An Analysis of macOS Security Features

Jack Clark

Ethical Hacking 3 - Mini Project
BSc Ethical Hacking
Abertay University
Dundee, United Kingdom
1601798@abertay.ac.uk

7 May 2019

Abstract

This paper documents macOS security features that have been added throughout the development of macOS. It covers the hardware that has been implemented to further this security and software updates to the most recent version, macOS Mojave (10.14). It discusses how these features are implemented protect the end user's data to a high level of security.

Contents

List of Acronyms

AES	Advanced Encryption Standard.
APFS	Apple File System.
EFI	Extensible Firmware Interface.
FDE	Full Disk Encryption.
HFS	Hierarchical File System.
JIT	Just-In-Time.
MRT	Malware Removal Tool.
OS	Operating System.
ROM	Read-Only Memory.
SBC	Secure Boot Chain.
SE	Secure Enclave.
SIP	System Integrity Protection.
T2	T2 Coprocessor.
UID	Unique Identifier.
VM	Virtual Machine.

1 Introduction

macOS is a UNIX based OS developed by Apple to operate it's line-up of Mac devices. Initially developed back in XXX, macOS was created for ease of use for the end user with a heavy influence on security and privacy.

1.1 Background

A common myth regarding macOS is that it doesn't "get any viruses". This is commonly believed due to it being based on UNIX, and to some vague extent is true. macOS does get malware, however, it is a lot less than that of other major operating systems, for example, Windows. Based on last year, 2018 saw 67.69 million malicious programs against Windows devices, in comparison to 93,403 for macOS (AV-TEST 2019). This figure shows the vast difference but is also to be expected as macOS currently holds a 10.65% market share (NetMarketShare 2019).

Even though macOS has significantly less malicious applications in the wild, the number has exponentially increased. As can be seen in Figure 1 below, malware for macOS is on a steep climb upwards.

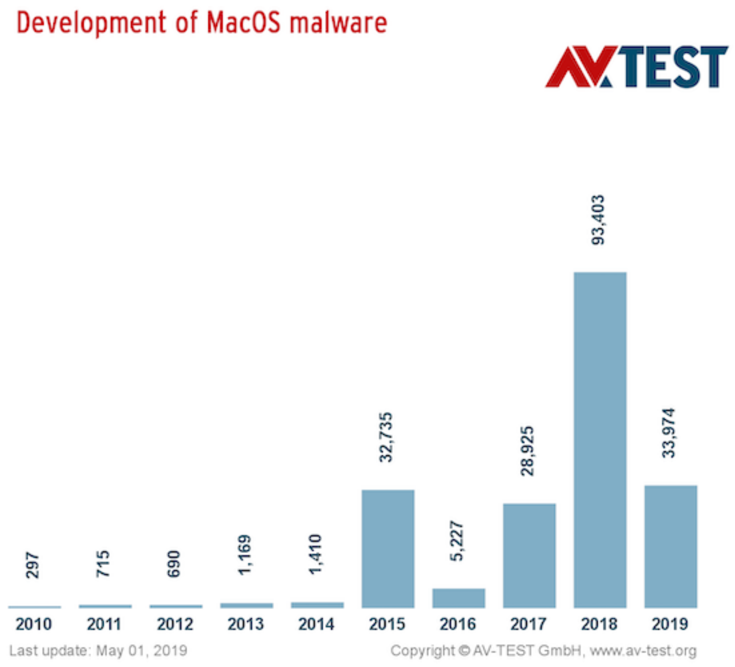


Figure 1: macOS Malware Growth Per Year (AV-TEST 2019)

Due to the large increase of macOS malware, Apple have recently been upgrading the security of macOS and their devices throughout the years. This allows users to have a safer experience while using their devices knowing that they are protected continuously on the Operating System (OS) and system level.

Since the first development of macOS, the current version is macOS 10.14, Mojave, and it featured several privacy and security updates. The most significant security updates to come to macOS since its creation include System Integrity Protection (SIP), FileVault, Application Sandbox and the quarantine system. Along with software security features, Apple released a new model of Mac devices that feature the T2 coprocessor, used primarily for secure operations.

1.2 Aims

This paper aims to discuss in detail the previously mentioned security features that have been implemented into macOS, including hardware features, and how they operate. It also aims to discuss how these features will work to improve the security of the end users data.

2 macOS Security

As previously mentioned, macOS security is a priority for Apple along with ease of use. Since the beginning of macOS, major security features have been released with each iteration, upgrading the security further each time. The most significant of these updates are listed below grouped by macOS version.

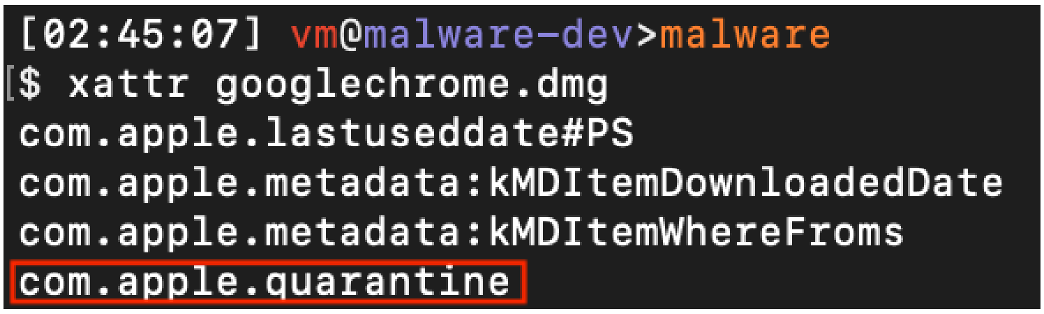
Please note, unless explicitly stated, any examples shown are conducted in a Virtual Machine (VM) with macOS Mojave (10.14).

At the time of writing, the current version of macOS is 10.14.4. *The following information is correct at the time of writing, however giving that updates release regularly may be incorrect at the time of reading.*

2.1 Snow Leopard (10.6)

2.1.1 XProtect

Introduced in OSX 10.6, XProtect was one of the first major security features that were released for macOS. XProtect, commonly named "safe downloads list", is part of the quarantine system used for the detection of potentially malicious or dangerous applications or files, such as images and PDF's, that have been download from the internet or another means. Whenever an application or file is downloaded, the quarantine flag is set on it to mark it as potentially malicious. The quarantine flag can be checked on a file by using the `xattr` command (Oakley 2016):

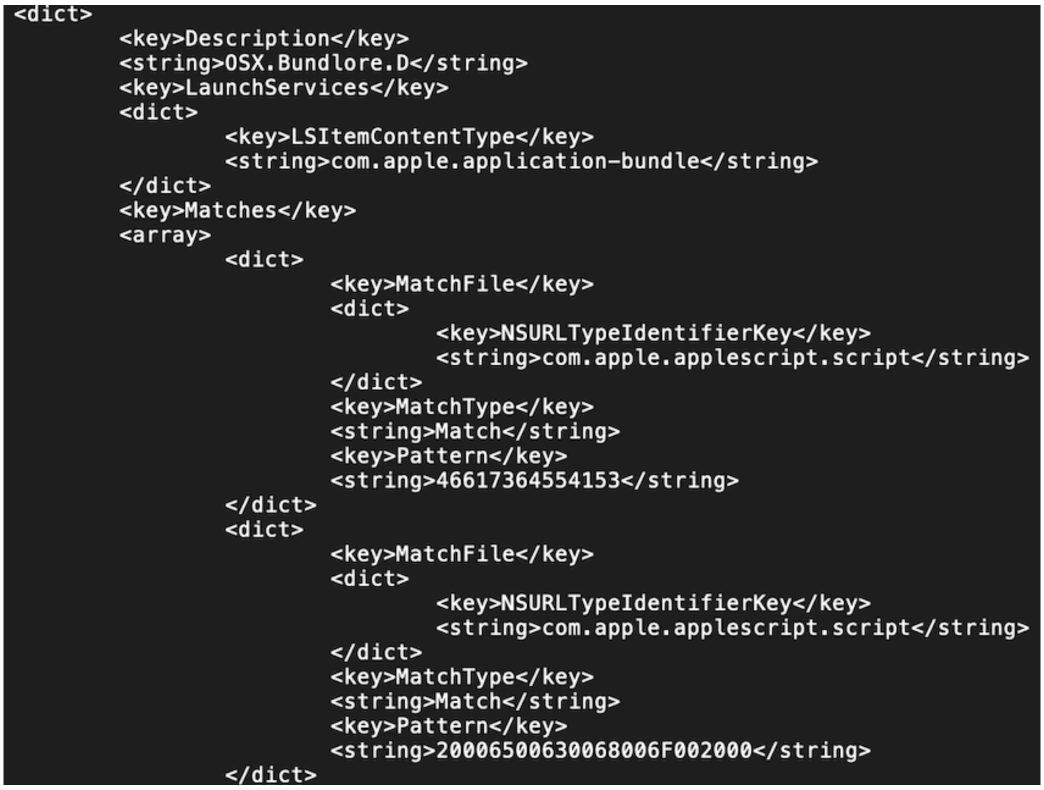


```
[02:45:07] vm@malware-dev>malware
[$ xattr googlechrome.dmg
com.apple.lastuseddate#PS
com.apple.metadata:kMDItemDownloadedDate
com.apple.metadata:kMDItemWhereFroms
com.apple.quarantine
```

Figure 2: `xattr` Example, Flag Highlighted in Red

The `com.apple.quarantine` flag is also used by Gatekeeper (discussed in section 2.3.1) which implements XProtect further.

When a quarantined file is attempted to be loaded or executed, the `.app`, for example, is compared to a `plist` file stored at `/System/Library/CoreServices/XProtect.bundle/Contents/Resources/XProtect.meta.plist` and the user is also made aware that the file was downloaded from the internet. An example of the latest `Xprotect.meta.plist` file is shown in Figure 3 below.

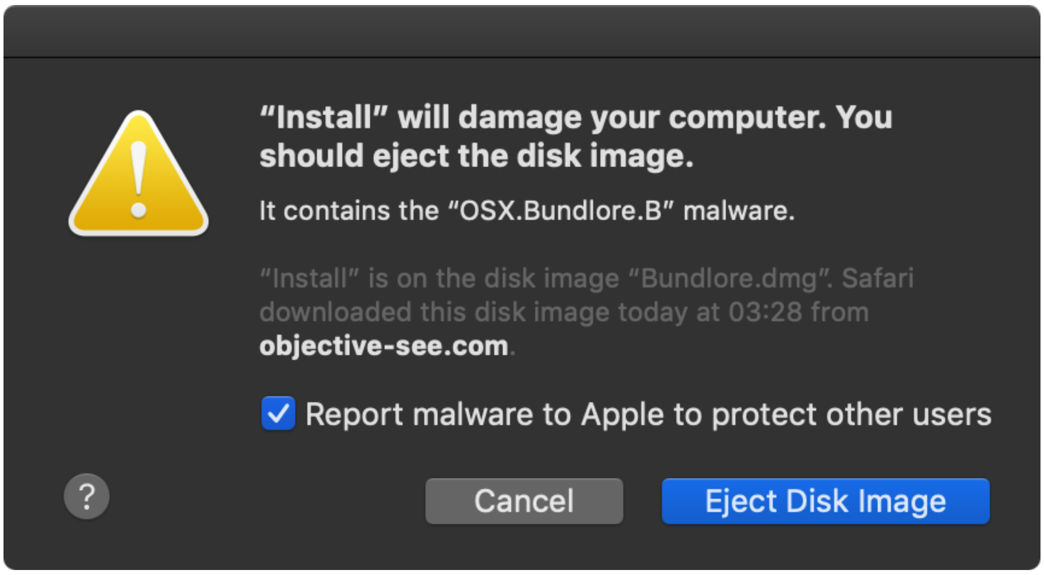


```
<dict>
        <key>Description</key>
        <string>OSX.Bundlore.D</string>
        <key>LaunchServices</key>
        <dict>
                <key>LSItemContentType</key>
                <string>com.apple.application-bundle</string>
        </dict>
        <key>Matches</key>
        <array>
                <dict>
                        <key>MatchFile</key>
                        <dict>
                                <key>NSURLTypeIdentifierKey</key>
                                <string>com.apple.applescript.script</string>
                        </dict>
                        <key>MatchType</key>
                        <string>Match</string>
                        <key>Pattern</key>
                        <string>46617364554153</string>
                </dict>
                <dict>
                        <key>MatchFile</key>
                        <dict>
                                <key>NSURLTypeIdentifierKey</key>
                                <string>com.apple.applescript.script</string>
                        </dict>
                        <key>MatchType</key>
                        <string>Match</string>
                        <key>Pattern</key>
                        <string>20006500630068006F002000</string>
                </dict>
```

Figure 3: `XProtect.meta.plist` Example

The above figure shows the partial entry for a macOS malware named 'Bundlore'[1]. When this malware is downloaded, XProtect will compare it to the `plist`. When this file is found by XProtect, the user is made aware and the following prompt is given:

[1]This malware was downloaded and executed within a VM. This provided a safe and secure environment for testing. The malware was retrieved from (Wardle 2019).


"Install" will damage your computer. You should eject the disk image.

It contains the "OSX.Bundlore.B" malware.

"Install" is on the disk image "Bundlore.dmg". Safari downloaded this disk image today at 03:28 from **objective-see.com**.

Figure 4: XProtect Detection

As can be seen, there is no way for the user to continue executing the application. A downside to XProtect is that it relies on Apple releasing an update to the blacklist and will only check the application when it is initially downloaded and installed, if an application is already installed before it is blacklisted then this will not be flagged as malicious.

2.1.2 Malware Removal Tool

To counter the issue with XProtect only stopping malware at the point of installation, Malware Removal Tool (MRT) is used as part of the quarantine system implemented in macOS. With XProtect protecting users at the point of installation, MRT will remove malware that has been deemed as malicious by Apple post-install, by similarly using a blacklist like XProtect.

MRT runs whenever it receives an update from Apple, at which point it will scan the device for any of the malware that is blacklisted and flag it for removal on the next restart of the device. This operates more commonly than XProtect however it still relies on blacklist updates from Apple meaning that if it is a lesser known malware, then it is unlikely to be removed by MRT automatically (Oakley 2018b).

Currently, the latest versions of XProtect and MRT are 2103 and 1.41.

2.2 Lion (10.7)

2.2.1 FileVault

Firstly released in macOS 10.3, FileVault was an encryption utility built directly into macOS to encrypt users home directories, providing protection to their documents. FileVault operated by encrypting and decrypting the user's data on-the-fly, ensuring that their information was always protected. (Trouton 2012)

Since then FileVault, now known as FileVault 2[2], has been completely redesigned in Lion. FileVault now offers Full Disk Encryption (FDE), meaning that the entire primary drive is encrypted. This is also encrypted at rest meaning that if an attacker were to retrieve a powered off/locked device, then there is no way to retrieve the data from it. Again, FileVault employs a decrypt on-the-fly policy. Apple have also updated the cipher used for encryption to AES-XTS, allowing for even stronger encryption but also a more seamless experience as the decryption/encryption process can be offloaded to the T2 Chip, if present. (Apple 2018b)

2.2.2 App Sandboxing

An additional feature that Apple introduced in macOS 10.7 was app sandboxing. This was developed from the implementation in iOS, and now runs every application from the App Store, or downloaded from the internet with the `app-sandbox` entitlement set, in its own sandbox enforced from the kernel level.

A sandbox allows for a program to be executed in its own private area of memory that is locked down from any other applications affecting it and vice versa. This means that if an application were to be vulnerable to a buffer overflow attack, for example, then if it were exploited it would only crash the app meanwhile the OS and other applications would remain operating. While an application is sandboxed, it has to request to access other data or hardware, for example, a microphone, and these requests must be included during development meaning that the application must be developed securely from the beginning (GeoSn0w 2018).

[2]This will be referred to as simply FileVault in the remainder of the paper

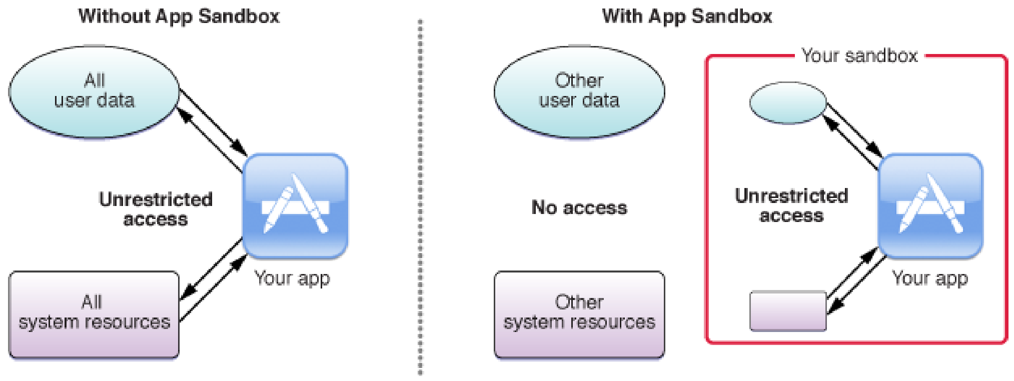
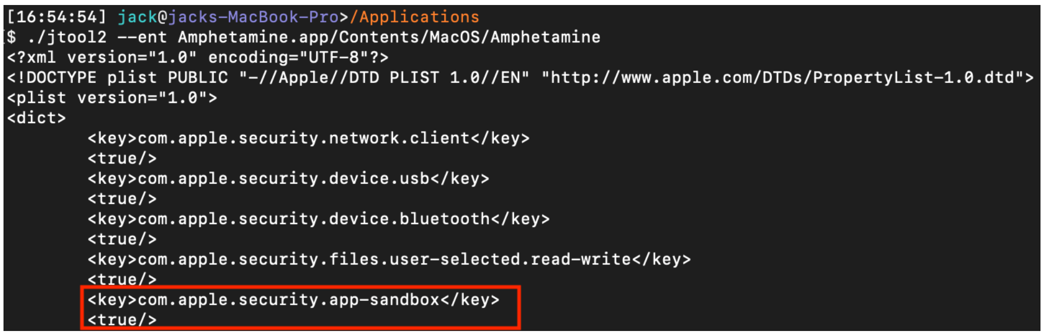


Figure 5: Sandbox Example, (Apple 2016)

As can be seen in Figure 5, the sandboxed application is given unrestricted access to only the requested items, providing greater security for the user and resilience to attacks for the OS. To check if an app has the `app-sandbox` entitlement, the tool `jtool2` can be used:



```
[16:54:54] jack@jacks-MacBook-Pro>/Applications
$ ./jtool2 --ent Amphetamine.app/Contents/MacOS/Amphetamine
<?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>com.apple.security.network.client</key>
        <true/>
        <key>com.apple.security.device.usb</key>
        <true/>
        <key>com.apple.security.device.bluetooth</key>
        <true/>
        <key>com.apple.security.files.user-selected.read-write</key>
        <true/>
        <key>com.apple.security.app-sandbox</key>
        <true/>
```

Figure 6: `jtool2` output, (Levin 2019)

In the above figure, highlighted with a red outline is the entitlement with the `true` flag set. This means that this application will be sandboxed when executed.

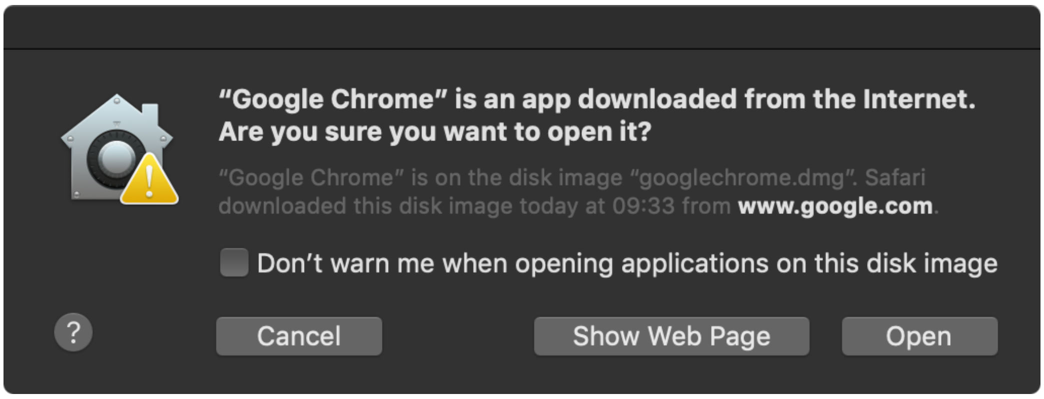
It is recommended that all applications are sandboxed, for obvious reasons such as increasing both the users and macOS security, however, this introduces barriers for app developers as there is no free-reign in terms of what their app can access.

2.3 Mountain Lion (10.8)

2.3.1 Gatekeeper

The final feature of the quarantine system that works hand in hand with XProtext and MRT is Gatekeeper. Introduced in 10.8 Gatekeeper is typically the most notable and known feature that assists in malware prevention.

Whenever an application or file, is downloaded (this includes throuhg AirDrop) the quarantine flag is set on the application bundle. When it is executed for the first time, Gatekeeper steps in before XProtect. Gatekeeper checks for code signing of the application bundle, which if validly signed will include the ID of the developer and also promises integrity of the bundle. The Gatekeeper prompt when executing an application is shown below in figure 7 (Oakley 2018b):

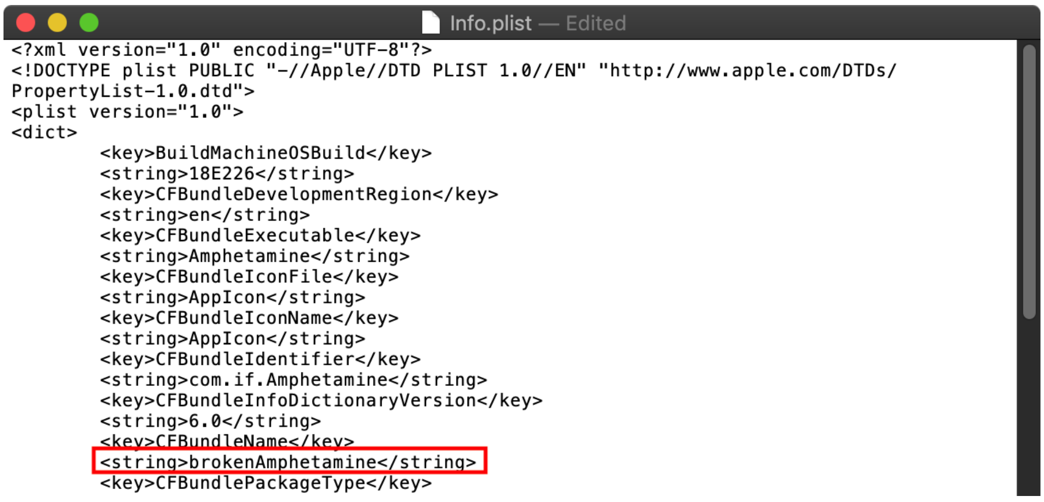

"Google Chrome" is an app downloaded from the Internet. Are you sure you want to open it?

"Google Chrome" is on the disk image "googlechrome.dmg". Safari downloaded this disk image today at 09:33 from **www.google.com**.

Figure 7: Gatekeeper Prompt on First Execution

As shown above, Gatekeeper can warn the user about a potentially dangerous application that they are executing. This prompt will vary depending on what Gatekeeper find, with Figure 7 being the most common along with a prompt that warns the user if an application bundle hasn't been signed. When this is the case, the user has to explicitly go to System Preferences to continue the execution of the application.

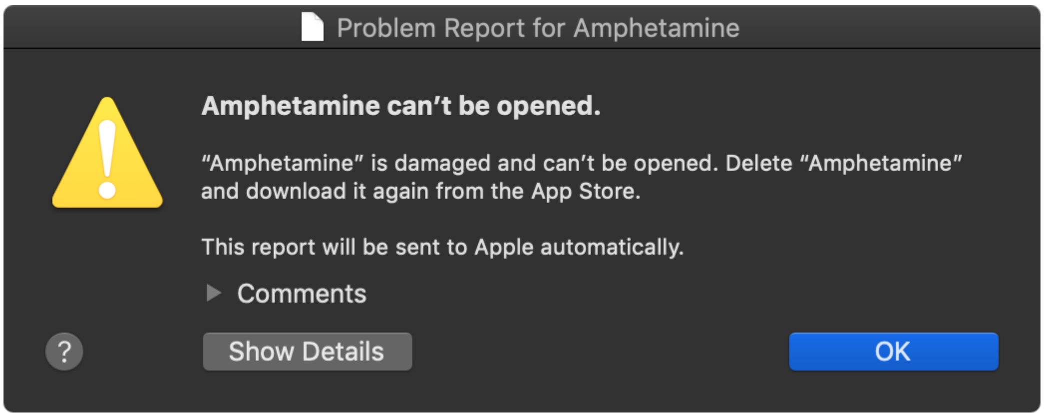
Another significant benefit to codesigning and then verification with Gatekeeper is the continuous integrity of the application bundle. When an application is signed, it means that typically even the smallest change to the bundle will force it to become corrupt. This prevents malware from gaining persistence inside an application this is installed. To show this, an application, Amphetamine, downloaded from the App Store that is currently signed is used. By editing the `Info.plist` file, that contains all the configuration information, it can potentially cause the app to become corrupt.


Info.plist — Edited

```
<?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>BuildMachineOSBuild</key>
	<string>18E226</string>
	<key>CFBundleDevelopmentRegion</key>
	<string>en</string>
	<key>CFBundleExecutable</key>
	<string>Amphetamine</string>
	<key>CFBundleIconFile</key>
	<string>AppIcon</string>
	<key>CFBundleIconName</key>
	<string>AppIcon</string>
	<key>CFBundleIdentifier</key>
	<string>com.if.Amphetamine</string>
	<key>CFBundleInfoDictionaryVersion</key>
	<string>6.0</string>
	<key>CFBundleName</key>
	<string>brokenAmphetamine</string>
	<key>CFBundlePackageType</key>
```

Figure 8: Editing `Info.plist` file

The above highlighted line is the value of `CFBundleName` and is used as a backup to show the name of the application. The value previously read the name of the app, `Amphetamine`. By editing the name of the application, which is potentially the least damaging value to be edited, causes the integrity of the application to fail giving the following alert:


Problem Report for Amphetamine

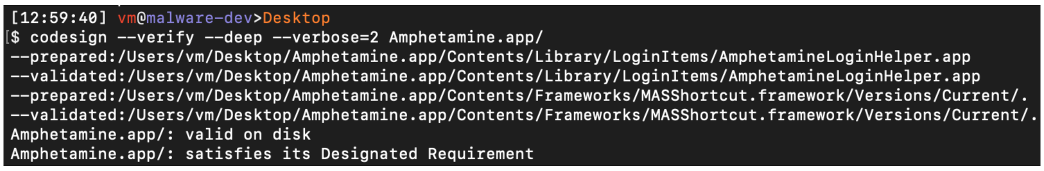
Amphetamine can’t be opened.

“Amphetamine” is damaged and can’t be opened. Delete “Amphetamine” and download it again from the App Store.

This report will be sent to Apple automatically.

Figure 9: Integrity Alert

An issue that Gatekeeper doesn’t cover however is when applications are already installed. If an applications certificate is revoked, then the integrity of any updates that are received can’t be verified. Gatekeeper doesn’t check, unless as mentioned when an application is first executed or the quarantine flag is set. However, using Terminal, Gatekeeper can be forced to check the certificates of application bundles:



```
[12:59:40] vm@malware-dev>Desktop
[$ codesign --verify --deep --verbose=2 Amphetamine.app/
--prepared:/Users/vm/Desktop/Amphetamine.app/Contents/Library/LoginItems/AmphetamineLoginHelper.app
--validated:/Users/vm/Desktop/Amphetamine.app/Contents/Library/LoginItems/AmphetamineLoginHelper.app
--prepared:/Users/vm/Desktop/Amphetamine.app/Contents/Frameworks/MASShortcut.framework/Versions/Current/.
--validated:/Users/vm/Desktop/Amphetamine.app/Contents/Frameworks/MASShortcut.framework/Versions/Current/.
Amphetamine.app/: valid on disk
Amphetamine.app/: satisfies its Designated Requirement
```

Figure 10: `codesign` - Mimics Gatekeeper Checks (Oakley 2019)

The final two lines of the above figure are the equivalent of passing the Gatekeeper checks. If the application weren't signed, then these bottom lines would state that it failed.

As Gatekeeper is the initial step of the quarantine system that is implemented in macOS, once Gatekeeper checks the signing status, XProtect is called to ensure that the file isn't recognised as malicious.

2.4 El Capitan (10.11)

2.4.1 System Integrity Protection

Potentially one of the largest updates to the macOS security landscape is SIP. SIP is enabled by default on all versions of macOS since its release and provides protection to the core macOS files. SIP prevents, either by user error or malicious attacks, any modifying, deleting or adding files or directories to any of the following directories (Apple 2018c):

- `/bin`
- `/System`
- `/sbin`
- `/usr`

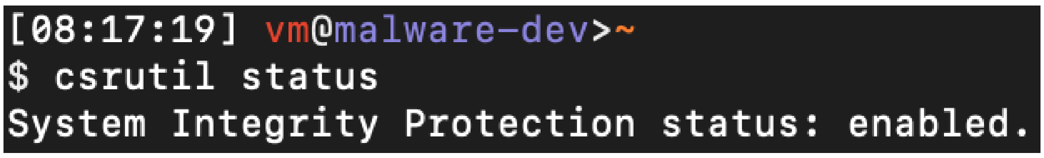
The above directories are key for the operation of macOS and can't be altered in any way other than by Apple, even making use of the `root` user in Terminal (macOS employs rootless operation as part of SIP). It is stated that the following directories have limited access for Developers, however, they are still under SIP:

- `/Applications`

- `/Library`
- `/~/Library`
- `/usr/local`

A full list of all directories and sub-directories protected by SIP can be found at `/System/Library/Sandbox/rootless.conf`. This file also lists exceptions of SIP, indicated with a $*$. (Trouton 2015)

By implementing SIP, it assists in preventing malware from gaining persistence by writing to the core macOS directories and also limits how much power the `root` user has. To check if SIP is enabled, the following command can be used:



```
[08:17:19] vm@malware-dev>~
$ csrutil status
System Integrity Protection status: enabled.
```

Figure 11: `csrutil` SIP Check

As can be seen above, the status reads enabled as it is by default. SIP can be disabled however it is heavily recommended not to disable it as it leaves the core system vulnerable.

SIP protects not only the core system files but also all pre-installed and built-in applications from Apple as well as some specific third-party applications. Any apps downloaded from the App Store are protected by SIP, unless they are sandboxed at which the sandbox gains priority as the protections for the app are stricter. (Trouton 2015)

2.5 High Sierra (10.13)

2.5.1 APFS

Since 1998 (Apple 1998), Apple have implemented `HFS` for its filesystem in macOS. With the introduction of macOS High Sierra, Apple started forcing users to automatically upgrade to Apple File System (APFS) (in macOS Sierra, APFS was an option).

APFS offers much better security than that of HFS. When FileVault[3] is enabled, with HFS it implements FDE, whereas with the addition of APFS,

[3]FileVault 2 must be enabled with APFS for the additional layer of encryption. Although APFS natively supports it, the feature is implemented with the assistance of FileVault 2

FDE is still implemented however it encrypts the drive on a system level (McElhearn 2017) and makes use of multi-key encryption. This means that if an attacker were to retrieve a device key, then they would still be unable to decrypt any of the user's files (Apple 2018a).

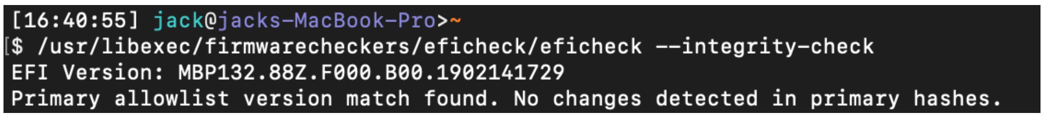
2.5.2 Firmware Validation

In recent years, various macOS firmware exploits have been discovered, the largest of which include Thunderstrike and Thunderstrike 2.

Both of the Thunderstrike exploits are so named as they make use of the Thunderbolt port on Mac devices to alter the firmware of the host device. Thunderstrike 2, however, is more advanced in that it can alter the cryptographic keys that are stored in the Extensible Firmware Interface (EFI), meaning that only updates signed with the attacker's private keys can be installed to replace the EFI. The same exploit has also been adapted to act as a worm, so that when a clean, uninfected Thunderbolt device is connected, it becomes a host for the exploit to infect a new Mac device (Ducklin 2015).

To combat this attack, Apple silently added a utility hidden within its core files to check the EFI weekly. The utility creates a cryptographic hash of the currently installed firmware and compares that to a list of known-good firmware.

The utility is located at `/usr/libexec/firmwarecheckers/eficheck/eficheck` and can be used at any point to validate the integrity of the EFI. The utility and result are shown in the below figure[4]:



```
[16:40:55] jack@jacks-MacBook-Pro>~
[$ /usr/libexec/firmwarecheckers/eficheck/eficheck --integrity-check
EFI Version: MBP132.88Z.F000.B00.1902141729
Primary allowlist version match found. No changes detected in primary hashes.
```

Figure 12: `eficheck` Validating EFI

In Figure 12 above, the `--integrity-check` flag is used to validate the EFI and the result of which shows that the currently installed EFI is valid and signed by Apple. Should the EFI have been altered then the check will fail and the user will be prompted to send a report to Apple, including a binary file of the currently installed EFI[5] (Oakley 2018a).

[4]This had to be tested outside a Virtual Machine due to the EFI handling inside

[5]Mac devices with a T2 chip do not support using `eficheck` as the firmware is validated and updated by different means.

2.6 Mojave (10.14)

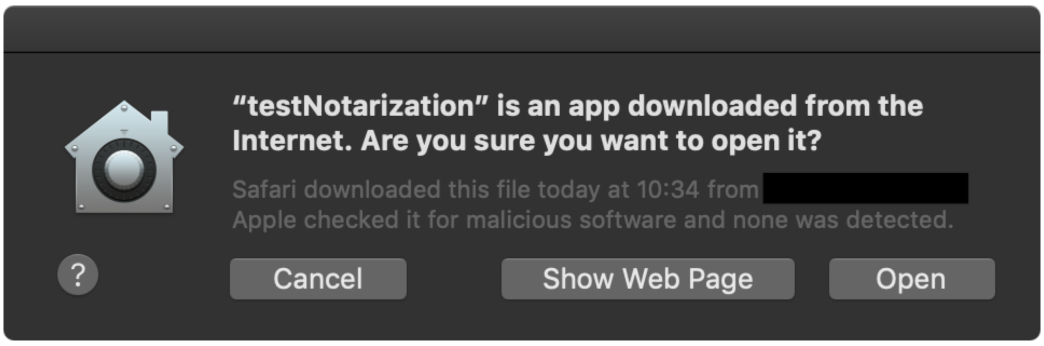
2.6.1 Notarization

At the time of writing, the latest major version of macOS is macOS 10.14 and when it was released last summer, it brought with it numerous privacy upgrades and a significant upgrade with application security.

Notarization forces developers who are distributing apps through the App Store to sign their applications with their Developer ID (this has been implemented before) and also upload it to Apple's Notary Service to be verified. By uploading the app, it can be checked for malicious code by Apple itself before it is uploaded to the App Store for distribution, therefore increasing users security when downloading apps.

For Notarization to pass, the app must also be hardened. Apple state that the app should use Sandboxing (however this is not a requirement), as discussed in Section 2.2.2, and Hardened Runtime, which gives the developer the chance to dictate what permissions are required for the app and allow/disallow memory or `JIT`-compiled code execution, along with other options (Apple 2019a).

With macOS 10.14, Gatekeeper has also been made aware regarding notarized apps as any application that is being developed for distribution can be notarized, providing users peace-of-mind when downloading apps outside of the App Store. This provides a different prompt when a notarized app is executed for the first time. To show this, a basic application was created and submitted to be notarized using a personal Developer ID. When executed the following prompt was given:

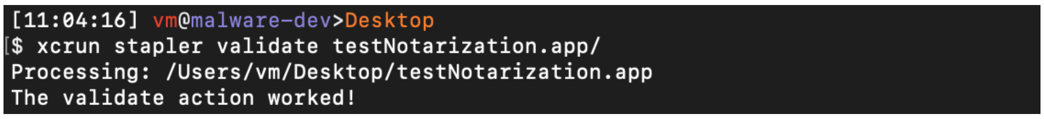

"testNotarization" is an app downloaded from the Internet. Are you sure you want to open it?

Safari downloaded this file today at 10:34 from [redacted]
Apple checked it for malicious software and none was detected.

Figure 13: Gatekeeper Prompt With Notarized App

The figure above shows that when an app is notarized, it states that "Apple checked it for malicious software and none was detected." which, as previously mentioned, can provide peace-of-mind for end users.

An app can also be checked for notarization before being executed using the following command:



```
[11:04:16] vm@malware-dev>Desktop
[$ xcrun stapler validate testNotarization.app/
Processing: /Users/vm/Desktop/testNotarization.app
The validate action worked!
```

Figure 14: Validating An App Is Notarized Using `stapler`

Notarizing an app has an additional benefit similar to code signing, it guarantees the integrity of the app. The difference between code singing and notarization however in regards to integrity is that if anything within the application is altered, then the app is no longer respected as notarized. With code signing as well, once one version of the app is signed, anything else can be signed whereas notarization forces every iteration and updated version of the app to be notarized again before it can be distributed (Oakley 2018c).

It has been recently announced that as of macOS 10.14.5, all Developer ID's that are created from the release date must have applications notarized before they can be distributed. This is a step towards the goal of macOS 10.15, where all applications are notarized, including by developers who previously held a Developer ID (Apple 2019b).

3 Hardware Security

As Apple have been making advancements in their OS and software security, they have also been making improvements in the hardware security of their Mac line-up. This includes the implementation of the in-house designed T2 Coprocessor (T2), the Secure Enclave (SE) and Secure Boot Chain (SBC).

3.1 T2

The T2 chip has been included in the Mac line-up since 2017, starting with the iMac Pro and now featured in the current MacBook Pro, MacBook Air and Mac Mini.

The T2 chip is custom made specifically for the Mac line-up and handles multiple tasks that require incredibly secure operation. The chip contains the SE for processing TouchID data alongside holding cryptographic keys for FileVault and Secure Boot.

The SE of the T2 is integral for protecting the user's information, so much so that if the device were to be compromised even down to the kernel level, then the SE would still maintain its integrity. This is due to the SE only being able to perform specific tasks that have been predefined and the OS not being able to communicate directly to the SE, as the T2 performs any processing and passes the result to the OS (Apple 2018b).

3.1.1 Storage Encryption

As previously mentioned, a Mac device with a T2 contains a dedicated AES-XTS encryption engine. This engine is built directly into the T2, and any encryption or decryption passes directly through the T2 to the main storage, with only the chip handling any processing and simply returning the results of the process. This allows for extremely efficient and secure processing as the OS doesn't handle any of the processing of the data. The cryptographic keys for the data are not only stored directly in the SE, but the keys rely on the UID of the flash storage chips, meaning that if any of the chips included in the process are removed and replaced, data can't be retrieved. To add to this, the UID is burned directly onto the silicon of the T2, so an attacker would need to remove the chip itself and read the UID directly off the silicon, which is incredibly difficult, on the verge of impossible (Apple 2018b).

Another feature that is available when both a T2 is present and the storage is formatted as APFS, is instant encryption when FileVault is enabled. As standard, when a user is setting up a Mac device for the first time, they are prompted to enable FileVault. If this isn't enabled, the data stored on the device is still encrypted however the UID is stored as is in the SE, therefore the user's data is still protected. It is still highly recommended to enable FileVault, so much so that the default option is for FileVault to be enabled. If a user enables FileVault at a later time, the data is instantly encrypted to a higher level of security as the UID is combined with the user's password and other cryptographic keys to protect the data, and then encrypted again with the new keys (Apple 2018b).

3.1.2 Secure Boot Chain

With a T2 chip present, the Mac can perform hardware-based Secure Boot. The SBC contains cryptographically signed instructions that at each stage is checked for integrity. Only once all steps have been checked will the device continue to boot. By checking the integrity of the boot code, it creates a hardware root of trust.

As the Mac is going through the boot process, a verified copy of the bootable firmware if copied from Read-Only Memory (ROM) to the T2. The code that is stored in ROM is injected during fabrication and so it can be natively trusted as there is no way to alter this code. When it is loaded to the T2, it is held there until the device is prepared and has to finalise booting at which point it will retrieve the protected and secure code from the T2 and then continue.

By implementing the SBC, it ensures that there has been no altering or tampering of the core boot files. This allows for a user to be certain that there is no malware in the form of rootkits being initialised whenever the device boots.

If there are any issues while booting, for example, if any of the primary stages of booting fail the cryptographic checks, the device will boot to macOS Recovery mode, allowing a user to reinstall macOS without a loss of data, or securely erase the device (Apple 2018b).

4 Discussion

In general, the steps that Apple have taken to improve the security of users personal data are significant. In comparison to before many of the aforementioned security features were implemented, it was trivial for malware to be installed on a device once access was gained, whereas nowadays it is exponentially more difficult.

With the addition of SIP, it makes it incredibly difficult for malware to gain persistence within the pre-installed apps or the core system files. In addition to that, the development of the quarantine system, implementing Gatekeeper, XProtect and MRT, has significantly increased the safety of users information by automatically detecting malware and removing it where possible. Before this was introduced, there were no systems to detect malware that was being installed on a device.

Introducing notarization allows for users to have peace-of-mind that an application has been verified by Apple and is safe to install. Meanwhile, the advancements of FileVault in combination of the T2 chip has generated a level of safety and security for personal information that is incredible in comparison to what existed before it.

5 Conclusion

This paper has two aims:

- To discuss the security features, including hardware, that have been implemented into macOS and how they operate
- To discuss how the security features are providing a level of security for the end user and their data

Based on these aims, Sections 2 through 3 discuss the security features that have been added to macOS throughout multiple updates and details how these features function. These sections also detail how the security features protect the user and their data with a high level of security.

All of the research that has been discussed was found from reliable sources. Howard Oakley is a long time macOS researcher alongside Jonathan Levin and Patrick Wardle. A large amount of the research came from self-published articles from Apple as well. All of the mentioned authors work was vital for this paper to be completed, as much of the information that is provided in this paper isn't documented by Apple and so it relied on the named authors and their work to assist in the research presented.

6 Future Work

As this paper was a majority of research, future work would include putting this research into practice. This would allow the author to develop tools for hardening macOS or performing malware analysis, now that a baseline understanding of the underlying OS has been created.

References

Apple (1998). *Technical Note TN1121: Mac OS 8.1.* URL: https://developer.apple.com/library/archive/technotes/tn/tn1121.html#//apple_ref/doc/uid/DTS10002961 (visited on 05/05/2019).

— (2016). *App Sandbox Design Guide.* URL: https://developer.apple.com/library/archive/documentation/Security/Conceptual/AppSandboxDesignGuide/AboutAppSandbox/AboutAppSandbox.html#//apple_ref/doc/uid/TP40011183-CH1-SW1 (visited on 05/02/2019).

— (2018a). *Apple File System Guide.* URL: https://developer.apple.com/library/archive/documentation/FileManagement/Conceptual/APFS_Guide/Features/Features.html (visited on 05/04/2019).

— (2018b). *Apple T2 Security Chip Overview.* URL: https://www.apple.com/mac/docs/Apple_T2_Security_Chip_Overview.pdf (visited on 05/04/2019).

— (2018c). *macOS Security: Overview for IT.* URL: https://www.apple.com/business/resources/docs/macOS_Security_Overview.pdf (visited on 05/05/2019).

— (2019a). *Hardened Runtime Entitlements.* URL: https://developer.apple.com/documentation/security/hardened_runtime_entitlements?language=objc (visited on 05/05/2019).

— (2019b). *macOS Security: Overview for IT.* URL: https://developer.apple.com/documentation/security/notarizing_your_app_before_distribution (visited on 05/05/2019).

Ducklin, Paul (2015). *Interested in Mac viruses? Here's Thunderstrike 2, a.k.a. the "firmworm".* URL: https://nakedsecurity.sophos.com/2015/08/05/interested-in-mac-viruses-heres-thunderstrike-2/ (visited on 05/04/2019).

GeoSn0w (2018). *A long evening with iOS and macOS Sandbox.* URL: https://geosn0w.github.io/A-Long-Evening-With-macOS%27s-Sandbox/ (visited on 05/03/2019).

Levin, Jonathan (2019). *jtool2.* URL: http://NewOSXBook.com/tools/jtool2.tgz (visited on 05/03/2019).

McElhearn, Kirk (2017). *The Ins and Outs of Apple's New File System, APFS.* URL: https://www.intego.com/mac-security-blog/the-ins-and-outs-of-apples-new-file-system-apfs/ (visited on 05/04/2019).

NetMarketShare (2019). *Operating System Market Share.* URL: https://netmarketshare.com/operating-system-market-share.aspx? (visited on 05/01/2019).

Oakley, Howard (2016). *Inside the OS X blacklist: XProtect.* URL: https://eclecticlight.co/2016/01/25/inside-the-os-x-blacklist-xprotect/ (visited on 05/02/2019).

— (2018a). *How High Sierra checks your EFI firmware.* URL: https://eclecticlight.co/2018/06/02/how-high-sierra-checks-your-efi-firmware/ (visited on 05/05/2019).

— (2018b). *Last Week on My Mac: Well-kept secrets, macOS malware protection.* URL: https://eclecticlight.co/2018/01/21/last-week-on-my-mac-well-kept-secrets-macos-malware-protection/ (visited on 05/02/2019).

— (2018c). *Notarization: a big step forward for users and developers.* URL: https://eclecticlight.co/2018/08/02/notarization-a-big-step-forward-for-users-and-developers/ (visited on 05/05/2019).

— (2019). *Code Signing For The Concerned: 4 Results and Testing.* URL: https://eclecticlight.co/2019/01/18/code-signing-for-the-concerned-4-results-and-testing/ (visited on 05/03/2019).

AV-TEST (2019). *Malware Statistics & Trends Report.* URL: https://www.av-test.org/en/statistics/malware/ (visited on 05/01/2019).

Trouton, Rich (2012). *FileVault 2 Decoded.* URL: https://macadmins.psu.edu/files/2012/11/psumacconf2012-filevault.pdf (visited on 05/04/2019).

— (2015). *System Integrity Protection – Adding another layer to Apple's security model.* URL: https://derflounder.wordpress.com/2015/10/01/system-integrity-protection-adding-another-layer-to-apples-security-model/ (visited on 05/04/2019).

Wardle, Patrick (2019). *Malware.* URL: https://www.objective-see.com/malware.html (visited on 05/02/2019).