SELinux Quick Start Guide

SELinux Quick Start Guide

Revision History

Revision $Revision: 1.1 $ $Date: 2006/01/03 17:21:26 $ Revised by: pd

Table of Contents

1. Introduction 1
2. SELinux Overview 3
2.1. History 3
2.2. The SELinux Security Model 3
2.3. Security Contexts 3
2.3.1. User Identities 3
2.3.2. Roles 4
2.3.3. Domains and Types 4
2.4. Security Policies 4
2.4.1. Access Decisions 4
2.4.2. Transition Decisions 4
2.5. Requirements 5
2.5.1. Kernel 5
2.5.2. Shared Library 5
2.5.3. Filesystems and Extended Attributes 5
2.5.4. User Utilities 5
2.5.5. SELinux Policy 5
3. Administration 7
3.1. Disabling SELinux 7
3.2. SELinux Commands 7
3.2.1. chcon 7
3.2.2. checkpolicy 7
3.2.3. fixfiles 7
3.2.4. getenforce 7
3.2.5. newrole 8
3.2.6. restorecon 8
3.2.7. run_init 8
3.2.8. sestatus 8
3.2.9. setenforce 8
3.2.10. setsebool 8
3.2.11. seuser 8
3.3. Modified Linux Commands 8
3.3.1. cp 8
3.3.2. id 8
3.3.3. ls 9
3.3.4. mv 9
3.3.5. ps 9
3.3.6. cron 9
3.3.7. rsync 9
3.3.8. ssh 9
3.3.9. tar 9
3.3.10. logrotate 9
3.3.11. Password Related Commands 9
3.4. The /selinux Filesystem 10
3.5. Filesystem Relabeling 10
3.6. Logging and Auditing 10
3.6.1. Reading Log Entries 10
3.6.2. audit2allow 10

3.6.3. Log Rate Limiting 10
4. SELinux Policy 13
4.1. Policy Rules 13
4.1.1. type 13
4.1.2. allow 13
4.1.3. auditallow 13
4.1.4. dontaudit 13
4.1.5. neverallow 13
4.2. Policy Macros 14
4.3. File Contexts 14
4.4. Compiling Policy 14
4.5. Policy Booleans 14
4.6. Policy Development Example 14
4.6.1. Localized customization 15
4.6.2. Daemon Policy Example 15
A. Resources 17
B. mysqld.te Policy Source 19

Chapter 1. Introduction

This guide is meant for the experienced systems administrator who needs to learn more about administering servers running Security Enhanced Linux. A brief overview of SELinux history and concepts will be followed by details of day to day administration of an SELinux server and how this differs from the vanilla Linux security model.

Chapter 2. SELinux Overview

This chapter provides a brief overview of SELinux, the concepts behind it, its implementation and its requirements.

2.1. History

SELinux was originally released under the GPL in late 2000 by the National Security Agency's Office of Information Assurance. Since then it has been developed by the open source community in collaboration with the NSA.

SELinux currently ships as a part of Fedora Core, and it supported by Red Hat as a part of Red Hat Enterprise Linux. Packages also exist for Debian, SuSe, and Gentoo, though these are unsupported by anyone at the time of this writing.

2.2. The SELinux Security Model

SELinux is based on the concept of Mandatory Access Control. Under MAC, administrators control all interactions of software on the system. The concept of least privilege is used, by default applications and users have no rights, as all rights must be granted by an administrator as part of the system's security policy.

This contrasts with the Discretionary Access Control concept that is the standard Linux security model. Under DAC, files are owned by a user and that user has full control over them. An attacker who penetrates an account can do anything with the files owned by that user. For example, an attacker penetration a web server has full control over all files owned by the webserver account. Worse, if an application runs under the context of the root user, an attacker penetrating it now has full control over the entire system.

MAC in effect provides each application with a virtual sandbox that only allows the application to perform the tasks it is designed for and explicitly allowed in the security policy to perform. For example, the webserver process may only be able to read web published files and serve them on a specified network port. An attacker penetrating it will not be able to perform any activities not expressly permitted to the process by the security policy, even if the process is running as the root user.

Standard Unix permissions are still present on the system, and will be consulted before the SELinux policy when access attempts are made. If the standard permissions would deny access, access is simply denied and SELinux is not involved. If the standard file permissions would allow access, the SELinux policy is consulted and access is either allowed or denied based on the security contexts of the source process and the targeted object.

2.3. Security Contexts

SELinux makes access decisions by checking the security context of the subject (a process associated with a user) against the action attempted (e.g. a file read) and the security context of the object (such as a file or network port).

A security context consists of three components: a user identity, a role, and a type (also known as a domain).

2.3.1. User Identities

A user identity indicates the SELinux user account associated with a subject or object. These should not be confused with the standard Linux user accounts in /etc/passwd, the Linux accounts are mapped to a corresponding SELinux account but this does not need to be a one-to-one relationship.

The standard strict policy does not specifically restrict access based on user accounts, but thie functionality exists and could be added as custom policy for specialized installations that require it. User accounts are however used to determine what roles a specific user is permitted to assume.

2.3.2. Roles

A role defines a set of permissions granted to a user. Users can change roles to any role permitted to their user identity by using the *newrole* command. Roles are conventionally named with an "_r" suffix.

The strict policy assigns all users to the user_r role. Administrators are given the staff_r role, and are allowed to transition from that role to the sysadm_r role. Under SeLinux, the root account alone gives no special privileges, instead the sysadm_r role is used to perform administration duties. The SELinux implementation in Fedora Core modifies the su command to automatically transition to the sysadm_r role when assuming the root identity without requiring a specific newrole command to be issued.

Most files on the system do not require a role, but every object must have all three parts of a security context. These files are assigned the role of object_r as a default.

2.3.3. Domains and Types

Domains and types are synonyms, typically the term "domain" is used when referring to processes and the term "type" is used referring to objects. Types are denoted by a "_t" suffix to distinguish them from user identites and roles.

Types are the primary method used by SELinux to make authorization decisions. The strict policy defines relatively few users and roles, but contains hundreds of types.

2.4. Security Policies

The heart of SELinux is the security policy. The policy assigns files to a security context, declares what processes are given what rights to these contexts, and handles transitions between security contexts. For example, when the init_t context launches Apache, the policy forces a transition to the httpd_t security domain.

Fedora Core ships with two distinct security policies, known as targeted and strict.

- The targeted policy confines only certain daemons known to be security issues, such as httpd and named, with the rest of the system running in an unconfined security context. This is the only policy supported by Red Hat for use on Red Hat Enterprise Linux.
- The strict policy is intended to fully implement SELinux controls on the entire system. Currently use of this policy is unsupported and considered developmental.

This document focuses on the strict policy, unless otherwise noted.

2.4.1. Access Decisions

Access decisions are made when a process running in a specified domain attempts to perform an action on a subject of a specified type. Policy is checked to determine if the desired operation is permitted.

2.4.2. Transition Decisions

Transition decisions determine what domain or type newly created processes or files are assigned to.

Executing a program in a child process may result in the new process running under a different domain than the parent process, if the program is defined as a domain entry point in the policy.

Newly created files are by default assigned the same type as their parent directory, but policy can override this behavior and specify a type to be assigned to specified files upon their creation.

2.5. Requirements

Due to the way SELinux must be embedded and directly affect the internal workings of any system it is installed on, it has many software requirements that must function together as a unified whole.

2.5.1. Kernel

SELinux requires a kernel component known as LSM, Linux Security Modules. This component is available as a patch to the 2.4 series of the Linux kernel, but is included in the main kernel tree of the 2.6 series.

2.5.2. Shared Library

The SELinux API is contained in a shared library which must be present on the system.

2.5.3. Filesystems and Extended Attributes

SELinux uses extended attributes to store security labels on each file. These extended attributes require the use of the ext2 or ext3 filesystems. The XFS filesystem is also known to work, but notably reiserfs is not compatible with these extended attributes.

2.5.4. User Utilities

Many common utilities need to be updated to either provide security context information or to take the filesystem extended attributes into account. See Chapter 3 for details on the specific utilities that need to be SELinux aware. These are linked against the SELinux shared library also required to be present on the system.

2.5.5. SELinux Policy

A functional compiled policy is required, since the default behavior of SELinux is to deny access unless an action is specifically allowed by the security policy. See Chapter 4 for more details about SELinux policy.

Chapter 3. Administration

This chapter will discuss specialized administration tasks unique to an SELinux enabled system, as well as detailing how some common systems administration tasks differ on an SELinux system.

3.1. Disabling SELinux

SELinux can be disabled completely at bootup by passing *selinux=0* to the kernel command line. This will completely disable SELinux until a reboot, and you will need to relabel files when rebooting to ensure proper operation.

A better alternative for temporarily disabling SELinux is to put the system into *permissive mode*. Permissive mode will log actions that would have been denied, but will not actually deny them. This contrasts with the normal *enforcing mode* that will actively deny access to actions not explicitly allowed by the currently running policy.

To put the system into permissive mode, issue the command *setenforce 0* while in the sysadm_r role or pass *enforcing=0* to the kernel command line at bootup. To resume enforcing mode, issue the command *setenforce 1*. Issuing the command *getenforce* will return the mode SELinux is running currently.

SELinux modes can also be set by editing the SELinux config file located at */etc/selinux/config*. The *SELINUX=* line can be set to *enforcing*, *permissive*, or *disabled*, and will take effect upon the next reboot.

3.2. SELinux Commands

SELinux includes a variety of specialized commands for its administration and use.

3.2.1. chcon

chcon is used to label a file or files with a specified security context.

3.2.2. checkpolicy

checkpolicy is a tool used to compile policy sources into a binary policy file. Generally it is not called directly, but invoked by the policy's Makefile. See Section 4.4 for more details on compiling SELinux policy.

3.2.3. fixfiles

fixfiles can be used to relabel the entire filesystem based on the current policy, or to relabel a packaged application's files based on the information included in that application's rpm package.

The command *touch /.autorelabel* can also be used to relabel the entire filesystem upon the next reboot of the system.

3.2.4. getenforce

getenforce returns the current enforcement state of the SELinux system, either *permissive* or *enforcing*. The permissive state will log denials but not actually enforce them, which can be very useful during policy development. The normal production state should always be enforcing.

3.2.5. newrole

The newrole command is used to switch roles. Typically the command would be issued as *newrole -r sysadm_r* to transition to the sysadm_r role for system administration tasks.

3.2.6. restorecon

restorecon is used to relabel selected files back to their default context, as defined in the security policy.

3.2.7. run_init

Rather than starting daemons by running the appropriate script in /etc/init.d, you must use *run_init* and pass the script path and arguments on its command line. This is required to perform the proper security transitions so that the daemon runs under its proper security context.

3.2.8. sestatus

sestatus displays the current status of SELinux, including the status (either permissive or enforcing), policy version, and the settings of all policy booleans.

3.2.9. setenforce

setenforce is used to toggle the SELinux status between enforcing and permissive. Issue *setenforce 0* to enter permissive mode, or *setenforce 1* to enter enforcing mode.

3.2.10. setsebool

setsebool is used to toggle policy booleans on or off. See Section 4.5 for an explanation of policy booleans.

3.2.11. seuser

seuser is used to create, delete, and modify SELinux users and roles. These are not to be confused with normal Linux user accounts.

3.3. Modified Linux Commands

Many standard Linux commands must be modified for use on an SELinux system. The commands need to take into account extended attributes and security context information.

3.3.1. cp

cp accepts a -Z flag to set the security context of the newly created file. If not specified, the context of the new file will default to that of the destination directory.

3.3.2. id

id now displays the current user's security context information along with the user and group information. It will also accept a -Z flag to display only the security context.

3.3.3. ls

ls accepts a -Z flag also, to display the security context of each file in the listing.

3.3.4. mv

An important note when using the *mv* command is that the new file will retain its security context when moved. For example, moving a file from a user home directory to an http served directory will result in the file retaining its user_home_t type, which, under normal policy, will not be readable by the httpd daemon.

3.3.5. ps

ps also accepts a -Z flag which displays the security context of each running process.

3.3.6. cron

cron is modified to provide a standard security context for all *cron* jobs.

3.3.7. rsync

rsync must be able to support extended attributes with the -X flag for proper use under SELinux.

3.3.8. ssh

A modified *ssh* must be installed to set the correct security context when a user logs in remotely.

3.3.9. tar

tar should not be used under SELinux, as it will not embed the extended attributes SELinux uses to store its security context information. *star* is the replacement tool that must be used as a replacement for *tar*.

3.3.10. logrotate

logrotate is modified to preserve the security context of the log files as they are rotated.

3.3.11. Password Related Commands

Any commands that interface with the /etc/passwd or /etc/shadow files must be SELinux aware in order to use the SELinux API to obtain password information and preserve the security context of these files. Examples of these types of commands are *useradd*, *groupadd*, *passwd*, the *pam* library, and *login*.

3.4. The /selinux Filesystem

The /selinux filesystem is a pseudo-filesystem, like /proc or /sys. It can be written to in order to change flags such as policy booleans or to switch between permissive and enforcing modes, and read from to find the current status of various parts of the SELinux system.

During normal operation, the commands from Section 3.2 will suffice. Previous versions of SELinux required more use of the /selinux filesystem for administration, but this is no longer necessary.

3.5. Filesystem Relabeling

Labels are how security contexts are associated with files and are stored as part of a file's extended attributes. If the system is started with SELinux disabled these labels can be inadvertently removed or become out of sync. A typical SELinux problem troubleshooting begins with a filesystem relabel to ensure that the labels are correct.

To relabel part or all of the filesystem on the fly without a reboot, see Section 3.2 and the *fixfiles* and *restorecon* commands. This is useful, but you can still run into problems with certain daemons by relabeling files without a clean reboot. Issue a *touch /.autorelabel* command to create the file */.autorelabel* which will trigger the automatic relabeling of the entire filesystem during the next boot process.

3.6. Logging and Auditing

SELinux logs all its messages to the system log, or to the audit log if running under Fedora Core with the auditd daemon running.

3.6.1. Reading Log Entries

Reading log entries is the bread and butter of SELinux administration. All troubleshooting of SELinux denials, as well as development of new security policy, involves reading the logs for SELinux denials.

```
type=AVC msg=audit(1117726540.077:9322426): avc: denied { read } for pid=10652 comm="tail" name=audit.log dev=dm-0
ino=328749 scontext=root:staff_r:staff_t tcontext=system_u:object_r:auditd_log_t tclass=file
```

The above logged denial tells an administrator that SELinux denied a read attempt on the file *audit.log* using the *tail* command. The source process context was *root:staff_r:staff_t*, and the targeted file's context was *system_u:object_r:auditd_log_t*. We know from this that the staff_t domain has no read access to the auditd_log_t file type. This is as it should be, if we had used a newrole command to transition to the sysadm_r role we would be running *tail* in the sysadm_t domain and access would have been granted.

3.6.2. audit2allow

audit2allow is a perl script that reads logged denials and turns them into rules that can be added to SELinux policy to thereafter allow the operations that were denied. It is not intended as an automatic policy generator, but as an aid to developing policy. *audit2allow* output should always be read carefully before adding it to the system policy, as it may allow more access than strictly necessary for the application involved. The tool is best used as a guideline when writing policy, not as an actual generator of policy itself. See Section 4.6 for examples of the use of audit2allow.

3.6.3. Log Rate Limiting

SELinux will sometimes limit the amount of log entries it writes during periods of high activity, in order to prevent flooding the logs with repeated entries of the same denial. This can cause difficulty when troubleshooting, as a denial can be occasionally made with no corresponding log entry.

There is currently no way of knowing when the rate limiting has been triggered. In the future it is planned to migrate SELinux to its own logging solution, but for now system administrators must keep in mind that log enties may be lost during periods of high activity.

Chapter 4. SELinux Policy

The policy is central to SELinux operation, since any operation not explicitly permitted in the policy will be denied by default. The policy also declares the default security contexts for every file on the system.

Policy is stored as a binary file, compiled from source. The source is not required to be present on the system to use SELinux, but policy cannot be modified without recompiling from source. This means that when installing new software either the policy must already contain rules for the new software or the policy must have these rules added and be recompiled. Keeping rules for uninstalled software in the binary policy will result in greater memory usage by the policy, but not keeping those rules will require a policy recompile after any software installation.

Policy rules and macros are saved as .te files in the *domains/program* subdirectory of the source tree before compiling into the binary policy, and file contexts are saved as .fc files in the *file_contexts/program* subdirectory. The files in these directories are the bulk of the SELinux policy, and the ones that system adminstrators will need to work with when adding to or changing existing policy.

4.1. Policy Rules

The default for action without a specific allow rule on an SELinux system is to deny it. Therefore, any action that the system should be allowed to perform must have a corresponding rule in the security policy or it will be denied.

4.1.1. type

The *type* statement is used to declare a security type for specified files. Attributes for the file type can also be indicated, for example, adding the attribute *sysadmfile* to a *type* statement declares that file type to be a system administration file, which is allowed to be edited by processes running in the *sysadm_t* domain.

4.1.2. allow

The *allow* rule allows a specified access and makes no log entry of the operation. This is by far the most common rule in the SELinux policy.

4.1.3. auditallow

The *auditallow* rule, despite its name, does not allow the specified operation, but will log the attempt. To allow an operation while logging it, you must have both an *allow* and an *auditallow* for the specified operation.

4.1.4. dontaudit

The *dontaudit* rule is used for those actions that the policy author wants to be denied, but not logged. Some applications attempt actions that they do not require for proper operation, and the policy will disallow these but not clutter the log with the unimportant denial if a *dontaudit* rule is used.

4.1.5. neverallow

neverallow exists to ensure that certain operations are never allowed by the policy, even if specifically allowed by an *allow* rule elsewhere within the policy source.

4.2. Policy Macros

Much of the SELinux security policy is written using m4 macros. For example, the *can_network* macro can be passed a domain as an argument and upon policy compilation the macro will expand to several allow lines that give the specified domain the ability to open ports and connect to remote nodes.

Macros save large amounts of time when writing policy, at the expense of some redundancy within the policy. Macros are kept in the *macros* subdirectory of the policy source. Policy authors should read and understand these macros, they are a large part of successfully writing new policy and understanding existing policy.

4.3. File Contexts

File contexts are declared as part of the SELinux policy, and are saved in .fc files in the policy sources. The context for a file is declared as a regular expression matching the full path of the file or files, followed by a security context these files should be set to.

When relabeling the system, the files are set to this default context defined in the policy. It is advisable when changing a file's context using *chcon* to add the change to the policy source and recompile, otherwise a future relableing will undo the change you made, reverting the file back to the default label set in the policy.

4.4. Compiling Policy

Policy is compiled using a Makefile and running make in the policy source directory. The various .fc and .te files are concatenated and any macros within them are expanded to produce the *policy.conf* file, which is then compiled into a *policy.<version>* binary policy file.

The policy Makefile accepts one of several targets:

- *policy* - Compiles the policy, but does not install or load it into memory.
- *install* - Compiles and installs policy, but does not load it into memory.
- *load* - Compiles, installs and loads the policy.
- *reload* - Compiles, installs and loads the policy. Currently equivalent to *load*.
- *relabel* - Relabels the filesystem based on the currently running policy, does not compile or install.

4.5. Policy Booleans

Policy booleans are the only way to change a running policy on the fly without recompilation. These are embedded within the policy as branching statements that depend on the current value of the boolean, which can be toggled by an adminstrator on the fly to change the policy behavior.

The *sestatus* command will display the full list of all booleans in the running policy and their current value.

4.6. Policy Development Example

Policy development can be a tricky process. It is generally best to perform policy development in *permissive mode*, if running in *enforcement mode* it is possible to render your system inoperable by making a policy development error.

4.6.1. Localized customization

All policy customization done for a specific machine should be added to the *$SELINUX_SRC/domains/misc/local.te* file to ensure that future policy updates do not alter or remove the changes. Changes to file contexts are similarly added to *$SELINUX_SRC/file_contexts/misc/local.fc*. If these files do not exist they should be created.

Policy for daemons are in individual .te files located in *$SELINUX_SRC/domains/program/* . Policy for daemons that are not installed on a given system can be moved out of this directory and into *$SELINUX_SRC/domains/program/unused/* , where they will not be added into the policy during compilation. This can save resources, but could cause problems if the daemon is ever installed without its corresponding policy file being re-compiled into the active policy.

4.6.2. Daemon Policy Example

The mysqld.te policy is a good example of SELinux daemon policy. A listing of this policy is provided in Appendix B and should be referred to during this section.

An understanding of m4 macros is extremely helpful in reading and understanding SELinux policy. A link to m4 documentation can be found in Appendix A. SELinux makes extensice use of macros in policy development, the *core_macros.te* and *global_macros.te* files located in *$SELINUX_SRC/macros/* contain many of these macros. Familiarity with these macros is a key to successful policy development.

Because SELinux is still undergoing such rapid development, there currently exists no documentation of the complete list of macros other than the policy sources themselves. Reading policy sources and familiarizing yourself with the commonly used macros is a key to successful policy development.

4.6.2.1. Macros

The mysqld policy begins with a macro call to the *daemon_domain* macro. This macro sets up types and access common to most service daemons and is found in the *global_macros.te* file. Other macros included in the MySQL policy are *etcdir_domain*, *log_domain*, and *tmp_domain* which create subdomains of the mysqld_t domain for configuration files, logfiles, and temporary files respectively.

Also called by the MySQL policy is the *can_network_server* macro, which allows the domain to open a network port and service requests on it, and the *can_ypbind* macro, which allows the use of an NIS server for user authentication.

4.6.2.2. Types

Several types are defined by the macros in the previous section, such as *mysqld_exec_t*, created by the daemon_domain macro and used as the type for the MySQL executable files. Also created by macros are *mysqld_etc_t*, *mysqld_log_t*, and *mysqld_tmp_t*, used for configuration files, logfiles and temporary files respectively.

Other types are explicitly defined in the *mysqld.te* policy file. For example, the line

type mysqld_db_t, file_type, sysadmfile

defines the *mysqld_db_t* file type used for the MySQL database files themselves and flags them with the *sysadmfile* attribute to allow the system administrator to manage them. The *create_dir_file* macro is then called with the appropriate parameters to allow the *mysqld_t* domain to create and use the *mysqld_db_t* files.

4.6.2.3. Ifdefs

Ifdefs are used to control conditional compilation of policy based on whether other components are present in the policy. For example, the *mysqld.te* policy contains an ifdef so that if *logrotate.te* is present on the system, the *logrotate_t* domain is given access to certain *mysqld_t* subdomains.

4.6.2.4. Allow Rules

The bulk of any policy file is composed of *allow* statements. An example is the line:

allow mysqld_t etc_t:dir search;

This allows the *mysqld_t* domain to perform a search operation on directories labeled with the *etc_t* type. Since policy is developed based on the logged denials when running the system in permissive mode, this indicates that mysql searches the */etc* directory for its own files and therefore we have this rule in the policy to allow that to happen.

4.6.2.5. Booleans

A policy boolean is defined at the end of the *mysqld.te* file. This demonstrates how to set certain rules to be conditional based on the runtime state of the boolean. In this example, the policy will allow processes running in a user domain to connect to MySQL, if and only if the *allow_user_mysql_connect* boolean is set to true. The line:

bool allow_user_mysql_connect false

defines the boolean and sets it to a default of false.

Appendix A. Resources

This is a list of valuable online resources of SELinux information. SELInux is currently undergoing extremely rapid development so the information in this document is very subject to change. The links contained in this section represent the best sources of current SELinux development status.

- **NSA SELinux** - The NSA (National Security Agency) SELinux page. Contains a good overview of the theory of information security behind SELinux.

  http://www.nsa.gov/selinux/

- **NSA SELinux Mailing List** - The NSA mailing list for SELinux development. The bleeding edge development of SELinux happens on this list, as well as discussion of the future direction of SELinux.

  http://www.nsa.gov/selinux/info/list.cfm

- **Unofficial SELinux FAQ** - This unofficial FAQ covers generalized SELinux information that is not specific to any one distribution. Contains more practical and less theoretical information than the official NSA SELinux FAQ.

  http://www.crypt.gen.nz/selinux/faq.html

- **Fedora Core 3 SELinux FAQ** - A good guide to the SELinux implementation in Fedora Core 3. Some of this has changed in the test releases of Fedora Core 4, but overall this is still a very useful source of information on that specific implementation. Heavy on practical details and adminstration commands.

  http://fedora.redhat.com/docs/selinux-faq-fc3/

- **Fedora SELinux Mailing List** - The Fedora SELinux mailing list discusses SELinux development in the context of the Fedora Core development build. A good resource for questions about practical SELinux matters.

  http://www.redhat.com/mailman/listinfo/fedora-selinux-list

- **Understanding and Customizing the Apache HTTP SELinux Policy** - A guide to the specific Apache policy shipping with Fedora Core 3. This is a good guide to SELinux policy, using real world httpd policy as an example.

  http://fedora.redhat.com/docs/selinux-apache-fc3/index.html

- **Red Hat SELinux Guide** - A guide to SELinux as shipped with Red Hat Enterprise Linux 4. Another good overview, though it only covers the *targeted* policy since the *strict* policy is not currently supported in RHEL4.

  http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/selinux-guide/

- **Tresys Technology SELinux** - The Tresys Technology SELinux pages. Tresys is a company providing SELinux development, support, and training. They are heavy contributors to SELinux development and their website has some very up to date training resources.

  http://www.tresys.com/selinux/

- **SELinux: NSA's Open Source Security Enhanced Linux** - O'Reilly publishes this SELinux book. Very good, but as with any printed material on something being developed as quickly as SELinux, a bit out of date.

  http://www.oreilly.com/catalog/selinux/index.html

- **GNU m4 Macro Processor** - Reading and writing policy relies on extensive use and understanding of m4 macros, documentation of which is provided at the following link.

  http://www.gnu.org/software/m4/manual/index.html

Appendix B. mysqld.te Policy Source

```
#DESC Mysqld - Database server
#
# Author:  Russell Coker <russell@coker.com.au>
# X-Debian-Packages: mysql-server
#

##################################
#
# Rules for the mysqld_t domain.
#
# mysqld_exec_t is the type of the mysqld executable.
#
daemon_domain(mysqld)

type mysqld_port_t, port_type;
allow mysqld_t mysqld_port_t:tcp_socket name_bind;

allow mysqld_t mysqld_var_run_t:sock_file create_file_perms;

etcdir_domain(mysqld)
type mysqld_db_t, file_type, sysadmfile;

log_domain(mysqld)

# for temporary tables
tmp_domain(mysqld)

allow mysqld_t usr_t:file { getattr read };

allow mysqld_t self:fifo_file { read write };
allow mysqld_t self:unix_stream_socket create_stream_socket_perms;
allow initrc_t mysqld_t:unix_stream_socket connectto;
allow initrc_t mysqld_var_run_t:sock_file write;

allow initrc_t mysqld_log_t:file { write append setattr ioctl };

allow mysqld_t self:capability { dac_override setgid setuid net_bind_service };
allow mysqld_t self:process { setsched getsched };

allow mysqld_t proc_t:file { getattr read };

# Allow access to the mysqld databases
create_dir_file(mysqld_t, mysqld_db_t)
allow mysqld_t var_lib_t:dir { getattr search };

can_network_server(mysqld_t)
can_ypbind(mysqld_t)

# read config files
r_dir_file(initrc_t, mysqld_etc_t)
allow mysqld_t { etc_t etc_runtime_t }:{ file lnk_file } { read getattr };

allow mysqld_t etc_t:dir search;
```

```
read_sysctl(mysqld_t)

can_unix_connect(sysadm_t, mysqld_t)

# for /root/.my.cnf - should not be needed
allow mysqld_t sysadm_home_dir_t:dir search;
allow mysqld_t sysadm_home_t:file { read getattr };

ifdef(‘logrotate.te’, ‘
r_dir_file(logrotate_t, mysqld_etc_t)
allow logrotate_t mysqld_db_t:dir search;
allow logrotate_t mysqld_var_run_t:dir search;
allow logrotate_t mysqld_var_run_t:sock_file write;
can_unix_connect(logrotate_t, mysqld_t)
’)

ifdef(‘daemontools.te’, ‘
domain_auto_trans( svc_run_t, mysqld_exec_t, mysqld_t)
allow svc_start_t mysqld_t:process signal;
svc_ipc_domain(mysqld_t)
’)dnl end ifdef daemontools

ifdef(‘distro_redhat’, ‘
allow initrc_t mysqld_db_t:dir create_dir_perms;

# because Fedora has the sock_file in the database directory
file_type_auto_trans(mysqld_t, mysqld_db_t, mysqld_var_run_t, sock_file)
’)
ifdef(‘targeted_policy’, ‘’, ‘
bool allow_user_mysql_connect false;

if (allow_user_mysql_connect) {
allow userdomain mysqld_var_run_t:dir search;
allow userdomain mysqld_var_run_t:sock_file write;
}
’)
```