Role-based access control in SELinux

Learn your way around this admin-friendly security administration layer

Serge E. Hallyn

February 13, 2008

Role-based access control (RBAC) is a general security model that simplifies administration by assigning roles to users and then assigning permissions to those roles. RBAC in Security-Enhanced Linux (SELinux) acts as a layer of abstraction between the user and the underlying type-enforcement (TE) model, which provides highly granular access control but is not geared for ease of management. Learn how the three pieces of an SELinux context (policy, kernel, and userspace) work together to enforce the RBAC and tie Linux® users into the TE policy.

The security policy implemented in Security-Enhanced Linux (SELinux) is type enforcement (TE) under a layer of role-based access control (RBAC). (SELinux also orthogonally implements multi-level security (MLS), which is outside the scope of this article.) TE is the most visible, and therefore the most well known, server because it enforces fine-grained permissions: when something breaks because of unexpected access denials, TE is most likely responsible. In TE, a process's security domain (its domain of influence over the system) is determined by the task's history and the currently executing program.

The concept of RBAC isn't discussed as often as TE and can be confusing because of the way it is integrated with TE. You generally think of RBAC as specifying the access that users in certain roles may receive. SELinux specifies the role-based access in terms of TE, however, so the goal of RBAC in SELinux is to allow management of privileges based on roles that the authorized user may assume, then restrict the domains of influence that a role may enter by specifying the TE domains with which a role may be combined into a valid context.

To see how this works, take a look at a very simple cash register accounting system using SELinux to provide the security guarantees. You'll see the same solution in two very different environments (see Downloadable resources to view the code for both):
Working with roles

Assume you've been asked by a department store to implement a secure register balance accounting system. For every cash register, the final amount must be counted by both the cashier and a manager. We first define two roles, cashier and manager. We assign roles to employees and require that both one cashier and one manager certify the amount in a register at the end of the cashier's shift.

While the policy files in the two systems may appear somewhat different, the same data layout and cash registering program is used in both systems. All data is stored under the directory /data and is accessed only through the program /bin/register.py. The program register.py can be used by both managers and cashiers to store values. To keep the code simple, it's lacking in features. A cashier can store cash register values only for himself for the day. A manager can store values for other employees and, when both the manager and cashier have stored the same value for the same cashier for the day, the manager can commit the values.

When cashier Bob stores a value using `register.py bob 109.95` at 9:00 p.m. on Dec 12, 2007, the file /data/cashier_r/bob/12-12-2007 is created with the contents "bob 09:00 109.95." Next, manager Mary stores the same value at 9:25pm, resulting in the file /data/mgr_r/bob/12-12-2007 being created with the contents "mary 09:25 109.95." Finally, Mary commits the values at 9:27 p.m. using the command `register.py bob commit`, which creates the file /data/final/bob/12-12-2007 with the values "mary 09:27 bob mary 109.95."

If Bob and Mary disagreed on their values, then Mary will be unable to commit the value for Bob. Mary will have to go talk to Bob, both will recount until they agree, then they can re-run the /bin/register.py commands above with identical values. The new values will be appended to previous values to facilitate later perusal by the store owner, and the /bin/register.py bob commit command will use the last entered values out of both /data/cashier_r/bob/12-12-2007 and /data/mgr_r/bob/12-12-2007.

Note: The code examples discussed here use SELinux for all the access-control needs. Our examples simply allow all users on the system full write access to all files and directories under /data. In a real deployment, however, you would want to implement defense in depth by also having some DAC permissions in effect. All managers may wish at some point to create files under /data/mgr_r/bob/ and /data/final/bob/, involving delicate use of UNIX® group permissions. But for simplicity, let's fully rely on SELinux to enforce access control.

First, prevent both managers and cashiers from accessing any files under /data except through the use of the register.py program. Bob, for instance, will log in to role cashier_r in type cashier_t. But cashier_t cannot read under /data. To do that, he must enter type cashier_register_t, which he can
enter only by executing /bin/register.py. Similarly, Mary will log in to role mgr_r in type mgr_t, but must execute /bin/register.py to enter mgr_register_t before having any access under /data.

The first bit of access control actually happens at login when a PAM module decides that Bob must log in to role cashier_r. It continues when the SELinux type enforcement server in the kernel refuses to allow bob_u:cashier_r:cashier_t to enter bob_u:cashier_r:cashier_register_t except by executing a file of type cashier_exec_t, a type that the administrator has assigned only to /bin/register.py.

Enforcement further continues in register.py itself when it refuses to allow a cashier to commit values or to store values for another user. This is further reinforced by the SELinux policy—and the kernel code's enforcement of the policy—which does not allow cashier_register_t to access files under /data/mgr_r or /data/final.

Implementing the cash-register system

The first implementation of the cash register accounting system is on top of the previously mentioned from-scratch SELinux system (available in Downloadable resources). Following are the steps to implement the cash register accounting system on top of a complete from-scratch system.

Mount the disk image (with the qemu image turned off) using:

```
mount -oloop,offset=32256 -t ext2 gentoo.img /mnt
```

Grab the code_for_fromscratch.tgz file from the Downloadable resources section, and untar it under the disk image by doing:

```
tar zxf selinuxregister.tgz -C /mnt
```

Now start the qemu image:

```
qemu -hda gentoo.img -m 512 -vnc :3 -kernel bzImage -append "ro root=/dev/hda1 -p"
```

Once you log in, you must compile and install the new policy and install the new PAM module:

```
cd /usr/src
checkpolicy -c 19 -o policy.bin policy.conf
cp policy.bin /etc/
rc-update add selinuxenforce default
cp /etc/pam.d/system-auth /etc/pam.d/system-auth.orig
cp /etc/pam.d/system-auth.new /etc/pam.d/system-auth
```

The SELinux users are created in policy, but you must create the Linux users that will correspond to them:
adduser mary
passwd mary
(mkdir /home/mary
adduser boss
passwd boss
(mkdir /home/boss
adduser bob
passwd bob
(mkdir /home/bob

Then create the directory structure for data storage:

mkdir /data
mkdir /data/cashier_r
mkdir /data/mgr_r
mkdir /data/final
chmod 777 /data/*

Finally, relabel the filesystem:

setfiles /usr/src/filecontexts /
poweroff

Now the image is ready. Restart it without the -p flag so that the SELinux policy is loaded:

qemu -hda gentoo.img -m 512 -vnc :3 -kernel bzImage \
-append "ro root=/dev/hda1"

Then log in as root, and try:

ls /data

Permission denied. Log out and log in as bob, our cashier. Register a value, for example:

register bob 25.22

Then try to cheat the system by doing:

register bob commit

Didn't work. Log out and log back in as Mary, our manager:

register bob commit

Ah, Mary first needs to enter her own value:

register bob 27
register bob commit

The values didn't match. Wonder what Bob committed?
cat /data/cashier_r/bob/(day)

Ah, you're not allowed to see that. You'll have to go and discuss this with Bob. Perhaps you recount your register and find that he was right. So:

```
register bob 25.22
register bob commit
```

This worked. Now you can log in as root and:

```
cat /data/final/bob/(day)
```

This shows all the values that have been entered.

**A closer look at type enforcement**

Different users using the same /bin/register program are able to read and write different files that they cannot access without the program. This is one of the core concepts of type enforcement: both the authorized context of the user and the code being executed should together determine the resulting process's "domain of influence" over the system (or TE domain).

Figure 1 shows the domains and types on our system:

**Figure 1. Domains and types**

![Diagram showing domains and types]

But what prevents Bob from logging in as a manager, or Mary from logging in as a cashier? More interestingly, how does Boss log in as both?
The first piece is done by a new PAM module. Briefly, PAM (pluggable authentication modules) allows small pieces of code to be executed at various authentication steps and allows a flexible specification of which modules to execute when. I've introduced a new one, pam_ctx.so, whose code is under /usr/src/pam_ctx. It searches the file /usermap.conf for the username being authenticated and finds a default context for this user. For Bob, this default context is cashier_u:cashier_r:cashier_t. For Mary, it is mgr_u:mgr_r:mgr_t. And for Boss, it is full_u:mgr_r:mgr_t. Note that all the contexts consist of three pieces separated by colons:

- The last piece is the domain, which ultimately decides the user's permissions on the system.
- The second piece is the role, which limits the domains that the user may enter.
- The first piece is the "SELinux user." Analogous to the role and domain, this limits the roles that the process may enter.

The PAM module sets the context by writing the context into the file /proc/$$/attr/exec, then executing a shell that is a valid entry type to the new domain. Looking at the policy source, you see that mgr_r may be associated only with domains mgr_t, mgr_register_t, and rolechange_t. Role cashier_r may be associated only with domains cashier_t and cashier_register_t. Similarly, SELinux user mgr_u may associate only with role mgr_r and cashier_u with cashier_r. User full_u may associate with either mgr_r or cashier_r.

Figure 2 shows how all these pieces correlate.

**Figure 2. How the pieces fit together**

The top row shows our SELinux users, the middle row lists the roles, and the bottom row lists domains. A valid security context can be constructed using one item from each row as long as the three are connected. In policy, the user definition:

```plaintext
user full_u roles { mgr_r cashier_r };
```

defines one of the users and its connections to roles, while the role definition:

```plaintext
role cashier_r types { cashier_t cashier_register_t };
```

defines a middle row item and its connections to the bottom row.
But what prevents Bob from simply writing "full_u:mgr_r:mgr_t" into /proc/$$/attr/exec? Several things. One comes back to type enforcement. Looking back at Figure 1, once you are in type cashier_t, you can enter only cashier_write_t. Also, the SELinux policy specifies which role transitions are allowed. So you have:

```
allow mgr_r cashier_r;
```

specifying that a process in role mgr_r may switch to a valid context in role cashier_r. But there is no line:

```
allow cashier_r mgr_r;
```

so Bob cannot transition to any context in cashier_r.

We also deny Mary the ability to enter the cashier_t domain. Looking at Figure 1, the domain transition itself is in fact allowed. And the policy line:

```
allow mgr_r cashier_r;
```

also allows the role transition. However, the policy specifies that it must be done by first entering rolechange_t through the /bin/role_change entry point. This program will not rewrite the SELinux user part of its context. Therefore, once logged into mgr_u:mgr_r:mgr_t, there is no way to transition into the cashier_r role without logging back in as a valid full_u user as authorized by the /usermap.conf file and executed by our pam_ctx.so module.

There is one thing that we did not forbid in our policy. Note that there are no inherent controls over SELinux user transitions. All such controls must be implemented by tying SELinux users to role and domains so that the role and domain transitions do not allow transitions to any valid context with another SELinux user—if that is our desire.

In particular, let's try the following. Log in as root and modify /bin/register.py to cause it to tell us its context. We'll add some lines to a new file addme, then insert that file near the top of /bin/register.py.

```
echo 0 > /selinux/enforce
cat > /root/addme << EOF
f=open("/proc/self/attr/current", "r")
print f.readlines()
f.close()
EOF
echo 1 > /selinux/enforce
```

Now use the down arrow to bring cursor below the import lines, then type Control-r for Read File, and type /root/addme. Now type Control-O to write the file, return to confirm the filename, then Control-X to exit. Finally, put SELinux back into enforcing mode.

```
echo 1 > /selinux/enforce
logout
```
Log in as bob, explicitly ask SELinux for a domain transition, then run register.py:

```
$ echo "full_u:cashier_r:cashier_register_t" > /proc/self/attr/exec
$ /bin/register.py bob 25
```

Now register.py is running as full_u:cashier_r:cashier_register_t! Echoing a context into the /proc/pid/attr/exec file causes SELinux to try transitioning to that context on the next exec. Of course, this will work only if the transition is valid. In this case, the transition was valid since we were not changing roles, and cashier_t executing a file of type cashier_exec_t is allowed to transition to cashier_register_t. If you try:

```
$ echo "full_u:mgr_r:mgr_register_t" > /proc/self/attr/exec
$ /bin/register.py bob 25
```

you will notice that permission is denied. Lastly, you can try:

```
$ echo "full_u:mgr_r:cashier_register_t" > /proc/self/attr/exec
$ /bin/register.py bob 25
```

This time permission was not denied, but the context was reported as cashier_u:cashier_r:cashier_register_t. Why the different behavior? Because full_u:mgr_r:mgr_register_t was a valid context, and so the next exec actually attempted the domain transition and failed. However, full_u:mgr_r:cashier_register_t was not even a valid context since mgr_r may not be associated with cashier_register_t. Had we checked the return value after echoing the context into /proc/self/attr/exec, we would have found that it had failed.

```
$ echo "full_u:mgr_r:cashier_register_t" > /proc/self/attr/exec
$ echo $?
1
```

So when you next ran register.py, it did not attempt the requested domain transition but simply proceeded with the default domain transition, which succeeded.

At this point, you may be thinking that our goals could have been met by using strictly TE without making use of roles or SELinux users. However, the use of roles and users can make future administration of the system easier. This will become clearer next when you see how to implement this system on Fedora Core 8.

**Using Fedora Core 8**

Fedora 8 comes by default with SELinux enabled. It integrates new SELinux technologies using loadable policy modules to ease policy customization and `semanage` to ease user and RBAC administration. (Semanage is used to configure certain elements of SELinux policy without requiring modification to or recompilation from policy sources.)

Let's begin by doing a very uninteresting almost-default install. First download Fedora-8-i386-DVD.iso (see the link in Related topics below). You might rename it to f8.img for convenience. You will use it as a cdrom image to install Fedora 8 under qemu:
dd if=/dev/zero of=f8.img bs=1G seek=10 count=1
qemu -hda f8.img -cdrom f8.iso -boot d -m 1024 -vnc 3

Then, launch VNCviewer:

vncviewer :3

In the VNC window, follow the directions for a completely default install with one exception: When asked about packages to install, deselect "Office and Productivity" and select "Software Development."

When the installation is complete, continue to follow the setup directions after reboot and choose a non-root account to create. Finally, when the image is ready, log in as that user.

Open a browser window to this article and download the source tarball for Fedora 8 (see the link in Related topics). Save it in your home directory as ~myuser/cash_register_f8.tgz.

From the "Applications" menu at top left, choose System Tools > Terminal. Then type su - and enter the root password to open a root shell. Now you are ready to set up the secure register balance accounting system.

(Note: if the system appears simply too slow to be bearable, you can exit X-windows by entering runlevel 3. You can do this by typing /sbin/init 3. You can always restart X-windows by re-entering runlevel 5 using /sbin/init 5. At runlevel 3, simply log in as root on the terminal.)

First, force a background daemon to exit so you can manually run yum:

killall -9 yum-updatesd

Next install the SELinux policy module development package:

yum install selinux-policy-devel.noarch

Now copy the example policy module directory, copy the cash register policy files into the example directory, and compile them:

cd /usr/share/selinux/
cp -r devel cash_register
cd cash_register
rm example.*
tar zxf ~myuser/cash_register_f8.tgz
mv register.py /bin
make

The policy is compiled into a binary policy module in the file cash_register.pp. To load it:

semodule -i cash_register.pp

Next, create the users mary and bob:
adduser bob
adduser mary
passwd bob
passwd mary

Now that the users exist, set up RBAC to log them into the appropriate roles:

semanage user -a -R cashier_r -P cashier bob_u
semanage login -a -s bob_u bob
semanage user -a -R mgr_r -P mgr mary_u
semanage login -a -s mary_u mary

The `semanage user` command creates a new SELinux user. The SELinux user is not the Linux username, but rather the first piece of the SELinux context (returned by `id -Z`) attached to a process and file. If you type `id -Z` at the terminal, you may see `system_u` or `unconfined_u`. While your Linux username and SELinux username may be made identical, they are not by themselves related. The login process, however, uses the Linux username to choose an SELinux user for your security context. As discussed in the previous section, the SELinux user is restricted in which roles it may be associated with, and likewise the SELinux role is restricted in the SELinux domains (types) it may be associated with.

You are using `semanage user` to create two SELinux users, mary_u and bob_u. At the same time, you are specifying the roles with which they may be associated. User bob_u may use only role cashier_r, while mary_u may use only mgr_r. You also must specify a prefix for the user's home directory type. For Mary, specify mgr, which will expand to mgr_home_dir_t for her home directory and mgr_home_t for files therein.

The `semanage login` commands tie Linux usernames to SELinux users. We specify that mary logs in as mary_u and bob as bob_u.

We do still need to create the preliminary directory structure for the cash register values:

```bash
mkdir /data
mkdir /data/final
mkdir /data/cashier_r
mkdir /data/mgr_r
chmod 777 /data/cashier_r
chmod 777 /data/mgr_r
chmod 777 /data/final
```

Finally, relabel all the files created and installed as well as the home directories for the new users:

```bash
fixfiles -f relabel /data /bin/register.py /home
```

Note that this time we did not define SELinux users in the policy. Rather, the `semanage` commands created the users and associated them with the appropriate roles.

If you only wanted to allow bob to log in as cashier_r and mary to log in as mgr_r, then things would be fine as is. However, perhaps you need user charlie to be able to log in as either mgr_r or cashier_r. This requires a few more changes. First, create the user:
adduser charlie
passwd charlie
semanage user -a -R mgr_r -R cashier_r -P mgr charlie_u
semanage login -a -s charlie_u charlie

Now tell PAM that charlie should be asked which role he wants to log in as. First, open /etc/pam.d/login and replace the line:

```
session required pam_selinux.so open
```

with:

```
session required pam_selinux.so open select_context
```

This tells the pam_selinux.so module that users should be able to choose a default context upon login. Next, tell the system the default type to use for role charlie_r. Upon login, Charlie will be allowed to specify a role other than the default (mgr_r, since we listed it first in the `semanage user` command). Options are any role you specified with the `-R` flag when you created the user. SELinux will then use the default type associated with the requested role, so you must specify a default type for cashier_r:

```
echo "cashier_r:cashier_t" >> \
/etc/selinux/targeted/contexts/default_type
```

Now when Charlie logs in on console (either Ctrl-Alt-F2 to get to a console or enter runlevel 3 as described earlier), he will be asked which role to log in to. The default will be mgr_r, but he can enter cashier_r instead. If he does so, he will be able to store values for himself as cashier, but as a result of the policy we have defined, will not be able to read any files in his home directory.

Note that register.py does not protect against Charlie storing values for his own cash register as both manager and cashier, then committing the values. Of course, such a change would be simple to implement.
### Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample from-scratch code</td>
<td>code_for_fromscratch.tgz</td>
<td>7KB</td>
</tr>
<tr>
<td>Sample Fedora Core 8 code</td>
<td>code_for_f8.tgz</td>
<td>2KB</td>
</tr>
</tbody>
</table>
Related topics

- Download Fedora-8-i386-DVD.iso.
- "SELinux from scratch" (developerWorks, May 2006) shows you how to build an SELinux-ready Gentoo system.
- See SELinux for distributions on SourceForge.
- In the developerWorks Linux zone, find more resources for Linux developers, and scan our most popular articles and tutorials.
- See all Linux tips and Linux tutorials on developerWorks.

© Copyright IBM Corporation 2008
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)