Red Hat Enterprise Linux 6

Managing Confined Services

Guide to configuring services under control of SELinux

Red Hat Enterprise Linux 6 Managing Confined Services
Guide to configuring services under control of SELinux
Edition 1.6

Author content-services-list@redhat.com

1801 Varsity Drive
Raleigh, NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701

The Managing Confined Services guide is designed to assist advanced users and administrators when using and configuring Security-Enhanced Linux (SELinux). It is focused on Red Hat Enterprise Linux and describes the components of SELinux as they pertain to services an advanced user or administrator might need to configure. Also included are real-world examples of configuring these services and demonstrations of how SELinux complements their operation.

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[1] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F2`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F1`** to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`mono-spaced bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click

[1] https://fedorahosted.org/liberation-fonts/

> **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

Mono-spaced Bold Italic or ***Proportional Bold Italic***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type **ssh *username*@*domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.
>
> The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.
>
> To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;
```

```
public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: *http://bugzilla.redhat.com/* against the product **Red Hat Enterprise Linux.**

When submitting a bug report, be sure to mention the manual's identifier: *doc-SELinux_Managing_Confined_Services_Guide* and version number: **6**.

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Introduction

Security-Enhanced Linux (SELinux) is an implementation of a *mandatory access control* mechanism in the Linux kernel, checking for allowed operations after standard *discretionary access controls* are checked. It was created by the National Security Agency and can enforce rules on files and processes in a Linux system, and on their actions, based on defined policy.

Security-Enhanced Linux (SELinux) refers to files, such as directories and devices, as objects. Processes, such as a user running a command or the Mozilla Firefox application, are referred to as subjects. Most operating systems use a Discretionary Access Control (DAC) system that controls how subjects interact with objects, and how subjects interact with each other. On operating systems using DAC, users control the permissions of files (objects) that they own. For example, on Linux operating systems, users could make their home directories world-readable, inadvertently giving other users and processes (subjects) access to potentially sensitive information.

DAC access decisions are only based on user identity and ownership, ignoring other security-relevant information such as the role of the user, the function and trustworthiness of the program, and the sensitivity and integrity of the data. Each user usually has complete discretion over their files, making it difficult to enforce a system-wide security policy. Furthermore, every program run by a user inherits all of the permissions granted to the user and is free to change access to the user's files, so minimal protection is provided against malicious software. Many system services and privileged programs must run with coarse-grained privileges that far exceed their requirements, so that a flaw in any one of these programs might be exploited to obtain further system access. [1]

The following is an example of permissions used on Linux operating systems that do not run Security-Enhanced Linux (SELinux). The permissions in these examples may differ from your system. Use the **`ls -l`** command to view file permissions:

```
$ ls -l file1
-rwxrw-r-- 1 user1 group1 0 2010-01-29 09:17 file1
```

The first three permission bits, **`rwx`**, control the access rights that the Linux **`user1`** user (in this case, the owner) has to **`file1`**. The next three permission bits, **`rw-`**, control the access rights that the Linux **`group1`** group has to **`file1`**. The last three permission bits, **`r--`**, control the access rights that everyone else has to **`file1`**, which includes all users and processes.

Security-Enhanced Linux (SELinux) adds Mandatory Access Control (MAC) to the Linux kernel, and is enabled by default in Red Hat Enterprise Linux. A general purpose MAC architecture needs the ability to enforce an administratively-set security policy over all processes and files in the system, basing decisions on labels containing a variety of security-relevant information. When properly implemented, it enables a system to adequately defend itself and offers critical support for application security by protecting against the tampering with, and bypassing of, secured applications. MAC provides strong separation of applications that permits the safe execution of untrustworthy applications. Its ability to limit the privileges associated with executing processes limits the scope of potential damage that can result from the exploitation of vulnerabilities in applications and system services. MAC enables information to be protected from legitimate users with limited authorization as well as from authorized users who have unwittingly executed malicious applications.[2]

[1] "Integrating Flexible Support for Security Policies into the Linux Operating System", by Peter Loscocco and Stephen Smalley. This paper was originally prepared for the National Security Agency and is, consequently, in the public domain. Refer to the *original paper* [http://www.nsa.gov/research/_files/selinux/papers/freenix01/index.shtml] for details.

[2] "Meeting Critical Security Objectives with Security-Enhanced Linux", by Peter Loscocco and Stephen Smalley. This paper was originally prepared for the National Security Agency and is in the public domain. Refer to the *original paper* [http://www.nsa.gov/research/_files/selinux/papers/ottawa01/index.shtml] for details.

The following is an example of the labels containing security-relevant information that are applied to processes, Linux users, and files, on Linux operating systems that run SELinux. This information is called the SELinux context, and is viewed using the **ls -Z** command:

```
$ ls -Z file1
-rwxrw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0      file1
```

In this example, SELinux provides a user (**unconfined_u**), a role (**object_r**), a type (**user_home_t**), and a level (**s0**) for the **file1** file. This information is used to make access control decisions. This example also displays the DAC rules, which are shown in the SELinux context via the **ls -Z** command. SELinux policy rules are checked after DAC rules. SELinux policy rules are not applied if DAC rules deny access first.

Targeted policy

Targeted policy is the default SELinux policy used in Red Hat Enterprise Linux. When using targeted policy, processes that are targeted run in a confined domain, and processes that are not targeted run in an unconfined domain. For example, by default, logged in users run in the **unconfined_t** domain, and system processes started by **init** run in the **initrc_t** domain - both of these domains are unconfined.

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you must tell SELinux how you are running services. This can be achieved via Booleans that allow parts of SELinux policy to be changed at runtime by the administrator, without requiring any knowledge of SELinux policy writing. This allows changes, such as allowing services access to NFS file systems, without having to reload or recompile SELinux policy. Boolean configuration is discussed throughout this guide using detailed examples.

Other changes, such as using non-default directories to store files for services, and changing services to run on non-default port numbers, require policy configuration to be updated via tools such as the **semanage** command, which is provided by the *policycoreutils-python* package. This command is discussed throughout this guide using detailed configuration examples.

2.1. Type Enforcement

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

2.2. Confined processes

Almost every service that listens on a network is confined in Red Hat Enterprise Linux. Also, most processes that run as the **root** user and perform tasks for users, such as the **passwd** application, are confined. When a process is confined, it runs in its own domain, such as the `httpd` process running in the **httpd_t** domain. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited.

The following example demonstrates how SELinux prevents the Apache HTTP Server (`httpd`) from reading files that are not correctly labeled, such as files intended for use by Samba. This is an example, and should not be used in production. It assumes that the *httpd*, *wget*, *setroubleshoot-server* and *audit* packages are installed, that the SELinux targeted policy is used, and that SELinux is running in enforcing mode:

1. Run the **sestatus** command to confirm that SELinux is enabled, is running in enforcing mode, and that targeted policy is being used:

```
$ /usr/sbin/sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 24
Policy from config file:        targeted
```

SELinux status: enabled is returned when SELinux is enabled. **Current mode: enforcing** is returned when SELinux is running in enforcing mode. **Policy from config file: targeted** is returned when the SELinux targeted policy is used.

2. As the root user, run the **touch /var/www/html/testfile** command to create a file.

3. Run the **ls -Z /var/www/html/testfile** command to view the SELinux context:

```
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 /var/www/html/testfile
```

   The **testfile** file is labeled with the SELinux **unconfined_u** user because a Linux user that is mapped to the **unconfined_u** SELinux user created the file. Role-Based Access Control (RBAC) is used for processes, not files. Roles do not have a meaning for files - the **object_r** role is a generic role used for files (on persistent storage and network file systems). Under the **/proc/** directory, files related to processes may use the **system_r** role.[1] The **httpd_sys_content_t** type allows the `httpd` process to access this file.

4. As the root user, run the **service httpd start** command to start the `httpd` process. The output is as follows if `httpd` starts successfully:

```
# /sbin/service httpd start
Starting httpd:                                            [  OK  ]
```

5. Change into a directory where your Linux user has write access to, and run the **wget http://localhost/testfile** command. Unless there are changes to the default configuration, this command succeeds:

```
--2009-12-01 11:40:28--  http://localhost/testfile
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 0 [text/plain]
Saving to: `testfile'

[ <=>                                ] 0     --.-K/s   in 0s

2009-12-01 11:40:28 (0.00 B/s) - `testfile' saved [0/0]
```

6. The **chcon** command relabels files; however, such label changes do not survive when the file system is relabeled. For permanent changes that survive a file system relabel, use the **semanage** command, which is discussed later. As the root user, run the following command to change the type to a type used by Samba:

   chcon -t samba_share_t /var/www/html/testfile

   Run the **ls -Z /var/www/html/testfile** command to view the changes:

```
-rw-r--r--  root root unconfined_u:object_r:samba_share_t:s0 /var/www/html/testfile
```

7. Note: the current DAC permissions allow the `httpd` process access to **testfile**. Change into a directory where your Linux user has write access to, and run the **wget http://localhost/testfile** command. Unless there are changes to the default configuration, this command fails:

```
--2009-12-01 11:43:18--  http://localhost/testfile
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 403 Forbidden
2009-12-01 11:43:18 ERROR 403: Forbidden.
```

8. As the root user, run the **rm /var/www/html/testfile** command to remove **testfile**.

9. If you do not require httpd to be running, as the root user, run the **service httpd stop** command to stop httpd:

```
# /sbin/service httpd stop
Stopping httpd:                                            [  OK  ]
```

This example demonstrated the additional security added by SELinux. DAC rules allowed the httpd process access to **testfile** in step 5, but because the file was then labeled with a type that the httpd process does not have access to, SELinux denied access. After step 7, if the *setroubleshoot-server* package is installed, an error similar to the following is logged to **/var/log/messages**:

```
setroubleshoot: SELinux is preventing httpd (httpd_t) "getattr" to /var/www/html/testfile
 (samba_share_t). For complete SELinux messages run sealert -l c05911d3-e680-4e42-8e36-
fe2ab9f8e654
```

Also, an error similar to the following is logged to **/var/log/httpd/error_log**:

```
[Tue Dec 01 11:43:18 2009] [error] [client 127.0.0.1] (13)Permission denied: access to /
testfile denied
```

2.3. Unconfined processes

Unconfined processes run in unconfined domains. For example, **init** programs run in the unconfined **initrc_t** domain, unconfined kernel processes run in the **kernel_t** domain, and unconfined Linux users run in the **unconfined_t** domain. For unconfined processes, SELinux policy rules are still applied, but policy rules exist that allow processes running in unconfined domains almost all access. Processes running in unconfined domains fall back to using DAC rules exclusively. If an unconfined process is compromised, SELinux does not prevent an attacker from gaining access to system resources and data, but of course, DAC rules are still used. SELinux is a security enhancement on top of DAC rules - it does not replace them.

The following example demonstrates how the Apache HTTP Server (httpd) can access data intended for use by Samba, when running unconfined. Note: in Red Hat Enterprise Linux, the httpd process runs in the confined **httpd_t** domain by default. This is an example, and should not be used in production. It assumes that the *httpd* and *wget* packages are installed, that SELinux targeted policy is used, and that SELinux is running in enforcing mode:

1. Run the **sestatus** command to confirm that SELinux is enabled, is running in enforcing mode, and that targeted policy is being used:

```
$ /usr/sbin/sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 24
Policy from config file:        targeted
```

SELinux status: enabled is returned when SELinux is enabled. **Current mode: enforcing** is returned when SELinux is running in enforcing mode. **Policy from config file: targeted** is returned when the SELinux targeted policy is used.

2. As the root user, run the **touch /var/www/html/test2file** command to create a file.

3. Run the **ls -Z /var/www/html/test2file** command to view the SELinux context:

```
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 /var/www/html/
test2file
```

test2file is labeled with the SELinux **unconfined_u** user because a Linux user that is mapped to the **unconfined_u** SELinux user created the file. RBAC is used for processes, not files. Roles do not have a meaning for files - the **object_r** role is a generic role used for files (on persistent storage and network file systems). Under the **/proc/** directory, files related to processes may use the **system_r** role.[2] The **httpd_sys_content_t** type allows the `httpd` process to access this file.

4. The **chcon** command relabels files; however, such label changes do not survive when the file system is relabeled. For permanent changes that survive a file system relabel, use the **semanage** command, which is discussed later. As the root user, run the following command to change the type to a type used by Samba:

chcon -t samba_share_t /var/www/html/test2file

Run the **ls -Z /var/www/html/test2file** command to view the changes:

```
-rw-r--r--  root root unconfined_u:object_r:samba_share_t:s0 /var/www/html/test2file
```

5. Run the **service httpd status** command to confirm that the `httpd` process is not running:

```
$ /sbin/service httpd status
httpd is stopped
```

If the output differs, run the **service httpd stop** command as the root user to stop the `httpd` process:

```
# /sbin/service httpd stop
Stopping httpd:                                            [  OK  ]
```

6. To make the `httpd` process run unconfined, run the following command as the root user to change the type of **/usr/sbin/httpd**, to a type that does not transition to a confined domain:

chcon -t unconfined_exec_t /usr/sbin/httpd

7. Run the **ls -Z /usr/sbin/httpd** command to confirm that **/usr/sbin/httpd** is labeled with the **unconfined_exec_t** type:

```
-rwxr-xr-x  root root system_u:object_r:unconfined_exec_t /usr/sbin/httpd
```

8. As the root user, run the **service httpd start** command to start the `httpd` process. The output is as follows if `httpd` starts successfully:

```
# /sbin/service httpd start
Starting httpd:                                            [  OK  ]
```

9. Run the **ps -eZ | grep httpd** command to view the `httpd` processes running in the **unconfined_t** domain:

```
$ ps -eZ | grep httpd
unconfined_u:system_r:unconfined_t 7721 ?      00:00:00 httpd
unconfined_u:system_r:unconfined_t 7723 ?      00:00:00 httpd
unconfined_u:system_r:unconfined_t 7724 ?      00:00:00 httpd
unconfined_u:system_r:unconfined_t 7725 ?      00:00:00 httpd
unconfined_u:system_r:unconfined_t 7726 ?      00:00:00 httpd
unconfined_u:system_r:unconfined_t 7727 ?      00:00:00 httpd
unconfined_u:system_r:unconfined_t 7728 ?      00:00:00 httpd
unconfined_u:system_r:unconfined_t 7729 ?      00:00:00 httpd
unconfined_u:system_r:unconfined_t 7730 ?      00:00:00 httpd
```

10. Change into a directory where your Linux user has write access to, and run the **wget http://localhost/test2file** command. Unless there are changes to the default configuration, this command succeeds:

```
--2009-12-01 11:55:47--  http://localhost/test2file
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 0 [text/plain]
Saving to: `test2file.1'

[ <=>                                 ]--.-K/s   in 0s

2009-12-01 11:55:47 (0.00 B/s) - `test2file.1' saved [0/0]
```

Although the `httpd` process does not have access to files labeled with the **samba_share_t** type, `httpd` is running in the unconfined **unconfined_t** domain, and falls back to using DAC rules, and as such, the **wget** command succeeds. Had `httpd` been running in the confined **httpd_t** domain, the **wget** command would have failed.

11. The **restorecon** command restores the default SELinux context for files. As the root user, run the **restorecon -v /usr/sbin/httpd** command to restore the default SELinux context for **/usr/sbin/httpd**:

```
# /sbin/restorecon -v /usr/sbin/httpd
restorecon reset /usr/sbin/httpd context system_u:object_r:unconfined_exec_t:s0-
>system_u:object_r:httpd_exec_t:s0
```

Run the **ls -Z /usr/sbin/httpd** command to confirm that **/usr/sbin/httpd** is labeled with the **httpd_exec_t** type:

```
$ ls -Z /usr/sbin/httpd
-rwxr-xr-x  root root system_u:object_r:httpd_exec_t   /usr/sbin/httpd
```

12. As the root user, run the **/sbin/service httpd restart** command to restart `httpd`. After restarting, run the **ps -eZ | grep httpd** to confirm that `httpd` is running in the confined **httpd_t** domain:

```
# /sbin/service httpd restart
Stopping httpd:                                            [  OK  ]
Starting httpd:                                            [  OK  ]
# ps -eZ | grep httpd
unconfined_u:system_r:httpd_t    8880 ?        00:00:00 httpd
unconfined_u:system_r:httpd_t    8882 ?        00:00:00 httpd
unconfined_u:system_r:httpd_t    8883 ?        00:00:00 httpd
unconfined_u:system_r:httpd_t    8884 ?        00:00:00 httpd
```

```
unconfined_u:system_r:httpd_t    8885 ?        00:00:00 httpd
unconfined_u:system_r:httpd_t    8886 ?        00:00:00 httpd
unconfined_u:system_r:httpd_t    8887 ?        00:00:00 httpd
unconfined_u:system_r:httpd_t    8888 ?        00:00:00 httpd
unconfined_u:system_r:httpd_t    8889 ?        00:00:00 httpd
```

13. As the root user, run the **`rm /var/www/html/test2file`** command to remove **`test2file`**.

14. If you do not require `httpd` to be running, as the root user, run the **`service httpd stop`** command to stop `httpd`:

```
# /sbin/service httpd stop
Stopping httpd:                                            [  OK  ]
```

The examples in these sections demonstrated how data can be protected from a compromised confined process (protected by SELinux), as well as how data is more accessible to an attacker from a compromised unconfined process (not protected by SELinux).

The Apache HTTP Server

From the *Apache HTTP Server Project*[1] page:

"The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards".[2]

In Red Hat Enterprise Linux, the *httpd* package provides the Apache HTTP Server. Run **rpm -q httpd** to see if the *httpd* package is installed. If it is not installed and you want to use the Apache HTTP Server, run the following command as the root user to install it:

```
yum install httpd
```

3.1. The Apache HTTP Server and SELinux

When SELinux is enabled, the Apache HTTP Server (`httpd`) runs confined by default. Confined processes run in their own domains, and are separated from other confined processes. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited. The following example demonstrates the `httpd` processes running in their own domain. This example assumes the *httpd*, *setroubleshoot*, *setroubleshoot-server* and *policycoreutils-python* packages are installed:

1. Run **getenforce** to confirm SELinux is running in enforcing mode:

   ```
   $ getenforce
   Enforcing
   ```

   The **getenforce** command returns **Enforcing** when SELinux is running in enforcing mode.

2. Run **service httpd start** as the root user to start `httpd`:

   ```
   # service httpd start
   Starting httpd:                                            [  OK  ]
   ```

3. Run **ps -eZ | grep httpd** to view the `httpd` processes:

   ```
   $ ps -eZ | grep httpd
   unconfined_u:system_r:httpd_t:s0 2850 ?        00:00:00 httpd
   unconfined_u:system_r:httpd_t:s0 2852 ?        00:00:00 httpd
   unconfined_u:system_r:httpd_t:s0 2853 ?        00:00:00 httpd
   unconfined_u:system_r:httpd_t:s0 2854 ?        00:00:00 httpd
   unconfined_u:system_r:httpd_t:s0 2855 ?        00:00:00 httpd
   unconfined_u:system_r:httpd_t:s0 2856 ?        00:00:00 httpd
   unconfined_u:system_r:httpd_t:s0 2857 ?        00:00:00 httpd
   unconfined_u:system_r:httpd_t:s0 2858 ?        00:00:00 httpd
   ```

[1] http://httpd.apache.org/

[2] From the "The Number One HTTP Server On The Internet" section of the Apache HTTP Server Project page: *http://httpd.apache.org/*. Copyright © 2009 The Apache Software Foundation. Accessed 7 July 2010.

```
unconfined_u:system_r:httpd_t:s0 2859 ?        00:00:00 httpd
```

The SELinux context associated with the `httpd` processes is **`unconfined_u:system_r:httpd_t:s0`**. The second last part of the context, **`httpd_t`**, is the type. A type defines a domain for processes and a type for files. In this case, the `httpd` processes are running in the **`httpd_t`** domain.

SELinux policy defines how processes running in confined domains (such as **`httpd_t`**) interact with files, other processes, and the system in general. Files must be labeled correctly to allow `httpd` access to them. For example, `httpd` can read files labeled with the **`httpd_sys_content_t`** type, but can not write to them, even if Linux (DAC) permissions allow write access. Booleans must be turned on to allow certain behavior, such as allowing scripts network access, allowing `httpd` access to NFS and CIFS file systems, and `httpd` being allowed to execute Common Gateway Interface (CGI) scripts.

When **`/etc/httpd/conf/httpd.conf`** is configured so `httpd` listens on a port other than TCP ports 80, 443, 488, 8008, 8009, or 8443, the **`semanage port`** command must be used to add the new port number to SELinux policy configuration. The following example demonstrates configuring `httpd` to listen on a port that is not already defined in SELinux policy configuration for `httpd`, and, as a consequence, `httpd` failing to start. This example also demonstrates how to then configure the SELinux system to allow `httpd` to successfully listen on a non-standard port that is not already defined in the policy. This example assumes the *httpd* package is installed. Run each command in the example as the root user:

1. Run **`service httpd status`** to confirm `httpd` is not running:

   ```
   # service httpd status
   httpd is stopped
   ```

   If the output differs, run **`service httpd stop`** to stop the process:

   ```
   # service httpd stop
   Stopping httpd:                                            [  OK  ]
   ```

2. Run **`semanage port -l | grep -w http_port_t`** to view the ports SELinux allows `httpd` to listen on:

   ```
   # semanage port -l | grep -w http_port_t
   http_port_t                    tcp      80, 443, 488, 8008, 8009, 8443
   ```

3. Edit **`/etc/httpd/conf/httpd.conf`** as the root user. Configure the **`Listen`** option so it lists a port that is not configured in SELinux policy configuration for `httpd`. In this example, `httpd` is configured to listen on port 12345:

   ```
   # Change this to Listen on specific IP addresses as shown below to
   # prevent Apache from glomming onto all bound IP addresses (0.0.0.0)
   #
   #Listen 12.34.56.78:80
   Listen 127.0.0.1:12345
   ```

4. Run **`service httpd start`** to start `httpd`:

```
# service httpd start
Starting httpd: (13)Permission denied: make_sock: could not bind to address
 127.0.0.1:12345
no listening sockets available, shutting down
Unable to open logs        [FAILED]
```

An SELinux denial similar to the following is logged:

```
setroubleshoot: SELinux is preventing the httpd (httpd_t) from binding to port 12345. For
 complete SELinux messages. run sealert -l f18bca99-db64-4c16-9719-1db89f0d8c77
```

5. For SELinux to allow `httpd` to listen on port 12345, as used in this example, the following command is required:

```
# semanage port -a -t http_port_t -p tcp 12345
```

6. Run **`service httpd start`** again to start `httpd` and have it listen on the new port:

```
# service httpd start
Starting httpd:         [  OK  ]
```

7. Now that SELinux has been configured to allow `httpd` to listen on a non-standard port (TCP 12345 in this example), `httpd` starts successfully on this port.

8. To prove that `httpd` is listening and communicating on TCP port 12345, open a telnet connection to the specified port and issue a HTTP GET command, as follows:

```
# telnet localhost 12345
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 02 Dec 2009 14:36:34 GMT
Server: Apache/2.2.13 (Red Hat)
Accept-Ranges: bytes
Content-Length: 3985
Content-Type: text/html; charset=UTF-8
[...continues...]
```

3.2. Types

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

The following example creates a new file in the **`/var/www/html/`** directory, and shows the file inheriting the **`httpd_sys_content_t`** type from its parent directory (**`/var/www/html/`**):

1. Run **ls -dZ /var/www/html** to view the SELinux context of **/var/www/html/**:

```
$ ls -dZ /var/www/html
drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t:s0 /var/www/html
```

   This shows **/var/www/html/** is labeled with the **httpd_sys_content_t** type.

2. Run **touch /var/www/html/file1** as the root user to create a new file.

3. Run **ls -Z /var/www/html/file1** to view the SELinux context:

```
$ ls -Z /var/www/html/file1
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 /var/www/html/file1
```

The **ls -Z** command shows **file1** labeled with the **httpd_sys_content_t** type. SELinux allows `httpd` to read files labeled with this type, but not write to them, even if Linux permissions allow write access. SELinux policy defines what types a process running in the **httpd_t** domain (where `httpd` runs) can read and write to. This helps prevent processes from accessing files intended for use by another process.

For example, `httpd` can access files labeled with the **httpd_sys_content_t** type (intended for the Apache HTTP Server), but by default, can not access files labeled with the **samba_share_t** type (intended for Samba). Also, files in user home directories are labeled with the **user_home_t** type: by default, this prevents `httpd` from reading or writing to files in user home directories.

The following lists some of the types used with `httpd`. Different types allow you to configure flexible access:

httpd_sys_content_t
: Use this type for static web content, such as **.html** files used by a static website. Files labeled with this type are accessible (read only) to `httpd` and scripts executed by `httpd`. By default, files and directories labeled with this type can not be written to or modified by `httpd` or other processes. Note: by default, files created in or copied into **/var/www/html/** are labeled with the **httpd_sys_content_t** type.

httpd_sys_script_exec_t
: Use this type for scripts you want `httpd` to execute. This type is commonly used for Common Gateway Interface (CGI) scripts in **/var/www/cgi-bin/**. By default, SELinux policy prevents `httpd` from executing CGI scripts. To allow this, label the scripts with the **httpd_sys_script_exec_t** type and turn the **httpd_enable_cgi** Boolean on. Scripts labeled with **httpd_sys_script_exec_t** run in the **httpd_sys_script_t** domain when executed by `httpd`. The **httpd_sys_script_t** domain has access to other system domains, such as **postgresql_t** and **mysqld_t**.

httpd_sys_content_rw_t
: Files labeled with this type can be written to by scripts labeled with the **httpd_sys_script_exec_t** type, but can not be modified by scripts labeled with any other type. You must use the **httpd_sys_content_rw_t** type to label files that will be read from and written to by scripts labeled with the **httpd_sys_script_exec_t** type.

httpd_sys_content_ra_t
: Files labeled with this type can be appended to by scripts labeled with the **httpd_sys_script_exec_t** type, but can not be modified by scripts labeled with any other

type. You must use the **httpd_sys_content_ra_t** type to label files that will be read from and appended to by scripts labeled with the **httpd_sys_script_exec_t** type.

httpd_unconfined_script_exec_t
Scripts labeled with this type run without SELinux protection. Only use this type for complex scripts, after exhausting all other options. It is better to use this type instead of turning SELinux protection off for `httpd`, or for the entire system.

Note

To see more of the available types for httpd, run the following command:

```
grep httpd /etc/selinux/targeted/contexts/files/file_contexts
```

Changing the SELinux Context

The type for files and directories can be changed with the **chcon** command. Changes made with **chcon** do not survive a file system relabel or the **restorecon** command. SELinux policy controls whether users are able to modify the SELinux context for any given file. The following example demonstrates creating a new directory and an **index.html** file for use by `httpd`, and labeling that file and directory to allow `httpd` access to them:

1. Run **mkdir -p /my/website** as the root user to create a top-level directory structure to store files to be used by `httpd`.

2. Files and directories that do not match a pattern in file-context configuration may be labeled with the **default_t** type. This type is inaccessible to confined services:

```
$ ls -dZ /my
drwxr-xr-x  root root unconfined_u:object_r:default_t:s0 /my
```

3. Run **chcon -R -t httpd_sys_content_t /my/** as the root user to change the type of the **/my/** directory and subdirectories, to a type accessible to `httpd`. Now, files created under **/my/website/** inherit the **httpd_sys_content_t** type, rather than the **default_t** type, and are therefore accessible to httpd:

```
# chcon -R -t httpd_sys_content_t /my/
# touch /my/website/index.html
# ls -Z /my/website/index.html
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0 /my/website/index.html
```

Refer to the Temporary Changes: chcon section of the Red Hat Enterprise Linux 6 SELinux User Guide for further information about **chcon**.

Use the **semanage fcontext** command (**semanage** is provided by the *policycoreutils-python* package) to make label changes that survive a relabel and the **restorecon** command. This command adds changes to file-context configuration. Then, run the **restorecon** command, which reads file-context configuration, to apply the label change. The following example demonstrates creating a new directory and an **index.html** file for use by `httpd`, and persistently changing the label of that directory and file to allow `httpd` access to them:

1. Run **mkdir -p /my/website** as the root user to create a top-level directory structure to store files to be used by `httpd`.

2. Run the following command as the root user to add the label change to file-context configuration:

```
semanage fcontext -a -t httpd_sys_content_t "/my(/.*)?"
```

   The **"/my(/.*)?"** expression means the label change applies to the **/my/** directory and all files and directories under it.

3. Run **touch /my/website/index.html** as the root user to create a new file.

4. Run **restorecon -R -v /my/** as the root user to apply the label changes (**restorecon** reads file-context configuration, which was modified by the **semanage** command in step 2):

```
# restorecon -R -v /my/
restorecon reset /my context unconfined_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
restorecon reset /my/website context unconfined_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
restorecon reset /my/website/index.html context unconfined_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
```

Refer to the Persistent Changes: semanage fcontext section of the Red Hat Enterprise Linux SELinux User Guide for further information on semanage.

3.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you must tell SELinux how you are running services. This can be achieved via Booleans that allow parts of SELinux policy to be changed at runtime, without any knowledge of SELinux policy writing. This allows changes, such as allowing services access to NFS file systems, without reloading or recompiling SELinux policy.

To modify the state of a Boolean, use the **setsebool** command. For example, to turn the **allow_httpd_anon_write** Boolean on, run the following command as the root user:

```
# setsebool -P allow_httpd_anon_write on
```

To turn a Boolean off, using the same example, simply change **on** to **off** in the command, as shown below:

```
# setsebool -P allow_httpd_anon_write off
```

Note

Do not use the **-P** option if you do not want **setsebool** changes to persist across reboots.

Below is a description of common Booleans available that cater for the way `httpd` is running:

allow_httpd_anon_write
When disabled, this Boolean allows `httpd` to only have read access to files labeled with the **public_content_rw_t** type. Enabling this Boolean will allow `httpd` to write to files labeled with the **public_content_rw_t** type, such as a public directory containing files for a public file transfer service.

allow_httpd_mod_auth_ntlm_winbind
Enabling this Boolean allows access to NTLM and Winbind authentication mechanisms via the **mod_auth_ntlm_winbind** module in `httpd`.

allow_httpd_mod_auth_pam
Enabling this Boolean allows access to PAM authentication mechanisms via the **mod_auth_pam** module in `httpd`.

allow_httpd_sys_script_anon_write
This Boolean defines whether or not HTTP scripts are allowed write access to files labeled with the **public_content_rw_t** type, as used in a public file transfer service.

httpd_builtin_scripting
This Boolean defines access to `httpd` scripting. Having this Boolean enabled is often required for PHP content.

httpd_can_network_connect
When disabled, this Boolean prevents HTTP scripts and modules from initiating a connection to a network or remote port. Turn this Boolean on to allow this access.

httpd_can_network_connect_db
When disabled, this Boolean prevents HTTP scripts and modules from initiating a connection to database servers. Turn this Boolean on to allow this access.

httpd_can_network_relay
Turn this Boolean on when `httpd` is being used as a forward or reverse proxy.

httpd_can_sendmail
When disabled, this Boolean prevents HTTP modules from sending mail. This can prevent spam attacks should a vulnerability be found in `httpd`. Turn this Boolean on to allow HTTP modules to send mail.

httpd_dbus_avahi
When off, this Boolean denies `httpd` access to the **avahi** service via **D-Bus**. Turn this Boolean on to allow this access.

httpd_enable_cgi
When disabled, this Boolean prevents `httpd` from executing CGI scripts. Turn this Boolean on to allow `httpd` to execute CGI scripts (CGI scripts must be labeled with the **httpd_sys_script_exec_t** type).

httpd_enable_ftp_server
Turning this Boolean on will allow `httpd` to listen on the FTP port and act as an FTP server.

httpd_enable_homedirs
When disabled, this Boolean prevents `httpd` from accessing user home directories. Turn this Boolean on to allow `httpd` access to user home directories; for example, content in **/home/*/**.

httpd_execmem
When enabled, this Boolean allows `httpd` to execute programs that require memory addresses that are both executable and writeable. Enabling this Boolean is not recommended from a security standpoint as it reduces protection against buffer overflows, however certain modules and applications (such as Java and Mono applications) require this privilege.

httpd_ssi_exec
This Boolean defines whether or not server side include (SSI) elements in a web page can be executed.

httpd_tty_comm
This Boolean defines whether or not `httpd` is allowed access to the controlling terminal. Usually this access is not required, however in cases such as configuring an SSL certificate file, terminal access is required to display and process a password prompt.

httpd_unified
When enabled, this Boolean allows **httpd_t** complete access to all of the `httpd` types (i.e. to execute, read, or write sys_content_t). When disabled, there is separation in place between web content that is read-only, writeable or executable. Disabling this Boolean ensures an extra level of security but adds the administrative overhead of having to individually label scripts and other web content based on the file access that each should have.

httpd_use_cifs
Turn this Boolean on to allow `httpd` access to files on CIFS file systems that are labeled with the **cifs_t** type, such as file systems mounted via Samba.

httpd_use_nfs
Turn this Boolean on to allow `httpd` access to files on NFS file systems that are labeled with the **nfs_t** type, such as file systems mounted via NFS.

3.4. Configuration examples

The following examples provide real-world demonstrations of how SELinux complements the Apache HTTP Server and how full function of the Apache HTTP Server can be maintained.

3.4.1. Running a static site

To create a static website, label the **.html** files for that website with the **httpd_sys_content_t** type. By default, the Apache HTTP Server can not write to files that are labeled with the **httpd_sys_content_t** type. The following example creates a new directory to store files for a read-only website:

1. Run **mkdir /mywebsite** as the root user to create a top-level directory.

2. As the root user, create a **/mywebsite/index.html** file. Copy and paste the following content into **/mywebsite/index.html**:

```
<html>
<h2>index.html from /mywebsite/</h2>
</html>
```

3. To allow the Apache HTTP Server read only access to **/mywebsite/**, as well as files and subdirectories under it, label **/mywebsite/** with the **httpd_sys_content_t** type. Run the following command as the root user to add the label change to file-context configuration:

```
# semanage fcontext -a -t httpd_sys_content_t "/mywebsite(/.*)?"
```

4. Run **restorecon -R -v /mywebsite** as the root user to make the label changes:

```
# restorecon -R -v /mywebsite
restorecon reset /mywebsite context unconfined_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
restorecon reset /mywebsite/index.html context unconfined_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
```

5. For this example, edit **/etc/httpd/conf/httpd.conf** as the root user. Comment out the existing **DocumentRoot** option. Add a **DocumentRoot "/mywebsite"** option. After editing, these options should look as follows:

```
#DocumentRoot "/var/www/html"
DocumentRoot "/mywebsite"
```

6. Run **service httpd status** as the root user to see the status of the Apache HTTP Server. If the server is stopped, run **service httpd start** as the root user to start it. If the server is running, run **service httpd restart** as the root user to restart the service (this also applies any changes made to **httpd.conf**).

7. Use a web browser to navigate to **http://localhost/index.html**. The following is displayed:

```
index.html from /mywebsite/
```

3.4.2. Sharing NFS and CIFS file systems

By default, NFS mounts on the client side are labeled with a default context defined by policy for NFS file systems. In common policies, this default context uses the **nfs_t** type. Also, by default, Samba shares mounted on the client side are labeled with a default context defined by policy. In common policies, this default context uses the **cifs_t** type.

Depending on policy configuration, services may not be able to read files labeled with the **nfs_t** or **cifs_t** types. This may prevent file systems labeled with these types from being mounted and then read or exported by other services. Booleans can be turned on or off to control which services are allowed to access the **nfs_t** and **cifs_t** types.

Turn the **httpd_use_nfs** Boolean on to allow `httpd` to access and share NFS file systems (labeled with the **nfs_t** type. Run the **setsebool** command as the root user to turn the Boolean on:

```
setsebool -P httpd_use_nfs on
```

Turn the **httpd_use_cifs** Boolean on to allow `httpd` to access and share CIFS file systems (labeled with the **cifs_t** type. Run the **setsebool** command as the root user to turn the Boolean on:

```
setsebool -P httpd_use_cifs on
```

Note

Do not use the **-P** option if you do not want **setsebool** changes to persist across reboots.

3.4.3. Sharing files between services

Type Enforcement helps prevent processes from accessing files intended for use by another process. For example, by default, Samba can not read files labeled with the **httpd_sys_content_t** type, which are intended for use by the Apache HTTP Server. Files can be shared between the Apache HTTP Server, FTP, rsync, and Samba, if the desired files are labeled with the **public_content_t** or **public_content_rw_t** type.

The following example creates a directory and files, and allows that directory and files to be shared (read only) through the Apache HTTP Server, FTP, rsync, and Samba:

1. Run **mkdir /shares** as the root user to create a new top-level directory to share files between multiple services.

2. Files and directories that do not match a pattern in file-context configuration may be labeled with the **default_t** type. This type is inaccessible to confined services:

```
$ ls -dZ /shares
drwxr-xr-x  root root unconfined_u:object_r:default_t:s0 /shares
```

3. As the root user, create a **/shares/index.html** file. Copy and paste the following content into **/shares/index.html**:

```
<html>
<body>
<p>Hello</p>
</body>
</html>
```

4. Labeling **/shares/** with the **public_content_t** type allows read-only access by the Apache HTTP Server, FTP, rsync, and Samba. Run the following command as the root user to add the label change to file-context configuration:

```
semanage fcontext -a -t public_content_t "/shares(/.*)?"
```

5. Run **restorecon -R -v /shares/** as the root user to apply the label changes:

```
# restorecon -R -v /shares/
restorecon reset /shares context unconfined_u:object_r:default_t:s0-
>system_u:object_r:public_content_t:s0
restorecon reset /shares/index.html context unconfined_u:object_r:default_t:s0-
>system_u:object_r:public_content_t:s0
```

To share **/shares/** through Samba:

1. Run **rpm -q samba samba-common samba-client** to confirm the *samba*, *samba-common*, and *samba-client* packages are installed (version numbers may differ):

```
$ rpm -q samba samba-common samba-client
samba-3.4.0-0.41.el6.3.i686
samba-common-3.4.0-0.41.el6.3.i686
samba-client-3.4.0-0.41.el6.3.i686
```

If any of these packages are not installed, install them by running **yum install *package-name*** as the root user.

2. Edit **/etc/samba/smb.conf** as the root user. Add the following entry to the bottom of this file to share the **/shares/** directory through Samba:

```
[shares]
comment = Documents for Apache HTTP Server, FTP, rsync, and Samba
path = /shares
public = yes
writeable = no
```

3. A Samba account is required to mount a Samba file system. Run **smbpasswd -a *username*** as the root user to create a Samba account, where *username* is an existing Linux user. For example, **smbpasswd -a testuser** creates a Samba account for the Linux testuser user:

```
# smbpasswd -a testuser
New SMB password: Enter a password
Retype new SMB password: Enter the same password again
Added user testuser.
```

Running **smbpasswd -a *username***, where *username* is the username of a Linux account that does not exist on the system, causes a **Cannot locate Unix account for '*username*'!** error.

4. Run **service smb start** as the root user to start the Samba service:

```
service smb start
Starting SMB services:                                     [  OK  ]
```

5. Run **smbclient -U *username* -L localhost** to list the available shares, where *username* is the Samba account added in step 3. When prompted for a password, enter the password assigned to the Samba account in step 3 (version numbers may differ):

```
$ smbclient -U username -L localhost
Enter username's password:
Domain=[HOSTNAME] OS=[Unix] Server=[Samba 3.4.0-0.41.el6]

Sharename       Type      Comment
---------       ----      -------
shares          Disk      Documents for Apache HTTP Server, FTP, rsync, and Samba
IPC$            IPC       IPC Service (Samba Server Version 3.4.0-0.41.el6)
username        Disk      Home Directories
Domain=[HOSTNAME] OS=[Unix] Server=[Samba 3.4.0-0.41.el6]

Server               Comment
---------            -------
```

```
Workgroup            Master
---------            -------
```

6. Run **mkdir /test/** as the root user to create a new directory. This directory will be used to mount the **shares** Samba share.

7. Run the following command as the root user to mount the **shares** Samba share to **/test/**, replacing *username* with the username from step 3:

```
mount //localhost/shares /test/ -o user=username
```

   Enter the password for *username*, which was configured in step 3.

8. Run **cat /test/index.html** to view the file, which is being shared through Samba:

```
$ cat /test/index.html
<html>
<body>
<p>Hello</p>
</body>
</html>
```

To share **/shares/** through the Apache HTTP Server:

1. Run **rpm -q httpd** to confirm the *httpd* package is installed (version number may differ):

```
$ rpm -q httpd
httpd-2.2.11-6.i386
```

   If this package is not installed, run **yum install httpd** as the root user to install it.

2. Change into the **/var/www/html/** directory. Run the following command as the root user to create a link (named **shares**) to the **/shares/** directory:

```
ln -s /shares/ shares
```

3. Run **service httpd start** as the root user to start the Apache HTTP Server:

```
service httpd start
Starting httpd:                                            [  OK  ]
```

4. Use a web browser to navigate to **http://localhost/shares**. The **/shares/index.html** file is displayed.

By default, the Apache HTTP Server reads an **index.html** file if it exists. If **/shares/** did not have **index.html**, and instead had **file1**, **file2**, and **file3**, a directory listing would occur when accessing **http://localhost/shares**:

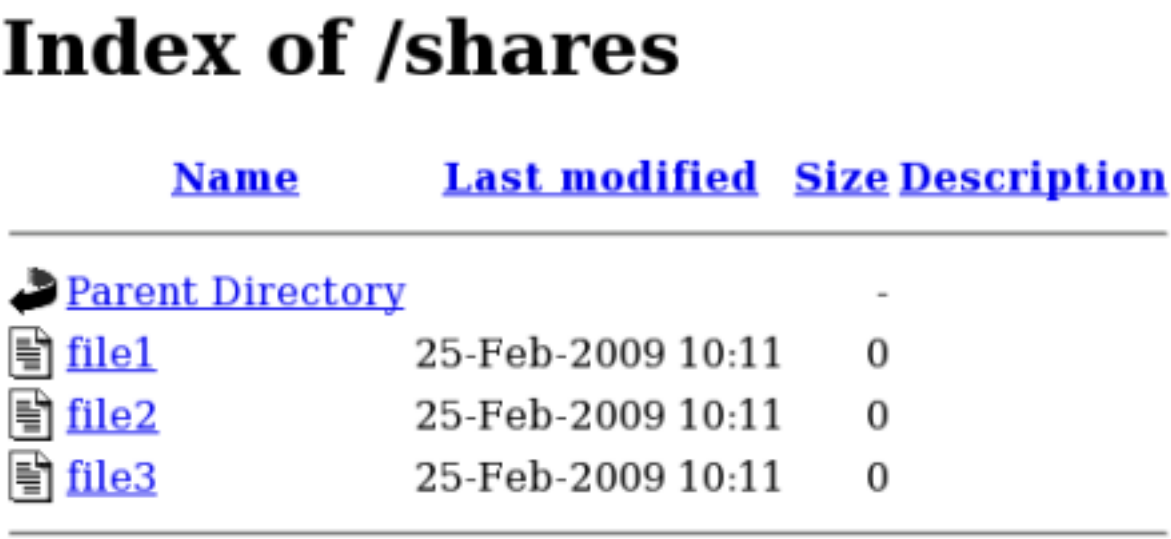
1. Run **rm -i /shares/index.html** as the root user to remove the **index.html** file.

2. Run **`touch /shares/file{1,2,3}`** as the root user to create three files in **`/shares/`**:

```
# touch /shares/file{1,2,3}
# ls -Z /shares/
-rw-r--r--  root root system_u:object_r:public_content_t:s0 file1
-rw-r--r--  root root unconfined_u:object_r:public_content_t:s0 file2
-rw-r--r--  root root unconfined_u:object_r:public_content_t:s0 file3
```

3. Run **`service httpd status`** as the root user to see the status of the Apache HTTP Server. If the server is stopped, run **`service httpd start`** as the root user to start it.

4. Use a web browser to navigate to **`http://localhost/shares`**. A directory listing is displayed:



3.4.4. Changing port numbers

Depending on policy configuration, services may only be allowed to run on certain port numbers. Attempting to change the port a service runs on without changing policy may result in the service failing to start. Run **`semanage port -l | grep -w "http_port_t"`** as the root user to list the ports SELinux allows `httpd` to listen on:

```
# semanage port -l | grep -w http_port_t
http_port_t                    tcp      80, 443, 488, 8008, 8009, 8443
```

By default, SELinux allows `http` to listen on TCP ports 80, 443, 488, 8008, 8009, or 8443. If **`/etc/httpd/conf/httpd.conf`** is configured so that `httpd` listens on any port not listed for **`http_port_t`**, `httpd` fails to start.

To configure `httpd` to run on a port other than TCP ports 80, 443, 488, 8008, 8009, or 8443:

1. Edit **`/etc/httpd/conf/httpd.conf`** as the root user so the **`Listen`** option lists a port that is not configured in SELinux policy for `httpd`. The following example configures `httpd` to listen on the 10.0.0.1 IP address, and on TCP port 12345:

```
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses (0.0.0.0)
#
```

```
#Listen 12.34.56.78:80
Listen 10.0.0.1:12345
```

2. Run **semanage port -a -t http_port_t -p tcp 12345** as the root user to add the port to SELinux policy configuration.

3. Run **semanage port -l | grep -w http_port_t** as the root user to confirm the port is added:

```
# semanage port -l | grep -w http_port_t
http_port_t                    tcp      12345, 80, 443, 488, 8008, 8009, 8443
```

If you no longer run `httpd` on port 12345, run **semanage port -d -t http_port_t -p tcp 12345** as the root user to remove the port from policy configuration.

Samba

From the *Samba*[1] website:

"Samba is an *Open Source*[2]/*Free Software*[3] suite that has, *since 1992*[4], provided file and print services to all manner of SMB/CIFS clients, including the numerous versions of Microsoft Windows operating systems. Samba is freely available under the *GNU General Public License*[5]."[6]

In Red Hat Enterprise Linux, the *samba* package provides the Samba server. Run **rpm -q samba** to see if the *samba* package is installed. If it is not installed and you want to use Samba, run the following command as the root user to install it:

```
yum install samba
```

4.1. Samba and SELinux

When SELinux is enabled, the Samba server (smbd) runs confined by default. Confined services run in their own domains, and are separated from other confined services. The following example demonstrates the smbd process running in its own domain. This example assumes the *samba* package is installed:

1. Run **getenforce** to confirm SELinux is running in enforcing mode:

   ```
   $ getenforce
   Enforcing
   ```

   The **getenforce** command returns **Enforcing** when SELinux is running in enforcing mode.

2. Run **service smbd start** as the root user to start smbd:

   ```
   service smb start
   Starting SMB services:                                     [  OK  ]
   ```

3. Run **ps -eZ | grep smb** to view the smbd processes:

   ```
   $ ps -eZ | grep smb
   unconfined_u:system_r:smbd_t:s0 16420 ?        00:00:00 smbd
   unconfined_u:system_r:smbd_t:s0 16422 ?        00:00:00 smbd
   ```

   The SELinux context associated with the smbd processes is **unconfined_u:system_r:smbd_t:s0**. The second last part of the context, **smbd_t**, is the type. A type defines a domain for processes and a type for files. In this case, the smbd processes are running in the smbd_t domain.

[1] http://samba.org/
[2] http://www.opensource.org/
[3] http://www.gnu.org/philosophy/free-sw.html
[4] http://us1.samba.org/samba/docs/10years.html
[5] http://us1.samba.org/samba/docs/GPL.html
[6] From the opening paragraph on the Samba website: *http://samba.org*. Accessed 20 January 2009.

Files must be labeled correctly to allow smbd to access and share them. For example, smbd can read and write to files labeled with the **samba_share_t** type, but by default, can not access files labeled with the **httpd_sys_content_t** type, which is intended for use by the Apache HTTP Server. Booleans must be turned on to allow certain behavior, such as allowing home directories and NFS file systems to be exported through Samba, as well as to allow Samba to act as a domain controller.

4.2. Types

Label files with the **samba_share_t** type to allow Samba to share them. Only label files you have created, and do not relabel system files with the **samba_share_t** type: Booleans can be turned on to share such files and directories. SELinux allows Samba to write to files labeled with the **samba_share_t** type, as long as **/etc/samba/smb.conf** and Linux permissions are set accordingly.

The **samba_etc_t** type is used on certain files in **/etc/samba/**, such as **smb.conf**. Do not manually label files with the **samba_etc_t** type. If files in **/etc/samba/** are not labeled correctly, run **restorecon -R -v /etc/samba** as the root user to restore such files to their default contexts. If **/etc/samba/smb.conf** is not labeled with the **samba_etc_t** type, the **service smb start** command may fail and an SELinux denial may be logged. The following is an example denial when **/etc/samba/smb.conf** was labeled with the **httpd_sys_content_t** type:

```
setroubleshoot: SELinux is preventing smbd (smbd_t) "read" to ./smb.conf
 (httpd_sys_content_t). For complete SELinux messages. run sealert -l deb33473-1069-482b-
bb50-e4cd05ab18af
```

4.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you must tell SELinux how you are running services. The following Booleans allow you to tell SELinux how you are running Samba:

allow_smbd_anon_write
: Having this Boolean enabled allows smbd to write to a public directory, such as an area reserved for common files that otherwise has no special access restrictions.

samba_create_home_dirs
: Having this Boolean enabled allows Samba to create new home directories independently. This is often done by mechanisms such as PAM.

samba_domain_controller
: When enabled, this Boolean allows Samba to act as a domain controller, as well as giving it permission to execute related commands such as **useradd**, **groupadd** and **passwd**.

samba_enable_home_dirs
: Enabling this Boolean allows Samba to share users' home directories.

samba_export_all_ro
: Export any file or directory, allowing read-only permissions. This allows files and directories that are not labeled with the **samba_share_t** type to be shared through Samba. When the **samba_export_all_ro** Boolean is on, but the **samba_export_all_rw** Boolean is off, write access to Samba shares is denied, even if write access is configured in **/etc/samba/smb.conf**, as well as Linux permissions allowing write access.

samba_export_all_rw
Export any file or directory, allowing read and write permissions. This allows files and directories that are not labeled with the **samba_share_t** type to be exported through Samba. Permissions in **/etc/samba/smb.conf** and Linux permissions must be configured to allow write access.

samba_run_unconfined
Having this Boolean enabled allows Samba to run unconfined scripts in the /var/lib/samba/scripts directory.

samba_share_fusefs
This Boolean must be enabled for Samba to share fusefs file systems.

samba_share_nfs
Disabling this Boolean prevents `smbd` from having full access to NFS shares via Samba. Enabling this Boolean will allow Samba to share NFS file systems.

use_samba_home_dirs
Enable this Boolean to use a remote server for Samba home directories.

virt_use_samba
Allow virtual machine access to CIFS files.

4.4. Configuration examples

The following examples provide real-world demonstrations of how SELinux complements the Samba server and how full function of the Samba server can be maintained.

4.4.1. Sharing directories you create

The following example creates a new directory, and shares that directory through Samba:

1. Run **rpm -q samba samba-common samba-client** to confirm the *samba*, *samba-common*, and *samba-client* packages are installed. If any of these packages are not installed, install them by running **yum install *package-name*** as the root user.

2. Run **mkdir /myshare** as the root user to create a new top-level directory to share files through Samba.

3. Run **touch /myshare/file1** as the root user to create an empty file. This file is used later to verify the Samba share mounted correctly.

4. SELinux allows Samba to read and write to files labeled with the **samba_share_t** type, as long as **/etc/samba/smb.conf** and Linux permissions are set accordingly. Run the following command as the root user to add the label change to file-context configuration:

```
semanage fcontext -a -t samba_share_t "/myshare(/.*)?"
```

5. Run **restorecon -R -v /myshare** as the root user to apply the label changes:

```
# restorecon -R -v /myshare
restorecon reset /myshare context unconfined_u:object_r:default_t:s0-
>system_u:object_r:samba_share_t:s0
restorecon reset /myshare/file1 context unconfined_u:object_r:default_t:s0-
>system_u:object_r:samba_share_t:s0
```

6. Edit **/etc/samba/smb.conf** as the root user. Add the following to the bottom of this file to share the /myshare/ directory through Samba:

```
[myshare]
comment = My share
path = /myshare
public = yes
writeable = no
```

7. A Samba account is required to mount a Samba file system. Run **smbpasswd -a *username*** as the root user to create a Samba account, where *username* is an existing Linux user. For example, **smbpasswd -a testuser** creates a Samba account for the Linux testuser user:

```
# smbpasswd -a testuser
New SMB password: Enter a password
Retype new SMB password: Enter the same password again
Added user testuser.
```

   Running **smbpasswd -a *username***, where *username* is the username of a Linux account that does not exist on the system, causes a **Cannot locate Unix account for '*username*'!** error.

8. Run **service smb start** as the root user to start the Samba service:

```
service smb start
Starting SMB services:                                     [  OK  ]
```

9. Run **smbclient -U *username* -L localhost** to list the available shares, where *username* is the Samba account added in step 7. When prompted for a password, enter the password assigned to the Samba account in step 7 (version numbers may differ):

```
$ smbclient -U username -L localhost
Enter username's password:
Domain=[HOSTNAME] OS=[Unix] Server=[Samba 3.4.0-0.41.el6]

Sharename       Type      Comment
---------       ----      -------
myshare         Disk      My share
IPC$            IPC       IPC Service (Samba Server Version 3.4.0-0.41.el6)
username        Disk      Home Directories
Domain=[HOSTNAME] OS=[Unix] Server=[Samba 3.4.0-0.41.el6]

Server               Comment
---------            -------

Workgroup            Master
---------            -------
```

10. Run **mkdir /test/** as the root user to create a new directory. This directory will be used to mount the **myshare** Samba share.

11. Run the following command as the root user to mount the **myshare** Samba share to **/test/**, replacing *username* with the username from step 7:

```
mount //localhost/myshare /test/ -o user=username
```

Enter the password for *username*, which was configured in step 7.

12. Run **ls /test/** to view the **file1** file created in step 3:

```
$ ls /test/
file1
```

4.4.2. Sharing a website

It may not be possible to label files with the **samba_share_t** type, for example, when wanting to share a website in **/var/www/html/**. For these cases, use the **samba_export_all_ro** Boolean to share any file or directory (regardless of the current label), allowing read only permissions, or the **samba_export_all_rw** Boolean to share any file or directory (regardless of the current label), allowing read and write permissions.

The following example creates a file for a website in **/var/www/html/**, and then shares that file through Samba, allowing read and write permissions. This example assumes the *httpd*, *samba*, *samba-common*, *samba-client*, and *wget* packages are installed:

1. As the root user, create a **/var/www/html/file1.html** file. Copy and paste the following content into **/var/www/html/file1.html**:

```
<html>
<h2>File being shared through the Apache HTTP Server and Samba.</h2>
</html>
```

2. Run **ls -Z /var/www/html/file1.html** to view the SELinux context of **file1.html**:

```
$ ls -Z /var/www/html/file1.html
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 /var/www/html/
file1.html
```

file1.index.html is labeled with the **httpd_sys_content_t**. By default, the Apache HTTP Server can access this type, but Samba can not.

3. Run **service httpd start** as the root user to start the Apache HTTP Server:

```
service httpd start
Starting httpd:                                            [  OK  ]
```

4. Change into a directory your user has write access to, and run the **wget http://localhost/file1.html** command. Unless there are changes to the default configuration, this command succeeds:

```
$ wget http://localhost/file1.html
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
```

```
HTTP request sent, awaiting response... 200 OK
Length: 84 [text/html]
Saving to: `file1.html.1'

100%[======================>] 84          --.-K/s   in 0s

`file1.html.1' saved [84/84]
```

5. Edit **/etc/samba/smb.conf** as the root user. Add the following to the bottom of this file to share the **/var/www/html/** directory through Samba:

```
[website]
comment = Sharing a website
path = /var/www/html/
public = no
writeable = no
```

6. The **/var/www/html/** directory is labeled with the **httpd_sys_content_t** type. By default, Samba can not access files and directories labeled with the **httpd_sys_content_t** type, even if Linux permissions allow it. To allow Samba access, run the following command as the root user to turn the **samba_export_all_ro** Boolean on:

```
setsebool -P samba_export_all_ro on
```

   Do not use the **-P** option if you do not want the change to persist across reboots. Note: turning the **samba_export_all_ro** Boolean on allows Samba to access any type.

7. Run **service smb start** as the root user to start smbd:

```
service smb start
Starting SMB services:                                     [  OK  ]
```

File Transfer Protocol

File Transfer Protocol (FTP) is one of the oldest and most commonly used protocols found on the Internet today. Its purpose is to reliably transfer files between computer hosts on a network without requiring the user to log directly into the remote host or have knowledge of how to use the remote system. It allows users to access files on remote systems using a standard set of simple commands.

The Very Secure FTP Daemon (`vsftpd`) is designed from the ground up to be fast, stable, and, most importantly, secure. Its ability to handle large numbers of connections efficiently and securely is why `vsftpd` is the only stand-alone FTP distributed with Red Hat Enterprise Linux.

In Red Hat Enterprise Linux, the *vsftpd* package provides the Very Secure FTP daemon. Run **`rpm -q vsftpd`** to see if *vsftpd* is installed:

```
$ rpm -q vsftpd
```

If you want an FTP server and the *vsftpd* package is not installed, run the following command as the root user to install it:

```
yum install vsftpd
```

5.1. FTP and SELinux

The `vsftpd` FTP daemon runs confined by default. SELinux policy defines how `vsftpd` interacts with files, processes, and with the system in general. For example, when an authenticated user logs in via FTP, they can not read from or write to files in their home directories: SELinux prevents `vsftpd` from accessing user home directories by default. Also, by default, `vsftpd` does not have access to NFS or CIFS file systems, and anonymous users do not have write access, even if such write access is configured in **`/etc/vsftpd/vsftpd.conf`**. Booleans can be turned on to allow the previously mentioned access.

The following example demonstrates an authenticated user logging in, and an SELinux denial when trying to view files in their home directory:

1. Run **`rpm -q ftp`** to see if the *ftp* package is installed. If it is not, run **`yum install ftp`** as the root user to install it.

2. Run **`rpm -q vsftpd`** to see if the *vsftpd* package is installed. If it is not, run **`yum install vsftpd`** as the root user to install it.

3. In Red Hat Enterprise Linux, `vsftpd` only allows anonymous users to log in by default. To allow authenticated users to log in, edit **`/etc/vsftpd/vsftpd.conf`** as the root user. Make sure the **`local_enable=YES`** option is uncommented:

   ```
   # Uncomment this to allow local users to log in.
   local_enable=YES
   ```

4. Run **`service vsftpd start`** as the root user to start `vsftpd`. If the service was running before editing **`vsftpd.conf`**, run **`service vsftpd restart`** as the root user to apply the configuration changes:

```
service vsftpd start
Starting vsftpd for vsftpd:                                [  OK  ]
```

5. Run **ftp localhost** as the user you are currently logged in with. When prompted for your name, make sure your username is displayed. If the correct username is displayed, press **Enter**, otherwise, enter the correct username:

```
$ ftp localhost
Connected to localhost (127.0.0.1).
220 (vsFTPd 2.1.0)
Name (localhost:username):
331 Please specify the password.
Password: Enter your password
500 OOPS: cannot change directory:/home/username
Login failed.
ftp>
```

6. An SELinux denial similar to the following is logged:

```
setroubleshoot: SELinux is preventing the ftp daemon from reading users home directories
 (username). For complete SELinux messages. run sealert -l c366e889-2553-4c16-
b73f-92f36a1730ce
```

7. Access to home directories has been denied by SELinux. This can be fixed by activating the **ftp_home_dir** Boolean. Enable this **ftp_home_dir** Boolean by running the following command as the root user:

```
# setsebool -P ftp_home_dir=1
```

Note

Do not use the -P option if you do not want changes to persist across reboots.

Try to log in again. Now that SELinux is allowing access to home directories via the **ftp_home_dir** Boolean, logging in will succeed.

5.2. Types

By default, anonymous users have read access to files in **/var/ftp/** when they log in via FTP. This directory is labeled with the **public_content_t** type, allowing only read access, even if write access is configured in **/etc/vsftpd/vsftpd.conf**. The **public_content_t** type is accessible to other services, such as Apache HTTP Server, Samba, and NFS.

Use one of the following types to share files through FTP:

public_content_t
Label files and directories you have created with the **public_content_t** type to share them read-only through vsftpd. Other services, such as Apache HTTP Server, Samba, and NFS, also have access to files labeled with this type. Files labeled with the **public_content_t** type can not be written to, even if Linux permissions allow write access. If you require write access, use the **public_content_rw_t** type.

public_content_rw_t
Label files and directories you have created with the **public_content_rw_t** type to share them with read and write permissions through vsftpd. Other services, such as Apache HTTP Server, Samba, and NFS, also have access to files labeled with this type. Remember that Booleans for each service must be turned on before they can write to files labeled with this type.

5.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you must tell SELinux how you are running services. The following Booleans allow you to tell SELinux how you are running vsftpd:

allow_ftpd_anon_write
When disabled, this Boolean prevents vsftpd from writing to files and directories labeled with the **public_content_rw_t** type. Turn this Boolean on to allow users to upload files via FTP. The directory where files are uploaded to must be labeled with the **public_content_rw_t** type and Linux permissions set accordingly.

allow_ftpd_full_access
When this Boolean is on, only Linux (DAC) permissions are used to control access, and authenticated users can read and write to files that are not labeled with the **public_content_t** or **public_content_rw_t** types.

allow_ftpd_use_cifs
Having this Boolean enabled allows vsftpd to access files and directories labeled with the **cifs_t** type; therefore, having this Boolean enabled allows you to share file systems mounted via Samba through vsftpd.

allow_ftpd_use_nfs
Having this Boolean enabled allows vsftpd to access files and directories labeled with the **nfs_t** type; therefore, having this Boolean enabled allows you to share file systems mounted via NFS through vsftpd.

ftp_home_dir
Having this Boolean enabled allows authenticated users to read and write to files in their home directories. When this Boolean is off, attempting to download a file from a home directory results in an error such as **550 Failed to open file**. An SELinux denial is logged.

ftpd_connect_db
Allow FTP daemons to initiate a connection to a database.

httpd_enable_ftp_server
Allow httpd to listen on the FTP port and act as a FTP server.

tftp_anon_write
Having this Boolean enabled allows TFTP access to a public directory, such as an area reserved for common files that otherwise has no special access restrictions.

5.4. Configuration Examples

5.4.1. Uploading to an FTP site

The following example creates an FTP site that allows a dedicated user to upload files. It creates the directory structure and the required SELinux configuration changes:

1. Run **setsebool ftp_home_dir=1** as the root user to enable access to FTP home directories.

2. Run **mkdir -p /myftp/pub** as the root user to create a new top-level directory.

3. Set Linux permissions on the **/myftp/pub/** directory to allow a Linux user write access. This example changes the owner and group from root to owner user1 and group root. Replace user1 with the user you want to give write access to:

```
# chown user1:root /myftp/pub
# chmod 775 /myftp/pub
```

   The **chown** command changes the owner and group permissions. The **chmod** command changes the mode, allowing the user1 user read, write, and execute permissions, and members of the root group read, write, and execute permissions. Everyone else has read and execute permissions: this is required to allow the Apache HTTP Server to read files from this directory.

4. When running SELinux, files and directories must be labeled correctly to allow access. Setting Linux permissions is not enough. Files labeled with the **public_content_t** type allow them to be read by FTP, Apache HTTP Server, Samba, and rsync. Files labeled with the **public_content_rw_t** type can be written to by FTP. Other services, such as Samba, require Booleans to be set before they can write to files labeled with the **public_content_rw_t** type. Label the top-level directory (**/myftp/**) with the **public_content_t** type, to prevent copied or newly-created files under **/myftp/** from being written to or modified by services. Run the following command as the root user to add the label change to file-context configuration:

```
semanage fcontext -a -t public_content_t /myftp
```

5. Run **restorecon -R -v /myftp/** to apply the label change:

```
# restorecon -R -v /myftp/
restorecon reset /myftp context unconfined_u:object_r:default_t:s0-
>system_u:object_r:public_content_t:s0
```

6. Confirm **/myftp** is labeled with the **public_content_t** type, and **/myftp/pub/** is labeled with the **default_t** type:

```
$ ls -dZ /myftp/
drwxr-xr-x. root root system_u:object_r:public_content_t:s0 /myftp/
$ ls -dZ /myftp/pub/
drwxrwxr-x. user1 root unconfined_u:object_r:default_t:s0 /myftp/pub/
```

7. FTP must be allowed to write to a directory before users can upload files via FTP. SELinux allows FTP to write to directories labeled with the **public_content_rw_t** type. This example uses **/myftp/pub/** as the directory FTP can write to. Run the following command as the root user to add the label change to file-context configuration:

```
semanage fcontext -a -t public_content_rw_t "/myftp/pub(/.*)?"
```

8. Run **restorecon -R -v /myftp/pub** as the root user to apply the label change:

```
# restorecon -R -v /myftp/pub
restorecon reset /myftp/pub context system_u:object_r:default_t:s0-
>system_u:object_r:public_content_rw_t:s0
```

9. The **allow_ftpd_anon_write** Boolean must be on to allow **vsftpd** to write to files that are labeled with the **public_content_rw_t** type. Run the following command as the root user to turn this Boolean on:

```
setsebool -P allow_ftpd_anon_write on
```

   Do not use the **-P** option if you do not want changes to persist across reboots.

The following example demonstrates logging in via FTP and uploading a file. This example uses the user1 user from the previous example, where user1 is the dedicated owner of the **/myftp/pub/** directory:

1. Run **cd ~/** to change into your home directory. Then, run **mkdir myftp** to create a directory to store files to upload via FTP.

2. Run **cd ~/myftp** to change into the **~/myftp/** directory. In this directory, create an **ftpupload** file. Copy the following contents into this file:

```
File upload via FTP from a home directory.
```

3. Run **getsebool allow_ftpd_anon_write** to confirm the **allow_ftpd_anon_write** Boolean is on:

```
$ getsebool allow_ftpd_anon_write
allow_ftpd_anon_write --> on
```

   If this Boolean is off, run **setsebool -P allow_ftpd_anon_write on** as the root user to turn it on. Do not use the **-P** option if you do not want the change to persist across reboots.

4. Run **service vsftpd start** as the root user to start `vsftpd`:

```
# service vsftpd start
Starting vsftpd for vsftpd:                                [  OK  ]
```

5. Run **ftp localhost**. When prompted for a username, enter the the username of the user who has write access, then, enter the correct password for that user:

```
$ ftp localhost
Connected to localhost (127.0.0.1).
220 (vsFTPd 2.1.0)
Name (localhost:username):
331 Please specify the password.
Password: Enter the correct password
230 Login successful.
Remote system type is UNIX.
```

```
Using binary mode to transfer files.
ftp> cd myftp
250 Directory successfully changed.
ftp> put ftpupload
local: ftpupload remote: ftpupload
227 Entering Passive Mode (127,0,0,1,241,41).
150 Ok to send data.
226 File receive OK.
ftp> 221 Goodbye.
```

The upload succeeds as the **allow_ftpd_anon_write** Boolean is enabled.

Network File System

From the *Red Hat Linux Reference Guide*[1]:

NFS (Network File System) allows hosts to mount partitions on a remote system and use them as though they are local file systems. This allows the system administrator to store resources in a central location on the network, providing authorized users continuous access to them.

In Red Hat Enterprise Linux, the *nfs-utils* package is required for full NFS support. Run **`rpm -q nfs-utils`** to see if the *nfs-utils* is installed. If it is not installed and you want to use NFS, run the following command as the root user to install it:

```
yum install nfs-utils
```

6.1. NFS and SELinux

When running SELinux, the NFS daemons are confined by default. SELinux policy does not allow NFS to share files by default. If you want to share NFS partitions, this can be configured via the **`nfs_export_all_ro`** and **`nfs_export_all_rw`** Booleans, as described in this section. These Booleans are however not required when files to be shared are labeled with the **`public_content_t`** or **`public_content_rw_t`** types. NFS can share files labeled with these types even if the **`nfs_export_all_ro`** and **`nfs_export_all_rw`** Booleans are off.

6.2. Types

By default, mounted NFS file systems on the client side are labeled with a default context defined by policy for NFS file systems. In common policies, this default context uses the **`nfs_t`** type.The following types are used with NFS. Different types allow you to configure flexible access:

`var_lib_nfs_t`
: This type is used for existing and new files copied to or created in the **`/var/lib/nfs`** directory. This type should not need to be changed in normal operation. To restore changes to the default settings, run the **`restorecon -R -v /var/lib/nfs`** command as the root user.

`nfsd_exec_t`
: The **`/usr/sbin/rpc.nfsd`** file is labeled with the **`nfsd_exec_t`**, as are other system executables and libraries related to NFS. Users should not label any files with this type. **`nfsd_exec_t`** will transition to **`nfs_t`**.

6.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you must tell SELinux how you are running services. The following Booleans allow you to tell SELinux how you are running NFS:

`allow_ftpd_use_nfs`
: When enabled, this Boolean allows `ftpd` access to NFS mounts.

[1] http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/ref-guide/ch-nfs.html

allow_nfsd_anon_write
When enabled, this Boolean allows `nfsd` to write to a public directory anonymously; such as to an area reserved for common files that otherwise has no special access restrictions.

httpd_use_nfs
When enabled, this Boolean will allow `httpd` to access files stored on a NFS filesystem.

nfs_export_all_ro
Export any file or directory via NFS, allowing read-only permissions.

nfs_export_all_rw
Export any file or directory via NFS, allowing read and write permissions.

qemu_use_nfs
Allow qemu to use NFS file systems.

samba_share_nfs
When disabled, this Boolean prevents `smbd` from having full access to NFS shares via Samba. Enabling this Boolean will allow Samba to share NFS file systems.

use_nfs_home_dirs
Having this Boolean enabled adds support for NFS home directories.

virt_use_nfs
Allow virtual machine access to NFS files.

xen_use_nfs
Allow Xen to use NFS files.

6.4. Configuration Examples

6.4.1. Sharing directories using NFS

The example in this section creates a directory and shares it using NFS and SELinux. Two hosts are used in this example; a NFS server with a hostname of **nfs-srv** with an IP address of **192.168.1.1**, and a client with a hostname of **nfs-client** and an IP address of **192.168.1.100**. Both hosts are on the same subnet (192.168.1.0/24). This is an example only and assumes that the *nfs-utils* package is installed, that the SELinux targeted policy is used, and that SELinux is running in enforced mode.

This example will show that while even with full network availability and Linux file permissions granting access to all users via NFS, SELinux is still able to block mounting of NFS file systems unless the proper permissions are given via SELinux Booleans.

6.4.1.1. Server setup

Steps 1-10 in the following procedure should be performed on the NFS server, **nfs-srv**.

1. Run the **setsebool** command to disable read/write mounting of NFS file systems:

```
setsebool -P nfs_export_all_rw off
```

Note

Do not use the **-P** option if you do not want **setsebool** changes to persist across reboots.

2. Run **rpm -q nfs-utils** to confirm the *nfs-utils* package is installed. The *nfs-utils* package provides support programs for using NFS and should be installed on a NFS server and on any clients in use. If this package is not installed, install it by running **yum install *nfs-utils*** as the root user.
3. Run **mkdir /myshare** as the root user to create a new top-level directory to share using NFS.
4. Run **touch /myshare/file1** as the root user to create a new empty file in the shared area. This file will be accessed later by the client.
5. To show that SELinux is still able to block access even when Linux permissions are completely open, give the **/myshare** directory full Linux access rights for all users:

```
# chmod -R 777 /myshare
```

Warning

This is an example only and these permissions should not be used in a production system.

6. Edit the **/etc/exports** file and add the following line to the top of the file:

```
/myshare  192.168.1.100(rw)
```

This entry shows the full path on the server to the shared folder **/myshare**, the host or network range that **nfs-srv** will share to (in this case the IP address of a single host, **nfs-client** at **192.168.1.100**), and finally the share permissions. Read and write permissions are given here, as indicated by **(rw)**.

7. The TCP and UDP ports used for NFS are assigned dynamically by rpcbind, which can cause problems when creating firewall rules. To simplify the process of allowing NFS traffic through the firewall in this example, edit the */etc/sysconfig/nfs* file and uncomment the **MOUNTD_PORT**,**STATD_PORT**,**LOCKD_TCPPORT** and **LOCKD_UDPPORT** variables. Changing the port numbers in this file is not required for this example.

   Ensure that incoming connections are allowed through the server's firewall. This can be done via the *system-config-firewall* tool:

   - TCP and UDP port 2049 for NFS.
   - TCP and UDP port 111 (rpcbind/sunrpc).
   - The TCP and UDP port specified by the **MOUNTD_PORT="port"** option.
   - The TCP and UDP port specified by the **STATD_PORT="port"** option.

- The TCP port specified by the **LOCKD_TCPPORT="port"** option.
- The UDP port specified by the **LOCKD_UDPPORT="port"** option.

8. Run **service nfs start** as the root user to start NFS and its related services:

```
# service nfs start
Starting NFS services:  [  OK  ]
Starting NFS quotas:  [  OK  ]
Starting NFS daemon:  [  OK  ]
Starting NFS mountd:  [  OK  ]
```

9. To ensure that the NFS subsystem export table is updated, run **exportfs -rv** as the root user:

```
# exportfs -rv
exporting 192.168.1.100:/myshare
```

10. Run **showmount -e** as the root user to show all exported file systems:

```
# showmount -e
Export list for nfs-srv:
/myshare 192.168.1.100
```

At this point the server **nfs-srv** has been configured to allow NFS communications to **nfs-client** at **192.168.1.100**, and full Linux file systems permissions are active. If SELinux were disabled, the client would be able to mount this share and have full access over it. However, as the **nfs_export_all_rw** Boolean is disabled, the client is currently not able to mount this file system, as shown in the following output. This step should be performed on the client, **nfs-client**:

```
[nfs-client]# mkdir /myshare
[nfs-client]# mount.nfs 192.168.1.1:/myshare /myshare
mount.nfs: access denied by server while mounting 192.168.1.1:/myshare/
```

Enable the SELinux Boolean that was disabled in Step 1 above, and the client will be able to successfully mount the shared file system. This step should be performed on the NFS server, **nfs-srv**:

```
[nfs-srv]# setsebool -P nfs_export_all_rw on
```

Restart the NFS daemon:

```
[nfs-srv]# service nfs restart
```

Now try to mount the NFS file system again. This step should be performed on the NFS client, **nfs-client**:

```
[nfs-client]# mount.nfs 192.168.1.1:/myshare /myshare
[nfs-client]#
```

```
[nfs-client]# ls /myshare
total 0
-rwxrwxrwx.  1 root root 0 2009-04-16 12:07 file1
[nfs-client]#
```

The file system has been mounted successfully by the client. This example demonstrates how SELinux adds another layer of protection and can still enforce SELinux permissions even when Linux permissions are set to give full rights to all users.

Berkeley Internet Name Domain

BIND performs name resolution services via the named daemon. BIND lets users locate computer resources and services by name instead of numerical addresses.

In Red Hat Enterprise Linux, the *bind* package provides a DNS server. Run **rpm -q bind** to see if the *bind* package is installed. If it is not installed, run the following command as the root user to install it:

```
yum install bind
```

7.1. BIND and SELinux

The default permissions on the **/var/named/slaves**,**/var/named/dynamic** and **/var/named/data** directories allow zone files to be updated via zone transfers and dynamic DNS updates. Files in **/var/named** are labeled with the **named_zone_t** type, which is used for master zone files.

For a slave server, configure **/etc/named.conf** to place slave zones in **/var/named/slaves**. The following is an example of a domain entry in **/etc/named.conf** for a slave DNS server that stores the zone file for **testdomain.com** in **/var/named/slaves**:

```
zone "testdomain.com" {
   type slave;
   masters { IP-address; };
   file "/var/named/slaves/db.testdomain.com";
         };
```

If a zone file is labeled **named_zone_t**, the **named_write_master_zones** Boolean must be enabled to allow zone transfers and dynamic DNS to update the zone file. Also, the mode of the parent directory has to be changed to allow the named user or group read, write and execue access.

If zone files in **/var/named/** are labeled with **named_cache_t** type, a file system relabel or running **restorecon -R /var/** will change their type to **named_zone_t**.

7.2. Types

The following types are used with BIND. Different types allow you to configure flexible access:

named_zone_t
: Used for master zone files. Other services can not modify files of this type. named can only modify files of this type if the **named_write_master_zones** Boolean is turned on.

named_cache_t
: By default, named can write to files labeled with this type, without additional Booleans being set. Files copied or created in the **/var/named/slaves**,**/var/named/dynamic** and **/var/named/data** directories are automatically labeled with the **named_cache_t** type.

7.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you must tell SELinux how you are running services. The following Booleans allow you to tell SELinux how you are running NFS:

named_write_master_zones
When disabled, this Boolean prevents named from writing to zone files or directories labeled with the **named_zone_t** type. named does not usually need to write to zone files; but in the case that it needs to, or if a secondary server needs to write to zone files, enable this Boolean to allow this action.

7.4. Configuration Examples

7.4.1. Dynamic DNS

BIND allows hosts to update their records in DNS and zone files dynamically. This is used when a host computer's IP address changes frequently and the DNS record requires real-time modification.

Use the **/var/named/dynamic** directory for zone files you want updated via dynamic DNS. Files created in or copied into **/var/named/dynamic** inherit Linux permissions that allow named to write to them. As such files are labeled with the **named_cache_t** type, SELinux allows named to write to them.

If a zone file in **/var/named/dynamic** is labeled with the **named_zone_t** type, dynamic DNS updates may not be successful for a certain period of time as the update needs to be written to a journal first before being merged. If the zone file is labeled with the **named_zone_t** type when the journal attempts to be merged, an error such as the following is logged:

```
named[PID]: dumping master file: rename: /var/named/dynamic/zone-name: permission denied
```

Also, the following SELinux denial is logged:

```
setroubleshoot: SELinux is preventing named (named_t) "unlink" to zone-name (named_zone_t)
```

To resolve this labeling issue, run the **restorecon -R -v /var/named/dynamic** command as the Linux root user.

Concurrent Versioning System

The Concurrent Versioning System (CVS) is a free revision-control system. It is used to monitor and keep track of modifications to a central set of files which are usually accessed by several different users. It is commonly used by programmers to manage a source code repository and is widely used by open source programmers.

In Red Hat Enterprise Linux, the *cvs* package provides CVS. Run **`rpm -q cvs`** to see if the *cvs* package is installed. If it is not installed and you want to use CVS, run the following command as the root user to install it:

```
yum install cvs
```

8.1. CVS and SELinux

The `cvs` daemon runs as **`cvs_t`**. By default in Red Hat Enterprise Linux, CVS is only allowed to read and write certain directories. The label **`cvs_data_t`** defines which areas the `cvs` daemon has read and write access to. When using CVS with SELinux, assigning the correct label is essential for clients to have full access to the area reserved for CVS data.

8.2. Types

The following types are used with CVS. Different types allow you to configure flexible access:

`cvs_data_t`
: This type is used for data in a CVS repository. CVS can only gain full access to data if it has this type.

`cvs_exec_t`
: This type is used for the **`/usr/bin/cvs`** binary.

8.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you must tell SELinux how you are running services. The following Boolean allows you to tell SELinux how you are running CVS:

`allow_cvs_read_shadow`
: This Boolean allows the `cvs` daemon to access the **`/etc/shadow`** file for user authentication.

8.4. Configuration Examples

8.4.1. Setting up CVS

This example describes a simple CVS setup and an SELinux configuration which allows remote access. Two hosts are used in this example; a CVS server with a hostname of **`cvs-srv`** with an IP address of **`192.168.1.1`** and a client with a hostname of **`cvs-client`** and an IP address of **`192.168.1.100`**. Both hosts are on the same subnet (192.168.1.0/24). This is an example only and assumes that the *cvs* and *xinetd* packages are installed, that the SELinux targeted policy is used, and that SELinux is running in enforced mode.

This example will show that even with full DAC permissions, SELinux can still enforce policy rules based on file labels and only allow access to certain areas that have been specifically labeled for access by CVS.

Note

Steps 1-9 should be performed on the CVS server, **cvs-srv**.

1. This example requires the *cvs* and *xinetd* packages. Run **rpm -q cvs** to see if the *cvs* package is installed. If it is not installed, run the following command as the root user to install *cvs*:

```
# yum install cvs
```

   Run **rpm -q xinetd** to see if the *xinetd* package is installed. If it is not installed, run the following command as the root user to install *xinetd*:

```
# yum install xinetd
```

2. Create a group named **CVS**. This can be done via the **groupadd CVS** command as the root user, or by using the *system-config-users* tool.

3. Create a user with a username of **cvsuser** and make this user a member of the CVS group. This can be done using the *system-config-users* tool.

4. Edit the **/etc/services** file and make sure that the CVS server has uncommented entries looking similar to the following:

```
cvspserver 2401/tcp   # CVS client/server operations
cvspserver 2401/udp   # CVS client/server operations
```

5. Create the CVS repository in the root area of the file system. When using SELinux, it is best to have the repository in the root file system so that recursive labels can be given to it without affecting any other subdirectories. For example, as the root user, create a **/cvs** directory to house the repository:

```
[root@cvs-srv]# mkdir /cvs
```

6. Give full permissions to the **/cvs** directory to all users:

```
[root@cvs-srv]# chmod -R 777 /cvs
```

Warning

This is an example only and these permissions should not be used in a production system.

7. Edit the **/etc/xinetd.d/cvs** file and make sure that the CVS section is uncommented and configured to use the **/cvs** directory. The file should look similar to:

```
service cvspserver
{
 disable = no
 port   = 2401
 socket_type  = stream
 protocol  = tcp
 wait   = no
 user   = root
 passenv   = PATH
 server   = /usr/bin/cvs
 env   = HOME=/cvs
 server_args  = -f --allow-root=/cvs pserver
# bind   = 127.0.0.1
```

8. Start the `xinetd` daemon by running **service xinetd start** as the root user.

9. Add a rule which allows inbound connections using TCP on port 2401 by using the *system-config-firewall* tool.

10. As the **cvsuser** user, run the following command:

```
[cvsuser@cvs-client]$ cvs -d /cvs init
```

11. At this point, CVS has been configured but SELinux will still deny logins and file access. To demonstrate this, set the $CVSROOT variable on **cvs-client** and try to log in remotely. The following step should be performed on **cvs-client**:

```
[cvsuser@cvs-client]$ export CVSROOT=:pserver:cvsuser@192.168.1.1:/cvs
[cvsuser@cvs-client]$
[cvsuser@cvs-client]$ cvs login
Logging in to :pserver:cvsuser@192.168.1.1:2401/cvs
CVS password: ********
cvs [login aborted]: unrecognized auth response from 192.168.100.1: cvs pserver: cannot
 open /cvs/CVSROOT/config: Permission denied
```

SELinux has blocked access. In order to get SELinux to allow this access, the following step should be performed on **cvs-srv**:

12. Change the context of the **/cvs** directory as the root user in order to recursively label any existing and new data in the **/cvs** directory, giving it the **cvs_data_t** type:

```
[root@cvs-srv]# semanage fcontext -a -t cvs_data_t '/cvs(/.*)?'
[root@cvs-srv]# restorecon -R -v /cvs
```

13. The client, **cvs-client** should now be able to log in and access all CVS resources in this repository:

```
[cvsuser@cvs-client]$ export CVSROOT=:pserver:cvsuser@192.168.1.1:/cvs
[cvsuser@cvs-client]$
[cvsuser@cvs-client]$ cvs login
```

```
Logging in to :pserver:cvsuser@192.168.1.1:2401/cvs
CVS password: ********
[cvsuser@cvs-client]$
```

Squid Caching Proxy

From the *Squid Caching Proxy*[1] project page:

"Squid is a caching proxy for the Web supporting HTTP, HTTPS, FTP, and more. It reduces bandwidth and improves response times by caching and reusing frequently-requested web pages. Squid has extensive access controls and makes a great server accelerator."

In Red Hat Enterprise Linux, the *squid* package provides the Squid Caching Proxy. Run **rpm -q squid** to see if the *squid* package is installed. If it is not installed and you want to use squid, run the following command as the root user to install it:

```
# yum install squid
```

9.1. Squid Caching Proxy and SELinux

When SELinux is enabled, squid runs confined by default. Confined processes run in their own domains, and are separated from other confined processes. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited. The following example demonstrates the squid processes running in their own domain. This example assumes the squid package is installed:

1. Run **getenforce** to confirm SELinux is running in enforcing mode:

   ```
   $ getenforce
   Enforcing
   ```

   The **getenforce** command returns **Enforcing** when SELinux is running in enforcing mode.

2. Run **service squid start** as the root user to start `squid`:

   ```
   # service squid start
   Starting squid:                                            [  OK  ]
   ```

3. Run **ps -eZ | grep squid** to view the `squid` processes:

   ```
   $ ps -eZ | grep squid
   unconfined_u:system_r:squid_t:s0 2522 ?        00:00:00 squid
   unconfined_u:system_r:squid_t:s0 2524 ?        00:00:00 squid
   unconfined_u:system_r:squid_t:s0 2526 ?        00:00:00 ncsa_auth
   unconfined_u:system_r:squid_t:s0 2527 ?        00:00:00 ncsa_auth
   unconfined_u:system_r:squid_t:s0 2528 ?        00:00:00 ncsa_auth
   unconfined_u:system_r:squid_t:s0 2529 ?        00:00:00 ncsa_auth
   unconfined_u:system_r:squid_t:s0 2530 ?        00:00:00 ncsa_auth
   unconfined_u:system_r:squid_t:s0 2531 ?        00:00:00 unlinkd
   ```

   The SELinux context associated with the `squid` processes is **unconfined_u:system_r:squid_t:s0**. The second last part of the context, **squid_t**, is the

[1] http://www.squid-cache.org/

type. A type defines a domain for processes and a type for files. In this case, the `squid` processes are running in the **`squid_t`** domain.

SELinux policy defines how processes running in confined domains, such as **`squid_t`**, interact with files, other processes, and the system in general. Files must be labeled correctly to allow squid access to them.

When **`/etc/squid/squid.conf`** is configured so `squid` listens on a port other than the default TCP ports 3128, 3401 or 4827, the **`semanage port`** command must be used to add the required port number to the SELinux policy configuration. The following example demonstrates configuring `squid` to listen on a port that is not initially defined in SELinux policy configuration for `squid`, and, as a consequence, `squid` failing to start. This example also demonstrates how to then configure the SELinux system to allow `squid` to successfully listen on a non-standard port that is not already defined in the policy. This example assumes the *squid* package is installed. Run each command in the example as the root user:

1. Run **`service squid status`** to confirm `squid` is not running:

```
# service squid status
squid is stopped
```

   If the output differs, run **`service squid stop`** to stop the process:

```
# service squid stop
Stopping squid:                                            [  OK  ]
```

2. Run **`semanage port -l | grep -w squid_port_t`** to view the ports SELinux allows `squid` to listen on:

```
semanage port -l | grep -w -i squid_port_t
squid_port_t                   tcp      3128, 3401, 4827
squid_port_t                   udp      3401, 4827
```

3. Edit **`/etc/squid/squid.conf`** as the root user. Configure the **`http_port`** option so it lists a port that is not configured in SELinux policy configuration for `squid`. In this example, `squid` is configured to listen on port 10000:

```
# Squid normally listens to port 3128
http_port 10000
```

4. Run the **`setsebool`** command to make sure the **`squid_connect_any`** Boolean is set to off. This ensures squid is only permitted to operate on specific ports:

```
setsebool -P squid_connect_any 0
```

5. Run **`service squid start`** to start `squid`:

```
# service squid start
```

```
Starting squid: ..................                          [FAILED]
```

An SELinux denial similar to the following is logged:

```
localhost setroubleshoot: SELinux is preventing the squid (squid_t) from binding to
 port 10000. For complete SELinux messages. run sealert -l 97136444-4497-4fff-a7a7-
c4d8442db982
```

6. For SELinux to allow `squid` to listen on port 10000, as used in this example, the following command is required:

```
# semanage port -a -t squid_port_t -p tcp 10000
```

7. Run **`service squid start`** again to start `squid` and have it listen on the new port:

```
# service squid start
Starting squid:         [  OK  ]
```

8. Now that SELinux has been configured to allow `squid` to listen on a non-standard port (TCP 10000 in this example), `squid` starts successfully on this port.

9.2. Types

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

The following types are used with `squid`. Different types allow you to configure flexible access:

`httpd_squid_script_exec_t`
This type is used for utilities such as **`cachemgr.cgi`**, which provides a variety of statistics about squid and its configuration.

`squid_cache_t`
Use this type for data that is cached by squid, as defined by the **`cache_dir`** directive in **`/etc/squid/squid.conf`**. By default, files created in or copied into **`/var/cache/squid`** and **`/var/spool/squid`** are labeled with the **`squid_cache_t`** type. Files for the *squidGuard*[2] URL redirector plugin for `squid` created in or copied to **`/var/squidGuard`** are also labeled with the **`squid_cache_t`** type. Squid is only able to use files and directories that are labeled with this type for its cached data.

`squid_conf_t`
This type is used for the directories and files that `squid` uses for its configuration. Existing files, or those created in or copied to **`/etc/squid`** and **`/usr/share/squid`** are labeled with this type, including error messages and icons.

[2] http://www.squidguard.org/

squid_exec_t
This type is used for the squid binary, **/usr/sbin/squid**.

squid_log_t
This type is used for logs. Existing files, or those created in or copied to **/var/log/squid** or **/var/log/squidGuard** must be labeled with this type.

squid_initrc_exec_t
This type is used for the initialization file required to start `squid` which is located at **/etc/rc.d/init.d/squid**.

squid_var_run_t
This type is used by files in **/var/run**, especially the process id (PID) named **/var/run/squid.pid** which is created by squid when it runs.

9.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you must tell SELinux how you are running services. The following Boolean allows you to tell SELinux how you are running Squid:

squid_connect_any
When enabled, this Boolean permits squid to initiate a connection to a remote host on any port.

9.4. Configuration Examples

9.4.1. Squid Connecting to Non-Standard Ports

The following example provides a real-world demonstration of how SELinux complements Squid by enforcing the above Boolean and by default only allowing access to certain ports. This example will then demonstrate how to change the Boolean and show that access is then allowed.

Note that this is an example only and demonstrates how SELinux can affect a simple configuration of Squid. Comprehensive documentation of Squid is beyond the scope of this document. Refer to the official *Squid documentation*[3] for further details. This example assumes that the Squid host has two network interfaces, Internet access, and that any firewall has been configured to allow access on the internal interface using the default TCP port on which Squid listens (TCP 3128).

1. As the root user, install the *squid* package. Run **rpm -q squid** to see if the *squid* package is installed. If it is not installed, run **yum install squid** as the root user to install it.

2. Edit the main configuration file, **/etc/squid/squid.conf** and confirm that the **cache_dir** directive is uncommented and looks similar to the following:

```
cache_dir ufs /var/spool/squid 100 16 256
```

This line specifies the default settings for the **cache_dir** directive to be used in this example; it consists of the Squid storage format (ufs), the directory on the system where the cache resides (/var/spool/squid), the amount of disk space in megabytes to be used for the cache (100), and

[3] http://www.squid-cache.org/Doc/

finally the number of first-level and second-level cache directories to be created (16 and 256 respectively).

3. In the same configuration file, make sure the **`http_access allow localnet`** directive is uncommented. This allows traffic from the **`localnet`** ACL which is automatically configured in a default installation of Squid on Red Hat Enterprise Linux. It will allow client machines on any existing RFC1918 network to have access through the proxy, which is sufficient for this simple example.

4. In the same configuration file, make sure the **`visible_hostname`** directive is uncommented and is configured to the hostname of the machine. The value should be the fully qualified domain name (FQDN) of the host:

```
visible_hostname squid.example.com
```

5. As the root user, run **`service squid start`** to start `squid`. As this is the first time `squid` has started, this command will initialise the cache directories as specified above in the **`cache_dir`** directive and will then start the `squid` daemon. The output is as follows if `squid` starts successfully:

```
# /sbin/service squid start
init_cache_dir /var/spool/squid... Starting squid: .       [  OK  ]
```

6. Confirm that the `squid` process ID (PID) has started as a confined service, as seen here by the **`squid_var_run_t`** value:

```
# ls -lZ /var/run/squid.pid
-rw-r--r--. root squid unconfined_u:object_r:squid_var_run_t:s0 /var/run/squid.pid
```

7. At this point, a client machine connected to the **`localnet`** ACL configured earlier is successfully able to use the internal interface of this host as its proxy. This can be configured in the settings for all common web browsers, or system-wide. Squid is now listening on the default port of the target machine (TCP 3128), but the target machine will only allow outgoing connections to other services on the Internet via common ports. This is a policy defined by SELinux itself. SELinux will deny access to non-standard ports, as shown in the next step:

8. When a client makes a request using a non-standard port through the Squid proxy such as a website listening on TCP port 10000, a denial similar to the following is logged:

```
SELinux is preventing the squid daemon from connecting to network port 10000
```

9. To allow this access, the **`squid_connect_any`** Boolean must be modified, as it is disabled by default. To turn the **`squid_connect_any`** Boolean on, run the following command as the root user:

```
# setsebool -P squid_connect_any on
```

Note

Do not use the **-P** option if you do not want **setsebool** changes to persist across reboots.

10. The client will now be able to access non-standard ports on the Internet as Squid is now permitted to initiate connections to any port, on behalf of its clients.

MySQL

From the *MySQL*[1] project page:

"The MySQL® database has become the world's most popular open source database because of its consistent fast performance, high reliability and ease of use. It's used on every continent -- Yes, even Antarctica! -- by individual Web developers as well as many of the world's largest and fastest-growing organizations to save time and money powering their high-volume Web sites, business-critical systems and packaged software -- including industry leaders such as Yahoo!, Alcatel-Lucent, Google, Nokia, YouTube, and Zappos.com."

In Red Hat Enterprise Linux, the *mysql-server* package provides MySQL. Run **rpm -q mysql-server** to see if the *mysql-server* package is installed. If it is not installed, run the following command as the root user to install it:

```
yum install mysql-server
```

10.1. MySQL and SELinux

When MySQL is enabled, it runs confined by default. Confined processes run in their own domains, and are separated from other confined processes. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited. The following example demonstrates the MySQL processes running in their own domain. This example assumes the *mysql* package is installed:

1. Run **getenforce** to confirm SELinux is running in enforcing mode:

   ```
   $ getenforce
   Enforcing
   ```

   The **getenforce** command returns **Enforcing** when SELinux is running in enforcing mode.

2. Run **service mysqld start** as the root user to start `mysqld`:

   ```
   # service mysqld start
   Initializing MySQL database:  Installing MySQL system tables... [  OK  ]
   Starting MySQL:                                            [  OK  ]
   ```

3. Run **ps -eZ | grep mysqld** to view the `mysqld` processes:

   ```
   $ ps -eZ | grep mysqld
   unconfined_u:system_r:mysqld_safe_t:s0 6035 pts/1 00:00:00 mysqld_safe
   unconfined_u:system_r:mysqld_t:s0 6123 pts/1   00:00:00 mysqld
   ```

   The SELinux context associated with the `mysqld` processes is **unconfined_u:system_r:mysqld_t:s0**. The second last part of the context, **mysqld_t**,

[1] http://www.mysql.com/why-mysql/

is the type. A type defines a domain for processes and a type for files. In this case, the `mysqld` processes are running in the **`mysqld_t`** domain.

10.2. Types

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

The following types are used with `mysql`. Different types allow you to configure flexible access:

`mysqld_db_t`
This type is used for the location of the MySQL database. In Red Hat Enterprise Linux, the default location for the database is **`/var/lib/mysql`**, however this can be changed. If the location for the MySQL database is changed, the new location must be labeled with this type. Refer to the following example for instructions on how to change the default database location and how to label the new section appropriately.

`mysqld_etc_t`
This type is used for the MySQL main configuration file **`/etc/my.cnf`** and any other configuration files in the **`/etc/mysql`** directory.

`mysqld_exec_t`
This type is used for the `mysqld` binary located at **`/usr/libexec/mysqld`**, which is the default location for the MySQL binary on Red Hat Enterprise Linux. Other systems may locate this binary at **`/usr/sbin/mysqld`** which should also be labeled with this type.

`mysqld_initrc_exec_t`
This type is used for the initialization file for MySQL, located at **`/etc/rc.d/init.d/mysqld`** by default in Red Hat Enterprise Linux.

`mysqld_log_t`
Logs for MySQL need to be labeled with this type for proper operation. All log files in **`/var/log/`** matching the **`mysql.*`** wildcard must be labeled with this type.

`mysqld_var_run_t`
This type is used by files in **`/var/run/mysqld`**, specifically the process id (PID) named **`/var/run/mysqld/mysqld.pid`** which is created by the `mysqld` daemon when it runs. This type is also used for related socket files such as **`/var/lib/mysql/mysql.sock`**. Files such as these must be labeled correctly for proper operation as a confined service.

10.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you must tell SELinux how you are running services. The following Boolean allows you to tell SELinux how you are running MySQL:

`allow_user_mysql_connect`
When enabled, this Boolean allows users to connect to MySQL.

`exim_can_connect_db`
When enabled, this Boolean allows the `exim` mailer to initiate connections to a database server.

`ftpd_connect_db`
When enabled, this Boolean allows `ftp` daemons to initiate connections to a database server.

httpd_can_network_connect_db
Enabling this Boolean is required for a web server to communicate with a database server.

10.4. Configuration Examples

10.4.1. MySQL Changing Database Location

When using Red Hat Enterprise Linux 6, the default location for MySQL to store its database is **/var/lib/mysql**. This is where SELinux expects it to be by default, and hence this area is already labeled appropriately for you, using the **mysqld_db_t** type.

The location where the database is stored can be changed depending on individual environment requirements or preferences, however it is important that SELinux is aware of this new location - that it is labeled accordingly. This example explains how to change the location of a MySQL database and then how to label the new location so that SELinux can still provide its protection mechanisms to the new area based on its contents.

Note that this is an example only and demonstrates how SELinux can affect MySQL. Comprehensive documentation of MySQL is beyond the scope of this document. Refer to the official *MySQL documentation*[2] for further details. This example assumes that the *mysql-server* and *setroubleshoot-server* packages are installed, that the `auditd` service is running, and that there is a valid database in the default location of **/var/lib/mysql**.

1. Run **ls -lZ /var/lib/mysql** to view the SELinux context of the default database location for `mysql`:

```
# ls -lZ /var/lib/mysql
drwx------. mysql mysql unconfined_u:object_r:mysqld_db_t:s0 mysql
```

   This shows **mysqld_db_t** which is the default context element for the location of database files. This context will have to be manually applied to the new database location that will be used in this example in order for it to function properly.

2. Enter **mysqlshow -u root -p** and enter the `mysqld` root password to show the available databases:

```
# mysqlshow -u root -p
Enter password: *******
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| test               |
| wikidb             |
+--------------------+
```

3. Shut down the `mysqld` daemon with **service mysqld stop** as the root user:

```
# service mysqld stop
```

[2] http://dev.mysql.com/doc/

```
Stopping MySQL:                                            [  OK  ]
```

4. Create a new directory for the new location of the database(s). In this example, **/opt/mysql** is used:

```
# mkdir -p /opt/mysql
```

5. Copy the database files from the old location to the new location:

```
# cp -R /var/lib/mysql/* /opt/mysql/
```

6. Change the ownership of this location to allow access by the mysql user and group. This sets the traditional Unix permissions which SELinux will still observe.

```
# chown -R mysql:mysql /opt/mysql
```

7. Run **ls -lZ /opt** to see the initial context of the new directory:

```
# ls -lZ /opt
drwxr-xr-x. mysql mysql unconfined_u:object_r:usr_t:s0   mysql
```

   The context **usr_t** of this newly created directory is not currently suitable to SELinux as a location for MySQL database files. Once the context has been changed, MySQL will be able to function properly in this area.

8. Open the main MySQL configuration file **/etc/my.cnf** with a text editor and modify the **datadir** option so that it refers to the new location. In this example the value that should be entered is **/opt/mysql**.

```
[mysqld]
datadir=/opt/mysql
```

   Save this file and exit.

9. Run **service mysqld start** as the root user to start `mysqld`. The service should fail to start, and a denial will be logged to **/var/log/messages**:

```
SELinux is preventing /usr/libexec/mysqld "write" access on /opt/mysql. For complete
 SELinux messages. run sealert -l b3f01aff-7fa6-4ebe-ad46-abaef6f8ad71
```

   The reason for this denial is that **/opt/mysql** is not labeled correctly for MySQL data files. SELinux is stopping MySQL from having access to the content labeled as **usr_t**. Perform the following steps to resolve this problem:

10. Run the **semanage** command to add a context mapping for **/opt/mysql**:

```
```

```
semanage fcontext -a -t mysqld_db_t "/opt/mysql(/.*)?"
```

11. This mapping is written to the **`/etc/selinux/targeted/contexts/files/file_contexts.local`** file:

```
# grep -i mysql /etc/selinux/targeted/contexts/files/file_contexts.local

/opt/mysql(/.*)?    system_u:object_r:mysqld_db_t:s0
```

12. Now use the **`restorecon`** command to apply this context mapping to the running system:

```
restorecon -R -v /opt/mysql
```

13. Now that the **`/opt/mysql`** location has been labeled with the correct context for MySQL, the `mysqld` daemon starts:

```
# service mysqld start
Starting MySQL:                                            [  OK  ]
```

14. Confirm the context has changed for **`/opt/mysql`**:

```
ls -lZ /opt
drwxr-xr-x. mysql mysql system_u:object_r:mysqld_db_t:s0 mysql
```

15. The location has been changed and labeled, and the `mysqld` daemon has started successfully. At this point all running services should be tested to confirm normal operation.

PostgreSQL

From the *PostgreSQL*[1] project page:

"PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness."

In Red Hat Enterprise Linux 6, the *postgresql-server* package provides PostgreSQL. Run **`rpm -q postgresql-server`** to see if the *postgresql-server* package is installed. If it is not installed, run the following command as the root user to install it:

```
yum install postgresql-server
```

11.1. PostgreSQL and SELinux

When PostgreSQL is enabled, it runs confined by default. Confined processes run in their own domains, and are separated from other confined processes. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited. The following example demonstrates the PostgreSQL processes running in their own domain. This example assumes the *postgresql-server* package is installed:

1. Run **`getenforce`** to confirm SELinux is running in enforcing mode:

   ```
   $ getenforce
   Enforcing
   ```

   The **`getenforce`** command returns **`Enforcing`** when SELinux is running in enforcing mode.

2. Run **`service postgresql start`** as the root user to start `postgresql`:

   ```
   service postgresql start
   Starting postgresql service:                               [  OK  ]
   ```

3. Run **`ps -eZ | grep postgres`** to view the `postgresql` processes:

   ```
   ps -eZ | grep postgres
   unconfined_u:system_r:postgresql_t:s0 395 ?    00:00:00 postmaster
   unconfined_u:system_r:postgresql_t:s0 397 ?    00:00:00 postmaster
   unconfined_u:system_r:postgresql_t:s0 399 ?    00:00:00 postmaster
   unconfined_u:system_r:postgresql_t:s0 400 ?    00:00:00 postmaster
   unconfined_u:system_r:postgresql_t:s0 401 ?    00:00:00 postmaster
   unconfined_u:system_r:postgresql_t:s0 402 ?    00:00:00 postmaster
   ```

   The SELinux context associated with the `postgresql` processes is **`unconfined_u:system_r:postgresql_t:s0`**. The second last part of the context,

[1] http://www.postgresql.org/about/

postgresql_t, is the type. A type defines a domain for processes and a type for files. In this case, the `postgresql` processes are running in the **postgresql_t** domain.

11.2. Types

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

The following types are used with `postgresql`. Different types allow you to configure flexible access:

postgresql_db_t
This type is used for several locations. The locations labeled with this type are used for data files for PostgreSQL:
- **/usr/lib/pgsql/test/regres**
- **/usr/share/jonas/pgsql**
- **/var/lib/pgsql/data**
- **/var/lib/postgres(ql)?**

postgresql_etc_t
This type is used for configuration files in **/etc/postgresql**.

postgresql_exec_t
This type is used for several locations. The locations labeled with this type are used for binaries for PostgreSQL:
- **/usr/bin/initdb(.sepgsql)?**
- **/usr/bin/(se)?postgres**
- **/usr/lib(64)?/postgresql/bin/.***
- **/usr/lib/phsql/test/regress/pg_regress**

postgresql_initrc_exec_t
This type is used for the PostgreSQL initialization file located at **/etc/rc.d/init.d/postgresql**.

postgresql_log_t
This type is used for several locations. The locations labeled with this type are used for log files:
- **/var/lib/pgsql/logfile**
- **/var/lib/pgsql/pgstartup.log**
- **/var/lib/sepgsql/pgstartup.log**
- **/var/log/postgresql**
- **/var/log/postgres.log.***
- **/var/log/rhdb/rhdb**
- **/var/log/sepostgresql.log.***

postgresql_var_run_t
This type is used for run-time files for PostgreSQL, such as the process id (PID) in **/var/run/postgresql**.

11.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you must tell SELinux how you are running services. The following Boolean allows you to tell SELinux how you are running PostgreSQL:

allow_user_postgresql_connect
Having this Boolean enabled allows any user domain (as defined by PostgreSQL) to make connections to the database server.

11.4. Configuration Examples

11.4.1. PostgreSQL Changing Database Location

When using Red Hat Enterprise Linux 6, the default location for PostgreSQL to store its database is **/var/lib/pgsql/data**. This is where SELinux expects it to be by default, and hence this area is already labeled appropriately for you, using the **postgresql_db_t** type.

The area where the database is located can be changed depending on individual environment requirements or preferences, however it is important that SELinux is aware of this new location - that it is labeled accordingly. This example explains how to change the location of a PostgreSQL database and then how to label the new location so that SELinux can still provide its protection mechanisms to the new area based on its contents.

Note that this is an example only and demonstrates how SELinux can affect PostgreSQL. Comprehensive documentation of PostgreSQL is beyond the scope of this document. Refer to the official *PostgreSQL documentation*[2] for further details. This example assumes that the *postgresql-server* package is installed.

1. Run **ls -lZ /var/lib/pgsql** to view the SELinux context of the default database location for `postgresql`:

```
# ls -lZ /var/lib/pgsql
drwx------. postgres postgres system_u:object_r:postgresql_db_t:s0 data
```

   This shows **postgresql_db_t** which is the default context element for the location of database files. This context will have to be manually applied to the new database location that will be used in this example in order for it to function properly.

2. Create a new directory for the new location of the database(s). In this example, **/opt/postgresql/data** is used. If you use a different location, replace the text in the following steps with your location:

```
# mkdir -p /opt/postgresql/data
```

[2] http://www.postgresql.org/docs/

3. Perform a directory listing of the new location. Note that the initial context of the new directory is *usr_t*. This context is not sufficient for SELinux to offer its protection mechanisms to PostgreSQL. Once the context has been changed, it will be able to function properly in the new area.

```
# ls -lZ /opt/postgresql/
drwxr-xr-x. root root unconfined_u:object_r:usr_t:s0   data
```

4. Change the ownership of the new location to allow access by the postgres user and group. This sets the traditional Unix permissions which SELinux will still observe.

```
# chown -R postgres:postgres /opt/postgresql
```

5. Open the PostgreSQL init file **/etc/rc.d/init.d/postgresql** with a text editor and modify the **PGDATA** and **PGLOG** variables to point to the new location:

```
# vi /etc/rc.d/init.d/postgresql
PGDATA=/opt/postgresql/data
PGLOG=/opt/postgresql/data/pgstartup.log
```

   Save this file and exit the text editor.

6. Initialize the database in the new location.

```
su - postgres -c "initdb -D /opt/postgresql/data"
```

7. Having changed the database location, starting the service will fail at this point:

```
# service postgresql start
Starting postgresql service:    [FAILED]
```

   SELinux has caused the service to not start. This is because the new location is not properly labelled. The following steps explain how to label the new location (**/opt/postgresql**) and start the postgresql service properly:

8. Run the **semanage** command to add a context mapping for **/opt/postgresql** and any other directories/files within it:

```
semanage fcontext -a -t postgresql_db_t "/opt/postgresql(/.*)?"
```

9. This mapping is written to the **/etc/selinux/targeted/contexts/files/file_contexts.local** file:

```
# grep -i postgresql /etc/selinux/targeted/contexts/files/file_contexts.local

/opt/postgresql(/.*)?    system_u:object_r:postgresql_db_t:s0
```

10. Now use the **restorecon** command to apply this context mapping to the running system:

```
restorecon -R -v /opt/postgresql
```

11. Now that the **/opt/postgresql** location has been labeled with the correct context for PostgreSQL, the `postgresql` service will start successfully:

```
# service postgresql start
Starting postgreSQL service:                                [  OK  ]
```

12. Confirm the context is correct for **/opt/postgresql**:

```
ls -lZ /opt
drwxr-xr-x. root root system_u:object_r:postgresql_db_t:s0 postgresql
```

13. Check with the **ps** command that the `postgresql` process displays the new location:

```
# ps aux | grep -i postmaster

postgres 21564  0.3  0.3  42308  4032 ?        S    10:13   0:00 /usr/bin/postmaster -p
 5432 -D /opt/postgresql/data
```

14. The location has been changed and labeled, and the `postgresql` daemon has started successfully. At this point all running services should be tested to confirm normal operation.

rsync

From the *Rsync*[1] project page:

"rsync is an open source utility that provides fast incremental file transfer."

When using Red Hat Enterprise Linux, the *rsync* package provides rsync. Run **rpm -q rsync** to see if the *rsync* package is installed. If it is not installed, run the following command as the root user to install it:

```
yum install rsync
```

12.1. rsync and SELinux

From the Red Hat Enterprise Linux 6 SELinux *rsync_selinux(8)* man page: "SELinux requires files to have an extended attribute to define the file type. Policy governs the access daemons have to these files. If you want to share files using the rsync daemon, you must label the files and directories *public_content_t*."

Like most services, correct labeling is required for SELinux to perform its protection mechanisms over `rsync`.

12.2. Types

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

The following types are used with **rsync**. Different types all you to configure flexible access:

public_content_t
: This is a generic type used for the location of files (and the actual files) to be shared via `rsync`. If a special directory is created to house files to be shared with `rsync`, the directory and its contents need to have this label applied to them.

rsync_exec_t
: This type is used for the **/usr/bin/rsync** system binary.

rsync_log_t
: This type is used for the `rsync` log file, located at **/var/log/rsync.log** by default. To change the location of the file rsync logs to, use the **--log-file=FILE** option to the **rsync** command at run-time.

rsync_var_run_t
: This type is used for the `rsyncd` lock file, located at **/var/run/rsyncd.lock**. This lock file is used by the `rsync` server to manage connection limits.

[1] http://www.samba.org/rsync/

12.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you must tell SELinux how you are running services. The following Booleans allow you to tell SELinux how you are running rsync:

`allow_rsync_anon_write`
Having this Boolean enabled allows `rsync` in the *rsync_t* domain to manage files, links and directories that have a type of *public_content_rw_t*. Often these are public files used for public file transfer services. Files and directories must be labeled **`public_content_rw_t`**.

`rsync_client`
Having this Boolean enabled aloows `rsync` to initiate connections to ports defined as *rsync_port_t*, as well as allowing `rsync` to manage files, links and directories that have a type of *rsync_data_t*. Note that the `rsync` daemon must be in the *rsync_t* domain in order for SELinux to enact its control over `rsync`. The configuration example in this chapter demonstrates `rsync` running in the *rsync_t* domain.

`rsync_export_all_ro`
Having this Boolean enabled allows `rsync` in the *rsync_t* domain to export NFS and CIFS file systems with read-only access to clients.

12.4. Configuration Examples

12.4.1. Rsync as a daemon

When using Red Hat Enterprise Linux, rsync can be used as a daemon so that multiple clients can directly communicate with it as a central server, in order to house centralized files and keep them synchronized. The following example will demonstrate running rsync as a daemon over a network socket in the correct domain, and how SELinux expects this daemon to be running on a pre-defined (in SELinux policy) TCP port. This example will then show how to modify SELinux policy to allow the `rsync` daemon to run normally on a non-standard port.

This example will be performed on a single system to demonstrate SELinux policy and its control over local daemons and processes. Note that this is an example only and demonstrates how SELinux can affect rsync. Comprehensive documentation of rsync is beyond the scope of this document. Refer to the official *rsync documentation*[2] for further details. This example assumes that the *rsync*, *setroubleshoot-server* and *audit* packages are installed, that the SELinux targeted policy is used and that SELinux is running in enforcing mode.

Getting rsync to launch as rsync_t

1. Run **`getenforce`** to confirm SELinux is running in enforcing mode:

```
$ getenforce
Enforcing
```

   The **`getenforce`** command returns **`Enforcing`** when SELinux is running in enforcing mode.

2. Run the **`which`** command to confirm that the rsync binary is in the system path:

[2] http://www.samba.org/rsync/documentation.html

```
$ which rsync
/usr/bin/rsync
```

3. When running `rsync` as a daemon, a configuration file should be used and saved as **`/etc/rsyncd.conf`**. Note that the following configuration file used in this example is very simple and is not indicative of all the possible options that are available, rather it is just enough to demonstrate the `rsync` daemon:

```
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
lock file = /var/run/rsync.lock
[files]
        path = /srv/files
        comment = file area
        read only = false
 timeout = 300
```

4. Now that a simple configuration file exists for rsync to operate in daemon mode, this step demonstrates that simply running **`rsync --daemon`** is not sufficient for SELinux to offer its protection over rsync. Refer to the following output:

```
# rsync --daemon

# ps x | grep rsync
 8231 ?        Ss     0:00 rsync --daemon
 8233 pts/3    S+     0:00 grep rsync

# ps -eZ | grep rsync
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 8231 ? 00:00:00 rsync
```

   Note that in the output from the final **`ps`** command, the context shows the `rsync` daemon running in the **`unconfined_t`** domain. This indicates that rsync has not transitioned to the **`rsync_t`** domain as it was launched by the **`rsync --daemon`** command. At this point SELinux can not enforce its rules and policy over this daemon. Refer to the following steps to see how to fix this problem. In the following steps, `rsync` will transition to the **`rsync_t`** domain by launching it from a properly-labeled init script. Only then can SELinux and its protection mechanisms have an effect over `rsync`. This `rsync` process should be killed before proceeding to the next step.

5. A custom init script for rsync is needed for this step. Save the following to **`/etc/rc.d/init.d/rsyncd`**.

```
#!/bin/bash

# Source function library.
. /etc/rc.d/init.d/functions

[ -f /usr/bin/rsync ] || exit 0

case "$1" in
start)
action "Starting rsyncd: " /usr/bin/rsync --daemon
;;
stop)
action "Stopping rsyncd: " killall rsync
```

```
;;
*)
echo "Usage: rsyncd {start|stop}"
exit 1
esac
exit 0
```

The following steps show how to label this script as **initrc_exec_t**:

6. Run the **semanage** command to add a context mapping for **/etc/rc.d/init.d/rsyncd**:

```
semanage fcontext -a -t initrc_exec_t "/etc/rc.d/init.d/rsyncd"
```

7. This mapping is written to the **/etc/selinux/targeted/contexts/files/file_contexts.local** file:

```
# grep rsync /etc/selinux/targeted/contexts/files/file_contexts.local

/etc/rc.d/init.d/rsyncd    system_u:object_r:initrc_exec_t:s0
```

8. Now use the **restorecon** command to apply this context mapping to the running system:

```
restorecon -R -v /etc/rc.d/init.d/rsyncd
```

9. Run the **ls -lZ** command to confirm the script has been labeled appropriately. Note that in the following output the script has been labeled as **initrc_exec_t**:

```
 ls -lZ /etc/rc.d/init.d/rsyncd
-rwxr-xr-x. root root system_u:object_r:initrc_exec_t:s0 /etc/rc.d/init.d/rsyncd
```

10. Launch `rsyncd` via the new script. Now that rsync has started from an init script that has been appropriately labeled, the process will start as **rsync_t**:

```
# service rsyncd start
Starting rsyncd:                                           [  OK  ]

ps -eZ | grep rsync
unconfined_u:system_r:rsync_t:s0 9794 ?        00:00:00 rsync
```

SELinux can now enforce its protection mechanisms over the `rsync` daemon as it is now runing in the **rsync_t** domain.

This example demonstrated how to get `rsyncd` running in the **rsync_t** domain. The next example shows how to get this daemon successfully running on a non-default port. TCP port 10000 is used in the next example.

Running the rsync daemon on a non-default port

1. Modify the **/etc/rsyncd.conf** file and add the **port = 10000** line at the top of the file in the global configuration area (ie., before any file areas are defined). The new configuration file will look like:

```
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
lock file = /var/run/rsync.lock
port = 10000
[files]
        path = /srv/files
        comment = file area
        read only = false
 timeout = 300
```

2. After launching rsync from the init script with this new setting, a denial similar to the following is logged by SELinux:

```
Jul 22 10:46:59 localhost setroubleshoot: SELinux is preventing the rsync (rsync_t)
 from binding to port 10000. For complete SELinux messages. run sealert -l
 c371ab34-639e-45ae-9e42-18855b5c2de8
```

3. Run the **semanage** command to add TCP port 10000 to SELinux policy in **rsync_port_t**:

```
# semanage port -a -t rsync_port_t -p tcp 10000
```

4. Now that TCP port 10000 has been added to SELinux policy for **rsync_port_t**, `rsyncd` will start and operate normally on this port:

```
# service rsyncd start
Starting rsyncd:                                           [  OK  ]
```

```
# netstat -lnp | grep 10000
tcp        0      0 0.0.0.0:10000   0.0.0.0:*      LISTEN      9910/rsync
```

SELinux has had its policy modified and is now permitting `rsyncd` to operate on TCP port 10000.

Postfix

From the *Postfix*[1] project page:

"What is Postfix? It is Wietse Venema's mailer that started life at IBM research as an alternative to the widely-used Sendmail program. Postfix attempts to be fast, easy to administer, and secure. The outside has a definite Sendmail-ish flavor, but the inside is completely different."

In Red Hat Enterprise Linux, the *postfix* package provides postfix. Run **`rpm -q postfix`** to see if the *postfix* package is installed. If it is not installed, run the following command as the root user to install it:

```
yum install postfix
```

13.1. Postfix and SELinux

When Postfix is enabled, it runs confined by default. Confined processes run in their own domains, and are separated from other confined processes. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited. The following example demonstrates the Postfix and related processes running in their own domain. This example assumes the *postfix* package is installed and that the Postfix service has been started:

1. Run **`getenforce`** to confirm SELinux is running in enforcing mode:

   ```
   $ getenforce
   Enforcing
   ```

   The **`getenforce`** command returns **`Enforcing`** when SELinux is running in enforcing mode.

2. Run **`service postfix start`** as the root user to start `postfix`:

   ```
   service postfix start
   Starting postfix:                          [  OK  ]
   ```

3. Run **`ps -eZ | grep postfix`** to view the `postfix` processes:

   ```
   ps -eZ | grep postfix
   system_u:system_r:postfix_master_t:s0 1651 ?   00:00:00 master
   system_u:system_r:postfix_pickup_t:s0 1662 ?   00:00:00 pickup
   system_u:system_r:postfix_qmgr_t:s0 1663 ?     00:00:00 qmgr
   ```

   For example, the SELinux context associated with the Postfix `master` process is **`unconfined_u:system_r:postfix_master_t:s0`**. The second last part of the context, **`postfix_master_t`**, is the type for this process. A type defines a domain for processes and a type for files. In this case, the `master` process is running in the **`postfix_master_t`** domain.

[1] http://www.postfix.org/

13.2. Types

Type Enforcement is the main permission control used in SELinux targeted policy. All files and processes are labeled with a type: types define a domain for processes and a type for files. SELinux policy rules define how types access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

The following types are used with **Postfix**. Different types all you to configure flexible access:

postfix_etc_t
This type is used for configuration files for Postfix in **/etc/postfix**.

postfix_data_t
This type is used for Postfix data files in **/var/lib/postfix**.

Note

To see the full list of files and their types for Postfix, run the following command:

```
$ grep postfix /etc/selinux/targeted/contexts/files/file_contexts
```

13.3. Booleans

SELinux is based on the least level of access required for a service to run. Services can be run in a variety of ways; therefore, you must tell SELinux how you are running services. The following Boolean allows you to tell SELinux how you are running Postfix:

allow_postfix_local_write_mail_spool
Having this Boolean enables Postfix to write to the local mail spool on the system. Postfix requires this Boolean to be enabled for normal operation when local spools are used.

13.4. Configuration Examples

13.4.1. SpamAssassin and Postfix

From the *SpamAssassin*[2] project page:

"Open Source mail filter, written in Perl, to identify spam using a wide range of heuristic tests on mail headers and body text. Free software."

When using Red Hat Enterprise Linux, the *spamassassin* package provides SpamAssassin. Run **rpm -q spamassassin** to see if the *spamassassin* package is installed. If it is not installed, run the following command as the root user to install it:

```
yum install spamassassin
```

SpamAssassin operates in tandom with a mailer such as Postfix to provide spam-filtering capabilities. In order for SpamAssassin to effectively intercept, analyze and filter mail, it must listen on a network

[2] http://spamassassin.apache.org/

interface. The default port for SpamAssassin is TCP/783, however this can be changed. The following example provides a real-world demonstration of how SELinux complements SpamAssassin by only allowing it access to a certain port by default. This example will then demonstrate how to change the port and have SpamAssassin operate on a non-default port.

Note that this is an example only and demonstrates how SELinux can affect a simple configuration of SpamAssassin. Comprehensive documentation of SpamAssassin is beyond the scope of this document. Refer to the official *SpamAssassin documentation*[3] for further details. This example assumes the *spamassassin* is installed, that any firewall has been configured to allow access on the ports in use, that the SELinux targeted policy is used, and that SELinux is running in enforcing mode:

Running SpamAssassin on a non-default port

1. Run the **semanage** command to show the port that SELinux allows `spamd` to listen on by default:

```
# semanage port -l | grep spamd
spamd_port_t  tcp 783
```

   This output shows that TCP/783 is defined in **spamd_port_t** as the port for SpamAssassin to operate on.

2. Edit the **/etc/sysconfig/spamassassin** configuration file and modify it so that it will start SpamAssassin on the example port TCP/10000:

```
# Options to spamd
SPAMDOPTIONS="-d -p 10000 -c m5 -H"
```

   This line now specifies that SpamAssassin will operate on port 10000. The rest of this example will show how to modify SELinux policy to allow this socket to be opened.

3. Start SpamAssassin and an error message similar to the following will appear:

```
# service spamassassin start
Starting spamd: [2203] warn: server socket setup failed, retry 1: spamd: could not create
 INET socket on 127.0.0.1:10000: Permission denied
[2203] warn: server socket setup failed, retry 2: spamd: could not create INET socket on
 127.0.0.1:10000: Permission denied
[2203] error: spamd: could not create INET socket on 127.0.0.1:10000: Permission denied
spamd: could not create INET socket on 127.0.0.1:10000: Permission denied
                                                           [FAILED]
```

   This output means that SELinux has blocked access to this port.

4. A denial similar to the following will be logged by SELinux:

```
SELinux is preventing the spamd (spamd_t) from binding to port 10000.
```

5. As the root user, run the **semanage** command to modify SELinux policy in order to allow SpamAssassin to operate on the example port (TCP/10000):

[3] http://spamassassin.apache.org/doc.html

```
semanage port -a -t spamd_port_t -p tcp 10000
```

6. Confirm that SpamAssassin will now start and is operating on TCP port 10000:

```
# service spamassassin start
Starting spamd:     [ OK ]

# netstat -lnp | grep 10000
tcp 0 0 127.0.0.1:10000 0.0.0.0:* LISTEN 2224/spamd.pid
```

7. At this point, `spamd` is properly operating on TCP port 10000 as it has been allowed access to that port by SELinux policy.

DHCP

DHCPD is the daemon used in Red Hat Enterprise Linux to dynamically deliver and configure Layer 3 TCP/IP details for clients.

The *dhcp* package provides the DHCP server, `dhcpd`. Run **`rpm -q dhcp`** to see if the *dhcp* package is installed. If it is not installed, run the following command as the root user to install it:

```
yum install dhcp
```

14.1. DHCP and SELinux

When DHCPD is enabled, it runs confined by default. Confined processes run in their own domains, and are separated from other confined processes. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited. The following example demonstrates the DHCPD and related processes running in their own domain. This example assumes the *dhcp* package is installed and that the DHCPD service has been started:

1. Run **`getenforce`** to confirm SELinux is running in enforcing mode:

   ```
   $ getenforce
   Enforcing
   ```

   The **`getenforce`** command returns **`Enforcing`** when SELinux is running in enforcing mode.

2. Run **`service dhcpd start`** as the root user to start `DHCPD`:

   ```
   service dhcpd start
   Starting dhcpd:                                [  OK  ]
   ```

3. Run **`ps -eZ | grep dhcpd`** to view the `dhcpd` processes:

   ```
   ps -eZ | grep dhcpd
   unconfined_u:system_r:dhcpd_t:s0 5483 ?        00:00:00 dhcpd
   ```

   The SELinux context associated with the dhcpd process is **`unconfined_u:system_r:dhcpd_t:s0`**.

14.2. Types

The following types are used with **`dhcpd`**:

`dhcp_etc_t`
: This type is mainly used for files in **`/etc`**, including configuration files.

`dhcp_var_run_t`
: This type is used for the PID file for dhcpd, in **`/var/run`**.

Note

To see the full list of files and their types for dhcp, run the following command:

```
$ grep dhcp /etc/selinux/targeted/contexts/files/file_contexts
```

References

The following references are pointers to additional information that is relevant to SELinux but beyond the scope of this guide. Note that due to the rapid development of SELinux, some of this material may only apply to specific releases of Red Hat Enterprise Linux.

Books

SELinux by Example
Mayer, MacMillan, and Caplan

Prentice Hall, 2007

SELinux: NSA's Open Source Security Enhanced Linux
Bill McCarty

O'Reilly Media Inc., 2004

Tutorials and Help

Tutorials and talks from Russell Coker
http://www.coker.com.au/selinux/talks/ibmtu-2004/

Dan Walsh's Journal
http://danwalsh.livejournal.com/

Red Hat Knowledgebase
http://kbase.redhat.com/

General Information

NSA SELinux main website
http://www.nsa.gov/research/selinux/index.shtml

NSA SELinux FAQ
http://www.nsa.gov/research/selinux/faqs.shtml

Mailing Lists

NSA SELinux mailing list
http://www.nsa.gov/research/selinux/list.shtml

Fedora SELinux mailing list
http://www.redhat.com/mailman/listinfo/fedora-selinux-list

Community

SELinux Project Wiki
http://selinuxproject.org/page/Main_Page

SELinux community page
http://selinux.sourceforge.net/

IRC
irc.freenode.net, #selinux