SE Linux Tutorial for Linux Kongress

Russell Coker <russell@coker.com.au>,
http://www.coker.com.au/

1 Introduction

The aim of this tutorial is an advanced hands-on training in administering NSA Security Enhanced Linux [1]. The Debian distribution is used because it's support for SE Linux is better than any other distribution, and because I did most of the Debian development work regarding SE Linux and know it well. Most of the material covered here should apply to other distributions when they have support for SE Linux.

To install a package on a Debian system the command *dpkg -i package.deb* is used.

At the start of this tutorial you will have a workstation with a root password of "1234" and a set of Debian packages in the */root/pkg* directory that can be installed. Also in the */usr/src* directory there will be a complete archive of kernel source.

2 Kernel Build

The first stage of installing SE Linux is to build a kernel with support for it. This involves firstly applying the *LSM* kernel patch from http://lsm.immunix.org/ which includes the SE Linux patch. For Debian this patch is in the *kernel-patch-2.4-lsm* and *kernel-patch-2.5-lsm* packages for the 2.4.x and 2.5.x kernels respectively.

Install the package *kernel-patch-2.4-lsm*.

In the /usr/src directory you will find the file *linux-2.4.19.tar.bz2*. Extract it with the command *tar xvjf linux-2.4.19.tar.bz2*. Now change to the *linux-2.4.19* directory that has been created and apply the kernel patch with the command */usr/src/kernel-patches/all/apply/lsm*.

After the patch is applied run the command *make menuconfig* to configure the kernel. There is a new section *Security options* which contains the options for SE Linux and other security options. For SE Linux turn on the options for *Capabilities Support* (which SE Linux requires), *NSA SELinux Support*, and *NSA SELinux Development Module*.

If a kernel is compiled with Development support then it boots into *permissive mode* by default, which means that it writes log messages instead of preventing operations. This is essential when you first setup SE Linux, otherwise a mistake in the policy could render the machine unbootable.

We will not compile the kernel now due to lack of time.

3 First Stage of Installation (kernel and policy)

The first stage of installing SE Linux is to install the *login* program from the package *login_20000902-12.se1_i386.deb* which is needed to assign the correct security context to the user when they login.

After this is installed then install the kernel image from *kernel-image-2.4.19lsm_1_i386.deb*. But DO NOT REBOOT!

It is important not to reboot until all the other files are ready for SE operation to avoid the problems of a partially working SE setup.

The next packages to install are the *selinux* package that has the core SE Linux administrative programs, and the *selinux-policy-default* package which has sample security policy files. These are in *selinux_2002070313-9_i386.deb* and *selinux-policy-default_2002070313-9_all.deb* respectively, and must be installed in this order.

The *selinux-policy-default* package will ask you a number of questions about policy files to remove. Your workstation has a fairly minimal configuration at the moment, so you can remove any policy files that don't interest you (apart from *named.te* and *irc.te* which we need for later exercises), later in the tutorial you may install other packages so you may want to leave the policy files in place for such packages). Also please note that the *sendmail.te* policy conflicts with that for other mail servers so it must be removed.

If you accidentally remove a *.te* file that you need, then later you can copy it to */etc/selinux/domains/program/* from */usr/share/selinux/policy/default/domains/program/* and run "*make -C /etc/selinux load*".

After you have finished deciding which policy files to remove the package will then compile and install the policy. Then it will label every file on the file system with a security *type*. The core of SE Linux is *Domain Type* access control (known as *DT*). DT relies on every object (file, directory, network port, etc) having a security type associated with it, and every process being in a security *domain*. So the labelling of all the files on the file system is a very important part of the installation!

The final package to install in this phase is dpkg (the Debian packge manager). The package contains a modified version of dpkg that will label files with the correct SE Linux *type* after installing them.

Now reboot the machine to use the new kernel.

4 Second Stage of Installation

In the previous section you installed the bare minimum SE Linux functionality, the SE kernel support and a policy for the kernel to load. The policy package relabelled all files on the file system with correct types for you. You also installed a login program to allow you to login with the correct context, and a modified version of *dpkg* to allow you to install programs in the correct context.

Now login to the machine, after entering your root password you will see the following:

```
lyta login: root
Password:
Last login: Fri Aug 16 03:38:02 2002 on vc/2
Linux server 2.4.19lsm #1 Tue Aug 6 14:53:07 CEST 2002 i686 unknown unknown GNU/Linux
```

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

Your default security context is root:user_r:user_t
Do you want to enter a new security context?[n]
```

Now this means that it will log you in with the *context* of *root:user_r:user_t* by default.

The security context is comprised of three parts, the first part is the *identity* which is the Unix username that you used to login. The next part is the *role* which is one of the roles that are assigned to the identity (an identity can have one role or multiple roles), role names customarily end in *_r*. The final part of the security context is the *domain*. The domain name ends in *_t*, it is the determining factor for all SE Linux security decisions. Each role is limited to a certain set of domains.

Therefore your *identity* (your Unix username) limits your choice of *role*, which determines which *domain* you can use, and therefore what access you get to the system.

Now the security context of *root:user_r:user_t* is not what you want, you want to be able to continue doing administration work and configuring all aspects of the system, the *user_t* domain is very limited. This means that you need to type "y" at the prompt to enter a new security context, giving the following result:

```
Your default security context is root:user_r:user_t
Do you want to enter a new security context?[n] y

        Enter role  (return for default) [user_r]
```

At this time you must type "sysadm_r" to select the system administration domain giving the following:

```
Do you want to enter a new security context?[n] y

        Enter role  (return for default) [user_r] sysadm_r
        Enter type  (return for default) [sysadm_t]
```

Now you have to press ENTER to select the default domain of *sysadm^*and you get a root shell with full administrative privileges.

However there is a problem! At the time we labelled the files on the file system the machine was running without SE Linux, so any files created after the file system was labelled but before we booted a SE kernel will not have a type associated with them (that means any files created during the shut-down process). To fix this we have to run "make -C /etc/selinux relabel".

Now you want put SE Linux in enforcing mode. The program *avc_toggle* is used to switch between *permissive* mode (where messages are logged and nothing else happens), and *enforcing* mode (where actions prohibited by policy are denied. Run *avc_toggle* now to put the machine in enforcing mode, and notice that it informs you of the new state by printing *enforcing* on the command line. Also a message such as the following will be logged by the kernel and appear on your screen:

avc: granted avc_toggle for pid=16153 exe=/usr/bin/avc_toggle scontext=root:sysadm_r:sysadm_t tcontext=system_u:system_r:kernel_t tclass=system

Now go to another virtual console and login as *root* with the default security context of *root:user_r:user_t*.

Notice that in the session where you are logged in with the *sysadm_t* domain you can run "*ls -l root*", *dmesg*, and "*ps aux*" and get the results that you are used to. Now compare the results of the same commands in the session that is logged in with the *user_t* domain.

When logged in as a non-administrative domain the *ps* command will only show you processes in the same domain and in certain other domains that you have access to. You can not see daemon processes, so if the system is running a daemon that has a security hole a hostile user will be less able to discover it's existence or know how to attack it. Also the root home directory is configured such that non-administrative users can not write to it (otherwise they could replace the *.bashrc* with a script that does something nasty). Finally *dmesg* is denied from most users so that they can't see details of system hardware, and when their actions are blocked by the security policy they can't see why.

Now you have verified that SE Linux is working, try running *avc_toggle* again and notice that the machine has re-entered *permissive* mode, this means that the login with the *user_t* domain can see all processes with *ps*, run *dmesg*, etc. After verifying this run *avc_toggle* once more to leave the machine in *enforcing* mode.

At any time if you need to know the mode the machine is in you can run the program *avc_enforcing*, and it will tell you. Try it now.

The next time you boot the machine you want it to run in enforcing mode, to do this run the command "*ln -s /sbin/avc_toggle /etc/rc.boot*" to make *avc_toggle* be run at boot time.

For the purposes of rest of the tutorial make sure you leave the machine in enforcing mode.

5 Installing Utility Programs

Now you have a machine that has every process running in a security *domain* and every file and directory has a *type* associated with it. But you are unable to determine what the domains and types are!

To do this you need to install some new packages, however the regular *dpkg* command will not work when SE Linux is in enforcing mode. There is a wrapper called *se_dpkg* which sets the correct security context that you can use instead. These are *shellutils* for a modified version of *id* and adds the *runas* command, *procps* for a modified *ps*, and *fileutils* adds the *chcon* command and has modified versions of *cp*, *mv*, *install*, *ls*, *mkdir*, *mkfifo*, and *mknod*.

Install the packages *shellutils_2.0.12-2.se1_i386.deb*, *procps_2.0.7-10.se1_i386.deb*, and *fileutils_4.1.10-2.se1_i386.deb* now by using the *se_dpkg* command. This command will prompt you for the password matching the *identity* that you are using (in this case the root password).

Run the command *id* from both of your root login sessions, notice how it reports the different security context used for the processes in each session.

Run the command "*ls –context /*" and observe the different types being used for the files.

Run the command "*ps ax –context*" from the *sysadm_t* login and observe the different security contexts being used for the daemons.

Run the command "*ls –context 'tty'*" from both sessions and observe that the tty devices have a different type for each login. This prevents unauthorised access to a tty. To test this use the *wall* command, note

that when you run *wall* as *sysadm_t* all users see it, but when you run it as *user_t* only other users in *user_t* see it.

Now login as *user_r* on another console and run *irc* to connect to the IRC server. Go to the other console and run "*ps ax –context*" and note how the IRC program runs in the *user_irc_t* domain, now run "*ls -atr –context*" to see how the file that the IRC client creates has a different context. This is so that the IRC client has no access to files or directories under your home directory, it can create and modify it's own configuration files and nothing else. Then if someone cracks the IRC client it can't kill any programs you run, or read or modify any files you own (apart from it's own configuration). After writing this policy I spent some time in IRC as *root* and had some amusing discussions with IRC users who seemed to think it was a security risk and could not be convinced otherwise.

Now join the #se channel, we will use that for distributing policy samples, and for some discussion during the tutorial. Also if you wish to discuss in German with your friends please join the #de channel as well (unfortunately I can't read German so I won't be on that channel).

6 Installing Modified System Programs

Now that you have installed the utility programs you next have to install modified versions of some system programs. Two important programs are *cron* and *logrotate*.

We need a modified version of *cron* to run the user cron jobs in the correct context, and it has to check the *type* of the crontab file to ensure that it hasn't been created by someone who is not authorised for that cron domain.

We need a modified *logrotate* so that when it creates a new log file it gives it the same *type* as the old file that has been renamed.

Install the packages *cron_3.0pl1-72.se3_i386.deb* and *logrotate_3.5.9-10.se1_i386.deb* now.

7 Creating User Accounts

On a SE Linux machine you may want to have users with a UID=0 (root users) who are not permitted to use the *sysadm_r* for administrative access. To keep them out they have to be prevented from accessing */etc/shadow*, otherwise they could run crack on the passwords.

This of course requires a mechanism for changing the password. There are wrappers for the programs *passwd*, *chsh*, *chfn*, *vipw*, and *useradd*, these are *spasswd*, *schsh*, *schfn*, *svipw*, and *suseradd* respectively. The main wrapper programs *spasswd*, *schsh*, and *schfn* do not allow specifying any parameters (so you can't change the password, shell, or finger details for anyone else's account). The *svipw*, and *suseradd* wrappers allow specifying parameters but can only be run by the administrator in domain *sysadm_t*. If you are logged in as root with administrative privileges then you can change someone else's password with *sadminpaswd*.

Now change the root password with *spasswd*, create a new user for your own use with *suseradd*, and edit the user's details with *svipw*.

After adding the new user you need to assign them to a role (or roles). To do this firstly edit the file */etc/selinux/users*. If you have an account named *john* that you want to allow for the *user_r* role then you

add "*user john roles user_r;*" to the file. If you want to grant access to the role *sysadm_r* as well then add "*user john roles { user_r sysadm_r };*". After saving the changes apply them with the command "*make -C /etc/selinux load*". Note that the default role is *user_r*, if a user is not permitted any other role then there is no need to specifically add them to the policy. However adding the user to the policy allows syslog messages about denied actions to include the user name (which makes it easier to determine which user is doing things you don't like), so it's handy to have.

After adding an entry for a user in the *users* file you must set them a default security context for their login sessions, this is determined by the file */etc/security/default_context*. In the case of an account *john* you would add *john:user_r:user_t* to the */etc/security/default_context* file.

The next thing to do is to relabel their home directory to the correct type through the following commands:

```
find /home/john | xargs chcon -h system\_u:object\_r:user\_home\_t
chcon -h system\_u:object\_r:user\_home\_dir\_t /home/john
```

Now install *ssh* from *ssh_3.4p1-2.se1_i386.deb*, create a new account named *test* that is only authorised for the *user_r* domain and tell the people on the computer next to yours the password and the IP address of your machine so that they can login.

Now login to each other's machines and have a look around. Try various methods of tracing what the other person is doing on your machine as *sysadm_t* while they (as *user_t*) can't see what you are deoing. Please play nice when logging into each other's machines (just in case they made a mistake).

8 Starting and Stopping Daemons

When a daemon is run it has to be run in the correct context to ensure that it gets access to the resources it needs, and that it is denied access to everything it shouldn't be accessing.

The daemon start scripts are run in the context *system_u:system_r:initrc_t*, and there are policy rules to cause a transition from the *initrc_t* domain to the domain the daemon runs in. Inspect the file */etc/selinux/domains/program/named.te* and notice that it has *daemon_domain(named)* on line 14. The *daemon_domain* macro is defined on line 932 of */etc/selinux/macros/global_macros.te* which calls the macro *daemon_base_domain* on line 901 of the same file. The relevant line is line 907 which calls the *domain_auto_trans* macro to setup an automatic domain transition. When the *initrc_t* domain executes a file of type *named_exec_t* it will be run in the new domain *named_t*.

For more information on SE Linux policy see Stephen D. Smalley's paper [2].

When logged in under the *sysadm_t* domain you can stop a daemon in the usual fashion. To start a daemon you need to use the *run_init* program to start it in the correct context, *run_init* will run a specified program in the *system_u:system_r:initrc_t* context. Stop the *named* daemon now with the command "*/etc/init.d/bind stop*", and restart it with the command "*run_init /etc/init.d/bind start*".

Now use the command "*ps ax –context — grep named*" to verify that it is running and in the correct domain.

9 Adding a New User Domain

To add a new user domain *second_t* and role *second_r* edit the file */etc/selinux/domains/user.te* add the following:

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

The first line creates the domain *second_t*, and the types *second_home_dir_t* and *second_home_t* (for the home directory and for files under the home directory respectively), *second_tmp_t* for files created under /tmp, *second_tmpfs_t* for shared memory created in the context of tmpfs, and *second_tty_device_t* and *second_devpts_t* for user labelling of tty devices and pseudo-tty devices respectively. It also creates all the basic policy rules for using these types.

The reason for having the types *second_home_dir_t* and *second_home_t* is so that the IRC client can be given access to user which is of type *second_home_dir_t* but not to user/Mail which is of type *second_home_t*.

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 no-one has done so yet). So to get the features we need for easily creating new domains etc we use *M4* macros.

Now create a new user for use in this domain with the *suseradd* program, add them to the */etc/selinux/users* file with an entry that gives them access to the role *second_r* (and no other roles), then run "*make -C /etc/selinux load*" to apply the new policy. Then run the following commands to apply the correct labels to the home directory:

```
find /home/user | xargs chcon -h system_u:object_r:second_home_t
chcon -h system_u:object_r:second_home_dir_t /home/user
```

The next thing to do is set the default domain for the new role, to do this edit the file */etc/security/default_type* and add the line "*second_r:second_t*". This is for the login prompt for the domain.

Now try logging in as the new user!

10 Setting up a Chroot Environment

Now we want to setup a chroot environment running the old Debian version named *Potato* from the *user_t* domain.

The administrator of the chroot environment can allow other people to login as root with administrative privileges (they can add accounts, change passwords etc), but they can't escape from the chroot. Also some of the files in the chroot environment can be set as read-only, so that the administrative user inside the chroot can't change then. This is both a protection against having the chroot environment "*cracked*", and a way of limiting what the root user inside the chroot can do.

Firstly add the line "*chroot(user, potato)*" to */etc/selinux/domains/user.te*. Then create the file */etc/selinux/domains/misc/custom.te* with the line "*file_type_auto_trans(user_t, src_t, potato_ro_t)*". Now load the policy.

Now login as *root* in the *user_r* role and extract the archive */usr/src/potato.tar.bz2* with the command "*cd /usr/src ; tar xvjf potato.tar.bz2*". This archive includes the script *mountit* as follows:

```
#!/bin/bash -e

if [ "$1" == "mount" ]; then
  cd $2
  mount -n none -t proc proc
  mount -n /dev dev --bind
fi
if [ "$1" == "umount" ]; then
  cd $2
  umount -n `pwd`/proc/
  umount -n `pwd`/dev
fi
```

Also the script *setperms* as follows:

```
#!/bin/bash -e

# the first parameter is the full path to the root directory
# the second parameter is the name of the chroot (the second parameter to
# the chroot() macro

echo cd $1
IDENTITY=`id -c | cut -f1 -d:`
BASECON=$IDENTITY:object_r:$2
find home root tmp var | xargs -l300 chcon ${BASECON}_rw_t
find etc/init.d | xargs -l300 chcon ${BASECON}_dropdown_t etc/init.d
chcon ${BASECON}_ro_t var var/lib
find /var/lib/dpkg | xargs -l300 chcon ${BASECON}_ro_t
chcon ${BASECON}_super_entry_t entry
```

The files in the chroot are currently labelled as *potato_ro_t* due to the "*file_type_auto_trans(user_t, src_t, potato_ro_t)*" which causes any files or directories created under a *src_t* directory by the *user_t* domain to be labelled as *potato_ro_t*.

So the before we try running anything we have to use the *setperms* script to label some of the files and directories as allowing writes, the command is "*/usr/src/potato/setperms /usr/src/potato potato*".

Now to get the chroot going you first have to mount */proc* and */mount* under the chroot with the *mountit* script, the command "*/usr/src/potato/mountit mount /usr/src/potato*" will do this.

Before you try logging in to the chroot environment you have to change the root password, the command "*cd /usr/src/potato ; chroot . passwd*" will do this.

Now we have the files setup for a chroot environment and a password in place, all we have to do is allow logins to it by starting a syslogd daemon and a ssh daemon. I have configured the ssh daemon to run on port 222 (so it doesn't conflict with the main sshd), so all you have to do is run "*cd /usr/src/potato ; chroot . /etc/init.d/sysklogd start*" to start the system log, and "*cd /usr/src/potato ; chroot . /etc/init.d/ssh start*" to start sshd.

References

[1] *Meeting Critical Security Objectives with Security-Enhanced Linux*
Peter A. Loscocco, NSA, loscocco@tycho.nsa.gov
Stephen D. Smalley, NAI Labs, ssmalley@nai.com
`http://www.nsa.gov/selinux/ottawa01-abs.html/`

[2] *Configuring the SELinux Policy*
Stephen D. Smalley, NAI Labs, ssmalley@nai.com
`http://www.nsa.gov/selinux/policy2-abs.html/`