Blueprints

# First Steps with Security-Enhanced Linux (SELinux): Hardening the Apache Web Server

Blueprints

# First Steps with Security-Enhanced Linux (SELinux): Hardening the Apache Web Server

**Note**

Before using this information and the product it supports, read the information in "Notices" on page 35.

**First Edition (August 2009)**

# Contents

# Introduction

This blueprint provides a brief introduction to basic Security-Enhanced Linux (SELinux) commands and concepts, including Boolean variables. In addition, the paper shows you how to increase the security of the Apache Web server with SELinux by using these concepts. Key tools and technologies discussed in this demonstration include security-enhanced Linux (SELinux), mandatory access control (MAC), getenforce, sestatus, getsebool, and setsebool.

## Intended audience

This blueprint is intended for Linux system or network administrators who want to learn more about securing their systems with SELinux. You should be familiar with installing and configuring Linux distributions, networks, and the Apache Web server.

## Scope and purpose

This paper provides a basic overview of SELinux, SELinux Boolean variables, and hardening Apache on Red Hat Enterprise Linux (RHEL) 5.3.

For more information about configuring RHEL 5.3, see the documentation supplied with your installation media or the distribution Web site. For more information about SELinux, see "Related information and downloads," on page 33.

## Software requirements

This blueprint is written and tested using Red Hat Enterprise Linux (RHEL) 5.3.

## Hardware requirements

The information contained in this blueprint is tested on different models of IBM System x and System p hardware. For a list of hardware supported by RHEL 5.3, see the documentation supplied with your Linux distribution.

## Author names

Robert Sisk

## Other contributors

Monza Lui

Kersten Richter

Robb Romans

## IBM Services

Linux offers flexibility, options, and competitive total cost of ownership with a world class enterprise operating system. Community innovation integrates leading-edge technologies and best practices into Linux.

IBM® is a leader in the Linux community with over 600 developers in the IBM Linux Technology Center working on over 100 open source projects in the community. IBM supports Linux on all IBM servers, storage, and middleware, offering the broadest flexibility to match your business needs.

For more information about IBM and Linux, go to ibm.com/linux  (https://www.ibm.com/linux)

## IBM Support

Questions and comments regarding this documentation can be posted on the developerWorks Security Blueprint Community Forum: http://www.ibm.com/developerworks/forums/forum.jspa?forumID=1271

The IBM developerWorks® discussion forums let you ask questions, share knowledge, ideas, and opinions about technologies and programming techniques with other developerWorks users. Use the forum content at your own risk. While IBM will attempt to provide a timely response to all postings, the use of this developerWorks forum does not guarantee a response to every question that is posted, nor do we validate the answers or the code that are offered.

## Typographic conventions

The following typographic conventions are used in this Blueprint:

| | |
|---|---|
| **Bold** | Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that the user selects. |
| *Italics* | Identifies parameters whose actual names or values are to be supplied by the user. |
| `Monospace` | Identifies examples of specific data values, examples of text like what you might see displayed, examples of portions of program code like what you might write as a programmer, messages from the system, or information you should actually type. |

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x® running Linux and PowerLinux™. You can learn more about the systems to which this information applies.

# Chapter 1. Scope, requirements, and support

This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## Systems to which this information applies

System x running Linux and PowerLinux

# Chapter 2. Security-Enhanced Linux overview

Security-Enhanced Linux (SELinux) is a component of the Linux operating system developed primarily by the United States National Security Agency. SELinux provides a method for creation and enforcement of mandatory access control (MAC) policies. These policies confine users and processes to the minimal amount of privilege required to perform assigned tasks.

For more information about the history of SELinux, see http://en.wikipedia.org/wiki/Selinux.

Since its release to the open source community in December 2000, the SELinux project has gained improvements such as predefined Boolean variables that make it easier to use. This paper helps you understand how to use these variables to configure SELinux policies on your system and to secure the Apache **httpd** daemon.

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## Access control: MAC and DAC

Access level is important to computer system security. To compromise a system, attackers try to gain any possible level of access and then try to escalate that level until they are able to obtain restricted data or make unapproved system modifications. Because each user has some level of system access, every user account on your system increases the potential for abuse. System security has historically relied on trusting users not to abuse their access, but this trust has proven to be problematic. Today, server consolidation leads to more users per system. Outsourcing of Systems Management gives legitimate access, often at the system administrator level, to unknown users. Because server consolidation and outsourcing can be financially advantageous, what can you do to prevent abuse on Linux systems? To begin to answer that question, let's take a look at discretionary access control (DAC) and mandatory access control (MAC) and their differences.

Discretionary access control (DAC), commonly known as *file permissions*, is the predominant access control mechanism in traditional UNIX and Linux systems. You may recognize the **drwxr-xr-x** or the **ugo** abbreviations for owner, group, and other permissions seen in a directory listing. In DAC, generally the resource owner (a user) controls who has access to a resource. For convenience, some users commonly set dangerous DAC file permissions that allow every user on the system to read, write, and execute many files that they own. In addition, a process started by a user can modify or delete any file to which the user has access. Processes that elevate their privileges high enough could therefore modify or delete system files. These instances are some of the disadvantages of DAC.

In contrast to DAC, mandatory access control (MAC) regulates user and process access to resources based upon an organizational (higher-level) security policy. This policy is a collection of rules that specify what types of access are allowed on a system. System policy is related to MAC in the same way that firewall rules are related to firewalls.

SELinux is a Linux kernel implementation of a flexible MAC mechanism called type enforcement. In type enforcement, a type identifier is assigned to every user and object. An object can be a file or a process. To access an object, a user must be authorized for that object type. These authorizations are defined in a SELinux policy. Let's work through some examples and you will develop a better understanding of MAC and how it relates to SELinux.

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## SELinux basics

It is a good practice not to use the root user unless necessary. However for demonstrating how to use SELinux, the root user is used in the examples in this blueprint. Some of the commands shown require root privileges to run them; for example, running **getenforce** and editing the **/etc/selinux/config** file.

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## Run modes

You can enable or disable SELinux policy enforcement on a Red Hat Enterprise Linux system during or after operating system installation.

When disabled, SELinux has no effect on the system. When enabled, SELinux runs in one of two modes:

- **Enforcing**: SELinux is enabled and SELinux policy is enforced
- **Permissive**: SELinux is enabled but it only logs warnings instead of enforcing the policy

When prompted during operating system installation, if you choose to enable SELinux, it is installed with a default security policy and set to run in the enforcing mode.

Confirm the status of SELinux on your system. Like in many UNIX or Linux operating systems, there is more than one way to perform a task. To check the current mode, run one of the following commands: **getenforce**, **sestatus**, or **cat /etc/selinux/config**.

- The **getenorce** command returns the current SELinux run mode, or `Disabled` if SELinux is not enabled. In the following example, **getenforce** shows that SELinux is enabled and enforcing the current SELinux policy:

  ```
  [root@localhost ~]$ getenforce
  Enforcing
  ```

  If your system is displaying `Permissive` or `Disabled` and you want to follow along with the instructions, change the **/etc/selinux/config** file to run in `Enforcing` mode before continuing with the demonstration. Remember that if you are in `Disabled` mode, you should change first to `Permissive` and then to `Enforcing`.
- The **setstatus** command returns the current run mode, along with information about the SELinux policy if SELinux is enabled. In the following example, **setstatus** shows that SELinux is enabled and enforcing the current SELinux policy:

  ```
  [root@localhost ~]$ sestatus
  SELinux status:                 enabled
  SELinuxfs mount:                /selinux
  Current mode:                   enforcing
  Mode from config file:          enforcing
  Policy version:                 21
  Policy from config file:        targeted
  ```
- The **/etc/selinux/config** file configures SELinux and controls the mode as well as the active policy. Changes to the **/etc/selinux/config** file become effective only after you reboot the system. In the following example, the file shows that the mode is set to `enforcing` and the current policy type is `targeted`.

```
[root@localhost ~]$ cat /etc/selinux/config
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - SELinux is fully disabled.
SELINUX=enforcing
# SELINUXTYPE= type of policy in use. Possible values are:
#       targeted - Only targeted network daemons are protected.
#       strict - Full SELinux protection.
SELINUXTYPE=targeted
```

To enable SELinux, you need to set the value of the **SELINUX** parameter in the **/etc/selinux/config** file to either `enforcing` or `permissive`. If you enable SELinux in the config file, you must reboot your system to start SELinux. We recommend that you set `SELINUX=permissive` if the file system has never been labeled, has not been labeled recently, or you are not sure when it was last labeled. Note that file system labeling is the process of assigning a label containing security-relevant information to each file. In SELinux a file label is composed of the user, role, and type such as `system_u:object_r:httpd_sys_content_t`. `Permissive` mode ensures that SELinux does not interfere with the boot sequence if a command in the sequence occurs before the file system relabel is completed. Once the system is up and running, you can change the SELinux mode to `enforcing`.

If you want to change the mode of SELinux on a running system, use the **setenforce** command. Entering `setenforce enforcing` changes the mode to `enforcing` and `setenforce permissive` changes the mode to `permissive`. To disable SELinux, edit the **/etc/selinux/config** file as described previously and reboot. You cannot disable or enable SELinux on a running system from the command line; you can only switch between `enforcing` and `permissive` when SELinux is enabled.

Change the mode of SELinux to `permissive` by entering the following command:

```
[root@localhost ~]$  setenforce permissive
```

Recheck the output from **getenforce**, **sestatus**, and **cat /etc/selinux/config**.

- The **getenforce** command returns `Permissive`, confirming the mode change:

  ```
  [root@localhost ~]$ getenforce
  Permissive
  ```

- The **sestatus** command also returns a `Permissive` mode value:

  ```
  [root@localhost ~]$sestatus
  SELinux status:                 enabled
  SELinuxfs mount:                /selinux
  Current mode:                   permissive
  Mode from config file:          enforcing
  Policy version:                 21
  Policy from config file:        targeted
  ```

- After changing the mode to `permissive`, both the **getenforce** and **sestatus** commands return the correct `permissive` mode. However, look carefully at the output from the **sestatus** command:

  ```
  [root@localhost ~]$ cat /etc/selinux/config
  # This file controls the state of SELinux on the system.
  # SELINUX= can take one of these three values:
  #       enforcing - SELinux security policy is enforced.
  #       permissive - SELinux prints warnings instead of enforcing.
  #       disabled - SELinux is fully disabled.
  SELINUX=enforcing
  # SELINUXTYPE= type of policy in use. Possible values are:
  #       targeted - Only targeted network daemons are protected.
  #       strict - Full SELinux protection.
  SELINUXTYPE=targeted
  [root@localhost ~]$
  ```

  The **Mode from config file** parameter is `enforcing`. This setting is consistent with the **cat /etc/selinux/config** output because the config file was not changed. This status implies that the changes

made by the **setenforce** command does not carry over to the next boot. If you reboot, SELinux returns to run state as configured in **/etc/selinux/conf** in `enforcing` mode.

Change the running mode back to `enforcing` by entering the following command:

```
[root@localhost ~]$  setenforce enforcing
```

The following output confirms the mode change:

```
[root@localhost ~]$ getenforce
Enforcing
```

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## Security contexts

The concept of type enforcement and the SELinux type identifier were discussed in the Overview. Let's explore these concepts in more detail.

The SELinux implementation of MAC employs a type enforcement mechanism that requires every subject and object to be assigned a type identifier. The terms subject and object are defined in the Bell-La Padula multilevel security model (see http://en.wikipedia.org/wiki/Bell-La_Padula_model for more information). Think of the subject as a user or a process and the object as a file or a process. Typically, a subject accesses an object; for example, a user modifies a file. When SELinux runs in enforcing mode, a subject cannot access an object unless the type identifier assigned to the subject is authorized to access the object. The default policy is to deny all access not specifically allowed. Authorization is determined by rules defined in the SELinux policy. An example of a rule granting access may be as simple as:

```
allow httpd_t httpd_sys_content_t : file {ioctol read getattr lock};
```

In this rule, the subject **http** daemon, assigned the type identifier of **httpd_t**, is given the permissions `ioctol`, `read`, `getattr`, and `lock` for any file object assigned the type identifier **httpd_sys_content_t**. In simple terms, the **http** daemon is allowed to read a file that is assigned the type identifier **httpd_sys_content_t**. This is a basic example of an allow rule type. There are many types of allow rules and some are very complex. There are also many type identifiers for use with subjects and objects. For more information about rule definitions, see: SELinux by Example in the “Related information and downloads,” on page 33 section.

SELinux adds type enforcement to standard Linux distributions. To access an object, the user must have both the appropriate file permissions (DAC) and the correct SELinux access. An SELinux security context contains three parts: the user, the role, and the type identifier. Running the **ls** command with the –Z switch displays the typical file information as well as the security context for each item in the subdirectory. In the following example, the security context for the **index.html** file is composed of `user_u` as the user, `object_r` as the role, and `httpd_sys_content_t` as the type identifier

```
[web_admin@localhost html]$ ls -Z index.html
-rw-r--r--  web_admin web_admin user_u:object_r:httpd_sys_content_t index.html
```

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

# Chapter 3. SELinux and Apache

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## Installing and running HTTPD

Now that you have a general understanding of the SELinux security context, you can secure an Apache Web server using SELinux. To follow along, you must have Apache installed on your system.

You can install Apache on Red Hat Linux by entering the following command:

```
[root@localhost html]$  yum install httpd
```

Next, start the Apache **http** daemon by entering `service httpd start`, as follows:

```
[root@localhost html]$ service httpd start
Starting httpd:
```

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## HTTPD and context types

Red Hat Enterprise Linux 5.3, at the time of this writing, uses **selinux-policy-2.4.6-203.el5**. This policy defines the security context for the **http** daemon object as **httpd_t**.

Because SELinux is running in `enforcing` mode, entering **/bin/ps axZ | grep httpd** produces the following output:

```
[root@localhost html]$ ps axZ | grep http
rootroot:system_r:httpd_t    2555 ?    Ss  0:00 /usr/sbin/httpd
rootroot:system_r:httpd_t    2593 ?    S   0:00 /usr/sbin/httpd
rootroot:system_r:httpd_t    2594 ?    S   0:00 /usr/sbin/httpd
root:system_r:httpd_t    2595 ?    S   0:00 /usr/sbin/httpd
root:system_r:httpd_t    2596 ?    S   0:00 /usr/sbin/httpd
root:system_r:httpd_t    2597 ?    S   0:00 /usr/sbin/httpd
root:system_r:httpd_t    2598 ?    S   0:00 /usr/sbin/httpd
root:system_r:httpd_t    2599 ?    S   0:00 /usr/sbin/httpd
root:system_r:httpd_t    2600 ?    S   0:00 /usr/sbin/httpd
```

The Z option to ps shows the security context for the **httpd** processes as `root:system_r:httpd_t`, confirming that **httpd** is running as the security type `httpd_t`.

The selinux-policy-2.4.6-203.el5 also defines several file security context types to be used with the **http** daemon. For a listing, see the man page for **httpd_selinux**. The `httpd_sys_content_t` context type is used for files and subdirectories containing content to be accessible by the **http** daemon and all **httpd** scripts. Entering `ls –Z` displays the security context for items in the default **http** directory (**/var/www/**), as follows:

```
[root@localhost ~]$ ls -Z /var/www/ | grep html

drwxr-xr-x  root      root system_u:object_r:httpd_sys_content_t html
```

The **/var/www/html** directory is the default location for all Web server content (defined by the variable setting of DocumentRoot `/var/www/html` in the **/etc/httpd/conf/httpd.conf http** configuration file). This directory is assigned the type `httpd_sys_content_t` as part of its security context which allows the **http** daemon to access its contents.

Any file or subdirectory inherits the security context of the directory in which it is created; therefore a file created in the html subdirectory inherits the `httpd_sys_content_t` type. In the following example, the root user creates the **index.html** file in the **/root** directory. The **index.html** inherits the security `root:object_r:user_home_t` context which is the expected security context for root in RHEL 5.3.

```
[root@localhost ~]$ touch /root/index.html
[root@localhost ~]$ ls -Z /root/index.html
-rw-r--r-- root root root:object_r:user_home_t /root/index.html
```

If the root user copies the newly created **index.html** file to the **/var/www/html/** directory, the file inherits the security context (`httpd_sys_content_t`) of the html subdirectory because a new copy of the file is created in the html subdirectory:

```
[root@localhost ~]$ cp /root/index.html /var/www/html
[root@localhost ~]$ ls -Z /var/www/html/index.html
-rw-r--r--  root root user_u:object_r:httpd_sys_content_t /var/www/html/index.html
```

If you move the **index.html** file instead of copying it, a new file is not created in the html subdirectory and **index.html** retains the `user_home_t` type:

```
[root@localhost ~]$  mv -f /root/index.html /var/www/html
[root@localhost ~]$ ls -Z /var/www/html/index.html
-rw-r--r--  root root user_u:object_r:user_home_t      /var/www/html/index.html
```

When a Web browser or network download agent like **wget** makes a request to the **http** daemon for the moved **index.html** file, with `user_home_t context`, the browser is denied access because SELinux is running in `enforcing` mode.

```
[root@localhost ~]# wget localhost/index.html
--21:10:00--  http://localhost/index.html
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 403 Forbidden
21:10:00 ERROR 403: Forbidden.
```

SELinux generates error messages in both **/var/log/messages** and **/var/log/httpd/error_log**. The following message in **/var/log/httpd/error_log** is not very helpful because it tells you only that access is being denied:

```
[Wed May 20 12:47:57 2009] [error] [client 172.16.1.100] (13)
Permission denied: access to /index.html denied
```

The following error message in **/var/log/messages** is more helpful because it tells you why SELinux is preventing access to the **/var/www/html/index.html** file - a potentially mislabeled file. Furthermore, it provides a command that you can use to produce a detailed summary of the issue.

```
May 20 12:22:48 localhost setroubleshoot: SELinux is preventing
the httpd from using potentially mislabeled files (/var/www/html/index.html).
For complete SELinux messages. run sealert -l 9e568d42-4b20-471c-9214-b98020c4d97a
```

Entering `sealert –l 9e568d42-4b20-471c-9214-b98020c4d97` as suggested in the previous error message returns the following detailed error message:

```
[root@localhost ~]$  sealert –l 9e568d42-4b20-471c-9214-b98020c4d97
Summary:
SELinux is preventing the httpd from using potentially mislabeled files (/var/www/html/index.html).
Detailed Description:
SELinux has denied httpd access to potentially mislabeled file(s) (/var/www/html/index.html).
This means that SELinux will not allow httpd to use these files. It is
common for users to edit files in their home directory or tmp directories and then
```

```
move (mv) them to system directories. The problem is that the files end up with the wrong
file context which confined applications are not allowed to access.
Allowing Access:
If you want httpd to access this files, you need to relabel them using restorecon -v
'/var/www/html/index.html'. You might want to relabel the entire directory using
restorecon -R -v '/var/www/html'.
Additional Information:
Source Context                root:system_r:httpd_t
Target Context                root:object_r:user_home_t
Target Objects                /var/www/html/index.html [ file ]
Source                        httpd
Source Path                   /usr/sbin/httpd
Port                          <Unknown>
Host                          localhost.localdomain
Source RPM Packages           httpd-2.2.3-22.el5
Target RPM Packages
Policy RPM                    selinux-policy-2.4.6-203.el5
Selinux Enabled               True
Policy Type                   targeted
MLS Enabled                   True
Enforcing Mode                Enforcing
Plugin Name                   home_tmp_bad_labels
Host Name                     localhost.localdomain
Platform                      Linux localhost.localdomain 2.6.18-128.1.10.el5 #1
                              SMP Wed Apr 29 13:55:17 EDT 2009 i686 i686
Alert Count                   24
First Seen                    Fri May 15 13:36:32 2009
Last Seen                     Wed May 20 12:47:56 2009
Local ID                      9e568d42-4b20-471c-9214-b98020c4d97a
Line Numbers
Raw Audit Messages
host=localhost.localdomain type=AVC msg=audit(1242838076.937:1141): avc: denied
{ getattr } for pid=3197 comm="httpd" path="/var/www/html/index.html" dev=dm-0
ino=3827354 scontext=root:system_r:httpd_t:s0 context=root:object_r:user_home_t:s0
tclass=file
host=localhost.localdomain type=SYSCALL msg=audit(1242838076.937:1141): arch=40000003
syscall=196 success=no exit=-13 a0=8eaa788 a1=bfc8d49c a2=419ff4 a3=2008171 items=0
ppid=3273 pid=3197 auid=500 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48
fsgid=48 tty=(none) ses=4 comm="httpd" exe="/usr/sbin/httpd"
subj=root:system_r:httpd_t:s0 key=(null)
```

Although called a summary, this output is a very detailed report that provides the necessary commands to resolve the issue. As shown below, entering `/sbin/restorecon -v '/var/www/html/index.html` as suggested not only resolves the problem, but also explains how you should change the security context for the **/var/www/html/index.html** file.

```
[root@localhost ~]$ restorecon -v '/var/www/html/index.html'
/sbin/restorecon reset /var/www/html/index.html context root:object_r:user_home_t:s0->
root:object_r:httpd_sys_content_t:s0
```

The previous **restorecon -v** command changed the security context of **/var/www/html/index.html** from `root:object_r:user_home_t` to `root:object_r:httpd_sys_content_t`. With a `root:object_r:httpd_sys_content_t` security context, the **http** daemon can now access **/var/www/html/index.html**.

Use a Web browser or **wget** to make another request to the **httpd** daemon for the **index.html** file with a restored security context. This time, the request is permitted:

```
[root@localhost ~]# wget localhost/index.html
--21:09:21--  http://localhost/index.html
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/html]
Saving to: 'index.html'
```

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

21:09:21 (0.00 B/s) - 'index.html' saved [0/0]
```

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## HTTPD and SELinux Booleans

SELinux has a set of built-in switches named Booleans or conditional policies that you can use to turn specific SELinux features on or off.

Entering the **getsebool -a | grep http** command lists the 23 Booleans related to the **http** daemon, which are a subset of the 234 Booleans currently defined in the **selinux-policy-2.4.6-203.el5** policy. These 23 Booleans allow you to customize SELinux policy for the **http** daemon during runtime without modifying, compiling, or loading a new policy. You can customize the level of **http** security by setting the relevant Boolean values or toggling between `on` and `off` values.

```
[root@localhost ~]$ getsebool -a | grep http
allow_httpd_anon_write --> off
allow_httpd_bugzilla_script_anon_write --> off
allow_httpd_mod_auth_pam --> off
allow_httpd_nagios_script_anon_write --> off
allow_httpd_prewikka_script_anon_write --> off
allow_httpd_squid_script_anon_write --> off
allow_httpd_sys_script_anon_write --> off
httpd_builtin_scripting --> on
httpd_can_network_connect --> off
httpd_can_network_connect_db --> off
httpd_can_network_relay --> off
httpd_can_sendmail --> on
httpd_disable_trans --> off
httpd_enable_cgi --> on
httpd_enable_ftp_server --> off
httpd_enable_homedirs --> on
httpd_rotatelogs_disable_trans --> off
httpd_ssi_exec --> off
httpd_suexec_disable_trans --> off
httpd_tty_comm --> on
httpd_unified --> on
httpd_use_cifs --> off
httpd_use_nfs --> off
```

SELinux provides three command-line tools for working with Booleans: **getsebool**, **setsebool**, and **togglesebool**. The **getsebool –a** command returns the current state of all the SELinux Booleans defined by the policy. You can also use the command without the **–a** option to return settings for one or more specific Booleans entered on the command line, as follows:

```
[root@localhost ~]$ getsebool httpd_enable_cgi
httpd_enable_cgi --> on
```

Use **setsebool** to set the current state of one or more Booleans by specifying the Boolean and its value. Acceptable values to enable a Boolean are `1`, `true`, and `on`. Acceptable values to disable a Boolean are `0`, `false`, and `off`. See the following cases for examples. You can use the **-P** option with the **setsebool** command to write the specified changes to the SELinux policy file. These changes are persistent across reboots; unwritten changes remain in effect until you change them or the system is rebooted.

Use **setsebool** to change status of the **httpd_enable_cgi** Boolean to `off`:

```
[root@localhost ~]$  setsebool httpd_enable_cgi  0
```

Confirm status change of the **httpd_enable_cgi** Boolean:

```
[root@localhost ~]$ getsebool httpd_enable_cgi
httpd_enable_cgi --> off
```

The **togglesebool** tool flips the current value of one or more Booleans. This tool does not have an option that writes the changes to the policy file. Changes remain in effect until changed or the system is rebooted.

Use the **togglesebool** tool to switch the status of the **httpd_enable_cgi** Boolean, as follows:

```
[root@localhost ~]$  togglesebool  httpd_enable_cgi
httpd_enable_cgi: active
```

Confirm the status change of the **httpd_enable_cgi** Boolean:

```
[root@localhost ~]$ getsebool httpd_enable_cgi
httpd_enable_cgi --> on
```

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

# Chapter 4. Configuring HTTPD security using SELinux

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## Securing Apache (static content only)

The default Red Hat Enterprise Linux 5.3 installation with SELinux running in `enforcing` mode provides a basic level of Web server security. You can increase that security level with a little effort.

Because security is related to the function of the system, let's start with a Web server that only serves static content from the **/var/www/html** directory.

1. Ensure that SELinux is enabled and running in `enforcing` mode:

```
[root@localhost ~]$ sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 21
Policy from config file:
```

2. Confirm that **httpd** is running as type `httpd_t`:

```
[root@localhost html]$ /bin/ps axZ | grep http
root:system_r:httpd_t    2555 ?    Ss  0:00 httpd
root:system_r:httpd_t    2593 ?    S   0:00 httpd
root:system_r:httpd_t    2594 ?    S   0:00 httpd
root:system_r:httpd_t    2595 ?    S   0:00 httpd
root:system_r:httpd_t    2596 ?    S   0:00 httpd
root:system_r:httpd_t    2597 ?    S   0:00 httpd
root:system_r:httpd_t    2598 ?    S   0:00 httpd
root:system_r:httpd_t    2599 ?    S   0:00 httpd
root:system_r:httpd_t    2600 ?    S   0:00 httpd
```

3. Confirm that the **/var/www/html** directory is assigned the `httpd_sys_content_t` context type:

```
[root@localhost ~]$ ls -Z /var/www/
drwxr-xr-x  root      root root:object_r:httpd_sys_script_exec_t cgi-bin
drwxr-xr-x  root      root root:object_r:httpd_sys_content_t error
drwxr-xr-x  root      root root:object_r:httpd_sys_content_t html
drwxr-xr-x  root      root root:object_r:httpd_sys_content_t icons
drwxr-xr-x  root      root root:object_r:httpd_sys_content_t manual
drwxr-xr-x  webalizer root root:object_r:httpd_sys_content_t usage
```

4. Confirm that the content to be served is assigned the `httpd_sys_content_t` context type. For example:

```
[root@localhost ~]$ ls -Z /var/www/html/index.html
-rw-r--r--  root root root:object_r:httpd_sys_content_t /var/www/html/index.html
```

Use a Web browser or **wget** to make a request to the **httpd** daemon for the **index.html** file and you should see that permission is granted.

To increase the level of protection provided by SELinux, disable any httpd-related features that you do not want by turning off their corresponding Boolean. By default, the following six Boolean are set to `on`. If you do not need these features, turn them off by setting their Boolean variables to `off`.

```
[root@glocalhost ~]# getsebool -a|grep http|grep "\-\-> on"
httpd_builtin_scripting --> on
httpd_can_sendmail --> on
```

```
httpd_enable_cgi --> on
httpd_enable_homedirs --> on
httpd_tty_comm --> on
httpd_unified --> on
```

**httpd_can_sendmail**
If the Web server does not use Sendmail, turn this Boolean to `off`. This action prevents unauthorized users from sending e-mail spam from this system.

**httpd_enable_homedirs**
When this Boolean is set to `on`, it allows **httpd** to read content from subdirectories located under user home directories. If the Web server is not configured to serve content from user home directories, set this Boolean to off.

**httpd_tty_comm**
By default, **httpd** is allowed to access the controlling terminal. This action is necessary in certain situations where **httpd** must prompt the user for a password. If the Web server does not require this feature, set the Boolean to `off`.

**httpd_unified**
This Boolean affects the transition of the **http** daemon to security domains defined in SELinux policy. Enabling this Boolean creates a single security domain for all http-labeled content. For more information, see SELinux by Example listed under the "Related information and downloads," on page 33 section.

**httpd_enable_cgi**
If your content does not use the Common Gateway Interface (CGI) protocol, set this Boolean to `off`. If you are unsure about using CGI in the Web server, try setting it to `off` and examine the log entries in the **/var/log/messages** file. The following example shows an error message from **/var/log/messages** resulting from SELinux blocking **httpd** execution of a CGI script:

```
May 28 15:48:37 localhost setroubleshoot: SELinux is preventing the
http daemon from executing cgi scripts. For complete SELinux
messages. run sealert -l 0fdf4649-60df-47b5-bfd5-a72772207adc
```

Entering `sealert -l 0fdf4649-60df-47b5-bfd5-a72772207adc` produces the following output:

```
Summary:
SELinux is preventing the http daemon from executing cgi scripts.
Detailed Description:
SELinux has denied the http daemon from executing a cgi script. httpd can be setup in
a locked down mode where cgi scripts are not allowed to be executed. If the httpd
server has been setup to not execute cgi scripts, this could signal a intrusion
attempt.
Allowing Access:
If you want httpd to be able to run cgi scripts, you need to turn on the
httpd_enable_cgi Boolean: "setsebool -P httpd_enable_cgi=1"
The following command will allow this access:
setsebool -P httpd_enable_cgi=1
Additional Information:
Source Context                root:system_r:httpd_t
Target Context                root:object_r:httpd_sys_script_exec_t
Target Objects                /var/www/cgi-bin [ dir ]
Source                        httpd
Source Path                   httpd
Port                          <Unknown>
Host                          localhost.localdomain
Source RPM Packages           httpd-2.2.3-22.el5
Target RPM Packages           httpd-2.2.3-22.el5
Policy RPM                    selinux-policy-2.4.6-203.el5
Selinux Enabled               True
Policy Type                   targeted
MLS Enabled                   True
Enforcing Mode                Enforcing
Plugin Name                   httpd_enable_cgi
Host Name                     localhost.localdomain
```

```
Platform                      Linux localhost.localdomain 2.6.18-128.1.10.el5 #1
                              SMP Wed Apr 29 13:55:17 EDT 2009 i686 i686
Alert Count                   1
First Seen                    Thu May 28 15:48:36 2009
Last Seen                     Thu May 28 15:48:36 2009
Local ID                      0fdf4649-60df-47b5-bfd5-a72772207adc
Line Numbers
Raw Audit Messages
host=localhost.localdomain type=AVC msg=audit(1243540116.963:248): avc:  denied
{ getattr } for  pid=2595 comm="httpd" path="/var/www/cgi-bin" dev=dm-0 ino=5527166
scontext=root:system_r:httpd_t:s0 tcontext=root:object_r:httpd_sys_script_exec_t:s0
tclass=dir
host=localhost.localdomain type=SYSCALL msg=audit(1243540116.963:248): arch=40000003
syscall=196 success=no exit=-13 a0=8bd0a88 a1=bfc790bc a2=4d0ff4 a3=2008171 items=0
ppid=2555 pid=2595 auid=4294967295 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48
sgid=48 fsgid=48 tty=(none) ses=4294967295 comm="httpd" exe="httpd"
subj=root:system_r:httpd_t:s0 key=(null)
```

At the end of the previous output, listed under the Raw Audit Messages are these lines:

```
"scontext=root:system_r:httpd_t:s0 tcontext=root:object_r:httpd_sys_script_exec_t:s0 tclass=dir"
```

This output shows you that **httpd** attempted to access a subdirectory with the `httpd_sys_script_exec_t` context type. This type is the context type of **/var/www/cgi-bin**, the directory where **httpd** looks for CGI scripts. The **httpd** daemon, with a `httpd_t` context type, was unable to access this subdirectory because the `httpd_enable_cgi` variable is set to off. With this configuration, SELinux does not allow a user or process of type `httpd_t` to access a directory, file, or process of type `httpd_sys_script_exec_t`. Therefore, the **http** daemon was denied access to the CGI script located in **/var/www/cgi-bin**. If you find similar messages in your log file, set the `httpd_enable_cgi` Boolean to `on`.

**httpd_builtin_scripting**

If you did not configure Apache to load scripting modules by changing the **/etc/httpd/conf/httpd.conf** configuration file, set this Boolean to `off`. If you are unsure, turn `httpd_builtin_scripting` to `off` and check the **/var/log/messages** file for any httpd-related SELinux warnings. See the description of `httpd_enable_cgi` for an example. PHP and other scripting modules run with the same level of access as the **http** daemon. Therefore, turning `httpd_builtin_scripting` to `off` reduces the amount of access available if the Web server is compromised.

To turn off all six of these Booleans and write the values to the policy file by using the **setsebool -P** command follow these steps:

1. Enter the **setsebool -P** command:

   ```
   [root@localhost ~]# setsebool -P httpd_can_sendmail=0
   httpd_enable_homedirs=0 httpd_tty_comm=0 httpd_unified=0
   httpd_enable_cgi=0 httpd_builtin_scripting=0
   ```

2. Check all the Boolean settings related to **httpd** by entering `getsebool –a | grep httpd`. The following output shows that all Boolean are set to `off`, including the six previously described variables which default to `on`.

   ```
   [root@localhost ~]$ getsebool -a | grep httpd
   allow_httpd_anon_write --> off
   allow_httpd_bugzilla_script_anon_write --> off
   allow_httpd_mod_auth_pam --> off
   allow_httpd_nagios_script_anon_write --> off
   allow_httpd_prewikka_script_anon_write --> off
   allow_httpd_squid_script_anon_write --> off
   allow_httpd_sys_script_anon_write --> off
   httpd_builtin_scripting --> off
   httpd_can_network_connect --> off
   httpd_can_network_connect_db --> off
   httpd_can_network_relay --> off
   httpd_can_sendmail --> off
   ```

```
httpd_disable_trans --> off
httpd_enable_cgi --> off
httpd_enable_ftp_server --> off
httpd_enable_homedirs --> off
httpd_rotatelogs_disable_trans --> off
httpd_ssi_exec --> off
httpd_suexec_disable_trans --> off
httpd_tty_comm --> off
httpd_unified --> off
httpd_use_cifs --> off
httpd_use_nfs --> off
```

3. Use a Web browser or **wget** to make another request to the **httpd** daemon for the **index.html** file and you should succeed. Rebooting your machine does not change this configuration.

This completes the necessary basic SELinux settings for hardening a Web server with static content. Next, look at hardening scripts accessed by the **http** daemon.

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## Hardening CGI scripts with SELinux

In the previous section, you used SELinux Booleans to disable scripting because the Web server used only static content. Beginning with that configuration, you can enable CGI scripting and use SELinux to secure the scripts.

1. Confirm that your Web server is configured as described in section “Securing Apache (static content only)” on page 13.
2. Red Hat Enterprise Linux provides a CGI script that you can use for testing. You can find the script at **/usr/lib/perl5/5.8.8/CGI/eg/tryit.cgi.** Copy this script to the **/var/www/cgi-bin/** directory, as follows:

   ```
   [root@localhost ~]$ cp /usr/lib/perl5/5.8.8/CGI/eg/tryit.cgi /var/www/cgi-bin/
   ```

3. Make sure that the first line of the **tryit.cgi** script contains the correct path to the perl binary. From the `which perl` output shown below, the path should be changed to **!#/usr/bin/perl**.

   ```
   [root@localhost ~]# which perl
   /usr/bin/perl
   [root@localhost ~]# head -1 /var/www/cgi-bin/tryit.cgi
   #!/usr/local/bin/perl
   ```

4. Confirm that **/var/www/cgi-bin** is assigned the `httpd_sys_script_exec_t` context type as follows:

   ```
   [root@localhost ~]$ ls -Z /var/www/ | grep cgi-bin
   drwxr-xr-x  root      root root:object_r:httpd_sys_script_exec_t cgi-bin
   ```

5. Allow and confirm read and execute permission for the **tryit.cgi** script to all users:

   ```
   [root@localhost cgi-bin]# chmod 555 /var/www/cgi-bin/tryit.cgi
   [root@localhost cgi-bin]# ls -Z
   -r-xr-xr-x  root root root:object_r:httpd_sys_script_exec_t tryit.cgi
   ```

6. Confirm that **/var/www/cgi-bin/tryit.cgi** is assigned the `httpd_sys_script_exec_t` context type:

   ```
   [root@localhost ~]$ ls -Z /var/www/cgi-bin/tryit.cgi
   -r-xr-xr-x  root root root:object_r:httpd_sys_script_exec_t
   /var/www/cgi-bin/tryit.cgi
   ```

7. Enable CGI scripting in SELinux and confirm that it is enabled:

   ```
   [root@localhost cgi-bin]$  setsebool   httpd_enable_cgi=1
   [root@localhost cgi-bin]$ getsebool httpd_enable_cgi
   httpd_enable_cgi --> on
   ```

8. Open a Web browser and type the Web server address into the location bar. Include the **/cgi-bin/tryit.cgi** in the URL. For example, type `http://192.168.1.100/cgi-bin/tryit.cgi`. The **tryit.cgi** script should return output similar to Figure 1:

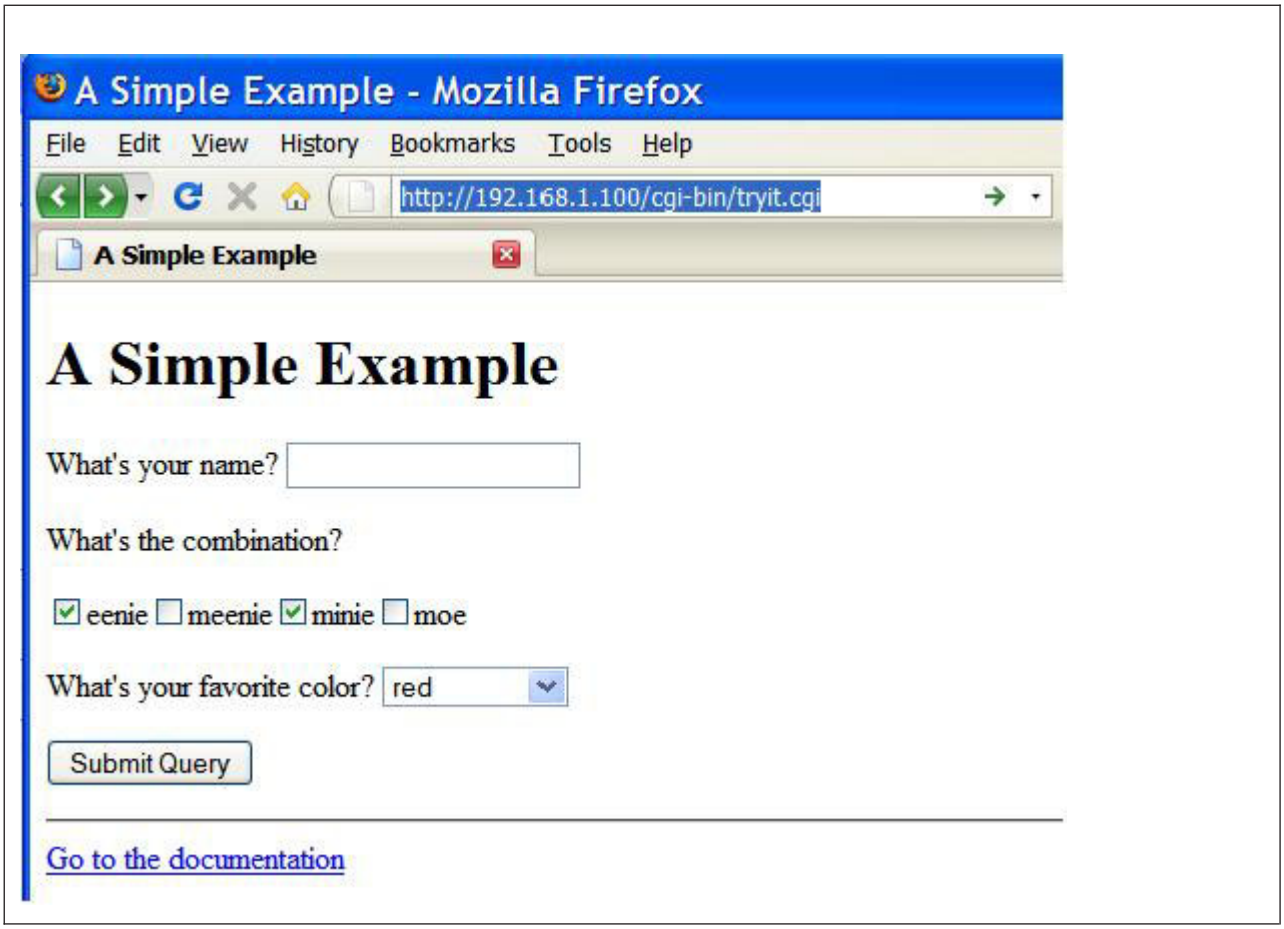


*Figure 1. Figure 1: A Simple Example*

9. Provide test answers to the form fields and click **Submit Query**. The **tryit.cgi** script should return output similar to Figure 2:

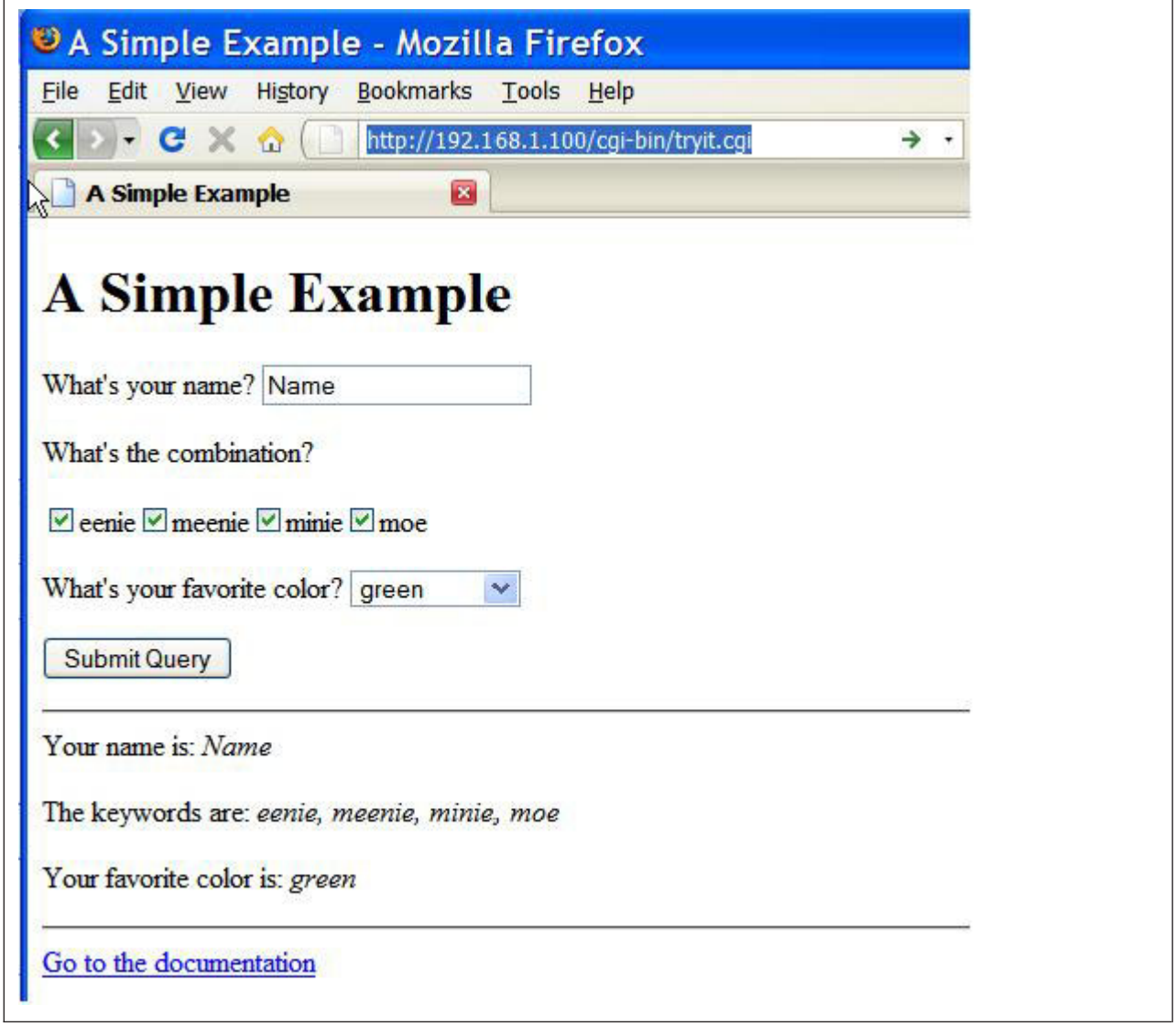


*Figure 2. Figure 2: A Simple Example with results*

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

# Chapter 5. First Steps with Security-Enhanced Linux (SELinux): Hardening the Apache Web Server

## Scope, requirements, and support

This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

### Systems to which this information applies

System x running Linux and PowerLinux

## Security-Enhanced Linux overview

Security-Enhanced Linux (SELinux) is a component of the Linux operating system developed primarily by the United States National Security Agency. SELinux provides a method for creation and enforcement of mandatory access control (MAC) policies. These policies confine users and processes to the minimal amount of privilege required to perform assigned tasks.

For more information about the history of SELinux, see http://en.wikipedia.org/wiki/Selinux.

Since its release to the open source community in December 2000, the SELinux project has gained improvements such as predefined Boolean variables that make it easier to use. This paper helps you understand how to use these variables to configure SELinux policies on your system and to secure the Apache **httpd** daemon.

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## Access control: MAC and DAC

Access level is important to computer system security. To compromise a system, attackers try to gain any possible level of access and then try to escalate that level until they are able to obtain restricted data or make unapproved system modifications. Because each user has some level of system access, every user account on your system increases the potential for abuse. System security has historically relied on trusting users not to abuse their access, but this trust has proven to be problematic. Today, server consolidation leads to more users per system. Outsourcing of Systems Management gives legitimate access, often at the system administrator level, to unknown users. Because server consolidation and outsourcing can be financially advantageous, what can you do to prevent abuse on Linux systems? To begin to answer that question, let's take a look at discretionary access control (DAC) and mandatory access control (MAC) and their differences.

Discretionary access control (DAC), commonly known as *file permissions*, is the predominant access control mechanism in traditional UNIX and Linux systems. You may recognize the **drwxr-xr-x** or the **ugo** abbreviations for owner, group, and other permissions seen in a directory listing. In DAC, generally the resource owner (a user) controls who has access to a resource. For convenience, some users commonly set dangerous DAC file permissions that allow every user on the system to read, write, and execute many files that they own. In addition, a process started by a user can modify or delete any file to which the user has access. Processes that elevate their privileges high enough could therefore modify or delete system files. These instances are some of the disadvantages of DAC.

In contrast to DAC, mandatory access control (MAC) regulates user and process access to resources based upon an organizational (higher-level) security policy. This policy is a collection of rules that specify what types of access are allowed on a system. System policy is related to MAC in the same way that firewall rules are related to firewalls.

SELinux is a Linux kernel implementation of a flexible MAC mechanism called type enforcement. In type enforcement, a type identifier is assigned to every user and object. An object can be a file or a process. To access an object, a user must be authorized for that object type. These authorizations are defined in a SELinux policy. Let's work through some examples and you will develop a better understanding of MAC and how it relates to SELinux.

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## SELinux basics

It is a good practice not to use the root user unless necessary. However for demonstrating how to use SELinux, the root user is used in the examples in this blueprint. Some of the commands shown require root privileges to run them; for example, running **getenforce** and editing the **/etc/selinux/config** file.

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

### Run modes

You can enable or disable SELinux policy enforcement on a Red Hat Enterprise Linux system during or after operating system installation.

When disabled, SELinux has no effect on the system. When enabled, SELinux runs in one of two modes:

- **Enforcing**: SELinux is enabled and SELinux policy is enforced
- **Permissive**: SELinux is enabled but it only logs warnings instead of enforcing the policy

When prompted during operating system installation, if you choose to enable SELinux, it is installed with a default security policy and set to run in the enforcing mode.

Confirm the status of SELinux on your system. Like in many UNIX or Linux operating systems, there is more than one way to perform a task. To check the current mode, run one of the following commands: **getenforce**, **sestatus**, or **cat /etc/selinux/config**.

- The **getenorce** command returns the current SELinux run mode, or `Disabled` if SELinux is not enabled. In the following example, **getenforce** shows that SELinux is enabled and enforcing the current SELinux policy:

  ```
  [root@localhost ~]$ getenforce
  Enforcing
  ```

  If your system is displaying `Permissive` or `Disabled` and you want to follow along with the instructions, change the **/etc/selinux/config** file to run in `Enforcing` mode before continuing with the demonstration. Remember that if you are in `Disabled` mode, you should change first to `Permissive` and then to `Enforcing`.
- The **setstatus** command returns the current run mode, along with information about the SELinux policy if SELinux is enabled. In the following example, **setstatus** shows that SELinux is enabled and enforcing the current SELinux policy:

  ```
  [root@localhost ~]$ sestatus
  SELinux status:                 enabled
  SELinuxfs mount:                /selinux
  ```

```
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 21
Policy from config file:        targeted
```

- The **/etc/selinux/config** file configures SELinux and controls the mode as well as the active policy. Changes to the **/etc/selinux/config** file become effective only after you reboot the system. In the following example, the file shows that the mode is set to `enforcing` and the current policy type is `targeted`.

```
[root@localhost ~]$ cat /etc/selinux/config
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - SELinux is fully disabled.
SELINUX=enforcing
# SELINUXTYPE= type of policy in use. Possible values are:
#       targeted - Only targeted network daemons are protected.
#       strict - Full SELinux protection.
SELINUXTYPE=targeted
```

To enable SELinux, you need to set the value of the **SELINUX** parameter in the **/etc/selinux/config** file to either `enforcing` or `permissive`. If you enable SELinux in the config file, you must reboot your system to start SELinux. We recommend that you set `SELINUX=permissive` if the file system has never been labeled, has not been labeled recently, or you are not sure when it was last labeled. Note that file system labeling is the process of assigning a label containing security-relevant information to each file. In SELinux a file label is composed of the user, role, and type such as `system_u:object_r:httpd_sys_content_t`. `Permissive` mode ensures that SELinux does not interfere with the boot sequence if a command in the sequence occurs before the file system relabel is completed. Once the system is up and running, you can change the SELinux mode to `enforcing`.

If you want to change the mode of SELinux on a running system, use the **setenforce** command. Entering `setenforce enforcing` changes the mode to `enforcing` and `setenforce permissive` changes the mode to `permissive`. To disable SELinux, edit the **/etc/selinux/config** file as described previously and reboot. You cannot disable or enable SELinux on a running system from the command line; you can only switch between `enforcing` and `permissive` when SELinux is enabled.

Change the mode of SELinux to `permissive` by entering the following command:

```
[root@localhost ~]$  setenforce permissive
```

Recheck the output from **getenforce**, **sestatus**, and **cat /etc/selinux/config**.

- The **getenforce** command returns `Permissive`, confirming the mode change:

```
[root@localhost ~]$ getenforce
Permissive
```

- The **sestatus** command also returns a `Permissive` mode value:

```
[root@localhost ~]$sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   permissive
Mode from config file:          enforcing
Policy version:                 21
Policy from config file:        targeted
```

- After changing the mode to `permissive`, both the **getenforce** and **sestatus** commands return the correct `permissive` mode. However, look carefully at the output from the **sestatus** command:

```
[root@localhost ~]$ cat /etc/selinux/config
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
```

```
#       disabled - SELinux is fully disabled.
SELINUX=enforcing
# SELINUXTYPE= type of policy in use. Possible values are:
#       targeted - Only targeted network daemons are protected.
#       strict - Full SELinux protection.
SELINUXTYPE=targeted
[root@localhost ~]$
```

The **Mode from config file** parameter is `enforcing`. This setting is consistent with the **cat /etc/selinux/config** output because the config file was not changed. This status implies that the changes made by the **setenforce** command does not carry over to the next boot. If you reboot, SELinux returns to run state as configured in **/etc/selinux/conf** in `enforcing` mode.

Change the running mode back to `enforcing` by entering the following command:

```
[root@localhost ~]$  setenforce enforcing
```

The following output confirms the mode change:

```
[root@localhost ~]$ getenforce
Enforcing
```

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## Security contexts

The concept of type enforcement and the SELinux type identifier were discussed in the Overview. Let's explore these concepts in more detail.

The SELinux implementation of MAC employs a type enforcement mechanism that requires every subject and object to be assigned a type identifier. The terms subject and object are defined in the Bell-La Padula multilevel security model (see http://en.wikipedia.org/wiki/Bell-La_Padula_model for more information). Think of the subject as a user or a process and the object as a file or a process. Typically, a subject accesses an object; for example, a user modifies a file. When SELinux runs in enforcing mode, a subject cannot access an object unless the type identifier assigned to the subject is authorized to access the object. The default policy is to deny all access not specifically allowed. Authorization is determined by rules defined in the SELinux policy. An example of a rule granting access may be as simple as:

```
allow httpd_t httpd_sys_content_t : file {ioctol read getattr lock};
```

In this rule, the subject **http** daemon, assigned the type identifier of **httpd_t**, is given the permissions `ioctol`, `read`, `getattr`, and `lock` for any file object assigned the type identifier **httpd_sys_content_t**. In simple terms, the **http** daemon is allowed to read a file that is assigned the type identifier **httpd_sys_content_t**. This is a basic example of an allow rule type. There are many types of allow rules and some are very complex. There are also many type identifiers for use with subjects and objects. For more information about rule definitions, see: SELinux by Example in the “Related information and downloads,” on page 33 section.

SELinux adds type enforcement to standard Linux distributions. To access an object, the user must have both the appropriate file permissions (DAC) and the correct SELinux access. An SELinux security context contains three parts: the user, the role, and the type identifier. Running the **ls** command with the –Z switch displays the typical file information as well as the security context for each item in the subdirectory. In the following example, the security context for the **index.html** file is composed of `user_u` as the user, `object_r` as the role, and `httpd_sys_content_t` as the type identifier

```
[web_admin@localhost html]$ ls -Z index.html
-rw-r--r--  web_admin web_admin user_u:object_r:httpd_sys_content_t index.html
```

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

---

## SELinux and Apache

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## Installing and running HTTPD

Now that you have a general understanding of the SELinux security context, you can secure an Apache Web server using SELinux. To follow along, you must have Apache installed on your system.

You can install Apache on Red Hat Linux by entering the following command:

```
[root@localhost html]$  yum install httpd
```

Next, start the Apache **http** daemon by entering `service httpd start`, as follows:

```
[root@localhost html]$ service httpd start
Starting httpd:
```

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## HTTPD and context types

Red Hat Enterprise Linux 5.3, at the time of this writing, uses **selinux-policy-2.4.6-203.el5**. This policy defines the security context for the **http** daemon object as **httpd_t**.

Because SELinux is running in `enforcing` mode, entering **/bin/ps axZ | grep httpd** produces the following output:

```
[root@localhost html]$ ps axZ | grep http
rootroot:system_r:httpd_t    2555 ?    Ss  0:00 /usr/sbin/httpd
rootroot:system_r:httpd_t    2593 ?    S   0:00 /usr/sbin/httpd
rootroot:system_r:httpd_t    2594 ?    S   0:00 /usr/sbin/httpd
root:system_r:httpd_t    2595 ?    S   0:00 /usr/sbin/httpd
root:system_r:httpd_t    2596 ?    S   0:00 /usr/sbin/httpd
root:system_r:httpd_t    2597 ?    S   0:00 /usr/sbin/httpd
root:system_r:httpd_t    2598 ?    S   0:00 /usr/sbin/httpd
root:system_r:httpd_t    2599 ?    S   0:00 /usr/sbin/httpd
root:system_r:httpd_t    2600 ?    S   0:00 /usr/sbin/httpd
```

The Z option to ps shows the security context for the **httpd** processes as `root:system_r:httpd_t`, confirming that **httpd** is running as the security type `httpd_t`.

The selinux-policy-2.4.6-203.el5 also defines several file security context types to be used with the **http** daemon. For a listing, see the man page for **httpd_selinux**. The `httpd_sys_content_t` context type is used for files and subdirectories containing content to be accessible by the **http** daemon and all **httpd** scripts. Entering `ls –Z` displays the security context for items in the default **http** directory (**/var/www/**), as follows:

```
[root@localhost ~]$ ls -Z /var/www/ | grep html

drwxr-xr-x  root      root system_u:object_r:httpd_sys_content_t html
```

The **/var/www/html** directory is the default location for all Web server content (defined by the variable setting of DocumentRoot `/var/www/html` in the **/etc/httpd/conf/httpd.conf http** configuration file). This directory is assigned the type `httpd_sys_content_t` as part of its security context which allows the **http** daemon to access its contents.

Any file or subdirectory inherits the security context of the directory in which it is created; therefore a file created in the html subdirectory inherits the `httpd_sys_content_t` type. In the following example, the root user creates the **index.html** file in the **/root** directory. The **index.html** inherits the security `root:object_r:user_home_t` context which is the expected security context for root in RHEL 5.3.

```
[root@localhost ~]$ touch /root/index.html
[root@localhost ~]$ ls -Z /root/index.html
-rw-r--r-- root root root:object_r:user_home_t /root/index.html
```

If the root user copies the newly created **index.html** file to the **/var/www/html/** directory, the file inherits the security context (`httpd_sys_content_t`) of the html subdirectory because a new copy of the file is created in the html subdirectory:

```
[root@localhost ~]$ cp /root/index.html /var/www/html
[root@localhost ~]$ ls -Z /var/www/html/index.html
-rw-r--r--  root root user_u:object_r:httpd_sys_content_t /var/www/html/index.html
```

If you move the **index.html** file instead of copying it, a new file is not created in the html subdirectory and **index.html** retains the `user_home_t` type:

```
[root@localhost ~]$  mv -f /root/index.html /var/www/html
[root@localhost ~]$ ls -Z /var/www/html/index.html
-rw-r--r--  root root user_u:object_r:user_home_t      /var/www/html/index.html
```

When a Web browser or network download agent like **wget** makes a request to the **http** daemon for the moved **index.html** file, with `user_home_t context`, the browser is denied access because SELinux is running in `enforcing` mode.

```
[root@localhost ~]# wget localhost/index.html
--21:10:00--  http://localhost/index.html
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 403 Forbidden
21:10:00 ERROR 403: Forbidden.
```

SELinux generates error messages in both **/var/log/messages** and **/var/log/httpd/error_log**. The following message in **/var/log/httpd/error_log** is not very helpful because it tells you only that access is being denied:

```
[Wed May 20 12:47:57 2009] [error] [client 172.16.1.100] (13)
Permission denied: access to /index.html denied
```

The following error message in **/var/log/messages** is more helpful because it tells you why SELinux is preventing access to the **/var/www/html/index.html** file - a potentially mislabeled file. Furthermore, it provides a command that you can use to produce a detailed summary of the issue.

```
May 20 12:22:48 localhost setroubleshoot: SELinux is preventing
the httpd from using potentially mislabeled files (/var/www/html/index.html).
For complete SELinux messages. run sealert -l 9e568d42-4b20-471c-9214-b98020c4d97a
```

Entering `sealert –l 9e568d42-4b20-471c-9214-b98020c4d97` as suggested in the previous error message returns the following detailed error message:

```
[root@localhost ~]$  sealert –l 9e568d42-4b20-471c-9214-b98020c4d97
Summary:
SELinux is preventing the httpd from using potentially mislabeled files (/var/www/html/index.html).
Detailed Description:
SELinux has denied httpd access to potentially mislabeled file(s) (/var/www/html/index.html).
This means that SELinux will not allow httpd to use these files. It is
common for users to edit files in their home directory or tmp directories and then
```

```
move (mv) them to system directories. The problem is that the files end up with the wrong
file context which confined applications are not allowed to access.
Allowing Access:
If you want httpd to access this files, you need to relabel them using restorecon -v
'/var/www/html/index.html'. You might want to relabel the entire directory using
restorecon -R -v '/var/www/html'.
Additional Information:
Source Context                root:system_r:httpd_t
Target Context                root:object_r:user_home_t
Target Objects                /var/www/html/index.html [ file ]
Source                        httpd
Source Path                   /usr/sbin/httpd
Port                          <Unknown>
Host                          localhost.localdomain
Source RPM Packages           httpd-2.2.3-22.el5
Target RPM Packages
Policy RPM                    selinux-policy-2.4.6-203.el5
Selinux Enabled               True
Policy Type                   targeted
MLS Enabled                   True
Enforcing Mode                Enforcing
Plugin Name                   home_tmp_bad_labels
Host Name                     localhost.localdomain
Platform                      Linux localhost.localdomain 2.6.18-128.1.10.el5 #1
                              SMP Wed Apr 29 13:55:17 EDT 2009 i686 i686
Alert Count                   24
First Seen                    Fri May 15 13:36:32 2009
Last Seen                     Wed May 20 12:47:56 2009
Local ID                      9e568d42-4b20-471c-9214-b98020c4d97a
Line Numbers
Raw Audit Messages
host=localhost.localdomain type=AVC msg=audit(1242838076.937:1141): avc: denied
{ getattr } for pid=3197 comm="httpd" path="/var/www/html/index.html" dev=dm-0
ino=3827354 scontext=root:system_r:httpd_t:s0 context=root:object_r:user_home_t:s0
tclass=file
host=localhost.localdomain type=SYSCALL msg=audit(1242838076.937:1141): arch=40000003
syscall=196 success=no exit=-13 a0=8eaa788 a1=bfc8d49c a2=419ff4 a3=2008171 items=0
ppid=3273 pid=3197 auid=500 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48
fsgid=48 tty=(none) ses=4 comm="httpd" exe="/usr/sbin/httpd"
subj=root:system_r:httpd_t:s0 key=(null)
```

Although called a summary, this output is a very detailed report that provides the necessary commands to resolve the issue. As shown below, entering `/sbin/restorecon -v '/var/www/html/index.html` as suggested not only resolves the problem, but also explains how you should change the security context for the **/var/www/html/index.html** file.

```
[root@localhost ~]$ restorecon -v '/var/www/html/index.html'
/sbin/restorecon reset /var/www/html/index.html context root:object_r:user_home_t:s0->
root:object_r:httpd_sys_content_t:s0
```

The previous **restorecon -v** command changed the security context of **/var/www/html/index.html** from `root:object_r:user_home_t` to `root:object_r:httpd_sys_content_t`. With a `root:object_r:httpd_sys_content_t` security context, the **http** daemon can now access **/var/www/html/index.html**.

Use a Web browser or **wget** to make another request to the **httpd** daemon for the **index.html** file with a restored security context. This time, the request is permitted:

```
[root@localhost ~]# wget localhost/index.html
--21:09:21--  http://localhost/index.html
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/html]
Saving to: 'index.html'
```

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

21:09:21 (0.00 B/s) - 'index.html' saved [0/0]
```

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## HTTPD and SELinux Booleans

SELinux has a set of built-in switches named Booleans or conditional policies that you can use to turn specific SELinux features on or off.

Entering the **getsebool -a | grep http** command lists the 23 Booleans related to the **http** daemon, which are a subset of the 234 Booleans currently defined in the **selinux-policy-2.4.6-203.el5** policy. These 23 Booleans allow you to customize SELinux policy for the **http** daemon during runtime without modifying, compiling, or loading a new policy. You can customize the level of **http** security by setting the relevant Boolean values or toggling between `on` and `off` values.

```
[root@localhost ~]$ getsebool -a | grep http
allow_httpd_anon_write --> off
allow_httpd_bugzilla_script_anon_write --> off
allow_httpd_mod_auth_pam --> off
allow_httpd_nagios_script_anon_write --> off
allow_httpd_prewikka_script_anon_write --> off
allow_httpd_squid_script_anon_write --> off
allow_httpd_sys_script_anon_write --> off
httpd_builtin_scripting --> on
httpd_can_network_connect --> off
httpd_can_network_connect_db --> off
httpd_can_network_relay --> off
httpd_can_sendmail --> on
httpd_disable_trans --> off
httpd_enable_cgi --> on
httpd_enable_ftp_server --> off
httpd_enable_homedirs --> on
httpd_rotatelogs_disable_trans --> off
httpd_ssi_exec --> off
httpd_suexec_disable_trans --> off
httpd_tty_comm --> on
httpd_unified --> on
httpd_use_cifs --> off
httpd_use_nfs --> off
```

SELinux provides three command-line tools for working with Booleans: **getsebool**, **setsebool**, and **togglesebool**. The **getsebool –a** command returns the current state of all the SELinux Booleans defined by the policy. You can also use the command without the **–a** option to return settings for one or more specific Booleans entered on the command line, as follows:

```
[root@localhost ~]$ getsebool httpd_enable_cgi
httpd_enable_cgi --> on
```

Use **setsebool** to set the current state of one or more Booleans by specifying the Boolean and its value. Acceptable values to enable a Boolean are `1`, `true`, and `on`. Acceptable values to disable a Boolean are `0`, `false`, and `off`. See the following cases for examples. You can use the **-P** option with the **setsebool** command to write the specified changes to the SELinux policy file. These changes are persistent across reboots; unwritten changes remain in effect until you change them or the system is rebooted.

Use **setsebool** to change status of the **httpd_enable_cgi** Boolean to `off`:

```
[root@localhost ~]$  setsebool httpd_enable_cgi  0
```

Confirm status change of the **httpd_enable_cgi** Boolean:

```
[root@localhost ~]$ getsebool httpd_enable_cgi
httpd_enable_cgi --> off
```

The **togglesebool** tool flips the current value of one or more Booleans. This tool does not have an option that writes the changes to the policy file. Changes remain in effect until changed or the system is rebooted.

Use the **togglesebool** tool to switch the status of the **httpd_enable_cgi** Boolean, as follows:

```
[root@localhost ~]$  togglesebool  httpd_enable_cgi
httpd_enable_cgi: active
```

Confirm the status change of the **httpd_enable_cgi** Boolean:

```
[root@localhost ~]$ getsebool httpd_enable_cgi
httpd_enable_cgi --> on
```

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## Configuring HTTPD security using SELinux

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## Securing Apache (static content only)

The default Red Hat Enterprise Linux 5.3 installation with SELinux running in `enforcing` mode provides a basic level of Web server security. You can increase that security level with a little effort.

Because security is related to the function of the system, let's start with a Web server that only serves static content from the **/var/www/html** directory.

1. Ensure that SELinux is enabled and running in `enforcing` mode:

   ```
   [root@localhost ~]$ sestatus
   SELinux status:                 enabled
   SELinuxfs mount:                /selinux
   Current mode:                   enforcing
   Mode from config file:          enforcing
   Policy version:                 21
   Policy from config file:
   ```

2. Confirm that **httpd** is running as type `httpd_t`:

   ```
   [root@localhost html]$ /bin/ps axZ | grep http
   root:system_r:httpd_t    2555 ?    Ss  0:00 httpd
   root:system_r:httpd_t    2593 ?    S   0:00 httpd
   root:system_r:httpd_t    2594 ?    S   0:00 httpd
   root:system_r:httpd_t    2595 ?    S   0:00 httpd
   root:system_r:httpd_t    2596 ?    S   0:00 httpd
   root:system_r:httpd_t    2597 ?    S   0:00 httpd
   root:system_r:httpd_t    2598 ?    S   0:00 httpd
   root:system_r:httpd_t    2599 ?    S   0:00 httpd
   root:system_r:httpd_t    2600 ?    S   0:00 httpd
   ```

3. Confirm that the **/var/www/html** directory is assigned the `httpd_sys_content_t` context type:

   ```
   [root@localhost ~]$ ls -Z /var/www/
   drwxr-xr-x  root      root root:object_r:httpd_sys_script_exec_t cgi-bin
   drwxr-xr-x  root      root root:object_r:httpd_sys_content_t error
   drwxr-xr-x  root      root root:object_r:httpd_sys_content_t html
   ```

```
drwxr-xr-x  root      root root:object_r:httpd_sys_content_t icons
drwxr-xr-x  root      root root:object_r:httpd_sys_content_t manual
drwxr-xr-x  webalizer root root:object_r:httpd_sys_content_t usage
```

4. Confirm that the content to be served is assigned the `httpd_sys_content_t` context type. For example:

```
[root@localhost ~]$ ls -Z /var/www/html/index.html
-rw-r--r--  root root root:object_r:httpd_sys_content_t /var/www/html/index.html
```

Use a Web browser or **wget** to make a request to the **httpd** daemon for the **index.html** file and you should see that permission is granted.

To increase the level of protection provided by SELinux, disable any httpd-related features that you do not want by turning off their corresponding Boolean. By default, the following six Boolean are set to `on`. If you do not need these features, turn them off by setting their Boolean variables to `off`.

```
[root@glocalhost ~]# getsebool -a|grep http|grep "\-\-> on"
httpd_builtin_scripting --> on
httpd_can_sendmail --> on
httpd_enable_cgi --> on
httpd_enable_homedirs --> on
httpd_tty_comm --> on
httpd_unified --> on
```

**httpd_can_sendmail**
: If the Web server does not use Sendmail, turn this Boolean to `off`. This action prevents unauthorized users from sending e-mail spam from this system.

**httpd_enable_homedirs**
: When this Boolean is set to `on`, it allows **httpd** to read content from subdirectories located under user home directories. If the Web server is not configured to serve content from user home directories, set this Boolean to off.

**httpd_tty_comm**
: By default, **httpd** is allowed to access the controlling terminal. This action is necessary in certain situations where **httpd** must prompt the user for a password. If the Web server does not require this feature, set the Boolean to `off`.

**httpd_unified**
: This Boolean affects the transition of the **http** daemon to security domains defined in SELinux policy. Enabling this Boolean creates a single security domain for all http-labeled content. For more information, see SELinux by Example listed under the “Related information and downloads,” on page 33 section.

**httpd_enable_cgi**
: If your content does not use the Common Gateway Interface (CGI) protocol, set this Boolean to `off`. If you are unsure about using CGI in the Web server, try setting it to `off` and examine the log entries in the **/var/log/messages** file. The following example shows an error message from **/var/log/messages** resulting from SELinux blocking **httpd** execution of a CGI script:

```
May 28 15:48:37 localhost setroubleshoot: SELinux is preventing the
http daemon from executing cgi scripts. For complete SELinux
messages. run sealert -l 0fdf4649-60df-47b5-bfd5-a72772207adc
```

Entering `sealert -l 0fdf4649-60df-47b5-bfd5-a72772207adc` produces the following output:

```
Summary:
SELinux is preventing the http daemon from executing cgi scripts.
Detailed Description:
SELinux has denied the http daemon from executing a cgi script. httpd can be setup in
a locked down mode where cgi scripts are not allowed to be executed. If the httpd
server has been setup to not execute cgi scripts, this could signal a intrusion
attempt.
Allowing Access:
If you want httpd to be able to run cgi scripts, you need to turn on the
httpd_enable_cgi Boolean: "setsebool -P httpd_enable_cgi=1"
```

```
The following command will allow this access:
setsebool -P httpd_enable_cgi=1
Additional Information:
Source Context                root:system_r:httpd_t
Target Context                root:object_r:httpd_sys_script_exec_t
Target Objects                /var/www/cgi-bin [ dir ]
Source                        httpd
Source Path                   httpd
Port                          <Unknown>
Host                          localhost.localdomain
Source RPM Packages           httpd-2.2.3-22.el5
Target RPM Packages           httpd-2.2.3-22.el5
Policy RPM                    selinux-policy-2.4.6-203.el5
Selinux Enabled               True
Policy Type                   targeted
MLS Enabled                   True
Enforcing Mode                Enforcing
Plugin Name                   httpd_enable_cgi
Host Name                     localhost.localdomain
Platform                      Linux localhost.localdomain 2.6.18-128.1.10.el5 #1
                              SMP Wed Apr 29 13:55:17 EDT 2009 i686 i686
Alert Count                   1
First Seen                    Thu May 28 15:48:36 2009
Last Seen                     Thu May 28 15:48:36 2009
Local ID                      0fdf4649-60df-47b5-bfd5-a72772207adc
Line Numbers
Raw Audit Messages
host=localhost.localdomain type=AVC msg=audit(1243540116.963:248): avc:  denied
{ getattr } for  pid=2595 comm="httpd" path="/var/www/cgi-bin" dev=dm-0 ino=5527166
scontext=root:system_r:httpd_t:s0 tcontext=root:object_r:httpd_sys_script_exec_t:s0
tclass=dir
host=localhost.localdomain type=SYSCALL msg=audit(1243540116.963:248): arch=40000003
syscall=196 success=no exit=-13 a0=8bd0a88 a1=bfc790bc a2=4d0ff4 a3=2008171 items=0
ppid=2555 pid=2595 auid=4294967295 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48
sgid=48 fsgid=48 tty=(none) ses=4294967295 comm="httpd" exe="httpd"
subj=root:system_r:httpd_t:s0 key=(null)
```

At the end of the previous output, listed under the Raw Audit Messages are these lines:

```
"scontext=root:system_r:httpd_t:s0 tcontext=root:object_r:httpd_sys_script_exec_t:s0 tclass=dir"
```

This output shows you that **httpd** attempted to access a subdirectory with the `httpd_sys_script_exec_t` context type. This type is the context type of **/var/www/cgi-bin**, the directory where **httpd** looks for CGI scripts. The **httpd** daemon, with a `httpd_t` context type, was unable to access this subdirectory because the `httpd_enable_cgi` variable is set to off. With this configuration, SELinux does not allow a user or process of type `httpd_t` to access a directory, file, or process of type `httpd_sys_script_exec_t`. Therefore, the **http** daemon was denied access to the CGI script located in **/var/www/cgi-bin**. If you find similar messages in your log file, set the `httpd_enable_cgi` Boolean to `on`.

**httpd_builtin_scripting**

If you did not configure Apache to load scripting modules by changing the **/etc/httpd/conf/httpd.conf** configuration file, set this Boolean to `off`. If you are unsure, turn `httpd_builtin_scripting` to `off` and check the **/var/log/messages** file for any httpd-related SELinux warnings. See the description of `httpd_enable_cgi` for an example. PHP and other scripting modules run with the same level of access as the **http** daemon. Therefore, turning `httpd_builtin_scripting` to `off` reduces the amount of access available if the Web server is compromised.

To turn off all six of these Booleans and write the values to the policy file by using the **setsebool -P** command follow these steps:

1. Enter the **setsebool -P** command:

```
[root@localhost ~]# setsebool -P httpd_can_sendmail=0
httpd_enable_homedirs=0 httpd_tty_comm=0 httpd_unified=0
httpd_enable_cgi=0 httpd_builtin_scripting=0
```

2. Check all the Boolean settings related to **httpd** by entering `getsebool –a | grep httpd`. The following output shows that all Boolean are set to `off`, including the six previously described variables which default to `on`.

```
[root@localhost ~]$ getsebool -a | grep httpd
allow_httpd_anon_write --> off
allow_httpd_bugzilla_script_anon_write --> off
allow_httpd_mod_auth_pam --> off
allow_httpd_nagios_script_anon_write --> off
allow_httpd_prewikka_script_anon_write --> off
allow_httpd_squid_script_anon_write --> off
allow_httpd_sys_script_anon_write --> off
httpd_builtin_scripting --> off
httpd_can_network_connect --> off
httpd_can_network_connect_db --> off
httpd_can_network_relay --> off
httpd_can_sendmail --> off
httpd_disable_trans --> off
httpd_enable_cgi --> off
httpd_enable_ftp_server --> off
httpd_enable_homedirs --> off
httpd_rotatelogs_disable_trans --> off
httpd_ssi_exec --> off
httpd_suexec_disable_trans --> off
httpd_tty_comm --> off
httpd_unified --> off
httpd_use_cifs --> off
httpd_use_nfs --> off
```

3. Use a Web browser or **wget** to make another request to the **httpd** daemon for the **index.html** file and you should succeed. Rebooting your machine does not change this configuration.

This completes the necessary basic SELinux settings for hardening a Web server with static content. Next, look at hardening scripts accessed by the **http** daemon.

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

## Hardening CGI scripts with SELinux

In the previous section, you used SELinux Booleans to disable scripting because the Web server used only static content. Beginning with that configuration, you can enable CGI scripting and use SELinux to secure the scripts.

1. Confirm that your Web server is configured as described in section “Securing Apache (static content only)” on page 13.
2. Red Hat Enterprise Linux provides a CGI script that you can use for testing. You can find the script at **/usr/lib/perl5/5.8.8/CGI/eg/tryit.cgi.** Copy this script to the **/var/www/cgi-bin/** directory, as follows:

```
[root@localhost ~]$ cp /usr/lib/perl5/5.8.8/CGI/eg/tryit.cgi /var/www/cgi-bin/
```

3. Make sure that the first line of the **tryit.cgi** script contains the correct path to the perl binary. From the `which perl` output shown below, the path should be changed to **!#/usr/bin/perl**.

```
[root@localhost ~]# which perl
/usr/bin/perl
[root@localhost ~]# head -1 /var/www/cgi-bin/tryit.cgi
#!/usr/local/bin/perl
```

4. Confirm that **/var/www/cgi-bin** is assigned the `httpd_sys_script_exec_t` context type as follows:

```
[root@localhost ~]$ ls -Z /var/www/ | grep cgi-bin
drwxr-xr-x  root      root root:object_r:httpd_sys_script_exec_t cgi-bin
```

5. Allow and confirm read and execute permission for the **tryit.cgi** script to all users:

```
[root@localhost cgi-bin]# chmod 555 /var/www/cgi-bin/tryit.cgi
[root@localhost cgi-bin]# ls -Z
-r-xr-xr-x  root root root:object_r:httpd_sys_script_exec_t tryit.cgi
```

6. Confirm that **/var/www/cgi-bin/tryit.cgi** is assigned the `httpd_sys_script_exec_t` context type:

```
[root@localhost ~]$ ls -Z /var/www/cgi-bin/tryit.cgi
-r-xr-xr-x  root root root:object_r:httpd_sys_script_exec_t
/var/www/cgi-bin/tryit.cgi
```

7. Enable CGI scripting in SELinux and confirm that it is enabled:

```
[root@localhost cgi-bin]$  setsebool   httpd_enable_cgi=1
[root@localhost cgi-bin]$ getsebool httpd_enable_cgi
httpd_enable_cgi --> on
```

8. Open a Web browser and type the Web server address into the location bar. Include the **/cgi-bin/tryit.cgi** in the URL. For example, type `http://192.168.1.100/cgi-bin/tryit.cgi`. The **tryit.cgi** script should return output similar to Figure 1:

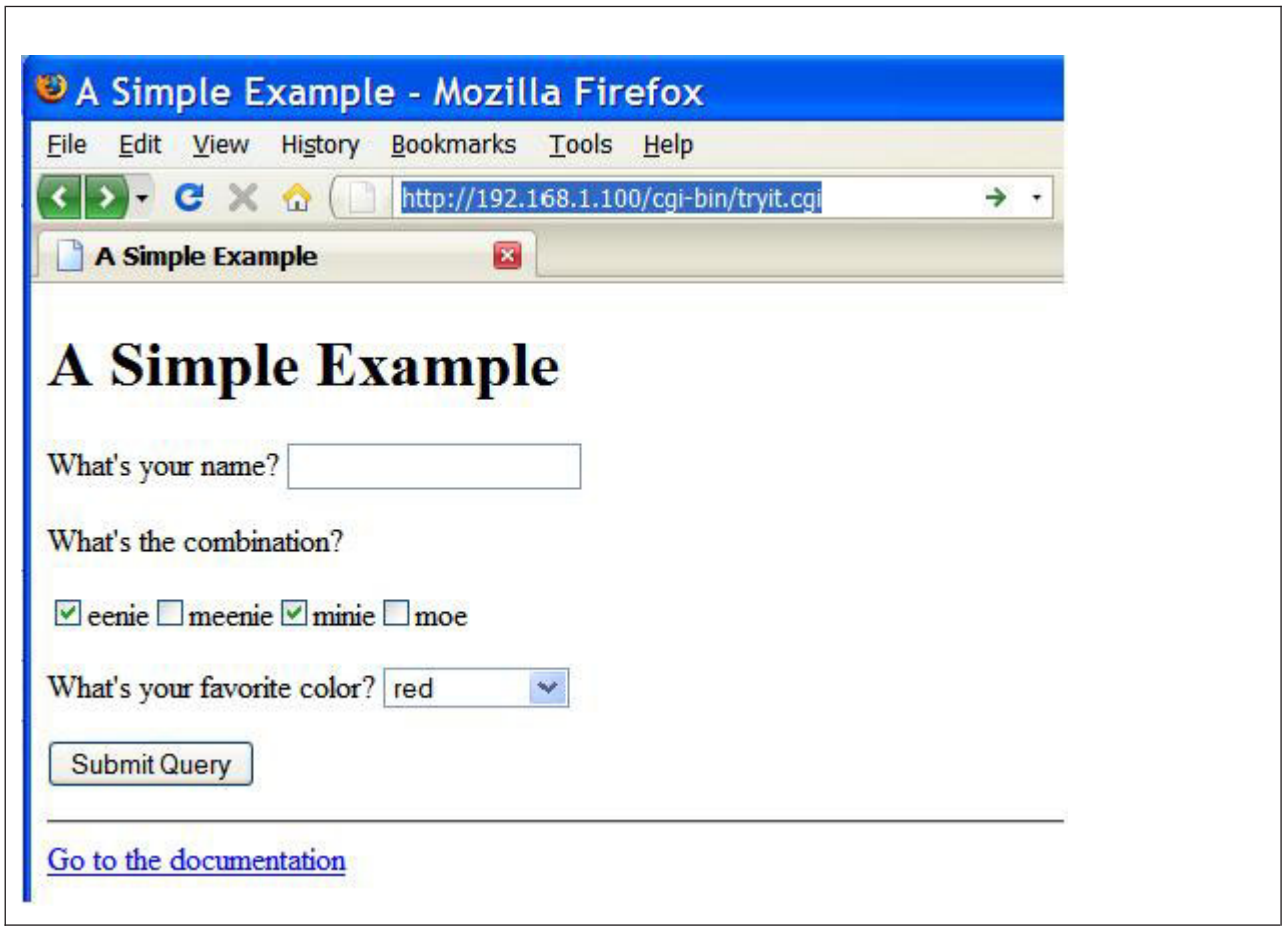


*Figure 3. Figure 1: A Simple Example*

9. Provide test answers to the form fields and click **Submit Query**. The **tryit.cgi** script should return output similar to Figure 2:

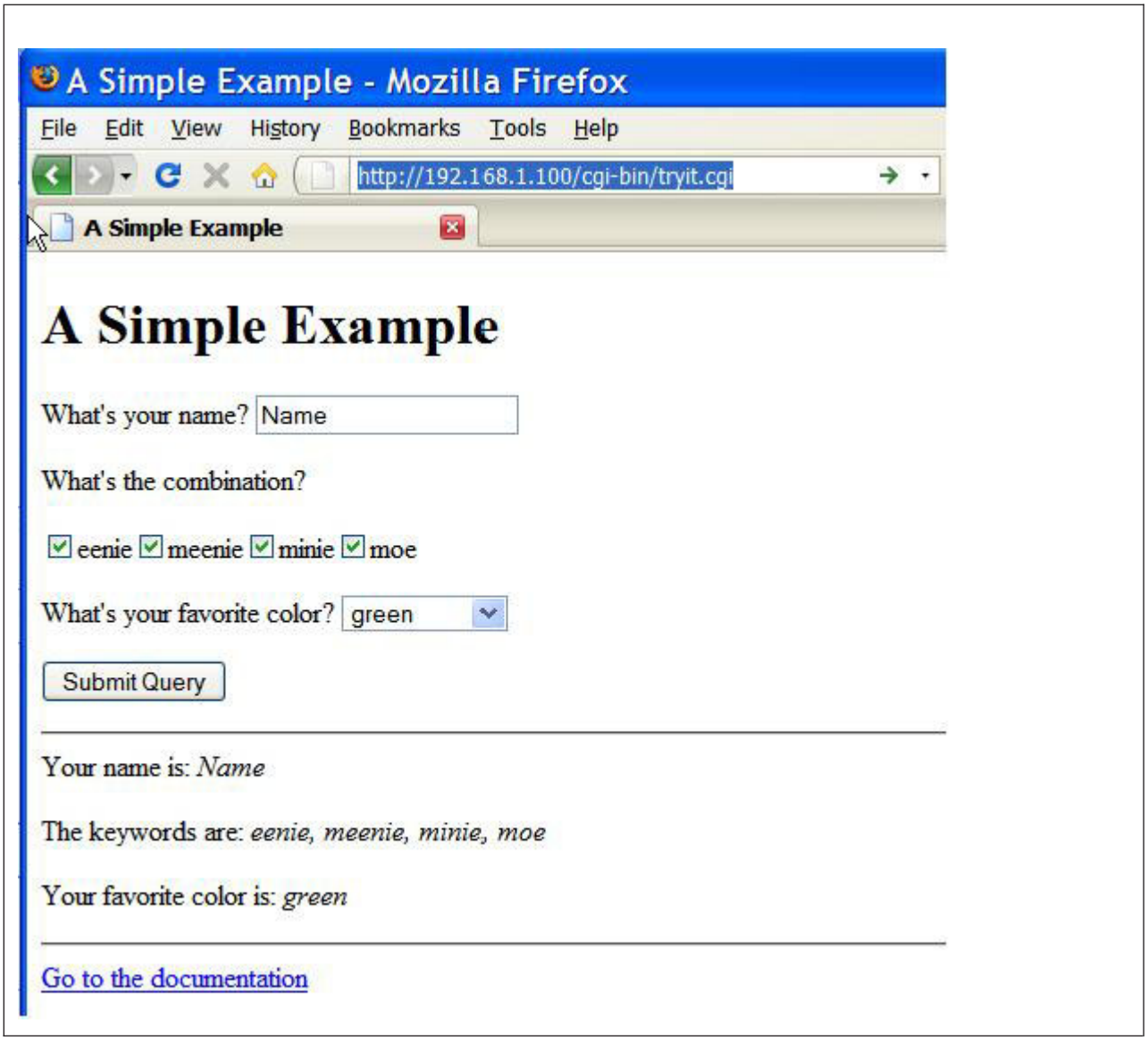


*Figure 4. Figure 2: A Simple Example with results*

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

# Appendix. Related information and downloads

## Related information

- Wikipedia: Security-Enhanced Linux
  http://en.wikipedia.org/wiki/Selinux
- Bell-La Padula model
  http://en.wikipedia.org/wiki/Bell-La_Padula_model
- NSA Security-Enhanced Linux
  http://www.nsa.gov/research/selinux/index.shtml
- Managing Red Hat Enterprise Linux 5 presentation
  http://people.redhat.com/dwalsh/SELinux/Presentations/ManageRHEL5.pdf
- developerWorks Security Blueprint Community Forum
  http://www.ibm.com/developerworks/forums/forum.jspa?forumID=1271
- Red Hat Enterprise Linux 4: Red Hat SELinux Guide
  http://www.linuxtopia.org/online_books/redhat_selinux_guide/rhlcommon-section-0055.html
- F. Mayer, K. MacMillan, D. Caplan, "SELinux By Example – Using Security Enhanced Linux" Prentice Hall, 2007

**Related reference**:

Chapter 1, “Scope, requirements, and support,” on page 1
This blueprint applies to System x running Linux and PowerLinux. You can learn more about the systems to which this information applies.

# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Dept. LRAS/Bldg. 903
11501 Burnet Road
Austin, TX 78758-3400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

## Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® and ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java™ and all Java-based trademarks and logos are registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Printed in USA