Juan Ortega

8/20/09

NTS222

# Security Enhanced Linux (SELinux)

Developed by the National Security Agency, Security Enhanced Linux provides a powerful, flexible, mandatory access control (MAC) mechanism through the use of Linux Security Modules (LSM) loaded onto the Linux Kernel. The implementation of MAC onto the Linux kernel was necessary in order to solve the weaknesses discretionary access control (DAC) by its nature developed. All *NIX operating systems today include DAC as its default set of access controls; typically every user has any combination of read, write, and execute permissions on a specific object. This security design would work perfectly if every program on the system had no flaws, security holes, or became malicious; unfortunately, has anyone who ever used a computer comes to learn, programs are not always trust worthy. Programs with vulnerabilities are even more troublesome, as they can be used to obtain higher privileges. The solution, use the principle of 'least privilege' and give objects sufficient authority to complete its job and nothing more, this would mean giving objects permissions as opposed to users. For example, even though the object is running as root, there is no reason why a daemon would need to execute /bin/sh, a DNS server would want to read /etc/shadow, or MySQL would want to read your home folder, the objects are confined to their tasks only. The only problem is the complexity; this would require giving every single object inside the Linux kernel detailed permissions; fortunately, SELinux has many security contexts to go by.

The three types of security contexts under SELinux: Type Enforcement (TE), Role-based access control (RBAC), and the optional Multilevel Security (MLS), are used when creating a security policy. Before we can go in detail on each of the security contexts we need to understand some terms. Every file, socket, directory, etc., are called *objects*, and every running process is called *domains* (or subjects). All resources are specified as *types* both objects and domains. Types are usually ended in _t suffix, for example you may declare passwd_t, shadow_t, home_folder_t as types that point to the objects. *Attributes* are used to refer to a group of types; for example, you may have the type's httpd_content_t, httpd_settings_t, httpd_server_t, all in an 'httpd' attribute. *Aliases* are essentially an alternative name to a type. Lastly, without getting into the SELinux architecture, the *access vector cache* (AVC) contains audit information on domains that were denied access; this is useful for troubleshooting security policies, or discovering malicious activity. Now that some basic terms were defined we can understand how TE works.

As the name indicates, Type Enforcement retains the majority of a security policy specifying permissions for every resource in the system. There are four Access Vector rules you can use: allow, dontaudit, auditallow, and neverallow. Auditing is the same as logging the access into the AVC to review, so the rules should be straight forward. Without getting too much into the syntax, the kind of permission a type may have is specified in classes. A *class* contains a group of permissions; for example, a 'dir' class contains the permissions: open, rmdir, search, remove_name add_name, and so ford. A 'file' class contains: append, create, execute, getattr, read, and so on. One last thing to touch upon is domain transitions. The +s attribute on a standard Linux system specifies an application be run with permissions as the owner regardless

of who executed the program. The ‘passwd’ program utilizes this feature because it needs to change the shadow file as root every time. Just like the program changes from its default user permissions to root, domain transitions change one domain to another with sufficient privileges. For more information on Type Enforcement, visit [1].

In addition to TE, SELinux supports a form of *role-based access control* used to group types and to restrict relationships between domain types and users. Besides having normal Linux users specified in the /etc/passwd file, SELinux has its own users or *user identifier* usually specified with the _u prefix. Users from Linux and SELinux may or may not be the same, they must be explicitly defined. For example, you can have a user_u associated with all Linux users or have a guest_u to specify guest accounts; this is great for separating privileges. Roles are defined with the _r prefix stating they are a collection of types. The role user_r may consist of type resources such as firefox_t, ping_t, chmod_t, cd_t. When a user is associated with a role such as user_u with user_r, all the Linux users consisting of user_u will be able to use types with those specified in the user_r; but before users in user_r can be allowed to run objects specified in user_r, TE rules have to added on top of RBAC. Objects themselves may be roles as well such as the object_r is hardcoded to define every old and new object in the system. For more information on RBAC, check out this article [2].

*Multilevel Security* is an optional security context built upon TE to provide *sensitivity* level of control, this type of form is mostly associated with government-classified data control. *Categories* are equivalent to the type of security clearance professionals have to acquire, such as confidential, secret, and top secret. In a MLS, objects and domains may only access resources that are equal or lower than their own category. By default this feature is off and the entire

SELinux system contains only one sensitivity level also known as *s0*. Because this form adds complexity to an already complex policy, many do not use this feature, but it is entirely up to the policy writer. For more information on MLS, CentOS includes a well written document [3].

No syntax would be complete without *Boolean Variables* and SELinux is no exception. If you are not familiar with programming logic, Boolean variables are conditional expressions, normally used as 'if' and 'else' associating true and false statements. In other words, as it goes in SELinux, it is an off and on switch used to change permissions without the need to recompile the policy. For example, say you want users to stop using the ping command or restrict them; you would declare a Boolean name called 'user_ping' and specify TE rules to allow or disallow depending if the Boolean is on or off (by default its off). Now without changing the security policy the switch can be changed at will by a simple command tool (setsebool).

Now that we've covered the bare essentials of SELinux, let us take a more hand-on approach using *apol*. Grab the latest version of Fedora Linux and install *setools*. At the time of this writing the latest version is Fedora 11. Once installed go to Application>System Tools>SELinux Policy Analysis, we are going to view the default policy Fedora comes with. After the application is open go to File>Open and click Browse. Open /etc/selinux/targeted/policy/policy.24 (might be different on yours). Now why is the policy in a folder labeled 'targeted'? NSA released the *example policy* when SELinux first came out, then NSA released *strict policy* which attempts to provide a domain type for every program that reasonably requires a private domain. The only problem was the policy was too, well strict and caused breakage with existing Linux application; it needed more flexibility for newcomers. To address the issue, Fedora came up with *targeted policy*, this new policy allowed most programs

to run as if they were not confined under SELinux (also known as *unconfined*). The targeted policy does however confine kernel resources known to the system; this approach gave some level of security to the system while the user does not notice a difference. Now that we know what the policy confines, let us look at some types.

Once you have the policy loaded into apol, you should by now understand some of the information presented. The Types tab includes every object and domain in the system explicitly defined, scroll down and notice some of your favorite command line tools are mentioned. The attributes groups again specify a group of types, if you double click on an attribute a pop-up will appear defining the types associated with that attribute. After exploring the Types tab, click on Classes/Perms. Hopefully you remember what classes are, if you do click on an object class and it will display all the possible permissions associated with that object. As you can see, this goes beyond the normal read, write, and execute permissions on DAC. Next, click on the Roles tab. Double click on user_r, as you can use a list of types appear and as previously stated, these types can be used to define TE rules to allow that object or domain to run. Notice their all grouped from guest_r, sysadm_r, system_r, webadm_r, you may define which role has access to what types such as “webadm_r” may only access web related tools and directories. After clicking on the User tab, you would notice the users are named quite similar to the roles, this is done for simplicity. Clicking on the users would indicate which role they belong too, such as the user_u belongs to object_r, system_r, and unconfined_r.

The last two tabs mentioned in this paper are Booleans and MLS, which by now you should have some understanding of them. Click on Booleans, all of the “True” and “False” statements specified. At will you may change the value of these at any time. Checkmark the

Booleans label under Search Options and go over some Booleans. You will be able to see the default and current state of each one. On the MLS tab, notice there is only a single (1) specified sensitivity, the reason has been already mentioned previously; under categories however there's hundreds. Categories are only there to allow the security writer to use them, even though the writer may specify more or different categories.

Hopefully by now you should have some basic understanding of SELinux and how it works. For simplicity, I intentionally skipped through many other details of SELinux; such as Architecture, Constraints, Object Labeling, Policy Syntax, and compiling policies because to explain each one will be beyond the scope of this paper, I would suggest reading *SELinux by example* [4] as it explains everything I just explained but in finer detail. If you are interesting in furthering your knowledge on SELinux and begin writing your own policies in a nice GUI format, I would suggest trying out [5]. SELinux is a beast getting easier only after it has been tamed.

## Sources

[1] Caplan, D., MacMillan K., Mayer, F. (2006). SELinux Concepts. Retrieved August 21, 2009

from informit Web site: http://www.informit.com/articles/article.aspx?p=606586&seqNum=2

[2] Nakamura, Y. (2006). SELinux Policy Editor RBAC (Role Based Access Control) guide (for Ver 2.0)). Retrieved August 21, 2009 from sourceforge Web site: http://seedit.sourceforge.net/doc/2.0/rbac_guide.pdf

[3] Multi-Level Security (MLS). Retrieved August 21, 2009 from centos Web site: http://www.centos.org/docs/5/html/Deployment_Guide-en-US/sec-mls-ov.html

[4] Mayer, F., Macmillan K., Caplan D. (2007). SELinux by Example. Prentice Hall. Also available at http://www.selinuxbyexample.com/

[5] SELinux Policy Editor. Retrieved August 21, 2009 from sourceforge Web site: http://seedit.sourceforge.net/