Praise for *Gray Hat Hacking: The Ethical Hacker’s Handbook, Fourth Edition*

“In the ever-changing world of information security, the *Gray Hat Hacking* books have proven a reliable resource for timely and relevant information. A definite recommendation for anyone looking for a crash course in the field.”

—Pedram Amini
Founder, OpenRCE and the Zero Day Initiative

“The fourth edition of *Gray Hat Hacking* brings even more invaluable knowledge to print. This book is a must read for anyone who wants to keep up with the high-paced world of information security.”

—David D. Rude II

*Lead Security Researcher and Senior Security Analyst, FusionX LLC.*

“This book provides deep insights from true wizards in the field of ethical hacking and penetration testing. The lavish illustrations, clear descriptions, and step-by-step descriptions are invaluable in building an understanding of some seriously powerful hacking techniques.”

—Ed Skoudis

*SANS Institute Fellow*
Dedicated to Shon Harris

I first met Shon in 2001 at a CISSP bootcamp. Honestly, I had just heard of CISSP a couple of months prior and signed up for a bootcamp in San Mateo, California. I was studying at the Naval Postgraduate School and the course was close, so off I went with no clue what I was in for. The CISSP certification is not an easy certification to obtain, to say the least. There is a mountain of information to absorb and recite in a six-hour exam! Lucky for me, Shon was my instructor at the CISSP bootcamp. Very quickly I came to respect Shon for her broad level of knowledge and skill in teaching what could easily be a mundane subject. The week was fun, and she kept it moving with insightful stories and examples. During the lunch sessions, I started to pick her brain about writing a book. I was impressed with the *CISSP All-in-One Exam Guide* and simply wanted to know about the process of writing a book. I must have made some sort of impression on her with my questions because within a few months she sent me an e-mail saying, “Hey, I remember you expressed interest in writing and I was wondering if you would like to help me on a new book project I have....” I was floored. After all, I had no experience in writing at that level before, and it was truly an honor to be asked by Shon to help. By this time, Shon had published several books already and had established quite a reputation in the field. The project we began that day eventually became the first edition of *Gray Hat Hacking: The Ethical Hacker’s Handbook*. The rest, as they say, is history.

From that book, there have been several other projects, including subsequent editions and a separate book called *Security Information and Event Management (SIEM) Implementation*. Each time I worked with Shon, I was amazed at her wide
range of knowledge and the ease with which she explained complex topics. She was truly gifted in that area, and we are all the beneficiary of that gift which she so gladly shared. Our field has become very complex and difficult to start as a career. During my career, many people have asked how I got started and how they should get started. I always direct them to Shon’s books as a starting point. The *CISSP All-in-One Exam Guide* alone has served myself and countless others as an endless resource, full of timeless truths. It has been assuring to have a copy on my shelf and now on my Kindle. I have referred back to it many times, and I am sure I will continue to do so.

In late 2007, I had just retired from the Marine Corps and was looking for what I wanted to do with the rest of my life. Right on cue, Shon contacted me and asked if I wanted to help her with a job she was working on with a large retailer in the Chicago area. I flew out to her hometown in San Antonio, Texas, and we put together a plan, which eventually grew to include five personnel and lasted some six years. At that client, we met many good friends, including Jeff Comstock and Casey Knodel, both of whom I am sure echo my sentiments that it was good working with Shon. This one client helped me establish my business, which eventually grew in size and was sold last year. Therefore, I owe a great deal to Shon, both personally and professionally. Quite simply, I don’t think I would be where I am in this field without Shon having been there and helping along the way. Recently, I was able to share that with Shon, and I am grateful that I had the opportunity before she passed.

I consider it a blessing to have crossed paths with Shon and to have worked with her on several projects. I remain proud to have called her my friend. I will miss her dearly, and I am sure I speak for the other authors and many others who would say: thanks, we love you and will not forget the kindness and generosity you showed us.

Allen Harper
Ethical hacker and friend of Shon Harris
EVP of Tangible Security, Inc.

En memoria a Fernando Regalado Juarez, mi guía, el que gracias a su visión y doble jornada de trabajo me dio una carrera profesional y este libro es el resultado de su esfuerzo. No pude darte este libro en persona, pero sé que te alegraras en el cielo papito.

— Daniel Regalado
To my best friend Mike Lester who is insightful, kind, fun-loving, and fiercely loyal. Thanks for helping me through tough times, Mike!
— Shon Harris

To my brothers and sisters in Christ, keep running the race. Let your light shine for Him, that others may be drawn to Him through you.
— Allen Harper

To all those who have served in and sacrificed for the U.S Armed Forces.
— Chris Eagle

To Jessica, the most amazing and beautiful person I know.
— Jonathan Ness

To my family and friends for their unconditional support and making this life funny and interesting.
— Branko Spasojevic

To my dad, whose guidance, support, and encouragement have always been a push in the right direction.
— Ryan Linn

To my lovely wife, LeAnne, and daughter, Audrey, thank you for your ongoing support!
— Stephen Sims

**ABOUT THE AUTHORS**

**Daniel Regalado**, aka Danux, is a Mexican senior malware and vulnerability researcher at FireEye and a former reverse engineer for Symantec. With more than 12 years in the security field, he has received several certifications such as CISSP, ISO 27001 Lead Author, OSCP, OSCE, and CREA, giving him the penetration testing and reverse engineering skills needed when dissecting Advanced Persistent Threats.
(APTs). He likes to give talks about his research; his most recent talk was presented at BsidesLV 2014. He also enjoys documenting interesting findings on his personal blog at danuxx.blogspot.com.

**Shon Harris** was the CEO and founder of Logical Security, as well as an author, educator, and security consultant. She was a former engineer with the U.S. Air Force Information Warfare unit and published several books on different disciplines within information security. Shon was also recognized as one of the top 25 women in information security by *Information Security Magazine*.

**Allen Harper**, CISSP, PCI QSA, is the executive vice president of Tangible Security, Inc. and lives in North Carolina. He retired from the Marine Corps after 20 years and a tour in Iraq. Prior to Tangible, he owned and operated a company, N2 Net Security, Inc. Additionally, he has served as a security analyst for the U.S. Department of the Treasury, Internal Revenue Service, and Computer Security Incident Response Center (CSIRC). He regularly speaks and teaches at events such as InfraGard, ISSA, Black Hat, and Techno.

**Chris Eagle** is a senior lecturer in the Computer Science Department at the Naval Postgraduate School (NPS) in Monterey, California. A computer engineer/scientist for 29 years, his research interests include computer network attack and defense, computer forensics, and reverse/anti-reverse engineering. He can often be found teaching at Black Hat or spending late nights working on capture the flag at Defcon.

**Jonathan Ness**, CHFI, is a lead software security engineer in Microsoft’s Security Response Center (MSRC). He and his coworkers ensure that Microsoft’s security updates comprehensively address reported vulnerabilities. He also leads the technical response of Microsoft’s incident response process that is engaged to address publicly disclosed vulnerabilities and exploits targeting Microsoft software. He serves one weekend each month as a security engineer in a reserve military unit.

**Branko Spasojevic** is a security engineer at Google. Before that he worked as a reverse engineer for Symantec where he analyzed various threats and APT groups.

**Ryan Linn**, CISSP, CSSLP, MCSE, CCNP-Security, OSCE, has more than 15 years of experience in information security. He has worked as a technical team leader, database administrator, Windows and UNIX systems administrator, network engineer, web application developer, systems programmer, information security engineer, and is currently a managing consultant doing network penetration testing.
Ryan has delivered his research about ATM security, network protocol attacks, and penetration testing tactics at numerous conferences, including Black Hat, Defcon, DerbyCon, Shmoocon, and SecTor to name a few. He is also an open source project contributor for projects such as Metasploit, Ettercap, and the Browser Exploitation Framework.

**Stephen Sims** is a senior instructor and course author with the SANS Institute. He has written multiple courses on penetration testing, exploit development, and reverse engineering, and currently lives in the San Francisco Bay Area working as a consultant. He regularly speaks internationally at conferences and organizations ranging from RSA and OWASP AppSec to the ThaiCERT and Australian Information Security Association (AISA). Previously, Stephen spent many years working as a security architect and engineer at various Fortune 500 companies.

**Disclaimer:** The views expressed in this book are those of the authors and not of the U.S. government, the Microsoft Corporation, or any other Company mentioned herein.

About the Technical Editors

**Rudolph Araujo** is a director of product marketing at FireEye, where he focuses on defining the messaging and go-to-market strategies for the various FireEye security products and services. Rudolph has many years of software development experience on UNIX and Windows. Prior to joining FireEye, he was a director of professional services at McAfee. As part of that role, he led McAfee and Foundstone Professional Services in a number of regions, including, most recently, the Western United States. He was also responsible for leading the software and application security service lines and led some of the largest security assessment projects such as audits of automobile and farm equipment electronic security, telematics security assessments, and security reviews of software systems such as virtualization hypervisors and hardware diagnostics. Rudolph has a master’s degree from Carnegie Mellon University specializing in information security. He is a contributor to a number of industry journals such as *Software Magazine*, where he writes a column on security. His work has also been published in academic journals as well as IEEE’s *Security & Privacy Magazine*. He has been honored for the last 10 years in a row with the Microsoft Security MVP Award in recognition of his thought leadership and contributions to the security communities.
Sergio Humberto Lopez Lopez is the founder and CEO of White Hat Consultores. For the past 12 years, he has focused on leading information security projects and services for Mexican and Latin-American companies, specifically for financial, government, and national security institutions. He is a professional consultant who holds the CISSP, CISM, and OSCP certifications, and has worked with several technology manufacturers such as CISCO, 3Com, and HP. He likes to spend his time pursuing business endeavors, researching hacking techniques, playing chess, and studying mathematics. Sergio holds a bachelor’s degree in physics and mathematics in ESFM, IPN and a master’s degree in electrical engineering from CINVESTAV in Mexico City.

CONTENTS AT A GLANCE

Part I  Crash Course: Preparing for the War
apter 1  Ethical Hacking and the Legal System
apter 2  Programming Survival Skills
apter 3  Static Analysis
apter 4  Advanced Analysis with IDA Pro
apter 5  World of Fuzzing
apter 6  Shellcode Strategies
apter 7  Writing Linux Shellcode
Part II  From Vulnerability to Exploit
apter 8  Spoofing-Based Attacks
apter 9  Exploiting Cisco Routers
apter 10  Basic Linux Exploits
apter 11  Advanced Linux Exploits
apter 12  Windows Exploits
Preface

Acknowledgments

Introduction

Part I  Crash Course: Preparing for the War

Chapter 1  Ethical Hacking and the Legal System

Why You Need to Understand Your Enemy’s Tactics

Recognizing Trouble When It Happens
The Ethical Hacking Process
The Penetration Testing Process
What Would an Unethical Hacker Do Differently?
The Rise of Cyberlaw
Understanding Individual Cyberlaws
The Controversy of “Hacking” Tools
Vulnerability Disclosure
Different Teams and Points of View
How Did We Get Here?
CERT’s Current Process
Organization for Internet Safety
Conflicts Will Still Exist
“No More Free Bugs”
Bug Bounty Programs
Summary
References
For Further Reading

Chapter 2  Programming Survival Skills
C Programming Language
Basic C Language Constructs
Sample Program
Compiling with gcc
Computer Memory
Random Access Memory (RAM)
Endian
Segmentation of Memory

Programs in Memory

Buffers

Strings in Memory

Pointers

Putting the Pieces of Memory Together

Intel Processors

Registers

Assembly Language Basics

Machine vs. Assembly vs. C

AT&T vs. NASM

Addressing Modes

Assembly File Structure

Assembling

Debugging with gdb

gdb Basics

Disassembly with gdb

Python Survival Skills

Getting Python

Hello World in Python

Python Objects

Strings

Numbers

Lists

Dictionaries
Chapter 3  Static Analysis

Ethical Reverse Engineering

Why Bother with Reverse Engineering?

Reverse Engineering Considerations

Source Code Analysis

Source Code Auditing Tools

The Utility of Source Code Auditing Tools

Manual Source Code Auditing

Automated Source Code Analysis

Binary Analysis

Manual Auditing of Binary Code

Automated Binary Analysis Tools

Summary

For Further Reading

Chapter 4  Advanced Analysis with IDA Pro

Static Analysis Challenges

Stripped Binaries

Statically Linked Programs and FLAIR

Data Structure Analysis

Quirks of Compiled C++ Code
Chapter 5  World of Fuzzing

Introduction to Fuzzing
Choosing a Target

Input Types
Ease of Automation

Complexity
Types of Fuzzers
Mutation Fuzzers
Generation Fuzzers

Getting Started
Finding the Fuzzing Templates
Lab 5-1: Collecting Samples from the Internet Archive
Choosing the Optimal Template Set with Code Coverage
Lab 5-2: Selecting the Best Samples for Fuzzing

Peach Fuzzing Framework
Peach Fuzzing Strategies
Speed Does Matter
Crash Analysis
Lab 5-3: Mutation Fuzzing with Peach
Chapter 6  Shellcode Strategies

User Space Shellcode

System Calls

Basic Shellcode

Port Binding Shellcode

Reverse Shellcode

Find Socket Shellcode

Command Execution Code

File Transfer Code

Multistage Shellcode

System Call Proxy Shellcode

Process Injection Shellcode

Other Shellcode Considerations

Shellcode Encoding

Self-Corrupting Shellcode

Disassembling Shellcode

Kernel Space Shellcode

Kernel Space Considerations

Summary

References

For Further Reading
Chapter 7  Writing Linux Shellcode

Basic Linux Shellcode

System Calls

System Calls by C

System Calls by Assembly

Exit System Call

setreuid System Call

Shell-Spawning Shellcode with execve

Implementing Port-Binding Shellcode

Linux Socket Programming

Assembly Program to Establish a Socket

Test the Shellcode

Implementing Reverse Connecting Shellcode

Reverse Connecting C Program

Reverse Connecting Assembly Program

Encoding Shellcode

Simple XOR Encoding

Structure of Encoded Shellcode

JMP/CALL XOR Decoder Example

FNSTENV XOR Example

Putting the Code Together

Automating Shellcode Generation with Metasploit

Generating Shellcode with Metasploit

Encoding Shellcode with Metasploit

Summary
For Further Study

Part II From Vulnerability to Exploit

Chapter 8 Spoofing-Based Attacks

What Is Spoofing?

ARP Spoofing
Lab 8-1: ARP Spoofing with Ettercap

Viewing Network Traffic

Modifying Network Traffic

DNS Spoofing
Lab 8-2: DNS Spoofing with Ettercap

Executing the Attack

NetBIOS Name Spoofing and LLMNR Spoofing
Lab 8-3: Attacking NetBIOS and LLMNR with Responder

Cracking NTLMv1 and NTLMv2 Hashes

Summary

For Further Reading

Chapter 9 Exploiting Cisco Routers

Attacking Community Strings and Passwords
Lab 9-1: Guessing Credentials with Ncrack and Metasploit
Lab 9-2: Guessing Community Strings with Onesixtyone and Metasploit

SNMP and TFTP
Lab 9-3: Downloading Configuration Files with Metasploit
Lab 9-4: Modifying Configurations with SNMP and TFTP

Attacking Cisco Passwords

Attacking Cisco Type 7 Passwords
Lab 9-5: Cracking Type 7 Passwords with Cain
Lab 9-6: Cracking Type 7 Passwords with Metasploit
Attacking Cisco Type 5 Passwords
Lab 9-7: Attacking Cisco Type 5 Passwords with John the Ripper
Middling Traffic with Tunnels
Lab 9-8: Setting Up a GRE Tunnel
Lab 9-9: Routing Traffic over a GRE Tunnel
Exploits and Other Attacks
Cisco Exploits
Maintaining Access on Cisco Devices
Summary
For Further Reading

Chapter 10 Basic Linux Exploits
Stack Operations
Function Calling Procedure
Buffer Overflows
Lab 10-1: Overflow of meet.c
Ramifications of Buffer Overflows
Local Buffer Overflow Exploits
Lab 10-2: Components of the Exploit
Lab 10-3: Exploiting Stack Overflows from the Command Line
Lab 10-4: Exploiting Stack Overflows with Generic Exploit Code
Lab 10-5: Exploiting Small Buffers
Exploit Development Process
Lab 10-6: Building Custom Exploits
Chapter 11  Advanced Linux Exploits

Format String Exploits

The Problem

Lab 11-1: Reading from Arbitrary Memory
Lab 11-2: Writing to Arbitrary Memory
Lab 11-3: Changing Program Execution

Memory Protection Schemes

Compiler Improvements

Lab 11-4: Bypassing Stack Protection

Kernel Patches and Scripts

Lab 11-5: Return to libc Exploits
Lab 11-6: Maintaining Privileges with ret2libc

Bottom Line

Summary

References

For Further Reading

Chapter 12  Windows Exploits

Compiling and Debugging Windows Programs

Lab 12-1: Compiling on Windows
Lab 12-2: Crashing the Program

Writing Windows Exploits

Exploit Development Process Review
Lab 12-3: Exploiting ProSSH Server

Understanding Structured Exception Handling (SEH)

Implementation of SEH

Summary

References

For Further Reading

Chapter 13  Bypassing Windows Memory Protections

Understanding Windows Memory Protections (XP SP3, Vista, 7, 8, Server 2008, and Server 2012)

Stack-Based Buffer Overrun Detection (/GS)

Safe Structured Exception Handling (SafeSEH)

SEH Overwrite Protection (SEHOP)

Heap Protections

Data Execution Prevention (DEP)

Address Space Layout Randomization (ASLR)

Enhanced Mitigation Experience Toolkit (EMET)

Bypassing Windows Memory Protections

Bypassing /GS

Bypassing SafeSEH

Bypassing ASLR

Bypassing DEP

Bypassing EMET

Bypassing SEHOP

Summary

References
For Further Reading

Chapter 14 Exploiting the Windows Access Control Model

Why Access Control Is Interesting to a Hacker

Most People Don’t Understand Access Control

Vulnerabilities You Find Are Easy to Exploit

You’ll Find Tons of Security Vulnerabilities

How Windows Access Control Works

Security Identifier

Access Token

Security Descriptor

The Access Check

Tools for Analyzing Access Control Configurations

Dumping the Process Token

Dumping the Security Descriptor

Special SIDs, Special Access, and “Access Denied”

Special SIDs

Special Access

Investigating “Access Denied”

Analyzing Access Control for Elevation of Privilege

Attack Patterns for Each Interesting Object Type

Attacking Services

Attacking Weak DACLs in the Windows Registry

Attacking Weak Directory DACLs

Attacking Weak File DACLs

What Other Object Types Are Out There?
Enumerating Shared Memory Sections

Enumerating Named Pipes

Enumerating Processes

Enumerating Other Named Kernel Objects (Semaphores, Mutexes, Events, and Devices)

Summary

For Further Reading

Chapter 15  Exploiting Web Applications

Overview of the Top 10 Web Vulnerabilities

MD5 Hash Injection

Lab 15-1: Injecting the Hash

Multibyte Encoding Injection

Understanding the Vulnerability

Lab 15-2: Leverage Multibyte Encoding

Hunting Cross-site Scripting (XSS)

Lab 15-3: Basic XSS Injection into a JavaScript Block

Unicode Normalization Forms Attack

Lab 15-4: Leveraging Unicode Normalization

Unicode Normalization Introduction

Normalization Forms

Preparing the Environment for Testing

XSS Testing via x5s the Plug-In

Launching the Attack Manually

Adding Your Own Test Case

Summary

References
Chapter 16  Exploiting IE: Smashing the Heap

Setting Up the Environment

WinDbg Configuration

Attaching the Browser to WinDbg

Introduction to Heap Spray

Spraying with HTML5

Lab 16-1: Heap Spray via HTML5

DOM Element Property Spray (DEPS)

Lab 16-2: Heap Spray via DEPS Technique

HeapLib2 Technique

Forcing New Allocations by Exhausting the Cache Blocks

Lab 16-3: HeapLib2 Spraying

Flash Spray with Byte Arrays

Lab 16-4: Basic Heap Spray with Flash

Flash Spray with Integer Vectors

Lab 16-5: Heap Spray with Flash Vectors

Leveraging Low Fragmentation Heap (LFH)

Summary

References

For Further Reading

Chapter 17  Exploiting IE: Use-After-Free Technique

Use-After-Free Overview

Debugging JavaScript

Dissecting Use-After-Free (UAF)
Lab 17-1: Dissecting UAF, Step by Step

Leveraging the UAF Vulnerability

Example 17-1: Connecting the Dots

Summary

References

For Further Reading

Chapter 18  Advanced Client-Side Exploitation with BeEF

BeEF Basics

Lab 18-1: Setting Up Beef

Lab 18-2: Using the BeEF Console

Hooking Browsers

Lab 18-3: The Basic XSS Hook

Lab 18-4: Hooking Browsers with Site Spoofing

Lab 18-5: Automatically Injecting Hooks with Shank

Fingerprinting with BeEF

Lab 18-6: Fingerprinting Browsers with BeEF

Lab 18-7: Fingerprinting Users with BeEF

Lab 18-8: Fingerprinting Computers with BeEF

Browser Exploitation

Lab 18-9: Exploiting Browsers with BeEF and Java

Exploiting Browsers with BeEF and Metasploit

Automating Attacks

Summary

For Further Reading

Chapter 19  One-Day Exploitation with Patch Diffing
Introduction to Binary Diffing

Application Diffing

Patch Diffing

Binary Diffing Tools

BinDiff
turbodiff

Lab 19-1: Our First Diff

Patch Management Process

Microsoft Patch Tuesday

Lab 19-2: Obtaining and Extracting Microsoft Patches

Examining the Patch

Lab 19-3: Diffing MS14-006 with turbodiff

Kernel Debugging

Lab 19-4: Kernel Debugging MS14-006

Summary

References

For Further Reading

Part III Advanced Malware Analysis

Chapter 20 Dissecting Android Malware

The Android Platform

Android Application Package

Application Manifest

Analyzing DEX

Java Decompilation

DEX Decompilation
DEX Disassembling

Example 20-1: Running APK in Emulator

Malware Analysis

Malware Analysis Primer

Example 20-2: Black-Box APK Monitoring with Droidbox

Summary

For Further Reading

Chapter 21 Dissecting Ransomware

History of Ransomware

Options for Paying the Ransom

Dissecting Ransomlock

Example 21-1: Dynamic Analysis

Example 21-2: Static Analysis

CryptoLocker

Summary

For Further Reading

Chapter 22 Analyzing 64-bit Malware

Overview of the AMD64 Architecture

AMD64 Calling Conventions

Decrypting C&C Domains

Example 22-1: Decrypting C&C Domains

Summary

For Further Reading

Chapter 23 Next-Generation Reverse Engineering

Notable IDA Plug-ins
This book has been developed by and for security professionals who are dedicated to working in an ethical and responsible manner to improve the overall security posture of individuals, corporations, and nations.

Each of the authors would like to thank the staff at McGraw-Hill Education. In particular, we would like to thank Meghan Manfre and Mary Demery. You really kept
us on track and helped us through the process. Your dedication to this project was truly noteworthy. Thanks.

Daniel Regalado le gustaría agradecer primero a Dios por permitirle conocerlo y cambiar su vida, a mi esposa Diana por aguantar tantas noches sin su esposo en la recamara, por su paciencia y ternura recibida, te amo! A mis hijos Fercho y Andrick por enseñarme que la vida es bella y finalmente pero no menos importante a la Familia Regalado Arias: Fernando, Adelina, Susana Erwin y Belem que desde que empecé este libro soñé con dedicárselos y que aunque disto mucho de ser el mejor profesional ellos me hacen sentir como tal, los amo y que Viva México!

Shon Harris would like to thank the other authors and the team members for their continued dedication to this project and continual contributions to the industry as a whole. Shon would also like to thank the crazy Fairbairn sisters—Kathy Conlon, Diane Marshall, and Kristy Gorenz for their lifelong support of Shon and her efforts.

Allen Harper would like to thank his wonderful wife, Corann, and daughters, Haley and Madison, for their support and understanding. It is wonderful to see our family grow stronger in Christ each year. I love you each dearly. In addition, Allen would like to thank the members of his Church for their love and support. In particular, Ronnie Jones, Bob Poole, and Donovan Smith have been true brothers in the Lord and great friends.

Chris Eagle would like to thank all his students past, present, and future for keeping him motivated to do what he loves.

Jonathan Ness would like to thank Jessica, his amazing wife, for tolerating the long hours required for him to write this book (and hold his job, and his second job, and third “job,” and all the side projects). Thanks also to the experts who freely contributed insights for the book: Didier Stevens for the generous PDF analysis help (and for providing the free PDF analysis tools at http://blog.didierstevens.com/programs/pdf-tools), Terry McCorkle for the web vulnerabilities guidance and advice, and John Lambert for his insight into the Windows security model. Finally, Jonathan would like to thank the mentors, teachers, coworkers, pastors, family, and friends who have guided him along his way, contributing more to his success than they’ll ever know.

Branko Spasojevic would like to thank his mother, sisters, and grandparents for all the support and knowledge they passed on. Another big thanks goes to the
friends and all the people who share their knowledge for free to benefit everyone. Some people who deserve special mention are: Sandra, Lina, Toni, Luka, Bojan, Leon, Ante, Hrvoje, Aleksandar, Franjo, Domagoj, Daniel, Santi, Keith, Piotr, and Andrea.

**Ryan Linn** would like to thank Heather for her support, encouragement, and advice as well as his family and friends for their support and for putting up with the long hours and infrequent communication while the book was coming together. Thanks also go out to Ed Skoudis for pushing me to do awesome things, and to HD, Egypt, Nate, Shawn, and all the other friends and family who have offered code assistance, guidance, and support when I’ve needed it the most.

**Stephen Sims** would like to thank his wife, LeAnne, and daughter, Audrey, for their ongoing support with the time needed to research, write, work, teach, and travel. Stephen would also like to thank his parents, George and Mary, and sister, Lisa, for their support from afar. Finally, a special thanks to all of the brilliant security researchers who contribute so much to the community with publications, lectures, and tools.

We, the authors, would also like to collectively thank Hex-Rays and Nicolas Falliere for the generous use of their tools IDA Pro and JEB.

---

*I have seen enough of one war never to wish to see another.*

—Thomas Jefferson

*I know not with what weapons World War III will be fought, but World War IV will be fought with sticks and stones.*

—Albert Einstein

*The art of war is simple enough. Find out where your enemy is. Get at him as soon as you can. Strike him as hard as you can, and keep moving on.*

—Ulysses S. Grant
The goal of this book is to help produce more highly skilled security professionals who are dedicated to protecting against malicious hacking activity. It has been proven over and over again that it is important to understand one’s enemies, including their tactics, skills, tools, and motivations. Corporations and nations have enemies that are very dedicated and talented. We must work together to understand the enemies’ processes and procedures to ensure we can properly thwart their destructive and malicious behavior.

The authors of this book want to provide you, the reader, with something we believe the industry needs: a holistic review of ethical hacking that is responsible and truly ethical in its intentions and material. This is why we keep releasing new editions of this book with a clear definition of what ethical hacking is and is not—something society is very confused about.

We have updated the material from the third edition and have attempted to deliver the most comprehensive and up-to-date assembly of techniques, procedures, and material with real hands-on labs that can be replicated by the readers. Twelve new chapters are presented and the other chapters have been updated.

In Part I, we prepare the readers for the war with all the necessary tools and techniques to get the best understanding of the more advanced topics. This section covers the following:

- White, black, and gray hat definitions and characteristics
- The slippery ethical issues that should be understood before carrying out any type of ethical hacking activities
- Programming, which is a must-have skill for a gray hat hacker to be able to create exploits or review source code
- Reverse engineering, which is a mandatory skill when dissecting malware or researching vulnerabilities
- Fuzzing, which is a wonderful skill for finding 0-day exploits
- Shellcodes, creating these from scratch will enable you to dissect them when you find them in the wild

In Part II, we explain advanced penetration methods and exploits that no other books cover today, with hands-on labs for testing. Many existing books cover the same old tools and methods that have been rehashed numerous times, but we have
chosen to go deeper into the advanced mechanisms that hackers have used in recent 0-days. We created hands-on labs for the following topics in this section:

NOTE To ensure your system is properly configured to perform the labs, we have provided a README file for each lab as well as any files you will need to perform the labs. These files are available for download from the McGraw-Hill Professional Computing Downloads page: www.mhprofessional.com/getpage.php?c=computing_downloads.php&cat=112. Please see the Appendix for more information.

• Network attacks against Cisco routers
• ARP, DNS, NetBIOS, and LLMNR spoofing
• Advanced Linux and Windows vulnerabilities and how they are exploited
• Windows exploits updated with the monay.py PyCommand plug-in from the Corelan team
• Exploiting web applications, but instead of looking at well-known attacks (SQLi, XSS, and so on), focusing on bypassing techniques like MD5 injection, MySQL type conversion flaws, and Unicode Normalization Form attacks
• The latest working heap spray techniques with proof-of-concept source code available for replication
• Use-After-Free (UAF), which is the attacker’s vulnerability of choice when exploiting browsers in 2014, dissecting every single step of the techniques used for this real 0-day
• The Browser Exploitation Framework (BeEF) and how to hook and exploit browsers in an automated way
• Patch diffing to find nonpublic vulnerabilities by dissecting Microsoft patches

In Part III, we dedicate a full chapter to each of the latest advanced techniques for dissecting malware. We cover the following topics in this section:

• **Android malware** Now that this malicious code has been ported to smartphones, understanding the process for reversing and emulating this malware in a secure environment is mandatory.

• **Ransomware** One of the most sophisticated threats, ransomware can take full control of your Desktop or encrypt your personal information until you pay a ransom. It is imperative that you know how it works and, most importantly, how to deactivate it.

• **64-bit malware** With malware being ported to 64-bit operating systems, you need to know how to reverse these kinds of binaries and the challenges that you’ll have to overcome.

• **Next-generation reverse engineering** The latest and greatest reverse engineering techniques are discussed as an extra bonus for readers.

If you are ready to take the next step to advance and deepen your understanding of ethical hacking, this is the book for you.
Ethical Hacking and the Legal System

This book has not been compiled and written to be used as a tool by individuals who wish to carry out malicious and destructive activities. It is a tool for people who are interested in extending or perfecting their skills to defend against such attacks and damaging acts.

In this chapter, we cover the following topics:

• Why you need to understand your enemy’s tactics
• The ethical hacking process
• The rise of cyberlaw
• Vulnerability disclosure

Why You Need to Understand Your Enemy’s Tactics

Understanding how attacks work is one of the most challenging aspects of defensive security. By familiarizing yourself with how hackers think and operate, you can better tailor your organization’s defenses to emerging threats and trends. If you don’t test defenses against attacks, the only people who will be testing your network will be bad guys. By learning offensive security, you will be able to test your defenses and determine which aspects are operating correctly and where any gaps exist.

The criminal community is changing. Over the last few years, their motivation has evolved from the thrill of figuring out how to exploit vulnerabilities to figuring out how to make revenue from their actions and getting paid for their skills. Attackers who were out to “have fun” without any real target in mind have, to a great extent, been replaced by people who are serious about benefiting financially from their activities. Attacks are getting not only more specific, but also increasingly sophisticated. The following are just a few examples of this trend:
• In October 2013, hackers infiltrated Adobe and stole 38 million account credentials as well as encrypted credit card numbers. Portions of the data were exposed on the Internet.¹

• In July 2013, Harbor Freight was hit with malware that aided in stealing card data from over 400 of its stores. This incident is one of many instances of malware being used to exfiltrate large amounts of credit card data from online retailers.²

• In May 2013, the Ponemon Institute released a report sponsored by Symantec that indicated breaches in the United States cost average companies approximately $188 per record.³ This coupled with reports that breaches resulted in more than 28,000 records being exposed mean that although attackers are making money, it’s costing companies more and more to deal with the compromises.

• At the peak of Christmas shopping in 2013, Target suffered one of the largest breaches to date. Between 40,000 and 70,000 individuals were potentially impacted by the losses. Target jumped ahead of the news reports in order to help people understand the breach as well as how the company was reacting to it. Target continues to maintain a site presence to provide information about new security measures put into place as well as how to deal with credit card fraud.⁴

A conservative estimate from Gartner pegs the average hourly cost of downtime for computer networks at $42,000.⁵ A company that suffers from a worse than average downtime of 175 hours per year can lose more than $7 million per year. Even when attacks are not newsworthy enough to be reported on TV or talked about in security industry circles, they still negatively affect companies’ bottom lines.

In addition to attackers who are trying to profit, some attackers are politically motivated. These attacks are labeled hacktivism. Both legal and illegal methods can be used to portray political ideology. Is it right to try to influence social change through the use of technology? Is web defacement covered under freedom of speech? Is it wrong to carry out a virtual “sit in” on a site that provides illegal content? During the 2009 Iran elections, was it unethical for an individual to set up a site that revealed discontent about the potential corrupt government elections? When Israeli invaded Gaza, many website defacements, DoS attacks, and website hijackings occurred. One’s viewpoint determines what is ethical or not.

Some attackers also create and sell zero-day attacks. A zero-day attack is one for which there is currently no fix available. Whoever is running the particular software
that contains that exploitable vulnerability is exposed, with little or no protection. The code for these types of attacks are advertised on special websites and sold to other attackers or organized crime rings.

**Recognizing Trouble When It Happens**

Network administrators, engineers, and security professionals must be able to recognize when an attack is underway or when one is imminent. It may seem like it should be easy to recognize an attack as it is happening—but only for the very “noisy” or overwhelming attacks such as denial-of-service (DoS) attacks. Many attackers fly under the radar and go unnoticed by security devices and security staff. By knowing *how* different types of attacks work, you can properly recognize and stop them.

You also need to know when an attack may be around the corner. If network staff is educated on attacker techniques and they see a ping sweep followed a day later by a port scan, they know their systems may soon be under attack. Many activities lead up to different types of attacks, so understanding these will help a company protect itself. The argument can be made that we now have more automated security products that identify these types of activities so we don’t have to see them coming. But, depending on the software, those activities may not be put in the necessary context and the software may make a dangerous decision. Computers can outperform any human on calculations and repetitive tasks, but we still have the ability to make necessary judgment calls because we understand the grays in life and do not just see things in 1s and 0s.

Hacking tools are really just software tools that carry out some specific types of procedure to achieve a desired result. The tools can be used for good (defensive) purposes or for bad (offensive) purposes. The good and the bad guys use the same exact toolset; the difference is their intent when operating these tools. It is imperative for security professionals to understand how to use these tools and how attacks are carried out if they are going to be of any use to their customers and to the industry.

**The Ethical Hacking Process**

To protect themselves, organizations may want to understand the impact and ability of an attacker. In this case, they may employ an *ethical hacker*, also known as a *penetration tester*, to simulate an attack against the environment. The techniques
that penetration testers employ are designed to emulate those of real attackers without causing damage; they enable organizations to better protect themselves against attack. But customers and aspiring hackers need to understand how this process works.

By defining penetration testing activities, stages, and steps, you can set expectations between yourself as a tester and your customer. Customers may not be external to an organization; they may be internal as well. Regardless of who you are testing and why, establishing scope and a common language helps those impacted understand what you are doing and why and smooths the process by reducing misunderstandings.

Before describing the process of penetration testing, we need to discuss the difference between penetration testing and vulnerability assessment. These activities have different goals, but are often confused with one another. During a vulnerability assessment, some type of automated scanning product is used to probe the ports and services on a range of IP addresses. Most of these products can also test for the type of operating system and application software running and the versions, patch levels, user accounts, and services that are also running. These findings are matched up with correlating vulnerabilities in the product’s database. The end result is a large pile of data that basically states, “Here is a list of your vulnerabilities and here is a list of things you need to do to fix them.”

The problem with most vulnerability scans is, although they indicate the severity of a vulnerability, they rarely indicate its impact. This is where penetration testing comes in. Vulnerability scanning allows you to identity a piece of software as being vulnerable to exploit; a penetration test takes this further by exploiting vulnerabilities and, for example, accessing sensitive information. Most vulnerability scanners indicate what might be vulnerable based on versioning and some more invasive checks, but a penetration test indicates whether the vulnerability scanner finding is real or a false positive.

When penetration testers attack, their ultimate goal is usually to break into a system and hop from system to system until they “own” the domain or environment. Unlike a vulnerability assessment, a penetration test does not stop with the identification of a possible vulnerability. Penetration testers leverage identified vulnerabilities until they own the domain or environment. Being “owned” means either having root privileges on the most critical Unix or Linux system or owning the
domain administrator account that can access and control all of the resources on the network. Testers do this to show the customer (company) what an actual attacker can do under the circumstances and the network’s current security posture.

Many times, while a penetration tester is carrying out her procedures to gain total control of the network, she will pick up significant trophies along the way. These trophies can include the CEO’s passwords, company trade-secret documentation, administrative passwords to all border routers, documents marked “confidential” that are held on the CFO’s and CIO’s laptops, or the combination to the company vault. These trophies are collected along the way so the decision makers understand the ramifications of these vulnerabilities. A security professional can talk for hours to the CEO, CIO, or COO about services, open ports, misconfigurations, and potential vulnerabilities without making a point that this audience would understand or care about. But showing the CFO her next year’s projections, showing the CIO all of the blueprints to next year’s product line, or telling the CEO that his password is “IAmWearingPanties,” will likely inspire them to learn more about firewalls and other countermeasures that should be put into place.

---

**CAUTION** No security professional should ever try to embarrass customers or make them feel inadequate for their lack of security. This is why the security professional has been invited into the environment. She is a guest and is there to help solve the problem, not point fingers. Also, in most cases, any sensitive data should not be read by the penetration testing team because of the possibilities of future lawsuits pertaining to the use of confidential information.

In this book, we cover advanced vulnerability detection, exploitation tools, and sophisticated penetration techniques. Then we’ll dig into the programming code to show you how skilled attackers identify vulnerabilities and develop new tools to exploit their findings. Let’s take a look at the ethical penetration testing process and see how it differs from that of unethical hacker activities.

---

*The Penetration Testing Process*

Once network administrators, engineers, and security professionals understand how attackers work, they can emulate their activities to carry out a useful penetration
test. But why would anyone want to emulate an attack? Because this is the only way to truly test an environment’s security level—you must know how it will react when a real attack is being carried out.

This book is laid out to walk you through these different steps so you can understand how many types of attacks take place. It can help you develop methodologies for emulating similar activities to test your company’s security posture.

Just in case you choose to use the information in this book for unintended purposes (malicious activity), later in this chapter, we will also cover several federal laws that have been put into place to scare you away from this activity. A wide range of computer crimes is taken seriously by today’s court system, and attackers are receiving hefty fines and jail sentences for their activities. Don’t let that be you. There is just as much fun and intellectual stimulation to be had working as a good guy—and no threat of jail time!

The penetration tester’s motivation for testing is going to be driven by the client. Whether it’s to access sensitive information, provide additional justification for ongoing projects, or to just test the security of the organization, it’s important to understand what the client is looking for before testing starts. Once you understand what the goals are, directing the rest of the testing stages is much easier. Let’s look at the typical steps in a penetration test.

1. **Ground rules** Establish the ground rules:
   - Set expectations and contact information between testers and customers.
   - Identify the parties involved and who is aware of the test.
   - Set start and stop dates and blackout periods.
   - Get formalized approval and a written agreement, including scope, signatures, and legal requirements, frequently called a *Statement of Work (SOW)*.
TIP Keep this document handy during testing. You may need it as a “get out of jail free” card

2. **Passive scanning** Gather as much information about the target as possible while maintaining zero contact between the penetration tester and the target. Passive scanning, otherwise known as *Open Source Intelligence (OSINT)*, can include

- Social networking sites
- Online databases
- Google, Monster.com, etc.
- Dumpster diving

3. **Active scanning and enumeration** Probe the target’s public exposure with scanning tools, which might include

- Commercial scanning tools
- Network mapping
- Banner grabbing
- War dialing
- DNS zone transfers
- Sniffing traffic
- Wireless war driving

4. **Fingerprinting** Perform a thorough probe of the target systems to identify

- Operating system type and patch level
- Applications and patch level
- Open ports
- Running services
- User accounts
5. **Selecting target system** Identify the most useful target(s).

6. **Exploiting the uncovered vulnerabilities** Execute the appropriate attack tools targeted at the suspected exposures.
   - Some may not work.
   - Some may kill services or even kill the server.
   - Some may be successful.

7. **Escalating privilege** Escalate the security context so the ethical hacker has more control.
   - Gaining root or administrative rights
   - Using cracked password for unauthorized access
   - Carrying out buffer overflow to gain local versus remote control

8. **Documenting and reporting** Document everything found, how it was found, the tools that were used, vulnerabilities that were exploited, the timeline of activities and successes, and so on.

---

**NOTE** A more detailed approach to the attacks that are part of each methodology are included throughout the book.

*What Would an Unethical Hacker Do Differently?*

1. Target selection
   - Motivated by a grudge or for fun or profit.
   - There are no ground rules, no hands-off targets, and the security team is definitely blind to the upcoming attack.

2. Intermediaries
• The attacker launches his attack from a different system (intermediary) than his own, or a series of other systems, to make it more difficult to track back to him in case the attack is detected.
• Intermediaries are often victims of the attacker as well.

3. Penetration testing steps described in the previous section
• Scanning
• Footprinting
• Selecting target system
• Fingerprinting
• Exploiting the uncovered vulnerabilities
• Escalating privilege

4. Preserving access
• This involves uploading and installing a rootkit, backdoor, Trojaned applications, and/or bots to assure that the attacker can regain access at a later time.

5. Covering tracks
• Scrubbing event and audit logs
• Hiding uploaded files
• Hiding the active processes that allow the attacker to regain access
• Disabling messages to security software and system logs to hide malicious processes and actions

6. Hardening the system
• After taking ownership of a system, an attacker may fix the open vulnerabilities so no other attacker can use the system for other purposes.

   How the attacker uses the compromised system depends on what his or her overall goals are, which could include stealing sensitive information, redirecting financial transactions, adding the systems to his or her bot network, extorting a company, and so on. The crux is that ethical and unethical hackers carry out basically the same activities only with different intentions. If the ethical hacker does not
identify the hole in the defenses first, the unethical hacker will surely slip in and make himself at home.

**The Rise of Cyberlaw**

We currently live in a very interesting time. Information security and the legal system are becoming intertwined in a way that is straining the resources of both systems. The information security world uses terms like *bits*, *packets*, and *bandwidth*, and the legal community uses words like *jurisdiction*, *liability*, and *statutory interpretation*. In the past, these two quite different sectors had their own focus, goals, and procedures and did not collide with one another. But as computers have become the new tools for doing business and for committing traditional and new crimes, the two worlds have had to approach each other independently and then interact in a new space—a space now sometimes referred to as *cyberlaw*.

Today’s CEOs and management not only need to worry about profit margins, market analysis, and mergers and acquisitions; now they also need to step into a world of practicing security with due care, understanding and complying with new government privacy and information security regulations, risking civil and criminal liability for security failures (including the possibility of being held personally liable for certain security breaches), and trying to comprehend and address the myriad of ways in which information security problems can affect their companies. Just as businesspeople must increasingly turn to security professionals for advice in seeking to protect their company’s assets, operations, and infrastructure, so, too, must they turn to legal professionals for assistance in navigating the changing legal landscape in the privacy and information security area. Legislators, governmental and private information security organizations, and law enforcement professionals are constantly updating laws and related investigative techniques in an effort to counter each new and emerging form of attack that the bad guys come up with. Security technology developers and other professionals are constantly trying to outsmart sophisticated attackers, and vice versa. In this context, the laws being enacted provide an accumulated and constantly evolving set of rules that attempts to stay in step with new types of crimes and how they are carried out.

Cyberlaw is a broad term encompassing many elements of the legal structure that are associated with this rapidly evolving area. The increasing prominence of cyberlaw is not surprising if you consider that the first daily act of millions of American workers is to turn on their computers (frequently after they have already
made ample use of their other Internet access devices and cell phones). These acts are innocuous to most people who have become accustomed to easy and robust connections to the Internet and other networks as a regular part of life. But this ease of access also results in business risk because network openness can also enable unauthorized access to networks, computers, and data, including access that violates various laws, some of which we briefly describe in this chapter.

Cyberlaw touches on many elements of business, including how a company contracts and interacts with its suppliers and customers, sets policies for employees handling data and accessing company systems, uses computers to comply with government regulations and programs, and so on. An important subset of these laws is the group of laws directed at preventing and punishing unauthorized access to computer networks and data. This section focuses on the most significant of these laws.

Because they are expected to work in the construct the laws provide, security professionals should be familiar with these laws. A misunderstanding of these ever-evolving laws, which is certainly possible given the complexity of computer crimes, can, in the extreme case, result in the innocent being prosecuted or the guilty remaining free. And usually it is the guilty ones who get to remain free.

Understanding Individual Cyberlaws

Many countries, particularly those whose economies have more fully integrated computing and telecommunications technologies, are struggling to develop laws and rules for dealing with computer crimes. We will cover selected US federal computer-crime laws in order to provide a sample of these many initiatives; a great deal of detail regarding these laws is omitted and numerous laws are not covered. This section is intended neither to provide a thorough treatment of each of these laws, nor to cover any more than the tip of the iceberg of the many US technology laws. Instead, it is meant to raise awareness of the importance of considering these laws in your work and activities as an information security professional. That in no way means that the rest of the world is allowing attackers to run free and wild. With just a finite number of pages, we cannot properly cover all legal systems in the world or all of the relevant laws in the United States. It is important that you spend the time necessary to fully understand the laws that are relevant to your specific location and activities in the information security area.
The following sections survey some of the many US federal computer crime statutes, including

- 18 USC 1029: Fraud and Related Activity in Connection with Access Devices
- 18 USC 1030: Fraud and Related Activity in Connection with Computers
- 18 USC 2510 et seq.: Wire and Electronic Communications Interception and Interception of Oral Communications
- 18 USC 2701 et seq.: Stored Wire and Electronic Communications and Transactional Records Access
- The Digital Millennium Copyright Act
- The Cyber Security Enhancement Act of 2002

18 USC Section 1029: The Access Device Statute
The purpose of the Access Device Statute is to curb unauthorized access to accounts; theft of money, products, and services; and similar crimes. It does so by criminalizing the possession, use, or trafficking of counterfeit or unauthorized access devices or device-making equipment, and other similar activities (described shortly), to prepare for, facilitate, or engage in unauthorized access to money, goods, and services. It defines and establishes penalties for fraud and illegal activity that can take place through the use of such counterfeit access devices.

The *elements* of a crime are generally the things that need to be shown in order for someone to be prosecuted for that crime. These elements include consideration of the potentially illegal activity in light of the precise definitions of *access device*, *counterfeit access device*, *unauthorized access device*, *scanning receiver*, and other definitions that together help to define the scope of the statute’s application.

The term *access device* refers to a type of application or piece of hardware that is created specifically to generate access credentials (passwords, credit card numbers, long-distance telephone service access codes, PINs, and so on) for the purpose of unauthorized access. Specifically, it is defined broadly to mean any card, plate, code, account number, electronic serial number, mobile identification number, personal identification number, or other telecommunications service, equipment, or instrument identifier, or other means of account access that can be used, alone or in conjunction with another access device, to obtain money, goods,
services, or any other thing of value, or that can be used to initiate a transfer of funds (other than a transfer originated solely by paper instrument).⁶

One example of a violation would be using a tool to steal credentials and then using those credentials to break into the Pepsi-Cola Network. If you were to steal the soda recipe, you would be guilty of “Using or obtaining an access device to gain unauthorized access and obtain anything of value totaling $1,000 or more during a one-year period.” This would result in a fine of upward of $10,000 or twice the value of the damages and up to 10 years in prison. If you were caught twice, you could get up to 20 years in prison.

Section 1029 addresses offenses that involve generating or illegally obtaining access credentials, which can involve just obtaining the credentials or obtaining and using them. These activities are considered criminal whether or not a computer is involved—unlike the statute discussed next, which pertains to crimes dealing specifically with computers.

18 USC Section 1030 of the Computer Fraud and Abuse Act

The Computer Fraud and Abuse Act (CFAA) (as amended by the USA Patriot Act) is an important federal law that addresses acts that compromise computer network security.⁷ It prohibits unauthorized access to computers and network systems, extortion through threats of such attacks, the transmission of code or programs that cause damage to computers, and other related actions. It addresses unauthorized access to government, financial institutions, and other computer and network systems, and provides for civil and criminal penalties for violators. The act outlines the jurisdiction of the FBI and Secret Service.

The term protected computer, as commonly put forth in the CFAA, means a computer used by the US government, financial institutions, or any system used in interstate or foreign commerce or communications. The CFAA is the most widely referenced statute in the prosecution of many types of computer crimes. A casual reading of the CFAA suggests that it only addresses computers used by government agencies and financial institutions, but there is a small (but important) clause that extends its reach. This clause says that the law applies also to any system “used in interstate or foreign commerce or communication.” The meaning of “used in interstate or foreign commerce or communication” is very broad, and, as a result, CFAA operates to protect nearly all computers and networks. Almost every computer connected to a network or the Internet is used for some type of commerce or
communication, so this small clause pulls nearly all computers and their uses under the protective umbrella of the CFAA. Amendments by the USA Patriot Act to the term “protected computer” under CFAA extended the definition to any computers located outside the United States, as long as they affect interstate or foreign commerce or communication of the United States. So if the United States can get the attackers, they will attempt to prosecute them no matter where in the world they live.

The CFAA has been used to prosecute many people for various crimes. Two types of unauthorized access can be prosecuted under the CFAA: these include wholly unauthorized access by outsiders, and also situations where individuals, such as employees, contractors, and others with permission, exceed their authorized access and commit crimes. The CFAA states that if someone accesses a computer in an unauthorized manner or exceeds his or her access rights, that individual can be found guilty of a federal crime. This clause allows companies to prosecute employees who carry out fraudulent activities by abusing (and exceeding) the access rights their company has given them.

In November 2013, US-CERT released an advisory about CryptoLocker Ransomware that will encrypt the contents of a computer and then charge the victim for the keys to unlock it. One area in which 18 USC Section 1030 would come into play would be if the CryptoLocker software was used to encrypt a government system. The CryptoLocker demands payment, which is considered extortion. Under the CFAA, if the attackers are caught this could yield up to a $250,000 fine as well as up to 10 years in prison for the first offense.

Under the CFAA, the FBI and the Secret Service have the responsibility for handling these types of crimes, and they have their own jurisdictions. The FBI is responsible for cases dealing with national security, financial institutions, and organized crime. The Secret Service’s jurisdiction encompasses any crimes pertaining to the Treasury Department and any other computer crime that does not fall within the FBI’s jurisdiction.
The Secret Service’s jurisdiction and responsibilities have grown since the Department of Homeland Security (DHS) was established. The Secret Service now deals with several areas to protect the nation and has established an Information Analysis and Infrastructure Protection division to coordinate activities in this area. This division’s responsibilities encompass the preventive procedures for protecting “critical infrastructure,” which includes such things as power grids, water supplies, and nuclear plants in addition to computer systems.

State Law Alternatives The amount of damage resulting from a violation of the CFAA can be relevant for either a criminal or civil action. As noted earlier, the CFAA provides for both criminal and civil liability for a violation. A criminal violation is brought by a government official and is punishable by either a fine or imprisonment or both. By contrast, a civil action can be brought by a governmental entity or a private citizen and usually seeks the recovery of payment of damages incurred and an injunction, which is a court order to prevent further actions prohibited under the statute. The amount of damages is relevant for some but not all of the activities that are prohibited by the statute. The victim must prove that damages have indeed occurred. In this case, damage is defined as disruption of the availability or integrity of data, a program, a system, or information. For most CFAA violations, the losses must equal at least $5,000 during any one-year period.

This all sounds great and might allow you to sleep better at night, but not all of the harm caused by a CFAA violation is easily quantifiable, or if quantifiable, may not exceed the $5,000 threshold. For example, when computers are used in distributed denial-of-service attacks or when processing power is being used to brute-force and uncover an encryption key, the issue of damages becomes cloudy. These losses do not always fit into a nice, neat formula to evaluate whether they total $5,000. The victim of an attack can suffer various qualitative harms that are much harder to quantify. If you find yourself in this type of situation, the CFAA might not provide adequate relief. In that context, this federal statute might not be a useful tool for you and your legal team.

Often victims will turn to state laws that may offer more flexibility when prosecuting an attacker. State laws that are relevant in the computer crime arena include both new state laws being passed by state legislatures in an attempt to
protect their residents and traditional state laws dealing with trespassing, theft, larceny, money laundering, and other crimes.

Resorting to state laws is not, however, always straightforward. First, there are 50 different states and nearly that many different “flavors” of state law. Thus, for example, trespass law varies from one state to the next, resulting in a single activity being treated in two very different ways under state law. Some states require a demonstration of damages as part of the claim of trespass (not unlike the CFAA requirement), whereas other states do not require a demonstration of damages in order to establish that an actionable trespass has occurred.

Importantly, a company will usually want to bring a case to the courts of a state that has the most favorable definition of a crime so it can most easily make its case. Companies will not, however, have total discretion as to where they bring the case to court. There must generally be some connection, or nexus, to a state in order for the courts of that state to have jurisdiction to hear a case.

**TIP** If you are considering prosecuting a computer crime that affected your company, start documenting the time people have to spend on the issue and other costs incurred in dealing with the attack. This lost paid employee time and other costs may be relevant in the measure of damages or, in the case of the CFAA or those states that require a showing of damages as part of a trespass case, to the success of the case.

As with all of the laws summarized in this chapter, information security professionals must be careful to confirm with each relevant party the specific scope and authorization for work to be performed. If these confirmations are not in place, it could lead to misunderstandings and, in the extreme case, prosecution under the Computer Fraud and Abuse Act or other applicable law. In the case of Sawyer vs. Department of Air Force, the court rejected an employee’s claim that alterations to computer contracts were made to demonstrate the lack of security safeguards and found the employee liable because the statute only required proof of use of a computer system for any unauthorized purpose.
18 USC Sections 2510, et. Seq., and 2701, et. Seq., of the Electronic Communications Privacy Act

These sections are part of the Electronic Communications Privacy Act (ECPA), which is intended to protect communications from unauthorized access. The ECPA, therefore, has a different focus than the CFAA, which is directed at protecting computers and network systems. Most people do not realize that the ECPA is made up of two main parts: one that amended the Wiretap Act and the other than amended the Stored Communications Act, each of which has its own definitions, provisions, and cases interpreting the law.

The Wiretap Act has been around since 1918, but the ECPA extended its reach to electronic communication when society moved in that direction. The Wiretap Act protects communications, including wire, oral, and data during transmission, from unauthorized access and disclosure (subject to exceptions). The Stored Communications Act protects some of the same types of communications before and/or after the communications are transmitted and stored electronically somewhere. Again, this sounds simple and sensible, but the split reflects a recognition that there are different risks and remedies associated with active versus stored communications.

The Wiretap Act generally provides that there cannot be any intentional interception of wire, oral, or electronic communication in an illegal manner. Among the continuing controversies under the Wiretap Act is the meaning of the word interception. Does it apply only when the data is being transmitted as electricity or light over some type of transmission medium? Does the interception have to occur at the time of the transmission? Does it apply to this transmission and to where it is temporarily stored on different hops between the sender and destination? Does it include access to the information received from an active interception, even if the person did not participate in the initial interception? The question of whether an interception has occurred is central to the issue of whether the Wiretap Act applies.

Although the ECPA seeks to limit unauthorized access to communications, it recognizes that some types of unauthorized access are necessary. For example, if the government wants to listen in on phone calls, Internet communication, email, network traffic, or you whispering into a tin can, it can do so if it complies with safeguards established under the ECPA that are intended to protect the privacy of persons who use those systems.
The DMCA is not often considered in a discussion of hacking and the question of information security, but it is relevant. The DMCA was passed in 1998 to implement the World Intellectual Property Organization Copyright Treaty (WIPO Copyright Treaty). The WIPO Treaty requires treaty parties to “provide adequate legal protection and effective legal remedies against the circumvention of effective technological measures that are used by authors,” and to restrict acts in respect to their works that are not authorized. Thus, while the CFAA protects computer systems and the ECPA protects communications, the DMCA protects certain (copyrighted) content itself from being accessed without authorization. The DMCA establishes both civil and criminal liability for the use, manufacture, and trafficking of devices that circumvent technological measures controlling access to, or protection of, the rights associated with copyrighted works.

The DMCA’s anti-circumvention provisions make it criminal to willfully, and for commercial advantage or private financial gain, circumvent technological measures that control access to protected copyrighted works. In hearings, the crime that the anti-circumvention provision is designed to prevent has been described as “the electronic equivalent of breaking into a locked room in order to obtain a copy of a book.”

Circumvention is to “descramble a scrambled work...decrypt an encrypted work, or otherwise...avoid, bypass, remove, deactivate, or impair a technological measure, without the authority of the copyright owner.” The legislative history provides that “if unauthorized access to a copyrighted work is effectively prevented through use of a password, it would be a violation of this section to defeat or bypass the password.” A “technological measure” that “effectively controls access” to a copyrighted work includes measures that “in the ordinary course of its operation, requires the application of information, or a process or a treatment, with the authority of the copyright owner, to gain access to the work.” Therefore, measures that can be deemed to “effectively control access to a work” would be those based on encryption, scrambling, authentication, or some other measure that requires the use of a key provided by a copyright owner to gain access to a work.

Said more directly, the Digital Millennium Copyright Act (DMCA) states that no one should attempt to tamper with and break an access control mechanism that is put into place to protect an item that is protected under the copyright law. If you have created a nifty little program that controls access to all of your written
interpretations of the grandness of the invention of pickled green olives, and someone tries to break this program to gain access to your copyright-protected insights and wisdom, the DMCA could come to your rescue.

The fear of many in the information security industry is that this provision could be interpreted and used to prosecute individuals carrying out commonly applied security practices. For example, a penetration test is a service performed by information security professionals in which an individual or team attempts to break or slip by access control mechanisms. Security classes are offered to teach people how these attacks take place so they can understand what countermeasures are appropriate and why. But how will people learn how to hack, crack, and uncover vulnerabilities and flaws if the DMCA indicates that classes, seminars, and the like cannot be conducted to teach the security professionals these skills?

The DMCA provides an explicit exemption allowing “encryption research” for identifying the flaws and vulnerabilities of encryption technologies. It also provides for an exception for engaging in an act of security testing (if the act does not infringe on copyrighted works or violate applicable law such as the CFAA), but it does not contain a broader exemption covering a variety of other activities that information security professionals might engage in. Yes, as you pull one string, three more show up. Again, you see why it’s important for information security professionals to have a fair degree of familiarity with these laws to avoid missteps.

Cyber Security Enhancement Act of 2002
Several years ago, Congress determined that the legal system still allowed for too much leeway for certain types of computer crimes and that some activities not labeled “illegal” needed to be. In July 2002, the House of Representatives voted to put stricter laws in place, and to dub this new collection of laws the Cyber Security Enhancement Act (CSEA) of 2002. The CSEA made a number of changes to federal law involving computer crimes.

The act stipulates that attackers who carry out certain computer crimes may now get a life sentence in jail. If an attacker carries out a crime that could result in another’s bodily harm or possible death, or a threat to public health or safety, the attacker could face life in prison. This does not necessarily mean that someone has to throw a server at another person’s head, but since almost everything today is run by some type of technology, personal harm or death could result from what would otherwise be a run-of-the-mill hacking attack. For example, if an attacker were to
compromise embedded computer chips that monitor hospital patients, cause fire trucks to report to wrong addresses, make all of the traffic lights change to green, or reconfigure airline controller software, the consequences could be catastrophic and under the CSEA result in the attacker spending the rest of her days in jail.

NOTE In 2013, a newer version of the Cyber Security Enhancement Act passed the House and is still on the docket for the Senate to take action, at the time of this writing. Its purpose includes funding for cybersecurity development, research, and technical standards.

The CSEA was also developed to supplement the Patriot Act, which increased the US government’s capabilities and power to monitor communications. One way in which this is done is that the CSEA allows service providers to report suspicious behavior without risking customer litigation. Before this act was put into place, service providers were in a sticky situation when it came to reporting possible criminal behavior or when trying to work with law enforcement. If a law enforcement agent requested information on a provider’s customer and the provider gave it to them without the customer’s knowledge or permission, the service provider could, in certain circumstances, be sued by the customer for unauthorized release of private information. Now service providers can report suspicious activities and work with law enforcement without having to tell the customer. This and other provisions of the Patriot Act have certainly gotten many civil rights monitors up in arms.

It is up to you which side of the fight you choose to play on—but remember that computer crimes are not treated as lightly as they were in the past. Trying out a new tool or pressing Start on an old tool may get you into a place you never intended—jail. So as your mother told you—be good, and may the Force be with you.

The Controversy of “Hacking” Tools
In most instances, the toolset used by malicious attackers is the same toolset used by security professionals. Many people do not understand this. In fact, the books, classes, articles, websites, and seminars on hacking could be legitimately renamed to
“security professional toolset education.” The problem arises when marketing people like to use the word hacking because it draws more attention and paying customers.

As covered earlier, ethical hackers go through the same processes and procedures as unethical hackers, so it only makes sense that they use the same basic toolset. It would not be useful to prove that attackers could not get through the security barriers with Tool A if attackers do not use Tool A. The ethical hacker has to know what the bad guys are using, know the new exploits that are out in the underground, and continually keep her skills and knowledgebase up to date. Why? Because, odds are against the company and the security professional. The security professional has to identify and address all of the vulnerabilities in an environment. The attacker only has to be really good at one or two exploits, or really lucky. A comparison can be made to the US Homeland Security responsibilities. The CIA and FBI are responsible for protecting the nation from the 10 million things terrorists could possibly think up and carry out. The terrorist only has to be successful at one of these 10 million things.

**Vulnerability Disclosure**

For years customers have demanded that operating systems and applications provide more and more functionality. Vendors continually scramble to meet this demand while also attempting to increase profits and market share. This combination of racing to market and maintaining a competitive advantage has resulted in software containing many flaws—flaws that range from mere nuisances to critical and dangerous vulnerabilities that directly affect a customer’s protection level.

The hacking community’s skill sets are continually increasing. It used to take the hacking community months to carry out a successful attack from an identified vulnerability; today it happens in days or even hours. The increase in interest and talent in the criminal community equates to quicker and more damaging attacks and malware for the industry to combat. It is imperative that vendors not sit on the discovery of true vulnerabilities, but instead work to release fixes to customers who need them as soon as possible.

For this to happen, ethical hackers must understand and follow the proper methods for disclosing identified vulnerabilities to the software vendor. If an
individual uncovers a vulnerability and illegally exploits it and/or tells others how to carry out this activity, he is considered a **black hat**. If an individual uncovers a vulnerability and exploits it with authorization, she is considered a **white hat**. If a different person uncovers a vulnerability, does not illegally exploit it or tell others how to do so, and works with the vendor to fix it, this person is considered a **gray hat**.

We promote using the knowledge that we are sharing with you in a responsible manner that will only help the industry—not hurt it. To do this, you should understand the policies, procedures, and guidelines that have been developed to allow hackers and vendors to work together.

**Different Teams and Points of View**

Unfortunately, almost all of today’s software products are riddled with flaws. These flaws can present serious security concerns for consumers. For customers who rely extensively on applications to perform core business functions, bugs can be crippling and, therefore, must be dealt with properly. How best to address the problem is a complicated issue because it involves two key players who usually have very different views on how to achieve a resolution.

The first player is the consumer. An individual or company buys a product, relies on it, and expects it to work. Often, the consumer owns a community of interconnected systems (a network) that all rely on the successful operation of software to do business. When the consumer finds a flaw, he reports it to the vendor and expects a solution in a reasonable timeframe.

The second player is the software vendor. The vendor develops the product and is responsible for its successful operation. The vendor is looked to by thousands of customers for technical expertise and leadership in the upkeep of its product. When a flaw is reported to the vendor, it is usually one of many that the vendor must deal with, and some fall through the cracks for one reason or another.

The issue of public disclosure has created quite a stir in the computing industry because each group views the issue so differently. Many believe knowledge is the public’s right, and all security vulnerability information should be disclosed as a matter of principle. Furthermore, many consumers feel that the only way to get truly quick results from a large software vendor is to pressure it to fix the problem by threatening to make the information public. Vendors have had the reputation of simply plodding along and delaying the fixes until a later version or patch is
scheduled for release, which will address the flaw. This approach doesn’t always consider the best interests of consumers, however, as they must sit and wait for the vendor to fix a vulnerability that puts their business at risk.

The vendor looks at the issue from a different perspective. Disclosing sensitive information about a software flaw causes two major problems. First, the details of the flaw will help attackers exploit the vulnerability. The vendor’s argument is that if the issue is kept confidential while a solution is being developed, attackers will not know how to exploit the flaw. Second, the release of this information can hurt the company’s reputation, even in circumstances when the reported flaw is later proven to be false. It is much like a smear campaign in a political race that appears as the headline story in a newspaper. Reputations are tarnished, and even if the story turns out to be untrue, a retraction is usually printed on the back page a week later. Vendors fear the same consequence for massive releases of vulnerability reports.

Because of these two distinct viewpoints, several organizations have rallied together to create policies, guidelines, and general suggestions on how to handle software vulnerability disclosures. This section will attempt to cover the issue from all sides and help educate you on the fundamentals behind the ethical disclosure of software vulnerabilities.

*How Did We Get Here?*

Before the mailing list Bugtraq was created, individuals who uncovered vulnerabilities and ways to exploit them just communicated directly with each other. The creation of Bugtraq provided an open forum for these individuals to discuss the same issues and work collectively. Easy access to ways of exploiting vulnerabilities gave way to the numerous script-kiddie point-and-click tools available today, which allow people who do not even understand a vulnerability to exploit it successfully. Bugtraq led to an increase in attacks on the Internet, on networks, and against vendors. Many vendors were up in arms, demanding a more responsible approach to vulnerability disclosure.

In 2002, Internet Security Systems (ISS) discovered several critical vulnerabilities in products like Apache web server, Solaris X Windows font service, and Internet Software Consortium BIND software. ISS worked with the vendors directly to come up with solutions. A patch that was developed and released by Sun Microsystems was flawed and had to be recalled. An Apache patch was not released to the public until after the vulnerability was posted through public disclosure, even though the
vendor knew about the vulnerability. Although these are older examples, these types of activities—and many more like them—left individuals and companies vulnerable; they were victims of attacks and eventually developed a deep feeling of distrust of software vendors. Critics also charged that security companies, like ISS, have alternative motives for releasing this type of information. They suggest that by releasing system flaws and vulnerabilities, they generate “good press” for themselves and thus promote new business and increased revenue.

Because of the failures and resulting controversy that ISS encountered, it decided to initiate its own disclosure policy to handle such incidents in the future. It created detailed procedures to follow when discovering a vulnerability and how and when that information would be released to the public. Although their policy is considered “responsible disclosure,” in general, it does include one important caveat—vulnerability details would be released to its customers and the public at a “prescribed period of time” after the vendor has been notified. ISS coordinates their public disclosure of the flaw with the vendor’s disclosure. This policy only fueled the people who feel that vulnerability information should be available for the public to protect themselves.

This dilemma, and many others, represent the continual disconnect among vendors, security companies, and gray hat hackers today. Differing views and individual motivations drive each group down various paths. The models of proper disclosure that are discussed in upcoming sections have helped these entities to come together and work in a more concerted effort, but much bitterness and controversy around this issue remains.

NOTE The range of emotion, the numerous debates, and controversy over the topic of full disclosure has been immense. Customers and security professionals alike are frustrated with software flaws that still exist in the products in the first place and the lack of effort from vendors to help in this critical area. Vendors are frustrated because exploitable code is continually released just as they are trying to develop fixes. We will not be taking one side or the other of this debate, but will do our best to tell you how you can help, and not hurt, the process.
CERT’s Current Process

The first place to turn to when discussing the proper disclosure of software vulnerabilities is the governing body known as the CERT Coordination Center (CC). CERT/CC is a federally funded research and development operation that focuses on Internet security and related issues. Established in 1988 in reaction to the first major virus outbreak on the Internet, the CERT/CC has evolved over the years, taking on more substantial roles in the industry, which include establishing and maintaining industry standards for the way technology vulnerabilities are disclosed and communicated. In 2000, the organization issued a policy that outlined the controversial practice of releasing software vulnerability information to the public. The policy covered the following areas:

- Full disclosure will be announced to the public within 45 days of being reported to CERT/CC. This timeframe will be executed even if the software vendor does not have an available patch or appropriate remedy. The only exception to this rigid deadline will be exceptionally serious threats or scenarios that would require a standard to be altered.

- CERT/CC will notify the software vendor of the vulnerability immediately so a solution can be created as soon as possible.

- Along with the description of the problem, CERT/CC will forward the name of the person reporting the vulnerability unless the reporter specifically requests to remain anonymous.

- During the 45-day window, CERT/CC will update the reporter on the current status of the vulnerability without revealing confidential information.

   CERT/CC states that its vulnerability policy was created with the express purpose of informing the public of potentially threatening situations while offering the software vendor an appropriate timeframe to fix the problem. The independent body further states that all decisions on the release of information to the public are based on what is best for the overall community.

   The decision to go with 45 days was met with controversy as consumers widely felt that was too much time to keep important vulnerability information concealed. The vendors, on the other hand, felt the pressure to create solutions in a short timeframe while also shouldering the obvious hits their reputations would take as news spread about flaws in their product. CERT/CC came to the conclusion that 45
days was sufficient enough time for vendors to get organized, while still taking into account the welfare of consumers.

To accommodate vendors and their perspective of the problem, CERT/CC performs the following:

• CERT/CC will make good faith efforts always to inform the vendor before releasing information so there are no surprises.

• CERT/CC will solicit vendor feedback in serious situations and offer that information in the public release statement. In instances when the vendor disagrees with the vulnerability assessment, the vendor’s opinion will be released as well, so both sides can have a voice.

• Information will be distributed to all related parties that have a stake in the situation prior to the disclosure. Examples of parties that could be privy to confidential information include participating vendors, experts who could provide useful insight, Internet Security Alliance members, and groups that may be in the critical path of the vulnerability.

Although there have been other guidelines developed and implemented after CERT’s model, CERT is usually the “middle man” between the bug finder and the vendor to try and help the process and enforce the necessary requirements of all of the parties involved.

Organization for Internet Safety

There are three basic types of vulnerability disclosures: full disclosure, partial disclosure, and nondisclosure. Each type has its advocates, and long lists of pros and cons can be debated regarding each type. The Organization for Internet Safety (OIS) was created to help meet the needs of all groups and is the policy that best fits into a partial disclosure classification. This section gives an overview of the OIS approach, as well as provides the step-by-step methodology that has been developed to provide a more equitable framework for both the user and the vendor.

A group of researchers and vendors formed the OIS with the goal of improving the way software vulnerabilities are handled. The OIS members included stake, BindView Corp., The SCO Group, Foundstone, Guardent, Internet Security Systems, McAfee, Microsoft Corporation, Network Associates, Oracle Corporation, SGI, and Symantec. The OIS shut down after serving its purpose, which was to create the vulnerability disclosure guidelines.
The OIS believed that vendors and consumers should work together to identify issues and devise reasonable resolutions for both parties. It tried to bring together a broad, valued panel that offered respected, unbiased opinions to make recommendations. The model was formed to accomplish two goals:

- Reduce the risk of software vulnerabilities by providing an improved method of identification, investigation, and resolution.
- Improve the overall engineering quality of software by tightening the security placed on the end product.

**Responsible Disclosure Phases**

Understanding the steps of responsible disclosure under the OIS model are critical. This process is summarized here; however, a detailed methodology with examples and process maps are available as part of the standard:

1. **Discovery** A flaw has been found. The researcher must discover if a vulnerability has already been reported or patched, ensure it can be reproduced consistently, and ensure it impacts the default configuration. If so, the discoverer creates a **vulnerability summary report (VSR)**.

2. **Notification** The discoverer submits his contact information as well as the VSR to the vendor referencing the vendor’s security policy. These details are sent to the address listed in its security policy or to one of the standard email addresses laid out in the OIS standard. The vendor must respond to this step.

3. **Validation** The vendor researches and validates the vulnerability. Regular status updates to the reporter are suggested during this phase.

4. **Findings** Once the vendor finishes its investigation, it confirms, disproves, or indicates inconclusive findings. The vendor is required to demonstrate research was done and typically meets this requirement by providing lists of products, versions, and tests performed.

5. **Resolution** If a flaw is inconclusive or is disproven, the weakness may be made public. If it is confirmed, the vendor typically has 30 days to issue a patch or fix.

6. **Release** The remedy is released as well as the notification.
Conflicts Will Still Exist

Those who discover vulnerabilities usually are motivated to protect the industry by identifying and helping remove dangerous software from commercial products. A little fame, admiration, and bragging rights are also nice for those who enjoy having their egos stroked. Vendors, on the other hand, are motivated to improve their product, avoid lawsuits, stay clear of bad press, and maintain a responsible public image.

There’s no question that software flaws are rampant. The Common Vulnerabilities and Exposures (CVE) list is a compilation of publicly known vulnerabilities. This list is over ten years old and catalogs more than 40,000 bugs. This list is frequently updated, and through a joint collaboration with MITRE and National Institute of Standards and Technology (NIST), the National Vulnerability Database (NVD) provides a searchable database for these CVE advisories at http://nvd.nist.gov/

Vulnerability reporting considerations include financial, legal, and moral ones for both researchers and vendors alike. Vulnerabilities can mean bad public relations for a vendor that, to improve its image, must release a patch once a flaw is made public. But, at the same time, vendors may decide to put the money into fixing software after it’s released to the public, rather than making it perfect (or closer to perfect) beforehand. In that way, they use vulnerability reporting as after-market security consulting.

Public disclosure helps improve security, according to information security expert Bruce Schneier. He says that the only reason vendors patch vulnerabilities is because of full disclosure, and that there’s no point in keeping a bug a secret—hackers will discover it anyway. Before full disclosure, he says, it was too easy for software companies to ignore the flaws and threaten the researcher with legal action. Ignoring the flaws was easier for vendors especially because an unreported flaw affected the software’s users much more than it affected the vendor.

Security expert Marcus Ranum takes a dim view of public disclosure of vulnerabilities. He says that an entire economy of researchers is trying to cash in on the vulnerabilities that they find and selling them to the highest bidder, whether for good or bad purposes. His take is that researchers are constantly seeking fame and that vulnerability disclosure is “rewarding bad behavior,” rather than making software better.
But the vulnerability researchers who find and report bugs have a different take, especially when they aren’t getting paid. Another issue that has arisen is that researchers are tired of working for free without legal protection.

“No More Free Bugs”
In 2009, several gray hat hackers—Charlie Miller, Alex Sotirov, and Dino Dai Zovi—publicly announced a new stance: “No More Free Bugs.” They argue that the value of software vulnerabilities often doesn’t get passed on to independent researchers who find legitimate, serious flaws in commercial software. Along with iDefense and ZDI, the software vendors themselves have their own employees and consultants who are supposed to find and fix bugs. (“No More Free Bugs” is targeted primarily at the for-profit software vendors that hire their own security engineer employees or consultants.)

The researchers involved in “No More Free Bugs” also argue that independent researchers are putting themselves at risk when they report vulnerabilities to vendors. They have no legal protection when they disclose a found vulnerability—so they’re not only working for free, but also opening themselves up to threats of legal action, too. And independent researchers don’t often have access to the right people at the software vendor, those who can create and release the necessary patches. For many vendors, vulnerabilities mainly represent threats to their reputation and bottom line, and they may stonewall researchers’ overtures, or worse. Although vendors create responsible disclosure guidelines for researchers to follow, they don’t maintain guidelines for how they treat the researchers.

Furthermore, these researchers say that software vendors often depend on them to find bugs rather than investing enough in finding vulnerabilities themselves. Uncovering flaws in today’s complex software takes time and skill, and the founders of the “No More Free Bugs” movement feel as though either the vendors should employ people to uncover these bugs and identify fixes or they should pay gray hats who uncover them and report them responsibly.

This group of researchers also calls for more legal options when carrying out and reporting on software flaws. In some cases, researchers have uncovered software flaws and vendors have then threatened these individuals with lawsuits to keep them quiet and help ensure the industry did not find out about the flaws.
NOTE For a sample list of security research that resulted in legal action as well as the outcome, visit http://attrition.org/errata/legal_threats/.

*Bug Bounty Programs*

In recent years, vendors have adopted some of the previous principles as part of Bug Bounty programs. Microsoft, for example, says it won’t sue researchers “that responsibly submit potential online services security vulnerabilities.” And Mozilla runs a “bug bounty program” that offers researchers a flat $500 fee (plus a T-shirt!) for reporting valid, critical vulnerabilities. In 2009, Google offered a cash bounty for the best vulnerability found in Native Client. Organizations have even developed a business plan on managing these bug bounty programs. One example is BugCrowd, a site that puts testers together with clients who want software tested and are willing to pay for it.

Although more and more software vendors are reacting appropriately when vulnerabilities are reported (because of market demand for secure products), many people believe that vendors will not spend the extra money, time, and resources to carry out this process properly until they are held legally liable for software security issues. The possible legal liability issues software vendors may or may not face in the future is a can of worms we will not get into, but these issues are gaining momentum in the industry.

The Zero-Day Initiative (ZDI) is another organization that pays for vulnerability disclosure. It offers a web portal for researchers to report and track vulnerabilities. ZDI performs identity checks on researchers who report vulnerabilities, including checking that the researcher isn’t on any government “do not do business with” lists. ZDI then validates the bug in a security lab before offering the researcher payment and contacting the vendor. ZDI also maintains its intrusion prevention system (IPS) program to write filters for whatever customer areas are affected by the vulnerability. The filter descriptions are designed to protect customers, but remain vague enough to keep details of unpatched flaws secret. ZDI works with the vendor on notifying the public when the patch is ready, giving the researcher credit if he or she requests it.

**Summary**
Before you can embark on an exploration of ethical hacking, you need to understand where ethical hacking and criminal activity are similar and deviate. With this knowledge, you can better understand what steps you need to take to model this malicious activity in order to help assess the security of environments with realistic benchmarks. While doing this, it’s also important to understand the legal aspects of the business process as well as any applicable local, state, and federal laws.

Through this chapter, we covered why understanding how malicious individuals work is important, and how the steps of the ethical hacking process map to the methodology of an attacker. We also covered a number of laws that impact ethical hackers in the United States, including DCMA and CFAA. We also detailed reasons to check on local laws before preforming penetration testing to ensure that there aren’t laws that are more strict than federal ones.

Finally, we covered why ethical disclosure is important and how to deal properly with the disclosure process. Armed with this information, you should understand the steps of getting work as an ethical hacker, ensuring that you stay safe while testing, and as you discover new flaws, how to contribute back to the community effectively.

References


For Further Reading
CHAPTER 2

Programming Survival Skills

Why study programming? Ethical gray hat hackers should study programming and learn as much about the subject as possible in order to find vulnerabilities in programs and get them fixed before unethical hackers take advantage of them. It is very much a foot race: if the vulnerability exists, who will find it first? The purpose of this chapter is to give you the survival skills necessary to understand upcoming chapters and later find the holes in software before the black hats do.

In this chapter, we cover the following topics:

- C programming language
- Computer memory
- Intel processors
- Assembly language basics
C Programming Language

The C programming language was developed in 1972 by Dennis Ritchie from AT&T Bell Labs. The language was heavily used in Unix and is thereby ubiquitous. In fact, much of the staple networking programs and operating systems are based in C.

Basic C Language Constructs

Although each C program is unique, there are common structures that can be found in most programs. We’ll discuss these in the next few sections.

main()

All C programs contain a main() structure (lowercase) that follows this format:

```c
<optional return value type> main(<optional argument>) {
  <optional procedure statements or function calls>;
}
```

where both the return value type and arguments are optional. If you use command-line arguments for main(), use the format

```c
<optional return value type> main(int argc, char * argv[]){
```

where the argc integer holds the number of arguments and the argv array holds the input arguments (strings). The parentheses and brackets are mandatory, but white space between these elements does not matter. The brackets are used to denote the beginning and end of a block of code. Although procedure and function calls are optional, the program would do nothing without them. Procedure statements are simply a series of commands that perform operations on data or variables and normally end with a semicolon.

Functions

Functions are self-contained bundles of algorithms that can be called for execution by main() or other functions. Technically, the main() structure of each C program is also a function; however, most programs contain other functions. The format is as follows:

```c
<optional return value type > function name (<optional function argument>){
```
The first line of a function is called the *signature*. By looking at it, you can tell if the function returns a value after executing or requires arguments that will be used in processing the procedures of the function.

The call to the function looks like this:

```plaintext
<optional variable to store the returned value => function name (arguments if called for by the function signature);
```

Again, notice the required semicolon at the end of the function call. In general, the semicolon is used on all stand-alone command lines (not bounded by brackets or parentheses).

Functions are used to modify the flow of a program. When a call to a function is made, the execution of the program temporarily jumps to the function. After execution of the called function has completed, the program continues executing on the line following the call. This process will make more sense during our discussion of stack operations in Chapter 10.

*Variables*

*Variables* are used in programs to store pieces of information that may change and may be used to dynamically influence the program. Table 2-1 shows some common types of variables.

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Use</th>
<th>Typical Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Stores signed integer values such as 314 or -314</td>
<td>4 bytes for 32-bit machines 2 bytes for 16-bit machines</td>
</tr>
<tr>
<td>float</td>
<td>Stores signed floating-point numbers such as -3.234</td>
<td>4 bytes</td>
</tr>
<tr>
<td>double</td>
<td>Stores large floating-point numbers</td>
<td>8 bytes</td>
</tr>
<tr>
<td>char</td>
<td>Stores a single character such as “d”</td>
<td>1 byte</td>
</tr>
</tbody>
</table>

**Table 2-1 Types of Variables**

When the program is compiled, most variables are preallocated memory of a fixed size according to system-specific definitions of size. Sizes in Table 2-1 are considered typical; there is no guarantee that you will get those exact sizes. It is left up to the hardware implementation to define this size. However, the function `sizeof()` is used in C to ensure that the correct sizes are allocated by the compiler.
Variables are typically defined near the top of a block of code. As the compiler chews up the code and builds a symbol table, it must be aware of a variable before it is used in the code later. This formal declaration of variables is done in the following manner:

<variable type> <variable name> <optional initialization starting with “=”>;

For example,

int a = 0;

where an integer (normally 4 bytes) is declared in memory with a name of `a` and an initial value of `0`.

Once declared, the assignment construct is used to change the value of a variable. For example, the statement

\( x = x + 1; \)

is an assignment statement containing a variable `x` modified by the `+` operator. The new value is stored into `x`. It is common to use the format

\[ \text{destination} = \text{source} \text{ with optional operators} \]

where `destination` is the location in which the final outcome is stored.

`printf`

The C language comes with many useful constructs for free (bundled in the libc library). One of the most commonly used constructs is the `printf` command, generally used to print output to the screen. There are two forms of the `printf` command:

`printf(<string>);`
`printf(<format string>, <list of variables/values>);`

The first format is straightforward and is used to display a simple string to the screen. The second format allows for more flexibility through the use of a format string that can be composed of normal characters and special symbols that act as placeholders for the list of variables following the comma. Commonly used format symbols are listed and described in Table 2-2.
Table 2-2 printf Format Symbols

These format symbols may be combined in any order to produce the desired output. Except for the \n symbol, the number of variables/values needs to match the number of symbols in the format string; otherwise, problems will arise, as described in our discussion of format string exploits in Chapter 11.

**scanf**
The `scanf` command complements the `printf` command and is generally used to get input from the user. The format is as follows:

```
scanf(<format string>, <list of variables/values>);
```

where the format string can contain format symbols such as those shown for `printf` in Table 2-2. For example, the following code will read an integer from the user and store it into the variable called `number`:

```
scanf("%d", &number);
```

Actually, the & symbol means we are storing the value into the memory location pointed to by `number`; that will make more sense when we talk about pointers later in the chapter in the “Pointers” section. For now, realize that you must use the & symbol before any variable name with `scanf`. The command is smart enough to change types on-the-fly, so if you were to enter a character in the previous command prompt, the command would convert the character into the decimal (ASCII) value automatically. Bounds checking is not done in regard to string size, however, which may lead to problems as discussed later in Chapter 10.

**strcpy/strncpy**
The `strcpy` command is probably the most dangerous command used in C. The format of the command is

```
strcpy(<destination>, <source>);
```
The purpose of the command is to copy each character in the source string (a series of characters ending with a null character: \0) into the destination string. This is particularly dangerous because there is no checking of the source’s size before it is copied over to the destination. In reality, we are talking about overwriting memory locations here, something which will be explained later in this chapter. Suffice it to say, when the source is larger than the space allocated for the destination, bad things happen (buffer overflows). A much safer command is the `strncpy` command. The format of that command is

`strncpy(<destination>, <source>, <width>);`

The width field is used to ensure that only a certain number of characters are copied from the source string to the destination string, allowing for greater control by the programmer.

---

**CAUTION** Using unbounded functions like `strcpy` is unsafe; however, most programming courses do not cover the dangers posed by these functions. In fact, if programmers would simply use the safer alternatives—for example, `strncpy`—then the entire class of buffer overflow attacks would be less prevalent. Obviously, programmers continue to use these dangerous functions since buffer overflows are the most common attack vector. That said, even bounded functions can suffer from incorrect width calculations.

*for and while Loops*

Loops are used in programming languages to iterate through a series of commands multiple times. The two common types are *for* and *while* loops.

*for* loops start counting at a beginning value, test the value for some condition, execute the statement, and increment the value for the next iteration. The format is as follows:

```
for(<beginning value>; <test value>; <change value>){
    <statement>;
}
```

Therefore, a *for* loop like
for(i=0; i<10; i++){
    printf("%d", i);
}

will print the numbers 0 to 9 on the same line (since \n is not used), like this: 0123456789.

With for loops, the condition is checked prior to the iteration of the statements in the loop, so it is possible that even the first iteration will not be executed. When the condition is not met, the flow of the program continues after the loop.

---

**NOTE** It is important to note the use of the less-than operator (<) in place of the less-than-or-equal-to operator (<=), which allows the loop to proceed one more time until i=10. This is an important concept that can lead to off-by-one errors. Also, note the count was started with 0. This is common in C and worth getting used to.

The while loop is used to iterate through a series of statements until a condition is met. The format is as follows:

```c
while(<conditional test>){
    <statement>;
}
```

Loops may also be nested within each other.

### if/else

The if/else construct is used to execute a series of statements if a certain condition is met; otherwise, the optional else block of statements is executed. If there is no else block of statements, the flow of the program will continue after the end of the closing if block bracket (}). The format is as follows:

```c
if(<condition>) {
    <statements to execute if condition is met>
} else{
    <statements to execute if the condition above is false>;
}
```

The braces may be omitted for single statements.
Comments
To assist in the readability and sharing of source code, programmers include comments in the code. There are two ways to place comments in code: //, or /* and */. The // indicates that any characters on the rest of that line are to be treated as comments and not acted on by the computer when the program executes. The /* and */ pair starts and stops a block of comments that may span multiple lines. The /* is used to start the comment, and the */ is used to indicate the end of the comment block.

Sample Program
You are now ready to review your first program. We will start by showing the program with // comments included, and will follow up with a discussion of the program:

```c
// hello.c // customary comment of program name
#include <stdio.h> // needed for screen printing
main ( ) { // required main function
    printf("Hello haxor"); // simply say hello
} // exit program
```

This very simple program prints “Hello haxor” to the screen using the printf function, included in the stdio.h library.

Now for one that’s a little more complex:

```c
// meet.c // needed for screen printing
#include <stdio.h> // greeting function to say hello
int greeting(char *temp1, char *temp2){ // string variable to hold the name
    char name[400]; // copy the function argument to name
    strcpy(name, temp2);
    printf("Hello %s %s\n", temp1, name); // print out the greeting
}

main(int argc, char * argv[]){ // note the format for arguments
    greeting(argv[1], argv[2]); // call function, pass title & name
    printf("Bye %s %s\n", argv[1], argv[2]); // say "bye"
} // exit program
```

This program takes two command-line arguments and calls the greeting() function, which prints “Hello” and the name given and a carriage return. When the greeting() function finishes, control is returned to main(), which prints out “Bye” and the name given. Finally, the program exits.

Compiling with gcc
Compiling is the process of turning human-readable source code into machine-readable binary files that can be digested by the computer and executed. More
specifically, a compiler takes source code and translates it into an intermediate set of files called *object code*. These files are nearly ready to execute but may contain unresolved references to symbols and functions not included in the original source code file. These symbols and references are resolved through a process called *linking*, as each object file is linked together into an executable binary file. We have simplified the process for you here.

When programming with C on Unix systems, the compiler of choice is GNU C Compiler (*gcc*). *gcc* offers plenty of options when compiling. The most commonly used flags are listed and described in Table 2-3.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-o &lt;filename&gt;</td>
<td>Saves the compiled binary with this name. The default is to save the output as a.out.</td>
</tr>
<tr>
<td>-S</td>
<td>Produces a file containing assembly instructions; saved with a .s extension.</td>
</tr>
<tr>
<td>-ggdb</td>
<td>Produces extra debugging information; useful when using GNU debugger (<em>gdb</em>).</td>
</tr>
<tr>
<td>-c</td>
<td>Compiles without linking; produces object files with a .o extension.</td>
</tr>
<tr>
<td>-mpreferred-stack-boundary=2</td>
<td>Compiles the program using a DWORD size stack, simplifying the debugging process while you learn.</td>
</tr>
<tr>
<td>-fno-stack-protector</td>
<td>Disables the stack protection; introduced with GCC 4.1. This option is useful when learning about buffer overflows, such as in Chapter 10.</td>
</tr>
<tr>
<td>-z execstack</td>
<td>Enables an executable stack, which was disabled by default in GCC 4.1. This option is useful when learning about buffer overflows, such as in Chapter 10.</td>
</tr>
</tbody>
</table>

**Table 2-3** Commonly Used *gcc* Flags

For example, to compile our meet.c program, you type

```
$gcc -o meet meet.c
```

Then, to execute the new program, you type

```
./meet Mr Haxor
Hello Mr Haxor
Bye Mr Haxor
$
```

**Computer Memory**

In the simplest terms, *computer memory* is an electronic mechanism that has the ability to store and retrieve data. The smallest amount of data that can be stored is 1
A bit, which can be represented by either a 1 or a 0 in memory. When you put 4 bits together, it is called a nibble, which can represent values from 0000 to –1111. There are exactly 16 binary values, ranging from 0 to 15, in decimal format. When you put two nibbles, or 8 bits, together, you get a byte, which can represent values from 0 to \((2^8 - 1)\), or 0 to 255 in decimal. When you put 2 bytes together, you get a word, which can represent values from 0 to \((2^{16} - 1)\), or 0 to 65,535 in decimal. Continuing to piece data together, if you put two words together, you get a double word, or DWORD, which can represent values from 0 to \((2^{32} - 1)\), or 0 to 4,294,967,295 in decimal.

There are many types of computer memory; we will focus on random access memory (RAM) and registers. Registers are special forms of memory embedded within processors, which will be discussed later in this chapter in the “Registers” section.

**Random Access Memory (RAM)**

In RAM, any piece of stored data can be retrieved at any time—thus, the term random access. However, RAM is volatile, meaning that when the computer is turned off, all data is lost from RAM. When discussing modern Intel-based products (x86), the memory is 32-bit addressable, meaning that the address bus the processor uses to select a particular memory address is 32 bits wide. Therefore, the most memory that can be addressed in an x86 processor is 4,294,967,295 bytes.

**Endian**

In his 1980 Internet Experiment Note (IEN) 137, “On Holy Wars and a Plea for Peace,” Danny Cohen summarized Swift’s *Gulliver’s Travels*, in part, as follows in his discussion of byte order:

Gulliver finds out that there is a law, proclaimed by the grandfather of the present ruler, requiring all citizens of Lilliput to break their eggs only at the little ends. Of course, all those citizens who broke their eggs at the big ends were angered by the proclamation. Civil war broke out between the Little-Endians and the Big-Endians, resulting in the Big-Endians taking refuge on a nearby island, the kingdom of Blefuscu.¹

The point of Cohen’s paper was to describe the two schools of thought when writing data into memory. Some feel that the low-order bytes should be written first (called “Little-Endians” by Cohen), whereas others think the high-order bytes should be written first (called “Big-Endians”). The difference really depends on the hardware
you are using. For example, Intel-based processors use the little-endian method, whereas Motorola-based processors use big-endian. This will come into play later as we talk about shellcode in Chapters 6 and 7.

Segmentation of Memory
The subject of segmentation could easily consume a chapter itself. However, the basic concept is simple. Each process (oversimplified as an executing program) needs to have access to its own areas in memory. After all, you would not want one process overwriting another process’s data. So memory is broken down into small segments and handed out to processes as needed. Registers, discussed later in the chapter, are used to store and keep track of the current segments a process maintains. Offset registers are used to keep track of where in the segment the critical pieces of data are kept.

Programs in Memory
When processes are loaded into memory, they are basically broken into many small sections. There are six main sections that we are concerned with, and we’ll discuss them in the following sections.

.text Section
The .text section basically corresponds to the .text portion of the binary executable file. It contains the machine instructions to get the task done. This section is marked as read-only and will cause a segmentation fault if written to. The size is fixed at runtime when the process is first loaded.

.data Section
The .data section is used to store global initialized variables, such as

```c
int a = 0;
```

The size of this section is fixed at runtime.

.bss Section
The below stack section (.bss) is used to store global noninitialized variables, such as

```c
int a;
```

The size of this section is fixed at runtime.

Heap Section
The heap section is used to store dynamically allocated variables and grows from the lower-addressed memory to the higher-addressed memory. The allocation of
memory is controlled through the `malloc()` and `free()` functions. For example, to declare an integer and have the memory allocated at runtime, you would use something like

```c
int i = malloc (sizeof (int)); // dynamically allocates an integer, contains
// the preexisting value of that memory
```

**Stack Section**

The *stack* section is used to keep track of function calls (recursively) and grows from the higher-addressed memory to the lower-addressed memory on most systems. As we will see, the fact that the stack grows in this manner allows the subject of buffer overflows to exist. Local variables exist in the stack section.

**Environment/Arguments Section**

The *environment/arguments* section is used to store a copy of system-level variables that may be required by the process during runtime. For example, among other things, the path, shell name, and hostname are made available to the running process. This section is writable, allowing its use in format string and buffer overflow exploits. Additionally, the command-line arguments are stored in this area. The sections of memory reside in the order presented. The memory space of a process looks like this:

![Memory Space Diagram](image)

**Buffers**

The term *buffer* refers to a storage place used to receive and hold data until it can be handled by a process. Since each process can have its own set of buffers, it is critical to keep them straight; this is done by allocating the memory within the .data or .bss section of the process’s memory. Remember, once allocated, the buffer is of fixed length. The buffer may hold any predefined type of data; however, for our purpose, we will focus on string-based buffers, which are used to store user input and variables.

**Strings in Memory**

Simply put, strings are just continuous arrays of character data in memory. The string is referenced in memory by the address of the first character. The string is terminated or ended by a null character (`\0` in C).
Pointers
Pointers are special pieces of memory that hold the address of other pieces of memory. Moving data around inside of memory is a relatively slow operation. It turns out that instead of moving data, keeping track of the location of items in memory through pointers and simply changing the pointers is much easier. Pointers are saved in 4 bytes of contiguous memory because memory addresses are 32 bits in length (4 bytes). For example, as mentioned, strings are referenced by the address of the first character in the array. That address value is called a pointer. So the variable declaration of a string in C is written as follows:

```c
char * str; // this is read, give me 4 bytes called str which is a pointer
            // to a Character variable (the first byte of the array).
```

Note that even though the size of the pointer is set at 4 bytes, the size of the string has not been set with the preceding command; therefore, this data is considered uninitialized and will be placed in the .bss section of the process memory.

Here is another example; if you wanted to store a pointer to an integer in memory, you would issue the following command in your C program:

```c
int * point1; // this is read, give me 4 bytes called point1, which is a
               // pointer to an integer variable.
```

To read the value of the memory address pointed to by the pointer, you dereference the pointer with the * symbol. Therefore, if you wanted to print the value of the integer pointed to by `point1` in the preceding code, you would use the following command:

```c
printf("%d", *point1);
```

where the * is used to dereference the pointer called `point1` and display the value of the integer using the `printf()` function.

Putting the Pieces of Memory Together
Now that you have the basics down, we will present a simple example to illustrate the use of memory in a program:
This program does not do much. First, several pieces of memory are allocated in different sections of the process memory. When main is executed, funct1() is called with an argument of 1. Once funct1() is called, the argument is passed to the function variable called c. Next, memory is allocated on the heap for a 10-byte string called str. Finally, the 5-byte string “abcde” is copied into the new variable called str. The function ends, and then the main() program ends.