# Linux Network Service Confinement Exercise

## Goal

Familiarize yourselves with the SE Linux tools and Linux capabilities. Exercise SE Linux type enforcement to confine a network program. Use capabilities and setuid to perform least privilege.

## Due

Submit your deliverables on compass by midnight February 16.

## Your Environment

Each machine has a Linux VM (Fedora Core 14) with the appropriate packages installed. Work in that VM.

## Scenario

Ponder Stibbons and his students from the High Energy Magic department have created a "Magic 8" service they wish to deploy on the Unreal University's main server. Given past experiences of other items created by this group, the system administrator is understandably concerned about deploying such an untested program on the production server.

The Unreal University has contacted you to investigate several ways to deploy the Magic 8 server in such a way to minimize harm to the rest of the system when things go wrong.

The Magic 8 server listens on port 77 and accepts TCP requests. Since it is listening on a low port, it needs to run as root. It needs to be able to read a configuration file (responses.txt) and write a log file (magic8.log).

Implement the following:

1. Build a type enforcement policy module to implement the network service confinement. Make sure to test in system enforcing mode (see "Enabling SE Linux" below) and test with the specific policy enforcing.
2. Use the setuid system to limit privileges needed to run the server.
3. Use the Linux capabilities to limit privileges needed to run the server.

### *Deliverables*

1. Submit your type enforcement policy files (.te, .if, and .fc). Describe how you created and tested the policy (e.g., how you used the wizard, and changes beyond the generated policy). Provide an audit or messages log entry showing how injected code violating the file access policy described above would be noted.
2. Describe your strategy for implementing the setuid and capability approaches to limiting the privileges required by magic8d. At the end, did your magic8d server still have to run as root at any point? What privileges did you need to create your semi-privileged magic8 server?

3. Submit any code changes you made to implement the setuid and capability privilege limitations.
4. Compare and contrast these three approaches. Do you need all three? Or does one approach eliminate the need for the other two approaches? What set of techniques would you suggest to deploy the Magic 8 service with the least negative security impact?

## Capabilities and Setuid

Refer back to the lecture on access control for pointers for using setuid mechanisms.

That lecture also refers to capabilities at a fairly high level. Your VM has the libcap and libcap-ng installed. The libcap-ng utilities are also installed and worth looking at:

- Filecap – View and set capabilities associated with executables
- Netcap – View the capabilities associated with network-facing processes
- Pscap – View the capabilities associated with all privileged, executing processes

## SE Linux Guidance

### *Some Introductory Exercises*

First, work through a few basic scenarios on the basic targeted policy. These exercises are from the "Fedora 13 Security-Enhanced Linux User Guide" which can be downloaded from http://docs.fedoraproject.org/en-US/Fedora/13/html/Security-Enhanced_Linux/index.html

Basic file and process security contexts.

1. Log in as alice. Run "ls -l" and "ls -Z". What is the security context of the files in Alice's directory?
2. Run "id -Z" as Alice. What is Alice's security context?

How is password changing constrained?

1. Run "ls -Z /usr/bin/passwd" and "ls -Z /etc/shadow".
2. Run passwd as Alice. Stop at the password prompt. In another shell window run "ps -eZ | grep passwd". What type is the password process running under? What type enforcement rules do you think direct password access?

How is a basic service confined?

1. As root, run "touch /var/www/html/testfile" and "ls -Z /var/www/html/testfile".
2. Start web server, "/etc/init.d/httpd/start".
3. As Alice run "wget http://localhost/testfile". Did it work?
4. As root run "chcon -t samba_share_t /var/www/html/testfile". This should change the type of the file at least until the next reboot.
5. Have Alice run wget again. Did it work?
6. Look at the logs at the end of /var/log/messages or /var/log/audit/audit.log.

Create a confined user.

1. By default, users in targeted policy are unconfined. Start “SE Linux Management” GUI from the System>Admin menu. Look at the users and the user mappings.
2. From the shell as root, use “useradd test-guest” to create a new user. “passwd test-guest” to give the new user a password (test).
3. From the SE Linux Management GUI, add a new login mapping entry that maps test-guest to xguest_u.
4. Login as test-guest.
5. Try to access http://illinois.edu three ways: via firefox, via wget, and via telnet with port 80 specified as the third argument. Which, if any approaches worked?
6. Try to invoke “su – alice”. Did it work? What is the security context of test-guest and /bin/su?

## *Policy editing*

With FC5 and beyond, modular policy is supported. You can use the Policy Generation GUI to get started on creating your own module to define policy for an application, daemon, or set of users. For each module there are potentially three files (actually four created by the policy generation gui).

- foo.te – This is the main and only required file. It includes the AV statements and any supporting type and role definitions.
- foo.if – The interface file. If your policy requires other modules to access what you define, you will need to create an if file.
- foo.fc – The file context. Describes the base labels of files. You may need to adjust this file during your policy developement
- foo.sh – Generated by the policy gui to invoke the appropriate build tools to compile and load your new module.

The type enforcement files use many M4 macros. Most of the macros are defined in the policy/support directory.

## *Confining magic8d*

There is a magic 8 subdirectory alice's home directory. It is installed at /usr/local/magic8. There is a magic8d, which is the fortune telling server and magic8, the fortune telling client. Launch magic8d in one window. From another window run **`magic8 localhost “Am I happy?”`**. It should return with a message chosen from response.txt.

The goal is to confine the server program, so magic8d can only access the files response.txt and magic8.log directly.

Use the “Selinux Policy Generation Tool” (under System) to get a starting point for the new magic8d policy module. This starts you through a wizard. Select the “User application” option to get started. Step through the wizard remembering what port your server listens on and what files it needs to access.  Remember that you have installed the program in /usr/local/magic8.  The program also writes syslog messages.

After completing the wizard, it will create a number of policy files for you.  Review the .te and .fc files.  If all looks good, execute the .sh file as root, which should compile and install your policy. The type policy is generated to run in permissive mode so you can debug the policy.  Make sure to do your tests with the “permissive” commented out.

Once the module successfully loads, launch magic8d.  Use “ps -eZ” to verify that magic8d is running under the special confined type magic8d.

Connect as a client.  Look in /var/log/messages and /var/log/audit/audit.log.  Is there anything interesting?  Look at Selinux Troubleshooter to get the detailed messages. The module is generated to run permissive.  Change policy and reload to get rid of errors that look detrimental.  Once things look good, comment out the permissive statement in the .te file and reload.  Does it work?  Are there any other log messages?

The default generated policy can be tightened up.  For example, magic8d only needs to read and not write response.txt.  Tune the policy to reduce access required by magic8d.

### *Enabling SELinux*

The /selinux portion of the file system is mapped to the runtime memory of the SELinux system much like the /proc file system maps out controls to the rest of the Linux system. The /selinux/enforce file controls whether the security server really enforces the policy or not.  The lab systems are configured to operate in *permissive* mode. This means the security server will be run, but the results will never really restrict access.  Instead only the error message will be logged, but the operation will be permitted.   Use the **getenforce** and **setenforce** commands to change the system between enforcing and permissive states.

## References

- Configuring the SELinux Policy, http://www.nsa.gov/research/_files/selinux/papers/policy2-abs.shtml - Up to date reference on the core policy language statements, but it does not address modular policy, MLS, or MCS.  The build description also addresses the old monolithic model.
- RedHat documentation on building, compiling, and loading your modular policy - http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5/html/Deployment_Guide/sec-sel-policy-customizing.html
- Joshua Brindle's description of how modular policies are implemented http://securityblog.org/brindle/2006/07/05/selinux-policy-module-primer/
- Libcap-ng project page - http://people.redhat.com/sgrubb/libcap-ng/