

The output for the F-20 kernel should be '29' depending on any package updates.

7) Compile the policy with `checkpolicy` to produce the binary policy file:

```
checkpolicy -c29 -o policy.29 policy.conf
```

The output from the compilation should be:

```
checkpolicy: loading policy configuration from policy.conf
checkpolicy: policy configuration loaded
checkpolicy: writing binary representation (version 29) to
policy.29
```

8) Make the following directories to store the policy:

```
mkdir /etc/selinux/monolithic-test/policy
mkdir -p /etc/selinux/monolithic-test/contexts/files
```

9) Copy the following files to SELinux policy area:

```
cp policy.29 /etc/selinux/monolithic-test/policy
cp seusers /etc/selinux/monolithic-test/seusers
cp dbus_contexts /etc/selinux/monolithic-test/contexts
cp x_contexts /etc/selinux/monolithic-test/contexts
cp default_contexts /etc/selinux/monolithic-test/contexts
cp file_contexts /etc/selinux/monolithic-test/contexts/files
```

10) The file and directory list in the `/etc/selinux/monolithic-test` directory area should now consist of the following:

```
monolithic-test:
   drwxr-xr-x contexts
   drwxr-xr-x policy
   -rw-r--r-- seusers

monolithic-test/contexts:
   -rw-r--r-- dbus_contexts
   -rw-r--r-- default_contexts
   drwxr-xr-x files
   -rw-r--r-- x_contexts

monolithic-test/contexts/files:
   -rw-r--r-- file_contexts

monolithic-test/policy:
   -rw-r--r-- policy.29
```

11) Edit the `/etc/selinux/config` file and change the entries shown. This will set permissive mode and the location of the policy that will be loaded on the next re-boot. Note - do not put any spaces after these entries.

```
SELINUX=permissive
SELINUXTYPE=monolithic-test
```

12) To allow file system relabeling to be actioned on reboot execute the following command:

```
touch /.autorelabel
```

13) Optionally clear the log files so that they are clear for easier reading after the reboot:

```
> /var/log/messages
> /var/log/audit/audit.log
```

14) Reboot the system. During the boot process, the file system should be re-labeled.

```
reboot
```

1.3.1 Checking the Build

Once the system has reloaded, SELinux will be running in 'permissive' mode. Logon as `root` and use either `seaudit`, `troubleshooter` or simply `tail` in a couple of 'terminal windows' to view the logs:

```
# In one terminal window run:
tail -f /var/log/messages

# In another terminal window run:
tail -f /var/log/audit/audit.log
```

There should be entries for the boot process in the `/var/log/messages` file, however the `/var/log/audit/audit.log` file should only contain entries for the audit daemon, user logon and role change for the logon process.

If the system is 'working' (i.e. it should be stable, load the desktop and allow utilities to be loaded from the menus), then SELinux can be set to enforcing mode by:

```
setenforce 1
```

The new policy will be enforced and the only entries in the logs should be about setting enforcing mode.

If the system is unstable when rebooted, then see the Problem Resolution section for a possible resolution.

1.4 Building the Base Policy Module

This exercise will build the mandatory base policy module that uses the same policy source file as the monolithic policy discussed above.

The basic steps to produce a simple base test policy are:

1. Ensure you are logged on as 'root' and SELinux is running in permissive mode (`setenforce 0`) to perform the build process. It is assumed that the files are built in the `./notebook-source/modular-base-policy` directory.
2. Produce a `base.conf` file using the build tool:

```
../../../notebook-tools/build-sepolicy -o base.conf \
-d ../../flask-files
```

3. Produce a `base.fc` file with the contents shown in the file_contexts file section. This will be used to relabel the file system.
4. Produce a `default_contexts` file with the contents shown in the default_contexts file section. This will be used to ensure that the correct context is used for the logon process (only really needed when the additional loadable modules are built).
5. Produce an `seusers` file with the contents shown in the seusers file section.
6. Produce a `dbus_contexts` file with the contents shown in the dbus_contexts file section. This is required for X-Windows to load as it uses the dbus messaging service that has a SELinux user space object manager.

15) Produce a `x_contexts` file with the contents shown in the x_contexts File section. This is required for the X-Windows object manager.

7. Compile the policy with `checkmodule` to produce an intermediate binary policy file:

```
checkmodule -o base.mod base.conf
```

The output from the compilation should be:

```
checkmodule:  loading policy configuration from base.conf
checkmodule:  policy configuration loaded
checkmodule:  writing binary representation to base.mod
```

8. Package the policy with `semodule_package`, this will produce a base policy module file (note – if successful there are no output messages):

```
semodule_package -o base.pp -m base.mod -f base.fc
```

9. Make the following directories to store the policy:

```
mkdir /etc/selinux/modular-test/policy
mkdir -p /etc/selinux/modular-test/contexts/files
mkdir -p /etc/selinux/modular-test/modules/active/modules
```

10. Copy the following files to SELinux policy area:

```
cp seusers /etc/selinux/modular-test
cp dbus_contexts /etc/selinux/modular-test/contexts
cp default_contexts /etc/selinux/modular-test/contexts
cp x_contexts /etc/selinux/modular-test/contexts
```

11. Install the base policy with `semodule`. This will produce a base policy and a number of files, some of which will be empty (note – if successful there are no output messages):

```
semodule -s modular-test -b base.pp
```

12. The file and directory list in the `/etc/selinux/modular-test` directory area should now consist of the following:

```
/etc/selinux/modular-test:
   drwxr-xr-x. contexts
   drwxr-xr-x. modules
   drwxr-xr-x. policy
-rw-r--r--. seusers

/etc/selinux/modular-test/contexts:
   -rw-r--r--. dbus_contexts
   -rw-r--r--. default_contexts
   drwxr-xr-x. files
   -rw-r--r--. netfilter_contexts
   -rw-r--r--. x_contexts

/etc/selinux/modular-test/contexts/files:
   -rw-r--r--. file_contexts
   -rw-r--r--. file_contexts.homedirs

/etc/selinux/modular-test/modules:
   drwx------. active
   -rw-------. semanage.read.LOCK
   -rw-------. semanage.trans.LOCK

/etc/selinux/modular-test/modules/active:
   -rw-r--r--. base.pp
   -rw-------. commit_num
   -rw-------. file_contexts
   -rw-r--r--. file_contexts.homedirs
   -rw-------. file_contexts.template
   -rw-------. homedir_template
   drwx------. modules
   -rw-------. netfilter_contexts
   -rw-r--r--. policy.kern
   -rw-------. seusers.final
   -rw-------. users_extra

/etc/selinux/modular-test/modules/active/modules:
   empty

/etc/selinux/modular-test/policy:
   -rw-r--r--. policy.29
```

13. Edit the `/etc/selinux/config` file and change the entries shown. This will set permissive mode and the policy location that will be loaded on the next re-boot. Note - do not put any spaces after these entries.

```
SELINUX=permissive
SELINUXTYPE=modular-test
```

This will set permissive mode so if the policy is too restrictive it will still allow a login at least. The SELinux policy name/location is also added (`modular-test`). Note do not put any spaces after the entries.

14. To allow a file system relabeling to be actioned on reboot execute the following command:

```
touch /.autorelabel
```

15. Optionally clear the log files so that they are clear for easier reading:

```
> /var/log/messages
> /var/log/audit/audit.log
```

16. Reboot the system. During the boot process, the file system should be re-labeled.

```
reboot
```

1.4.1 Checking the Base Policy Build

Once the system has reloaded, SELinux will be running in ‘permissive’ mode. Logon as root and follow the same routine as defined in the Checking The Build section.

2. Building the Message Filter Loadable Modules

2.1 Overview of modules

In the sections that follow there are a number of loadable modules built with supporting 'C' programs that form a very simple message filter service as shown in Figure 2.1. The external and internal gateways are client / server applications making use of SEMARK services that are built into `iptables` as discussed in SELinux Networking section of 'The Foundations' volume. The server component can also read and write files to / from a protected directory area (or message queues). The message filter itself is a simple file mover application that moves files from one queue to another.

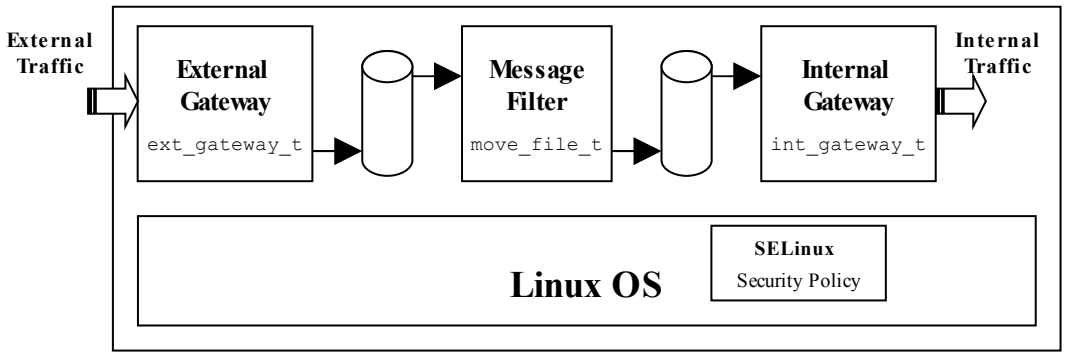

Figure 2.1: Message Filter Components

The components that form the message filter are:

External Gateway – This has a loadable module `ext_gateway.conf` that defines the policy for the external gateway, it also includes an `optional` section that is loaded when other message filter modules are loaded. The gateway requires client / server applications (client.c and server.c) to be compiled for testing and `iptables` (the mangle table) to be loaded for SECMARK testing.

NetLabel Service – This is a simple `netlabel.conf` module that just adds a label at peer level.

Internal Gateway – This is a version of the external gateway module that has been modified to handle internal processing permissions (`int_gateway.conf`). It requires additional entries in the iptables as it uses a different network port. The same client / server applications are used as for the external gateway.

File Mover - This has a loadable module `move_file.conf` that defines the policy for moving files between the external and internal gateways. There is also an application (`move_file.c`) that copies files from one message queue to another (but controlled by the policy).

The security policy for the message filter is simply:

1. No other application must use the secured ports configured in the `iptables` and allocated to the gateways. The secure ports are:

   port `1111` and labeled `int_gateway_packet_t`

port `9999` and labeled `ext_gateway_packet_t`

All other ports are labeled: `default_secmark_packet_t`

2. The message queues and files must be protected from all possible access (read, write, delete etc.) by other domains.

The assumptions are:

1. The SELinux policy will always be in enforcing mode while the message filter is active.
2. The SELinux message filter modules may be in permissive mode for the initial file and directory configuration / initialisation via `restorecon` (this is so that permissions such as `relabelto` / `relabelfrom` are limited to the absolute minimum, in fact only the `iptables` need relabeling permissions as they are loaded under the `unconfined_t` domain).

The modules are built and tested in the following sequence:

1. The external gateway is built along with the client / server applications. This is used to demonstrate the basic secmark functionality using the iptables.
2. The NetLabel module is then built to demonstrate adding a netlabel to the peer network service.
3. The internal gateway and the file mover application and module are finally built to demonstrate the overall message filter as shown in Figure 2.1.

Any comments or views on the modules, applications and their testing are welcome.

2.2 Building the SECMARK Test Loadable Module

The SECMARK tests make use of the external gateway loadable module. The objective of this module is to prove that SECMARK labels can be added to packets, and that depending on the label assigned, those packets can be granted access to the correct domain and denied access other domains using SELinux enforcement.

The tests will use various client / server configurations using the network loop back (`lo`) interface (see Figure 2.2) as follows:

1. Use a 'secure' client / server running in the `ext_gateway_t` domain that will show that packets labeled:

   ```
   system_u:object_r:ext_gateway_packet_t
   ```

   on ports `9999` will get through, while other ports will NOT get through, as they would be labeled:

   ```
   system_u:object_r:default_secmark_packet_t
   ```

2. Use an 'insecure' client / server running in the `unconfined_t` domain that will show that packets labeled:

   ```
   system_u:object_r: ext_gateway_packet_t
   ```

   will NOT get through, while other ports will get through, as they would be labeled:

   ```
   system_u:object_r:default_secmark_packet_t
   ```

3. A mixture of secure and insecure client / server configurations to show access is denied by SELinux unless both services are running in the `ext_gateway_t` domain using port `9999` on the `lo` network.

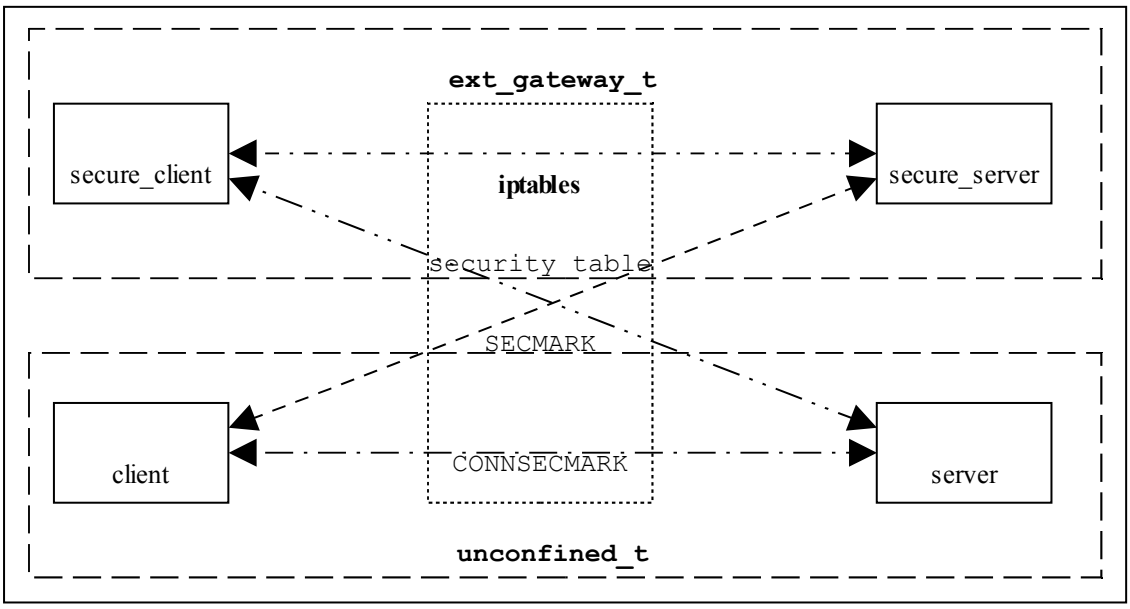

Figure 2.2: SECMARK Testing – *The scenarios for testing the access allowed for SECMARK packets. Note that not all of these tests will be described.*

The SECMARK test loadable module (`ext_gateway.conf`) has a `boolean` called `ext_gateway_audit` that by default enables the transition, send and receive audit events to be logged when successful, these events are shown in the test results below. The `auditallow` statements can be disabled by using the following command:

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

To test SECMARK functionality the following will need to be built and installed:

- The base loadable module built in the Building a Base Loadable Module Policy section is installed and active.
- A loadable policy module (`ext_gateway`) that will enforce the SECMARK policy configured via `iptables`. Note the following points:
    a. The `ext_gateway` module requires a new role of `message_filter_r` to be added to SELinux. This has only been added to demonstrate a `role_transition` rule.
    b. The `ext_gateway` module has an `optional` section that is only enabled when other modules are loaded as a further exercise for the message filter service.
- An `iptables` configuration file that will set-up the mangle table to mark packets with `SECMARK` and `CONNSECMARK` labels.
- Two executable clients (`secure_client` and `client`) and two executable servers (`secure_server` and `server`) that will be used to test the SECMARK functionality. Note that the clients and servers are built using

common source code, and are only labeled differently to allow testing (the secure executables are labeled `secure_services_exec_t` while the others are labeled by default with `unconfined_t`).

The following steps need to be followed to build the test services, although there is a simple Makefile that can be used. It is assumed that the services are installed in `notebook-source/basic-selinux-policy/kernel-language/message-filter/gateways`:

1. Ensure you are logged on as 'root' and SELinux is running in permissive mode (`setenforce 0`) to perform the build process.
2. Use the `ext_gateway.conf` loadable module file supplied in the directory.
3. Produce a `gateway.fc` file (a segment that will be added to the `file_contexts` file during the build) with the contents shown below. This will be used to relabel application files and directories.

```
/usr/local/bin/secure_client system_u:object_r:secure_services_exec_t
/usr/local/bin/secure_server system_u:object_r:secure_services_exec_t
/usr/local/bin/client system_u:object_r:unconfined_t
/usr/local/bin/server system_u:object_r:unconfined_t
```

4. Compile the policy with `checkmodule` to produce an intermediate binary policy file:

```
checkmodule -m ext_gateway.conf -o ext_gateway.mod
```

5. Package the policy with `semodule_package`, this will produce a policy module file:

```
semodule_package -o ext_gateway.pp -m ext_gateway.mod -f gateway.fc
```

6. Install the loadable module with `semodule` (note – if successful there are no output messages):

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

7. If there are no errors reported, then the loadable module has been added to the policy store and loaded as a part of the policy. The policy module can be checked by:

```
semodule -s modular-test -l
```

8. Use the 'C' application called `client.c` and compile two versions of the client by running:

```
gcc -o secure_client client.c -lselinux
gcc -o client client.c -lselinux
```

9. Use the 'C' application called `server.c` and compile two versions of the server by running:

```
gcc -o secure_server server.c -lselinux
```

```
gcc -o server server.c -lselinux
```

10. Move the binaries to `/usr/local/bin`:

```
cp client /usr/local/bin
cp secure_client /usr/local/bin
cp server /usr/local/bin
cp secure_server /usr/local/bin
```

11. Produce a script called `iptables_secmark` with the contents shown below. This will be used to load the `iptables` '`security`' table (Note: The entries for the internal gateway are commented out. This is because the module has not been built yet and leaving these in would produce an error when loading the table with SELinux in enforcing mode).

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

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

# This INPUT rule sets all packets to default_secmark_packet_t: as it is
# executed first:
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

# These rules will replace the above context with the internal or
# external gateway if port 9999 or 1111 is found in either the source or
# destination port of the packet:
iptables -t security -A INPUT -i lo -p tcp --dport 9999 -j SECMARK
--selctx system_u:object_r:ext_gateway_packet_t
iptables -t security -A INPUT -i lo -p tcp --sport 9999 -j SECMARK
--selctx system_u:object_r:ext_gateway_packet_t
#
# These are not required until using the internal gateway:
#iptables -t security -A INPUT -i lo -p tcp --dport 1111 -j SECMARK
--selctx system_u:object_r:int_gateway_packet_t
#iptables -t security -A INPUT -i lo -p tcp --sport 1111 -j SECMARK
--selctx system_u:object_r:int_gateway_packet_t

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

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

# This OUTPUT rule sets all packets to default_secmark_packet_t: as it is
# executed first:
iptables -t security -A OUTPUT -o lo -p tcp -d 127.0.0.0/8 -j SECMARK
--selctx system_u:object_r:default_secmark_packet_t

# These rules will replace the above context with the internal or
# external gateway if port 9999 or 1111 is found in either the source or
# destination port of the packet:

iptables -t security -A OUTPUT -o lo -p tcp --dport 9999 -j SECMARK
--selctx system_u:object_r:ext_gateway_packet_t
iptables -t security -A OUTPUT -o lo -p tcp --sport 9999 -j SECMARK
--selctx system_u:object_r:ext_gateway_packet_t
#
# These are not required until using the internal gateway:
#iptables -t security -A OUTPUT -o lo -p tcp --dport 1111 -j SECMARK
--selctx system_u:object_r:int_gateway_packet_t
#iptables -t security -A OUTPUT -o lo -p tcp --sport 1111 -j SECMARK
--selctx system_u:object_r:int_gateway_packet_t

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

```
iptables -t security -L
```

12. Produce a `restorefiles_gateway` file with the contents shown below. This will be used by the `restorecon` command to relabel the SECMARK test client / server executables after compilation and if any updates are done later.

```
/usr/local/bin/secure_client
/usr/local/bin/secure_server
/usr/local/bin/client
/usr/local/bin/server
```

13. Run the `restorecon(8)` command to relabel the secure versions of the client / server as follows:

```
restorecon -f restorefiles_gateway
```

14. Check that the secure versions of the client / server are labeled correctly using `ls -Z /usr/local/bin`.

```
unconfined_u:object_r:unconfined_t                client
unconfined_u:object_r:secure_services_exec_t      secure_client
unconfined_u:object_r:secure_services_exec_t      secure_server
unconfined_u:object_r:unconfined_t                server
```

15. Add the `message_filter_r` role by running semanage as follows:

```
semanage user -m -R "message_filter_r unconfined_r" user_u
```

    Note: Also need to add the `unconfined_r` role otherwise semanage will remove it from the policy.

The installation process is now complete, the testing is discussed in the next section.

2.2.1 Testing the Module

To test the SECMARK functionality it is recommended that three virtual terminal sessions are opened (as shown in Figure 2.3) for:

1. Running clients as they will display status messages if successful.
2. Running the servers as they display messages when connections are made with the clients.
3. Viewing the audit log file. Note that the module has `auditallow` rules on `packet { send recv }` so that these events can be seen.

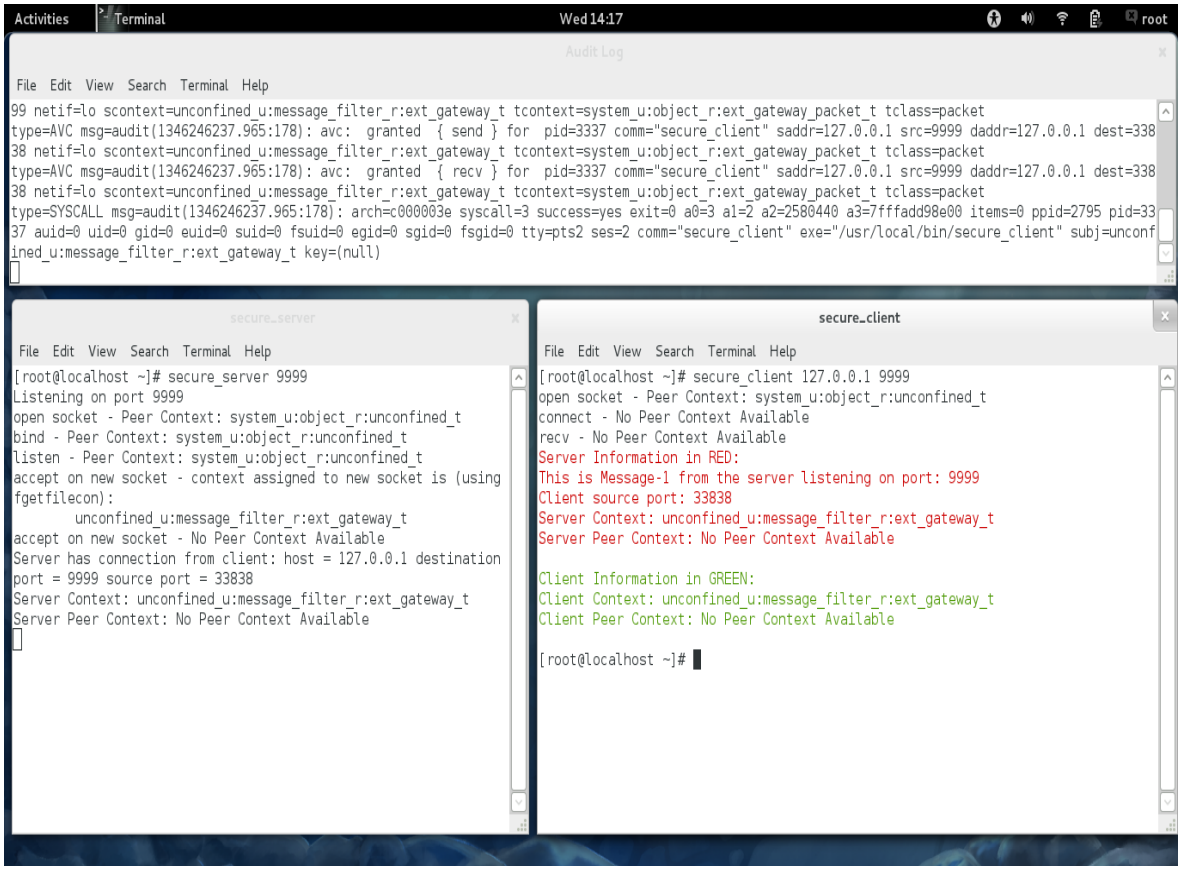

Figure 2.3: Testing using three virtual terminal sessions

2.2.1.1 Running the Tests

It is assumed that there are three terminal sessions logged in as root as shown in Figure 2.3), with the client and server windows both at the directory with the executable secmark code and scripts, and the third window for tailing the `audit.log` file.

Before starting the tests:

1. In the window that will display the audit log, execute the following command:

```
tail -f /var/log/audit/audit.log.
```

2. In a window run the following command to load the `iptables`:

```
./iptables_secmark
```

   Note that it is important to load the iptables as explained in the Importance of Loading the `iptables` section below.

3. In a window run the following command to start enforcing policy:

```
setenforce 1
```

Note that the server must be started before the client. To exit any of the server sessions press `ctrl/c`.

Test 1 – Running secure server and secure client sessions on port `9999` using the loopback interface (`127.0.0.1`):

1. In a window run the following command to start the secure server:

```
secure_server 9999
```

2. In a window run the following command to start the secure client:

```
secure_client 127.0.0.1 9999
```

The `audit.log` should contain only granted events on `transition`, `send` and `recv` (note that the transition also transitioned the role to `message_filter_r`):

```
type=AVC msg=audit(1346246513.236:179): avc:  granted  { transition } for
pid=3750 comm="bash" path="/usr/local/bin/secure_server" dev="dm-1"
ino=149395 scontext=unconfined_u:unconfined_r:unconfined_t
tcontext=unconfined_u:message_filter_r:ext_gateway_t tclass=process
type=SYSCALL msg=audit(1346246513.236:179): arch=c000003e syscall=59
success=yes exit=0 a0=1afa250 a1=1b176c0 a2=1b17ac0 a3=18 items=0 ppid=1834
pid=3750 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0
tty=pts1 ses=2 comm="secure_server" exe="/usr/local/bin/secure_server"
subj=unconfined_u:message_filter_r:ext_gateway_t key=(null)

type=AVC msg=audit(1346246523.519:180): avc:  granted  { transition } for
pid=3768 comm="bash" path="/usr/local/bin/secure_client" dev="dm-1"
ino=149393 scontext=unconfined_u:unconfined_r:unconfined_t
tcontext=unconfined_u:message_filter_r:ext_gateway_t tclass=process
type=SYSCALL msg=audit(1346246523.519:180): arch=c000003e syscall=59
success=yes exit=0 a0=1b2b350 a1=1b36c60 a2=1b13ac0 a3=20 items=0 ppid=2795
pid=3768 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0
tty=pts2 ses=2 comm="secure_client" exe="/usr/local/bin/secure_client"
subj=unconfined_u:message_filter_r:ext_gateway_t key=(null)

type=AVC msg=audit(1346246523.520:181): avc:  granted  { send } for
pid=3768 comm="secure_client" saddr=127.0.0.1 src=33839 daddr=127.0.0.1
dest=9999 netif=lo scontext=unconfined_u:message_filter_r:ext_gateway_t
tcontext=system_u:object_r:ext_gateway_packet_t tclass=packet

type=AVC msg=audit(1346246523.520:181): avc:  granted  { recv } for
pid=3768 comm="secure_client" saddr=127.0.0.1 src=33839 daddr=127.0.0.1
dest=9999 netif=lo scontext=unconfined_u:message_filter_r:ext_gateway_t
tcontext=system_u:object_r:ext_gateway_packet_t tclass=packet
.....
type=AVC msg=audit(1346246523.520:182): avc:  granted  { send } for
pid=3750 comm="secure_server" saddr=127.0.0.1 src=9999 daddr=127.0.0.1
dest=33839 netif=lo scontext=unconfined_u:message_filter_r:ext_gateway_t
tcontext=system_u:object_r:ext_gateway_packet_t tclass=packet

type=AVC msg=audit(1346246523.520:182): avc:  granted  { recv } for
pid=3750 comm="secure_server" saddr=127.0.0.1 src=9999 daddr=127.0.0.1
dest=33839 netif=lo scontext=unconfined_u:message_filter_r:ext_gateway_t
tcontext=system_u:object_r:ext_gateway_packet_t tclass=packet
.....
```

Test 2 – Running the server on port 9999 and the secure client on port 1234 using the loopback interface:

1. In a window run the following command to start the server:

```
server 9999
```

2. In a window run the following command to start the secure client:

```
secure_client 127.0.0.1 1234

Note: ctrl/c will exit the session
```

There should be a 'Connection refused' message and AVC audit messages where the secure client is granted the `transition` but denied the `send`:

```
# Note that the client is allowed to transition:
type=AVC msg=audit(1346246671.409:185): avc:  granted  { transition } for
pid=4093 comm="bash" path="/usr/local/bin/secure_client" dev="dm-1"
ino=149393 scontext=unconfined_u:unconfined_r:unconfined_t
tcontext=unconfined_u:message_filter_r:ext_gateway_t tclass=process
type=SYSCALL msg=audit(1346246671.409:185): arch=c000003e syscall=59
success=yes exit=0 a0=1b36c90 a1=1b15000 a2=1b13ac0 a3=20 items=0 ppid=2795
pid=4093 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0
tty=pts2 ses=2 comm="secure_client" exe="/usr/local/bin/secure_client"
subj=unconfined_u:message_filter_r:ext_gateway_t key=(null)


# But is not allowed to send message to the server as the
# packet is marked default_secmark_packet_t:
type=AVC msg=audit(1346246671.410:186): avc:  denied  { send } for
pid=4093 comm="secure_client" saddr=127.0.0.1 src=56783 daddr=127.0.0.1
dest=1234 netif=lo scontext=unconfined_u:message_filter_r:ext_gateway_t
tcontext=system_u:object_r:default_secmark_packet_t tclass=packet
type=SYSCALL msg=audit(1346246671.410:186): arch=c000003e syscall=42
success=no exit=-111 a0=3 a1=7fffa5e06ad0 a2=10 a3=7fffa5e06840 items=0
ppid=2795 pid=4093 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0
fsgid=0 tty=pts2 ses=2 comm="secure_client"
exe="/usr/local/bin/secure_client"
subj=unconfined_u:message_filter_r:ext_gateway_t key=(null)
```

Test 3 – Running both client and server sessions using port 1234 on the loopback interface (`127.0.0.1`):

1. In a window run the following command to start the server:

```
server 1234
```

2. In a window run the following command to start the client:

```
client 127.0.0.1 1234
```

The `audit.log` should contain only granted events on `send` and `recv` (note that there is NO transition and the role remains as `unconfined_r`):

```
type=AVC msg=audit(1249742778.361:34): avc:  granted  { send } for  pid=2964
comm="client" saddr=127.0.0.1 src=42943 daddr=127.0.0.1 dest=1234 netif=lo
scontext=unconfined_u:unconfined_r:unconfined_t
tcontext=system_u:object_r:default_secmark_packet_t tclass=packet

type=AVC msg=audit(1249742778.361:34): avc:  granted  { recv } for  pid=2964
comm="client" saddr=127.0.0.1 src=42943 daddr=127.0.0.1 dest=1234 netif=lo
scontext=unconfined_u:unconfined_r:unconfined_t
tcontext=system_u:object_r:default_secmark_packet_t tclass=packet
....
....
type=AVC msg=audit(1249742778.362:35): avc:  granted  { send } for  pid=2961
comm="server" saddr=127.0.0.1 src=1234 daddr=127.0.0.1 dest=42943 netif=lo
scontext=unconfined_u:unconfined_r:unconfined_t
tcontext=system_u:object_r:default_secmark_packet_t tclass=packet
```

```
type=AVC msg=audit(1249742778.362:35): avc:  granted  { recv } for  pid=2961
comm="server" saddr=127.0.0.1 src=1234 daddr=127.0.0.1 dest=42943 netif=lo
scontext=unconfined_u:unconfined_r:unconfined_t
tcontext=system_u:object_r:default_secmark_packet_t tclass=packet
```

Test 4 – Running the server on port 9999 and the secure client on port 9999 using the loopback interface:

3. In a window run the following command to start the server:

```
server 9999
```

4. In a window run the following command to start the secure client:

```
secure_client 127.0.0.1 9999

Note: ctrl/c will exit the session
```

The client will hang, waiting for a message that will not succeed. The AVC audit messages show that the secure client has been granted the `transition` and `send` but denied the `recv` from the standard server (but note that the server was allowed to accept the connection):

```
type=AVC msg=audit(1249742873.035:38): avc:  granted  { transition } for
pid=2987 comm="bash" path="/usr/local/bin/secure_client" dev=dm-0 ino=354307
scontext=unconfined_u:unconfined_r:unconfined_t
tcontext=unconfined_u:message_filter_r:ext_gateway_t tclass=process
type=SYSCALL msg=audit(1249742873.035:38): arch=40000003 syscall=11 success=yes
exit=0 a0=8801cf0 a1=87e9ca8 a2=87ee8e8 a3=0 items=0 ppid=2496 pid=2987 auid=0
uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=1
comm="secure_client" exe="/usr/local/bin/secure_client"
subj=unconfined_u:message_filter_r:ext_gateway_t key=(null)

type=AVC msg=audit(1249742873.041:39): avc:  granted  { send } for  pid=2987
comm="secure_client" saddr=127.0.0.1 src=35900 daddr=127.0.0.1 dest=9999
netif=lo scontext=unconfined_u:message_filter_r:ext_gateway_t
tcontext=system_u:object_r:ext_gateway_packet_t tclass=packet

type=AVC msg=audit(1249742873.041:39): avc:  denied  { recv } for  pid=2987
comm="secure_client" saddr=127.0.0.1 src=35900 daddr=127.0.0.1 dest=9999
netif=lo scontext=unconfined_u:unconfined_r:unconfined_t
tcontext=system_u:object_r:ext_gateway_packet_t tclass=packet
```

The reader can experiment with the remaining scenarios to find if there are any holes in the configuration.

2.2.2 Points to Note

2.2.2.1 Importance of Loading the `iptables`

The external gateway policy module relies on the fact that the `iptables` are loaded correctly to label the network packets. If they are not loaded, or (for example) the command:

```
iptables -t security -F
```

was allowed to be run that removes the `security` table entries, then the network packets would be labeled with the initial SID default of `unconfined_t`. The result is of course that all packets would be allowed. For example, running the

secure_client and standard server on port 9999 with no iptables loaded would have the following audit.log entries (as all traffic on all ports would flow, as no policy is being enforced):

```
type=AVC msg=audit(1247241956.542:32): avc:  granted  { transition } for
pid=2876 comm="bash" path="/usr/local/bin/secure_client" dev=dm-0 ino=354307
scontext=unconfined_u:unconfined_r:unconfined_t
tcontext=unconfined_u:message_filter_r:ext_gateway_t tclass=process
type=SYSCALL msg=audit(1247241956.542:32): arch=40000003 syscall=11 success=yes
exit=0 a0=9474a68 a1=947f460 a2=946d8e8 a3=0 items=0 ppid=2634 pid=2876 auid=0
uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=1
comm="secure_client" exe="/usr/local/bin/secure_client"
subj=unconfined_u:message_filter_r:ext_gateway_t key=(null)
```

Compare this audit.log trail with those shown in Test 4 that was run using the same scenario except that the iptables were loaded, thus denying the recv.

2.2.2.2 Running tests out of sequence

The server component allows files to be created in an 'in queue', and read / unlinked for the 'out queue' when running the message filter test. However should the message filter tests be run (see the Testing the Message Filter Build section) before the internal gateway and file mover loadable modules are loaded, the following will be noted:

1. When running the unconfined client / server, files can be written (server 1234 in with client 127.0.0.1 1234), moved (move_file) and then read / unlinked (server 1234 out with client 127.0.0.1 1234). This is because the base policy allows unconfined_t to do anything it likes.
2. When running the secure client / server, files cannot be written to the 'in queue' or read / unlinked from the 'out queue'. This is because the ext_gateway_t process does not have the required allow permissions to do this.

2.3 Building the NetLabel Loadable Module

This simple module enables a NetLabel netlabel_peer_t label to be added to the network connection to show that additional information at the peer level (as secmark handles packet level labeling) can be added.

Because this basic policy has the Policy Capabilities[1] network_peer_controls set to '0', the full peer controls are not enabled, however the legacy implementation will use the tcp_socket object class with the recvfrom permission to manage peer labeling for this example.

For an example where the network_peer_controls is set to '1', allowing the use of the new controls.

The following steps need to be followed to build the test services (it is assumed that the files are built in the notebook-source/basic-selinux-policy/message-filter/netlabel directory):

[1] See the SELinux Filesystem section in 'The Foundations' volume.

1. Ensure you are logged on as 'root' and SELinux is running in permissive mode (`setenforce 0`) to perform the build process.
2. Download and install the NetLabel rpm:

```
yum install netlabel_tools
# yum will then install netlabel_tools
```

3. Produce a `netlabel.conf` loadable module file with a text editor (such as `vi` or `gedit`) containing the contents shown below:

```
module netlabel 1.1.0;
#
################################################################################
#                                                                              #
# This Loadable Module will allow the netlabels to be added and checked       #
# within the client / server applications that form part of the SECMARK        #
# test examples.                                                               #
# Note: This module assumes that:                                              #
#        /selinux/policy_capabilities/network_peer_controls = 0                #
#                                                                              #
# (1) Install netlabel_tools (yum install netlabel_tools)                      #
#                                                                              #
# (2) Install this loadable module.                                            #
#                                                                              #
# (3) Run the following netlabelctl command:                                   #
#        netlabelctl unlbl add interface:lo address:127.0.0.1 \                #
#                    label:system_u:object_r:netlabel_peer_test_t              #
#                                                                              #
# (4) Run netlabelctl -p unlbl list command to check all is okay.              #
#                                                                              #
# (5) Run the secure and standard client/server that should now display        #
#     the netlabel_peer_test_t as the peer context.                            #
#                                                                              #
################################################################################
#

require {
   type ext_gateway_t, unconfined_t;
   class tcp_socket { recvfrom };
}
type netlabel_peer_test_t;
type socket_t;

# These are used when /selinux/policy_capabilities/network_peer_controls = 0
allow ext_gateway_t netlabel_peer_test_t : tcp_socket recvfrom;
allow unconfined_t netlabel_peer_test_t : tcp_socket recvfrom;

#
####################### START OPTIONAL SECTION ##############################
#
optional {
   require {
       # This is defined in the int_gateway.conf module:
       type int_gateway_t;
   }
   allow int_gateway_t netlabel_peer_test_t : tcp_socket recvfrom;
}
#
######################### END OPTIONAL SECTION ##############################
#
```

4. Compile and install the module as follows:

```
checkmodule -m netlabel.conf -o netlabel.mod
semodule_package -o netlabel.pp -m netlabel.mod
semodule -v -s modular-test -i netlabel.pp
```

5. Run the following command to add the `netlabel_peer_test_t` label as follows:

```
netlabelctl unlbl add interface:lo address:127.0.0.1 \
    label:system_u:object_r:netlabel_peer_test_t
```

6. Run enforcing mode:

```
setenforce 1
```

7. Run either the client / server or secure_client / secure_server applications as shown in the SECMARK tests. There should now be a peer context displayed as shown in Figure 2.4.

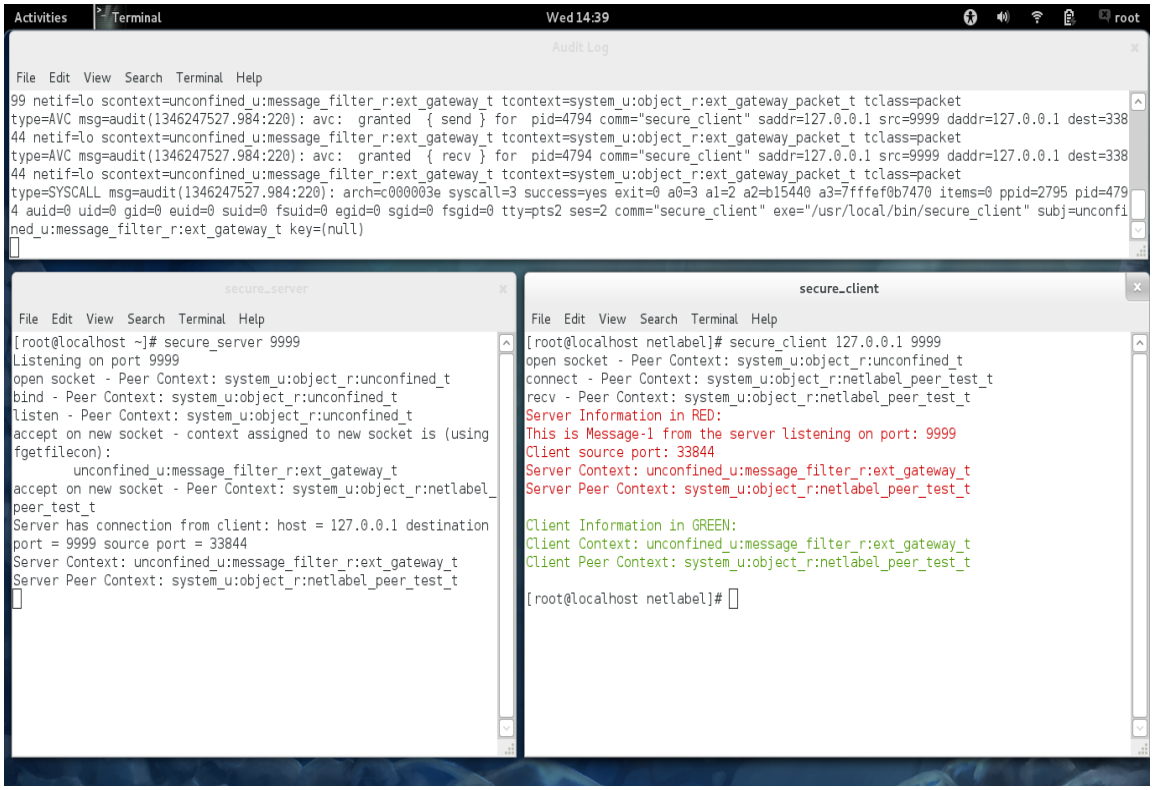

Figure 2.4: Running the secure client / server with NetLabel enabled

To remove the NetLabel label, the following command can be run:

```
netlabelctl unlbl del interface:lo address:127.0.0.1 \
  label:system_u:object_r:netlabel_peer_test_t
```

2.4 Building the Remaining Message Filter Service

To complete the overall message filter shown in Figure 2.1, the internal gateway and file mover applications and policy modules need to be built. These are explained in this section plus how to test the modules via simple helper scripts. The source and scripts are included in the source code rpm package.

The following will be built in this section:

1. The internal gateway policy module.
2. The file mover application.
3. The file mover policy module.

2.4.1 Internal Gateway Loadable Policy Module

This loadable module will apply policy rules for the internal gateway. The policy applies `dontaudit` rules for those permissions known not to cause problems.

The following steps need to be followed to build the internal gateway module. It is assumed that the services are installed in `notebook-source/basic-selinux-policy/kernel-language/message-filter/gateways`:

1. Ensure you are logged on as 'root' and SELinux is running in permissive mode (`setenforce 0`) to perform the build process.
2. Use the `int_gateway.conf` loadable module file supplied in the `gateways` directory.
3. Compile the policy with `checkmodule` to produce an intermediate binary policy file:

```
checkmodule -m int_gateway.conf -o int_gateway.mod
```

   The output from the compilation should be:

```
checkmodule:  loading policy configuration from base.conf
checkmodule:  policy configuration loaded
checkmodule:  writing binary representation to base.mod
```

4. Package the policy with `semodule_package`, this will produce a policy module file (note – if successful there are no output messages):

```
semodule_package -o int_gateway.pp -m int_gateway.mod
```

5. Install the loadable module with `semodule` (note – if successful there are no output messages):

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

6. If there are no errors reported, then the loadable module has been added to the policy store and loaded as a part of the policy. The policy module can be checked by:

```
semodule -s modular-test -l
```

   The results should be:

```
ext_gateway 1.1.0
int_gateway 1.1.0
```

```
netlabel    1.1.0
```

The file mover application can now be built.

2.4.2 File Move Application

This 'C' program will move files from one directory to another and works in conjunction with the `move_file.conf` loadable module that will apply the policy rules.

The following steps need to be followed to build the file move application and it is assumed that the services are installed in `notebook-source/basic-selinux-policy/message-filter/move_file`:

1. Use the `move_file.c` program supplied in the `move_file` directory.

2. Compile the `move_file.c` program:

   ```
   gcc -o move_file move_file.c
   ```

3. Move the binary to `/usr/local/bin`:

   ```
   mv move_file /usr/local/bin
   ```

To complete the message filter, the file mover loadable module will now be built.

2.4.3 File Mover Loadable Policy Module

This loadable module will allow a file to be moved from one directory to another using the file mover application built above with minimum privileges. The policy applies `dontaudit` rules for those permissions known not to cause problems.

Note that in the policy there is a statement that allows a counter to be displayed on the console for testing purposes.

The following steps need to be followed to build the file mover module and it is assumed that the services are installed in `notebook-source/basic-selinux-policy/message-filter/move_file`:

1. Ensure you are logged on as ‘root’ and SELinux is running in permissive mode (`setenforce 0`) to perform the build process.

2. Use the `move_file.conf` module supplied in the `move_file` directory.

3. Produce a `move_file.fc` file (a segment that will be added to `file_contexts` file during the build) with the contents shown below. This will be used to relabel application files and directories.

   ```
   # The Move File process makes use of two directory structures
   # (in & out) that are labeled as follows:

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

   # Ensure that any files are also relabeled:
   ```

```
/usr/message_queue/in_queue(/.*)? -- system_u:object_r:in_file_t
/usr/message_queue/out_queue(/.*)? -- system_u:object_r:out_file_t

# The Move File 'C' application is labeled:
/usr/local/bin/move_file -- system_u:object_r:move_file_exec_t
```

4. Produce a `restorecon_files` file with the contents shown below. This will be used by the `restorecon` command to relabel application files and directories after any updates.

```
/usr/message_queue/in_queue
/usr/message_queue/out_queue
/usr/local/bin/move_file
```

5. Compile the policy with `checkmodule` to produce an intermediate binary policy file:

```
checkmodule -m move_file.conf -o move_file.mod
```

The output from the compilation should be:

```
checkmodule:  loading policy configuration from base.conf
checkmodule:  policy configuration loaded
checkmodule:  writing binary representation to base.mod
```

6. Package the policy with `semodule_package`, this will produce a policy module file (note – if successful there are no output messages):

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

7. Make the directories required by the application. These need to be created because when `semodule` loads the policy, it will run `setfiles` to set the file contexts correctly (using the contents of the `move_file.fc` file produced in step 3).

```
mkdir -p /usr/message_queue/in_queue
mkdir -p /usr/message_queue/out_queue
```

8. Install the loadable module with `semodule` (note – if successful there are no output messages):

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

9. If there are no errors reported, then the loadable module has been added to the policy store and loaded as a part of the policy. The policy module can be checked by:

```
semodule -s modular-test -l
```

The results should be:

```
ext_gateway 1.1.0
int_gateway 1.1.0
```

```
move_file    1.1.0
netlabel     1.1.0
```

10. Uncomment the internal gateway entries in the `iptables` file (`notebook-source/basic-selinux-policy/kernel-language/message-filter/gateways/iptables_secmark`) that was produced in step 13 of the Building the SECMARK Test Loadable Module section:

```
....
# These are not required until using the internal gateway:
iptables -t security -A INPUT -i lo -p tcp --dport 1111 -j SECMARK
--selctx system_u:object_r:int_gateway_packet_t
iptables -t security -A INPUT -i lo -p tcp --sport 1111 -j SECMARK
--selctx system_u:object_r:int_gateway_packet_t
.....
....
#-------------- OUTPUT IP Stream --------------------#
....
#
# These are not required until using the internal gateway:
iptables -t security -A OUTPUT -o lo -p tcp --dport 1111 -j SECMARK
--selctx system_u:object_r:int_gateway_packet_t
iptables -t security -A OUTPUT -o lo -p tcp --sport 1111 -j SECMARK
--selctx system_u:object_r:int_gateway_packet_t
....
```

11. Ensure all the files are correctly labeled by running the `restorecon` command using the input file produced in step 4 above:

```
restorecon -r -f restorecon_file
```

12. Run enforcing mode:

```
setenforce 1
```

The message filter should now be ready to test.

2.4.4 Testing the Message Filter Build

To test the message filter it is recommended that four virtual terminal sessions are opened (as shown in Figure 2.5) for:

1. Running the external gateway client as it will display status messages if successful. This is shown on bottom left hand side using port 9999. Note that this process is run directly from the command line by `secure_server 9999` as it will automatically transition to the `ext_gateway_t` domain by the policy rules.

1. Running the internal gateway client as it will display status messages if successful.. This is shown on bottom right hand side using port 1111. Note that this process (and the secure server for the internal gateway) has to be run via the `runcon` command because of the type enforcement rules discussed in the Type Enforcement Rules section of 'The Foundations' volume.

2. Running the servers as they display messages when connections are made with the clients.

3. Viewing the audit log file. Note that the module has `auditallow` rules on `packet { send recv }` so that these events can be seen. This is top left.
4. Starting and viewing the file mover application as this will be run to display a count of the files being moved. This is top right.

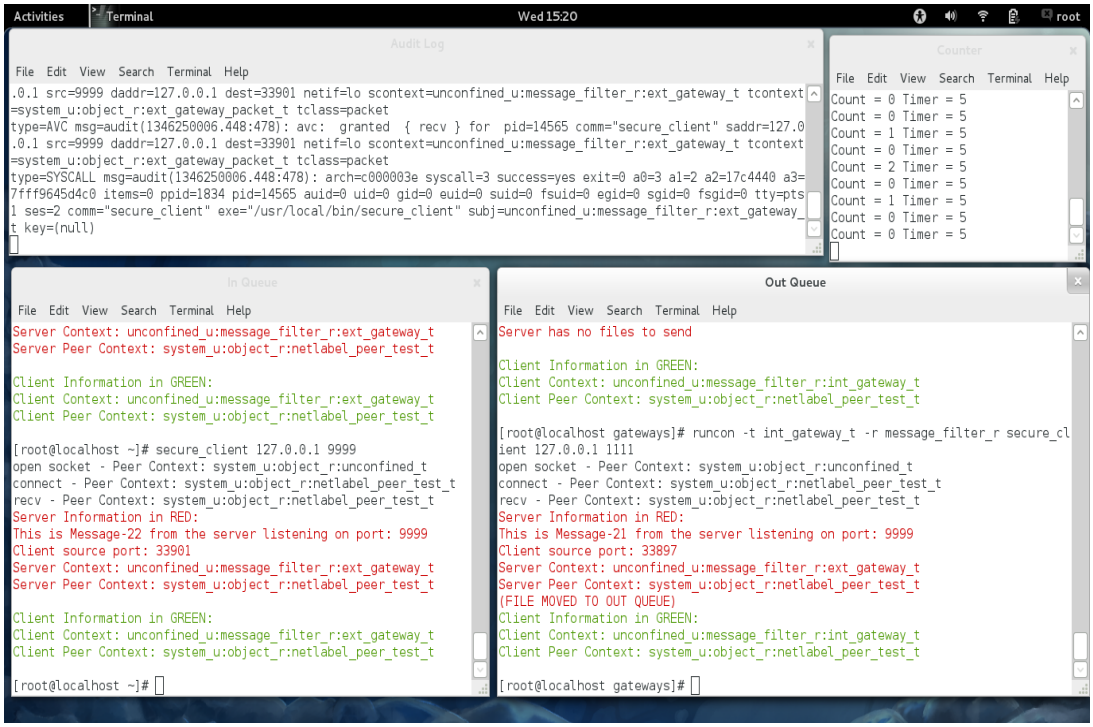

Figure 2.5: Testing the message filter service

If there are four terminal sessions logged in as root as shown in Figure 2.5, then the follow commands will need to be executed to show the message filter is working:

1. In the session that will display the audit log, execute the following command:

```
tail -f /var/log/audit/audit.log.
```

2. In a session run the following command to load the `iptables` (it is assumed that the current directory is where the file is located):

```
./iptables_secmark
```

3. Each of the server processes for the gateways will be run in background using one of the sessions with the following commands:

```
# Start the external gateway in background with the 'in' argument
# so that files are created in the in_queue with the communications
# traffic:

secure_server 9999 in &
```

```
# Start the internal gateway in background using the runcon
# command with the 'out' argument so that files are read from the
# out_queue:
```

```
runcon -t int_gateway_t -r message_filter_r secure_server 1111 out &
```

4. In a session start the file mover application with a time in seconds argument so that it will loop and display the number of files moved:

```
move_file 5
```

5. In a session start the secure external gateway client:

```
secure_client 127.0.0.1 9999
```

6. In a session start the secure internal gateway client using the `runcon` command:

```
runcon  -t int_gateway_t -r message_filter_r secure_client 127.0.0.1 1111
```

7. Keep repeating the client commands shown in steps 5 and 6 and the messages should be displayed in each window as the clients are run.

   If the external gateway client is run a number of times, the messages will be read from the `in_queue` by the file mover and queued to the `out_queue`, the internal gateway client can then be run to read each message off the `out_queue`. The queues can be investigated for their context by using `ls -Z`, however to do this, enforcing mode must be off otherwise `unconfined_t` (that is the logon sessions domain) cannot read these areas.