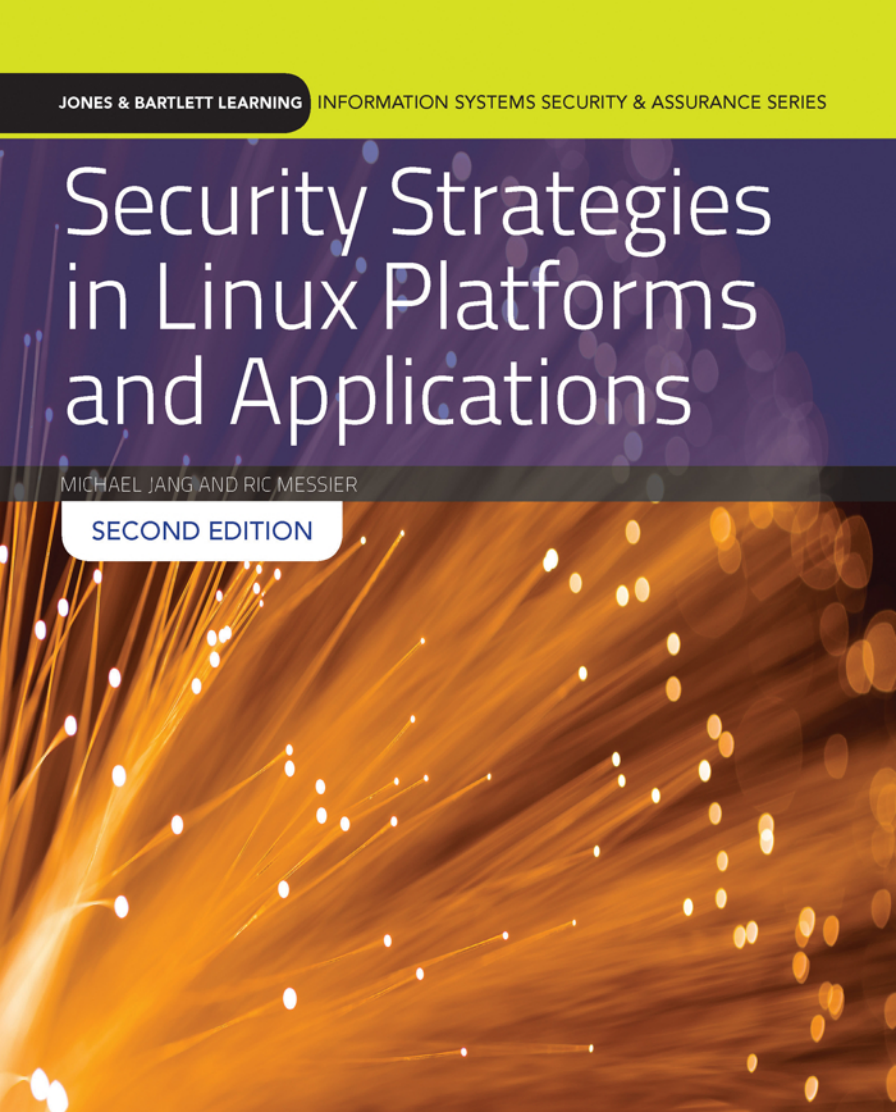
JONES & BARTLETT LEARNING INFORMATION SYSTEMS SECURITY & ASSURANCE SERIES

Security Strategies in Linux Platforms and Applications

MICHAEL JANG AND RIC MESSIER

SECOND EDITION

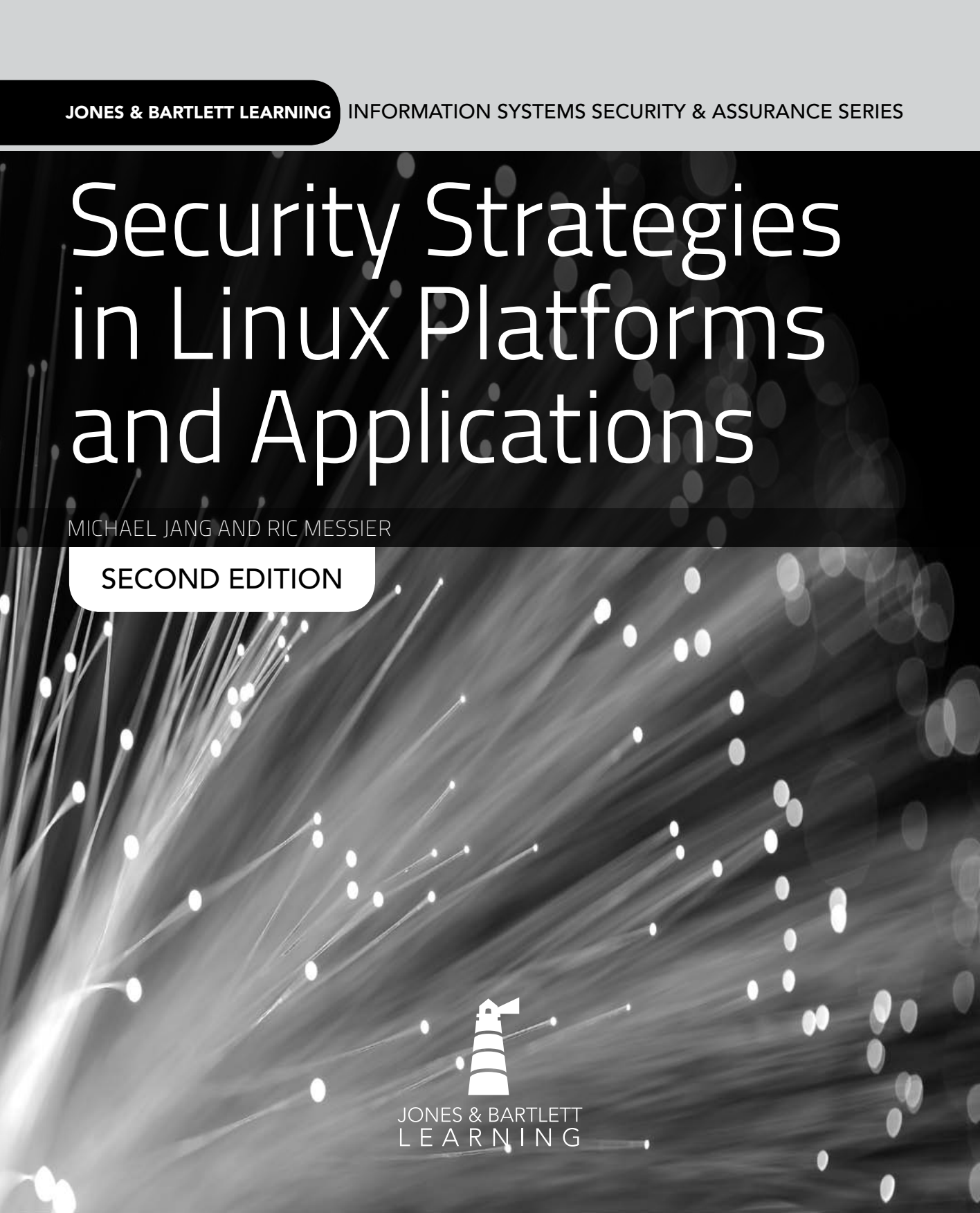


JONES & BARTLETT LEARNING INFORMATION SYSTEMS SECURITY & ASSURANCE SERIES

Security Strategies in Linux Platforms and Applications

MICHAEL JANG AND RIC MESSIER

SECOND EDITION



JONES & BARTLETT
LEARNING

World Headquarters
Jones & Bartlett Learning
5 Wall Street
Burlington, MA 01803
978-443-5000
info@jblearning.com
www.jblearning.com

Jones & Bartlett Learning books and products are available through most bookstores and online booksellers. To contact Jones & Bartlett Learning directly, call 800-832-0034, fax 978-443-8000, or visit our website, www.jblearning.com.

Substantial discounts on bulk quantities of Jones & Bartlett Learning publications are available to corporations, professional associations, and other qualified organizations. For details and specific discount information, contact the special sales department at Jones & Bartlett Learning via the above contact information or send an email to specialsales@jblearning.com.

Production Credits
Chief Executive Officer: Ty Field
President: James Homer
Chief Product Officer: Eduardo Moura
SVP, Curriculum Solutions: Christopher Will
Director of Sales, Curriculum Solutions: Randi Roger
Editorial Management: High Stakes Writing, LLC, Lawrence J. Goodrich, President
Copy Editor, High Stakes Writing: Kate Shoup
Product Manager: Rainna Erikson
Product Management Assistant: Edward Hinman
Production Manager: Tina Chen
Associate Production Editor: Kristen Rogers
Senior Marketing Manager: Andrea DeFronzo
Manufacturing and Inventory Control Supervisor: Amy Bacus
Composition: Gamut+Hue, LLC
Cover Design: Scott Moden
Rights & Media Manager: Joanna Lundeen
Rights & Media Research Coordinator: Mary Flatley
Cover Image: © leungchopan/Shutterstock
Printing and Binding: Edwards Brothers Malloy
Cover Printing: Edwards Brothers Malloy

ISBN: 978-1-284-09065-9

Library of Congress Cataloging-in-Publication Data
Jang, Michael H.
Security strategies in Linux platforms and applications / Michael Jang, Ric Messier. — Second edition.
pages cm
Includes bibliographical references and index.
ISBN 978-1-284-09065-9
1. Linux. 2. Operating systems (Computers) 3. Computer security. I. Messier, Ric. II. Title.
QA76.76.O63J385 2016
005.8—dc23
2015028823

6048

Printed in the United States of America
19 18 17 16 15 10 9 8 7 6 5 4 3 2 1

Contents

To my beautiful wife, Donna,
who has made life worth living again
—Michael Jang

To those who have made me who I am today:
Berkeley Breathed and Hunter S. Thompson
—Ric Messier

Preface

Purpose of This Book

This book is part of the Information Systems Security & Assurance Series from Jones & Bartlett Learning (*www.jblearning.com*). Designed for courses and curriculums in IT Security, Cybersecurity, Information Assurance, and Information Systems Security, this series features a comprehensive, consistent treatment of the most current thinking and trends in this critical subject area. These titles deliver fundamental information-security principles packed with real-world applications and examples. Authored by professionals experienced in information systems security, they deliver comprehensive information on all aspects of information security. Reviewed word for word by leading technical experts in the field, these books are not just current, but forward-thinking—putting you in the position to solve the cybersecurity challenges not just of today, but of tomorrow as well.

Security Strategies in Linux Platforms and Applications, Second Edition, covers every major aspect of security on a Linux system. The first part of this book describes the risks, threats, and vulnerabilities associated with Linux as an operating system. Linux is one of the predominant operating systems used for public-facing servers on the Internet. As a result, a big focus for this book is on implementing strategies that you can use to protect your system implementations, even in cases where they are public facing. To that end, this book uses examples from two of the major distributions built for the server, Red Hat Enterprise Linux and Ubuntu (Server Edition).

With Linux, security is much more than just firewalls and permissions. Part 2 of the book shows you how to take advantage of the layers of security available to Linux—user and group options, filesystems, and security options for important services, as well as the security modules associated with AppArmor and SELinux. It also covers encryption options where available.

The final part of this book explores the use of both open source and proprietary tools when building a layered security strategy for your Linux operating system environments. With these tools, you can define a system baseline, audit the system state, monitor system performance, test network vulnerabilities, detect security breaches, and more. You will also learn basic practices associated with security alerts and updates, which are just as important.

As with any operating system, a Linux implementation requires strategies to harden it against attack. Linux is based on another operating system with a very long history, and it inherits the lessons learned over that history as well as some of the challenges. With Linux, you get a lot of eyes looking at the programs, which many consider to be a benefit of using open source programs and operating systems. While there are advantages,

however, there are risks associated as well. Fortunately, a large community is built around improving Linux and the various software packages that go into it. This includes the National Security Agency (NSA), which initially developed a set of security extensions that has since been implemented into the Linux kernel itself.

When you are finished with this book, you will understand the importance of custom firewalls, restrictions on key services, golden baseline systems, and custom local repositories. You will even understand how to customize and recompile the Linux kernel. You will be able to use open source and commercial tools to test the integrity of various systems on the network. The data you get from such tools will identify weaknesses and help you create more secure systems.

Learning Features

The writing style of this book is practical and conversational. Each chapter begins with a statement of learning objectives. Step-by-step examples of information security concepts and procedures are presented throughout the text. Illustrations are used both to clarify the material and to vary the presentation. The text is sprinkled with notes, tips, FYIs, warnings, and sidebars to alert the reader to additional helpful information related to the subject under discussion. Chapter assessments appear at the end of each chapter, with solutions provided in the back of the book.

Throughout this book are references to commands and directives. They may be included in the body of a paragraph in a monospaced font, like this: `apt-get update`. Other commands or directives may be indented between paragraphs, like the directive shown here:

```
deb http://us.archive.ubuntu.com/ubuntu/ lucid main restricted
```

When a command is indented between paragraphs, it's meant to include a Linux command line prompt. You will note two different prompts in the book. The first prompt is represented with a `$`. As shown here, it represents the command-line prompt from a regular user account:

```
$ ls -ltr > list_of_files
```

The second prompt is represented by a `#`. As shown here, it represents the command-line prompt from a root administrative account:

```
# /usr/sbin/apachectl restart
```

Sometimes, the command or directive is so long, it has to be broken into multiple lines due to the formatting requirements of this book. Line wraps are indicated by a curved arrow, as is shown at the start of what looks like the second line of the `iptables` command. It is just a continuation arrow, which would be typed as a continuous command on the command line or an appropriate configuration file.

```
iptables -A RH-Firewall-1-INPUT -i eth0 -s 10.0.0.0/8
➥-j LOG --log-prefix "Dropped private class A addresses".
```

Chapter summaries are included in the text to provide a rapid review of the material and to help students understand the relative importance of the concepts presented.

Audience

The material is suitable for undergraduate or graduate computer science majors or information science majors, students at a two-year technical college or community college who have a basic technical background, or readers who have a basic understanding of IT security and want to expand their knowledge. It assumes basic knowledge of Linux administration at the command-line interface.

Acknowledgments

I would like to thank Jones & Bartlett Learning and David Kim of Security Evolutions for the opportunity to write this book and be a part of the Information Systems Security & Assurance Series project. This book required a more substantial team effort than ordinary book projects. I would also like to thank the amazing project manager, Kim Lindros; the top-notch technical reviewer, Mike Chapple; the sharp copy editor, Kate Shoup; the marvelous compositor, Mia Saunders; the eagle-eyed proofreader, Ruth Walker; and Larry Goodrich along with Angela Silvia of High Stakes Writing for facilitating the entire process.

In addition, I acknowledge the gracious help of Billy Austin of the SAINT corporation, along with Mike Johnson of AccessData with respect to their products. The author also acknowledges the fortitude of Linux security professionals everywhere, white-hat hackers at heart who have to deal with cultural biases from the mainstream security community along with the legitimate fears of the open source community.

Most importantly, I could not do what I do without the help and support of my wife, Donna. She makes everything possible for me.

Michael Jang

Writing any book is a process. Revising an existing book for a second edition is also a process. It takes a team of people to get from conception to completion. Thanks to Mike, Kate, Mia, Larry, and everyone else who helped get this second edition to the goal line.

Mostly, I'd like to acknowledge all those people who jump into things without any idea what they are getting themselves into. This fearlessness is the best way to jump into something new and guarantee that you are going to learn a lot. Try it some time if you haven't already.

Ric Messier

About the Authors

MICHAEL JANG is a full-time writer, specializing in Linux and related certifications. His experience with computers dates back to the days of badly shuffled punch cards. He has written books such as *RHCE Red Hat Certified Engineer Study Guide*, *LPIC-1 In Depth*, *Ubuntu Server Administration*, and *Linux Annoyances for Geeks*. He is also the author of numerous video courses, and teaches preparation courses on Red Hat certification.

RIC MESSIER has been working with Unix and Unix-like operating systems since the mid-1980s. In the intervening decades, he has done system administration, network engineering, penetration testing, and programming; developed managed security services; and worked in operations security and a number of other jobs in between.

Ric is a security professional who has worked with a number of companies from large Internet service providers to small software companies. He has run a small networking and security consulting practice for the last several years. Additionally, he has taught courses at both the graduate and undergraduate level. Currently, in addition to writing books and recording training videos, he is the program director for Cyber Security and Digital Forensics at Champlain College in Burlington, Vt. He also maintains a blog on information security and digital forensics at *securitykilroy.blogspot.com*.

PART ONE

Is Linux Really Secure?

CHAPTER

1 Security Threats to Linux

ONE OF THE MOST SIGNIFICANT ATTACKS in the more than 40-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 of 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 Web sites 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 two-thirds of the systems on the Internet. So the better we can understand how to provide security to these servers, the less vulnerable to attack they will be. Of course, the reality is that no operating system is secure. Security isn't a state. Security results from appropriate controls and processes, and can't be measured at a point in time. 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.

Because the topic under consideration here is Linux, how does Unix factor into the equation? As it turns out, one must go back several decades to explain it.

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 multi-user, 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 over-designed, 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 there were several versions of Unix 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

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.

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.

Since its release, a number of open source projects have contributed to Linux. One of the most significant over the last 20 years has been **GNU's Not Unix (GNU)**, which was an attempt to create a Unix-like operating system.

Linux itself is a very 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 there are many package-management systems available. RedHat (Fedora, RedHat Enterprise Linux, CentOS) uses RedHat 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.

The idea behind open source goes back many decades to a time when programmers just wrote programs for the fun of it, leaving the source around for someone else to look at and improve. You didn't pay for software. The system software came with the machine you bought and the computer companies made their money on hardware.

The 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.
- 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 securely. 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

FYI

There are a number of well-known aphorisms that suggest 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.

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 put them at risk.

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 an attacker can get access to the repository, he or she can upload modified source code that may include a back door or some other malicious modification.

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.

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 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.

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

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.

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.

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. Now, however, it doesn't even register in the top 25 Linux distributions according to DistroWatch. At the time of this writing, it sits at number 33. Other distributions have fallen over time. **RedHat** was popular for a long time. Its level of support made it a top choice for a lot of users looking for Linux. Now, however, there is no so-called RedHat distribution. It has fragmented into RedHat Enterprise Linux, a piece of commercial software that you have to buy from RedHat, and Fedora, which is the development distribution for RedHat. **Fedora** is more cutting-edge and is where RedHat tries out new concepts to get them stable before rolling them into RedHat Enterprise Linux.

Currently at the top of the popularity charts are **Mint** and **Ubuntu**, both derivatives of Debian. **Debian** has been around for a long time. It was created about 20 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.

NOTE

Gentoo is named after a particular species of small, fast penguin.

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.

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 is sent is the data that is 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 is 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 is either written or read gets corrupted. Fortunately, you can use cyclical