SELinux Set up a secure machine without hampering your workflow

SELinux: Guard

Bob Moss explains how to lock down your Linux system without the added security hindering you as you go about those all-important day-to-day tasks.

Our expert

Bob Moss
Bob splits his time between coding things and studying for his computer science degree course.

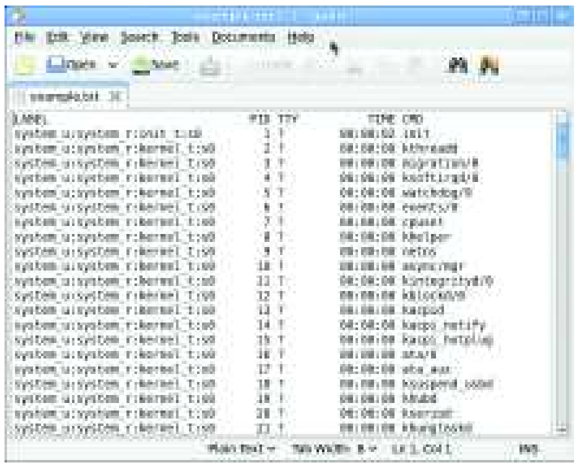
For those of you who haven't heard of SE 'Security Enhanced' Linux before, you're probably not alone. SELinux comes with most mainstream distributions, but is disabled by default (or even actively turned off by users), which leaves this powerful security tool under-used and reflecting forlornly on its poor fortune.

You may wonder why this might be. Back in the early days after the US National Security Agency released it under an open source licence, SELinux was merged with the mainstream Linux kernel. However, it tended to get in the way of what people were doing on a day-to-day basis with their Red Hat systems, and because it was simply too unwieldy or complex for the average user to tinker with, people usually just gave up and disabled the tool entirely.

However, things have come a long way since those days and even competitors such as *AppArmor* have sprung up with a view to providing a similarly high level of security to the standard Linux desktop.

So, what's the point of SELinux? What it does, essentially, is categorise all users, processes and files, and then grant these 'objects' only the bare minimum of permissions that they need to fulfil their function. If the object tries to do anything else, or interferes with what another object is doing, it's stopped and then reported to you in accordance with your security policy.

However, SELinux is only distributed with rules that deal with the most common scenarios, meaning you could one day be trying to complete one of your important daily tasks and find that SELinux jumps in and stops you.



```
LABEL                              PID TTY      TIME CMD
system_u:system_r:init_t:s0          1 ?    00:00:02 init
system_u:system_r:kernel_t:s0        2 ?    00:00:00 kthreadd
system_u:system_r:kernel_t:s0        3 ?    00:00:00 migration/0
system_u:system_r:kernel_t:s0        4 ?    00:00:00 ksoftirqd/0
system_u:system_r:kernel_t:s0        5 ?    00:00:00 watchdog/0
system_u:system_r:kernel_t:s0        6 ?    00:00:00 events/0
system_u:system_r:kernel_t:s0        7 ?    00:00:00 cpuset
system_u:system_r:kernel_t:s0        8 ?    00:00:00 khelper
system_u:system_r:kernel_t:s0        9 ?    00:00:00 netns
system_u:system_r:kernel_t:s0       10 ?    00:00:00 async/mgr
system_u:system_r:kernel_t:s0       11 ?    00:00:00 kintegrityd/0
system_u:system_r:kernel_t:s0       12 ?    00:00:00 kblockd/0
system_u:system_r:kernel_t:s0       13 ?    00:00:00 kacpid
system_u:system_r:kernel_t:s0       14 ?    00:00:00 kacpi_notify
system_u:system_r:kernel_t:s0       15 ?    00:00:00 kacpi_hotplug
system_u:system_r:kernel_t:s0       16 ?    00:00:00 ata/0
system_u:system_r:kernel_t:s0       17 ?    00:00:00 ata_aux
system_u:system_r:kernel_t:s0       18 ?    00:00:00 ksuspend_usbd
system_u:system_r:kernel_t:s0       19 ?    00:00:00 khubd
system_u:system_r:kernel_t:s0       20 ?    00:00:00 kseriod
system_u:system_r:kernel_t:s0       21 ?    00:00:00 khungtaskd
```

› **Your policies aren't just restricted to files - they can act on running processes and users too.**

But before you throw your computer out of the window in a fit of pique, or deactivate SELinux immediately, it's mercifully easy to give yourself the permissions you need to do these things as and when you need to. Over time, your patience will pay off as you find yourself with a much more secure system than before. This tutorial will explain how SELinux security policies work, how to change the SELinux running mode for particular scenarios and how to check the permissions you have, once you've finished tailoring it to your typical computing habits.

First steps

The first thing to do is find out whether or not you already have SELinux installed on your system. Fedora users should find that it's installed by default, thanks to its Red Hat heritage – simply type the following line into a standard user terminal to see which mode SELinux is operating in:

```
getenforce
```

You should see one of three results appear: **enforcing**, **permissive** or **disabled**. The first result tells you that SELinux is enforcing all the defined security policies and reporting any breaches. **Permissive** indicates that SELinux is reporting breaches but not actively enforcing the security policies that have been defined, and the third result speaks for itself.

Ubuntu users will need to take a few extra steps to install SELinux, because *AppArmor* is typically installed by default. Have a read of the 'AppArmor alternative' box on this spread and, if you decide to proceed with SELinux, install it via your package manager. Once it's all finished, type the following command into a terminal to start it:

```
sudo sed -i 's/SELINUX=.*/SELINUX=enforcing/' /etc/
selinux/config
```

This changes the SELinux state from directly within its configuration files. You could also change **enforcing** to

your system

The AppArmor alternative

Those who have Ubuntu or OpenSUSE should find that *AppArmor* is installed by default, although users of the latter distro will need to enable it manually (as shown in the screenshot). Anyone who's used SELinux should find certain aspects of *AppArmor* very familiar. For example, *AppArmor* has Enforce, Complain and Disabled modes, which correspond directly to the three you would use in SELinux. It's also a little easier to start and stop – Ubuntu users can simply use the following command:

```
rcapparmor start
```

Substitute the **start** for **stop** to produce the opposite effect. You can also use **reload** to reload all current *AppArmor* profiles (the equivalent of SELinux rules), or **status** to display how many profiles are loaded in Complain and Enforce modes. Event-logging is similarly straightforward to manipulate:

```
racaaeventd start
```

As before, you can simply use **stop** instead of **start** to have the opposite effect.

One advantage of *AppArmor* is its Learning mode, which watches how your machine is being used for as long as you choose and then uses this information to decide what should and shouldn't be allowed to happen.

Chinks in the armour

This is particularly useful for webserver configurations, where localised access would ideally be restricted, and web pages would only be accessed by particular people in certain ways.

While this sounds like a fantastic idea, there are some limitations. It's easy for *AppArmor* to pick up bad habits while in Learning mode, meaning that it might see an event that you would want to be against security policy, but wouldn't receive a report about it because *AppArmor* considers it to be normal behaviour. This can be mitigated with care, and in certain scenarios might not be too costly, but this could be considered a downside for some. SELinux doesn't have this problem, because it's

distributed with a set of security policies for the most commonly occurring scenarios and it then leaves it to the user to define more specific policies as necessary. This does require more effort and some prior knowledge, but also means that you should hear about any undesired behaviour on your machine as soon as it occurs.

Both of these approaches have pros and cons, but after this tutorial you should be able to make a more informed choice about which you install on your favoured distribution.

permissive or **disabled** as appropriate, to change the current SELinux operating mode as you require.

Fedora users have a much easier route to enabling SELinux and changing the operating mode. One possibility is to use the GUI tool in the System > Administration area of the Gnome desktop and change it using the drop-down. Then click **Apply**. If you prefer, you can perform the same action using the command line, by typing the following:

```
echo 0 > /selinux/enforce
```

The **0** operator tells SELinux to disable Enforce mode for the duration of the current session and sets it to Permissive mode instead. If you decide you want to re-activate Enforce mode, simply run the same command again but with a **1** instead of the **0**.

If you'd rather make the mode change permanent, open up **/etc/selinux/config** in your text editor with root permissions and change the relevant **SELINUX** line to:

```
SELINUX=enforcing
```

Again, you can change the **enforcing** part to **permissive** or **disabled**, depending on your requirements.

You may be wondering at this point why you'd want to change SELinux operating modes in the first place. The reason it's important to be able to is because if you need to perform any large maintenance tasks or troubleshoot problems, you may find that SELinux policies (which were designed for daily desktop use) get in the way. Therefore, it's useful in these instances, but for most of the time, you'll be using your desktop as usual and should be able simply to leave your tailored SELinux configuration running quietly in the background without it bothering you.

"Your patience will pay off as you get a much more secure system."

Allow me

Now that SELinux is running, you may occasionally find that things you did before have stopped working (or you may see a dialog box telling you that you're about to be reported back to yourself). When this happens, you could try a useful tool named *Audit2allow*, which takes the details of the most recent actions that SELinux has prevented and gives you the ability to allow them. Before you can use this though, check whether the *auditd* daemon is running, by using the following command:

```
ps -ef | grep auditd
```

If you see **auditd** in the list that appears and it doesn't have **grep** in front of it, then you know that the daemon is running. Don't panic if it doesn't appear though, because the only difference to the command below is to substitute the **a** for a **d**:

```
audit2allow -l -a -M local
```

The **-l** option reads the denials since the last policy reload (in most cases, this will be from when you logged in), and the **-M** moves the denials into a module you can use for a rule. Bear in mind that if you want to use this line more than once, you'll need to change the name **local** to avoid any conflicts. When you're happy with the rule, all that's left to do is to build it by typing **semodule -i local.pp**.

SELinux: Make

Now that you know how to use SELinux, why not create your own rules?

You've now set up SELinux and know how to create a new rule to help you out when you're blocked from completing your daily tasks on your Linux box. However, there may be certain files and parts of your system that you'd prefer to keep away from prying eyes, or think could do with some extra protection. In these instances, you'll need to create your own rules.

However, rules are not much good if you have no means of checking which SELinux policies apply in a particular situation, so we'll cover that here as well.

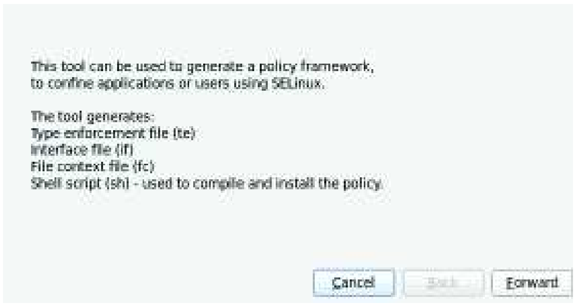
Now that you've established that SELinux is present and running in the right operating mode, learnt how to bypass being blocked for legitimate activity, the next step is to start applying your own specific rules. If you wanted to, you could just stick with Fedora's handy in-built rule-making wizard, but if you've got this far, you probably want to know more about what's going on behind the scenes with security.

The first consideration is the SELinux concept of a 'user'. This isn't the same as your typical system user, who can elevate to root using **sudo** or **su**. In fact, many system users will have the same SELinux user, so for this reason it's easier to regard SELinux users as a kind of 'user group', each of which could contain any number of different system users, all of whom share the same roles and permissions. When an SELinux user is mentioned in a rule or system policy, there tends to be a **_u** appended to the end of the name (**bob_u**, for example).

Quick tip

You can get even more information on SELinux from **http://selinuxproject.org**.

> **"There may be parts of your system that could do with protection."**


This tool can be used to generate a policy framework, to confine applications or users using SELinux.

The tool generates:
Type enforcement file (te)
Interface file (if)
File context file (fc)
Shell script (sh) - used to compile and install the policy.

› **Fedora comes with a wizard that enables you to set up your own security policies quickly and easily.**

Next, let's consider user 'roles'. An SELinux user can have as many or as few of these as you specify for them, and each role is defined by the SELinux policy. Examples of roles you might bestow are 'visitor user' or 'website administrator'. Users of objects associated with a role usually have **_r** at the end of the name to follow convention, so these too are relatively quick to spot when looking through a policy.

Each rule could also refer to a 'type' (on the internet, you may occasionally see this called a 'domain'), and this determines what resources can and can't be accessed. Again, by convention, the type name is typically suffixed with **_t**

Armed with these three pieces of information, the next step is to create a 'context' for your rule. This is essentially a means of putting the user, role and type together in the first half of the rule, or you could use them individually.

Keep it in context

Optionally, you can add a 'range' to this context, but this isn't strictly necessary unless you're applying a rule to many different types. Your context could take on the following form:

```
user:role:type:range
```

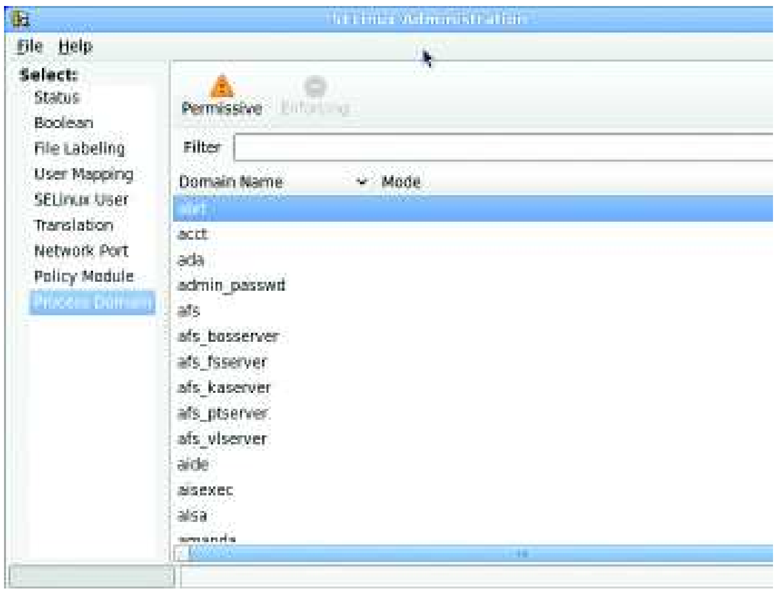
While an example context without a range would be:

```
system_u:system_r:xserver_t
```

In this context, you can see that you're referencing a **system** user with a **system** role with the access privileges of the **xserver** type (which is rather specific). If you used them each without the colons, you'd be defining a rule for all system users, all users with the system role and all users with the same access privileges as **xserver_t**, which would be a much broader group of users.

Now that you have an idea of what a 'context' is, you could apply your context to any object classes and grant it the relevant permissions. SELinux provides myriad object classes

› **If the command line scares you, this GUI front-end will help you manage SELinux just as well.**



```
abrt
acct
ada
admin_passwd
afs
afs_bosserver
afs_fsserver
afs_kaserver
afs_ptserver
afs_vlserver
aide
aisexec
alsa
amanda
```

your own rules

(better described as a 'category of objects') for users, processes and files. Examples of object classes would be **dir** for directories and **file** for files.

Each of these objects then has a set of permissions that you can enable or disable. For example, **file** would have permissions to create, unlink (delete), read and write, while a **network** object might have permissions to create a connection or send data. You can see an exhaustive list of all the available object classes and their associated permissions at the SELinux project website at **http://bit.ly/9idu42**.

Applying rules

An example of allowing your object class permissions to two different types of users would be:

```
allow user_t user_home_t:file { create read write unlink };
```

As you can see, you use **allow** to allow these particular types of user to perform these actions on a file (the opposite would be to change this command to **deny**).

You can apply similar rules to types, names or the combination of the three that were discussed earlier in the tutorial, meaning that you can allow or deny any aspect of your system to any particular object, giving you maximum say over how your computer works.

Now that you understand about rules, you may find that there are times when the global system default of enforcing doesn't work for a particular situation. For example, you might need to modify part of your system temporarily before reverting to the SELinux default.

Rather than changing the global SELinux operating mode and affecting your entire system, you could simply alter the operating mode of a particular rule using **semanage**:

```
semanage permissive -a myapp_t
```

Try substituting **myapp_t** for any other object or context. Then simply replace the **a** with a **d** to change the object in question back to the system default.

Once you've finished manipulating SELinux to work the way you want it to, you may find yourself needing to troubleshoot a problem – or you might simply be curious to know what SELinux policies will affect you in a particular situation. This is particularly handy if you're trying to transfer a file or install a rogue package that SELinux is protecting you from. There are two lines you can use in the terminal, and both use the magic and case-sensitive **-Z** that you'll see mentioned on countless sites around the web:

```
ls -alZ *
```

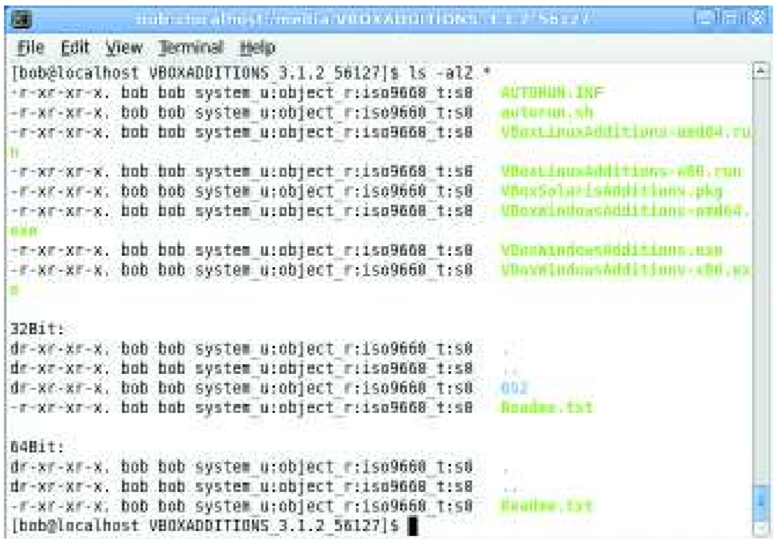
This line will list every file in the current directory with its associated ownership, permissions and SELinux rule (simply substitute the * for a filename or path as necessary). You can do something similar for the processes that are currently running on your system with the following line:

```
ps -eZ > example.txt
```

This will list all running processes, along with their associated SELinux rules, and then insert this information nicely into a text file for your analysis. It's possible to have all this data displayed directly in your terminal window, but this is far less useful because the results aren't always particularly pretty or easy for you to interpret.

And finally, if you need to check out what your currently active system user's associated SELinux policy is, simply type **id -Z** into your terminal.

So, armed with a bit of patience and these instructions, you can get on with your day-to-day tasks, safe in the knowledge that your computer is well secured. LXF



```
[bob@localhost VBOXADDITIONS_3.1.2_56127]$ ls -alZ *
-r-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   AUTORUN.INF
-r-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   autorun.sh
-r-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   VBoxLinuxAdditions-amd64.ru
n
-r-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   VBoxLinuxAdditions-x86.run
-r-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   VBoxSolarisAdditions.pkg
-r-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   VBoxWindowsAdditions-amd64.
exe
-r-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   VBoxWindowsAdditions.exe
-r-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   VBoxWindowsAdditions-x86.ex
e

32Bit:
dr-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   .
dr-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   ..
dr-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   OS2
-r-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   Readme.txt

64Bit:
dr-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   .
dr-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   ..
-r-xr-xr-x. bob bob system_u:object_r:iso9660_t:s0   Readme.txt
[bob@localhost VBOXADDITIONS_3.1.2_56127]$ █
```

› **You can either apply blanket SELinux policies across all your files, or create your own special rules that will work in particular cases.**

For the more adventurous

For the benefit of those of you who want to get to the real nuts and bolts of how SELinux works or wish to write and compile their own security policies without the aid of GUI or command line tools, it's useful to know how SELinux is arranged on the root partition.

All the global configuration files used and modified by supporting applications and utilities are available via **/etc/selinux**. The policy-specific files will be in the same folder, named accordingly, and with an accompanying **modules** folder where additional policy settings reside. You should also find these policy names in the **/etc/selinux/config** file next to a line starting **SELINUXTYPE=**.

Kernel-specific SELinux modules and files are located under the **/selinux** directory and contain your current configuration and active security policies. Unless you're absolutely sure of what you're doing, it's best to keep well away from this folder, because the consequences of doing something wrong are potentially disastrous. You could crash your system, or worse – lock yourself out of it!

Writing and compiling your own SELinux policies without tools is beyond the scope of this tutorial, but the Red Hat website covers the latter stage in a tutorial. But again, this is for advanced users, as the warnings on the site indicate: **http://bit.ly/clXESW**.