14
Security Tips and Tricks for the Busy Bee

In this final chapter, I'd like to do a roundup of some quick tips and tricks that don't necessarily fit in with the previous chapters. Think of these tips as time savers for the busy administrator. First, you will learn about some quick ways to see which system services are running, in order to ensure that nothing that isn't needed is running. Then, we'll look at how to password protect the GRUB 2 bootloader, how to securely configure BIOS/UEFI to help prevent attacks on a physically accessible machine, and how to use a checklist to perform a secure initial system setup.

In this chapter, we will cover the following topics:

- Auditing system services
- Password protecting the GRUB2 configuration
- Securely configuring and then password protecting UEFI/BIOS
- Using a security checklist when setting up your system

Technical requirements

The code files for this chapter are available here: `https://github.com/PacktPublishing/Mastering-Linux-Security-and-Hardening-Second-Edition`.

Auditing system services

A basic tenet of server administration, regardless of which operating system we're talking about, is to never have anything that you don't absolutely need installed on a server. You especially don't want any unnecessary network services running because that would give the bad guys extra ways to get into your system. And, there's always a chance that some evil hacker might have planted something that acts as a network service, and you'd definitely want to know about that. In this section, we'll look at a few different ways to audit your system to ensure that no unnecessary network services are running on it.

Auditing system services with systemctl

On Linux systems that come with systemd, the `systemctl` command is pretty much a universal command that does many things for you. In addition to controlling your system's services, it can also show you the status of those services, like so:

```
donnie@linux-0ro8:~> sudo systemctl -t service --state=active
```

Here's the breakdown of the preceding command:

- `-t service`: We want to view information about the services – or, what used to be called daemons – on the system.
- `--state=active`: This specifies that we want to view information about all the system services that are actually running.

A partial output of this command looks something like this:

```
UNIT                                                LOAD   ACTIVE SUB
DESCRIPTION
accounts-daemon.service                             loaded active running
Accounts Service
after-local.service                                 loaded active exited
/etc/init.d/after.local Compatibility
alsa-restore.service                                loaded active exited
Save/Restore Sound Card State
apparmor.service                                    loaded active exited
Load AppArmor profiles
auditd.service                                      loaded active running
Security Auditing Service
avahi-daemon.service                                loaded active running
Avahi mDNS/DNS-SD Stack
cron.service                                        loaded active running
Command Scheduler
. . .
. . .
```

Generally, you won't want to see quite this much information, although you might at times. This command shows the status of every service that's running on your system. What really interests us now is the network services that can allow someone to connect to your system. So, let's look at how to narrow things down a bit.

Auditing network services with netstat

The following are two reasons why you would want to keep track of what network services are running on your system:

- To ensure that no legitimate network services that you don't need are running
- To ensure that you don't have any malware that's listening for network connections from its master

The `netstat` command is both handy and easy to use. First, let's say that you want to see a list of network services that are listening, waiting for someone to connect to them (due to formatting restrictions, I can only show part of the output here. We'll look at some lines that I can't show here in just a moment. Also, you can download the text file with the full output from the Packt Publishing website):

```
donnie@linux-0ro8:~> netstat -lp -A inet
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
```

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name
tcp 0 0 *:ideafarm-door *:* LISTEN -
tcp 0 0 localhost:40432 *:* LISTEN 3296/SpiderOakONE
tcp 0 0 *:ssh *:* LISTEN -
tcp 0 0 localhost:ipp *:* LISTEN -
tcp 0 0 localhost:smtp *:* LISTEN -
tcp 0 0 *:db-lsp *:* LISTEN 3246/dropbox
tcp 0 0 *:37468 *:* LISTEN 3296/SpiderOakONE
tcp 0 0 localhost:17600 *:* LISTEN 3246/dropbox
. . .
. . .
```

Here's the breakdown:

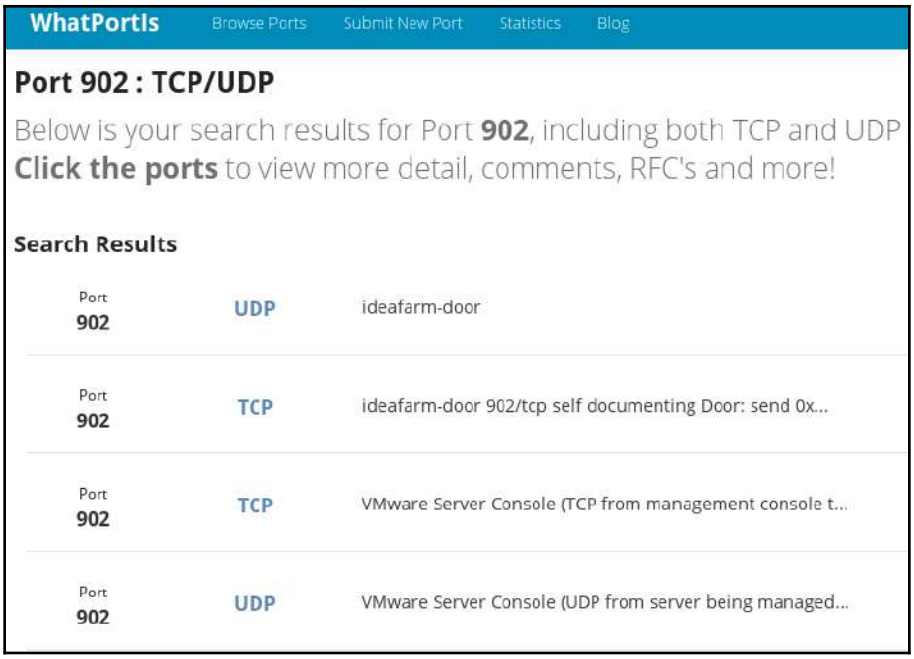
- `-lp`: The `l` means that we want to see which network ports are listening. In other words, we want to see which network ports are waiting for someone to connect to them. The `p` means that we want to see the name and process ID number of the program or service that is listening on each port.
- `-A inet`: This means that we only want to see information about the network protocols that are members of the `inet` family. In other words, we want to see information about the `raw`, `tcp`, and `udp` network sockets, but we don't want to see anything about the Unix sockets that only deal with interprocess communications within the operating system.

Since this output is from the openSUSE workstation that I just happen to be using at the moment, you won't see any of the usual server-type services here. However, you will see a few things that you likely won't want to see on your servers. For example, let's look at the very first item:

```
Proto Recv-Q Send-Q Local Address      Foreign Address         State
PID/Program name
tcp        0      0 *:ideafarm-door    *:*                     LISTEN
-
```

The `Local Address` column specifies the local address and port of this listening socket. The asterisk means that this socket is on the local network, while `ideafarm-door` is the name of the network port that is listening. (By default, `netstat` will show you the names of ports whenever possible by pulling the port information out of the `/etc/services` file.)

Now, because I didn't know what the `ideafarm-door` service is, I used my favorite search engine to find out. By plugging the term `ideafarm-door` into DuckDuckGo, I found the answer:



The top search result took me to a site named **WhatPortIs**. According to this, `ideafarm-door` is, in reality, port `902`, which belongs to the **VMware Server Console**. Okay, that makes sense because I do have VMware Player installed on this machine. So, that's all good.

You can check out the **WhatPortIs** site here: `http://whatportis.com/`.

Next on the list is the following:

```
tcp        0      0 localhost:40432     *:*         LISTEN
3296/SpiderOakONE
```

This item shows the local address as `localhost` and that the listening port is port `40432`. This time, the `PID/Program Name` column actually tells us what this is. `SpiderOak ONE` is a cloud-based backup service that you might or might not want to see running on your server.

Now, let's look at a few more items:

```
tcp 0      0 *:db-lsp                    *:*       LISTEN      3246/dropbox
tcp 0      0 *:37468                     *:*       LISTEN
3296/SpiderOakONE
tcp 0      0 localhost:17600             *:*       LISTEN      3246/dropbox
tcp 0      0 localhost:17603             *:*       LISTEN      3246/dropbox
```

Here, we can see that `dropbox` and `SpiderOakONE` are both listed with the asterisk for the local address. So, they're both using the local network address. The name of the port for `dropbox` is `db-lsp`, which stands for Dropbox LAN Sync Protocol. The `SpiderOakONE` port doesn't have an official name, so it's just listed as port `37468`. The bottom two lines show that `dropbox` also uses the local machine's address, on ports `17600` and `17603`.

So far, we've looked at nothing but TCP network sockets. Let's see how they differ from UDP sockets:

```
udp        0      0 192.168.204.1:ntp       *:*
-
udp        0      0 172.16.249.1:ntp        *:*
-
udp        0      0 linux-0ro8:ntp          *:*
-
```

The first thing to note is that there's nothing under the `State` column. That's because, with UDP, there are no states. They are actually listening for data packets to come in, and they're ready to send data packets out. But since that's about all that UDP sockets can do, there was really no sense in defining different states for them.

In the first two lines, we can see some strange local addresses. That's because I have both VMware Player and VirtualBox installed on this workstation. The local addresses of these two sockets are for the VMware and VirtualBox virtual network adapters. The last line shows the hostname of my OpenSUSE workstation as the local address. In all three cases, the port is the Network Time Protocol port, for time synchronization.

Now, let's look at one last set of UDP items:

```
udp        0      0 *:58102         *:*
5598/chromium --pas
udp        0      0 *:db-lsp-disc   *:*
3246/dropbox
udp        0      0 *:43782         *:*
5598/chromium --pas
udp        0      0 *:36764         *:*

udp        0      0 *:21327         *:*
```

```
3296/SpiderOakONE
udp        0      0 *:mdns          *:*
5598/chromium --pas
```

Here, we can see that my Chromium web browser is ready to accept network packets on a few different ports. We can also see that Dropbox uses UDP to accept discovery requests from other local machines that have Dropbox installed. I assume that port `21327` performs the same function for SpiderOak ONE.

Of course, since this machine is my workhorse workstation, Dropbox and SpiderOak ONE are almost indispensable to me. I installed them myself, so I've always known that they were there. However, if you see anything like this on a server, you'll want to investigate to see if the server admins know that these programs are installed, and then find out why they're installed. It could be that they're performing a legitimate function, and it could be that they're not.

A difference between Dropbox and SpiderOak ONE is that with Dropbox, your files don't get encrypted until they've been uploaded to the Dropbox servers. So, the Dropbox folk have the encryption keys to your files. On the other hand, SpiderOak ONE encrypts your files on your local machine, and the encryption keys never leave your possession. So, if you really do need a cloud-based backup service and you're dealing with sensitive files, something such as SpiderOak ONE would definitely be better than Dropbox. (And no, the SpiderOak ONE folk aren't paying me to say that.)

If you want to see port numbers and IP addresses instead of network names, add the `n` option. As before, here's the partial output:

```
donnie@linux-0ro8:~> netstat -lpn -A inet
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address      Foreign Address     State
PID/Program name
tcp        0      0 0.0.0.0:902        0.0.0.0:*           LISTEN      -
tcp        0      0 127.0.0.1:40432    0.0.0.0:*           LISTEN
3296/SpiderOakONE
tcp        0      0 0.0.0.0:22         0.0.0.0:*           LISTEN      -
tcp        0      0 127.0.0.1:631      0.0.0.0:*           LISTEN      -
tcp        0      0 127.0.0.1:25       0.0.0.0:*           LISTEN      -
tcp        0      0 0.0.0.0:17500      0.0.0.0:*           LISTEN
3246/dropbox
tcp        0      0 0.0.0.0:37468      0.0.0.0:*           LISTEN
3296/SpiderOakONE
```

```
tcp        0      0 127.0.0.1:17600    0.0.0.0:*           LISTEN
3246/dropbox
. . .
. . .
```

All you have to do to view the established TCP connections is to leave out the `l` option. On my workstation, this makes for a very long list, so I'll only show a few items:

```
donnie@linux-0ro8:~> netstat -p -A inet
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address      Foreign Address         State
PID/Program name
tcp        1      0 linux-0ro8:41670   ec2-54-88-208-223:https CLOSE_WAIT
3246/dropbox
tcp        0      0 linux-0ro8:59810   74-126-144-106.wa:https ESTABLISHED
3296/SpiderOakONE
tcp        0      0 linux-0ro8:58712   74-126-144-105.wa:https ESTABLISHED
3296/SpiderOakONE
tcp        0      0 linux-0ro8:caerpc  atl14s78-in-f2.1e:https ESTABLISHED
10098/firefox
. . .
. . .
```

The `Foreign Address` column shows the address and port number of the machine at the remote end of the connection. The first item shows that the connection with a Dropbox server is in a `CLOSE_WAIT` state. This means that the Dropbox server has closed the connection, and we're now waiting on the local machine to close the socket.

Because the names of those foreign addresses don't make much sense, let's add the `n` option to see the IP addresses instead:

```
donnie@linux-0ro8:~> netstat -np -A inet
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address         Foreign Address      State
PID/Program name
tcp        0      1 192.168.0.222:59594   37.187.24.170:443    SYN_SENT
10098/firefox
tcp        0      0 192.168.0.222:59810   74.126.144.106:443   ESTABLISHED
3296/SpiderOakONE
tcp        0      0 192.168.0.222:58712   74.126.144.105:443   ESTABLISHED
3296/SpiderOakONE
tcp        0      0 192.168.0.222:38606   34.240.121.144:443   ESTABLISHED
10098/firefox
```

```
. . .
. . .
```

This time, we can see something new. The first item shows a `SYN_SENT` state for the Firefox connection. This means that the local machine is trying to establish a connection to the foreign IP address. Also, under `Local Address`, we can see the static IP address for my openSUSE workstation.

If I had space to display the entire `netstat` output here, you'd see nothing but `tcp` under the `Proto` column. That's because the UDP protocol doesn't establish connections in the same way that the TCP protocol does.

Something to keep in mind is that rootkits can replace legitimate Linux utilities with their own trojaned versions. For example, a rootkit could have its own trojaned version of `netstat` that would show all network processes except for those that are associated with the rootkit. That's why you want something such as Rootkit Hunter in your toolbox.

If you need more information about `netstat`, see the `netstat` man page.

Hands-on lab – viewing network services with netstat

In this lab, you'll practice what you've just learned about `netstat`. Do this on a virtual machine that has a desktop interface so that you can use Firefox to visit websites. Follow these steps:

1. View the list of network services that are listening for a connection:

```
netstat -lp -A inet
netstat -lpn -A inet
```

2. View the list of established connections:

```
netstat -p -A inet
netstat -pn -A inet
```

3. Open Firefox and navigate to any website. Then, repeat *Step 2*.
4. From your host machine, log into the virtual machine via SSH. Then, repeat *Step 2*.

You've reached the end of the lab – congratulations!

You've just seen how to audit network services with `netstat`. Now, let's learn how to do this with Nmap.

Auditing network services with Nmap

The `netstat` tool is very good, and it can give you lots of good information about what's going on with your network services. The slight downside is that you have to log in to every individual host on your network in order to use it.

If you'd like to remotely audit your network to see what services are running on each computer, without having to log in to each and every one, then you need a tool such as Nmap. It's available for all the major operating systems, so even if you're stuck having to use Windows on your workstation, you're in luck. An up to date version is built into Kali Linux, if that's what you're using. It's also in the repositories of every major Linux distro, but the version that's in the Linux repositories is usually quite old. So, if you're using something other than Kali, your best bet is just to download Nmap from its creator's website.

You can download Nmap for all of the major operating systems from `https://nmap.org/download.html`.
In all cases, you'll also find instructions for installation.

You'll use Nmap the same way on all operating systems, with only one exception. On Linux and macOS machines, you'll preface certain Nmap commands with `sudo`, while on Windows machines, you won't. (Although, on Windows 10, you might have to open the command.exe terminal as an administrator.) Since I just happen to be working on my trusty openSUSE workstation, I'll show you how it works on Linux. Let's start by doing a SYN packet scan:

```
donnie@linux-0ro8:~> sudo nmap -sS 192.168.0.37

Starting Nmap 6.47 ( http://nmap.org ) at 2017-12-24 19:32 EST
Nmap scan report for 192.168.0.37
Host is up (0.00016s latency).
Not shown: 996 closed ports
PORT STATE SERVICE
22/tcp open ssh
515/tcp open printer
631/tcp open ipp
5900/tcp open vnc
MAC Address: 00:0A:95:8B:E0:C0 (Apple)

Nmap done: 1 IP address (1 host up) scanned in 57.41 seconds
donnie@linux-0ro8:~>
```

Here's the breakdown:

- `-sS`: The lowercase `s` denotes the type of scan that we want to perform. The uppercase `S` denotes that we're doing a SYN packet scan. (More on that in a moment.)
- `192.168.0.37`: In this case, I'm only scanning a single machine. However, I could also scan either a group of machines or an entire network.
- `Not shown: 996 closed ports`: The fact that it's showing all of these closed ports instead of `filtered` ports tells me that there's no firewall on this machine. (Again, more on that in a moment.)

  Next, we can see a list of ports that are open. (More on that in a moment.)

  The MAC address of this machine indicates that it's an Apple product of some sort. In a moment, I'll show you how to get more details about what kind of Apple product that it might be.

Now, let's look at this more in detail.

Port states

An Nmap scan will show the target machine's ports in one of three states:

- `filtered`: This means that the port is blocked by a firewall.
- `open`: This means that the port is not blocked by a firewall and that the service that's associated with that port is running.
- `closed`: This means that the port is not blocked by a firewall, and that the service that's associated with that port is not running.

So, in our scan of the Apple machine, we can see that the Secure Shell service is ready to accept connections on port `22`, that the print service is ready to accept connections on ports `515` and `631`, and that the **Virtual Network Computing** (**VNC**) service is ready to accept connections on port `5900`. All of these ports would be of interest to a security-minded administrator. If Secure Shell is running, it would be interesting to know if it's configured securely. The fact that the print service is running means that this up to use the **Internet Printing Protocol** (**IPP**). It would be interesting to know why we're using IPP instead of just regular network printing, and it would also be interesting to know if there are any security concerns with this version of IPP. And of course, we already know that VNC isn't a secure protocol, so we would want to know why it's even running at all. We also saw that no ports are listed as `filtered`, so we would also want to know why there's no firewall on this machine.

One little secret that I'll finally reveal is that this machine is the same one that I used for the OpenVAS scan demos. So, we already have some of the needed information. The OpenVAS scan told us that Secure Shell on this machine uses weak encryption algorithms and that there's a security vulnerability with the print service. In just a bit, I'll show you how to get some of that information with Nmap.

Scan types

There are lots of different scanning options, each with its own purpose. The SYN packet scan that we're using here is considered a stealthy type of scan because it generates less network traffic and fewer system log entries than certain other types of scans. With this type of scan, Nmap sends a SYN packet to a port on the target machine, as if it were trying to create a TCP connection to that machine. If the target machine responds with a SYN/ACK packet, it means that the port is in an `open` state and is ready to create the TCP connection. If the target machine responds with an RST packet, it means that the port is in a `closed` state. If there's no response at all, it means that the port is `filtered`, blocked by a firewall. As a normal Linux administrator, this is one of the types of scans that you would do most of the time.

The `-sS` scan shows you the state of TCP ports, but it doesn't show you the state of UDP ports. To see the UDP ports, use the `-sU` option:

```
donnie@linux-0ro8:~> sudo nmap -sU 192.168.0.37

Starting Nmap 6.47 ( http://nmap.org ) at 2017-12-28 12:41 EST
Nmap scan report for 192.168.0.37
Host is up (0.00018s latency).
Not shown: 996 closed ports
PORT     STATE         SERVICE
123/udp  open          ntp
631/udp  open|filtered ipp
3283/udp open|filtered netassistant
5353/udp open          zeroconf
MAC Address: 00:0A:95:8B:E0:C0 (Apple)

Nmap done: 1 IP address (1 host up) scanned in 119.91 seconds
donnie@linux-0ro8:~>
```

Here, you can see something a bit different: two ports are listed as `open|filtered`. That's because, due to the way that UDP ports respond to Nmap scans, Nmap can't always tell whether a UDP port is `open` or `filtered`. In this case, we know that these two ports are probably open because we've already seen that their corresponding TCP ports are open.

ACK packet scans can also be useful, but not to see the state of the target machine's network services. Rather, it's a good option for when you need to see if there might be a firewall blocking the way between you and the target machine. An ACK scan command looks like this:

```
sudo nmap -sA 192.168.0.37
```

You're not limited to scanning just a single machine at a time. You can scan either a group of machines or an entire subnet at once:

```
sudo nmap -sS 192.168.0.1-128
sudo nmap -sS 192.168.0.0/24
```

The first command scans only the first 128 hosts on this network segment. The second command scans all 254 hosts on a subnet that's using a 24-bit netmask.

A discovery scan is useful for when you need to just see what devices are on the network:

```
sudo nmap -sn 192.168.0.0/24
```

With the `-sn` option, Nmap will detect whether you're scanning the local subnet or a remote subnet. If the subnet is local, Nmap will send out an **Address Resolution Protocol** (**ARP**) broadcast that requests the IPv4 addresses of every device on the subnet. That's a reliable way of discovering devices because ARP isn't something that will ever be blocked by a device's firewall. (I mean, without ARP, the network would cease to function.) However, ARP broadcasts can't go across a router, which means that you can't use ARP to discover hosts on a remote subnet. So, if Nmap detects that you're doing a discovery scan on a remote subnet, it will send out ping packets instead of ARP broadcasts. Using ping packets for discovery isn't as reliable as using ARP because some network devices can be configured to ignore ping packets. Anyway, here's an example from my own home network:

```
donnie@linux-0ro8:~> sudo nmap -sn 192.168.0.0/24

Starting Nmap 6.47 ( http://nmap.org ) at 2017-12-25 14:48 EST
Nmap scan report for 192.168.0.1
Host is up (0.00043s latency).
MAC Address: 00:18:01:02:3A:57 (Actiontec Electronics)
Nmap scan report for 192.168.0.3
Host is up (0.0044s latency).
MAC Address: 44:E4:D9:34:34:80 (Cisco Systems)
Nmap scan report for 192.168.0.5
Host is up (0.00026s latency).
MAC Address: 1C:1B:0D:0A:2A:76 (Unknown)
. . .
. . .
```

We can see three hosts in this snippet, and there are three lines of output for each host. The first line shows the IP address, the second shows whether the host is up, and the third shows the MAC address of the host's network adapter. The first three pairs of characters in each MAC address denote the manufacturer of that network adapter. (For the record, that unknown network adapter is on a recent model Gigabyte motherboard. I have no idea why it's not in the Nmap database.)

The final scan that we'll look at does four things for us:

- It identifies `open`, `closed`, and `filtered` TCP ports.
- It identifies the versions of the running services.
- It runs a set of vulnerability scanning scripts that come with Nmap.
- It attempts to identify the operating system of the target host.

The scan command that does all of these things looks like this:

```
sudo nmap -A 192.168.0.37
```

I guess that you could think of the `-A` option as the all option since it really does do it all. (Well, almost all, since it doesn't scan UDP ports.) First, here's the command that I ran to do the scan:

```
donnie@linux-0ro8:~> sudo nmap -A 192.168.0.37
```

Here are the results, broken down into sections for formatting purposes:

```
Starting Nmap 6.47 ( http://nmap.org ) at 2017-12-24 19:33 EST
Nmap scan report for 192.168.0.37
Host is up (0.00016s latency).
Not shown: 996 closed ports
```

Right away, we can see that there's no active firewall on this machine because no ports are in the `filtered` state. By default, Nmap scans only the most 1,000 most popular ports. Since 996 ports are in the `closed` state, we obviously only have four active network services that would listen on any of these 1,000 ports:

```
PORT STATE SERVICE VERSION
22/tcp open ssh OpenSSH 5.1 (protocol 1.99)
|_ssh-hostkey: ERROR: Script execution failed (use -d to debug)
|_sshv1: Server supports SSHv1
515/tcp open printer?
```

Port `22` is open for Secure Shell access, which we would normally expect. However, look at the SSH version. Version 5.1 is a really old version of OpenSSH. (At the time of writing, the current version is version 8.1.) What's worse is that this OpenSSH server supports version 1 of the Secure Shell protocol. Version 1 is seriously flawed and is easily exploitable, so you never want to see this on your network.

Next, we have amplifying information on the print service vulnerability that we found with the OpenVAS scan:

```
631/tcp open ipp CUPS 1.1
| http-methods: Potentially risky methods: PUT
|_See http://nmap.org/nsedoc/scripts/http-methods.html
| http-robots.txt: 1 disallowed entry
|_/
|_http-title: Common UNIX Printing System
```

In the `631/tcp` line, we can see that the associated service is `ipp`. This protocol is based on the same HTTP that we use to look at web pages. The two methods that HTTP uses to send data from a client to a server are `POST` and `PUT`. What we really want is for every HTTP server to use the `POST` method because the `PUT` method makes it very easy for someone to compromise a server by manipulating a URL. So, if you scan a server and find that it allows using the `PUT` method for any kind of HTTP communications, you have a potential problem. In this case, the solution would be to update the operating system and hope that the updates fix the problem. If this were a web server, you'd want to have a chat with the web server administrators to let them know what you found.

Next, we can see that the VNC service is running on this machine:

```
5900/tcp open vnc Apple remote desktop vnc
| vnc-info:
| Protocol version: 3.889
| Security types:
|_ Mac OS X security type (30)
1 service unrecognized despite returning data. If you know the
service/version, please submit the following fingerprint at
http://www.insecure.org/cgi-bin/servicefp-submit.cgi :
SF-Port515-TCP:V=6.47%I=7%D=12/24%Time=5A40479E%P=x86_64-suse-linux-gnu%r(
SF:GetRequest,1,"\x01");
MAC Address: 00:0A:95:8B:E0:C0 (Apple)
Device type: general purpose
```

VNC can be handy at times. It's like Microsoft's Remote Desktop service for Windows, except that it's free, open source software. But it's also a security problem because it's an unencrypted protocol. So, all your information goes across the network in plain text. If you must use VNC, tunnel it run it through an SSH tunnel.

Next, let's see what Nmap found out about the operating system of our target machine:

```
Running: Apple Mac OS X 10.4.X
OS CPE: cpe:/o:apple:mac_os_x:10.4.10
OS details: Apple Mac OS X 10.4.10 - 10.4.11 (Tiger) (Darwin 8.10.0 -
8.11.1)
Network Distance: 1 hop
Service Info: OS: Mac OS X; CPE: cpe:/o:apple:mac_os_x
```

Wait, what? macOS X 10.4? Isn't that really, really ancient? Well, yeah, it is. The secret that I've been guarding for the past couple of chapters is that the target machine for my OpenVAS and Nmap scan demos has been my ancient, collectible Apple eMac from the year 2003. I figured that scanning it would give us some interesting results to look at, and it would appear that I was right. (And yes, that is eMac, not iMac.)

The final thing we can see is the `TRACEROUTE` information. It's not very interesting, though, because the target machine was sitting right next to me, with only one Cisco switch between us:

```
TRACEROUTE
HOP RTT ADDRESS
1 0.16 ms 192.168.0.37

OS and Service detection performed. Please report any incorrect results at
http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 213.92 seconds
donnie@linux-0ro8:~>
```

Let's say that the target machine has had its SSH service changed to some alternate port, instead of having it run on the default port, `22`. If you scan the machine with a normal `-sS` or `-sT` scan, Nmap won't correctly identify the SSH service on that alternate port. However, a `-A` scan will correctly identify the SSH service, regardless of which port it's using.

Okay; let's take a look at the lab.

Hands-on lab – scanning with Nmap

In this lab, you'll see the results of scanning a machine with various services either enabled or disabled. You'll start with a virtual machine that has its firewall disabled. Let's get started:

1. Briefly peruse the Nmap help screen by using the following command:

```
nmap
```

2. From either your host machine or from another virtual machine, perform these scans against a virtual machine that has its firewall disabled (substitute your own IP address for the one I'm using here):

```
sudo nmap -sS 192.168.0.252
sudo nmap -sT 192.168.0.252
sudo nmap -SU 192.168.0.252
sudo nmap -A 192.168.0.252
sudo nmap -sA 192.168.0.252
```

3. Stop the SSH service on the target machine on Ubuntu:

```
sudo systemctl stop ssh
```

   On CentOS, use this command:

```
sudo systemctl stop sshd
```

4. Repeat s*tep 2*.

You've reached the end of this lab – congratulations!

Now that you've seen how to scan a system, let's look at the GRUB2 bootloader.

Password protecting the GRUB 2 bootloader

People sometimes forget passwords, even if they're administrators. And sometimes, people buy used computers but forget to ask the seller what the password is. (Yes, I've done that.) That's okay, though, because all of the major operating systems have ways to let you either reset or recover a lost administrator password. That's handy, except that it does kind of make the whole idea of having login passwords a rather moot point when someone has physical access to the machine. Let's say that your laptop has just been stolen. If you haven't encrypted the hard drive, it would only take a few minutes for the thief to reset the password and steal your data. If you have encrypted the drive, the level of protection would depend on which operating system you're running. With standard Windows folder encryption, the thief would be able to access the encrypted folders just by resetting the password. With LUKS whole-disk encryption on a Linux machine, the thief wouldn't be able to get past the point of having to enter the encryption passphrase.