Security Enhanced Linux – Differences in Coding Practices

SDEV-385 Operating Systems Architecture

Homework Week 13

November 27, 2020

Jarred Glaser

Security enhanced Linux provides additional security features for protecting the operating system from malicious attacks. Consequently, programmers must be aware of some differences in coding practices when working in a Linux environment configured with SE Linux. This paper will discuss the unique features of SE Linux, and some considerations programmers must take when working in this environment.

Security Enhanced Linux adds security features to Linux that make it more resilient against attacks from unauthorized users and systems. One helpful feature is the ability to control access on more variables such as which users or applications can access certain resources. Another feature is that Linux access controls (-rwxr-xr-x) are not modifiable by the individual user or user programs. Instead, they are controlled by policies that are loaded onto the system. SE Linux also provides finer granularity to access controls. Instead of just the main read, write, and execute access controls, there are also additional specifications which control things like who can unlink, append, or move a file. Finally, access to other resources such as network resources can be more strictly controlled in a SE Linux configuration. These are only scratching the surface of all that is available in SE Linux, but they do provide a high-level overview, and include some of the important features that we should think about when programming in an SE Linux environment.

One important consideration when programming in SE Linux is around access controls. It is not uncommon for C/C++ programs to change, edit, move, and work with files and filesystems. These usually include system calls, and often require the calling program to edit the access controls of the file. A simple example of doing this in a regular Linux environment can be seen below:

|  |
| --- |
| #include <sys/types.h>  #include <sys/stat.h>  int main() {  chmod("./myfile", S\_IRWXU); // enables owner to rwx file  } |

However, in SE Linux access controls are more restricted, and we may not have as much control over what files and directories can have their permissions changed. In this case, we would want to make sure we understand what access control mechanism the environment is using. For instance if the system is using Mandatory Access Control (MAC), which many SE Linux distributions enforce, then the user will not have access to the files they create by default, and necessary access will need to be granted before the access controls can be changed. MAC would break a program using the above code snippet, so programmers would need to control for this behavior.

Another common programming consideration is around network connections and ports. If a programmer is designing a web server that needs to accept incoming requests on a system with SE Linux, special considerations will need to be taken. By default, SE Linux policies only allow services access to recognized ports associated with the service. If we wanted to add a new port that our server can listen on, we would have to add a rule to allow it. We can do this through a command like the following example where we open the http port 81:

|  |
| --- |
| # semanage port -a -t http\_port\_t -p tcp 81 |

If the ports for our web server do not have rules properly set up, our code will run into many issues when attempting to receive requests.

One final example of a consideration users should take when programming in SE Linux is attempting to execute protected memory. SE Linux restricts execution of certain areas of memory operations depending on the configuration. When this is true, if a program tries to execute in protected memory, an execmod error will occur. One way to safely work around this is to copy the are of memory into a new location, map it, and execute the second one. Essentially, we “generate” the code in an unprotected area for execution.

In conclusion, Security Enhanced Linux is a powerful configuration that can substantially increase the overall security of a Linux system. There are many additional configurations that come with SE Linux that allow system admins to have much more detailed control of different security details in the system. When programming in an SE Linux environment, programmers should be aware of some additional hurdles such as handling access controls, server connections, and non-executable memory locations.

**References**

(2020). Chapter 43. Security and SELinux. Retrieved from <http://web.mit.edu/rhel-doc/5/RHEL-5-manual/Deployment_Guide-en-US/selg-overview.html#sec-acm-ov>

(2020). SELinux Project Wiki. Retrieved from <http://www.selinuxproject.org/page/Main_Page>

(2006). SELinux Memory Protection Tests. Retrieved from <https://akkadia.org/drepper/selinux-mem.html>

(2020). SELinux. Retrieved from <https://wiki.centos.org/HowTos/SELinux#Allowing_Access_to_a_Port>

(2020). Week 13: Lecture - Linux Security [Lecture Notes]. Retrieved from <https://champlain.instructure.com/courses/1414532/pages/week-13-lecture-linux-security?module_item_id=62587571>