

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](mailto:specialsales@jblearning.com).

**Copyright © 2017 by Jones & Bartlett Learning, LLC, an Ascend Learning Company**

All rights reserved. No part of the material protected by this copyright may be reproduced or utilized in any form, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the copyright owner.

The content, statements, views, and opinions herein are the sole expression of the respective authors and not that of Jones & Bartlett Learning, LLC. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not constitute or imply its endorsement or recommendation by Jones & Bartlett Learning, LLC and such reference shall not be used for advertising or product endorsement purposes. All trademarks displayed are the trademarks of the parties noted herein. *Security Strategies in Linux Platforms and Applications, Second Edition* is an independent publication and has not been authorized, sponsored, or otherwise approved by the owners of the trademarks or service marks referenced in this product.

Microsoft, Internet Explorer, Windows, Microsoft Office, Microsoft Security Development Lifecycle, and Microsoft Baseline Security Analyzer are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. (ISC)<sup>2</sup>, CISSP, ISSAP, ISSMP, ISSEP, CSSLP, CCFP, CAP, SSCP, and CBK are registered and service marks of (ISC)<sup>2</sup>, Inc.

There may be images in this book that feature models; these models do not necessarily endorse, represent, or participate in the activities represented in the images. Any screenshots in this product are for educational and instructive purposes only. Any individuals and scenarios featured in the case studies throughout this product may be real or fictitious, but are used for instructional purposes only.

This publication is designed to provide accurate and authoritative information in regard to the Subject Matter covered. It is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional service. If legal advice or other expert assistance is required, the service of a competent professional person should be sought.

**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

Preface	xix
Acknowledgments	xxiii

## **PART ONE** Is Linux Really Secure? 1

### **CHAPTER 1** Security Threats to Linux 2

The Origins of Linux	4
Security in an Open Source World	5
Linux Distributions	8
The C-I-A Triad	9
Linux as a Security Device	11
Linux in the Enterprise	13
Recent Security Issues	14
<b>CHAPTER SUMMARY</b>	<b>16</b>
<b>KEY CONCEPTS AND TERMS</b>	<b>16</b>
<b>CHAPTER 1 ASSESSMENT</b>	<b>17</b>

### **CHAPTER 2** Basic Components of Linux Security 18

Linux Security Relates to the Kernel	19
The Basic Linux Kernel Philosophy	20
Basic Linux Kernels	20
Distribution-Specific Linux Kernels	21
Custom Linux Kernels	21
Linux Kernel Security Options	24
Securing a System During the Boot Process	24
Physical Security	24
The Threat of the Live CD	24
Boot Process Security	25
More Boot Process Issues	25
Virtual Physical Security	26
Linux Security Issues Beyond the Basic Operating System	26
Service Process Security	26
Security Issues with the GUI	27

Linux User Authentication Databases	28
Protecting Files with Ownership, Permissions, and Access Controls	30
Firewalls and Mandatory Access Controls in a Layered Defense	31
Firewall Support Options	31
Mandatory Access Control Support	33
Protecting Networks Using Encrypted Communication	34
Tracking the Latest Linux Security Updates	35
Linux Security Updates for Regular Users	35
Linux Security Updates for Home Hobbyists	35
Linux Security Updates for Power Users	36
Security Updates for Linux Administrators	36
Linux Security Update Administration	37
The Effect of Virtualization on Security	37
Variations Between Distributions	38
A Basic Comparison: Red Hat and Ubuntu	38
More Diversity in Services	39
<b>CHAPTER SUMMARY</b>	<b>42</b>
<b>KEY CONCEPTS AND TERMS</b>	<b>43</b>
<b>CHAPTER 2 ASSESSMENT</b>	<b>43</b>

## **PART TWO** Layered Security and Linux 45

### **CHAPTER 3** Starting Off: Getting Up and Running 46

Picking a Distribution	47
Picking a Delivery Platform	51
Physical System	52
Virtual Machines	53
Cloud Services	55
Choosing a Boot Loader	58
Linux Loader	58
Grand Unified Boot Loader	59
Services	61
Runlevels	65
Wrappers	68
inetd and xinetd	68
R-services	70
<b>CHAPTER SUMMARY</b>	<b>71</b>
<b>KEY CONCEPTS AND TERMS</b>	<b>72</b>
<b>CHAPTER 3 ASSESSMENT</b>	<b>72</b>

<b>CHAPTER 4</b>	<b>User Privileges and Permissions</b>	<b>74</b>
	The Shadow Password Suite	75
	/etc/passwd	76
	/etc/group	76
	/etc/shadow	77
	/etc/gshadow	79
	Defaults for the Shadow Password Suite	79
	Shadow Password Suite Commands	81
	Available User Privileges	81
	Securing Groups of Users	84
	User Private Group Scheme	84
	Create a Special Group	84
	Configuring the Hierarchy of Administrative Privileges	85
	Administrative Privileges in Services	86
	The su and sg Commands	86
	Options with sudo and /etc/sudoers	87
	Regular and Special Permissions	90
	The Set User ID Bit	90
	The Set Group ID Bit	91
	The Sticky Bit	92
	Tracking Access Through Logs	92
	Authorization Log Options	92
	Authorization Log Files	93
	Pluggable Authentication Modules	94
	The Structure of a PAM Configuration File	94
	PAM Configuration for Users	96
	Authorizing Access with the Polkit	96
	How the Polkit Works	97
	Polkit Concepts	97
	The Polkit and Local Authority	97
	Network User Verification Tools	98
	NIS If You Must	99
	LDAP Shares Authentication	100
	Best Practices: User Privileges and Permissions	100
	<b>CHAPTER SUMMARY</b>	<b>102</b>
	<b>KEY CONCEPTS AND TERMS</b>	<b>102</b>
	<b>CHAPTER 4 ASSESSMENT</b>	<b>102</b>

<b>CHAPTER 5</b>	<b>Filesystems, Volumes, and Encryption</b>	<b>104</b>
Filesystem Organization	105	
Filesystem Basics	105	
The Filesystem Hierarchy Standard	106	
Good Volume Organization Can Help Secure a System	108	
Read-Only Mount Points	111	
How Options for Journals, Formats, and File Sizes Affect Security	112	
Partition Types	113	
The Right Format Choice	113	
Available Format Tools	114	
Using Encryption	114	
Encryption Tools	114	
Encrypted Files	115	
Encrypted Directories	118	
Encrypted Partitions and Volumes	119	
Local File and Folder Permissions	120	
Basic File Ownership Concepts	121	
Basic File-Permission Concepts	121	
Changing File Permissions	122	
Networked File and Folder Permissions	124	
NFS Issues	124	
Samba/CIFS Network Permissions	125	
Network Permissions for the vsftpd Daemon	127	
Configuring and Implementing Quotas on a Filesystem	128	
The Quota Configuration Process	129	
Quota Management	130	
Quota Reports	131	
How to Configure and Implement Access Control Lists on a Filesystem	132	
Configure a Filesystem for ACLs	132	
ACL Commands	133	
Configure Files and Directories with ACLs	133	
Best Practices: Filesystems, Volumes, and Encryption	134	
<b>CHAPTER SUMMARY</b>	<b>135</b>	
<b>KEY CONCEPTS AND TERMS</b>	<b>136</b>	
<b>CHAPTER 5 ASSESSMENT</b>	<b>136</b>	

<b>CHAPTER 6</b>	<b>Securing Services</b>	<b>138</b>
	Starting a Hardened System	140
	Service Management	145
	SysV Init	146
	Upstart	151
	Systemd	152
	Hardening Services	154
	Using Mandatory Access Controls	157
	Security Enhanced Linux	157
	AppArmor	159
	Servers Versus Desktops	160
	Protecting Against Development Tools	161
	<b>CHAPTER SUMMARY</b>	<b>163</b>
	<b>KEY CONCEPTS AND TERMS</b>	<b>164</b>
	<b>CHAPTER 6 ASSESSMENT</b>	<b>164</b>

<b>CHAPTER 7</b>	<b>Networks, Firewalls, and More</b>	<b>166</b>
	Services on Every TCP/IP Port	167
	Protocols and Numbers in /etc/services	168
	Protection by the Protocol and Number	168
	Obscurity and the Open Port Problem	169
	Obscure Ports	169
	Opening Obscure Open Ports	169
	Obscurity by Other Means	170
	Protect with TCP Wrapper	171
	What Services Are TCP Wrapped?	171
	Configure TCP Wrapper Protection	171
	Packet-Filtering Firewalls	173
	Basic Firewall Commands	174
	Firewalld	183
	A Firewall for the Demilitarized Zone	185
	A Firewall for the Internal Network	187
	Alternate Attack Vectors	187
	Attacks Through Nonstandard Connections	188
	Attacks on Scheduling Services	189
	Wireless-Network Issues	191
	Linux and Wireless Hardware	191
	Encrypting Wireless Networks	191
	Bluetooth Connections	192



Security Enhanced Linux	193
The Power of SELinux	194
Basic SELinux Configuration	194
Configuration from the Command Line	194
The SELinux Administration Tool	196
The SELinux Troubleshooter	197
SELinux Boolean Settings	197
Setting Up AppArmor Profiles	202
Basic AppArmor Configuration	202
AppArmor Configuration Files	202
AppArmor Profiles	203
AppArmor Access Modes	204
Sample AppArmor Profiles	204
AppArmor Configuration and Management Commands	204
An AppArmor Configuration Tool	206
Best Practices: Networks, Firewalls, and TCP/IP Communications	206
<b>CHAPTER SUMMARY</b>	<b>208</b>
<b>KEY CONCEPTS AND TERMS</b>	<b>208</b>
<b>CHAPTER 7 ASSESSMENT</b>	<b>209</b>

## **CHAPTER 8** **Networked Filesystems and Remote Access** **210**

Basic Principles for Systems with Shared Networking Services	211
Configure an NTP Server	212
Install and Configure a Kerberos Server	212
Basic Kerberos Configuration	213
Additional Kerberos Configuration Options	215
Securing NFS as If It Were Local	216
Configure NFS Kerberos Tickets	216
Configure NFS Shares for Kerberos	216
Keeping vsftpd Very Secure	217
Configuration Options for vsftpd	217
Additional vsftpd Configuration Files	219
Linux as a More Secure Windows Server	219
Samba Global Options	220
Samba as a Primary Domain Controller	224
Making Sure SSH Stays Secure	225
The Secure Shell Server	225
The Secure Shell Client	228
Create a Secure Shell Passphrase	228

Basic Principles of Encryption on Networks	230
Host-to-Host IPSec on Red Hat	231
Host-to-Host IPSec on Ubuntu	231
Network-to-Network IPSec on Red Hat	233
Network-to-Network IPSec on Ubuntu	233
Helping Users Who Must Use Telnet	233
Persuade Users to Convert to SSH	234
Install More Secure Telnet Servers and Clients	235
Securing Modem Connections	235
The Basics of RADIUS	236
RADIUS Configuration Files	236
Moving Away from Cleartext Access	236
The Simple rsync Solution	238
E-mail Clients	238
Best Practices: Networked Filesystems and Remote Access	239
<b>CHAPTER SUMMARY</b>	<b>241</b>
<b>KEY CONCEPTS AND TERMS</b>	<b>241</b>
<b>CHAPTER 8 ASSESSMENT</b>	<b>242</b>

## **CHAPTER 9** Networked Application Security 243

Options for Secure Web Sites with Apache	244
The LAMP Stack	245
Apache Modules	247
Security-Related Apache Directives	248
Configure Protection on a Web Site	251
Configure a Secure Web site	252
Configure a Certificate Authority	252
mod_security	254
Working with Squid	255
Basic Squid Configuration	256
Security-Related Squid Directives	257
Limit Remote Access with Squid	258
Protecting DNS Services with BIND	258
The Basics of DNS on the Internet	258
DNS Network Configuration	259
Secure BIND Configuration	259
A BIND Database	261
DNS Targets to Protect	261
Domain Name System Security Extensions	261

Mail Transfer Agents	263
Open Source sendmail	263
The Postfix Alternative	266
Dovecot for POP and IMAP	267
More E-mail Services	268
Using Asterisk	268
Basic Asterisk Configuration	269
Security Risks with Asterisk	269
Limiting Printers	270
Printer Administrators	271
Shared Printers	271
Remote Administration	271
The CUPS Administrative Tool	272
Protecting Time Services	273
Obscuring Local and Network Services	273
Best Practices: Networked Application Security	274
<b>CHAPTER SUMMARY</b>	<b>275</b>
<b>KEY CONCEPTS AND TERMS</b>	<b>276</b>
<b>CHAPTER 9 ASSESSMENT</b>	<b>276</b>

## **CHAPTER 10** Kernel Security Risk Mitigation 278

Distribution-Specific Functional Kernels	279
Kernels by Architecture	280
Kernels for Different Functions	281
The Stock Kernel	282
Kernel Numbering Systems	283
Production Releases and More	283
Download the Stock Kernel	284
Stock Kernel Patches and Upgrades	284
Managing Security and Kernel Updates	285
Stock Kernel Security Issues	285
Distribution-Specific Kernel Security Issues	286
Installing an Updated Kernel	286
Development Software for Custom Kernels	287
Red Hat Kernel Development Software	287
Ubuntu Kernel Development Software	288
Kernel-Development Tools	288
Before Customizing a Kernel	289
Start the Kernel Customization Process	289
Kernel Configuration Options	291

Building Your Own Secure Kernel	299
Download Kernel Source Code	300
Download Ubuntu Kernel Source Code	300
Download Red Hat Kernel Source Code	300
Install Required Development Tools	301
Navigate to the Directory with the Source Code	301
Compile a Kernel on Ubuntu Systems	302
Compile a Kernel on Red Hat Systems	302
Compile a Stock Kernel	302
Install the New Kernel and More	303
Check the Boot Loader	303
Test the Result	303
Increasing Security Using Kernels and the /proc/ Filesystem	304
Don't Reply to Broadcasts	304
Protect from Bad ICMP Messages	305
Protect from SYN Floods	305
Activate Reverse Path Filtering	305
Close Access to Routing Tables	306
Avoid Source Routing	306
Don't Pass Traffic Between Networks	307
Log Spoofed, Source-Routed, and Redirected Packets	307
Best Practices: Kernel Security Risk Mitigation	307
<b>CHAPTER SUMMARY</b>	<b>309</b>
<b>KEY CONCEPTS AND TERMS</b>	<b>309</b>
<b>CHAPTER 10 ASSESSMENT</b>	<b>309</b>

## **PART THREE** Building a Layered Linux Security Strategy 311

### **CHAPTER 11** Managing Security Alerts and Updates 312

Keeping Up with Distribution Security	313
Red Hat Alerts	314
Red Hat Enterprise Linux	314
CentOS Linux	314
Fedora Core Linux	315
Ubuntu Alerts	315
Keeping Up with Application Security	316
The OpenOffice.org Suite	317
Web Browsers	317
Adobe Applications	318
Service Applications	318

Antivirus Options for Linux Systems	320
The Clam AntiVirus System	321
AVG Antivirus	322
The Kaspersky Antivirus Alternative	322
SpamAssassin	322
Detecting Other Malware	323
Using Bug Reports	323
Ubuntu’s Launchpad	324
Red Hat’s Bugzilla	325
Application-Specific Bug Reports	325
Security in an Open Source World	327
The Institute for Security and Open Methodologies	328
The National Security Agency	328
The Free Software Foundation	328
User Procedures	329
Deciding Between Automated Updates or Analyzed Alerts	329
Do You Trust Your Distribution?	330
Do You Trust Application Developers?	330
Do You Trust Service Developers?	330
Linux Patch Management	331
Standard yum Updates	332
Updates on Fedora	332
Updates on Red Hat Enterprise Linux	333
Standard apt - * Updates	333
Options for Update Managers	335
Configuring Automated Updates	335
Automatic Red Hat Updates	337
Pushing or Pulling Updates	338
Local or Remote Repositories	338
Configuring a Local Repository	338
Commercial Update Managers	339
The Red Hat Network	340
Canonical Landscape	341
Novell’s ZENworks	341
Open Source Update Managers	342
Various apt - * Commands	342
Various yum commands	343
Red Hat Spacewalk	345
Best Practices: Security Operations Management	345
<b>CHAPTER SUMMARY</b>	<b>346</b>
<b>KEY CONCEPTS AND TERMS</b>	<b>347</b>
<b>CHAPTER 11 ASSESSMENT</b>	<b>347</b>

**CHAPTER 12 Building and Maintaining a Security Baseline 349**

Configuring a Simple Baseline	350
A Minimal Red Hat Baseline	351
A Minimal Ubuntu Baseline	353
Read-Only or Live Bootable Operating Systems	354
Appropriate Read-Only Filesystems	355
Live CDs and DVDs	356
Keeping the Baseline Up to Date	356
A Gold Baseline	357
Baseline Backups	359
Monitoring Local Logs	359
The System and Kernel Log Services	359
Logs from Individual Services	363
Consolidating and Securing Remote Logs	365
Default rsyslog Configuration	365
The Standard rsyslog Configuration File	365
Identifying a Baseline System State	368
Collect a List of Packages	368
Compare Files, Permissions, and Ownership	369
Define the Baseline Network Configuration	370
Collect Runtime Information	370
Checking for Changes with Integrity Scanners	371
Tripwire	371
Advanced Intrusion Detection Environment	372
Best Practices: Building and Maintaining a Secure Baseline	373
<b>CHAPTER SUMMARY</b>	<b>374</b>
<b>KEY CONCEPTS AND TERMS</b>	<b>374</b>
<b>CHAPTER 12 ASSESSMENT</b>	<b>374</b>

**CHAPTER 13 Testing and Reporting 376**

Testing Every Component of a Layered Defense	377
Testing a Firewall	377
Testing Various Services	378
Testing Passwords	381
Testing Mandatory Access Control Systems	382
Checking for Open Network Ports	382
The telnet Command	382
The netstat Command	383
The lsof Command	386
The nmap Command	387

Running Integrity Checks of Installed Files and Executables	392
Verifying a Package	393
Performing a Tripwire Check	394
Testing with the Advanced Intrusion Detection Environment	395
Ensuring that Security Does Not Prevent Legitimate Access	398
Reasonable Password Policies	398
Allowing Access from Legitimate Systems	401
Monitoring Virtualized Hardware	401
Virtual Machine Hardware	402
Virtual Machine Options	402
Monitoring the Kernel-Based Virtual Machine (KVM)	403
Standard Open Source Security-Testing Tools	404
Snort	405
Netcat and the nc Command	407
Vulnerability Scanners for Linux	408
Nessus	408
OpenVAS	410
Nexpose	410
Where to Install Security-Testing Tools	412
Hint: Not Where Attackers Can Use Them Against You	412
Some Tools Are Already Available on Live CDs	413
Best Practices: Testing and Reporting	415
<b>CHAPTER SUMMARY</b>	<b>416</b>
<b>KEY CONCEPTS AND TERMS</b>	<b>416</b>
<b>CHAPTER 13 ASSESSMENT</b>	<b>417</b>

## **CHAPTER 14** Detecting and Responding to Security Breaches **418**

Performing Regular Performance Audits	419
The Basic Tools: ps and top	420
The System Status Package	421
For Additional Analysis	421
Making Sure Users Stay Within Secure Limits	422
Appropriate Policies	423
Education	423
User Installation of Problematic Services	424
Logging Access into the Network	424
Identifying Users Who Have Logged In	424
System Authentication Logs	425
Monitoring Account Behavior for Security Issues	426
Downloaded Packages and Source Code	426
Executable Files	426

Creating an Incident Response Plan	427
Increased Vigilance	428
Should You Leave the System On?	428
Acquiring the Memory Contents	429
Having Live Linux CDs Ready for Forensics Purposes	433
Helix Live Response	433
SANS Investigative Forensics Toolkit	435
Digital Evidence and Forensics Toolkit	435
Build Your Own Media	435
Forensic Live Media	436
When You Put Your Plan into Action	437
Confirming the Breach	437
Identifying Compromised Systems	438
Having Replacement Systems in Place	438
Secure Backup and Recovery Tools	439
Disk Images for Later Investigation	439
The <code>rsync</code> Command	440
Mount Encrypted Filesystems	440
The Right Way to Save Compromised Data as Evidence	441
Basic Principles for Evidence	441
Remembering the Volatile Data	442
Preserving the Hard Disks	442
Disaster Recovery from a Security Breach	442
Determining What Happened	443
Prevention	443
Replacement	443
How and When to Share with the Open Source Community	444
If the Security Issue Is Known...	444
If the Security Issue Has Not Been Reported...	444
Best Practices: Security Breach Detection and Response	445
<b>CHAPTER SUMMARY</b>	<b>446</b>
<b>KEY CONCEPTS AND TERMS</b>	<b>446</b>
<b>CHAPTER 14 ASSESSMENT</b>	<b>447</b>

## **CHAPTER 15** Best Practices and Emerging Technologies **448**

Maintaining a Gold Baseline	449
Monitoring Security Reports	450
Working Through Updates	450
Recalibrating System Integrity	450
Ensuring Availability with Redundancy	451
A Gold Physical Baseline	451
A Gold Virtual Baseline Host	451



Identifying Your Support Options	453
Red Hat Support Options	454
Canonical Support Options	455
Open Source Community Support	455
Checking Compliance with Security Policies	456
User Security	456
Administrator Security	456
Keeping the Linux Operating System Up to Date	457
Baseline Updates	457
Functional Bugs	458
New Releases	458
Keeping Distribution-Related Applications Up to Date	459
Server Applications	459
Desktop Applications	461
Managing Third-Party Applications	461
Licensing Issues	461
Support Issues	462
Sharing Problems and Solutions with the Community	462
Which Community?	462
Sharing with Developers	463
Sharing on Mailing Lists	464
Testing New Components Before Putting Them into Production	464
Testing Updates	465
Documenting Results	465
Beta Testing	466
Keeping Up with Security on Your Systems	466
A New Firewall Command	466
More Mandatory Access Controls	466
Penetration-Testing Tools	467
Single Sign-On	468
Incident Response	468
<b>CHAPTER SUMMARY</b>	<b>469</b>
<b>KEY CONCEPTS AND TERMS</b>	<b>470</b>
<b>CHAPTER 15 ASSESSMENT</b>	<b>470</b>
<b>APPENDIX A Answer Key</b>	<b>471</b>
<b>APPENDIX B Standard Acronyms</b>	<b>473</b>
<b>Glossary of Key Terms</b>	<b>477</b>
<b>References</b>	<b>491</b>
<b>Index</b>	<b>497</b>

*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](http://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](http://securitykilroy.blogspot.com).

## PART ONE

# Is Linux Really Secure?

© Rodolfo CiviDreamstime.com

CHAPTER 1

**Security Threats to Linux** 2

CHAPTER 2

**Basic Components of Linux Security** 18

# 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.

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.

One of the 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

**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.

**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.



## 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.

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

**NOTE**

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

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