Cybersecurity

Subject: Restricted execution environment

A straightforward way to reduce the risk of the system compromise, is to restrict the privileges and resources available to users. In many situation, even system administrators do no need to make continual use of their superuser privileges all the time and all-system-wide. Here is, where the resource limits and privilege delegation become handy.

On Unix/Linux systems, the ulimit *mechanism can be used to restrict the application execution environment, including the operating system shell. On the other hand, it is possible to temporarily elevate regular-user privileges, by delegating selected administrative permissions to them. This can be achieved using two mechanisms. The first are the SUID and SGID file attributes, the second is the* sudo *policy.*

1. Resource limits

The resource limit mechanism in Linux allows the system to constrain the working environment of selected user accounts. Thus, when a user application is running on such a restricted account, the resource limit mechanism will enforce restrictions on the use of system resources available to this application.

Resource limits can be set up using the ulimit (*user limit*) command. It is also possible to use the PAM[1] mechanism, which allows to configure global limits for users and groups of the operating system.

1.1 Limits mechanism capabilities

The limits mechanism may alter the following environmental parameters:

- maximum number of simultaneously running processes,
- maximum amount of memory used,
- maximum CPU time consumption,
- amount of memory occupied by data,
- maximum number of simultaneously open files
- maximum file size,
- maximum core file size,
- maximum size of occupied resources,
- maximum stack size,
- address space size (∗),
- maximum number of parallel logons to the system (∗),
- user processes priorities (∗),
- maximum number of file locks,
- maximum number of pending signals to be sent,
- maximum amount of memory for message queues,
- maximum priority to which the user process priority can be increased (∗),
- maximum real-time priority (∗).

Limits marked with asterisks (∗) are only available via PAM configuration.

[1] PAM (*Pluggable Authentication Modules*) will be subject to some further exercise

Hard limits are imposed by the system administrator. The user himself can tighten them even more within the limits of the *soft* limits, i.e. he can limit himself even more.

2. SUID and SGID mechanisms

We can assign a substitution flag (SUID and SGID bit) with an executable program. Setting it causes the operating system to execute the program with a different effective UID (EUID) or GID (EGID) than those of the invoking user (allowing this way to acquire different access permissions). If SUID bit has been set, the program executes with the EUID of its owner, and the SGID bit specifies that the program will run with the EGID of its designated group.

The SUID and SGID bits can be set with the use of the well-known chmod program, e.g.:

```
chmod u+s file
```

sets the SUID bit, whereas:

```
chmod g+s file
```

sets the SGID bit.

The SUID and SGID are simple tools for temporal privilege elevation. Unfortunately, the use of this mechanism is very risky, and only in few cases it is truly necessary, e.g. for the standard passwd command.

3. Impersonation (sudo)

The sudo impersonation mechanism was designed to allow selected regular users to execute commands with explicitly defined privileges of another account, e.g. the root account. The sudo policy specifies which programs can be run by which users, and with whose privileges (along with any additional conditions, such as requirement of additional authentication at the command invocation).

3.1 Sudo configuration

☝ The sudo policy configuration file /etc/sudoers should be edited only (!) with the special visudo command.

This file contains two parts:

- optional alias definition (in fact multivalued variables)
- policy rules defining who can run what and with what rights.

There are 4 types of aliases:

- user_alias
- runas_alias
- host_alias
- cmnd_alias

Each type of alias refers to specific values, e.g. user_alias can contain names of users and groups. Below are some simple examples:

```
User_Alias    FULLTIMERS = agent007, jbond
User_Alias    ADVUSER = sherlock, hercule
Runas_Alias   OP = root
Runas_Alias   DB = oracle
Host_Alias    ORACLE = dblab, ora_srv, baza
Host_Alias    CS = 150.254.0.0/16
Cmnd_Alias    KILL = /usr/sbin/kill
Cmnd_Alias    SHELLS = /bin/sh, /bin/bash
```

The policy rules define which user can execute particular commands with permissions of a specified *target account*. By default, if the target account is not specified, the command will be run with administrator

privileges. A policy rule can also specify whether the user will have to enter a password to run a command with target's privileges. For instance:

```
FULLTIMERS    CS =  NOPASSWD: SHELLS, KILL
```

– specifies that `FULLTIMERS` (here, `agent007` and `jbond` users) can execute `SHELLS` and `KILL` commands with `root` privileges on `CS` computers without the need for authentication (password);

```
sherlock      ALL = (DB) NOPASSWD: SHELLS
```

– according to this rule, user `sherlock` can execute `SHELLS` commands with `DB` privileges (here, `oracle` account) on all computers without a password;

```
ADVUSER       ALL, !ORACLE = (www) ALL, (OP) KILL, (DB) /bin/ls
```

– `sherlock` and `hercule` users, on all computers except `ORACLE`, can execute all commands with `www` user privileges, as well as `KILL` commands with `OP` privileges, and the `/bin/ls` command with `DB` privileges.

According to the policy rules above, user `hercule` is allowed to execute e.g. the following command:

```
hercule@db2:~> sudo -u www ls -l /var/oracle/secret/
```

More examples, with appropriate descriptions, can be found in the user manual for `sudoers(5)`.

🕮 Resources:

http://wiki.archlinux.org/index.php/sudo
http://www.sudo.ws/sudo/sudo.man.html
http://www.sudo.ws/sudo/sudoers.man.html
http://www.sudo.ws/sudo/visudo.man.html

🗒 Problems to discuss:

- How are *hard* and *soft* limits related?
- Why is it necessary to edit the sudo policy with the dedicated command, visudo?
- Whose password will be required, if the sudo policy requires a password to run a command: that of the user invoking the command, or that of the target account?
- What role play SUID and SGID bits in the case of non-executable files (i.e. non binary-code programs)?
- What are the most important advantages and disadvantages of the above mechanisms?
- Would it be worth to expand any of these mechanisms? What functionality should be added?