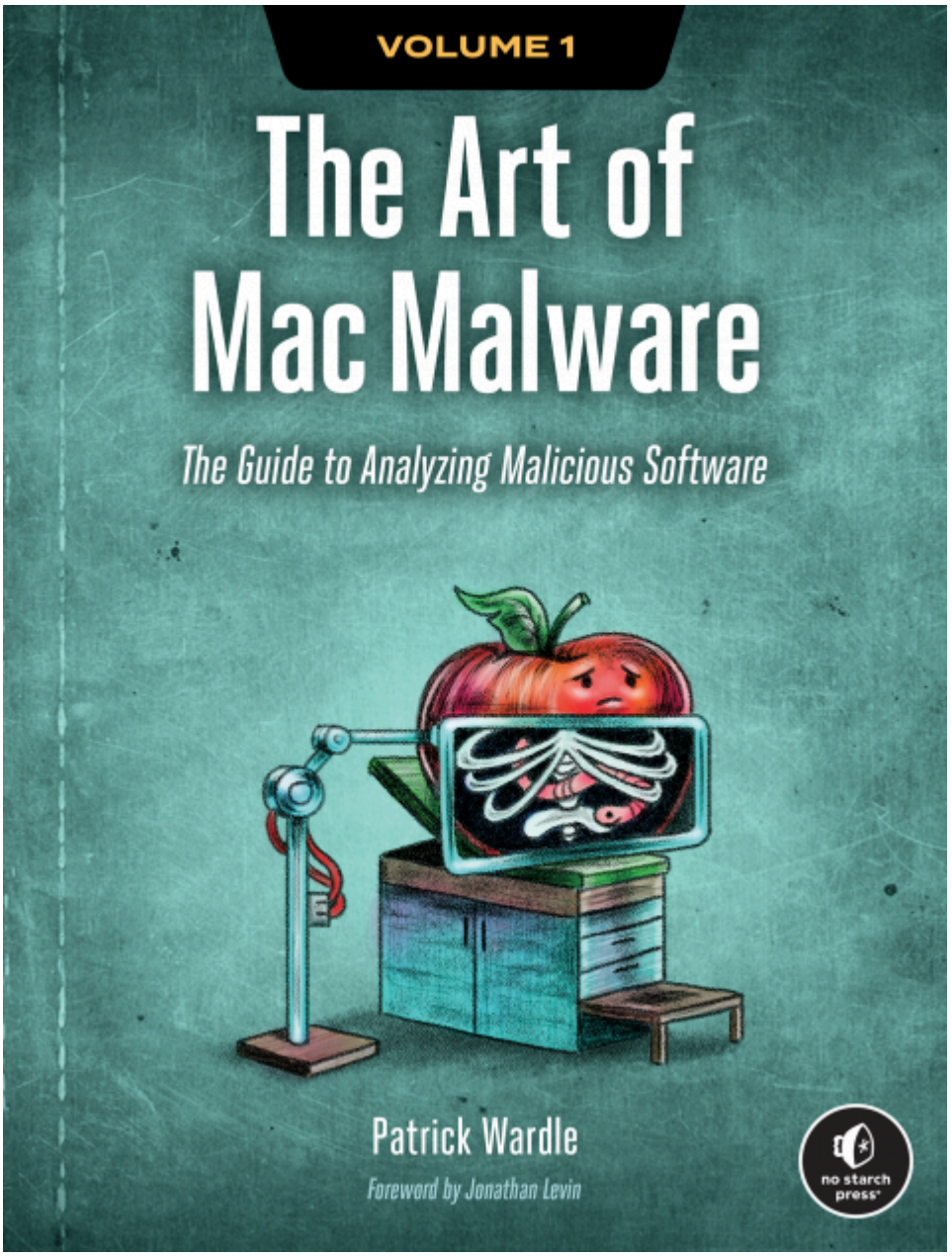
The Art of Mac Malware

Archived: 2026-04-05 17:23:17 UTC

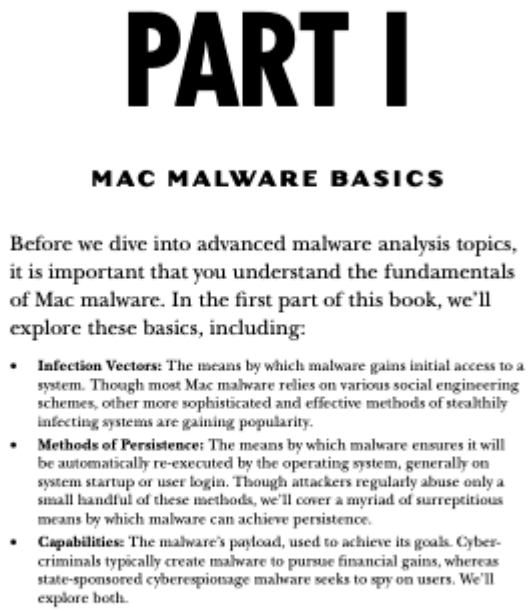
Defenders must understand how malware works to counter threats targeting Apple products. This volume explores infection methods, persistence mechanisms, and reverse-engineering techniques to analyze malicious code. Front Matter



-
- The front matter of the book includes a table of contents, acknowledgements, introduction and a forward (by the noted macOS researcher/author Jonathan Levin).

Read Chapter »

Part I: Mac Malware Basics



PART I

MAC MALWARE BASICS

Before we dive into advanced malware analysis topics, it is important that you understand the fundamentals of Mac malware. In the first part of this book, we'll explore these basics, including:

- **Infection Vectors:** The means by which malware gains initial access to a system. Though most Mac malware relies on various social engineering schemes, other more sophisticated and effective methods of stealthily infecting systems are gaining popularity.
- **Methods of Persistence:** The means by which malware ensures it will be automatically re-executed by the operating system, generally on system startup or user login. Though attackers regularly abuse only a small handful of these methods, we'll cover a myriad of surreptitious means by which malware can achieve persistence.
- **Capabilities:** The malware's payload, used to achieve its goals. Cybercriminals typically create malware to pursue financial gains, whereas state-sponsored cyberespionage malware seeks to spy on users. We'll explore both.

-
-

  Before we dive into advanced malware analysis topics, it is important that you understand the fundamentals of Mac malware.

  In the first part of this book, we'll explore these basics, including: infection vectors, methods of persistence, and capabilities.

  Read Chapter »

Chapter 1: Infection Vectors

1

INFECTION VECTORS

A malware's infection vector is the means by which it gains access to a system. Throughout the years, malware authors have relied on mechanisms ranging from simple social engineering tricks to advanced, remote zero-day exploits to infect Macs. In this chapter, we'll discuss many of the most common techniques used by Mac malware authors.

By far the most popular method of infecting Macs with malicious code involves tricking users into infecting themselves, generally by directly downloading and running the malicious code. (By contrast, techniques like remote exploitation are far less prevalent.) To achieve this, attackers often make use of common social engineering attacks, including tech-support scams, disseminating fake updates, fake applications, trojanized applications, and infected pirated applications.

Apple, of course, is keenly aware of macOS infection trends and the fact that the majority of such infections require explicit user interaction in

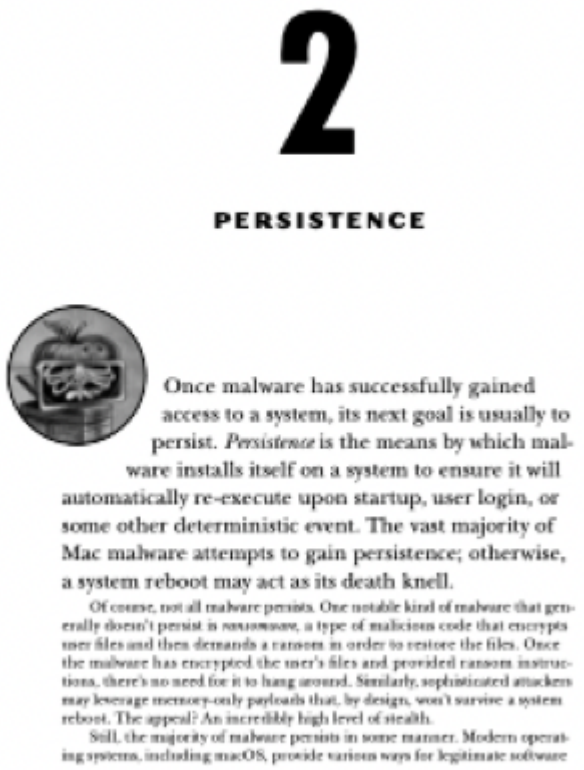
-
-

A malware's infection vector is the means by which it gains access to a system. Throughout the years, malware authors have relied on mechanisms ranging from simple social engineering tricks to advanced, remote zero-day exploits to infect Macs.

In this chapter, we'll discuss many of the most common techniques used by Mac malware authors.

Read Chapter »

Chapter 2: Persistence


2

PERSISTENCE

Once malware has successfully gained access to a system, its next goal is usually to persist. *Persistence* is the means by which malware installs itself on a system to ensure it will automatically re-execute upon startup, user login, or some other deterministic event. The vast majority of Mac malware attempts to gain persistence; otherwise, a system reboot may act as its death knell.

Of course, not all malware persists. One notable kind of malware that generally doesn't persist is *ransomware*, a type of malicious code that encrypts user files and then demands a ransom in order to restore the files. Once the malware has encrypted the user's files and provided ransom instructions, there's no need for it to hang around. Similarly, sophisticated attackers may leverage memory-only payloads that, by design, won't survive a system reboot. The appeal? An incredibly high level of stealth.

Still, the majority of malware persists in some manner. Modern operating systems, including macOS, provide various ways for legitimate software

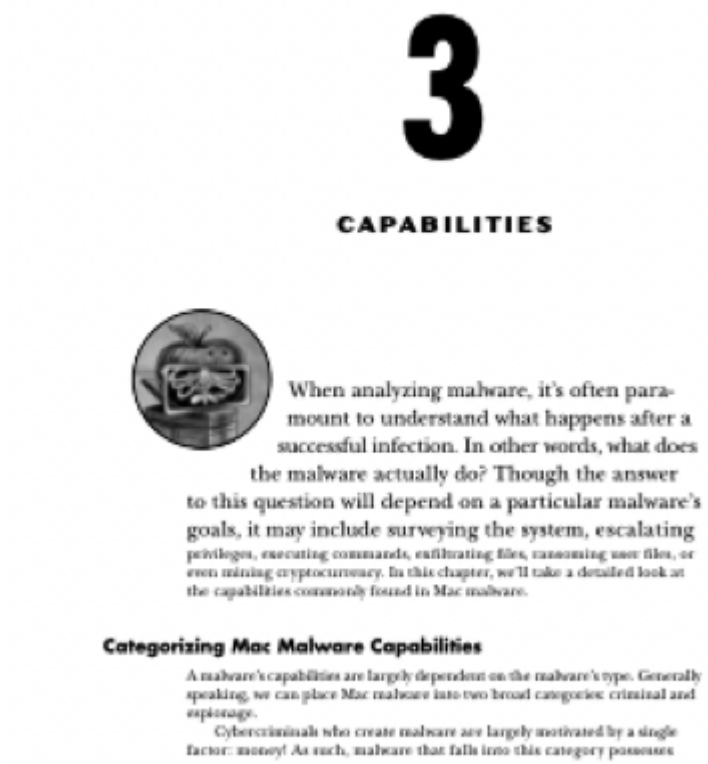
-
-

Once malware has successfully gained access to a system, its next goal is usually to persist. Persistence is the means by which malware installs itself on a system to ensure it will automatically re-execute upon startup, user login, or some other deterministic event.

Though attackers regularly abuse only a small handful of these methods, we'll cover a myriad of surreptitious means by which malware can achieve persistence.

Read Chapter »

Chapter 3: Capabilities



3

CAPABILITIES

When analyzing malware, it's often paramount to understand what happens after a successful infection. In other words, what does the malware actually do? Though the answer to this question will depend on a particular malware's goals, it may include surveying the system, escalating privileges, executing commands, exfiltrating files, ransoming user files, or even mining cryptocurrency. In this chapter, we'll take a detailed look at the capabilities commonly found in Mac malware.

Categorizing Mac Malware Capabilities

A malware's capabilities are largely dependent on the malware's type. Generally speaking, we can place Mac malware into two broad categories: criminal and espionage.

Cybercriminals who create malware are largely motivated by a single factor: money! As such, malware that falls into this category possesses

-
-

When analyzing malware, it's often paramount to understand what happens after a successful infection. In other words, what does the malware actually do? Though the answer to this question will depend on a particular malware's goals, it may include surveying the system, escalating privileges, executing commands, exfiltrating files, ransoming user files, or even mining cryptocurrency.

In this chapter, we'll take a detailed look at the capabilities commonly found in Mac malware.

Read Chapter »

Part II: Mac Malware Analysis

PART II

MAC MALWARE ANALYSIS

Now that you understand Mac malware's infection vectors, persistence mechanisms, and capabilities, let's discuss how you can effectively analyze malicious samples. We'll cover both static and dynamic approaches:

- **Static Analysis:** The examination of a sample without executing it. This approach leverages various tools that can statically extract information from a sample. Often, the analysis culminates with the use of a disassembler or decompiler.
- **Dynamic Analysis:** The examination of a sample during its execution. This approach most commonly leverages passive monitoring tools, though it might employ more powerful tools, such as a debugger, as well.

Using these analysis techniques, we'll determine whether a sample is indeed malicious and, if so, answer questions such as the following: What infection vector does it use to infect a Mac? What, if any, persistence mechanism is used to maintain access? What are its ultimate objectives and capabilities?

With the answers to these questions, we can determine exactly what threat the malware poses to Mac users, as well as create detection, prevention, and disinfection mechanisms to thwart it.

-
-

Now that you understand Mac malware's infection vectors, persistence mechanisms, and capabilities, let's discuss how you can effectively analyze malicious samples.

We’ll cover both static and dynamic approaches:

Read Chapter »

Chapter 4: Nonbinary Analysis

-
-

  This chapter focuses on the static analysis of "nonbinary" file formats, such as packages, disk images, and scripts, that you'll commonly encounter while analyzing Mac malware.

  Read Chapter »

Chapter 5: Binary Triage


5

BINARY TRIAGE

In the last chapter, I introduced static analysis tools and techniques and applied them to various nonbinary file formats, such as distribution mediums and scripts. In this chapter, we'll continue our discussion of static analysis by focusing on Apple's native executable file format, the venerable Mach object file format (Mach-O). As the majority of Mac malware is compiled into Mach-Os, all Mac malware analysts should understand the structure of these binaries, as at a minimum, this will allow you to differentiate the benign from the malicious.

The Mach-O File Format

Like with all binary file formats, analyzing and understanding Mach-O files requires specific analysis tools, often culminating in the use of a binary disassembler. Executable binary file formats are rather complex, and the Mach-O is no exception. The good news is that you'll need only an elementary understanding of the format, as well as a few related concepts,

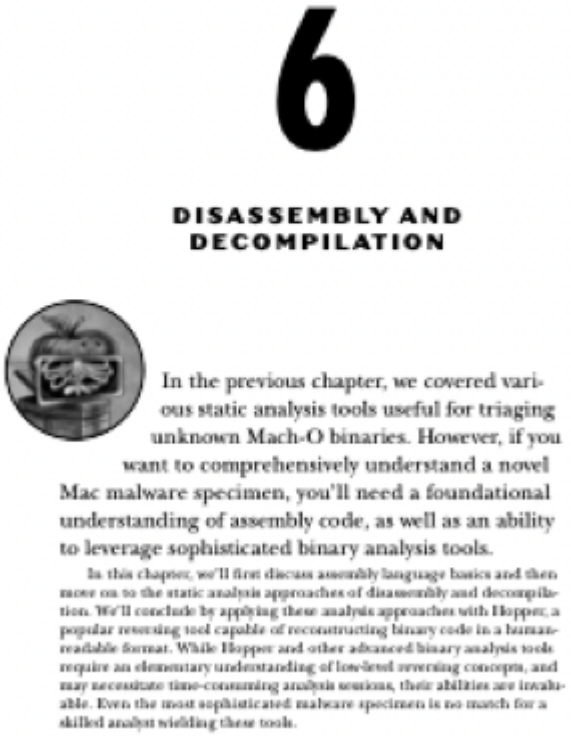
-
-

In this chapter, we'll continue our discussion of static analysis by focusing on Apple's native executable file format, the venerable Mach object file format (Mach-O).

As the majority of Mac malware is compiled into Mach-Os, all Mac malware analysts should understand the structure of these binaries, as at a minimum, this will allow you to differentiate the benign from the malicious.

Read Chapter »

Chapter 6: Disassembly and Decompilation



-
-

  If you want to comprehensively understand a novel Mac malware specimen, you'll need a foundational understanding of assembly code, as well as an ability to leverage sophisticated binary analysis tools.

  In this chapter, we'll first discuss assembly language basics and then move on to the static analysis approaches of disassembly and decompilation. We'll conclude by applying these analysis approaches with Hopper, a popular reversing tool capable of reconstructing binary code in a human- readable format.

  Read Chapter »

Chapter 7: Dynamic Analysis Tools


7

DYNAMIC ANALYSIS TOOLS

In the previous chapters, we discussed methods of static analysis used to examine files without actually running them. Often, however, it may be more efficient to simply execute a malicious file to passively observe its behavior and actions. This is especially true when malware authors have implemented mechanisms designed specifically to complicate or even thwart static analysis, such as encrypting embedded strings and configuration information or dynamically loading more code at runtime.

WindTail provides an illustrative example. The addresses of its command and control servers (generally something a malware analyst would seek to uncover) are embedded directly within the malware but encrypted. It is possible to manually decode these encrypted addresses, as the encryption key is hardcoded within the malware. However, it is far easier to simply execute the malware. Then, using a dynamic analysis tool such as a network

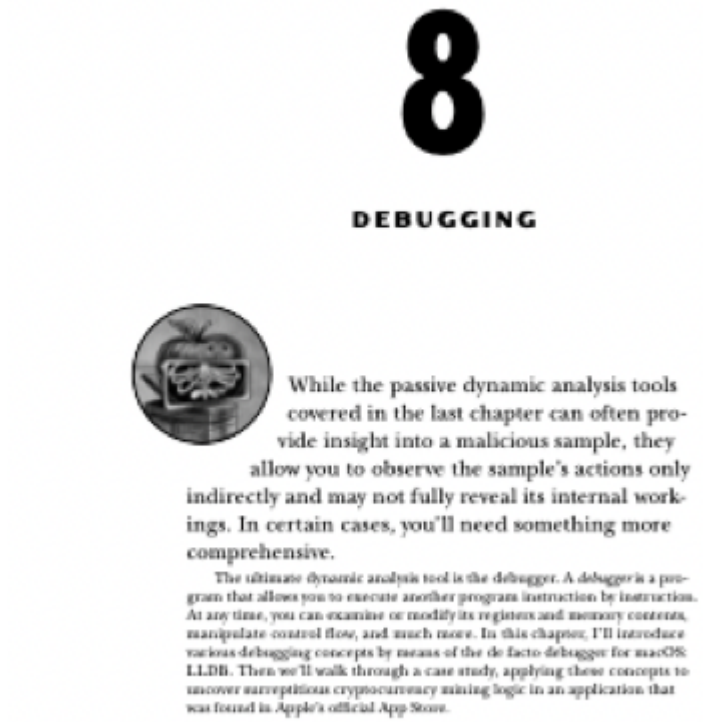
-
-

  When analyzing a malicious sample, it may be more efficient to simply execute the item and passively observe its behavior and actions.

  This is especially true when malware authors have implemented mechanisms designed specifically to complicate or even thwart static analysis, such as encrypting embedded strings and configuration information or dynamically loading more code at runtime.

  Read Chapter »

Chapter 8: Debugging



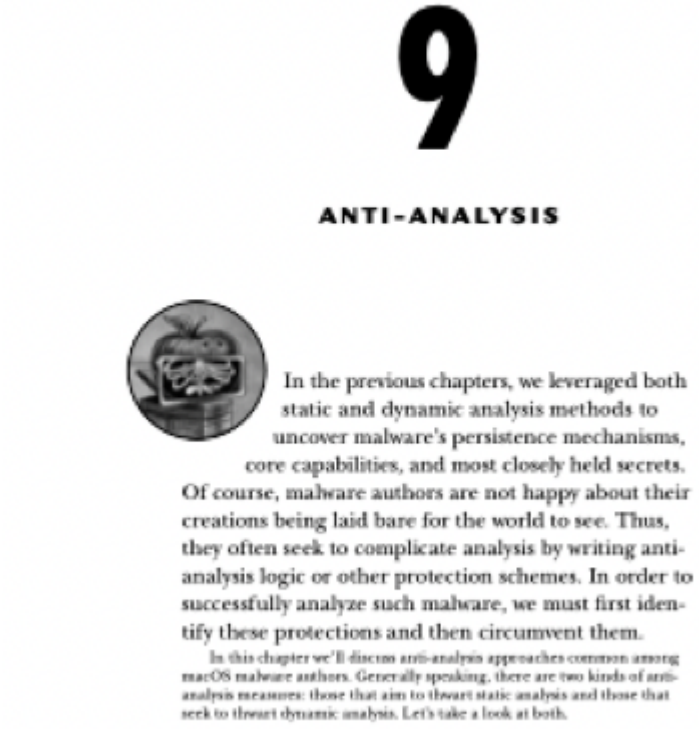
-
-

  The ultimate dynamic analysis tool is the debugger. A debugger is a program that allows you to execute another program instruction by instruction. At any time, you can examine or modify its registers and memory contents, manipulate control flow, and much more.

  In this chapter, I'll introduce various debugging concepts by means of the de facto debugger for macOS: LLDB.

  Read Chapter »

Chapter 9: Anti-Analysis



-
-

  In the previous chapters, we leveraged both static and dynamic analysis methods to uncover malware's persistence mechanisms, core capabilities, and most closely held secrets. Of course, malware authors are not happy about their creations being laid bare for the world to see. Thus, they often seek to complicate analysis by writing anti- analysis logic or other protection schemes.

  In this chapter we'll discuss anti-analysis approaches common among macOS malware authors.

  Read Chapter »

Part III: Analyzing EvilQuest

PART III

ANALYZING EVILQUEST

It's time to put the universal adage "practice makes perfect" into, well, practice. In Part III of this book, you'll apply all that you've learned in Parts I and II to thoroughly analyze the intriguing Mac malware specimen known as EvilQuest. Discovered in the summer of 2020, this malware appeared at first blush to be little more than a run-of-the-mill piece of ransomware. However, further analysis uncovered something far more sophisticated.

You'll get the most out of this section by following along and performing the analysis with me. First, make sure you've created a safe analysis environment; return to this book's introduction for guidelines on doing so. Then download the EvilQuest specimen from Objective-See's Mac malware collection at https://objective-see.com/downloads/malware/EvilQuest.zip. Use the password infect3d to decrypt the malicious sample.

Ready to dive in together? Let's go!

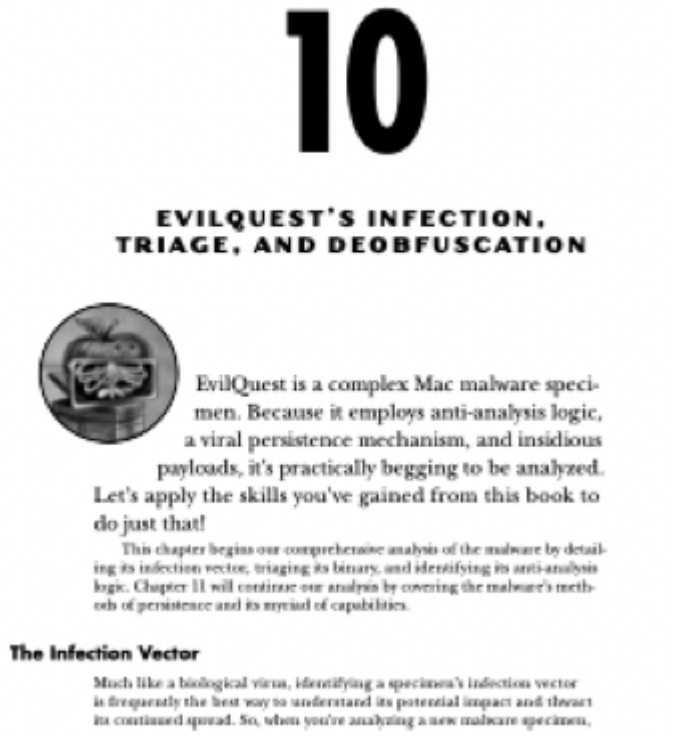
-
-

It's time to put the universal adage "practice makes perfect" into, well, practice.

In Part III of this book, you'll apply all that you've learned in Parts I and II to thoroughly analyze the intriguing Mac malware specimen known as EvilQuest.

Read Chapter »

Chapter 10: EvilQuest's Infection, Triage, and Deobfuscation


10

EVILQUEST'S INFECTION, TRIAGE, AND DEOBFUSCATION

EvilQuest is a complex Mac malware specimen. Because it employs anti-analysis logic, a viral persistence mechanism, and insidious payloads, it's practically begging to be analyzed. Let's apply the skills you've gained from this book to do just that!

This chapter begins our comprehensive analysis of the malware by detailing its infection vector, triaging its binary, and identifying its anti-analysis logic. Chapter 11 will continue our analysis by covering the malware's methods of persistence and its myriad of capabilities.

The Infection Vector

Much like a biological virus, identifying a specimen's infection vector is frequently the best way to understand its potential impact and thwart its continued spread. So, when you're analyzing a new malware specimen,

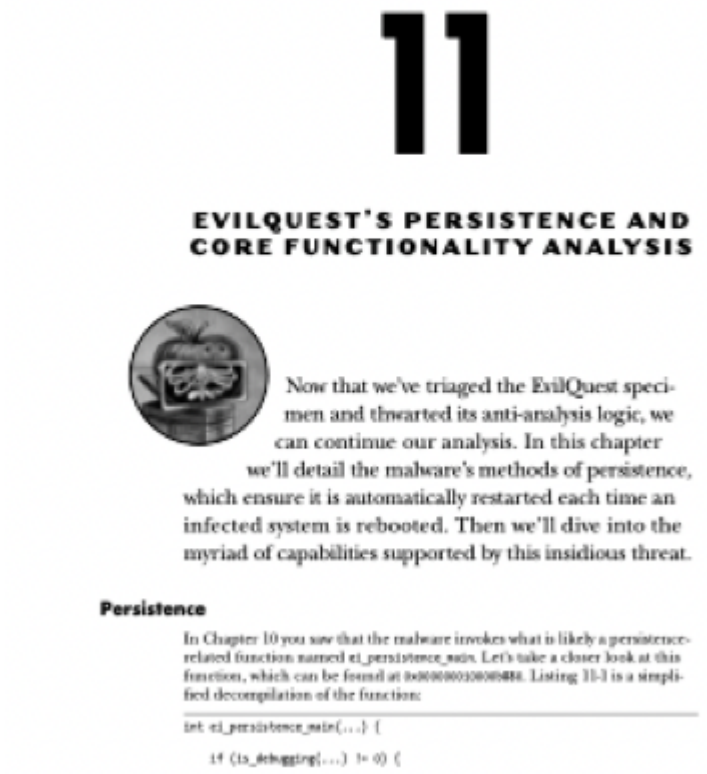
-
-

EvilQuest is a complex Mac malware specimen. Because it employs anti-analysis logic, a viral persistence mechanism, and insidious payloads, it's practically begging to be analyzed. Let's apply the skills you've gained from this book to do just that!

This chapter begins our comprehensive analysis of the malware by detail- ing its infection vector, triaging its binary, and identifying its anti-analysis logic.

Read Chapter »

Chapter 11: EvilQuest's Persistence and Core Functionality


11

EVILQUEST'S PERSISTENCE AND CORE FUNCTIONALITY ANALYSIS

Now that we've triaged the EvilQuest specimen and thwarted its anti-analysis logic, we can continue our analysis. In this chapter we'll detail the malware's methods of persistence, which ensure it is automatically restarted each time an infected system is rebooted. Then we'll dive into the myriad of capabilities supported by this insidious threat.

Persistence

In Chapter 10 you saw that the malware invokes what is likely a persistence-related function named ei_persistence_main. Let's take a closer look at this function, which can be found at 0x0000001000000. Listing 11-1 is a simplified decompilation of the function:

```
int ei_persistence_main(...) {
    if (is_debugging(...) != 0) {
```

-
-

Now that we've triaged the EvilQuest specimen and thwarted its anti-analysis logic, we can continue our analysis.

In this chapter we'll detail the malware's methods of persistence, which ensure it is automatically restarted each time an infected system is rebooted. Then we'll dive into the myriad of capabilities supported by this insidious threat.

Read Chapter »

End Matter

-

- The book's end matters contains its index, and more.

  Read Chapter »

Source: https://taomm.org/vol1/read.html