Creating SELinux Policies Simplified

Irfan Habib

Abstract

SELinux does not have to be as hard to manage as many people may think.

Modern Linux distributions ship with a plethora of security features and tools, and one of the most important features added to the kernel has been the inclusion of SELinux.

SELinux solves one the most challenging problems in security. How to control trusted users or processes. Trusted users, such as root in the *nix domain, have unlimited and unrestricted access to a system. They should have, as this account is supposed to be used only by the system administrator; however, this leads to a problem. What happens if the root account itself gets hacked, leaving the hacker with complete control of the system. Malicious users are not the only problem; misconfiguration of a security tool, such as iptables, can have a profound effect as well. Besides this, imagine a security vulnerability is discovered for a service you deployed on your server, and no patch is made available in a timely manner. In this case, your system is vulnerable. SELinux secures Linux systems from these sorts of security issues by implementing mandatory access controls (MACs) in the Linux kernel. SELinux is based on the Flask security architecture. Discussion of the Flask architecture is avoided in this article, as excellent documentation about it is easily available on the Internet.

To understand mandatory access control, we must go through the currently deployed security model called discretionary access control (DAC). In a DAC system, access to objects is restricted based on their classifications. This type of control is discretionary in the sense that a subject with a certain set of access permissions is capable of passing those permissions on to another subject. For example, any program you run while logged on as a certain user has the same access rights that you have. Rights are set by another user (for example, root).

Any particular permission (read, write, execute and so on) can be thought of as a two-dimensional graph with users on one axis and objects on another. In essence, DAC systems check the validity of credentials presented to them against stored information.

As mentioned, the SELinux security model is mandatory access control, or MAC. This controls access in a different manner. Whereas DAC security models are authentication-based, MAC systems rely on authorization, not only of users but also of each object loaded by the system.

A MAC system controls objects individually and makes decisions on the rights and/or permissions of objects based on a security policy, which can define what rights the object should be accorded, based on different variables.

An example of how discretionary versus mandatory access control styles could affect the operation of a computer is a Python script. If the script allows an external entity to insert and execute malicious code on a computer system under a DAC system, the malicious code now has the same access rights as the code that executed it -- the Python script.
A MAC system can restrict the rights of a certain process to only the resources needed for normal operation. A Python script may create a process (or it may be forbidden), but that process might not have the same set of permissions as the process that created it. Thus, the MAC approach is seen as secure.

**A Brief Look at SELinux Internals**

In SELinux, the security policy configuration is defined in a text file written in the m4 language. It is compiled when the security policy is finalized and, at boot time, loaded into the memory. Only the security server can make policy decisions on the permissions of an object.

Security policy enforcement is done by components called object managers, which receive requests from client objects, submit queries to the security server and enforce the resulting decisions.

The SELinux implementation of the security server uses a combination of two security paradigms, called Type Enforcement (TE) and Role-Based Access Control (RBAC).

**Type Enforcement**

Type Enforcement makes security decisions based on what kind of object requests the permissions. For example, object types could include a regular file, a directory, a process or a socket. Type enforcement is an object labeling system that, combined with access mapping (from the domain of the object requesting permission and to the type of the object requested), returns a decision that defines the permissible actions of the object.

**Role-Based Access Control**

Role-based access control assigns permissions to objects in a computer system based on the role they play within that system. In practice, this means a process would have its permissions based on its parent process, the user logged on at the time and any number of other variables.

How processes, filesystem objects and sockets communicate with each other is defined by the security policy. In particular, the security policy governs how different types and roles may interact, along with any specific rules.

At present, SELinux provides binary compatibility with existing applications and source compatibility with kernel modules. The current implementation of SELinux is x86-specific.

**Getting and Installing SELinux**

SELinux is included in many distributions nowadays, and even if it does not come with the main distribution, distribution-specific packages usually are available for popular distributions. The sources for SELinux are available from [http://www.nsa.gov/selinux/code/download0.cfm](http://www.nsa.gov/selinux/code/download0.cfm).

**Installation of seedit**

seedit is a user-friendly Webmin-based tool that enables an administrator to administer SELinux policies from a Web browser. The seedit Webmin interface allows users to perform every action they can normally
perform by writing manual scripts in m4 in a point-and-click environment.

seedit is available from http://seedit.sourceforge.net.

Once installed, the SELinux policy is located in $SELINUX/(seedit-something)/policy/policy.conf (where $SELINUX is the root directory of your SELinux installation -- usually it is /etc/selinux/).

Let's familiarize ourselves with the seedit interface. Fire up a browser, point it to http://localhost:10000 and go to the System→SELinux Configuration section.

You will see six icons labeled configure ACLs, define domain transitions, define relationship between users and roles, create new domain/roles, delete domains/roles and update configuration.

Confgure ACLs

Here you can define virtually all access control to nearly every object in the system. This includes allowing/disallowing read, write or executable files to entire directories or individual files and allowing/disallowing access to networking capabilities. In this section, you also can define ports in which the specific application can work. Let's say we assign port 80 to Apache; if it is started to work on port 81, SELinux terminates this process.

IPC access control can be defined in this section also. You can define what kind of IPC mechanism this particular application can use and with which applications the particular application can communicate.

Other access controls that can be defined in this section include various administrative access controls, such as kernel communication privileges, SELinux operations, process information retrieval and so forth.

Defining Domain Transitions

In the domain transaction section, you can define which processes can spawn the current application -- for example, by default, the seedit policy defines this domain transition for MySQL: kernel→init→mysqld. This means the kernel can start init, and init in turn can launch the MySQL daemon. If the application has a daemon associated with it, domain transitions should be defined or the daemon will never be able to start.

So in this section, the user can define domain transition, alter existing ones or remove them altogether.

Define Relationship between Roles and Users

Roles are privileges that an object (like users) can have on a system. For example, there may be a role that allows access to all files in the system. Thus, in this section, users can associate roles with specific users on the system.

The next two sections are self-explanatory. They allow you to define a new domain/role and remove a domain/role. Defining a domain/role is the first step when adding access control rules for a new application or user.

The last section is the update configuration section; it allows the user to update the policy and recompile it and load it.
seedit comes with a plethora of predefined policies, which cover every popular server/daemon in a Linux system -- from the kernel to the MySQL daemon.

**Defining a New Policy for a Daemon Application**

Defining an SELinux policy for a daemon is an iterative process. The first step is to “register” the daemon with SELinux, by declaring a domain/role in the seedit Webmin interface. This is done in the Add Domain/role section discussed previously. The convention is that domains should end with a _t, and role definitions should end with an _r. So, there could be a role, such as admin_r, and a domain, such as mysqld_t.

Defining domain transitions is another important step. Here you have to allow the parent processes of the daemon to spawn the concerned daemon. Usually if the daemon activates at startup, you need to define a domain transition from init to the concerned daemon.

Define the initial access control list for the daemon. Now, it is not possible for the user to define an exhaustive ACL for the daemon when it is initially installed, rather what is usually done is that an initial ACL list is defined, which is very restrictive. Every time the application tries to access an object and is not allowed to do so, an access violation message occurs; such messages can be accessed from /var/log/messages. As you use the daemon, SELinux will report some violations. Considering that the daemon for which we are trying to write rules is called foobar, the violation messages will look like this:

```
|avc: denied { write } for pid=7279 exe=/usr/bin/foobar comm=ifup
name=dhclient-eth0.conf dev=hda12 ino=57400
scontext=system_u:system_r:foobar_t tcontext=system_u:object_r:etc_t
tclass=file
```

This violation states that a write attempt was made to the file dhclient-eth0.conf by our application. This violation can be removed by giving the application write access to the dhclient-eth0.conf file. This violation can be addressed by going to the Configure ACL section of the seedit Webmin interface, under the File ACL section, and then browsing to the place where this file exists, and giving the application foobar write access to it.

Another violation might look like this:

```
|avc: denied { create } for pid=7279 exe=/usr/bin/foobar
scontext=root:system_r:foobar_t tcontext=root:system_r:foobar_t\
tclass=udp_socket
```

This violation reports that the application tried to create a UDP socket and was denied. To remove this violation, we can simply add networking features to the access control of the foobar_t domain. This can be addressed by going to the Configure ACL section and in the Network ACL section, selecting Allow Network for the domain foobar_t.

All access violations can be addressed in the Configure ACL section of the seedit Webmin interface.

After every policy update, reload it via the Update Policy option in the seedit Webmin interface and restart the daemon. As other violations occur, update it again and so on, until you get nearly no violation messages.

There are other ways to generate access rules for an application with the help of audit2allow tool. But using it can lead to general rules, which can lead to security problems. You always can refine the rules the
When creating or updating policies, make sure you have set your SELinux installation to permissive mode. SELinux has three modes: enforcing, disabled and permissive. In enforcing mode, all access controls are enforced according to the defined policy. In permissive mode, the policy is not enforced; however, violation messages are shown when any violation to the policy occurs. Disabled mode completely disables SELinux.