7.3 Configuring Linux

An in-depth review of Linux security would be a lengthy task indeed. One reason is the diversity of Linux setups. Users could be using Debian, Red Hat, Ubuntu, or other Linux distributions. Some might be working from the shell, while others might be working from some graphical user interfaces such as KDE or GNOME. Fortunately, many of the same security concepts that apply to Windows can be applied to Linux. The only differences lie in the implementation, as explained in the following list:

- User and account policies should be set up the same in Linux as they are in Windows, with only a few minor differences. These differences are more a matter of using different names in Linux than in Windows. For example, Linux does not have an administrator account; it has a root account.
- All services (called daemons in Linux) not in use should be shut down.
- The browser must be configured securely.
- You must routinely patch the operating system.

In addition to some tactics that are common to Windows and Linux, a few approaches are different for the two operating systems:

- No application should run as the root user unless it is absolutely necessary. Remember that the root user is equivalent to the administrator account in Windows. Also, remember that all applications in Linux run as if started by a particular user, and therefore having an application run as root user would give it all administrative privileges.
- The root password must be complex and must be changed frequently. This is the same as with Windows administrator passwords.
- Disable all console-equivalent access for regular users. This means blocking access to programs such as shutdown, reboot, and halt for regular users on your server.
- Hide your system information. When you log in to a Linux box, it displays by default the Linux distribution name, version, kernel version, and the name of the server. This information can be a starting point for intruders. You should just prompt users with a “Login:” prompt.

7.3.1 Disable Services

Every running service (daemon) executes a code on the server. If there is a vulnerability within that code, it is a potential weakness that can be leveraged by an attacker; it also consumes resources in the form of RAM and CPU cycles.

Many operating systems ship with a number of services enabled by default, many of which you may not use. These services should be disabled to reduce the attack surface on your servers. Of course you should not just start disabling services with reckless abandon—before disabling a service, it is prudent to ascertain exactly what it does and determine if you require it.

There are a number of ways to ascertain which services run on a UNIX system, the easiest of which is to use the "**ps**" command to list any running services. Exact argument syntax can vary between versions, but the "**ps ax**" syntax works on most systems and will list every processes that would be running at that given time. Regarding minor variations in syntax on your operating system, check the manual page for "**ps**" using the command "**man ps**".

Services should be disabled in start-up scripts ("**rc**" or "**init**", depending on operating system) unless your system uses "**systemd**", in which case you can refer to the following discussion on "**systemd**". The "**kill**" command will merely stop any service that is running during that specific time, which will start once more during a reboot. The commands on Linux are typically one of: "**rc-update**", "**update-rc.d**", or "**service**". On BSD-based systems, you typically edit the file /etc/rc.conf.

For example, on several flavours of Linux the service command can be used to stop the sshd service: **service sshd stop**

To start sshd (one time): **service start sshd**

And to disable it from starting after a reboot:**update-rc.d -f sshd remove**

Some Linux distributions have moved towards using "**systemd**" as opposed to SysV startup scripts to manage services. "**systemd**" can be used to perform other administrative functions with regards to services, such as reloading configuration and displaying dependency information.

To stop sshd (one time): **systemctl stop sshd**
To enable sshd upon every reboot: **systemctl enable sshd**

And to disable sshd upon further reboots: **systemctl disable sshd**

Older Unix/Linux operating systems may use inetd or xinetd to manage services rather than rc or init scripts. (x)inetd is used to preserve system resources by being almost the only service running and starting other services on demand, rather than leaving them all running all of the time. If this is the case, services can be disabled by editing the inetd.conf or xinetd.conf files, typically located in the /etc/ directory.

7.3.2 File Permissions

Most Unix/Linux file systems have a concept of permissions—that is, files which users and groups can read, write, or execute. Most of them also have the SETUID (set user ID upon execution) permission, which allows a nonroot user to execute a file with the permission of the owning user, typically root. This is because the normal operation of that command, even to a nonroot user, requires root privileges, such as su or sudo.

Typically, an operating system will set adequate file permissions on the system files during installation. However, as you create files and directories, permissions will be created according to your umask settings. As a general rule, the umask on a system should only be made more restrictive than the default. Cases where a less restrictive umask is required should be infrequent enough so that chmod can be used to resolve the issue. Your umask settings can be viewed and edited using the umask command. See man umask1 for further detail on this topic.

Incorrect file permissions can leave files readable by users other than whom it is intended for. Many people wrongly believe that because a user has to be authenticated to log in to a host, leaving world or group readable files on disk is not a problem. However, they do not consider that services also run using their own user accounts.

Take, for example, a system running a web server such as Apache, nginx, or lighttpd; these web servers typically run under a user ID of their own such as “www-data.” If the files you create are readable by “www-data”, then, if they are configured to do so, accidentally or otherwise, the web server has permission to read that file and to

potentially serve it to a browser. By restricting file system-level access, we can prevent this from happening—even if the web server is configured to do so, as it will no longer have permission to open the file.

As an example, the file test can be read and written to by the owner _www, it can be read and executed by the group staff, and can be read by anybody. This is denoted by the rw-, r-x, and r-- permissions in the directory listing:

```
$ ls -al test

-rw-r-xr--  1 _wwwstaff  1228 16 Apr 05:22 test
```

In the Unix/Linux file system listing, there are 10 hyphens (-), the last 9 of which correspond to read, write, and execute permissions for the owner, group and other (everyone). A hyphen indicates the permission is not set; a letter indicates that it is set. Other special characters appear less often; for example, an S signifies that the SETUID flag has been set.

If we wish to ensure that others can no longer see this file, then we can modify the permissions. We can alter them using the chmod command (o= sets the other permissions to nothing):

```
$ sudo chmod o= test

$ ls -la test

-rw-r-x---  1 _wwwstaff  1228 16 Apr 05:22 test
```

7.3.3 File Integrity

File Integrity Management tools monitor key files on the file system and alert the administrator in the event that they change.These tools can be used to ensure that key system files have not been tampered with, as in the case of a rootkit, and that files have not been added to directories without the administrator’s permission, or configuration files have been modified, as can be the case with backdoors in web applications, for example.

There are both commercial tools and free/open source tools available through your preferred package management tool. Examples of open source tools that perform file integrity monitoring include Samhain and OSSEC. If you wish to spend money to obtain extra features like

providing integration with your existing management systems, there are also a number of commercial tools available.

Alternatively, if you cannot for whatever reason to install file integrity monitoring tools, many configuration management tools can be configured to report on modified configuration files on the file system as part of their normal operation. This is not their primary function and does not offer the same level of coverage, and so it is not as robust as a dedicated tool. However, if you are in a situation where you cannot deploy security tools but do have configuration management in place, this may be of some use.

7.3.4 Separate Disk Partitions

Disk partitions within Unix/Linux can be used not only to distribute the file system across several physical or logical partitions, but also to restrict certain types of action depending on which partition they may take place on. Options can be placed on each mount point in /etc/fstab.

There are some minor differences between different flavours of Unix/Linux with regards to the options, and so consulting the system manual page—using man mount—before using options is recommended.

Some of the most useful and common mount point options, from a security perspective, are:

nodev

Do not interpret any special dev devices. If no special dev devices are expected, this option should be used. Typically only the /dev/ mount point should contain special dev devices.

nosuid

Do not allow setuid execution. Certain core system functions, such as su and sudo will require setuid execution, thus this option should be used carefully. Attackers can use setuid binaries as a method of backdooring a system to quickly obtain root privileges from a standard user account. Setuid execution is probably not required outside of the system-installed bin and sbin directories. You can check for the location of setuid binaries using the following command:

$ sudo find / -perm -4000

Binaries that are specifically setuid root, as opposed to any setuid binary, can be located using the following variant:

$ sudo find / -user root -perm -4000

ro

Mount the file system read-only. If data does not need to be written or updated, this option may be used to prevent modification. This removes the ability for an attacker to modify files stored in this location such as config files and static website content.

noexec

Prevents execution, of any type, from that particular mount point. This can be set on mount points used exclusively for data and document storage. It prevents an attacker from using it as a location to execute tools he may have loadedonto a system in order to be able to defeat certain classes of exploit.

7.3.5 Chroot

chroot alters the apparent root directory of a running process and any children processes. The most important aspect of this is that the process inside the chroot jail cannot access files outside of its new apparent root directory, which is particularly useful in the case of ensuring that a poorly configured or exploited service cannot access anything more than it needs to.

There are two ways in which chroot can be initiated:

The process in question can use the chroot system call and chroot itself voluntarily. Typically, these processes will contain chroot options within their configuration files, most notably allowing the user to set the new apparent root directory.

The chroot wrapper can be used on the command line when executing the command. Typically this would look something like:

sudo chroot /chroot/dir/ /chroot/dir/bin/binary -args

For details of specific chroot syntax for your flavor of Unix, consult man chroot.3

It should be noted, however, that there is a common misconception. It is often misunderstood that chroot offers some security features. To be accurate it does not. Chroot jails are not impossible to break out of,

especially if the process within the chroot jail runs with root privileges. Typically processes that are specifically designed to use chroot will drop their root privileges as soon as possible so as to mitigate this risk. Additionally, chroot does not offer to the process any protection from privileged users outside of the chroot on the same system.

Neither of these are reasons to abandon chroot. On the contrary; they should simply be taken into consideration when designing use cases as it is not an impenetrable fortress. To be precise, it is a method that reinforces restriction to file system access.