PART ONE

Is Linux Really Secure?

CHAPTER 1

Security Threats to Linux

ONE OF THE MOST SIGNIFICANT ATTACKS in the more than 50-year history of the Internet happened in the late 1980s. The overall numbers may not have been impressive, but when you look at it from a percentage perspective, it may have been the most devastating attack ever. In November 1988, Robert T. Morris released a worm from a system located at the Massachusetts Institute of Technology (MIT), although he was at Cornell. The worm is estimated to have attacked 10 percent of the systems that were then connected to the Internet. The impact of the attack continued over several days while various networks that were attached to the NSFNet backbone were disconnected to get systems restored and patched. (The NSFNet was created by the National Science Foundation in the 1980s to pull together all the disparate specialized and regional networks. Initially, the links to the NSFNet were 56Kbps. The NSFNet took over where the ARPANET left off and became the launch point for what we now call the Internet.)

In addition to increasing the awareness of system vulnerabilities, the worm led to the creation of a Computer Emergency Response Team (CERT) at Carnegie Mellon. There were other actions taken to help coordinate activities in the case of another network-wide attack. Less than 15 years later, these coordination efforts were necessary when the first documented and large-scale distributed denial of service (DDoS) attack took place in February of 2000. A young man from Montreal, who called himself Mafiaboy, launched attacks against a number of prominent websites using the Stacheldraht tool in control of a botnet.

The significance of these two events is that both targeted system weaknesses on Unix-like operating systems. The worm attacked several Unix services that could easily be exploited by remote users. Some of these services were weak and unsecure to begin with, while others were simply a result of exploitable bugs in the software. The DDoS attacks were a result, in part, of the way the networking stacks within these operating systems were written. They were vulnerable to particular types of attacks that rendered the systems incapable of taking in more network requests. While some of this was because of the way the network protocols were designed, some was also a result of the implementation of these protocols within the operating system itself.

According to W3 Techs Web technology surveys, servers based on the Unix operating system make up more than three quarters of web servers on the Internet. While not all attacks today originate as technical exploits in external services, there are enough of them that it's worth understanding Linux to know how to limit potential vulnerabilities. According to the Open Web Application Security Project (OWASP), one of the top 10 most common sources of vulnerabilities is misconfiguration. The reality is that no operating system can be called secure. Security isn't a state. Security results from appropriate controls and processes and can't be measured at a point in time. Additionally, the operating system is just the foundation, and it's the implementation that ends up being vulnerable in many cases. Understanding the appropriate technical means that need to be implemented is part of the process, but the state of a system constantly changes. This is due in no small part to influences from the outside. Among these is the so-called "research" constantly performed both by those who hope to improve the resilience of the system against attack and by those who wish to weaken that resilience.

You may have noticed that the percentage of web servers referred to Unix and this is a book about Linux, so why mention Unix? As it turns out, the answer to that question requires us to go back in time more than five decades.

Chapter 1 Topics

This chapter covers the following topics and concepts:

- What the origins of Linux are
- How security works in the open-source world
- What distributions of Linux exist
- What the C-I-A triad is
- How Linux operates as a security device
- What Linux's place in the enterprise is
- What some examples of recent security issues are

Chapter 1 Goals

When you complete this chapter, you will be able to:

- Describe the basics of security in an open-source world
- Explain various roles of Linux systems in the IT architecture
- Differentiate between Linux and the operating environment that runs on top of Linux
- Explain threats that can target Linux

The Origins of Linux

The **Linux** operating system is part of a very long and complicated family tree that began in the 1960s. In 1964, MIT joined with General Electric and Bell Labs to create a multiuser, time-sharing operating system. At the time, computers cost hundreds of thousands if not millions of dollars and were generally used only by a single user or process at a time. The goal of this project was to create an operating system that would allow multiple processes to run, seemingly simultaneously. The operating system was named Multics, short for Multiplexed Information and Computing Service, and its design reflected the concern about protecting one user from another user.

Two members of the Multics team from Bell Labs, Ken Thompson and Dennis Ritchie, became concerned that the system was becoming overly complex because of the design goals. In 1969, Bell Labs pulled out of the five-year-old project. When that happened, Thompson and Ritchie decided to develop their own operating system with goals almost entirely opposite to those of Multics. Thompson and Ritchie called their operating system Unics as a play on the name Multics. The "Uni" in Unics stood for *uniplexed*, suggesting that the goal was to create a small, workable operating system that didn't have multiuser functionality as one of its aims. Where Thompson and Ritchie felt Multics was overdesigned, they worked to create a system that was easier to use but still employed the hierarchical file system and the shell that were created for Multics. In general, though, where Multics favored large, complex user commands, Unics was developed with a lot of very small, single-purpose commands that could be chained together to create more complex functionality.

Fast forward 20 years or so and the name Unics had been changed to **Unix**, AT&T had been broken apart, and several versions of Unix were available from AT&T, University of California at Berkeley, Microsoft, and others. By the mid-1980s, one of the advantages of Unix was that the source code was readily available and the design was simple enough that it made a good source of study for computer science students.

In 1987, a computer science professor and textbook author named Andrew Tanenbaum released a Unix-like operating system called MINIX in an appendix to an operating system textbook. The operating system was also available on a set of floppy disks. Where Unix was primarily a large system operating system, MINIX was targeted at IBM PC-compatible systems. Not surprisingly, a large number of students began using the source code and talking about it on USENET. One of those students was Linus Torvalds, who began adding features and modifying what was in MINIX. Torvalds went on to release his version of the operating system, which he called Linux. Linux was first released on October 5, 1991.

FYI

Linux is only the operating system, also called the **kernel**. This is what interfaces with the hardware to manage memory and file systems and make sure programs are run. Sun Microsystems used to make a distinction between its operating system, which it called SunOS, and the operating environment, which was Solaris. The operating environment is all of the programs and the user interface that communicates with the user and the kernel.

FYI

Because the GNU project developed the common Unix utilities that users would employ if they were using a command-line shell, not to mention the compiler that is commonly used to build the Linux kernel, many GNU proponents prefer that the entire package be referred to as GNU/Linux. This has been the source of a number of long and heated debates over the years. In fact, it is sometimes referred to as a *religious war* because neither side is likely to convert the other side to their way of thinking. While it may be slightly inaccurate, the term *Linux* is generally used to describe the entire operating environment. Otherwise, you may have to start referring to a complete system as KDE/Apache/GNU/Linux or something equally unwieldy just to make sure all the different groups get appropriate billing.

Since its release, a number of open-source projects have contributed to Linux. One of the most impactful over the last 20 years has been **GNU's Not Unix (GNU)**, which began as attempt to create a Unix-like operating system but ended up contributing a lot of tools that are used in the overall environment over the top of the operating system. This is sometimes called userland, since the tools GNU has created are the ones the user was most likely to use on a regular basis. This has changed over time, as more graphical tools and programs are developed, while the GNU tools have remained primarily command line in nature.

Linux commonly describes a fractured collective of different distributions. A **distribution** is a collection of a kernel, userland, graphical interface, and package-management system. The package-management system is used to install software. Package management is developed by the maintainers of the distribution, and many package-management systems are available. **Red Hat** (**Fedora**, Red Hat Enterprise Linux, CentOS) uses Red Hat Package Manager (RPM), though it also supports the Yellowdog Updater, Modified (Yum) that will check dependencies and download and install requested packages. Debian-based systems like Mint and Ubuntu use the Advanced Package Tool (APT) and the related utilities.

Security in an Open-Source World

Linux is part of a large collection of software developed by teams that provide access to the source code and all the programming language text from which the final executable is generated for anyone who wants to look at it. This approach is called **open-source** because the source code is open for anyone to see. Software developed by companies that ask you to pay for the program is commonly called *closed source* in addition to being commercial software.

NOTE
In a university setting, having access to source code enabled you to learn from what others were doing. If someone else had a better idea and improved what was there, then everyone could learn and benefit from it.

Open-source was common in some areas of computing historically, since software wasn't the commodity then that it is today. In the 1950s and 1960s, you may not have paid for software. The system software may have come with the machine you purchased, and the computer companies made their money on hardware. This wasn't always the case, depending on whether the computer vendor manufactured operating systems and hardware.

One thing about programming is that everyone has a different style. Some people are far better at writing efficient code, while others are better at writing code that performs a lot of checks, making the resulting program more resistant to attack. Because of this, having access to source code means a couple of things:

- You can learn from the source code. You can see clearly what it does. You can have a better understanding of how the program operates. You can understand how the programmer thinks, in some cases. Complex software requires a lot of design decisions.
- If you are so inclined, you can make fixes to the source in case there are bugs.

One of the leading proponents of open-source software is Richard Stallman, who created the GNU project while he was at MIT. Stallman believed all source code should be available. This should not be read as a belief that everything should be without cost. There is a concept of *gratis versus libre*, commonly rendered in the open-source community as "free as in free speech, not free as in free beer." *Gratis* means without cost, as in free beer. *Libre* means without restriction, as in free speech. Stallman has long said that he believes that if he finds something that isn't working right with a piece of software, he should be able to go into the source code and fix it. Without access to the source, he doesn't have that freedom, which he believes is essential.

What sorts of security issues are there with open-source? First, just because the source code is open doesn't mean the project has processes in place to ensure code is written to reduce or remove vulnerabilities in the software. It also doesn't mean there has been rigorous testing. If the only testing that has been done is by the developer, then there hasn't been sufficient testing done on the source code. Developers have a different focus when they are testing code they have written than someone who is intent on doing complete security testing or even just regression testing. Larger projects have the advantage of ensuring that people who are competent at testing perform full testing before releases are issued. Smaller projects don't have the luxury of dedicating a lot of people to testing, which can potentially leave vulnerabilities in the software.

FYI

There is a well-known aphorism that suggests that the more bugs you find, the more bugs there are. Proponents of open-source suggest that making sure everyone has access to the code will lead to fewer bugs. More eyeballs means there are more people who can find issues and fix them. Open-source detractors may counter that by saying not all open-source developers are highly skilled and trained. This can lead to more bugs because the best programmers may not always contribute to well-used software projects.

Open-source projects put their source code out on the open Internet at public repositories. Along with the source code, there is generally a cryptographic hash generated to demonstrate that what you downloaded is what you are actually looking for. Where you are protected with this is if the download gets corrupted. It doesn't protect you if the tarball has been altered unless the person doing the altering is really dumb. If the tarball gets modified, an attacker is going to generate a new MD5 and replace the existing one so everything looks correct. If attackers can get access to the repository, they can upload modified source code that may include a back door or some other malicious modification.

This has been an issue in not only open-source projects but commercial projects as well. This has introduced some better control over the source code. It requires the development leads to have better control over code being introduced into the project. Code changes should not ever be introduced without someone having oversight on the code. Modern source code repositories use branches, so even if code is introduced that may be malicious or have vulnerabilities, there is no guarantee the code will make it into the main or production branch.

NOTE

Malicious code modifications have happened with open-source projects in the past. One example was the ProFTP server that was hijacked in 2010. The source code available for download had been replaced with an altered copy that included a back door. Ironically, the person got in through an unpatched vulnerability in the FTP server software that was serving up the source code.

Commercial software may suffer from the problem of lengthy processes that can get in the way of the speedy resolution of problems. A vulnerability must be logged, investigated, and then perhaps brought before a program or project manager for prioritization before the issue can be resolved. Once it's been resolved, the fix likely has to wait for the next build schedule, at which point the entire build must be regression tested and unit tested to make sure all the issues have been resolved. A company like Microsoft batches all of its updates (unless they are considered critical) and releases them all at one time. An advantage to an open-source project is that it may not suffer from this process-heavy path to get a vulnerability fixed. This can be an enormous benefit, but it can sometimes be balanced by less documentation or less testing in a rush to get the fix out with an updated version. Open-source projects, depending on their size, may be less concerned with release schedules and just issue a new minor version when a bug gets resolved. This isn't always the case, of course.

One of the key ideas behind open-source projects like Linux (and all of the packages that go into a regular Linux distribution) is that you are less constrained by human resource issues. Anyone can pick up the source code and also pick up a bug and go fix it. This helps with speed to resolution, but it may not guarantee a high-quality fix. It also doesn't guarantee you will actually get contributors to your project. It does, though, help to get more eyes looking at the problem. Commercial software developers don't always get the best developers either, which isn't always necessary to find and fix vulnerabilities.

NOTE

Nessus is a vulnerability scanner that began life as an open-source project. However, the primary developers discovered they weren't getting a lot of help, so they closed the source and started a business selling Nessus.

One of biggest advantages to open-source projects is the ability for anyone to start a project and have anyone else who is interested work on it. It doesn't require business plans and meetings to determine funding levels and returns on investment or marketing strategies. It just takes someone willing to get started. There are a lot of ways to post source code so someone else can take a look at it and make alterations. Most open-source software is licensed in such a way that any changes are required to also remain open. This is another contribution of the GNU Project and its founder Richard Stallman in particular.

FYI

Stallman developed the GNU **General Public License (GPL)**. Stallman doesn't use the term *copyright* when talking about rights and privileges that are due software authors. Instead, he uses the term *copyleft*. Under copyleft and the GPL, any software that is based on GPLed software is required to retain the same rights as the original software. In other words, if I create a software project that I license using the GPL and you take my software, make some modifications to it, and want to release it yourself, you would also have to release it under the GPL.

Linux Distributions

There are a large number of Linux distributions, and their popularity waxes and wanes over time. Slackware, one of the early Linux distributions, retained tremendous popularity for a number of years. For many years, it dropped in popularity, recently having a bit of a resurgence with some revitalized efforts in refreshing it. At the time of this writing, it is number 20 on DistroWatch.

Red Hat was once a predominant distribution, but it split into Fedora Core as a branch for development work and Red Hat Enterprise Linux (RHEL), which was a commercial offering. CentOS is a distribution of Linux based on released versions of RHEL. In 2020, development of CentOS was stopped by Red Hat, though the development of CentOS Stream continues. The original developer of CentOS, Gregory Kurtzer, has since created the Rocky Linux project. Additionally, there is another Linux distribution called AlmaLinux that is based on RHEL.

Some popular Linux distributions today are **Mint** and **Ubuntu**, both derivatives of Debian. **Debian** has been around for a long time. It was created nearly 30 years ago by Ian Murdock, who wanted to pay homage to his girlfriend at the time, Debra. Debian is a merging of the name Ian with the name Debra. Debian has long been known in the Linux community as a very stable distribution of Linux. Often, it has been well behind what are considered current versions of packages because the maintainers were more interested in an operating system that was solid and didn't crash than they were with keeping up with the bleeding edge.

Most distributions have pre-compiled packages. The distribution determines all the dependencies, meaning you might end up with a lot of extra packages that you don't really want because some package has dependencies built in from another package. One way to avoid this is to build your own version of Linux. Some distributions, such as Linux From Scratch and Gentoo Linux, are source-based distributions. The idea behind these sorts of distributions is that you decide how much or how little you want to put into it. This may have the upside of making it much faster and more responsive. However, the downside to these distributions is that all packages must be compiled from source, and compilation and installation can be a very time-consuming process. Getting one of these distributions up and running may take several hours or even the better part of a day, depending on the speed of your machine and your Internet connection. When you are finished, you will have exactly what you want, but every time you want to update, you will need to go through the compilation process again.

WARNING

Source-based distributions are not for the faint of heart and are probably not best attempted by novice users.

What you may have noticed is there are parent distributions that spawn a lot of other projects that use that initial distribution as a starting point. This was the case with Red Hat and perhaps especially Red Hat Enterprise Linux. It was also true with Debian and then even Ubuntu, which was based on Debian. Another parent Linux distribution, which has spawned some of the current most popular Linux distributions is Arch Linux. Arch is based on the idea of keeping it simple. Rather than building software packages with all the dependencies possible, like some Linux distributions do to be as useable as possible, Arch is based on keeping everything simple, including the dependencies the packages are built with. Two Linux distributions near the top of DistroWatch are Manjaro Linux and EndeavorOS, both of which are based on Arch Linux.

The C-I-A Triad

When it comes to security, there are three fundamental concepts. You may sometimes hear these referred to as C-I-A, the C-I-A triad, the A-I-C triad (to distinguish it from the U.S. intelligence agency), or maybe just the triad. The three concepts are as follows:

- **Confidentiality**—Keeping secrets is the essence of **confidentiality**. If someone says something to you in a crowded room, you won't be assured of much in the way of confidentiality because it would be very easy for someone to overhear what the two of you are saying. If someone were to tap your phone and listen to your conversations, your confidentiality would be violated. This is certainly true when it comes to computer communications. If someone could listen in on network communications by port spanning on a switch, performing a spoofing attack, tapping the physical network cable, or some other method, your confidentiality would be violated. One common way to protect against this is to use encryption. This is not a flawless answer, of course, because not all encryption is created equal. Also, there are issues with keeping the encryption keys secret and protected. In general, however, if you are worried about confidentiality, you will want to find a way to ensure someone can't listen in on or intercept your conversations.
- **Integrity**—Ensuring that the data that are sent are the data that are received is what **integrity** is all about. It's also about protecting against corruption. The MD5 hash mentioned earlier that often accompanies software distributions is used to maintain the integrity of the data that are being transmitted. Note that integrity pertains to more than software downloads or messages that are emailed. It also relates to data at rest on disks. Any magnetic media like a hard drive or tape drive can be altered unexpectedly over a long period of time or from large electromagnetic pulses. Additionally, hardware sometimes fails, which might mean that data that are either written or read get corrupted. Fortunately, you can use cyclical redundancy checks or cryptographic hashes as ways of ensuring that data have not been altered.
- **Availability**—If software is really buggy and crashes a lot, you will have **availability** issues. If you have integrity issues, as described in the preceding bullet, it might lead to availability issues. A failure of integrity may mean the data as it should be is not available to you when you need it. A denial of service (DoS) attack is a specific and malicious form of a denial of service condition. If a service fails for whatever reason, you will have a denial of service condition because normal users will be denied the use of the service. If the Ethernet card on a web server suddenly fails, for instance, you will have a denial of service condition because the users who are trying to get to the web server will be denied service. However, it's not malicious, so it's not an attack. It is definitely a problem of availability, however.

TIP

If an event falls into one of these buckets—confidentiality, integrity, or availability—it is considered to be a security issue.

Some security professionals don't consider the C-I-A triad to be sufficient to discuss all the problems in the world of security. As a result, you will sometimes hear the term *nonrepudiation* discussed separately because it doesn't fit nicely within the triad. Nonrepudiation is usually about preventing a party involved in a transaction from denying that the transaction occurred. One way this manifests is through cryptography. In public key cryptography, everyone has a private and a public key. Sometimes the private key is used to digitally sign a document, like an email message. The private key is supposed to be protected and under the control of the owner of the key. If someone receives a message that was signed with that key, nonrepudiation dictates that the private key owner can't say it didn't come from him or her. If the owner does, he or she is admitting that the key is no longer appropriately controlled.

In the late 1990s, Donn Parker, a security professional, proposed an expansion of the C-I-A triad to cover additional concepts he felt were critical to the realm of information security. This collection of ideas is called the *Parkerian hexad* and includes the following:

- Confidentiality
- Possession or control
- Integrity
- Authenticity
- Availability
- Utility

Although the Parkerian hexad is well respected within the security community, it hasn't overtaken the C-I-A triad as a foundational idea and set of concepts with respect to information security.

Linux as a Security Device

If a Linux distribution makes a distinction between server and desktop/workstation, it has more to do with the number of packages that may be installed by default than it does with any differences in the look and behavior of the operating system once everything is loaded up. Typically, a server operating system comes without a user interface other than the command-line shell. This limits the number of packages that are installed, which makes the installed system less vulnerable to attack. A smaller number of packages means a smaller surface area for attack, which hopefully limits the potential for exploitation of the system. Limiting the number of software packages installed is a fundamental step to system hardening.

When you *harden* an operating system, you make it more resistant to attack. To do this, you remove any software you aren't going to be using. As a result, there are fewer ways in for an adversary. There are also fewer packages you have to monitor for updates in case vulnerabilities are found in the software. Hardening also involves removing all but the most critical users from the system. (Of course, this is always a good idea.) Making sure permissions are restricted on files and directories and removing all but the necessary system services are also common approaches to hardening the system.

After you have a completely hardened operating system, you may choose to deploy it as a bastion host. A **bastion host** is a system that is presented to the outside world that can be used to protect other systems that are more sensitive. Traditionally, a bastion host was a gateway system. For example, you might deploy a bastion host running a Secure Shell (SSH) server. An administrator might connect to the bastion host to gain administrative access to more sensitive network segments that shouldn't be exposed to the outside world, whether that world was the Internet or just the desktop network within an enterprise. The bastion host was used in these situations as a security device. The definition of a bastion host varies, though, depending on whom you talk to. Some consider any Internet-facing device to be a bastion host.

NOTE
The word *bastion* simply means a fortified place. If you have a hardened operating system that has been completely locked down, you might say you have a bastion host.

Linux has long been used as a platform for developing security services, however. Certainly many Internet-facing services have been developed on Unix-like operating systems like the Apache web server, Sendmail, Postfix, various FTP servers, and a number of other software packages designed to provide specific services to users. Linux, though, has also been an area where specific security services have been developed and deployed. One of the first to come to mind is Snort. **Snort** is an intrusion-detection service that watches network traffic as it goes by, looking for areas of concern. Any packet or frame that passes by the Snort sensor that looks suspicious generates an alert.

NOTE
The Snort project has been acquired by Cisco and is a foundation of some of its products now, though Snort remains open-source and available for anyone to use.

Snort is not the only intrusion-detection service that's available under Linux, of course. A number of host-based intrusion-detection systems have been deployed under Linux. In fact, one of the first host-based intrusion-detection systems was developed for Unix-like operating systems. Tripwire, now a commercial product, was developed to monitor critical operating system files. If any of them changed, Tripwire generated an alert. Tripwire and other host-based intrusion-detection systems like AIDE, LIDS, Wazuh, and Samhain all run under Linux and provide similar functionality.

A Linux system may be deployed as a place to collect log files from across the network. The system logger (syslog) facility was developed a long time ago on Unix-like operating systems. There are a number of implementations of it available for Linux, including rsyslog and syslog-ng. When you deploy one of these packages and configure it to allow other systems to send logs to it, you have a centralized syslog server. There is a lot of value to a system like this. If all your logs are in one place, all you need is one program to monitor the logs to look for anomalies that should trigger an alert. There are a number of packages you can use, like Logwatch, that will monitor log files and generate an alert.

Today, a common approach to logging and log management is to use a security information and event manager (SIEM). These are commonly commercial products perhaps because of their complexity. You may notice as you look at a lot of successful open-source software projects, including Nessus and Snort mentioned earlier, eventually either become commercial or take on commercial aspects, in part because software development in large scale is complex and requires a lot of resources. There are even some products that fall just a little short of being a full SIEM, while still having a lot more capability than just being a log aggregator. ElasticStack is an example of this type of software. It is open-source but also has commercial offerings built around the core open-source offerings.

NOTE
Some may recognize ElasticStack by another name—ELK stack. This is because the components of ElasticStack, ElasticSearch, Logstash, and Kibana, have the initials E, L, and K. Together, they become the ELK stack, which has been renamed to ElasticStack.

Of course, Linux also makes a good firewall. A **firewall** is hardware or software capable of blocking networking communications based on established criteria, or *rules*. Where Linux previously used a package called ipchains, the current firewall software available for Linux is called iptables. Iptables is a program that you use to create firewall rules that live within the kernel. The operating system—specifically, the network stack inside the operating system—keeps track of the rules and makes decisions about how to process network packages based on them. You can set up rules to allow, drop, or log specific packets from the network. If you have multiple network interfaces and a network behind your Linux firewall, you can have it perform network address translation (NAT), effectively hiding all the systems behind your Linux firewall.

You can make Linux into a network-based firewall, or you can simply deploy a host-based firewall on your Apache web server. This offers another layer of protection to the services that you offer (should you want to offer web services to people who can connect to your system). On top of the firewall, you can also deploy encryption services using **Transport Layer Security (TLS)**. This is a means of encrypting different services and can be deployed on a Linux-based system. In fact, you can use TLS or even IP Security (IPSec) on your Linux system as a gateway for a virtual private network (VPN). A few packages will support this functionality under Linux.

You can see the wide range of security capabilities that a Linux system can support and its value in the security strategy for an enterprise. A small business may even make use of Linux servers at very little cost to help better secure their network because they can easily acquire a Linux distribution for free. This can be considerably less expensive than trying to purchase a number of appliances that will separately do all the things that a Linux system can do by itself. In fact, Linux will support a number of different systems installed on it using virtualization software. You can end up with a number of virtual machines installed within a single Linux host. A *virtual machine* is an operating system that doesn't run directly on the user's hardware. You may have multiple virtual machines running on a system. This gives you a single system to manage from a physical perspective, but it also enables you to logically separate your different functions onto different systems. An attacker may never know that the different systems being attacked are all resident on a single virtualized system.

Taking virtualization to the next level, we can move up from virtualizing machines to virtualizing applications. Containers are a way of virtualizing applications, by tagging the memory assigned to an application so it's not possible for the application to refer to memory that has not been tagged in the same way. While Docker is popular container management software, there are also tools that are more native to Linux. The package LXC is the set of programs required to manage native Linux containers.

Linux in the Enterprise

As mentioned, Unix-like operating systems in all of their forms, including Linux, comprise roughly two-thirds of all web servers on the Internet. The web server is one of just a couple of network services that are commonly exposed to outside systems from within an enterprise. The web server is the way customers get access to information, products, and services offered by businesses. Additionally, many business-to-business services make use of web services to function, so there is a lot of exposure from this one type of system. **Apache** is a common software package used to provide web services, but it is by no means the only one. In fact, the Apache Foundation was created out of the efforts to develop a web server to manage licensing but also as an umbrella organization to manage a number of other projects. Another very common web server application is Nginx.

When it comes to Linux in the enterprise, it has a lot of additional purposes. One is as an application server. An **application server** is software that provides a framework inside which applications can be written. As an example, you may write a Java application that runs inside the application server and provides services to users over a network. The Apache Foundation is also responsible for developing Tomcat, which is a Java application server used to serve up web services where all the business logic and functions are written in Java on the server side. JBoss is another application server designed for Java-based applications. You can think of this as analogous to the Internet Information Server (IIS) on the Windows side using .NET on the server to write application software. The interface for this software is the web browser on the client's machine. Linux systems may also make good security systems. You may find Linux systems used to perform functions like serving as firewalls or intrusion-detection systems. In fact, Linux may be used as the operating system for a number of security-based appliances.

NOTE

If it isn't Linux, it may well be a Linux relation. Some firewalls have been known to use BSD as their operating system. BSD, of course, also has a Unix heritage and performs in a similar way to Linux, but the design of the kernel, the core of the operating system, is different. If you can operate a Linux system and understand how it works, you can easily pick up a BSD system.

Actually, while we are on the subject of .NET, you can install .NET on a Linux system using the Mono package. This package provides the virtualization for .NET programs, similar to the way Java has a virtual machine to run Java programs. It also provides all of the functional libraries that are what .NET is all about, making application development considerably easier for programmers. In fact, many Microsoft software is making its way to Linux systems, including Visual Studio and PowerShell.

While Linux isn't as common on the desktop as Windows or macOS, there are some users who use it. This often happens with system and network administrators because the tools that come built into most Linux distributions are helpful for those positions. Even if users commonly use Windows or even macOS on their desktops, they may still run virtual machines that have Linux as a guest OS. This provides the functionality of the other desktop while still offering the benefits of a Linux system, without having to run a second system.

You may also use Linux as a proxy server or a gateway in the network. You may use it to perform some filtering between network segments or even use it as a jump host or bastion server. One advantage to Linux is that a lot of packages support functions that are commonly found on Windows systems, so you could use a Linux server to perform as a Remote Desktop Protocol (RDP) system. You can also use it as a file server that makes use of Server Message Block (SMB) or Common Internet File System (CIFS) systems. Of course, you can also use more common Unix-based file sharing like the Network File System (NFS) to connect to either other Linux systems or other Unix-like systems.

Thanks to this versatility, Linux can perform a wide variety of functions within the enterprise. This versatility can also expose it to vulnerabilities, however. The more functions you add to a system, the more complexity you create. The more complexity, the more potential for problems that you can't see. This is where testing is important before deployment. Even if you test extensively, however, there will be behaviors you won't see from everyday users accessing the system.

Recent Security Issues

Security continues to be a hot topic in the news. While there have been serious attacks for years (if not decades), the mainstream news media is just now starting to pick up on what is happening and to report them. One reason for this is the prevalence of notification laws. Most states in the United States have laws that indicate that when a business has been breached by an attacker, the business must notify customers of the breach. This is because customers have the right to know when their personal information is at risk. Businesses commonly store a lot of information about their customers. In many cases, it can be personally identifiable information (PII), meaning that the information could be used to target a customer or use the information to create a new identity based on the customer's information. This new identity, which might be used to acquire credit and purchase goods and services or be sold to someone else, can do significant damage to the customer who is affected.

The media likes to refer to these criminals as *hackers*. The computing community, however, has long used the word *hacker* to describe someone who shows great technical skill. It may also refer to someone who shows some amount of creativity in pulling off a specific task. As an example, students at MIT refer to pranks around the campus as *hacks*. Many of these hacks are legendary, including placing a police car on the top of the Great Dome. These hacks have been going on for decades—long before computers became commonplace. It's because of this that those who were interested in computing at MIT began to refer to neat programming tricks or well-done programs as *hacks*. As a result, those who were engaged in the act of programming were often called *hackers*.

For a long time, to distinguish between the common technical prowess definition of hacker and those who were engaged in malicious or outright illegal actions, the term *cracker* was used. These days, however, neither term—hacker nor cracker—really describes what these people actually are. In most cases, they are simply criminals who happen to be using computers to perform their crimes. For this reason, the terms *attacker* or *adversary* will be used in this book. These terms are specific and not sensationalistic. Someone who is trying to get into a fortified system is an attacker.

Someone who I am squaring off against is an adversary. They are clear words that describe the relationship of that person to the system or network you are attempting to protect.

You may run across the following terms, whether in the technology community or sometimes in the media.

- **Black-hat hacker**—The presence of a black hat often signals the bad guy in a Western movie. These types of characters would commonly wear a black hat so you could clearly identify the bad guy in the film. A **black-hat hacker**, then, is someone who performs attacks against victims for malicious purposes. This term has been used for so long that a series of conferences were named after the term.
- **White-hat hacker**—In contrast, someone who wears a white hat is good. So a **white-hat hacker** may use some of the same techniques as a black-hat hacker, but he or she is a good guy. A white-hat hacker uses the same skills as a black-hat hacker but uses them to improve the overall security of systems or organizations.
- **Gray-hat hacker**—In most cases, there is a range of actions that a white-hat hacker will not undertake. Someone who will undertake those activities may be called a **gray-hat hacker**. This person is somewhere in between a white-hat hacker and a black-hat hacker. When you mix white and black, you get gray.

These are really shorthand terms to describe which side of a particular situation someone may be on. It's especially useful in penetration testing, when someone deliberately tries to get into a network or system. The white-hat hacker is on the inside while the gray-hat hacker is on the outside. A black-hat hacker wouldn't commonly be involved. He or she has no interest in finding holes for the purpose of fixing them, which is the intent of a penetration test.

There was a time when the most common sort of attacker was a young person who was either trying something out or trying to learn something new. It may also have been a result of so-called Internet Relay Chat (IRC) wars, where a common approach was to launch a DoS attack against someone who had said something annoying to you. Those days are gone. Today, the Internet is filled with professional organizations and their highly skilled employees, running businesses that exist to steal information or extort money. They may also perform other illegal actions. These are not exclusively criminal businesses. They may be related to the military for different countries, sometimes called nation-states. Sometimes, attackers are looking for information that could be sold. This could also be intellectual property, which might be used by a government or quasi-government organization. You may see either corporate espionage or inter-country espionage. Of course, another sort of attack that you will commonly see is just a malware attack to get more zombie hosts for a botnet. This is also an attack for financial gain because the zombies may be running web server software as a front to an illegal storefront for, say, pharmaceuticals. The botnet may also be used to rent to someone who just wants to attack someone else for political or financial gain.

CHAPTER SUMMARY

Linux has a long history as a standalone operating system as well as the heritage that comes from being an implementation of the concepts of the Unix operating system. Linux, though, is only the kernel, which is sometimes called the operating system. (Often, people refer to the operating environment as the operating system.) In the case of Linux, the operating environment brings in pieces from a number of other groups and packages, including the GNU Project and the Apache Foundation. Additionally, when you add in the graphical interface, you are talking about a handful of other groups. Because all of those components run on the foundation of the kernel, the entire system is commonly referred to as Linux, although some would consider that a misnomer.

Security is as important with Linux as it is with any other operating system. Linux is not immune to bugs or viruses. While it hasn't been as prone in recent years to attack, there have been both well-known attacks and viruses that have targeted the Linux system. In the end, any operating system that is poorly managed and configured can become vulnerable to attack. For this reason, security is commonly considered by security professionals to be a process and not a state. Security professionals would not consider a system to be either secure or unsecure. It may be vulnerable to exploit, but that's not the same as being unsecure.

Linux does have a lot of versatility, primarily because it has been open-source since its inception. Having access to the source code and the ability to understand what is going on with it opens the door to a lot of development work on packages to extend its functionality. This extension takes us into the areas of security devices as well as the desktop and certainly as a server operating system. Linux remains the most popular choice for a web server or at least for the device that sits in front of a web server either as a load balancer, a reverse proxy, or a web application firewall to protect web applications from attack.

KEY CONCEPTS AND TERMS

Apache
Application server
Availability
Bastion host
Black-hat hacker
Confidentiality
Debian
Distribution
Fedora
Firewall
General Public License (GPL)
GNU's Not Unix (GNU)
Gray-hat hacker
Integrity
Kernel
Linux
Mint
Open-source
Red Hat
Snort
Transport Layer Security (TLS)
Ubuntu
Unix
White-hat hacker

CHAPTER 1 ASSESSMENT

1. Which of the following concepts is part of the C-I-A triad?

A. Authority
B. Access
C. Authenticity
D. Availability

2. Which of the following components makes up the core of the Linux operating system?

A. Cloned software from Unix
B. The kernel
C. Linux libraries
D. Linux services

3. Which of the following is an open-source license?

A. Freeware
B. Public domain
C. GNU GPL
D. Adobe licensing

4. From the following options, select a security advantage of open-source software.

A. The efforts of the open-source community
B. Secrecy in the source code
C. No information released before a solution is available
D. None of the above

5. What percentage of Internet web servers use Unix-based operating systems?

A. 15 percent
B. 25 percent
C. 50 percent
D. 75 percent

6. Which of the following might be used to perform intrusion-detection services?

A. Apache
B. X
C. Snort
D. Mandrake

7. The open-source license associated with the GNU project is ________.

8. Security is a(n) ________.

9. Linux is based on which of the following?

A. Debian
B. Ubuntu
C. Xinu
D. Unix

10. What is the primary purpose of hardening an operating system?

A. Creating a bastion host
B. Building a web server
C. Protecting against attack
D. Building a kernel

11. What is one problem associated with hand-building a kernel?

A. Leaving your system unusable
B. Creating a custom kernel
C. Violating the GPL
D. Not having the right tools

12. What does the acronym C-I-A stand for?

A. Correctness, Intrusion, and Analysis
B. Confidentiality, Intrusion, and Availability
C. Confidentiality, Integrity, and Availability
D. Clarity, Integrity, and Availability