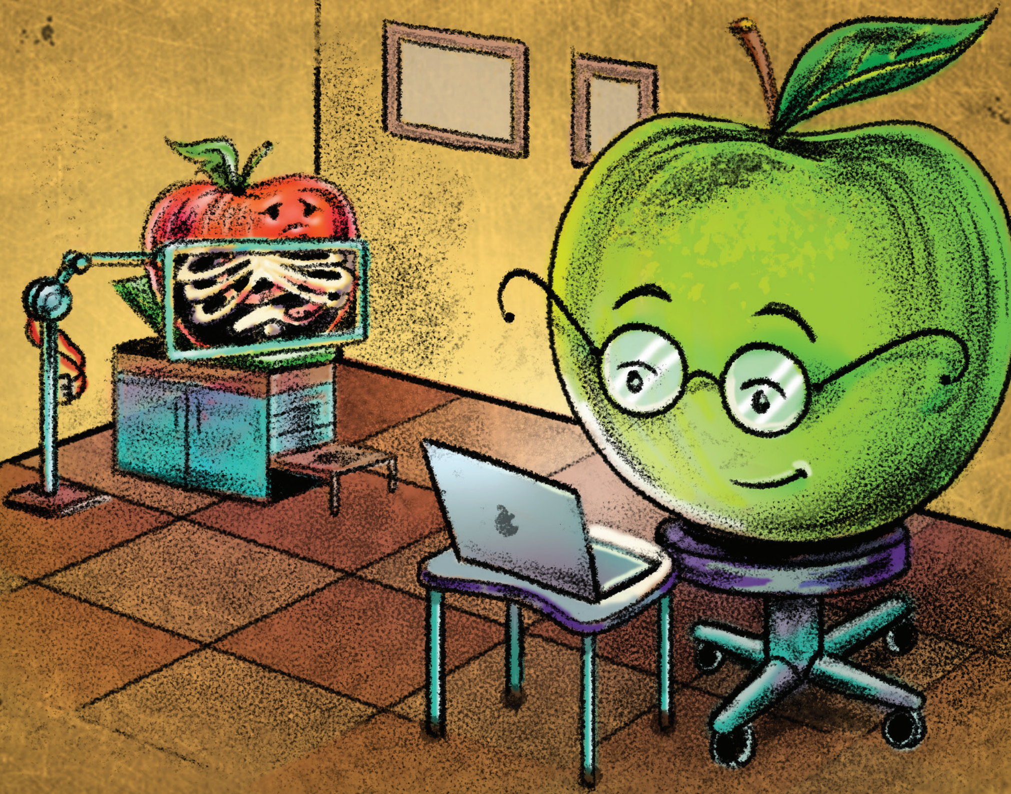
VOLUME 2

The Art of Mac Malware

Detecting Malicious Software



Patrick Wardle

no starch press®

THE ART OF MAC MALWARE, VOLUME 2

THE ART OF MAC MALWARE

Volume 2

Detecting Malicious Software

by Patrick Wardle

no starch press®

San Francisco

First printing

ISBN-13: 978-1-7185-0378-6 (print)
ISBN-13: 978-1-7185-0379-3 (ebook)

Published by No Starch Press®, Inc.
245 8th Street, San Francisco, CA 94103
phone: +1.415.863.9900
www.nostarch.com; info@nostarch.com

Publisher: William Pollock
Managing Editor: Jill Franklin
Production Manager: Sabrina Plomitallo-González
Production Editor: Jennifer Kepler
Developmental Editor: Frances Saux
Cover Illustrator: Garry Booth
Interior Design: Octopod Studios
Technical Reviewer: Tom McGuire
Copyeditor: Lisa McCoy
Proofreader: Audrey Doyle
Indexer: BIM Creatives, LLC

Library of Congress Control Number: 2024034450

To my loving and patient parents, Stephen and Norma.
And to Andy #UnaMas para siempre.

About the Author

Patrick Wardle is the founder of Objective-See, a nonprofit dedicated to creating free, open source macOS security tools and organizing the "Objective by the Sea" Apple security conference. He is also the co-founder and CEO of DoubleYou, a cybersecurity startup focused on empowering the builders of Apple-focused security tools. Having worked at both NASA and the National Security Agency and having presented at countless security conferences, he is intimately familiar with aliens, spies, and talking nerdy.

About the Technical Reviewer

Tom McGuire has been working in the security industry since the late 1990s. He is the chief technical officer of a cybersecurity firm and a senior instructor at the Johns Hopkins Whiting School of Engineering, where he teaches reverse engineering, operating system security, cryptology, and cyber risk management. When not doing security stuff, he can be found hanging out with his family and watching the Red Sox.

BRIEF CONTENTS

Foreword xvii

Acknowledgments xix

Introduction xxi

PART I: DATA COLLECTION 1

Chapter 1: Examining Processes 3

Chapter 2: Parsing Binaries 39

Chapter 3: Code Signing 75

Chapter 4: Network State and Statistics 101

Chapter 5: Persistence 119

PART II: SYSTEM MONITORING 139

Chapter 6: Log Monitoring 141

Chapter 7: Network Monitoring 155

Chapter 8: Endpoint Security 179

Chapter 9: Muting and Authorization Events 205

PART III: TOOL DEVELOPMENT 231

Chapter 10: Persistence Enumerator 233

Chapter 11: Persistence Monitor 253

Chapter 12: Mic and Webcam Monitor 279

Chapter 13: DNS Monitor 297

Chapter 14: Case Studies 313

Index 329

CONTENTS IN DETAIL

FOREWORD xvii

ACKNOWLEDGMENTS xix

INTRODUCTION xxi

What You'll Find in This Book . . . xxiii
Who Should Read This Book? . . . xxiv
The Code and Malware Specimens . . . xxv
Development Environment . . . xxvi
    Code Signing Requirements . . . xxvi
    Entitlements . . . xxvii
Safely Analyzing Malware . . . xxvii
Additional Resources . . . xxix
    Books . . . xxix
    Websites . . . xxix

PART I: DATA COLLECTION 1

1
EXAMINING PROCESSES 3

Process Enumeration . . . 4
Audit Tokens . . . 5
Paths and Names . . . 6
    Identifying Hidden Files and Directories . . . 6
    Obtaining the Paths of Deleted Binaries . . . 7
    Validating Process Names . . . 8
Process Arguments . . . 9
Process Hierarchies . . . 13
    Finding the Parent . . . 14
    Returning the Process Responsible for Spawning Another . . . 16
    Retrieving Information with Application Services APIs . . . 17
Environment Information . . . 19
Code Signing . . . 24
Loaded Libraries . . . 24
Open Files . . . 28
    proc_pidinfo . . . 29
    lsof . . . 30
Other Information . . . 31
    Execution State . . . 32
    Execution Architecture . . . 32
    Start Time . . . 34
    CPU Utilization . . . 35
Conclusion . . . 36

2
PARSING BINARIES 39

Universal Binaries . . . 39
  Inspecting . . . 40
  Parsing . . . 42
Mach-O Headers . . . 50
Load Commands . . . 53
Extracting Dependencies . . . 54
  Finding Dependency Paths . . . 54
  Analyzing Dependencies . . . 56
Extracting Symbols . . . 59
Detecting Packed Binaries . . . 62
  Dependencies and Symbols . . . 63
  Section and Segment Names . . . 63
  Entropy Calculations . . . 67
Detecting Encrypted Binaries . . . 70
Conclusion . . . 71

3
CODE SIGNING 75

The Importance of Code Signing in Malware Detection . . . 76
Disk Images . . . 78
  Manually Verifying Signatures . . . 78
  Extracting Code Signing Information . . . 79
  Extracting Notarization Information . . . 82
  Running the Tool . . . 84
Packages . . . 84
  Reverse Engineering pkgutil . . . 86
  Accessing Framework Functions . . . 88
  Validating the Package . . . 90
  Checking Package Notarization . . . 91
  Running the Tool . . . 92
On-Disk Applications and Executables . . . 93
Running Processes . . . 95
Detecting False Positives . . . 96
Code Signing Error Codes . . . 97
Conclusion . . . 97

4
NETWORK STATE AND STATISTICS 101

Host-Based vs. Network-Centric Collection . . . 102
Malicious Networking Activity . . . 102
Capturing the Network State . . . 105
  Retrieving Process File Descriptors . . . 106
  Extracting Network Sockets . . . 106
  Obtaining Socket Details . . . 107
  Running the Tool . . . 111
Enumerating Network Connections . . . 111
  Linking to NetworkStatistics . . . 113
  Creating Network Statistic Managers . . . 113

Defining Callback Logic . . . 114
Starting Queries . . . 115
Running the Tool . . . 115
Conclusion . . . 117

5
PERSISTENCE 119

Examples of Persistent Malware . . . 120
Background Task Management . . . 123
Examining the Subsystem . . . 124
Dissecting sfltool . . . 127
Writing a Background Task Management Database Parser . . . 130
Finding the Database Path . . . 130
Deserializing Background Task Management Files . . . 131
Accessing Metadata . . . 134
Identifying Malicious Items . . . 135
Using DumpBTM in Your Own Code . . . 136
Conclusion . . . 137

PART II: SYSTEM MONITORING 139

6
LOG MONITORING 141

Exploring Log Information . . . 142
The Unified Logging Subsystem . . . 143
Manually Querying the log Utility . . . 144
Reverse Engineering log APIs . . . 145
Streaming Log Data . . . 146
Extracting Log Object Properties . . . 148
Determining Resource Consumption . . . 151
Conclusion . . . 152

7
NETWORK MONITORING 155

Obtaining Regular Snapshots . . . 156
DNS Monitoring . . . 157
Using the NetworkExtension Framework . . . 159
Activating a System Extension . . . 160
Enabling the Monitoring . . . 161
Writing the Extension . . . 162
Filter Data Providers . . . 170
Enabling Filtering . . . 170
Writing the Extension . . . 171
Querying the Flow . . . 173
Running the Monitor . . . 174
Conclusion . . . 176

8
ENDPOINT SECURITY 179

The Endpoint Security Workflow . . . 180
  Events of Interest . . . 182
  Clients, Handler Blocks, and Event Handling . . . 185
Creating a Process Monitor . . . 190
  Subscribing to Events . . . 191
  Extracting Process Objects . . . 191
  Extracting Process Information . . . 192
  Stopping the Client . . . 199
File Monitoring . . . 200
Conclusion . . . 203

9
MUTING AND AUTHORIZATION EVENTS 205

Muting . . . 206
Mute Inversion . . . 209
  Beginning Mute Inversion . . . 210
  Monitoring Directory Access . . . 211
Authorization Events . . . 213
  Creating a Client and Subscribing to Events . . . 213
  Meeting Message Deadlines . . . 215
  Checking Binary Origins . . . 217
  Blocking Background Task Management Bypasses . . . 219
Building a File Protector . . . 223
Conclusion . . . 228

PART III: TOOL DEVELOPMENT 231

10
PERSISTENCE ENUMERATOR 233

Tool Design . . . 234
  Command Line Options . . . 235
  Plug-ins . . . 235
  Persistent Item Types . . . 238
Exploring the Plug-ins . . . 240
  Background Task Management . . . 241
  Browser Extension . . . 242
  Dynamic Library Insertion . . . 246
  Dynamic Library Proxying and Hijacking . . . 249
Conclusion . . . 252

11
PERSISTENCE MONITOR 253

Entitlements . . . 254
  Applying for Endpoint Security Entitlements . . . 254
  Registering App IDs . . . 254

Creating Provisioning Profiles . . . 255
Enabling Entitlements in Xcode . . . 255
Tool Design . . . 257
Plug-ins . . . 258
Background Task Management Events . . . 261
XPC . . . 265
Creating Listeners and Delegates . . . 265
Extracting Audit Tokens . . . 266
Extracting Code Signing Details . . . 268
Setting Client Requirements . . . 270
Enabling Remote Connections . . . 271
Exposing Methods . . . 272
Initiating Connections . . . 274
Invoking Remote Methods . . . 275
Conclusion . . . 276

12
MIC AND WEBCAM MONITOR 279

Tool Design . . . 280
Mic and Camera Enumeration . . . 281
Audio Monitoring . . . 282
Camera Monitoring . . . 285
Device Connections and Disconnections . . . 286
Responsible Process Identification . . . 288
Triggering Scripts . . . 291
Stopping . . . 293
Conclusion . . . 294

13
DNS MONITOR 297

Network Extension Deployment Prerequisites . . . 298
Packaging the Extension . . . 299
Tool Design . . . 301
The App . . . 301
The Extension . . . 302
Interprocess Communication . . . 303
Building and Dumping DNS Caches . . . 304
Blocking DNS Traffic . . . 307
Classifying Endpoints . . . 310
Conclusion . . . 311

14
CASE STUDIES 313

Shazam's Mic Access . . . 313
DazzleSpy Detection . . . 315
Exploit Detection . . . 315
Persistence . . . 317
Network Access . . . 319

The 3CX Supply Chain Attack 319
File Monitoring 320
Network Monitoring 322
Process Monitoring 323
Capturing Self-Deletion 325
Detecting Exfiltration 326
Conclusion 327

INDEX **329**

FOREWORD

I first encountered Patrick while writing a book about ARM assembly internals and reverse engineering. Apple had recently released its ARM-based Apple Silicon chip, and Patrick was the first to publicly analyze what was then the only known malware sample compiled for it. I could tell Patrick was someone who stayed ahead of the curve and embraced technological changes, so we started collaborating on a chapter about reversing ARM64 macOS malware.

As the macOS ecosystem and architecture evolved, Patrick continued to research macOS threats despite architectural changes, consistently contributing to the malware analysis and detection community. His free resources, including timely and comprehensive research on the latest macOS threats, the nonprofit conference Objective by the Sea, numerous educational initiatives, and open source detection tools, have impacted countless people in this industry.

Writing security tools for macOS requires both time and resilience. We all know how quickly technology can advance, and Apple Silicon introduced changes so profound that they affected the processor architecture. Whether you are new to the field or an experienced professional, keeping up with changes to the ecosystem you focus on is crucial for success. Patrick has come up with novel ways to make threat detection possible on macOS. (You know you're doing something right when major antivirus companies attempt to use your detection code commercially, without permission.)

In *The Art of Mac Malware*, Volume 2, Patrick tackles proactive defense, focusing on specific programming techniques and macOS internals for detecting and countering threats. This book's in-depth approach sets it apart: instead of merely analyzing a single malware sample, it describes the APIs and techniques necessary to identify infection patterns, automate macOS threat detection, and develop custom tools. You'll learn how to create software that identifies infections in real time, moving beyond postmortem analysis.

If you want to study these techniques, you'd better learn from the best—someone who has built such tools, has battle-tested them in practice, and continues to adapt to any changes that could render these techniques ineffective.

Maria Markstedter
Founder of Azeria Labs and
Forbes Person of the Year in Cybersecurity

ACKNOWLEDGMENTS

A computer is made up of countless components crafted and assembled by many individual engineers. Although I'm pretty sure I'm not a computer, I too feel I am a product of many individuals, and despite the single name on the cover, you wouldn't be holding this book in your hands without them.

First and foremost, I want to acknowledge my parents, who expertly navigated the complexities of raising a child, deftly sublimating my rebellious tendencies into a love of learning that has benefited me ever since. Similarly, I am forever grateful to my older brother, Keelian, who has challenged and inspired me. (Nothing like a never-ending sibling rivalry to bring out the best in us . . . right?)

I also want to thank my many co-workers and colleagues at the National Security Agency and in the larger infosec community, whose guidance and support have been invaluable over the years. Though there are far too many to name in this short section, a few—namely, my close friends and colleagues Kasey, Tom, Josh, and Jon—have had a profoundly positive influence on both my personal life and career. Others, such as the brilliant Jonathan Levin and Arnaud Abbati, have selflessly provided indispensable technical insights and mentorship, giving me the confidence and expertise to write this book. I am lucky to count both as close friends. I also want to thank my DoubleYou co-founder, Mike, for our decade-spanning friendship and for partnering with me as we build something epic together.

To my confidant, companion, and muse, Andy: words cannot express how grateful I am for your insights, guidance, support, and love.

I also want to acknowledge the many patrons of Objective-See, whose continued support made this book, and my vision of free, open source Mac security tools, a reality. The companies that participate in the Friends of Objective-See programs not only support the mission of the Objective-See Foundation but also have helped this book see the light of day. For that, I am forever grateful. These Friends of Objective-See include Kandji, Jamf, 1Password, MacPaw, Palo Alto Networks, Malwarebytes, iVerify, Huntress, SmugMug, Halo Privacy, and the Mitten Mac.

Last, but certainly not least, are the many individuals who worked directly on the book. These delightful (and, yes, sometimes strict) humans kept me roughly on schedule to bring this book to fruition. They include my good friend Tom McGuire (Tmac), who put countless hours into the rather thankless job of technical editor, and the incredibly professional and hardworking crew at No Starch Press, including founder Bill Pollock and the book's main editor, Frances Saux.

INTRODUCTION

We are, unfortunately, living in a golden age of Mac malware. Sales of Mac computers continue to flourish year over year,[1] while industry reports predict that Mac will become the dominant platform in enterprise environments.[2] As Apple's share of the global computer market grows, Macs have become an ever-more compelling target for opportunistic hackers and malware authors. Some studies have even found, on average, more threats and malware on Mac systems than on Windows ones.[3]

When it comes to protecting Macs and their users, analyzing malware (the topic of *The Art of Mac Malware,* Volume 1) is only half the battle. Detecting malicious code in the first place is the other, perhaps even more important, piece. There are many approaches to detecting malicious code, each with pros and cons. At one end of the detection spectrum, we can leverage databases of malware signatures. By scanning binaries for sequences of malicious bytes, we can efficiently identify known threats. However, we fail to detect new malware or variants. This downside is troublesome. To see why, consider the case of the malware known as FruitFly. Carefully crafted by a single programmer and deployed in a highly targeted manner, it remained undetected for over a decade, as no antivirus program contained a signature to detect it. The malware spied on unknowing victims using Macs' mics and webcams, leading to damaging real-life consequences.[4]

At the other end of the detection spectrum are *behavior-based heuristics*, which focus on a malicious program's actions or impact on a system. To understand this approach, consider the last time you were sick. Perhaps your illness started with a runny nose, a headache, a sore throat, or a stomachache. While you probably didn't know exactly what pathogen had infected you, your body's symptoms indicated that you were no longer your normal, healthy self. We can use a similar strategy to generically and heuristically detect digital pathogens: by looking for symptoms and anomalies.

Even novel and stealthy malware specimens will produce observable events when they interact with a system. Some, such as the spawning of a newly persisted unsigned process, may be easy to detect. Others, like the surreptitious planting of a trojanized dynamic library or a covert exfiltration channel, are more subtle. Regardless, if we can programmatically detect these behaviors, we should be able to ascertain whether a system is infected and, by identifying the responsible process, pinpoint the infection.

This book focuses on heuristic-based approaches, which are the only way to combat the sophisticated and never-before-seen threats that are targeting macOS with increasing frequency. We'll write code capable of detecting anomalies and then pinpoint software that has maliciously infiltrated a system. In the process, we'll dive into the macOS operating system, touching on topics such as private frameworks, reverse engineering proprietary system components, and much more.

Of course, the heuristic-based detection approach has some downsides. While it should be able to pinpoint any malicious item on a system, it likely won't be able to identify the specific malware strain. For example, it should notice an unauthorized program surreptitiously accessing the mic or webcam, but it won't know whether the responsible process is the malware FruitFly. Is this a significant downside? I don't believe so, as the malware responsible for the infection may be unknown anyway, and you can always deploy a signature-based detection engine to cover the known basics.

Another challenge is that heuristic-based detections can suffer from false positives. For example, malware authors often leverage executable packers to obfuscate their malicious creations, but so could legitimate software developers. Thus, no heuristic-based detection approach should focus on a single heuristic when attempting to classify an item as malicious. Instead, the detection should always look for multiple anomalous behaviors and leverage approaches that reduce false positives, such as code signing information, before flagging something as suspicious or likely malicious. If you have the luxury to do so, you could enlist a human to validate any flagged items.

What You'll Find in This Book

At its core, this book describes how to write code to detect macOS malware. It's broken into three parts.

Just as a doctor performs tests and collects data to make a diagnosis, so too must malware detectors. In Part I: Data Collection, we discuss programmatic methods of collecting the data snapshots essential for detecting symptoms of infections. We'll start simple, by describing methods of enumerating and querying running processes on a system. In subsequent chapters, we'll dive into more advanced concepts, such as directly parsing binaries, extracting and validating code signing information, and uncovering persistence by interacting with proprietary system components. Where relevant, we'll show snippets of malware as examples. The chapters in this part are as follows:

Chapter 1: Examining Processes Because the majority of Mac malware specimens run as stand-alone processes, examining various information and metadata about each running process is a great place to start when seeking to uncover infections.

Chapter 2: Parsing Binaries Backing any process on a macOS system is a universal or Mach-O binary. In this chapter, we show how to parse these binaries to reveal anomalies.

Chapter 3: Code Signing Any heuristic-based detection approach is prone to false positives. By extracting and validating code signing information, as we do in this chapter, we can reduce false positives while increasing the effectiveness of any malware detection tool.

Chapter 4: Network State and Statistics This chapter describes methods of programmatically capturing snapshots of a host's network state and network statistics. Most Mac malware will access the network, and these snapshots should reveal this unauthorized network access.

Chapter 5: Persistence Malware will persist in order to survive a system reboot. Persistence causes modifications to the host, and this chapter highlights exactly how to programmatically detect these changes.

While Part I covers methods of obtaining snapshots of data, Part II: System Monitoring covers continuous approaches to monitoring a system for symptoms of an infection. For example, we'll discuss frameworks and application programming interfaces (APIs) that allow us to monitor the system logs and create powerful file, process, and network monitors. This part includes the following chapters:

Chapter 6: Log Monitoring The system, or *universal*, log contains a myriad of data that can reveal most infections. Apple doesn't provide public APIs to ingest streaming log messages, so this chapter delves into the private frameworks and APIs you can use in your own tools.

Chapter 7: Network Monitoring This chapter is dedicated to Apple's *NetworkExtension* framework, whose APIs provide the capabilities for building powerful network monitoring tools that can uncover any malware that uses the host's network.

Chapter 8: Endpoint Security If you're building comprehensive malware detection tools on macOS, you should make use of the powerful Endpoint Security framework and its APIs. This chapter introduces Endpoint Security.

Chapter 9: Muting and Authorization Events This chapter covers more advanced Endpoint Security topics, including authorization events, muting, and more.

In 2015, I founded Objective-See, which is now a nonprofit organization that makes free, open source security tools for macOS. Part III: Tool Development delves into several of Objective-See's most popular tools. Capable of generically detecting a wide range of macOS malware, these tools leverage many of the approaches covered in Parts I and II. Once you understand their design and internals, you'll be well on the way to building your own malware detection tools. We'll end the book by pitting these tools against a wide range of sophisticated macOS malware. For each specimen, we'll discuss its infection vector, methods of persistence, and capabilities and then highlight how the tools can uncover these symptoms. The chapters in this part are as follows:

Chapter 10: Persistence Enumerator Who's there? Most Mac malware persists to survive system reboots, so a tool capable of enumerating all persistent software should reveal any persistently installed malware. This chapter covers exactly such a tool: KnockKnock.

Chapter 11: Persistence Monitor Inspired by its sibling KnockKnock, BlockBlock leverages Endpoint Security to detect malware by monitoring persistence events in real time.

Chapter 12: Mic and Webcam Monitor Some of the most insidious Mac malware specimens spy on victims via the webcam or listen to them via the mic. This chapter focuses on OverSight, a tool that leverages core audio and media APIs as well as the logging subsystem to detect malware accessing these devices.

Chapter 13: DNS Monitor Malware attempting to connect to remote domains—for example, for tasking or to exfiltrate data—will generate DNS traffic. This chapter shows how DNSMonitor leverages Apple's *NetworkExtension* framework to monitor and block any unauthorized DNS traffic on a macOS host.

Chapter 14: Case Studies It's one thing to make claims about the effectiveness of security tools and quite another to back them up. In this final chapter, we pit our security tools against several notably sophisticated and stealthy malware specimens to see how they stack up.

Who Should Read This Book?

You'll get the most out of this book if you understand cybersecurity fundamentals, malware basics, and programming. These aren't prerequisites, however, and I'll explain all important concepts. You'll also find it helpful to read my other book, *The Art of Mac Malware*, Volume 1 (No Starch Press, 2022), which will introduce you to foundational macOS malware topics we won't cover again here. Beyond these considerations, I wrote this book with particular readers in mind:

Students As an undergraduate studying computer science, I had a keen interest in understanding and detecting computer viruses and yearned for a book such as this one. If you're working toward a technical degree and would like to learn more about malware detection approaches, perhaps to enhance or complement your studies, this book is for you.

Malware analysts My career as a malware analyst began at the National Security Agency, where I studied Windows-based malware and exploits that targeted US military systems. When I left the agency, I began studying macOS threats but encountered a lack of resources on the topic. This book aims to fill this gap. If you're a Windows or Linux malware analyst (or even a Mac malware analyst hoping to grow your skills), this book should provide you with insight into how to detect threats targeting macOS systems.

Mac system administrators The days of the homogeneous Windows-based enterprise have largely disappeared. Today, Macs in the enterprise are commonplace, giving rise to dedicated Mac system administrators and (unfortunately) malware authors focused on enterprise systems running macOS. If you're a Mac system administrator, it's imperative that you understand how to detect the threats targeting the systems you seek to defend. This book aims to provide such an understanding (and much more).

Developers At its core, this book presents approaches to writing code capable of generically detecting Mac malware. If your job involves writing security-focused tools for macOS, this book will be useful to you.

Even if you're not a programmer, you may find a book on the programmatic detection of malware to be worth a read. Detecting malware involves much more than just writing code. We'll delve into macOS internals, touch on reverse engineering topics, and discuss various malware specimens, including their capabilities and functionality.

The Code and Malware Specimens

You can access all code samples, malware specimens, and tools discussed in this book at *https://github.com/objective-see*. The TAOMM repository organizes code samples by chapter, and the Malware repository contains an encrypted sample of each malware specimen. Use the password *infect3d* to decrypt the samples.

WARNING *The code in the TAOMM repository is provided largely for illustrative purposes, prioritizing brevity over other aspects such as comprehensive error checking. As such, it should not be used verbatim, for example, in deployed security products. Keep in mind also that the collection in the Malware repository contains live malware. Please don't infect yourself! (Or if you do, at least don't blame me.)*

The book aims to present language-agnostic algorithms and approaches, but the majority of the code herein is written in Objective-C. I chose not to use Swift, a great language for writing Apple apps, because it poses specific challenges in the context of security tools. For example, the book often leverages private frameworks, which are easy to access in Objective-C but would require additional components, such as bridging headers, in Swift. Similarly, interfacing with frameworks that expose interfaces and APIs in C, such as the all-important Endpoint Security, is straightforward in Objective-C. Accessing these interfaces in Swift often involves a maddening amount of type-casting and unwrapping of `OpaquePointer` and `UnsafeMutablePointer` values.

I wrote all code on macOS 14 and tested it on recent versions of macOS, including 13, 14, and 15. Where relevant, I'll discuss coding approaches that diverge across versions (for example, when an older API has been replaced by a more modern counterpart). The discussion will allow you to write tools compatible with multiple versions of the operating system and ensure that you continue to support older versions. To discover any new techniques that become available as the operating system updates in the future, check out the Objective-See GitHub repositories for up-to-date versions of my open source security tools, which implement the majority of the code discussed in this book.

To help you piece together disparate parts of the larger programs presented over the course of each chapter, I've numbered the book's code listings using sequential listing numbers (such as Listing 1-1, Listing 1-2, and so on). Malware samples and command line examples won't have listing numbers.

Development Environment

Before you begin, I recommend installing Xcode, Apple's integrated development environment (IDE) and the de facto product for creating security tools on macOS. Available for free on the official Mac App Store, Xcode offers a user-friendly platform for developing software. I used Xcode to write and compile all code samples and tools in this book, so I suggest having a basic understanding of this tool. While I don't provide a detailed guide on Xcode usage here, many excellent free tutorials are available online.

Code Signing Requirements

Speaking of compiling code: if you've dabbled in software development on macOS, you've likely run into challenges related to Apple's code signing requirements or, worse, entitlements. For security reasons, Apple checks a program's code signing information before allowing it to run. (We discuss code signing in more detail in Chapter 3.)

Luckily, macOS allows code to be signed in an ad hoc manner, meaning you don't have to shell out $99 to Apple for a Developer ID if you're developing security tools that will run locally. In Xcode, under Signing and

Capabilities, check the **Automatically Manage Signing** option and make sure the Signing Certificate is set to **Sign to Run Locally**.

Entitlements

Tools that leverage system extensions or Endpoint Security require special entitlements, such as *com.apple.developer.endpoint-security.client*, to run. In Part III, we cover how to obtain these entitlements from Apple to build distributable tools. Obtaining entitlements requires a paid Developer ID account, however.

For local development and testing, you can work around entitlement requirements by disabling System Integrity Protection (SIP).[5] Apple provides documentation on how to disable SIP, which involves booting your Mac into Recovery Mode to run the command `csrutil disable`.[6]

You'll also have to disable Apple Mobile File Integrity (AMFI); otherwise, entitled binaries that aren't wholly signed and notarized won't run. With SIP disabled, you can disable AMFI by executing the following, with root privileges, from the terminal:

```
nvram boot-args="amfi_get_out_of_my_way=1"
```

Use `nvram -p` to confirm the boot arguments were set correctly. Finally, reboot.

It's worth stressing that disabling these macOS security mechanisms greatly reduces the security of the system. As such, it's best to do so only within a virtual machine or on a dedicated development test machine. To re-enable SIP in Recovery Mode, run `csrutil enable`, and to re-enable AMFI, delete the boot arguments by running `nvram -d boot-args`.

Safely Analyzing Malware

This book demonstrates many programmatic techniques for detecting Mac malware. In the book's final chapter, you can even follow along as we pit our tools against various malware specimens. If you plan to run the code snippets in the book or build and test your own tools against this malware, be sure to handle the specimens with the utmost care.

One approach to malware analysis is to use a stand-alone computer as a dedicated analysis machine. You should set up this machine in the most minimal of ways, with services such as file sharing disabled. In terms of networking, the majority of malware will require internet access to fully function (for example, to communicate with a command-and-control server for tasking), so you should connect your machine to the network in some manner. At a minimum, I recommend routing the network traffic through a VPN to hide your location from any attacker who might be on the other end.

However, leveraging a stand-alone computer for your analysis has downsides, including cost and complexity. The latter becomes especially apparent if you want to revert the analysis system to a clean baseline state (for example, to rerun a sample or when analyzing a new specimen). Although you could reinstall the operating system or, if using Apple File System (APFS), return to a baseline snapshot, these are both time-consuming endeavors.

To address these drawbacks, you can instead leverage a virtual machine for your analysis system. Various companies, such as VMware and Parallels, offer virtualized options for macOS systems. The idea is simple: virtualize a new instance of the operating system that you can isolate from your underlying environment and, most notably, revert to its original state at the click of a button. To install a new virtual machine, follow the instructions provided by each vendor. This typically involves downloading an operating system installer or updater, dragging and dropping it into the virtualization program, and then clicking through the remaining setup.

NOTE *Unfortunately, Apple Silicon systems have limitations when it comes to virtualizing macOS. Vendors such as Parallels provide prebuilt virtual machines compatible with Apple Silicon but don't yet support features such as snapshots.*

Before performing any analysis, make sure to disable any sharing between the virtual machine and the base system. For example, it would be rather unfortunate to run a ransomware sample only to find that it has also encrypted any shared files on your host system. Virtual machines also offer options for networking, such as host-only and bridged. The former will exclusively allow network connections with the host, which may be useful in various analysis situations, such as when you're setting up a local command-and-control server.

I noted that the ability to revert a virtual machine to its original state can greatly speed up malware analysis by allowing you to return to earlier stages in the analysis process. You should always take a snapshot before you begin your analysis so you can bring the virtual machine back to a known clean slate when you're done. During your analysis session, you should also make judicious use of snapshots. For example, take a snapshot immediately prior to allowing the malware to execute some core logic. If the malware fails to perform the expected action (perhaps because it detected one of your analysis tools and prematurely exited), or if your analysis tools failed to gather the data required for your analysis, simply revert to the snapshot, make any necessary changes to your analysis environment or tools, and then allow the malware to re-execute. On dedicated analysis machines or virtual machines that don't support snapshots, APFS snapshots are likely your best bet.

The main drawback to the virtual machine analysis approach is that malware may contain logic to thwart virtual machines. If the malware can successfully detect that it's being virtualized, it will often exit in an attempt to avoid continued analysis. See Chapter 9 of *The Art of Mac Malware,* Volume 1, for approaches to identifying and overcoming this logic.

For more information about setting up an analysis environment, including the specific steps for configuring an isolated virtual machine, see Phil Stokes's *How to Reverse Malware on macOS Without Getting Infected*.[7]

Additional Resources

For further reading, I recommend the following resources.

Books

The following list contains some of my favorite books on topics such as reverse engineering, macOS internals, and general malware analysis. While a few of these books are older, the core reversing and analysis topics should remain timeless.

- *Blue Fox: Arm Assembly Internals and Reverse Engineering* by Maria Markstedter (Wiley, 2023)
- *x86 Software Reverse-Engineering, Cracking, and Counter-Measures* by Stephanie and Christopher Domas (Wiley, 2024)
- The *macOS/iOS (*OS) Internals* trilogy by Jonathan Levin (Technologeeks Press, 2017)
- *The Art of Computer Virus Research and Defense* by Péter Ször (Addison-Wesley Professional, 2005)
- *Reversing: Secrets of Reverse Engineering* by Eldad Eilam (Wiley, 2005)
- *OS X Incident Response: Scripting and Analysis* by Jaron Bradley (Syngress, 2016)

Websites

There used to be a dearth of information about Mac malware analysis online. Today, the situation has greatly improved. Several websites collect information on this topic, and blogs such as my very own on the Objective-See website are dedicated to Mac security topics. The following is an inexhaustive list of some of my favorites:

- *https://papers.put.as*: A fairly exhaustive archive of papers and presentations on macOS security topics and malware analysis
- *https://themittenmac.com*: The website of the noted macOS security researcher and author Jaron Bradley that includes incident response tools and threat-hunting knowledge for macOS
- *https://objective-see.org/blog.html*: My blog, which for the last decade has published my research and that of fellow security researchers on the topics of macOS malware, exploits, and more

Notes

1. "Worldwide PC Shipments Decline Another 15.0% in the Third Quarter of 2022, According to IDC Tracker," *Business Wire*, October 9, 2022, *https://www.businesswire.com/news/home/20221009005049/en/Worldwide-PC-Shipments-Decline-Another-15.0-in-the-Third-Quarter-of-2022-According-to-IDC-Tracker.*

2. "Jamf Q3 Data Confirms Rapid Mac Adoption Across the Enterprise," *Computer World*, November 11, 2022, *https://www.computerworld.com/article/3679730/jamf-q3-data-confirms-rapid-mac-adoption-across-the-enterprise.html*.
3. "Malwarebytes Finds Mac Threats Outpace Windows for the First Time in Latest State of Malware Report," *Malwarebytes*, February 11, 2020, *https://www.malwarebytes.com/press/2020/02/11/malwarebytes-finds-mac-threats-outpace-windows-for-the-first-time-in-latest-state-of-malware-report*.
4. US Department of Justice, Office of Public Affairs, "Ohio Computer Programmer Indicted for Infecting Thousands of Computers with Malicious Software and Gaining Access to Victims' Communications and Personal Information," press release no. 18-21, January 10, 2018, *https://www.justice.gov/opa/pr/ohio-computer-programmer-indicted-infecting-thousands-computers-malicious-software-and*.
5. "System Extensions and DriverKit," Apple, accessed May 25, 2024, *https://developer.apple.com/system-extensions/*.
6. "Disabling and Enabling System Integrity Protection," Apple, accessed May 25, 2024, *https://developer.apple.com/documentation/security/disabling_and_enabling_system_integrity_protection?language=objc*.
7. Phil Stokes, *How to Reverse Malware on macOS Without Getting Infected*, August 14, 2019, *https://go.sentinelone.com/rs/327-MNM-087/images/reverse_mw_final_9.pdf*.