Jamf Threat Labs discovers new malware embedded in pirated applications

Research led by: Ferdous Saljooki and Jaron Bradley

Introduction

Jamf Threat Labs has detected a series of pirated macOS applications that have been modified to communicate to attacker infrastructure. These applications are being hosted on Chinese pirating websites in order to gain victims. Once detonated, the malware will download and execute multiple payloads in the background in order to secretly compromise the victim's machine.

Discovery

While triaging through various threat alerts we've created at Jamf Threat Labs, we stumbled upon an executable that had the name `.fseventsd`. This executable is notable as it's both hidden (starts with a period) and uses the name of a process built into the operating system. On top of that, it's also not signed by Apple and at the time of our research had no indication of being malicious on VirusTotal, as shown below.

Such characteristics often warrant further investigation. Using VirusTotal we were able to determine that this curious-looking `.fseventsd` binary was originally uploaded as part of a greater DMG file. After looking for similar files on VirusTotal, we discovered three pirated applications that had all been backdoored with the same malware. Upon searching for these applications across the internet, we discovered that many were being hosted on `macyy[.]cn`, a Chinese website that provides links to many pirated applications. We also discovered two additional DMGs trojanized in the same manner that had not yet made their way to VirusTotal. Below is a list of compromised DMGs.

Analysis of the Malware

Each pirated application resulted in the execution of malicious activity. At a high level this activity is as follows:

Malicious dylib: A malicious library loaded by the application that acts as a dropper executing each time the application is opened

Backdoor: A binary downloaded by the malicious dylib that uses the Khepri open-source C2 and post-exploitation tool

Persistent downloader: A binary downloaded by the malicious dylib that sets up persistence and downloads additional payloads

Below we will break down each of these items in detail using the malicious FinalShell.dmg referenced above. Each application bundle has had its Mach-O executable modified with an additional load command.

Below are two images. The first shows the load commands of the legitimate FinalShell.app executable, while the second shows the load commands of the malicious FinalShell.app.

Here we see that the malicious FinalShell application has an additional dylib library titled libpng.dylib loaded at runtime. This dylib is loaded each time the application is opened thus starting the malware in the background.

Analysis of the malicious dylib

This technique of hooking malware in via malicious dylib is considered fairly advanced as far as macOS malware goes. However, it does result in breaking the application signature. As a result, the apps are being distributed online as unsigned applications — a detail that many users who are downloading pirated applications likely don't care about.

Looking within the dylib, we see that there are a series of fairly obvious strings.

Upon execution of the FinalShell application, the dylib reaches out to two different URLs to download two payloads located at `hxxp://download[.]finalshell[.]cc/bd.log` and `hxxp://download[.]finalshell[.]cc/fl01.log`. The malware uses `sscanf` and the format string `%*[^//]//%[^/]%s` to extract the domain and path from the URL to make a HTTP GET request.

Both of the downloaded executables are encoded with a custom XOR routine and get decoded in-memory before being written to disk. Presumably this is to prevent any researchers from analyzing the binaries if they happen to come across the download URL.

The below function takes an encoded byte, applies a combination of XOR and subtraction operations with the XOR key `0x7a`, and produces a decrypted byte. The use of both XOR and subtraction makes the encryption slightly more complex than a simple XOR cipher. The masking with `0xff` ensures that all operations stay within the valid range for a byte, accounting for any underflow during subtraction.

A script to decode the encoded executables can be found at the end of this blog post.

After decoding the executables, the dylib will write the first download (bd.log) to the path `/tmp/.test` and the second download (fl01.log) to the path `/Users/Shared/.fseventsd`. Executable permissions are then applied and the payloads are subsequently launched with a call to `execve`. It's important to note that the dylib will get loaded each time the FinalShell application is launched. This means the attacker can swap the encoded executable on the server for a new payload at any given time and it will be downloaded and executed the next time the user opens the application.

Before executing the newly downloaded `/tmp/.test` binary, the attacker sets the arguments to represent that of ssh, which we can see by running the launchctl procinfo command after placing a breakpoint on it at the start of the newly executed malware.

We found this interesting for multiple reasons. First off, if a power user did happen to spot these fake arguments, ssh is likely to cause a panic and further investigation. Secondly, the path to ssh that was chosen is not the true path to ssh on macOS. Finally, as we'll get into in the next section, the `/tmp/.test` executable will overwrite these arguments as soon as it's executed.

Analysis of the backdoor

Despite being called `/tmp/.test`, this executable is a fully fledged backdoor built upon the open-source Khepri command and control project. This executable remains hidden in the temporary directory without being relocated, which means that it will be deleted when the system shuts down. However, it will be created and run again the next time the application is opened and the malicious dylib is loaded.

As mentioned above, one particularly interesting technique that the malware uses is replacing its command-line arguments in order to further blend in with the operating system. It does this by copying the arguments "distnoted agent" into the processes argv[0] value using the strcopy function. After this occurs, we see that the arguments now reflect that of "distnoted agent"

This technique is primarily effective against the "ps" output where it will now look similar to the other "distnoted agent" processes that likely exist on the user's system.

If using a program that looks at process names based on sources other than the process's argv[0] value, such as TrueTree, we see that the real name of the process is displayed, as shown below.

Upon examining the Khepri GitHub project, we did not see an option to perform this argv replacement. It's possible that it was manually added by the malware author.

Much of the Khepri functionality can be easily seen on the client's beacon.h header.

Analysis of the persistent downloader

The executable located at the path `/Users/Shared/.fseventsd` functions as a persistent downloader that is capable of executing arbitrary payloads from the attacker's server. The malware will first create a LaunchAgent at the path `~/Library/LaunchAgents/com.apple.fsevents.plist`. The `com.apple` prefix is an evasion technique we've seen across various macOS malware to disguise itself as a legitimate Apple plist.

The LaunchAgent will ensure the `.fseventsd` binary persists by executing it at the hidden path `/Users/Shared/.fseventsd`. The malware will then send a HTTP GET request to the attacker server at the URL `hXXp://bd.vscode.digital/fs.log`.

We've seen similar URLs with the path `/fs.log` across the trojanized applications. However, at the time of our analysis these servers do not appear to be online and are not responding to the requests.

The malware will execute a series of system calls to copy the bytes from the HTTP response to a new file located at `/tmp/.fseventsds`. This unknown executable is then launched.

The Khepri backdoor discussed in the previous section also has functionality to execute payloads. However, in order for the Khepri backdoor to be operational, the application must be opened by the user, as this persistent downloader will run anytime the system starts.

Similarities to ZuRu Malware

We've noted a number of similarities of this malware to the ZuRu malware originally discovered in 2021 and blogged about by Objective-See and Trend Micro.

The ZuRu malware was originally found in pirated applications iTerm, SecureCRT, Navicat Premium and Microsoft Remote Desktop Client. Upon opening the infected application, the user was presented with an operational app, but logic held within an added dylib would execute a Python script in the background to grab sensitive files and upload them to an attacker server. It's possible that this malware is a successor to the ZuRu malware given its targeted applications, modified load commands and attacker infrastructure.

Similar to findings of the ZuRu malware in 2021, this malware also appears to primarily target victims in China based on the uploads we've seen to VirusTotal, the hosting of apps on pirated Chinese websites and the attacker infrastructure that communicates with Chinese IP addresses, as shown below.

Conclusion

This is not the first time Jamf Threat Labs has seen malware within pirated applications. One of the major difficulties about dealing with users who are installing pirated applications is the fact that they are expecting to see security alerts due to the fact that the software isn't legitimate. This leaves them willing to skip past any security warning prompts built into the operating system such as Gatekeeper, which informs the user that these applications are not safe to open. Jamf Threat Labs remains vigilant in detecting these changes to keep customers safe by blocking this malware with our threat prevention feature in Jamf Protect.

Below is a list of indicators associated with each pirated application.

The following script can be used to decode the XOR encoded binaries downloaded by the malicious dylib.