

CONTENTS IN DETAIL

ACKNOWLEDGMENTS	xix
------------------------	------------

INTRODUCTION	xxi
---------------------	------------

About This Book	xxii
Who This Book Is For	xxiii
The Approach	xxiv
What You'll Need	xxvi
What's in the Book	xxvi

PART I INTO THE MACHINE: AWAKENING CODE SPELLS

1 SPELL OF THE BINARY ORACLE: WHEN HARDWARE WHISPERS IN OPCODES	3
--	----------

A Computer as a Stack	3
From the Bottom Up	4
From the Top Down	5
The Simplest Computer in the World	5
The Memory	6
The CPU	7
A Complete Computer Model	11
How Programs Work	11
Writing Your First Program	13
Running Your First Program	14
Your First C Program	17
Assembling Your First Program for Different Processors	21
ARM	21
MIPS	23
RISC-V	25
Summary	26

2		
SPELL OF ESSENCE: DISTILLING C TO ITS PRIMORDIAL ASM		27
Your First Real ASM Program		28
Producing Your First Binary		28
Compiling and Linking the Program		29
Interacting with the OS Through System Calls		31
From C Down to ASM		33
Dissecting the C Code		34
Using <code>_start</code> vs. <code>main</code>		38
Generating Static vs. Dynamic Binaries		40
Getting Rid of <code>libc</code>		43
Avoiding an Executable Stack		45
Aren't Those Binaries Too Big?		48
A Closer Look at the Linker		55
Customizing Default Values with Linker Flags		55
Compiling and Linking C Programs Separately		57
Building ASM Programs Using <code>libc</code>		59
Handling Standard Libraries and Start Files		60
System Call Conventions Across Architectures		61
x86_64		62
ARM32		64
AArch64		65
MIPS32		66
MIPS64		68
RISC-V		69
Summary		71

3		
SPELL OF REBIRTH: TRANSMUTING GREETINGS INTO SHELLCODE		73
Representing Information		74
Bits, Bytes, Words, and Beyond		74
Processor Native Word Sizes		74
Decimal, Binary, Hexadecimal, and More		75
Negative Numbers		77
Text Strings		79
A Closer Look at Memory		81
A Simplified Hardware Memory Model		81
Native Word Size and Memory Alignment		85
Little vs. Big Endian		86
"Hello, World!" in ASM		87
Labels and Assembler Commands		88
Pointers		88
"Hello, World!" in C		90
Using the GNU Debugger		91
Declaring Pointers in C		92

Writing Your First Shellcode	94
x86_64	95
ARM32	97
A Short Digression on Addressing Modes	99
AArch64	100
MIPS32	101
MIPS64	102
RISC-V	104
Summary	106

4

SPELL OF THE OVERFLOWING FRAME: MASTERING STACK SECRETS

107

An Overview of Stacks	108
How Stacks Are Implemented	109
How Stacks Are Used	111
The Basics of Functions	111
Function Support in Assembly	112
Parameters and Return Values	113
The Stack Frame	115
Function Prologues and Epilogues	116
Do We Need the Stack Frame?	119
What Else Happens When You Leave a Function?	119
Local Variables	120
Declaration vs. Initialization	126
Function Parameters	128
Buffer Overflows	132
Creating a Vulnerable Program	133
Instrumenting the Binary	135
Triggering an Overflow	137
Writing Your First Exploit	140
ARM32 Functions	143
Simple Local Variables	144
Local Buffers and Canaries	145
AArch64 Functions	146
Simple Local Variables	147
Local Buffers and Canaries	148
MIPS Functions	149
Simple Local Variables	150
Local Buffers and Canaries	150
RISC-V Functions	152
Simple Local Variables	153
Local Buffers and Canaries	154
Summary	156

5

SPELL OF THE UNDEAD: WRITING A REPL TO ANIMATE A BOTNET 157

A REPL in C	158
Arrays	159
Loops	159
Conditionals	161
A REPL for x86_64	162
Your First Library	162
The REPL	164
Jump Instructions	166
Processor Flags	168
Buffer Initialization	169
String Instructions	172
A REPL for ARM	174
Jump Instructions	176
Flags and Conditions	177
Buffer Initialization	179
A REPL for AArch64	182
Jump Instructions	184
Buffer Initialization	185
A REPL for MIPS	187
Jump Instructions	189
Buffer Initialization	191
A REPL for RISC-V	192
Jump Instructions	194
Buffer Initialization	196
Creating a Minimal Botnet	196
The C2 REPL	197
The Zombie REPL	199
Deployment	200
Summary	202

PART II

ONTO THE NETWORK: UNVEILING MYSTIC GATEWAYS

6

VEIL OF SHADOWS: UNLOCKING HIDDEN BACKDOORS 205

How Backdoors Work	206
A Brief Networking Primer	206
Types of Network Connections	207
Sockets	207
The Network and Transport Layers	207
Writing a Minimal Backdoor in C	210
Creating a Socket	212
Manipulating File Descriptors	215

Launching a Shell	217
Covering Your Tracks	218
Detecting the Backdoor	218
Writing Your First x86_64 ASM Backdoor	220
Preparing the Backdoor for the Target System	222
Making the Backdoor Smaller	225
Adapting the Backdoor for Other Processors	230
ARM	230
MIPS	233
RISC-V	236
Summary	239

7 VEIL OF ASH: THE DROPPER THAT LEAVES NO FLAME BEHIND 241

Creating a Dropper Using Existing Tools	242
Writing a Dropper in C	243
Working with a Build System	245
Automating the Dropper Tasks	246
Avoiding Forensic Analysis	248
Writing to a RAM Disk	249
Writing to Memory with memfd_create	250
Running from Memory with fexecve	251
Adapting the Dropper to Assembly	255
x86_64	255
MIPS	258
ARM	260
RISC-V	267
A Real-World Dropper	269
Summary	271

PART III THROUGH THE NETWORK: CASTING CRAWLERS

8 CASTING THE TWIN SERPENT: SUMMONING A TWO-HEADED WORM 275

How Mobile Agent Systems (and Worms) Work	276
Implementing a Remote Shellcode Executor	278
Configuring the Server	280
Accepting Connections	281
Running a Shellcode	282
Testing the Shellcode Execution Service	284

Simulating a Target Network	286
Setting Up Test Machines	287
Creating a Virtual Network	289
Starting the Test Environment	290
Coding a File Transfer Server	292
Preparing the File for Transfer	293
Serving the File	295
Designing an Agent or Worm	296
Creating the Worm Payload	297
Getting a List of Target Machines	298
Connecting to the Target Machines	299
Writing a Two-Stage Worm	301
Deploying the Worm	303
Extending the Test Environment	304
Launching the Test Network	306
Starting the File Server	307
Creating the Initiator Machine	307
Running the Worm	308
Troubleshooting	309
How Worms Can Go Wrong	311
Packet Storms	311
Race Conditions	312
Summary	314

9 CASTING THE LONE STRAND: SINGLE-STAGE WORMCRAFT 315

From Two Stages to One	316
The x86_64 Worm	316
The Payload	316
The Vulnerable Machines	317
The Main Loop	318
The ARM Worm	321
The Payload	321
The Vulnerable Machines	321
The Main Loop	322
The MIPS Worm	324
The Payload	324
The Vulnerable Machines	329
The Main Loop	331
The RISC-V Worm	332
The Payload	332
The Vulnerable Machines	333
The Main Loop	334
Summary	335

10 CASTING THE LOST HOSTS: CONJURING NETWORK SCANNERS 337

Why We Need a Network Scanner	338
Writing a Port Scanner in C	338
Getting the Current IP Address and Netmask	340
Calculating the Scope of the Scan	345
Performing the Scan	346
Speeding Up the Scanner	347
Blocking vs. Non-Blocking I/O	348
A Non-Blocking Scanning Loop	349
Adapting the Port Scanner for x86_64	351
Getting the Current IP Address and Netmask	351
Calculating the Scope of the Scan	353
Performing the Scan	355
Adapting the Port Scanner for Other Platforms	358
ARM	358
MIPS	360
RISC-V	361
Summary	366

APPENDIXES: THE FORGE

A THE SORCERER'S FORGE: PREPARING YOUR WORKSHOP 371

Setting Up Your Development Environment	371
Updating Your Environment	372
Running Binaries	373

B SECRETS OF THE FORGE: ARCANE TRICKS FOR LOW-LEVEL TOOLS 375

GNU Assembler Tricks	375
Intel Syntax	375
The lea Instruction on x86_64	376
Delay Slots	376
GNU Debugger Tricks	377
Configuring the Debugger for Individual Projects	377
Customizing the Disassembler	377
Changing the Default Syntax	378
Dumping Data	378
Tracking Forks	378
Debugging Programs with qemu	379

Compilation Tricks	380
RISC-V Extensions	380
Useful gcc Flags	381
Function Prototypes and Warnings	383
objdump Tricks	384
Dumping Code in Intel Syntax	384
Showing Real Code	384
objcopy Tricks	384
Manipulating ELF Sections	384
Manipulating Symbols	385
Appending Files	386
Running Binaries with binfmt_misc	387
Producing Minimal ELF Binaries	388
Handcrafting ELF's with nasm	388
Handcrafting ELF's with the GNU Assembler	391
Symbol Resolution and Relocation	394
Quick Reference Tables	396

C

THE SORCERER'S ANVIL: FULL SPELLCRAFT LISTINGS 399

The Complete x86_64 Worm Source Code	399
The Complete ARM Worm Source Code	403
The Complete MIPS Worm Source Code	407
The Complete RISC-V Worm Source Code	410

INDEX

415