

"The most useful technical security book I've read this year. A must-have for all who protect systems from malicious software."

-**Lenny Zeltser**, Security Practice Director at Savvis and Senior Faculty Member at SANS Institute

"The ultimate guide for anyone interested in malware analysis."

-**Ryan Olson**, Director, VeriSign iDefense Rapid Response Team

"Every page is filled with practical malware knowledge, innovative ideas, and useful tools. Worth its weight in gold!"

-**Aaron Walters**, Lead Developer of Volatility and VP of Security R&D at Terremark

# Malware Analyst's Cookbook and DVD

TOOLS AND TECHNIQUES FOR FIGHTING MALICIOUS CODE

Michael Hale Ligh, Steven Adair, Blake Hartstein, and Matthew Richard



# **Malware Analyst's Cookbook and DVD**



# Malware Analyst's Cookbook and DVD

Tools and Techniques for  
Fighting Malicious Code

Michael Hale Ligh  
Steven Adair  
Blake Hartstein  
Matthew Richard



WILEY

Wiley Publishing, Inc.

## Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code

Published by  
Wiley Publishing, Inc.  
10475 Crosspoint Boulevard  
Indianapolis, IN 46256  
[www.wiley.com](http://www.wiley.com)

Copyright © 2011 by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-0-470-61303-0

ISBN: 978-1-118-00336-7 (ebk)

ISBN: 978-1-118-00829-4 (ebk)

ISBN: 978-1-118-00830-0 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

**Limit of Liability/Disclaimer of Warranty:** The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or website is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or website may provide or recommendations it may make. Further, readers should be aware that Internet websites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

**Library of Congress Control Number:** 2010933462

**Trademarks:** Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. Wiley Publishing, Inc. is not associated with any product or vendor mentioned in this book.

*To my family for helping me shape my life and to my wife  
Suzanne for always giving me something to look forward to.*

*—Michael Hale Ligh*

*To my new wife and love of my life Irene and my family.  
Without your support over the many years, I would not be where  
I am or who I am today.*

*—Steven Adair*

# Credits

**Executive Editor**

Carol Long

**Project Editor**

Maureen Spears

**Technical Editor**

Michael Gregg

**Production Editor**

Kathleen Wisor

**Copy Editor**

Nancy Rappaport

**Editorial Director**

Robyn B. Siesky

**Editorial Manager**

Mary Beth Wakefield

**Freelance Editorial Manager**

Rosemarie Graham

**Marketing Manager**

Ashley Zurcher

**Production Manager**

Tim Tate

**Vice President and****Executive Group Publisher**

Richard Swadley

**Vice President and Executive Publisher**

Barry Pruett

**Associate Publisher**

Jim Minatel

**Project Coordinator, Cover**

Lynsey Stanford

**Compositor**

Maureen Forys,

Happenstance Type-O-Rama

**Proofreader**

Word One New York

**Indexer**

Robert Swanson

**Cover Image**

Digital Vision/Getty Images

**Cover Designer**

Ryan Sneed

# About the Authors

**Michael Hale Ligh** is a Malicious Code Analyst at Verisign iDefense, where he specializes in developing tools to detect, decrypt, and investigate malware. In the past few years, he has taught malware analysis courses and trained hundreds of students in Rio De Janeiro, Shanghai, Kuala Lumpur, London, Washington D.C., and New York City. Before iDefense, Michael worked as a vulnerability researcher, providing ethical hacking services to one of the nation's largest healthcare providers. Due to this position, he gained a strong background in reverse-engineering and operating system internals. Before that, Michael defended networks and performed forensic investigations for financial institutions throughout New England. He is currently Chief of Special Projects at MNIN Security LLC.

**Steven Adair** is a security researcher with The Shadowserver Foundation and a Principal Architect at eTouch Federal Systems. At Shadowserver, Steven analyzes malware, tracks botnets, and investigates cyber-attacks of all kinds with an emphasis on those linked to cyber-espionage. Steven frequently presents on these topics at international conferences and co-authored the paper "Shadows in the Cloud: Investigating Cyber Espionage 2.0." In his day job, he leads the Cyber Threat operations for a Federal Agency, proactively detecting, mitigating and preventing cyber-intrusions. He has successfully implemented enterprise-wide anti-malware solutions across global networks by marrying best practices with new and innovative techniques. Steven is knee deep in malware daily, whether it be supporting his company's customer or spending his free time with Shadowserver.

**Blake Hartstein** is a Rapid Response Engineer at Verisign iDefense. He is responsible for analyzing and reporting on suspicious activity and malware. He is the author of the Jsunpack tool that aims to automatically analyze and detect web-based exploits, which he presented at Shmoocon 2009 and 2010. Blake has also authored and contributed Snort rules to the Emerging Threats project.

**Matthew Richard** is Malicious Code Operations Lead at Raytheon Corporation, where he is responsible for analyzing and reporting on malicious code. Matthew was previously Director of Rapid Response at iDefense. For 7 years before that, Matthew created and ran a managed security service used by 130 banks and credit unions. In addition, he has done independent forensic consulting for a number of national and global companies. Matthew currently holds the CISSP, GCIA, GCFA, and GREM certifications.

# Acknowledgments

**M**ichael would like to thank his current and past employers for providing an environment that encourages and stimulates creativity. He would like to thank his coworkers and everyone who has shared knowledge in the past. In particular, Aaron Walters and Ryan Smith for never hesitating to engage and debate interesting new ideas and techniques. A special thanks goes out to the guys who took time out of the busy days to review our book: Lenny Zeltser, Tyler Hudak, and Ryan Olson.

Steven would like to extend his gratitude to those who spend countless hours behind the scenes investigating malware and fighting cyber-crime. He would also like to thank his fellow members of the Shadowserver Foundation for their hard work and dedication towards making the Internet a safer place for us all.

We would also like to thank the following:

- Maureen Spears and Carol A. Long from Wiley Publishing, for helping us get through our first book.
- Ilfak Guilfanov (and the team at Hex-Rays) and Halvar Flake (and the team at Zynamics) for allowing us to use some of their really neat tools.
- All the developers of the tools that we referenced throughout the book. In particular, Frank Boldewin, Mario Vilas, Harlan Carvey, and Jesse Kornblum, who also helped review some recipes in their realm of expertise.
- The authors of other books, blogs, and websites that contribute to the collective knowledge of the community.

—Michael, Steven, Blake, and Matthew

# Contents

Introduction .....	xv
On The Book's DVD .....	xxiii
<b>1</b> Anonymizing Your Activities .....	1
<i>Recipe 1-1: Anonymous Web Browsing with Tor.</i> .....	3
<i>Recipe 1-2: Wrapping Wget and Network Clients with Torsocks</i> .....	5
<i>Recipe 1-3: Multi-platform Tor-enabled Downloader in Python</i> .....	7
<i>Recipe 1-4: Forwarding Traffic through Open Proxies</i> .....	12
<i>Recipe 1-5: Using SSH Tunnels to Proxy Connections</i> .....	16
<i>Recipe 1-6: Privacy-enhanced Web browsing with Privoxy</i> .....	18
<i>Recipe 1-7: Anonymous Surfing with Anonymouse.org.</i> .....	20
<i>Recipe 1-8: Internet Access through Cellular Networks</i> .....	21
<i>Recipe 1-9: Using VPNs with Anonymizer Universal</i> .....	23
<b>2</b> Honey pots .....	27
<i>Recipe 2-1: Collecting Malware Samples with Nepenthes.</i> .....	29
<i>Recipe 2-2: Real-Time Attack Monitoring with IRC Logging</i> .....	32
<i>Recipe 2-3: Accepting Nepenthes Submissions over HTTP with Python.</i> .....	34
<i>Recipe 2-4: Collecting Malware Samples with Dionaea</i> .....	37
<i>Recipe 2-5: Accepting Dionaea Submissions over HTTP with Python</i> .....	40
<i>Recipe 2-6: Real-time Event Notification and Binary Sharing with XMPP</i> .....	41
<i>Recipe 2-7: Analyzing and Replaying Attacks Logged by Dionea.</i> .....	43
<i>Recipe 2-8: Passive Identification of Remote Systems with p0f.</i> .....	44
<i>Recipe 2-9: Graphing Dionaea Attack Patterns with SQLite and Gnuplot</i> .....	46
<b>3</b> Malware Classification .....	51
<i>Recipe 3-1: Examining Existing ClamAV Signatures</i> .....	52
<i>Recipe 3-2: Creating a Custom ClamAV Database.</i> .....	54
<i>Recipe 3-3: Converting ClamAV Signatures to YARA.</i> .....	59
<i>Recipe 3-4: Identifying Packers with YARA and PEiD.</i> .....	61
<i>Recipe 3-5: Detecting Malware Capabilities with YARA</i> .....	63
<i>Recipe 3-6: File Type Identification and Hashing in Python.</i> .....	68
<i>Recipe 3-7: Writing a Multiple-AV Scanner in Python</i> .....	70

Recipe 3-8: Detecting Malicious PE Files in Python . . . . .	75
Recipe 3-9: Finding Similar Malware with ssdeep . . . . .	79
Recipe 3-10: Detecting Self-modifying Code with ssdeep . . . . .	82
Recipe 3-11: Comparing Binaries with IDA and BinDiff . . . . .	83
<b>4 Sandboxes and Multi-AV Scanners . . . . .</b>	<b>89</b>
Recipe 4-1: Scanning Files with VirusTotal . . . . .	90
Recipe 4-2: Scanning Files with Jotti . . . . .	92
Recipe 4-3: Scanning Files with NoVirusThanks . . . . .	93
Recipe 4-4: Database-Enabled Multi-AV Uploader in Python . . . . .	96
Recipe 4-5: Analyzing Malware with ThreatExpert . . . . .	100
Recipe 4-6: Analyzing Malware with CWSandbox . . . . .	102
Recipe 4-7: Analyzing Malware with Anubis . . . . .	104
Recipe 4-8: Writing AutoIT Scripts for Joebox . . . . .	105
Recipe 4-9: Defeating Path-dependent Malware with Joebox . . . . .	107
Recipe 4-10: Defeating Process-dependent DLLs with Joebox . . . . .	109
Recipe 4-11: Setting an Active HTTP Proxy with Joebox . . . . .	111
Recipe 4-12: Scanning for Artifacts with Sandbox Results . . . . .	112
<b>5 Researching Domains and IP Addresses . . . . .</b>	<b>119</b>
Recipe 5-1: Researching Domains with WHOIS . . . . .	120
Recipe 5-2: Resolving DNS Hostnames . . . . .	125
Recipe 5-3: Obtaining IP WHOIS Records . . . . .	129
Recipe 5-4: Querying Passive DNS with BFK . . . . .	132
Recipe 5-5: Checking DNS Records with Robtex . . . . .	133
Recipe 5-6: Performing a Reverse IP Search with DomainTools . . . . .	134
Recipe 5-7: Initiating Zone Transfers with dig . . . . .	135
Recipe 5-8: Brute-forcing Subdomains with dnsmap . . . . .	137
Recipe 5-9: Mapping IP Addresses to ASNs via Shadowserver . . . . .	138
Recipe 5-10: Checking IP Reputation with RBLs . . . . .	140
Recipe 5-11: Detecting Fast Flux with Passive DNS and TTLs . . . . .	143
Recipe 5-12: Tracking Fast Flux Domains . . . . .	146
Recipe 5-13: Static Maps with Maxmind, matplotlib, and pygeoip . . . . .	148
Recipe 5-14: Interactive Maps with Google Charts API . . . . .	152
<b>6 Documents, Shellcode, and URLs . . . . .</b>	<b>155</b>
Recipe 6-1: Analyzing JavaScript with Spidermonkey . . . . .	156
Recipe 6-2: Automatically Decoding JavaScript with Jsunpack . . . . .	159
Recipe 6-3: Optimizing Jsunpack-n Decodings for Speed and Completeness . . . . .	162
Recipe 6-4: Triggering exploits by Emulating Browser DOM Elements . . . . .	163

Recipe 6-5: Extracting JavaScript from PDF Files with pdf.py . . . . .	168
Recipe 6-6: Triggering Exploits by Faking PDF Software Versions . . . . .	172
Recipe 6-7: Leveraging Didier Stevens's PDF Tools . . . . .	175
Recipe 6-8: Determining which Vulnerabilities a PDF File Exploits . . . . .	178
Recipe 6-9: Disassembling Shellcode with DiStorm . . . . .	185
Recipe 6-10: Emulating Shellcode with Libemu . . . . .	190
Recipe 6-11: Analyzing Microsoft Office Files with OfficeMalScanner . . . . .	193
Recipe 6-12: Debugging Office Shellcode with DisView and MalHost-setup . . . . .	200
Recipe 6-13: Extracting HTTP Files from Packet Captures with Jsunpack . . . . .	204
Recipe 6-14: Graphing URL Relationships with Jsunpack . . . . .	206
<b>7 Malware Labs . . . . .</b>	<b>211</b>
Recipe 7-1: Routing TCP/IP Connections in Your Lab . . . . .	215
Recipe 7-2: Capturing and Analyzing Network Traffic . . . . .	217
Recipe 7-3: Simulating the Internet with INetSim . . . . .	221
Recipe 7-4: Manipulating HTTP/HTTPS with Burp Suite . . . . .	225
Recipe 7-5: Using Joe Stewart's Truman . . . . .	228
Recipe 7-6: Preserving Physical Systems with Deep Freeze . . . . .	229
Recipe 7-7: Cloning and Imaging Disks with FOG . . . . .	232
Recipe 7-8: Automating FOG Tasks with the MySQL Database . . . . .	236
<b>8 Automation . . . . .</b>	<b>239</b>
Recipe 8-1: Automated Malware Analysis with VirtualBox . . . . .	242
Recipe 8-2: Working with VirtualBox Disk and Memory Images . . . . .	248
Recipe 8-3: Automated Malware Analysis with VMware . . . . .	250
Recipe 8-4: Capturing Packets with TShark via Python . . . . .	254
Recipe 8-5: Collecting Network Logs with INetSim via Python . . . . .	256
Recipe 8-6: Analyzing Memory Dumps with Volatility . . . . .	258
Recipe 8-7: Putting all the Sandbox Pieces Together . . . . .	260
Recipe 8-8: Automated Analysis with ZeroWine and QEMU . . . . .	271
Recipe 8-9: Automated Analysis with Sandboxie and Buster . . . . .	276
<b>9 Dynamic Analysis . . . . .</b>	<b>283</b>
Recipe 9-1: Logging API calls with Process Monitor . . . . .	286
Recipe 9-2: Change Detection with Regshot . . . . .	288
Recipe 9-3: Receiving File System Change Notifications . . . . .	290
Recipe 9-4: Receiving Registry Change Notifications . . . . .	294
Recipe 9-5: Handle Table Diffing . . . . .	295
Recipe 9-6: Exploring Code Injection with HandleDiff . . . . .	300
Recipe 9-7: Watching Bankpatch.C Disable Windows File Protection . . . . .	301

Recipe 9-8: Building an API Monitor with Microsoft Detours . . . . .	304
Recipe 9-9: Following Child Processes with Your API Monitor . . . . .	311
Recipe 9-10: Capturing Process, Thread, and Image Load Events . . . . .	314
Recipe 9-11: Preventing Processes from Terminating . . . . .	321
Recipe 9-12: Preventing Malware from Deleting Files . . . . .	324
Recipe 9-13: Preventing Drivers from Loading . . . . .	325
Recipe 9-14: Using the Data Preservation Module . . . . .	327
Recipe 9-15: Creating a Custom Command Shell with ReactOS . . . . .	330
<b>10 Malware Forensics . . . . .</b>	<b>337</b>
Recipe 10-1: Discovering Alternate Data Streams with TSK . . . . .	337
Recipe 10-2: Detecting Hidden Files and Directories with TSK . . . . .	341
Recipe 10-3: Finding Hidden Registry Data with Microsoft's Offline API . . . . .	349
Recipe 10-4: Bypassing Poison Ivy's Locked Files . . . . .	355
Recipe 10-5: Bypassing Conficker's File System ACL Restrictions . . . . .	359
Recipe 10-6: Scanning for Rootkits with GMER . . . . .	363
Recipe 10-7: Detecting HTML Injection by Inspecting IE's DOM . . . . .	367
Recipe 10-8: Registry Forensics with RegRipper Plug-ins . . . . .	377
Recipe 10-9: Detecting Rogue-Installed PKI Certificates . . . . .	384
Recipe 10-10: Examining Malware that Leaks Data into the Registry . . . . .	388
<b>11 Debugging Malware . . . . .</b>	<b>395</b>
Recipe 11-1: Opening and Attaching to Processes . . . . .	396
Recipe 11-2: Configuring a JIT Debugger for Shellcode Analysis . . . . .	398
Recipe 11-3: Getting Familiar with the Debugger GUI . . . . .	400
Recipe 11-4: Exploring Process Memory and Resources . . . . .	407
Recipe 11-5: Controlling Program Execution . . . . .	410
Recipe 11-6: Setting and Catching Breakpoints . . . . .	412
Recipe 11-7: Using Conditional Log Breakpoints . . . . .	415
Recipe 11-8: Debugging with Python Scripts and PyCommands . . . . .	418
Recipe 11-9: Detecting Shellcode in Binary Files . . . . .	421
Recipe 11-10: Investigating Silentbanker's API Hooks . . . . .	426
Recipe 11-11: Manipulating Process Memory with WinAppDbg Tools . . . . .	431
Recipe 11-12: Designing a Python API Monitor with WinAppDbg . . . . .	433
<b>12 De-Obfuscation . . . . .</b>	<b>441</b>
Recipe 12-1: Reversing XOR Algorithms in Python . . . . .	441
Recipe 12-2: Detecting XOR Encoded Data with yaratize . . . . .	446
Recipe 12-3: Decoding Base64 with Special Alphabets . . . . .	448
Recipe 12-4: Isolating Encrypted Data in Packet Captures . . . . .	452

Recipe 12-5: Finding Crypto with SnD Reverser Tool, FindCrypt, and Kanal . . . . .	454
Recipe 12-6: Porting OpenSSL Symbols with Zynamics BinDiff . . . . .	456
Recipe 12-7: Decrypting Data in Python with PyCrypto . . . . .	458
Recipe 12-8: Finding OEP in Packed Malware . . . . .	461
Recipe 12-9: Dumping Process Memory with LordPE . . . . .	465
Recipe 12-10: Rebuilding Import Tables with ImpREC . . . . .	467
Recipe 12-11: Cracking Domain Generation Algorithms . . . . .	476
Recipe 12-12: Decoding Strings with x86emu and Python . . . . .	481
<b>13 Working with DLLs . . . . .</b>	<b>487</b>
Recipe 13-1: Enumerating DLL Exports . . . . .	488
Recipe 13-2: Executing DLLs with rundll32.exe . . . . .	491
Recipe 13-3: Bypassing Host Process Restrictions . . . . .	493
Recipe 13-4: Calling DLL Exports Remotely with rundll32ex . . . . .	495
Recipe 13-5: Debugging DLLs with LOADDLL.EXE . . . . .	499
Recipe 13-6: Catching Breakpoints on DLL Entry Points . . . . .	501
Recipe 13-7: Executing DLLs as a Windows Service . . . . .	502
Recipe 13-8: Converting DLLs to Standalone Executables . . . . .	507
<b>14 Kernel Debugging . . . . .</b>	<b>511</b>
Recipe 14-1: Local Debugging with LiveKd . . . . .	513
Recipe 14-2: Enabling the Kernel's Debug Boot Switch . . . . .	514
Recipe 14-3: Debug a VMware Workstation Guest (on Windows) . . . . .	517
Recipe 14-4: Debug a Parallels Guest (on Mac OS X) . . . . .	519
Recipe 14-5: Introduction to WinDbg Commands And Controls . . . . .	521
Recipe 14-6: Exploring Processes and Process Contexts . . . . .	528
Recipe 14-7: Exploring Kernel Memory . . . . .	534
Recipe 14-8: Catching Breakpoints on Driver Load . . . . .	540
Recipe 14-9: Unpacking Drivers to OEP . . . . .	548
Recipe 14-10: Dumping and Rebuilding Drivers . . . . .	555
Recipe 14-11: Detecting Rootkits with WinDbg Scripts . . . . .	561
Recipe 14-12: Kernel Debugging with IDA Pro . . . . .	566
<b>15 Memory Forensics with Volatility . . . . .</b>	<b>571</b>
Recipe 15-1: Dumping Memory with MoonSols Windows Memory Toolkit . . . . .	572
Recipe 15-2: Remote, Read-only Memory Acquisition with F-Response . . . . .	575
Recipe 15-3: Accessing Virtual Machine Memory Files . . . . .	576
Recipe 15-4: Volatility in a Nutshell . . . . .	578
Recipe 15-5: Investigating processes in Memory Dumps . . . . .	581
Recipe 15-6: Detecting DKOM Attacks with psscan . . . . .	588

Recipe 15-7: Exploring csrss.exe's Alternate Process Listings. . . . .	591
Recipe 15-8: Recognizing Process Context Tricks . . . . .	593
<b>16</b> Memory Forensics: Code Injection and Extraction. . . . .	601
Recipe 16-1: Hunting Suspicious Loaded DLLs . . . . .	603
Recipe 16-2: Detecting Unlinked DLLs with ldr_modules . . . . .	605
Recipe 16-3: Exploring Virtual Address Descriptors (VAD). . . . .	610
Recipe 16-4: Translating Page Protections . . . . .	614
Recipe 16-5: Finding Artifacts in Process Memory . . . . .	617
Recipe 16-6: Identifying Injected Code with Malfind and YARA . . . . .	619
Recipe 16-7: Rebuilding Executable Images from Memory . . . . .	627
Recipe 16-8: Scanning for Imported Functions with impscan. . . . .	629
Recipe 16-9: Dumping Suspicious Kernel Modules . . . . .	633
<b>17</b> Memory Forensics: Rootkits . . . . .	637
Recipe 17-1: Detecting IAT Hooks. . . . .	637
Recipe 17-2: Detecting EAT Hooks . . . . .	639
Recipe 17-3: Detecting Inline API Hooks. . . . .	641
Recipe 17-4: Detecting Interrupt Descriptor Table (IDT) Hooks . . . . .	644
Recipe 17-5: Detecting Driver IRP Hooks . . . . .	646
Recipe 17-6: Detecting SSDT Hooks . . . . .	650
Recipe 17-7: Automating Damn Near Everything with ssdt_ex . . . . .	654
Recipe 17-8: Finding Rootkits with Detached Kernel Threads . . . . .	655
Recipe 17-9: Identifying System-Wide Notification Routines . . . . .	658
Recipe 17-10: Locating Rogue Service Processes with svcscan . . . . .	661
Recipe 17-11: Scanning for Mutex Objects with mutantscan. . . . .	669
<b>18</b> Memory Forensics: Network and Registry . . . . .	673
Recipe 18-1: Exploring Socket and Connection Objects . . . . .	673
Recipe 18-2: Analyzing Network Artifacts Left by Zeus. . . . .	678
Recipe 18-3: Detecting Attempts to Hide TCP/IP Activity. . . . .	680
Recipe 18-4: Detecting Raw Sockets and Promiscuous NICs . . . . .	682
Recipe 18-5: Analyzing Registry Artifacts with Memory Registry Tools . . . . .	685
Recipe 18-6: Sorting Keys by Last Written Timestamp . . . . .	689
Recipe 18-7: Using Volatility with RegRipper . . . . .	692
Index. . . . .	695

# Introduction

**M**alware Analyst's Cookbook is a collection of solutions and tutorials designed to enhance the skill set and analytical capabilities of anyone who works with, or against, malware. Whether you're performing a forensic investigation, responding to an incident, or reverse-engineering malware for fun or as a profession, this book teaches you creative ways to accomplish your goals. The material for this book was designed with several objectives in mind. The first is that we wanted to convey our many years of experience in dealing with malicious code in a manner friendly enough for non-technical readers to understand, but complex enough so that technical readers won't fall asleep. That being said, malware analysis requires a well-balanced combination of many different skills. We expect that our readers have at least a general familiarity with the following topics:

- Networking and TCP/IP
- Operating system internals (Windows and Unix)
- Computer security
- Forensics and incident response
- Programming (C, C++, Python, and Perl)
- Reverse-engineering
- Vulnerability research
- Malware basics

Our second objective is to teach you how various tools work, rather than just how to use the tools. If you understand what goes on when you click a button (or type a command) as opposed to just knowing which button to click, you'll be better equipped to perform an analysis on the tool's output instead of just collecting the output. We realize that not everyone can or wants to program, so we've included over 50 tools on the DVD that accompanies the book; and we discuss hundreds of others throughout the text. One thing we tried to avoid is providing links to every tool under the sun. We limit our discussions to tools that we're familiar with, and—as much as possible—tools that are freely available.

Lastly, this book is *not* a comprehensive guide to all tasks you should perform during examination of a malware sample or during a forensic investigation. We tried to include solutions to problems that are common enough to be most beneficial to you, but rare enough to not be covered in other books or websites. Furthermore, although malware can target many platforms such as Windows, Linux, Mac OS X, mobile devices, and hardware/firmware components, our book focuses primarily on analyzing Windows malware.

## Who Should Read This Book

If you want to learn about malware, you should read this book. We expect our readers to be forensic investigators, incident responders, system administrators, security engineers, penetration testers, malware analysts (of course), vulnerability researchers, and anyone looking to be more involved in security. If you find yourself in any of the following situations, then you are within our target audience:

- You're a member of your organization's incident handling, incident response, or forensics team and want to learn some new tools and techniques for dealing with malware.
- You work as a systems, security, or network administrator and want to understand how you can protect end users more effectively.
- You're a member of your country's Computer Emergency Response Team (CERT) and need to identify and investigate malware intrusions.
- You work at an antivirus or research company and need practical examples of analyzing and reporting on modern malware.
- You're an aspiring student hoping to learn techniques that colleges and universities just don't teach.
- You work in the IT field and have recently become bored, so you're looking for a new specialty to compliment your technical knowledge.

## How This Book Is Organized

This book is organized as a set of recipes that solve specific problems, present new tools, or discuss how to detect and analyze malware in interesting ways. Some of the recipes are stand-alone, meaning the problem, discussion, and solution are presented in the same recipe. Other recipes flow together and describe a sequence of actions that you can use to solve a larger problem. The book covers a large array of topics and becomes continually more advanced and specialized as it goes on. Here is a preview of what you can find in each chapter:

- **Chapter 1, Anonymizing Your Activities:** Describes how you conduct online investigations without exposing your own identity. You'll use this knowledge to stay safe when following along with exercises in the book and when conducting research in the future.
- **Chapter 2, Honeypots:** Describes how you can use honeypots to collect the malware being distributed by bots and worms. Using these techniques, you can grab new variants of malware families from the wild, share them in real time with other

researchers, analyze attack patterns, or build a workflow to automatically analyze the samples.

- **Chapter 3, Malware Classification:** Shows you how to identify, classify, and organize malware. You'll learn how to detect malicious files using custom antivirus signatures, determine the relationship between samples, and figure out exactly what functionality attackers may have introduced into a new variant.
- **Chapter 4, Sandboxes and Multi-AV Scanners:** Describes how you can leverage online virus scanners and public sandboxes. You'll learn how to use scripts to control the behavior of your sample in the target sandbox, how to submit samples on command line with Python scripts, how to store results to a database, and how to scan for malicious artifacts based on sandbox results.
- **Chapter 5, Researching Domains and IP Addresses:** Shows you how to identify and correlate information regarding domains, hostnames, and IP addresses. You'll learn how to track fast flux domains, determine the alleged owner of a domain, locate other systems owned by the same group of attackers, and create static or interactive maps based on the geographical location of IP addresses.
- **Chapter 6, Documents, Shellcode, and URLs:** In this chapter, you'll learn to analyze JavaScript, PDFs, Office documents, and packet captures for signs of malicious activity. We discuss how to extract shellcode from exploits and analyze it within a debugger or in an emulated environment.
- **Chapter 7, Malware Labs:** Shows how to build a safe, flexible, and inexpensive lab in which to execute and monitor malicious code. We discuss solutions involving virtual or physical machines and using real or simulated Internet.
- **Chapter 8, Automation:** Describes how you can automate the execution of malware in VMware or VirtualBox virtual machines. The chapter introduces several Python scripts to create custom reports about the malware's behavior, including network traffic logs and artifacts created in physical memory.
- **Chapter 9, Dynamic Analysis:** One of the best ways to understand malware behavior is to execute it and watch what it does. In this chapter, we cover how to build your own API monitor, how to prevent certain evidence from being destroyed, how to log file system and Registry activity in real time *without* using hooks, how to compare changes to a process's handle table, and how to log commands that attackers send through backdoors.
- **Chapter 10, Malware Forensics:** Focuses on ways to detect rootkits and stealth malware using forensic tools. We show you how to scan the file system and Registry for hidden data, how to bypass locked file restrictions and remove stubborn malware, how to detect HTML injection and how to investigate a new form of Registry "slack" space.

- **Chapter 11, Debugging Malware:** Shows how you can use a debugger to analyze, control, and manipulate a malware sample's behaviors. You'll learn how to script debugging sessions with Python and how to create debugger plug-ins that monitor API calls, output HTML behavior reports, and automatically highlight suspicious activity.
- **Chapter 12, De-obfuscation:** Describes how you can decode, decrypt, and unpack data that attackers intentionally try to hide from you. We walk you through the process of reverse-engineering a malware sample that encrypts its network traffic so you can recover stolen data. In this chapter, you also learn techniques to crack domain generation algorithms.
- **Chapter 13, Working with DLLs:** Describes how to analyze malware distributed as Dynamic Link Libraries (DLLs). You'll learn how to enumerate and examine a DLL's exported functions, how to run the DLL in a process of your choice (and bypass host process restrictions), how to execute DLLs as a Windows service, and how to convert DLLs to standalone executables.
- **Chapter 14, Kernel Debugging:** Some of the most malicious malware operates only in kernel mode. This chapter covers how to debug the kernel of a virtual machine infected with malware to understand its low-level functionality. You learn how to create scripts for WinDbg, unpack kernel drivers, and to leverage IDA Pro's debugger plug-ins.
- **Chapter 15, Memory Forensics with Volatility:** Shows how to acquire memory samples from physical and virtual machines, how to install the Volatility advanced memory forensics platform and associated plug-ins, and how to begin your analysis by detecting process context tricks and DKOM attacks.
- **Chapter 16, Memory Forensics: Code Injection and Extraction:** Describes how you can detect and extract code (unlinked DLLs, shellcode, and so on) hiding in process memory. You'll learn to rebuild binaries, including user mode programs and kernel drivers, from memory samples and how to rebuild the import address tables (IAT) of packed malware based on information in the memory dump.
- **Chapter 17, Memory Forensics: Rootkits:** Describes how to detect various forms of rootkit activity, including the presence of IAT, EAT, Inline, driver IRP, IDT, and SSDT hooks on a system. You'll learn how to identify malware that hides in kernel memory without a loaded driver, how to locate system-wide notification routines, and how to detect attempts to hide running Windows services.
- **Chapter 18, Network and Registry:** Shows how to explore the artifacts created on a system due to a malware sample's network activity. You'll learn to detect active connections, listening sockets, and the use of raw sockets and promiscuous mode network cards. This chapter also covers how to extract volatile Registry keys and values from memory.

## Setting Up Your Environment

We performed most of the development and testing of Windows tools on 32-bit Windows XP and Windows 7 machines using Microsoft's Visual Studio and Windows Driver Kit. If you need to recompile our tools for any reason (for example to fix a bug), or if you're interested in building your own tools based on source code that we've provided, then you can download the development environments here:

- **The Windows Driver Kit:** <http://www.microsoft.com/whdc/devtools/WDK/default.msp>
- **Visual Studio C++ Express:** <http://www.microsoft.com/express/Downloads/#2010-Visual-CPP>

As for the Python tools, we developed and tested them on Linux (mainly Ubuntu 9.04, 9.10, or 10.04) and Mac OS X 10.4 and 10.5. You'll find that a majority of the Python tools are multi-platform and run wherever Python runs. If you need to install Python, you can get it from the website at <http://python.org/download/>. We recommend using Python version 2.6 or greater (but not 3.x), because it will be most compatible with the tools on the book's DVD.

Throughout the book, when we discuss how to install various tools on Linux, we assume you're using Ubuntu. As long as you know your way around a Linux system, you're comfortable compiling packages from source, and you know how to solve basic dependency issues, then you shouldn't have a problem using any other Linux distribution. We chose Ubuntu because a majority of the tools (or libraries on which the tools depend) that we reference in the book are either preinstalled, available through the apt-get package manager, or the developers of the tools specifically say that their tools work on Ubuntu.

You have a few options for getting access to an Ubuntu machine:

- **Download Ubuntu directly:** <http://www.ubuntu.com/desktop/get-ubuntu/download>
- **Download Lenny Zeltser's REMnux:** <http://REMnux.org>. REMnux is an Ubuntu system preconfigured with various open source malware analysis tools. REMnux is available as a VMware appliance or ISO image.
- **Download Rob Lee's SANS SIFT Workstation:** <https://computer-forensics2.sans.org/community/siftkit/>. SIFT is an Ubuntu system preconfigured with various forensic tools. SIFT is available as a VMware appliance or ISO image.

We always try to provide a URL to the tools we mention in a recipe. However, we use some tools significantly more than others, thus they appear in five to ten recipes. Instead

of linking to each tool each time, here is a list of the tools that you should have access to throughout all chapters:

- **Sysinternals Suite:** <http://technet.microsoft.com/en-us/sysinternals/bb842062.aspx>
- **Wireshark:** <http://www.wireshark.org/>
- **IDA Pro and Hex-Rays:** <http://www.hex-rays.com/idapro/>
- **Volatility:** <http://code.google.com/p/volatility/>
- **WinDbg Debugger:** <http://www.microsoft.com/whdc/devtools/debugging/default.msp>
- **YARA:** <http://code.google.com/p/yara-project/>
- **Process Hacker:** <http://processhacker.sourceforge.net/>

You should note a few final things before you begin working with the material in the book. Many of the tools require administrative privileges to install *and* execute. Typically, mixing malicious code and administrative privileges isn't a good idea, so you must be sure to properly secure your environment (see Chapter 7 for setting up a virtual machine if you do not already have one). You must also be aware of any laws that may prohibit you from collecting, analyzing, sharing, or reporting on malicious code. Just because we discuss a technique in the book does not mean it's legal in the city or country in which you reside.

## Conventions

To help you get the most from the text and keep track of what's happening, we've used a number of conventions throughout the book.

### RECIPE X-X: RECIPE TITLE

Boxes like this contain recipes, which solve specific problems, present new tools, or discuss how to detect and analyze malware in interesting ways. Recipes may contain helpful steps, supporting figures, and notes from the authors. They also may have supporting materials associated with them on the companion DVD. If they do have supporting DVD materials, you will see a DVD icon and descriptive text, as follows:



*You can find supporting material for this recipe on the companion DVD.*

For your further reading and research, recipes may also have endnotes<sup>1</sup> that cite Internet or other supporting sources. You will find endnote references at the end of the recipe. Endnotes are numbered sequentially throughout a chapter.

<sup>1</sup> This is an endnote. This is the format for a website source

**NOTE**

Tips, hints, tricks, and asides to the current discussion look like this.

As for other conventions in the text:

- New terms and important words appear in *italics* when first introduced.
- Keyboard combinations are treated like this: Ctrl+R.
- File names are in parafont, (filename.txt), URLs and code (API functions and variable names) within the text are treated like so: `www.site.org`, `LoadLibrary`, `var1`.
- This book uses monofont type with no highlighting for most code examples. Code fragments may be broken into multiple lines or truncated to fit on the page:

```
This is an example of monofont type with a long \
    line of code that needed to be broken.
This truncated line shows how [REMOVED]
```

- This book uses bolding to emphasize code. User input for commands and code that is of particular importance appears in bold:

```
$ date ; typing into a Unix shell
Wed Sep  1 14:30:20 EDT 2010
C:\> date ; typing into a Windows shell
Wed 09/01/2010
```



# On The Book's DVD

The book's DVD contains evidence files, videos, source code, and programs that you can use to follow along with recipes or to conduct your own investigations and analysis. It also contains the full-size, original images and figures that you can view, since they appear in black and white in the book. The files are organized on the DVD in folders named according to the chapter and recipe number. Most of the tools on the DVD are written in C, Python, or Perl and carry a GPLv2 or GPLv3 license. You can use a majority of them as-is, but a few may require small modifications depending on your system's configuration. Thus, even if you're not a programmer, you should take a look at the top of the source file to see if there are any notes regarding dependencies, the platforms on which we tested the tools, and any variables that you may need to change according to your environment.

We do not guarantee that all programs are bug free (who does?), thus, we welcome feature requests and bug reports addressed to [malwarecookbook@gmail.com](mailto:malwarecookbook@gmail.com). If we do provide updates for the code in the future, you can always find the most recent versions at <http://www.malwarecookbook.com>.

The following table shows a summary of the tools that you can find on the DVD, including the corresponding recipe number, programming language, and intended platform.

Recipe	Tool	Description	Language	Platform
1-3	<code>torwget.py</code>	Multi-platform TOR-enabled URL fetcher	Python	All
2-3	<code>wwwhoneynet.tgz</code>	CGI scripts to accept submissions from nepenthes and dionaea honeypots	Python	All
3-3	<code>clamav_to_yara.py</code>	Convert ClamAV antivirus signatures to YARA rules	Python	All
3-4	<code>peid_to_yara.py</code>	Convert PEiD packer signatures to YARA rules	Python	All
3-7	<code>av_multiscan.py</code>	Script to implement your own antivirus multi-scanner	Python	All
3-8	<code>pescanner.py</code>	Detect malicious PE file attributes	Python	All
3-10	<code>ssdeep_procs.py</code>	Detect self-mutating code on live Windows systems using ssdeep	Python	Windows only (XP/7)

Recipe	Tool	Description	Language	Platform
4-4	avsubmit.py	Command-line interface to VirusTotal, ThreatExpert, Jotti, and NoVirusThanks	Python	All
4-12	dbmgr.py	Malware artifacts database manager	Python	All
4-12	artifactsscanner.py	Application to scan live Windows systems for artifacts (files, Registry keys, mutexes) left by malware	Python	Windows only (XP/7)
5-13	mapper.py	Create static PNG images of IP addresses plotted on a map using GeoIP	Python	All
5-14	googlegeoip.py	Create dynamic/interactive geographical maps of IP addresses using Google charts	Python	All
6-9	sc_distorm.py	Script to produce disassemblies (via DiStorm) of shellcode and optionally apply an XOR mask	Python	All
8-1	vmauto.py	Python class for automating malware execution in VirtualBox and VMware guests	Python	All
8-1	mybox.py	Sample automation script for VirtualBox based on vmauto.py	Python	All
8-7	myvmware.py	Sample automation script for VMware based on vmauto.py	Python	All
8-7	analysis.py	Python class for building sandboxes with support for analyzing network traffic, packet captures, and memory.	Python	Linux
9-3	RegFsNotify.exe	Tool to detect changes to the Registry and file system in real time (from user mode without API hooks)	C	Windows only (XP/7)
9-5	HandleDiff.exe	Tool to detect changes to the handle tables of all processes on a system (useful to analyze the side-effects of code injecting malware)	C	Windows only (XP/7)
9-10	Preservation.zip	Kernel driver for monitoring notification routines, preventing processes from terminating, preventing files from being deleted, and preventing other drivers from loading	C	Windows XP only

Recipe	Tool	Description	Language	Platform
9-15	cmd.exe	Custom command shell (cmd.exe) for logging malware activity and backdoor activity	C	Windows only (XP/7)
10-2	tsk-xview.exe	Cross-view based rootkit detection tool based on The Sleuth Kit API and Microsoft's Offline Registry API.	C	Windows XP only
10-4	closehandle.exe	Command-line tool to remotely close a handle that another process has open	C	Windows only (XP/7)
10-7	HTMLInjection Detector.exe	Detect HTML injection attacks on banking and financial websites	C	Windows XP only
10-8	routes.pl	RegRipper plug-in for printing a computer's routing table	Perl	All
10-8	pendingdelete.pl	RegRipper plug-in for printing files that are pending deletion.	Perl	All
10-8	disallowrun.pl	RegRipper plug-in for printing processes that malware prevents from running	Perl	All
10-8	shellexecute-hooks.pl	RegRipper plug-in for printing ShellExecute hooks (a method of DLL injection)	Perl	All
10-9	dumpcerts.pl	Parse::Win32Registry module to extract and examine cryptography certificates stored in Registry hives	Perl	All
10-10	somethingelse.pl	Parse::Win32Registry module for finding hidden binary data in the Registry	Perl	All
11-2	scloader.exe	Executable wrapper for launching shell code in a debugger	C	Windows only (XP/7)
11-9	scd.py	Immunity Debugger PyCommand for finding shellcode in arbitrary binary files	Python	Windows only (XP/7)
11-10	findhooks.py	Immunity Debugger PyCommand for finding Inline-style user mode API hooks	Python	Windows only (XP/7)
11-12	pymon.py	WinAppDbg plug-in for monitoring API calls, alerting on suspicious flags/parameters and producing an HTML report	Python	Windows only (XP/7)

Recipe	Tool	Description	Language	Platform
12-1	xortools.py	Python library for encoding/decoding XOR, including brute force methods and automated YARA signature generation	Python	All
12-10	trickimprec.py	Immunity Debugger PyCommand for assistance when rebuilding import tables with Import REconstructor	Python	Windows only (XP/7)
12-11	kraken.py	Immunity Debugger PyCommand for cracking Kraken's Domain Generation Algorithm (DGA)	Python	Windows only (XP/7)
12-12	sbstrings.py	Immunity Debugger PyCommand for decrypting Silent Banker strings.	Python	Windows only (XP/7)
13-4	rundll32ex.exe	Extended version of rundll32.exe that allows you to run DLLs in other processes, call exported functions, and pass parameters	C	Windows XP only
13-7	install_svc.bat	Batch script for installing a service DLL (for dynamic analysis of the DLL)	Batch	Windows only
13-7	install_svc.py	Python script for installing a service DLL and supplying optional arguments to the service	Python	Windows only
13-8	dll2exe.py	Python script for converting a DLL into a standalone executable	Python	All
14-8	DriverEntryFinder	Kernel driver to find the correct address in kernel memory to set breakpoints for catching new drivers as they load	C	Windows XP only
14-10	windbg_to_ida.py	Python script to convert WinDbg output into data that can be imported into IDA	Python	All
14-11	WinDbgNotify.txt	WinDbg script for identifying malicious notification routines.	WinDbg scripting language	Windows only

# 1

## Anonymizing Your Activities

*In our daily lives we like to have a certain level of privacy. We have curtains on our windows, doors for our offices, and even special screen protectors for computers to keep out prying eyes. This idea of wanting privacy also extends to the use of the Internet. We do not want people knowing what we typed in Google, what we said in our Instant Message conversations, or what websites we visited. Unfortunately, your private information is largely available if someone is watching. When doing any number of things on the Internet, there are plenty of reasons you might want to go incognito. However, that does not mean you're doing anything wrong or illegal.*

**T**he justification for anonymity when researching malware and bad guys is pretty straightforward. You do not want information to show up in logs and other records that might tie back to you or your organization. For example, let's say you work at a financial firm and you recently detected that a banking trojan infected several of your systems. You collected malicious domain names, IP addresses, and other data related to the malware. The next steps you take in your research may lead you to websites owned by the criminals. As a result, if you are not taking precautions to stay anonymous, your IP address will show up in various logs and be visible to miscreants.

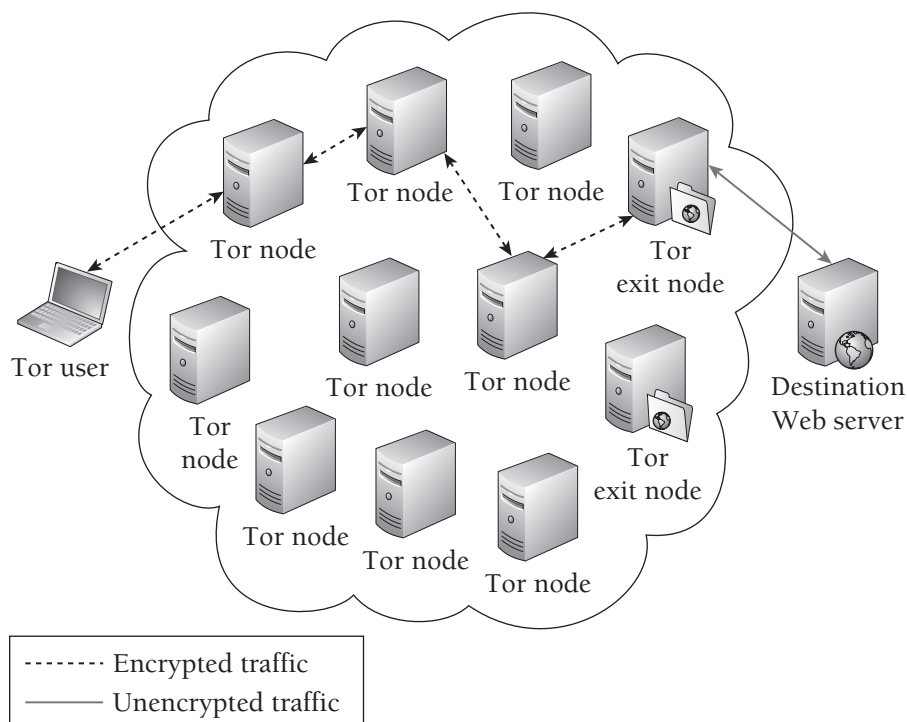
If the criminals can identify you or the organization from which you conduct your research, they may change tactics or go into hiding, thus spoiling your investigation. Even worse, they may turn the tables and attack you in a personal way (such as identity theft) or launch a distributed denial of service (DDoS) attack against your IP address. For example, the Storm worm initiated DDoS attacks against machines that scanned an infected system (see <http://www.securityfocus.com/news/11482>).

This chapter contains several methods that you can use to conduct research without blowing your cover. We've positioned this chapter to be first in the book, so you can use the techniques when following along with examples in the remaining chapters. Keep in mind that you may never truly be anonymous in what you are doing, but more privacy is better than no privacy!

## The Onion Router (Tor)

A widely known and accepted solution for staying anonymous on the Internet is *Tor*. Tor, despite being an acronym, is written with only the first letter capitalized and stands for *The Onion Router* or *the onion routing network*. The project has a long history stemming from a project run by the Naval Research Laboratory. You can read all about it at <http://www.torproject.org>.

Tor is a network of computers around the world that forward requests in an encrypted manner from the start of the request until it reaches the last machine in the network, which is known as an exit node. At this point, the request is decrypted and passed to the destination server. *Exit nodes* are specifically used as the last hop for traffic leaving the Tor network and then as the first hop for returning traffic. When you use Tor, the systems with which you are communicating see all incoming traffic as if it originated from the exit node. They do not know where you are located or what your actual IP address is. Furthermore, the other systems in the Tor network cannot determine your location either, because they are essentially forwarding traffic with no knowledge of where it actually originated. The responses to your requests will return to your system, but as far as the Tor network is concerned, you are just another hop along the way. In essence, you are anonymous. Figure 1-1 shows a simplified view of the Tor network.



**Figure 1-1:** Simplified Tor Diagram