The Shellcoder's Handbook
Discovering and Exploiting Security Holes
Second Edition

Chris Anley
John Heasman
Felix "FX" Linder
Gerardo Richarte

The Shellcoder's Handbook: Discovering and Exploiting Security Holes
(1st Edition) was written by Jack Koziol, David Litchfield, Dave Aitel,
Chris Anley, Sinan Eren, Neel Mehta, and Riley Hassell.
Contents

About the Authors vii
Acknowledgments xi
Introduction to the Second Edition xxiii

Part I Introduction to Exploitation: Linux on x86

Chapter 1 Before You Begin 3
Basic Concepts 3
Memory Management 4
Assembly 6
Recognizing C and C++ Code Constructs in Assembly 7
Conclusion 10

Chapter 2 Stack Overflows 11
Buffers 12
The Stack 13
Functions and the Stack 15
Overflowing Buffers on the Stack 18
Controlling EIP 22
An Interesting Diversion 23
Using an Exploit to Get Root Privileges 25
The Address Problem 27
The NOP Method 33
Defeating a Non-Executable Stack 35
Return to libc 35
Conclusion 39
Chapter 3 Shellcode
Understanding System Calls 42
Writing Shellcode for the exit() Syscall 44
Injectable Shellcode 48
Spawning a Shell 50
Conclusion 59

Chapter 4 Introduction to Format String Bugs
Prerequisites 61
What Is a Format String? 61
What Is a Format String Bug? 63
Format String Exploits 68
  Crashing Services 69
  Information Leakage 70
Controlling Execution for Exploitation 75
Why Did This Happen? 84
Format String Technique Roundup 85
Conclusion 88

Chapter 5 Introduction to Heap Overflows
What Is a Heap? 90
  How a Heap Works 91
Finding Heap Overflows 91
  Basic Heap Overflows 93
  Intermediate Heap Overflows 98
    Advanced Heap Overflow Exploitation 105
Conclusion 107

Part II Other Platforms—Windows, Solaris, OS/X, and Cisco

Chapter 6 The Wild World of Windows
How Does Windows Differ from Linux? 111
  Win32 API and PE-COFF 112
Heaps 114
  Threading 115
The Genius and Idiocy of the Distributed Common Object Model and DCE-RPC 116
  Recon 118
  Exploitation 120
  Tokens and Impersonation 120
  Exception Handling under Win32 122
Debugging Windows 124
  Bugs in Win32 124
  Writing Windows Shellcode 125
  A Hacker’s Guide to the Win32 API 126
  A Windows Family Tree from the Hacker’s Perspective 126
Conclusion 127
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Windows Shellcode</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td>Syntax and Filters</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td>Setting Up</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td>Parsing the PEB</td>
<td>132</td>
</tr>
<tr>
<td></td>
<td>Heapoverflow.c Analysis</td>
<td>132</td>
</tr>
<tr>
<td></td>
<td>Searching with Windows Exception Handling</td>
<td>148</td>
</tr>
<tr>
<td></td>
<td>Popping a Shell</td>
<td>153</td>
</tr>
<tr>
<td></td>
<td>Why You Should Never Pop a Shell on Windows</td>
<td>153</td>
</tr>
<tr>
<td></td>
<td>Conclusion</td>
<td>154</td>
</tr>
<tr>
<td>8</td>
<td>Windows Overflows</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>Stack-Based Buffer Overflows</td>
<td>156</td>
</tr>
<tr>
<td></td>
<td>Frame-Based Exception Handlers</td>
<td>156</td>
</tr>
<tr>
<td></td>
<td>Abusing Frame-Based Exception Handling on Windows 2003 Server</td>
<td>161</td>
</tr>
<tr>
<td></td>
<td>A Final Note about Frame-Based Handler Overwrites</td>
<td>166</td>
</tr>
<tr>
<td></td>
<td>Stack Protection and Windows 2003 Server</td>
<td>166</td>
</tr>
<tr>
<td></td>
<td>Heap-Based Buffer Overflows</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td>The Process Heap</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td>Dynamic Heaps</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td>Working with the Heap</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td>How the Heap Works</td>
<td>174</td>
</tr>
<tr>
<td></td>
<td>Exploiting Heap-Based Overflows</td>
<td>178</td>
</tr>
<tr>
<td></td>
<td>Overwrite Pointer to RtlEnterCriticalSection in the PEB</td>
<td>178</td>
</tr>
<tr>
<td></td>
<td>Overwrite Pointer to Unhandled Exception Filter</td>
<td>185</td>
</tr>
<tr>
<td></td>
<td>Repairing the Heap</td>
<td>191</td>
</tr>
<tr>
<td></td>
<td>Other Aspects of Heap-Based Overflows</td>
<td>193</td>
</tr>
<tr>
<td></td>
<td>Wrapping Up the Heap</td>
<td>194</td>
</tr>
<tr>
<td></td>
<td>Other Overflows</td>
<td>194</td>
</tr>
<tr>
<td></td>
<td>.data Section Overflows</td>
<td>194</td>
</tr>
<tr>
<td></td>
<td>TEB/PEB Overflows</td>
<td>196</td>
</tr>
<tr>
<td></td>
<td>Exploiting Buffer Overflows and Non-Executable Stacks</td>
<td>197</td>
</tr>
<tr>
<td></td>
<td>Conclusion</td>
<td>203</td>
</tr>
<tr>
<td>9</td>
<td>Overcoming Filters</td>
<td>205</td>
</tr>
<tr>
<td></td>
<td>Writing Exploits for Use with an Alphanumeric Filter</td>
<td>205</td>
</tr>
<tr>
<td></td>
<td>Writing Exploits for Use with a Unicode Filter</td>
<td>209</td>
</tr>
<tr>
<td></td>
<td>What Is Unicode?</td>
<td>210</td>
</tr>
<tr>
<td></td>
<td>Converting from ASCII to Unicode</td>
<td>210</td>
</tr>
<tr>
<td></td>
<td>Exploiting Unicode-Based Vulnerabilities</td>
<td>211</td>
</tr>
<tr>
<td></td>
<td>The Available Instruction Set in Unicode Exploits</td>
<td>212</td>
</tr>
<tr>
<td></td>
<td>The Venetian Method</td>
<td>213</td>
</tr>
<tr>
<td></td>
<td>An ASCII Venetian Implementation</td>
<td>214</td>
</tr>
<tr>
<td></td>
<td>Decoder and Decoding</td>
<td>218</td>
</tr>
<tr>
<td></td>
<td>The Decoder Code</td>
<td>219</td>
</tr>
<tr>
<td></td>
<td>Getting a Fix on the Buffer Address</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>Conclusion</td>
<td>221</td>
</tr>
</tbody>
</table>
## Chapter 12 OS X Shellcode
- OS X Is Just BSD, Right? 314
- Is OS X Open Source? 314
- OS X for the Unix-aware 315
  - Password Cracking 316
- OS X PowerPC Shellcode 316
- OS X Intel Shellcode 324
  - Example Shellcode 326
  - ret2libc 327
  - ret2str(l)cpy 329
- OS X Cross-Platform Shellcode 332
- OS X Heap Exploitation 333
- Bug Hunting on OS X 335
- Some Interesting Bugs 335
- Essential Reading for OS X Exploits 337
- Conclusion 338

## Chapter 13 Cisco IOS Exploitation
- An Overview of Cisco IOS 339
  - Hardware Platforms 340
  - Software Packages 340
  - IOS System Architecture 343
- Vulnerabilities in Cisco IOS 346
  - Protocol Parsing Code 347
  - Services on the Router 347
  - Security Features 348
  - The Command-Line Interface 348
- Reverse Engineering IOS 349
  - Taking the Images Apart 349
  - Diffing IOS Images 350
  - Runtime Analysis 351
- Exploiting Cisco IOS 357
  - Stack Overflows 357
  - Heap Overflows 359
  - Shellcodes 364
- Conclusion 373

## Chapter 14 Protection Mechanisms
- Protections 375
  - Non-Executable Stack 376
  - W^X (Either Writable or Executable) Memory 381
  - Stack Data Protection 388
  - AAAS: ASCII Armored Address Space 394
  - ASLR: Address Space Layout Randomization 396
  - Heap Protections 399
  - Windows SEH Protections 407
  - Other Protections 411
### Implementation Differences
- **Windows**
- **Linux**
- **OpenBSD**
- **Mac OS X**
- **Solaris**
- **Conclusion**

### Part III  Vulnerability Discovery

#### Chapter 15  Establishing a Working Environment 429
- **What You Need for Reference**
- **What You Need for Code**
  - gcc
  - gdb
  - NASM
  - WinDbg
  - OllyDbg
  - Visual C++
  - Python
- **What You Need for Investigation**
  - Useful Custom Scripts/Tools
  - All Platforms
  - Unix
  - Windows
- **What You Need to Know**
  - Paper Archives
- **Optimizing Shellcode Development**
  - Plan the Exploit
  - Write the Shellcode in Inline Assembler
  - Maintain a Shellcode Library
  - Make It Continue Nicely
  - Make the Exploit Stable
  - Make It Steal the Connection
- **Conclusion**

#### Chapter 16  Fault Injection 445
- **Design Overview**
- **Input Generation**
- **Fault Injection**
- **Modification Engines**
- **Fault Delivery**
- **Nagel Algorithm**
- **Timing**
- **Heuristics**
- **Stateless versus State-Based Protocols**
- **Fault Monitoring**
  - Using a Debugger
  - FaultMon
## Contents

Beyond Recognition: A Real Vulnerability versus a Bug  501  
Conclusion  501

### Chapter 19 Instrumented Investigation: A Manual Approach  503

- Philosophy  503
- Oracle extproc Overflow  504
- Common Architectural Failures  508
  - Problems Happen at Boundaries  508
  - Problems Happen When Data Is Translated  509
  - Problems Cluster in Areas of Asymmetry  511
  - Problems Occur When Authentication and Authorization Are Confused  512
  - Problems Occur in the Dumbest Places  512
- Bypassing Input Validation and Attack Detection  513
  - Stripping Bad Data  513
  - Using Alternate Encodings  514
  - Using File-Handling Features  515
  - Evading Attack Signatures  517
  - Defeating Length Limitations  517
- Windows 2000 SNMP DOS  520
- Finding DOS Attacks  521
- SQL-UDP  522
- Conclusion  523

### Chapter 20 Tracing for Vulnerabilities  525

- Overview  526
  - A Vulnerable Program  527
  - Component Design  529
  - Building VulnTrace  538
  - Using VulnTrace  543
  - Advanced Techniques  546
- Conclusion  548

### Chapter 21 Binary Auditing: Hacking Closed Source Software  549

- Binary versus Source-Code Auditing: The Obvious Differences  550
- IDA Pro—The Tool of the Trade  550
  - Features: A Quick Crash Course  551
  - Debugging Symbols  552
- Binary Auditing Introduction  552
  - Stack Frames  552
  - Calling Conventions  554
  - Compiler-Generated Code  556
  - memcpy-Like Code Constructs  560
  - strlen-Like Code Constructs  560
  - C++ Code Constructs  561
  - The this Pointer  561
- Reconstructing Class Definitions  562
  - vtables  562
  - Quick but Useful Tidbits  563
Chapter 25  Unix Kernel Overflows  627
   Kernel Vulnerability Types  627
   Oday Kernel Vulnerabilities  636
      OpenBSD exec_ibcs2_coff_prep_zmagic() Stack Overflow  636
      The Vulnerability  638
   Solaris vfs_getvfssw() Loadable Kernel Module  642
      Traversal Vulnerability  644
      The sysfs() System Call  645
      The mount() System Call  645
   Conclusion  646

Chapter 26  Exploiting Unix Kernel Vulnerabilities  647
   The exec_ibcs2_coff_prep_zmagic() Vulnerability  647
      Calculating Offsets and Breakpoints  652
      Overwriting the Return Address and Redirecting Execution  654
      Locating the Process Descriptor (or the Proc Structure)  655
      Kernel Mode Payload Creation  658
      Returning Back from Kernel Payload  659
      Getting root (uid=0)  665
   Solaris vfs_getvfssw() Loadable Kernel Module Path Traversal Exploit  672
      Crafting the Exploit  673
      The Kernel Module to Load  674
      Getting root (uid=0)  678
   Conclusion  678

Chapter 27  Hacking the Windows Kernel  681
   Windows Kernel Mode Flaws—An Increasingly Hunted Species  681
   Introduction to the Windows Kernel  682
   Common Kernel-Mode Programming Flaws  683
      Stack Overflows  684
      Heap Overflows  688
      Insufficient Validation of User-Mode Addresses  688
      Repurposing Attacks  689
      Shared Object Attacks  689
   Windows System Calls  690
      Understanding System Calls  690
      Attacking System Calls  692
   Communicating with Device Drivers  693
      I/O Control Code Components  693
      Finding Flaws in IOCTL Handlers  694
   Kernel-Mode Payloads  695
      Elevating a User-Mode Process  696
      Running an Arbitrary User-Mode Payload  699
      Subverting Kernel Security  701
      Installing a Rootkit  703
   Essential Reading for Kernel Shellcoders  703
   Conclusion  704

Index  705