

THE EXPERT'S VOICE® IN SECURITY

Foreword by
Vint Cerf,
a Founding Father
of the Internet

Foundations of Security

What Every Programmer Needs to Know

*What every programmer needs to know about security,
illustrated with running examples of web applications
and stories of what's gone wrong in the past.*

Neil Daswani, Christoph Kern,
and Anita Kesavan

Foreword by Vinton G. Cerf

Apress®

Foundations of Security

What Every Programmer Needs
to Know



Neil Daswani, Christoph Kern,
and Anita Kesavan

Foundations of Security: What Every Programmer Needs to Know

Copyright © 2007 by Neil Daswani, Christoph Kern, and Anita Kesavan

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-59059-784-2

ISBN-10 (pbk): 1-59059-784-2

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Lead Editor: Jonathan Gennick

Technical Reviewer: Dan Pilone

Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Matt Wade

Project Manager: Kylie Johnston

Copy Edit Manager: Nicole Flores

Copy Editor: Damon Larson

Assistant Production Director: Kari Brooks-Copony

Production Editor: Ellie Fountain

Compositor: Dina Quan

Proofreader: Liz Welch

Indexer: Julie Grady

Artist: Kinetic Publishing Services, LLC

Cover Designer: Kurt Krames

Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit <http://www.springeronline.com>.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit <http://www.apress.com>.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com> in the Source Code/Download section.

*This book is dedicated to Dad, who provided me my foundations,
and Mom, who taught me what I needed to know.*

—N. Daswani

Contents at a Glance

Foreword	xv
About the Authors	xvii
About the Technical Reviewer	xix
Acknowledgments	xxi
Preface	xxiii

PART 1 ■ ■ ■ Security Design Principles

■ CHAPTER 1	Security Goals	3
■ CHAPTER 2	Secure Systems Design	25
■ CHAPTER 3	Secure Design Principles	61
■ CHAPTER 4	Exercises for Part 1	77

PART 2 ■ ■ ■ Secure Programming Techniques

■ CHAPTER 5	Worms and Other Malware	83
■ CHAPTER 6	Buffer Overflows	93
■ CHAPTER 7	Client-State Manipulation	107
■ CHAPTER 8	SQL Injection	123
■ CHAPTER 9	Password Security	139
■ CHAPTER 10	Cross-Domain Security in Web Applications	155
■ CHAPTER 11	Exercises for Part 2	197

PART 3 ■ ■ ■ Introduction to Cryptography

■ CHAPTER 12	Symmetric Key Cryptography	203
■ CHAPTER 13	Asymmetric Key Cryptography	221
■ CHAPTER 14	Key Management and Exchange	227
■ CHAPTER 15	MACs and Signatures	239
■ CHAPTER 16	Exercises for Part 3	251

PART 4 ■ ■ ■ Appendixes

■ APPENDIX A	Defense-in-Depth: The FLI Model	255
■ APPENDIX B	Source Code Listings	261
■ REFERENCES		267
■ INDEX		277

Contents

Foreword	xv
About the Authors	xvii
About the Technical Reviewer	xix
Acknowledgments	xxi
Preface	xxiii

PART 1 ■ ■ ■ Security Design Principles

■ CHAPTER 1	Security Goals	3
	1.1. Security Is Holistic	3
	1.1.1. Physical Security	4
	1.1.2. Technological Security	4
	1.1.3. Policies and Procedures	6
	1.2. Authentication	7
	1.2.1. Something You Know	7
	1.2.2. Something You Have	8
	1.2.3. Something You Are	10
	1.2.4. Final Notes on Authentication	11
	1.3. Authorization	12
	1.3.1. Access Control Lists (ACLs)	13
	1.3.2. Access Control Models	14
	1.3.3. The Bell-LaPadula Model	15
	1.4. Confidentiality	17
	1.5. Message/Data Integrity	18
	1.6. Accountability	19
	1.7. Availability	20
	1.8. Non-repudiation	21
	1.9. Concepts at Work	22

CHAPTER 2	Secure Systems Design	25
2.1.	Understanding Threats	25
2.1.1.	Defacement	26
2.1.2.	Infiltration	26
2.1.3.	Phishing	27
2.1.4.	Pharming	28
2.1.5.	Insider Threats	28
2.1.6.	Click Fraud	29
2.1.7.	Denial-of-Service (DoS)	29
2.1.8.	Data Theft and Data Loss	30
2.2.	Designing-In Security	30
2.2.1.	Windows 98	31
2.2.2.	The Internet	31
2.2.3.	Turtle Shell Architectures	34
2.3.	Convenience and Security	35
2.4.	SimpleWebServer Code Example	35
2.4.1.	Hypertext Transfer Protocol (HTTP)	35
2.4.2.	Code Walkthrough	36
2.5.	Security in Software Requirements	44
2.5.1.	Specifying Error Handling Requirements	44
2.5.2.	Sharing Requirements with Quality Assurance (QA)	46
2.5.3.	Handling Internal Errors Securely	47
2.5.4.	Including Validation and Fraud Checks	48
2.5.5.	Writing Measurable Security Requirements	50
2.5.6.	Security or Bust	50
2.6.	Security by Obscurity	51
2.6.1.	Flaws in the Approach	51
2.6.2.	SimpleWebServer Obscurity	52
2.6.3.	Things to Avoid	55
2.7.	Open vs. Closed Source	57
2.8.	A Game of Economics	58
2.9.	“Good Enough” Security	59
CHAPTER 3	Secure Design Principles	61
3.1.	The Principle of Least Privilege	61
3.2.	Defense-in-Depth	63
3.2.1.	Prevent, Detect, Contain, and Recover	63
3.2.2.	Don’t Forget Containment and Recovery	64
3.2.3.	Password Security Example	65

3.3. Diversity-in-Defense	65
3.4. Securing the Weakest Link	66
3.4.1. Weak Passwords	66
3.4.2. People	66
3.4.3. Implementation Vulnerabilities	67
3.5. Fail-Safe Stance	67
3.5.1. SimpleWebServer Fail-Safe Example	67
3.5.2. Attempted Fix 1: Checking the File Length	69
3.5.3. Attempted Fix 2: Don't Store the File in Memory	69
3.5.4. Fix: Don't Store the File in Memory, and Impose a Download Limit	70
3.6. Secure by Default	71
3.7. Simplicity	72
3.8. Usability	73
3.9. Security Features Do Not Imply Security	74
 ■ CHAPTER 4 Exercises for Part 1	77

PART 2 ■ ■ ■ Secure Programming Techniques

■ CHAPTER 5 Worms and Other Malware	83
5.1. What Is a Worm?	83
5.2. An Abridged History of Worms	84
5.2.1. The Morris Worm: What It Did	84
5.2.2. The Morris Worm: What We Learned	85
5.2.3. The Creation of CERT	86
5.2.4. The Code Red Worm	86
5.2.5. The Nimda Worm	87
5.2.6. The Blaster and SQL Slammer Worms	87
5.3. More Malware	89
 ■ CHAPTER 6 Buffer Overflows	93
6.1. Anatomy of a Buffer Overflow	93
6.1.1. A Small Example	94
6.1.2. A More Detailed Example	94
6.1.3. The <code>safe_gets()</code> Function	98
6.2. Safe String Libraries	100

6.3. Additional Approaches	101
6.3.1. StackGuard	101
6.3.2. Static Analysis Tools	102
6.4. Performance.	103
6.5. Heap-Based Overflows.	103
6.6. Other Memory Corruption Vulnerabilities	103
6.6.1. Format String Vulnerabilities	104
6.6.2. Integer Overflows.	104
 CHAPTER 7 Client-State Manipulation	 107
7.1. Pizza Delivery Web Site Example	108
7.1.1. Attack Scenario.	110
7.1.2. Solution 1: Authoritative State Stays at Server	112
7.1.3. Solution 2: Signed State Sent to Client.	114
7.2. Using HTTP POST Instead of GET	117
7.3. Cookies	119
7.4. JavaScript.	121
 CHAPTER 8 SQL Injection	 123
8.1. Attack Scenario	124
8.2. Solutions	130
8.2.1. Why Blacklisting Does Not Work	130
8.2.2. Whitelisting-Based Input Validation.	132
8.2.3. Escaping	132
8.2.4. Second Order SQL Injection	133
8.2.5. Prepared Statements and Bind Variables.	134
8.2.6. Mitigating the Impact of SQL Injection Attacks.	136
 CHAPTER 9 Password Security	 139
9.1. A Strawman Proposal	139
9.2. Hashing.	141
9.3. Offline Dictionary Attacks.	143
9.4. Salting	144

9.5. Online Dictionary Attacks	150
9.6. Additional Password Security Techniques	151
9.6.1. Strong Passwords	151
9.6.2. “Honeypot” Passwords	151
9.6.3. Password Filtering	151
9.6.4. Aging Passwords	152
9.6.5. Pronounceable Passwords	152
9.6.6. Limited Login Attempts	152
9.6.7. Artificial Delays	152
9.6.8. Last Login	153
9.6.9. Image Authentication	153
9.6.10. One-Time Passwords	154

■ CHAPTER 10 Cross-Domain Security in Web Applications 155

10.1. Interaction Between Web Pages from Different Domains	156
10.1.1. HTML, JavaScript, and the Same-Origin Policy	156
10.1.2. Possible Interactions of Documents from Different Origins	157
10.1.3. HTTP Request Authentication	159
10.1.4. Lifetime of Cached Cookies and HTTP Authentication Credentials	160
10.2. Attack Patterns	161
10.2.1. Cross-Site Request Forgery (XSRF)	162
10.2.2. Cross-Site Script Inclusion (XSSI)	164
10.2.3. Cross-Site Scripting (XSS)	165
10.3. Preventing XSRF	169
10.3.1. Inspecting Referer Headers	170
10.3.2. Validation via User-Provided Secret	170
10.3.3. Validation via Action Token	171
10.3.4. Security Analysis of the Action Token Scheme	173
10.4. Preventing XSSI	176
10.4.1. Authentication via Action Token	176
10.4.2. Restriction to POST Requests	177
10.4.3. Preventing Resource Access for Cost Reasons	177

10.5. Preventing XSS	178
10.5.1. General Considerations	179
10.5.2. Simple Text	180
10.5.3. Tag Attributes (e.g., Form Field Value Attributes)	181
10.5.4. URL Attributes (href and src)	183
10.5.5. Style Attributes	185
10.5.6. Within Style Tags	186
10.5.7. In JavaScript Context	186
10.5.8. JavaScript-Valued Attributes	189
10.5.9. Redirects, Cookies, and Header Injection	190
10.5.10. Filters for “Safe” Subsets of HTML	191
10.5.11. Unspecified Charsets, Browser-Side Charset Guessing, and UTF-7 XSS Attacks	192
10.5.12. Non-HTML Documents and Internet Explorer Content-Type Sniffing	193
10.5.13. Mitigating the Impact of XSS Attacks	194

■ CHAPTER 11 Exercises for Part 2	197
---	-----

PART 3 ■ ■ ■ Introduction to Cryptography

■ CHAPTER 12 Symmetric Key Cryptography	203
12.1. Introduction to Encryption	204
12.1.1. Substitution Ciphers	204
12.1.2. Notation and Terminology	205
12.1.3. Block Ciphers	205
12.1.4. Security by Obscurity: Recap	208
12.1.5. Encrypting More Data	208
12.1.6. AES Code Example	210
12.2. Stream Ciphers	217
12.2.1. One-Time Pad	217
12.2.2. RC4	217
12.3. Steganography	219
12.3.1. What Is Steganography?	219
12.3.2. Steganography vs. Cryptography	220

CHAPTER 13	Asymmetric Key Cryptography	221
13.1.	Why Asymmetric Key Cryptography?	221
13.2.	RSA	223
13.3.	Elliptic Curve Cryptography (ECC)	223
13.4.	Symmetric vs. Asymmetric Key Cryptography	224
13.5.	Certificate Authorities	224
13.6.	Identity-Based Encryption (IBE)	225
13.7.	Authentication with Encryption	225
CHAPTER 14	Key Management and Exchange	227
14.1.	Types of Keys	227
14.1.1.	Identity Keys	227
14.1.2.	Conversation or Session Keys	227
14.1.3.	Integrity Keys	228
14.2.	Key Generation	228
14.2.1.	Random Number Generation	229
14.2.2.	The rand() function	230
14.2.3.	Random Device Files	230
14.2.4.	Random APIs	231
14.3.	Key (Secret) Storage	231
14.3.1.	Keys in Source Code	231
14.3.2.	Storing the Key in a File on Disk	233
14.3.3.	“Hard to Reach” Places	233
14.3.4.	Storing Secrets in External Devices	233
14.4.	Key Agreement and Exchange	235
14.4.1.	Using Asymmetric Keys	236
14.4.2.	Diffie-Hellman (DH)	236
CHAPTER 15	MACs and Signatures	239
15.1.	Secure Hash Functions	239
15.2.	Message Authentication Codes (MACs)	240
15.2.1.	CBC MACs	240
15.2.2.	HMAC	241
15.3.	Signatures	242
15.3.1.	Certificates and Certificate Authorities (CAs)	243
15.3.2.	Signing and Verifying	246
15.3.3.	Registration Authorities (RAs)	246
15.3.4.	Web of Trust	247

15.4. Attacks Against Hash Functions	247
15.5. SSL	247
15.5.1. Server-Authenticated-Only	248
15.5.2. Mutual Authentication	249

■ CHAPTER 16 Exercises for Part 3	251
---	-----

PART 4 ■ ■ ■ Appendixes

■ APPENDIX A Defense-in-Depth: The FLI Model	255
A.1. Protecting Against Failure	256
A.2. Protecting Against Lies	257
A.3. Protecting Against Infiltration	257
A.4. Other Techniques	258
A.5. Using an FLI-like Model	258
A.6. References	258
■ APPENDIX B Source Code Listings	261
■ REFERENCES	267
■ INDEX	277

Foreword

When Neil Daswani and Christoph Kern invited me to write a foreword to the book you are reading now, I accepted without hesitation and with a good deal of pleasure. This timely volume is solidly grounded in theory and practice and is targeted at helping programmers increase the security of the software they write. Despite the long history of programming, it seems as if bug-free and resilient software continues to elude us. This problem is exacerbated in networked environments because attacks against the vulnerabilities in software can come from any number of other computers and, in the Internet, that might mean millions of potential attackers. Indeed, the computers of the Internet that interact with each other are in some sense performing unplanned and unpredictable tests between the software complements of pairs of machines. Two machines that start out identically configured will soon become divergent as new software is downloaded as a consequence of surfing the World Wide Web or as updates are applied unevenly among the interacting machines. This richly diverse environment exposes unexpected vulnerabilities, some of which may be exploited deliberately by hackers intent on causing trouble or damage and who may even have pecuniary motivations for their behavior. So-called bot armies are available in the millions to be directed against chosen targets, overwhelming the defenses of some systems by the sheer volume of the attack. In other cases, known weaknesses are exploited to gain control of the target machines or to introduce viruses, worms, or Trojan horses that will do further damage.

Programmers writing for networked environments have a particularly heavy responsibility to be fully aware of the way in which these vulnerabilities may come about and have a duty to do everything they can to discover and remove them or to assure that they are eliminated by careful design, implementation, and testing. It takes discipline and a certain amount of paranoia to write secure software. In some ways it is like driving defensively. You must assume you are operating in a hostile environment where no other computer can be trusted without demonstrating appropriate and verifiable credentials. Even this is not enough. In a kind of nightmare scenario, someone with a USB memory stick can bypass all network defenses and inject software directly into the computer. Such memory sticks emulate disks and can easily pick up viruses or worms when they are used on unprotected computers, and when reused elsewhere, can propagate the problem. All input must be viewed with suspicion until cleared of the possibility of malformation.

Vulnerability can exist at all layers of the Internet protocol architecture and within the operating systems. It is naive to imagine that simply encrypting traffic flowing between pairs of computers on the Internet is sufficient to protect against exploitation. An obvious example is a virus attached to an e-mail that is sent through the Internet fully encrypted at the IP layer using IPsec. Once the message is decrypted packet by packet and reassembled, the virus will be fully ready to do its damage unless it is detected at the application layer by the e-mail client, or possibly by the mail transport agent that delivers the e-mail to the target recipient.

It is vital to understand not only how various attacks are carried out, but also how the vulnerabilities that enable these attacks arise. Programs that fail to check that inputs are properly