Getting Started with SE Linux HOWTO: the new SE Linux

Faye Coker
faye@lurking-grue.org
Last update: 18 March 2004

This document has been revised for the new SE Linux. The old "Getting Started with SE Linux HOWTO" will remain in place as a legacy document, but it is highly recommened that all new installs of SE Linux use the new SE Linux. The new SE Linux runs on 2.6.x kernels and has also been backported for 2.4.x. This document is largely a duplicate of that for the old SE Linux, with modifications made where necessary.

This document is a general introduction to NSA Security Enhanced Linux. It is mainly focused on Debian and as such command examples given for package management are largely Debian based. This document is tailored towards people wanting to get started with SE Linux so there's no confusing advanced stuff here. See the Resources section for links to other SE Linux material.

Table of Contents

---

1. Introduction

This document was put together in response to people asking if an intro level HOWTO was available for getting started with SE Linux. It covers the more basic aspects of SE Linux such as terminology, installation and adding users in addition to a few other areas. A more advanced HOWTO-type of document will follow, including areas such as how to edit policy files (which causes a little too much information overload with users new to SE

Linux and is not included here).

1.1. Feedback

Comments on this document are welcome. Please email faye@lurking-grue.org

1.2. Disclaimer

This document is a guide only. I strongly recommend you install SE Linux on a test machine before deploying on a production server.

1.3. New features of the new SE Linux

The new SE Linux has a number of new features, listed below.

***/selinux* filesystem**
A */selinux* filesystem is now included. Part of the installation process requires you to edit */etc/fstab* accordingly. The */selinux* filesystem is similar to */proc* in that it is also a pseudo filesystem. Doing a *ls -l /selinux* shows

```
total 0
-rw-rw-rw-    1 root     root            0 Nov 25 11:27 access
-rw-rw-rw-    1 root     root            0 Nov 25 11:27 context
-rw-rw-rw-    1 root     root            0 Nov 25 11:27 create
-rw-------    1 root     root            0 Nov 25 14:19 enforce
-rw-------    1 root     root            0 Nov 25 11:27 load
-r--r--r--    1 root     root            0 Nov 25 11:27 policyvers
-rw-rw-rw-    1 root     root            0 Nov 25 11:27 relabel
-rw-rw-rw-    1 root     root            0 Nov 25 11:27 user
```

Running the cat command on the file "enforce" will show either a 1 for enforcing mode, or 0 for permissive mode.

Use of extended attributes
The new SE Linux uses extended attributes to store security contexts. You must build your kernel with extended attribute support. Extended attributes are a name-data tuple-- for example, **security.selinux** is the name of an attribute and the security context is the data. You can see the security context of a file with the command *ls --context filename* (further explained in this document) if SE Linux is running, but if you want to see the extended attributes when SE Linux isn't (or is) running, use the *getfattr* command. Note that you must first install the package **attr** and from there read the man page for *getfattr*. Running the command as follows gives

```
faye@kaos:~$ getfattr -m . -d /etc/passwd
getfattr: Removing leading '/' from absolute path names
# file: etc/passwd
security.selinux="system_u:object_r:etc_t\000"
```

The attribute **security.selinux** has the context which matches the file you are querying, so in the above case the context is **system_u:object_r:etc_t** All files on ext2 and ext3 filesystems on the new SE Linux have the attribute **security.selinux** (a key new feature). If you were to boot to a non SE Linux kernel, the extended attributes would still be there and you could still see them. The extended attribute is set when you run *setfiles* which sets the file security contexts during a *make relabel* operation.

Loading SE Linux policy from init
init is now responsible for mounting the */selinux* filesystem, and then loads the policy after that.

SIDs and PSIDs no longer used
SIDs (Security Identifiers) were used in the old SE Linux in the interface to the kernel (from userspace). PSIDs (Persistent SIDs) were used in the kernel code for mapping files to contexts for files and directories on disk. See the NSA's document Configuring the SELinux Policy" document for more information. In the new SE Linux, the extended attributes contain the context so SIDs and PSIDs are no longer necessary.

-Z shortcut option
-Z can be used instead of typing --context after a command such as ls or ps.

No chsid command: uses chcon instead
The *chsid* command was used in the old SE Linux to change the context of a file. The new SE Linux uses the *chcon* command which changes the context of a file. *chcon* was available in the old SE Linux but has been improved for the new SE Linux, with options for setting the user or type. See the manpage for more details.

1.4. policy source directory for Fedora users

On Debian, the policy source directory is */etc/selinux*. On Fedora it is */etc/security/selinux/src/policy*. In this document I refer to the Debian policy source directory, so if you're a Fedora user, substitute */etc/selinux* with */etc/security/selinux/src/policy*.

---

2. Overview

Following is a short discussion on why you should consider using SE Linux and how it fundamentally works. Section 2.2 defines terms that will frequently be used in subsequent sections, so get familiar with them now.

2.1 Why SE Linux?

SE Linux offers greater security for your system. Users can be assigned predefined roles so that they can not access files or processes that they do not own. There is no "chmod 777" equivalent operation. This differs from regular Unix permissions in that the user defined roles, or security contexts they are placed in, have limited access to files and other resources but in a far more controlled fashion. Take a user's *.rhosts* file on a regular Unix system. If they make it world writeable then anyone can login and do lots of damage. Under SE Linux, you can control whether or not the user has the ability to change the permissions on their *.rhosts* file, and also prevent other people from writing to it even after the owner has made it world writeable.

A common question is how SE Linux permissions relate to standard Unix permissions. When you do a certain operation, the Unix permissions are checked first. If they allow your operation then SE Linux will check next and allow or deny as appropriate. But if the Unix permissions don't let you do something, the requested operation stops there and the SE Linux checks aren't performed.

Another example is if there was an exploitable bug in */usr/bin/passwd* which could run *chmod 666 /etc/shadow* SE Linux permissions would still prevent anyone from inappropriately accessing the file.

Rather than regurgitate what is written in the NSA's FAQ, I'll point you straight there. Reading the white papers the NSA has published is a must.

2.2 Terminology used

The following terms will appear frequently in this document, and form core concepts of SE Linux. It is somewhat tricky to define one word without including the other terms so I realise my definitions include things that need defining ;)

2.2.1 *identity*

An identity under SE Linux is not the same as the traditional Unix uid (user id). They can coexist together on the same system, but are quite different. identities under SE Linux form part of a security context which will affect what domains can be entered, i.e. what essentially can be done. An SE Linux identity and a standard Unix login name may have the same textual representation (and in most cases they do), however it is important to understand that they are two different things. Running the *su* command does not change the user identity under SE Linux.

Example:
An unprivileged user with the login name faye runs the *id* command (under SE Linux) and sees the security context of

```
context=faye:user_r:user_t
```

The identity portion of the security context in this case is "faye". Now, if faye su's to root and runs *id*, she will see the security context is still

```
context=faye:user_r:user_t
```

so the identity remains the same, and has not changed to root. However, if identity faye has been granted access to enter the sysadm_r role and does so (with the *newrole -r* command which will be covered later), and runs *id* command again, she will now see

```
context=faye:sysadm_r:sysadm_t
```

So the identity remains the same but the role and domain (second and third fields respectively) have changed. Maintaining the identity in this manner is useful where user accountability is required. It is also crucial to system security in that the user identity will determine what roles and domains can be used.

2.2.2 *domain*

Every process runs in a domain. A domain directly determines the access a process has. A domain is basically a list of what processes can do, or what actions a process can perform on different types. Think of a domain like a standard Unix uid. Say root has a program and does a *chmod 4777* on that program (making it setuid root). Anyone on the system, even the nobody user, can run this program as root thereby creating a security issue. With SE Linux however, if you have a process which triggers a domain transition to a privileged domain, if the role of the process is not authorised to enter a particular domain, then the program can't be run.

Some examples of domains are sysadm_t which is the system administration domain, and user_t which is the general unprivileged user domain. *init* runs in the init_t domain, and *named* runs in the named_t domain.

2.2.3 *type*

A type is assigned to an object and determines who gets to access that object. The definition for domain is roughly the same, except a domain applies to process and a type applies to objects such as directories, files, sockets, etc.

2.2.4 *role*

A role determines what domains can be used. The domains that a user role can access are predefined in policy configuration files. If a role is not authorised to enter a domain (in the policy database), then it will be denied.

Example:
In order to allow a user from the user_t domain (the unprivileged user domain) to execute the *passwd* command, the following is specified in the relevant config file:

```
role user_r types user_passwd_t
```

It shows that a user in the user role (user_r) is allowed to enter the user_passwd_t domain i.e. they can run the *passwd* command.

2.2.5 *security context*

A security context has all the attributes that are associated with things like files, directories, processes, TCP sockets and so forth. A security context is made up of the identity, role and domain or type. You can check your own current security context by running *id* under SE Linux.

There is a very important distinction which needs to be made here, between a domain and a type, as it tends to cause a little confusion later on if you don't understand it from the start.

Processes have a domain. When you check the security context of a process (for an example, see the explanation of "transition", below), the final field is the domain such as user_passwd_t (if you were running the passwd command).

Objects such as files, directories, sockets etc have types. When you use the *ls --context* command on a file for instance, the final field is the type, such as user_home_t for a file created in the home directory of a user in the user_r role.

Here's where a little confusion can creep in, and that's whether something is a domain or a type. Consider the /proc filesystem. Every process has a domain, and /proc has directories for each process. Each process has a label, or rather, a security context applied to a file. But in the /proc world, the label contains a type, and not a domain. Even though /proc is representing running processes, the entries under /proc are considered files and therefore have a type instead of a domain.

Running *ls --context /proc* shows the following listing for the init process (with a process id of 1):

```
dr-xr-xr-x  root     root     system_u:system_r:init_t         1
```

The label, or security context, shows that this file has a type of init_t However it also means that the init process is running in the init_t domain. Each file or directory under /proc that has a process id for a filename will follow this convention, i.e. the type listed for that process in the output of a *ls --context* command will also be the domain that process is running in.

Another thing to note is that commands such as *chsid* (change the security id) and *chcon* (change context) don't work on /proc as /proc does not support changing of labels.

The security context of a file, for example, can vary depending on the domain that creates it. By default, a new file or directory inherits the same type as its parent directory, however you can have policies set up to do otherwise.

Example:
user faye creates a file named test in her home directory. She then runs the command *ls --context test* and sees

```
-rw-r--r--  faye     faye     faye:object_r:user_home_t        test
```

She then creates a file in /tmp called tmptest and runs the command *ls --context /tmp/tmptest* This time, the result is

```
-rw-r--r--  faye     faye     faye:object_r:user_tmp_t       /tmp/tmptest
```

In the first example, the security context includes the type "user_home_t" which is the default type for the home directory of an unprivileged user in the user_r role. After running the second *ls --context* command, you can see that the type is user_tmp_t which is the default type that is used for files created by a user_t process, in a directory with a tmp_t type.

2.2.6 *transition*

br>A transition decision, also referred to as a labelling decision, determines which security context will be assigned for a requested operation. There are two main types of transition. Firstly, there is a transition of process domains which is used when you execute a process of a specified type. Secondly, there is a transition of file type used when you create a file under a particular directory.

Example:
For the second type of transition (transition of file type), refer to the example give in the "security context" section above. When running the ls --context commands you can see what the file types are labelled as (i.e. user_home_t and user_tmp_t in the above examples). So, here you can see that when the user created a file in /tmp a transition to the user_tmp_t domain occurred and the new file has been labelled as such.

For transition of process domains, consider the following example. Run ssh as a non privileged user, or more specifically, from the user_t domain (remember you can use the id command to check your security context). Then run *ps ax --context* and note what is listed for ssh. Assuming user faye does this, she sees

```
faye:user_r:user_ssh_t
```

as part of the output listing. The ssh process is being run in the user_ssh_t domain because the executable is of type ssh_exec_t and the user_r has been granted access to the user_ssh_t domain.

2.2.7 *policy*

Policies are a set of rules governing things such as the roles a user has access to; which roles can enter which domains and which domains can access which types. You can edit your policy files according to how you want your system set up.

---

3. Installation

The following section will explain how to go about obtaining the packages for installation, and how to go about installing the packages for the new SE Linux. As I run Debian, I will base installation instructions on that. It is assumed you know how to install packages for your distribution, how to compile a kernel and how to apply kernel patches.

If you are upgrading from the old SE Linux to the new, and are running an SE Linux kernel, go in to permissive mode (with the avc_toggle

command) and follow these instructions.

3.1 Installing base packages for Debian

For Debian unstable:

Put the following in your */etc/apt/sources.list* file:

```
deb http://www.coker.com.au/newselinux/ ./
```

The packages for unstable are maintained by Russell Coker.

At the time of writing (late November 2003) no new SE Linux packages are available for Debian stable. The .deb files can be obtained from the site above. Be sure to get the latest versions of each one. I'm not including the full package name as they'll keep changing, but they begin with the following package names listed.

Packages needed for an install of the new SE Linux on Debian unstable are as follows. You don't need to boot with an SE Linux kernel before installing them, so you can install them now:

- **libselinux1**

  Contains the shared libraries for the new SE Linux.

- **selinux-policy-default**

  Contains sample policy files for lots of commonly used programs such as postfix, sendmail, X and so forth.

- **checkpolicy**

  Contains the security policy compiler.

- **policycoreutils**

  Contains core utilities such as setfiles, load_policy, newrole etc.

- **selinux-utils**

  Contains utilities for operations such as querying the policy.

- **selinux-doc**

  Contains some documentation.

Additional packages for Debian that you need are listed below.

- **kernel-patch-2.4-lsm**

  A kernel for the LSM hooks and SE Linux.

- **coreutils**

  The coreutils package contains modified versions of commands such as cp, mv, ls and so forth.

- **procps**

  The procps package contains modified versions of commands such as ps and top.

- **sysvinit**

  Contains a patch to load the policy upon boot.

- **dpkg**

  A modified dpkg is needed so that it will label the files correctly after a package is installed.

- **libpam-modules**

  Sometimes a program needs to access the passwd and shadow files for authentication. If such a program were to be compromised, an attacker could get the shadow file, run something like Crack on it and you're in trouble. With the modified libpam-modules, a program does not have to get direct access to the shadow file. Instead,it (the program that wants to authenticate a user) runs a helper program which compares a supplied password with */etc/shadow* and gives a return code of 1 or 0 to indicate whether or not there was a match. This means you have to crack the wrapper program and not the calling code in order to see the contents of the shadow file.

- **logrotate**

  Contains a modified version of the logrotate package which preserves the SE Linux security contexts on new files.

- **cron**

  A modified cron package is needed so that cron can run scripts in the correct domains, and check ownership of files to serve as entrypoint for the cron domain.

3.1.1 Modified Debian package management tools

Debian users should be familiar with the *dselect*, *apt-get* and *dpkg* commands. *se_dselect*, *se_apt-get* and *se_dpkg* are actually wrapper scripts to run the regular versions of dselect, apt-get and dpkg but with some additional extras. The same applies for *se_dpkg-reconfigure*. Why are modified versions of this needed? Because when packages are installed, the files need to be labelled correctly as well as the scripts having to be started in the right context.

When using these commands, you will be prompted for a password (the password of the identity you are using). Why? Take the *se_dselect* command for example. As sysadm_r:sysadm_t run *se_dselect*. Now, as sysadm_r:sysadm_t in another window, run the command *ps ax --context|grep dselect* and you'll see something like this:

```
 5292    404 system_u:system_r:dpkg_t                dselect
```

Notice the security context that dselect is now running in. You will be changing identity, role and domain so you need to be authenticated first.

3.2 Installing base packages for Fedora

RPMs for the new SE Linux may be found at ftp://people.redhat.com/dwalsh/SELinux

The RPMs at the above location are maintained by Dan Walsh.

When I installed SE Linux on my Fedora test machine, this is what I did:

* Edited the yum.conf file to contain the following:

```
[main]
cachedir=/var/cache/yum
debuglevel=2
logfile=/var/log/yum.log
pkgpolicy=newest
distroverpkg=fedora-release
tolerant=1
exactarch=1

[development]
name=Fedora Core $releasever - Development Tree
#baseurl=http://download.fedora.redhat.com/pub/fedora/linux/core/development/i386
baseurl=http://mirror.dulug.duke.edu/pub/fedora/linux/core/development/i386

[SELinux]
name=SELinux repository
baseurl=ftp://people.redhat.com/dwalsh/SELinux/Fedora
```

* Ran the command

```
yum install policy checkpolicy policycoreutils policy-sources pam passwd vixie-cron
```

* After all of the above was installed, I did

```
cd /etc/security/selinux/src/policy
make load
make relabel
```

* Rebooted the machine.

3.3 Installing SE Linux related packages.

Under the old SE Linux it was best to install things in a certain order, such as installing the modified login package first. With the new SE Linux, dependencies should take care of things.

3.3.1 Installing the LSM kernel image

The Debian *kernel-patch-2.4-lsm* package takes care of applying both the LSM patch for the new SE Linux. The patch contains LSM kernel hooks, and SE Linux is the code that uses these hooks.

Now read */usr/share/doc/kernel-patch-2.4-lsm/README.Debian* and follow the instructions for setting the CONFIG_ options when compiling your kernel. Then go ahead and compile your new kernel or use Debian's *make-kpkg* package to create a kernel image .deb that you can then install.

Below is an extract of */usr/share/doc/kernel-patch-2.4-lsm/README.Debian*:

```
This patch supplies the Linux Security Modules.  It is needed for NSA Security
Enhanced Linux (among other things).

To apply automaticaly, set PATCH_THE_KERNEL=YES before first running of
make-kpkg (from package: kernel-package) and "make-kpkg clean" to remove.

When configuring your kernel do the following:
(Under Networking Options, enable Network Packet Filtering.
Under Security Options, enable Capabilities and enable
both IP Networking and SELinux as built-in options.)

  This means having the following in your /usr/src/linux/.config:
  CONFIG_NETFILTER=y
  CONFIG_INET=y
  CONFIG_SECURITY=y
  CONFIG_SECURITY_CAPABILITIES=y
  CONFIG_SECURITY_SELINUX=y
  CONFIG_SECURITY_DTE=n
  CONFIG_SECURITY_OWLSM=n
  CONFIG_LIDS=n

  This release of SE Linux depends on XATTR's.  For the Ext3 file system
```

```
use the following settings:
CONFIG_EXT3_FS_XATTR=y
CONFIG_EXT3_FS_XATTR_SHARING=y
CONFIG_EXT3_FS_SECURITY=y

The options CONFIG_EXT3_FS_XATTR_USER and CONFIG_EXT3_FS_XATTR_TRUSTED are
not required for SE Linux, but do not do any harm either.
```

If you compile your kernel with CONFIG_SECURITY_SELINUX_DEVELOP turned on, your machine boots with permissive mode, and must manually be switched to enforcing mode. But if you compile without it, your machine boots in to enforcing mode with no option of going back to permissive mode. See Section 4.4: Permissive and Enforcing mode.

If you are using ext2 you will need to build your kernel with CONFIG_EXT2_FS_XATTR set to Y. If you have a ext3 filesystem and want to mount it and not create any xattr's, then compiling ext2 with no XATTR support is a good option. An ext filesystem can be mounted as either ext2 or ext3. The idea is that you can mount the ext3 filesystem with no xattr's as ext2 (just edit your /etc/fstab file accordingly).

3.3.2 Installing the selinux-policy-default package

This Debian package contains the default security policy files. The equivalent RPM is the **policy-sources** package.

When installing this package on a Debian system, you will be prompted to answer a series of questions about which policies you'd like to install. Basically it's up to you to determine what you do and don't need. If you accidentally answer No to something you think you may need, don't worry. At a later time you can copy it from */usr/share/selinux/policy/default/domains/program/* over to */etc/selinux/domains/program* and then run the command *make -C /etc/selinux load* from any directory.

A brief mention about the sendmail.te policy-- it is best to remove this as it conflicts with other mail server policy files. Unless you want to run sendmail of course, in which case you don't install the policy files for another mail server.

The prompts will look something like this:

```
Removal of unwanted policy files
Do you want domains/program/amavis.te:Amavis anti-virus
Yes/No/Display[Y/n/d]?
```

Selecting Y will install the amavis.te policy file. Selecting n will not install it (but you can copy it later as described above). Selecting d will display the policy file concerned.

When you have finished answering the prompts, the policy will then be compiled and the policies that you have the .te files for will be installed.

A crucial part of the installation occurs at this point. Every file will now be labelled with a security context.

3.3.3 Editing your */etc/fstab* file and creating the */selinux* mount point

Before rebooting, you must first edit your */etc/fstab* file and create the */selinux* mount point. So create the mount point of */selinux* and set the permissions to mode 500. Now edit your */etc/fstab* to include the following:

```
none            /selinux        selinuxfs       noauto  0       0
```

3.3.4 Running *make relabel*

If you are running a 2.6.x kernel with XATTR support enabled, after creating the */selinux* mount point and editing your */etc/fstab* you must now run the command *make -C /etc/selinux relabel* This command relabels the filesystem with the correct security contexts. Note that you must run the command again after rebooting (discussed further down). If you are running a 2.4.x kernel you can't run this command now, as the 2.4.x non-SE Linux kernel does not allow you to assign the extended attributes.

3.3.5 Editing */etc/pam.d/login* and */etc/pam.d/ssh*

Before rebooting we must edit the */etc/pam.d/login* and *ssh* files (*sshd* on Fedora) so that a shell will be started in the right context. Add the following:

```
session required        pam_selinux.so
```

to both */etc/pam.d/login* and */etc/pam.d/ssh*

3.3.6 Adding users

Before rebooting you can add a new SE Linux user with the command *useradd* and edit the *users* file, and you can also do it after you reboot. In this document we'll do the latter.

You are now ready to reboot your machine, so go ahead. As soon as you have booted in to a SE Linux kernel, you MUST relabel all file systems.

---

4. Logging In

The following sections describe logging in to the system, and explain a little more about user security contexts. The final section discusses permissive mode and enforcing mode.

4.1 Supplying a user context at login

By this stage, you should have rebooted and are (hopefully) at a login prompt. When you installed the *selinux-policy-default* package (or the *policy-sources* package on Fedora), policy files were installed to enable you to log in to the system with a default user role (as we haven't added a

user ourselves yet).

Login as root as you would normally do. Your security context will default to root:user_r:user_t. Type *id* and your security context should be similar to the following (we're looking at the security context portion so ignore the rest for now):

```
uid=0(root) gid=0(root) groups=0(root) context=root:user_r:user_t
```

So, in that line, the security context is

```
root:user_r:user_t
```

Now let's assume you have previously allowed your own account access to another role. This is covered in Section 5: Creating User Accounts. There are two ways of switching to this new role. The first is when you log in. Assume user faye is authorised to enter the sysadm_t domain. User faye logs in at the console. At the "Your default context is faye:user_r:user_t. Do you want to choose a different one? [n]" prompt, she hits y and presses enter. She'll see something like:

```
[1] faye:user_r:user_t
[2] faye:sysadm_r:sysadm_t
Enter number of choice:
```

In this example, you can see that user identity "faye" has previously been granted access to the sysadm_r role and sysadm_t domain. The only options that will be displayed are those that your user identity has been granted access to. Please note that this worked in the old SE Linux, and will be a configuration option in the new SE Linux (not available at the time of writing this), with the default set to Off.

If user faye selected option two (to become sysadm_r) and then ran the id command, she'd see the security context of

```
context=faye:sysadm_r:sysadm_t
```

which means she is now in the sysadm_r role.

The second method of changing security contexts follows.

4.2 Changing context with the *newrole -r* command

The second way of changing your security context is to use the *newrole -r* command. The proper syntax is

```
newrole -r role
```

where role is the role you want to become. So assume you want to become sysadm_r. You'd then use

```
newrole -r sysadm_r
```

You will be asked to supply your password for your user identity, which you can check with the *id* command. If you do not have authorisation to enter a new role, you will see something like (assuming user identity fred tries to run the command)

```
fred:sysadm_r:sysadm_t is not a valid context
```

This message means that fred can not enter the sysadm_r:sysadm_t role:domain because he has not been authorised to do so.

After successfully changing roles, run the id command to check your security context.

4.3 Running commands in the sysadm_t domain

You have to be in the sysadm_r role in order to run commands in sysadm_t domain. At this point we need to do a little tidying up after our install, so let's go to a virtual console session and log in as root. You won't be asked if you wish to change contexts.

I'll have to apologise here for some seemingly going-around-in-circles instructions. We haven't actually allowed the root user access to the sysadm_r role so far in this HOWTO so you're probably thinking hang on, root is only allowed access to user_r:user_t so how can we actually do sysadmin stuff? Well, we're running in permissive mode which logs stuff but doesn't actually enforce our security policies. Plus you can use the *newrole -r* command as explained above to change to the sysadm_r role. Running the *newrole* command is the way to do it. If you try to do something you're not authorised to do, you can get screenful after screenful of error messages which isn't fun.

So, become sysadm_r then run the id command to check you are in fact sysadm_r:sysadm_t.

Now we can have a little fun in the sysadm_r role. When we installed everything in Section 3, all files on the system were labelled with a type but the machine wasn't running SE Linux back then. So if a file was created *after the labelling process took place, but before the machine was rebooted with the SE Linux kernel* then that file will not have a type associated with it. Imagine files that may have been created during that shut down. They won't have a type associated with them. Also, consider this scenario. If you delete a file, then that file's inode number can be used for a new file that is created and this new file may have the type of the file that was deleted. This can be a real pain.

Consider the */etc/nologin* file. This file is created when the *shutdown* command is issued. If this file exists at boot time, only root will be allowed to log in. What if your startup scripts can't delete this file? If */etc/nologin* has the wrong label, the startup scripts might not be able to touch it which creates a little problem. If your root identity is configured to have a default role of sysadm_r upon login, then you can log in and delete this file, problem solved.

But what if you have configured your root identity to *not* get the sysadm_r role upon login? Your root identity context might be root:user_r:user_t in this case. But the user_t domain doesn't let you delete anything in */etc*. See the problem? You can log in as root, but not have the sysadm_r privileges to actually do anything.

Imagine again, in this scenario, that you have a user identity of your own, let's use the "faye" identity again. Identity faye is configured to become sysadm_r upon logging in. So identity faye can do all the sysadm_r stuff that the root identity (running as user_r role in user_t domain) can't. faye might have the power, but in this case the faye identity is powerless, as it won't be able to login due to the fact that */etc/nologin* exists and doesn't allow non-root users to log in.

This is why labelling files correctly is critical. Let's return to the part about files that were created after the labelling process but before the booting of

the SE Linux kernel. In order to fix this, we have to run

```
make -C /etc/selinux relabel
```

This command will ensure that all the files on your system are labelled correctly. Depending on how fast your machine is and how many files you have, this may take a while. As a rough estimate, it will take as long as a *find /* command. This is why you want to use the newrole command to change to sysadm_r and then run the make command above-- if you're in a domain (like user_t) that doesn't allow access to other domains, you will get tens of thousands of lines of "permission denied" types of messages scrolling. Urgh.

4.4 Permissive and Enforcing mode

Permissive mode is when your SE Linux machine is essentially logging SE Linux related messages, but not much else (so you can still do the same stuff you'd do as root on a non-SE Linux machine). **Enforcing mode** kicks in all your policies such as denying access. Basically enforcing mode does what you have configured SE Linux policies to do. Permissive mode is good for debugging as you can check out the messages that get logged (check the output of the *dmesg* command).

A word of caution is needed here: Only boot to enforcing mode when you're sure everything is working properly! Do this by running in permissive mode for a while. Permissive mode assigns labels to files and so forth, but doesn't actually enforce anything, instead everything is logged. Some people compile a kernel with no CONFIG_SECURITY_SELINUX_DEVELOP support which means you can't specify permissive mode at al.

To switch between permissive and enforcing mode, you need to run the command *echo "1" > /etc/selinux/enforce* to turn on enforcing mode. Substitute 1 with 0 to run in permissive mode. The old SE Linux used the *avc_toggle* command which is not in the new SE Linux. Simply cat */etc/selinux/enforce* to see which mode you're running in (the old SE Linux used the command *avc_enforcing* to do this).

See "Section 9: Explanation of common log messages" for an example of the message logged when you switch modes.

If you compile your kernel with development mode turned on (meaning your machine boots to permissive mode but has to manually be swtiched to enforcing mode), you can switch to enforcing mode at boot time by creating a startup script, or by passing the parameter *enforcing=1* to your kernel during boot (such as editing your lilo.conf to include *append="enforcing=1"* for instance).

4.5 Comparison of running commands in different user roles

We'll now run some commands when in different security contexts. Switch to enforcing mode. In your user_r role, run the command *ps ax --context* and observe the output. Don't forget that *ps ax -Z* does the same thing. When in user_r role, you will see processes that are in the user_t domain, as well as any other domains that allow that access, that is read access to the directories under */proc*. If you did not have read access to the */proc* subdirectories, then the process would not be shown in the *ps ax* output.

Now, in the sysadm_t domain, run the *ps ax --context* command. This time, you will see all processes on the system regardless of the domain they are in. When in sysadm_t domain, you have access to other domains which the user_t domain does not. This is why, when in user_t domain, you can not see every process on the system. Imagine a malicious user being able to see all system processes. She notices a daemon running which has a known security hole that she can exploit. If the user_t domain can not see daemon processes, the risks of a system user exploiting this hole are greatly reduced if they can't see it in the first place.

Another thing to consider is the security risk of programs that take passwords on the command line. In a default Linux setup such passwords are available for all users to read. When SE Linux prevents you from seeing a process in the *ps* output, it reduces the risk of such things (of course a password on the command line is still a bad idea).

Change back to permissive mode. You will now be able to see all system processes if you do the *ps ax* command from the user_t domain.

---

5. Creating user accounts

Now for the fun stuff! We will create an SE Linux user and assign them a role and then set a default security context for users. Under the old SE Linux, wrapper programs existed for programs like vipw (svipw was the command to use), useradd (useradd), passwd (spasswd), chfn (schfn) etc. Under the new SE Linux, these programs have their regular names (i.e. not svipw etc etc).

5.1 Creating a new user

We'll now create our new user. Let's call him setest.

Switch to the sysadm_r:sysadm_t role:domain. Now use the *useradd* command to add user setest:

```
root@kaos:~# id
uid=0(root) gid=0(root) groups=0(root) context=faye:sysadm_r:sysadm_t sid=398
```

Run id as above to check that your uid is 0 and that you are in the sysadm_r:sysadm_t role:domain. If your uid is that of your regular account, then from your regular account *su* to root first, then run the *newrole -r* command.

```
root@kaos:~# useradd -c "SE Linux test user" -m -d /home/setest -g users -s /bin/bash -u 1005 setest
root@kaos:~# finger setest
Login: setest                           Name: SE Linux test user
Directory: /home/setest                 Shell: /bin/bash
Never logged in.
No mail.
No Plan.
root@kaos:~# passwd setest
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

The setest user has now been added.

5.2 Assigning roles to users and applying the changes

Now we want to assign a role to user setest. Let's say we want him to have access to the user_r role. The configuration file concerned with this is */etc/selinux/users* so open it up with your favourite editor and take a quick read through it.

At the end of the file, add the following line:

```
user setest roles { user_r };
```

This line means that user setest is authorised to enter the user_r role. If you want user setest to also have access to the sysadm_r role as well, you would instead add

```
user setest roles { user_r sysadm_r };
```

We now have to apply the changes we made to the */etc/selinux/users* file. To do this, run the following command when in sysadm_r:sysadm_t role:domain:

```
make -C /etc/selinux load
```

This takes a little while as a policy database is being created then compressed with *gzip*. When the command finishes executing, you will see something like this towards the end:

```
Success
touch tmp/load
make: Leaving directory '/usr/share/selinux/policy/current'
```

With a default role of user_r, the user does not specifically have to be added to */etc/selinux/users*. Only add them to this file if you want them to: have access to a user role other than user_r; be able to change their own password, or to have SE Linux log messages that contain their username where applicable.

We now have to define a default security context.

5.3 Setting the default security context for users

After adding the new user to */etc/selinux/users*, a default security context must be assigned to their login sessions. The configuration file for this is */etc/security/default_context* so take a look at it. You'll see the following line:

```
system_r:local_login_t  user_r:user_t
```

When a user logs in locally (i.e. at the console), the */bin/login* program will run in the domain local_login_t and will then assign a user role and domain of user_r and user_t respectively. If the line above was actually

```
system_r:local_login_t  sysadm_r:sysadm_t user_r:user_t
```

and the user logging in is authorised to enter the sysadm_t domain, then they will land in the sysadm_t domain upon logging in. If not, the user_t domain will be used.

Look at the line

```
system_r:sshd_t         user_r:user_t
```

This means that for all logins done via ssh, the user will land in the user_r:user_t role:domain.

5.4 Relabelling the user's home directory

If you have used *useradd* to add a new user with the user_r role, then the relabelling will have been taken care of. If however, you've used something like *svipw* to create the user, or the user role was *not* user_r, then no relabelling has taken place and you will have to run the following command:

```
find /home/setest -print0 | xargs -0 chcon -h system_u:object_r:user_home_t ; chcon -h system_u:object_r:user_home_dir_t /home/setest
```

This command takes all the files in */home/setest* and runs the *chcon* (change file security context) command on them. The user's home directory is assigned the type user_home_dir_t and files underneath the user's home directory are assigned the type user_home_t. Sometimes, a process may be granted access to a user's home directory, but not to any files or directories below, hence the two different types.

---

6. Adding a new user domain

Let's now create a user domain of our own, and call it second_t . We'll also create a new role called second_r . Create the second_r role first following the steps in the previous section (which just assigned the user_r role and did not actually create it) but don't run the make command in Section 5.2. Instead, after you've edited */etc/selinux/users* come back to this part and read the next section on editing the user domains file.

The reason why I don't want you to run the make command is because the previous section just assigned a role of user_r which is the default. But we will be creating a new role, and as such we need a new domain to go with it as outlined in the next few sections.

6.1 Editing the user domains file

The configuration file concerned with user domains is */etc/selinux/domains/user.te* . Have a read through it. Now add the lines

```
full_user_role(second)
allow system_r second_r
allow sysadm_r second_r
```

It doesn't matter where you add them, towards the top is okay. Take note of this comment:

```
# if adding new user roles make sure you edit the in_user_role macro in
# macros/user_macros.te to match
```

So now we edit */etc/selinux/macros/user_macros.te* to match. Open that file for editing and search for the string in_user_role (it's near the end of the file). Add in "role second_r types $1;" so this portion of the file now looks like

```
undefine(‘in_user_role’)
define(‘in_user_role’, ‘
role user_r types $1;
role second_r types $1;
’)
```

Going back to the first line of configuration code at the start of this section (full_user_role(second)), this creates the domain second_t and the types second_home_dir_t and second_home_t (for the home directory and files under the home directory respectively). A type of second_tmp_t is created for files created under /tmp. Type second_tmpfs_t is created for shared memory created when in the tmpfs context. Finally, types of second_tty_device_t and second_devpts_t are created for user labelling of tty devices and pseudo tty devices respectively. It also creates the basic policy rules for using these types.

SE Linux does not internally support any type of object orientation, inheritance of domains/types, etc. Also there is currently no policy language that supports such features (one could be written, but nobody has done so yet). So to get the features we need for easily creating new domains, we use M4 macros.

We'll now create yet another user for use in this new domain (second_t) and access to the second_r role.

6.2 Creating a new test user (again)

Using *useradd*, create a new user (let's say the new user is called "spike"). Add spike to */etc/selinux/users* with an entry that only gives him access to the second_r role and no others. Then run

```
make -C /etc/selinux load
```

to apply the new policy.

The next thing to do is set the default domain for the new role. To do this, we edit the file */etc/security/default_type* and add the line

```
second_r:second_t
```

We now have to manually relabel /home/spike and its contents. The *useradd* command did not do this, as it only supports relabelling for user_r roles. Run the following command:

```
find /home/spike -print0| xargs -0 chcon -h system_u:object_r:second_home_t ; chcon -h system_u:object_r:second_home_dir_t /home/spike
```

Now try logging in as spike.

---

7. Explanation of log file messages

The following are examples of logged messages. I'll explain what each part of a log message means. For easier reading, I've split the log messages across lines.

Sometimes the logs aren't as clear as you would like, so it's handy to know that for ReiserFS and Ext2/Ext3 (the file systems supported by SE Linux) the root inode is number 2.

The XFS and JFS file systems have not been thoroughly tested at this time.

Example 1

```
avc:  denied  { getattr } for  pid=6011 exe=/usr/bin/vim path=/etc/shadow dev=03:03 ino=123456 \

scontext=faye:user_r:user_t tcontext=system_u:object_r:shadow_t tclass=file
```

In this example, an unprivileged user (faye) attempted to edit */etc/shadow* when the system was in enforcing mode.

The "avc: denied" means that the operation was refused.
The "{ getattr }" means that someone tried to stat() the file. In this case, the file's attributes were looked up first (or at least, the operation tried to look them up), couldn't get those attributes and gave up.

The contents of the braces {} contain the operation or operations that were relevant to what SE Linux was doing. SE Linux can audit both allow and deny events, and in this case was auditing a deny and as such, tells you want was denied.

"for pid=" is the process id of your operation.
"exe=/usr/bin/vim" is the command you executed (in this case, vim).
"path=/etc/shadow" is the path to the object you tried to perform an operation on.
"dev=03:03" is the device number of the block device used for the file system concerned. So the first "03" means hda and the second "03" is 3, so this "dev=03:03" refers to /dev/hda3 (or if you're running devfs /dev/ide/host0/bus0/target0/lun0/part3). When SE Linux is auditing permissions it doesn't know the full path of the object you're trying to perform an operation on so it can't log anything but the device that you mounted and the location within the device. All it knows is the path relative to the file system, and the block device number for the file system. Say you access /etc/shadow. SE Linux doesn't know this file is in the root file system. All it knows is that the file is /etc/shadow within the file system it is working on.
"ino=123456" is the inode number of the object (in this case /etc/shadow)
"scontext=faye:user_r:user_t" is the source context of the process performing the operation.
"tcontext=system_u:object_r:shadow_t" is the security context of the target object (/etc/shadow).
"tclass=file" means that the target object is a file.

Example 2

```
avc:  granted  { avc_toggle } for  pid=6073 exe=/sbin/avc_toggle \
scontext=faye:sysadm_r:sysadm_t tcontext=system_u:system_r:kernel_t tclass=system
```

The "avc: granted" means that your operation was accepted and executed.
The "{ avc_toggle }" means that a program called the avc_toggle() system call.
The "tclass=system" means that the target process belongs to the system class.

Example 3

```
avc:  denied  { append } for  pid=6153 exe=/bin/bash path=/.bash_history dev=03:03 ino=498 \
scontext=faye:user_r:user_t tcontext=faye:object_r:root_t tclass=file
```

This message means that idenity faye in the user_r:user_t role:domain tried to append to root's *.bash_history* file which is of type root_t, and was denied.

Example 4

```
avc:  denied  { write } for  pid=605 exe=/bin/touch dev=09:03 ino=2 \
scontext=root:user_r:user_t tcontext=system_u:object_r:root_t tclass=dir
```

In this example, notice that the path is missing. However, we can tell it is the root directory because of the inode number which is 2.

---

8. Resources

Below are links to more SE Linux related material. Please read the NSA white papers.

NSA official site

The NSA's official SE Linux site is at http://www.nsa.gov/selinux

The official SE Linux FAQ is at http://www.nsa.gov/selinux/info/faq.cfm

NSA published papers, technical reports and presentations can be found at http://www.nsa.gov/selinux/info/docs.cfm

Mailing List and Archives

The NSA run a mailing list for discussion of all things SE Linux. Follow the instructions at http://www.nsa.gov/selinux/list.html to subscribe. The same URL describes how to obtain list archives.

Sourceforge project SE Linux

The SE Linux project at Sourceforge is located at http:/sourceforge.net/projects/selinux.
