Chapter 16

Securing the system step by step

The Linux system was created in 1991. It was already popular 15 years ago due to its stability and simplicity. It is also considered to be one of the safest operating systems. This doesn't mean, however, that the security of the system after installation can be ignored. This chapter contains several fundamental steps that should be taken into consideration when installing Linux. The system securing techniques presented here are relevant to each popular distribution of the Linux system.

Preparation of the hard disk

During installation of the system we face the issue of disk partitioning. The majority of beginning users leave the matter to the partition creator, which will not always divide our disk as we would like it to. It is therefore worth stopping here to think what would be the best way to partition the disk for a system in which security has priority.

There are many reasons to create several partitions. Below we present a safe way to divide the disk. There is no performance penalty in partitioning in this way, and as the result of this we will obtain four partitions, which will help increase the security of our environment. The partitions we should create are, in sequence:

a) /swap

The swap partition is an exchange partition. When the system lacks RAM memory it writes the information to disk, specifically to the swap file. The size of the partition should, therefore, depend on the size of the operating

memory available. It is a rule of thumb that the system works optimally when the swap file is twice the size of available RAM.

b) /boot

The boot partition is the place where the kernel of our system and the so-called "boot loader" are. The boot loader is a program that starts up right after switching on the computer. It loads the kernel into memory and then it starts it up. The boot partition shouldn't have big dimensions. A frequent habit of hackers is to place rootkits in this partition. Maintaining a small size of the boot partition also develops good habits, such as optimal compilation of the system kernel. The size of this partition can therefore vary within the limits of around 20 to 40 MB.

c) /var

The var directory is a place where, for example, various kinds of system logs are kept. A primitive hacker could clog our disk with unnecessary logs, and consequently could cause system immobilization. If the /var directory is located in a separate partition, and somebody performs such an attack, he will not be able to fill up the whole disk, but only this partition, while the computer will continue to work. The size of this partition should amount from several to many gigabytes, depending on the general capacity of our disk and the installed applications.

d) /

This is the main partition of our system. In this location the system, and the programs that the reader chooses to add to the computer, will be installed. It is good to dedicate the entire space remaining on the hard disk to this partition.

If we install the system on the production server, to which many users will have access, it is worth thinking about a separate partition for the folder /home. Data belonging to the system users are located in it. Creating this

partition can prove to be an efficient and safe move. If the reader is using the Linux system on a home computer, a /home partition is superfluous.

Choice of installation

After partitioning the disk it is time to choose the programs to install on the system. The "full installation" option is surely convenient. However, it is not a secure solution. Each popular Linux distribution gives us the ability to choose programs to install individually. Our first step should be to get to know all the software programs and to decide if they are necessary for us or not. For each of the programs (packages), a short description of its content and destination should be available. We should remember that a smaller quantity of installed software provides fewer opportunities to a hacker to gain control over the system. If for some reason we have to choose the full installation, we should be prepared for the fact that our system is not as safe as it could be. Most distributions contain tools enabling us to add or remove programs, including those that are already installed. It is good to review the list of installed packages from time to time and to remove the ones that are not used often.

Administrator's password

Creating an appropriate password is literally a key issue for the security of our system. We cannot forget the great importance of having a difficult password. Even the most complicated protection is superfluous if it is secured by an easy password. Therefore in order for cracking of the password to be impossible it has to have at least 12 characters, and must contain letters, numbers, and special characters. We shouldn't choose passwords that have any meaning. The best password should be a random character sequence. Below are examples of easy and difficult passwords:

Easy passwords: sex, security, 1234567890qwer
Difficult passwords: lD45&fG^%a, F6gV$xNio8&, 31##7cVvF69666

Firewalls

A firewall is a mechanism to filter the traffic in the network. Let's assume that we want a WWW server (port 80) and FTP (port 21) to run on our computer. We don't want to install any other services. It would be good to filter all incoming traffic directed to ports other than 80 and 21. That way, a cracker will have fewer possibilities of performing an attack. The best tool working under the Linux system that is suitable for this purpose is iptables.

Here is an example iptables script, which should start up each time the computer is switched on (**/CD/Chapter16/Listings/firewall.sh**):

```
#!/bin/bash

# There are two rules determining typical behavior.
# If the packet doesn't meet any of the rules we reject all
# incoming and redirected packets.
iptables -P INPUT DROP
iptables -P FORWARD DROP

# Rules are added (-A) to the incoming network. They let through
# packets coming from the established connection and packets of the interface
# loopback.
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -s 127.0.0.1 -d 127.0.0.1 -i lo -j ACCEPT

# This rule added to the INPUT network accepts WWW connections.
iptables -A INPUT -p tcp --dport 80 -i eth0 -j ACCEPT

# This rule added to the INPUT network, accepts FTP connections.
iptables -A INPUT -p tcp --dport 21 -i eth0 -j ACCEPT
iptables -A INPUT -p tcp --dport 20 -i eth0 -j ACCEPT
```

We will now try to write the above script and give it rights to execute:

```
[root@localhost]# chmod +x firewall.sh
```

Next, we need to start this up, like every other program, from the console level:

```
[root@localhost]# ./firewall.sh
```

To see if our firewall really works, we will now try to display all active iptables rules:

```
[root@localhost]# iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED
ACCEPT     all  --  localhost            localhost
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:http
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:ftp
ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:ftp-data

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

We will now analyze in sequence the operations performed on the incoming, outgoing, and redirected packets.

a) Incoming packets (INPUT)

As standard, all outgoing packets should be rejected (DROP), which is stated in the line Chain INPUT (policy DROP). The next lines tell which packets should be accepted (ACCEPT) for which host and port.

b) Outgoing packets (OUTPUT)

As standard, all outgoing packets should be let through (ACCEPT). Otherwise our system couldn't make any connections.

c) Redirected packets (FORWARD)

As standard, all redirected packets should be rejected (DROP). This is important for the servers carrying network traffic. As for the home computer, we should reject these packets, because in theory they shouldn't be appearing.

A firewall of this type should be perfectly sufficient for a home machine.

The xinetd superserver

Xinetd is an integral component of the Linux system. It is what is known as a network superserver. Its task is to listen in on certain ports and to transfer incoming connections to specific programs. We will now assume that we have an FTP server. It can work in two modes: as a separate service or as a service started up by xinetd. If many people use our FTP server, it would be best to command it to run constantly in the background as a separate service. However, if there are only a few logins, starting up our FTP server with each connection would be better in terms of performance. The inetd configuration file is /etc/xinetd.d. The syntax of its each line is:

```
service sane
{
        disable = no
        port = 6566
        socket_type = stream
        protocol = tcp
        wait = no
        user = saned
        group = saned
        server = /usr/sbin/saned
}
```

service	- Name of one of the services shown in the file /etc/services.
disable	- Service is active (yes) or not (no)
socket_type	- This can be either stream (for TCP sockets) or dgram (for UDP).
port	- Port number on which server is waiting for connection
protocol	- It is either TCP or UDP.
user/group	- ID of user/group whose rights will be used by the process.
server	- Access path to the proper server program

In the example mentioned above, xinetd daemon will start the Sane service, using the stream socket type and TCP protocol. The calls are to be taken immediately (wait = no), and the program /usr/sbin/saned will be executed with saned user privileges. The service will be launched on 6566th port. We can verify this in /etc/services file, which contains a list of all standard services.

```
$ cat /etc/services | grep sane
sane    6566/tcp #SANE Control Port
```

Knowing the structure of the configuration file, we can check whether the /etc/xinetd.d directory contains services that could be disabled (using disable = yes option.) It is recommended to keep this directory clean. We should not host the services we do not use.

SSH configuration

An advantage of Unix systems is their remote management ability. In the past telnet was used for this purpose. However, it wasn't encrypted, which was its biggest weakness. Another communication method, SSH (secure shell), was therefore developed, and it replaced telnet very rapidly. SSH is very secure and cracking it is almost impossible. Its only weak point can be the *client* program and servers that use it for communication. Therefore it's worth dedicating some time to configuring them safely.

The configuration file of the sshd server is by default /etc/ssh/sshd_config. Data related to such things as the protocol version and how to login using keys are found in it. Below is an example safe sshd configuration:

```
# Use of the safer second version protocol.
Protocol 2

# Don't allow root to login remotely.
PermitRootLogin no

# Request a password.
PasswordAuthentication yes

# Don't login users with an empty password.
PermitEmptyPasswords no

# Other less important options.
StrictModes yes
X11Forwarding no
PrintMotd no
```

In order for sshd to reload its configuration, it is necessary to execute the command below:

```
[root@localhost]# /etc/init.d/sshd restart
```

This sends information about reloading to the process with the PID number saved in the file /var/run/sshd.pid.

Hiding information and the SUID bit

We have to be sure that we don't leave any gaps in the system. There is no reason to give the user more information than he needs. A smart move is to remove or change all welcome messages (e.g., /etc/motd). It would be a good idea to change the content of the files /etc/issue and /etc/issue.net to, for instance, "Welcome to Windows XP." These files are printed on the screen by the program "login" before the user logs into the system, and if they present false system information they can mislead a potential cracker.

To change the current password the "passwd" command is used. This program saves the new password in the file /etc/shadow, to which we don't have access in practice. To be more precise, we don't have access, but the program is able to change our password. This is caused by the assigning of a so-called SUID bit to it.

```
[root@localhost]# ls -l /usr/bin/passwd
-r-s--x--x  1 root root 26168 2010-06-29 12:01 /usr/bin/passwd
```

This can be ascertained by looking at the sequence of its access rights, namely "-r-s--x--x." The "s" letter shows that the program has the SUID bit (+s) set. This program also belongs to the root user, who has access to the file /etc/shadow. Therefore, if we assign the bit +s to the program, it will work with the rights of the owner of the file being executed, in our case the administrator. Programs of this type often constitute a huge threat to the system. They are to be found abundantly in each distribution and the user should pay close attention to their function. Finding programs with the SUID bit set and removing this bit from them can significantly reduce a hacker's chances. Below is an example perl script, which will help us to simplify the process of removal of the SUID bits from the programs.

Let's save it as "sup.pl" (**/CD/Chapter16/Listings/sup.pl**).

```
#!/usr/bin/perl

if ($#ARGV < 0)
{
          system("find / -perm +4000 2>/dev/null > suid.txt");
          exit;
}

if ($ARGV[0] =~ /-u/)
{
         open(UPDATE, "<suid.txt") or die "Can't read file: $!";

         while (<UPDATE>)
         {
              if (/^#/)
              {

                s/#//;

                system("chmod -s $_");
               }
          }
          close(UPDATE);
}
{
         print "Usage: perl sup.pl <-u>n";
         exit;
}
                                          }
```

We assign execution rights to our script:

```
[root@localhost]# chmod +x sup.pl
```

Next, we start it up and wait for results.

```
[root@localhost] ./sup.pl
```

The script has tracked down all the programs with the set SUID bit on the disk and has written the result to the "suid.txt" file. Each of the programs found, if it contains an error, can be used by the hacker to gain full control over the system. Let's have a look at the list content:

```
[root@localhost] cat suid.txt
/bin/su
/bin/mount
/bin/umount
```

```
/bin/ping
/bin/ping6
/sbin/unix_chkpwd
/sbin/usernetctl
/usr/X11R6/bin/Xwrapper
/usr/bin/passwd
/usr/bin/sudoedit
/usr/bin/sudo
/usr/sbin/traceroute
```

If any of the programs doesn't require the SUID bit or is unnecessary for us, we can disable it with the # character. We only want users to have the ability to change their own passwords. Therefore we disable everything except "passwd." The file after modification should look like this:

```
[root@localhost] cat suid.txt
#/bin/su
#/bin/mount
#/bin/umount
#/bin/ping
#/bin/ping6
#/sbin/unix_chkpwd
#/sbin/usernetctl
#/usr/X11R6/bin/Xwrapper
/usr/bin/passwd
#/usr/bin/sudoedit
#/usr/bin/sudo
#/usr/sbin/traceroute
```

Next we start up our script with the -u parameter to get rid of unwanted bits:

```
[root@localhost] ./sup.pl -u
```

The SUID bit has just been taken away from the disabled programs. Let's check to be sure. For this purpose we can scan the disk once again to look for them:

```
[root@localhost] ./sup.pl
[root@localhost] cat suid.txt
/usr/bin/passwd
```

Everything is running according to our expectations.

The last thing we will do will be to allow the administrator to log on only from a specific terminal. The file /etc/securetty lists the devices on which the administrator can log in.

Let's have a look at the content of this file:

```
[root@localhost]# cat /etc/securetty
tty1
…
tty11
ttyS0
ttyS1
vc/1
...
vc/11
```

It is recommended that we disable (using the # character) all entries except the chosen terminal (e.g. tty4). The chances of someone with physical access to our computer being able to figure out the administrator's password are very low. It is likely that after the first unsuccessful attempt to log in on the first terminal, the person will become discouraged and our system will remain intact.

Software to improve system security

There are many applications created especially with the aim of improving the security of our system. The best source for them is the site:

```
http://packetstormsecurity.org
```

A useful program that can be found there is "ckrootkit." It checks to be sure our system is not infected with any rootkit and allows us to determine if there has been a system violation.

We can also install an intrusion detection system. IDS programs, however, are beyond the scope of this chapter and will be discussed separately. However, the snort program is definitely worth a look. It can be found at:

```
http://www.snort.org
```

This program is one of most popular and effective IDS tools. The installation and use of snort with its default settings are not complicated. Unfortunately, poor configuration can cause many false alarms.

The kernel – the system's weak point

Each operating system contains a kernel, which has access rights to all data located on the hard disk. If an attacker gains access to the kernel, he will also gain access to all other computer resources. Linux, like any other system, is not error-free. Once in a while, errors can occur in the system kernel itself. Unfortunately we cannot fully protect ourselves against them. The only solution is to follow the services that report errors in the software and to update the system each time an irregularity in the kernel is reported. To check the kernel version in use it is necessary to give the command:

```
[root@localhost]# uname -a
Linux computer 2.6.29 #1 Tue Jul 29 15:55:20 CEST 2010 i686 GNU Linux
```

As we can see, we have the kernel version number 2.6.29. It is a good idea to look at the page containing kernel resources and to update the system as often as possible. This is a very good habit to get into. The reader can find the current version of the kernel under the address:

```
http://www.kernel.org
```

What next?

This chapter is only a short introduction to the methods of securing the Linux system. We can do much more to make our system more secure. Configuring snort and writing rules adapted to our network are a good place to start. Writing complicated iptables scripts, verifying the integrity of the file system, and custom modification of kernel resources can significantly influence the security level of our system.