Detecting and Mitigating the GateKeeper User Override on macOS in an Enterprise Environment

Author: Antonio Piazza, antman_007@hotmail.com
Advisor: Domenica Crognale

Accepted: September 8, 2022

Abstract

For red teamers, social engineering a macOS user into executing an application is a common way to gain code execution on a remote macOS client machine. Apple's development of their macOS built-in security mechanism, GateKeeper, has made this a more difficult task, but not impossible. It is effortless for a macOS user to bypass GateKeeper by simply right-clicking to execute a potentially malicious application. An adversary can convince the user to override GateKeeper in this manner and gain remote code execution on the user's system. In fact, many adversaries have done just that. This could lead to further exploitation of a corporate network, so quickly detecting this user activity is essential. While this is a crucial detection, endpoint security products seem to lack the capability. This research explores the detection possibilities for the GateKeeper user override. Developing a GateKeeper detection will allow corporate security teams to protect their environments from users being socially engineered into executing malware. This is an essential step in increasing the defenses of our macOS corporate environments.

1. Introduction

Some red teams conduct operations against environments containing macOS endpoints. For many of these operations, teams conduct phishing campaigns. The teams develop a pretext for the phishing emails to send to the target macOS users, enticing the target to click on a link to download a malicious application.

Some red teams develop these malicious applications in-house, where the application contains a payload that, when executed, would establish a Command & Control (C2) connection back to the attacker-controlled server. This application could be a macOS installer package that the developer had not signed and thus was not notarized by Apple.

The red team could convince the target macOS user to download and execute the application through the phishing email pretext. If the user double-clicks the malicious package to open it, GateKeeper prohibits the execution.

To circumvent this problem, red teams could convince the user to override GateKeeper by right-clicking to open the application rather than by double-clicking it. This GateKeeper user override has a non-malicious purpose and is a convenient way for users to execute unsigned and un-notarized applications. Still, it is easy to see how this could be a significant security problem in a controlled enterprise environment.

GateKeeper is a great security mechanism built into macOS, but as discussed above, it can easily be circumvented. Apple describes GateKeeper as follows:

> macOS includes a security technology called Gatekeeper, which is designed to help ensure that only trusted software runs on a user's Mac. When a user downloads and opens an app, a plug-in, or an installer package from outside the App Store, Gatekeeper verifies that the software is from an identified developer, is notarized by Apple to be free of known malicious content and hasn't been altered. ("GateKeeper and Runtime Protection in macOS," 2021).

Antonio Piazza, antman_007@hotmail.com

Organizations could use a mobile device management (MDM) solution to completely disallow users the ability to download applications outside the Apple App Store. However, many organizations do not do this because they want to allow certain users to run applications that Apple does not provide in the App Store.

Suppose an organization does not wholly control application downloading and permits users to download applications outside the Apple App Store. In that case, the organization must ensure they have visibility of users who right-click to open applications and override GateKeeper, or they might be putting their enterprise at risk of a user opening a malicious application and exposing their network to infection. This research will show how one might create a viable detection at scale for an enterprise environment to protect it from users potentially executing malicious applications.

2. Understanding GateKeeper

"GateKeeper was introduced by Apple in macOS 10.7.5 (OSX Lion), stating that it 'helps protect your Mac from apps that could adversely affect it" (Levin, 2019, p. 96).

GateKeeper is a macOS security feature built upon its predecessor, File Quarantine.

> Apple introduced the notion of File Quarantine well before GateKeeper (in 10.5, OSX Leopard), but the two features interoperate well. Quarantine serves as the first line of defense, and GateKeeper as the second (and de-facto last), in the fight against untrusted code (Levin, 2019, p. 96).

When a user downloads content outside the App Store, macOS adds an extended attribute called the quarantine attribute, to the downloaded content. Users can use the macOS native *xattr* tool to display and manipulate quarantine and other extended attributes.

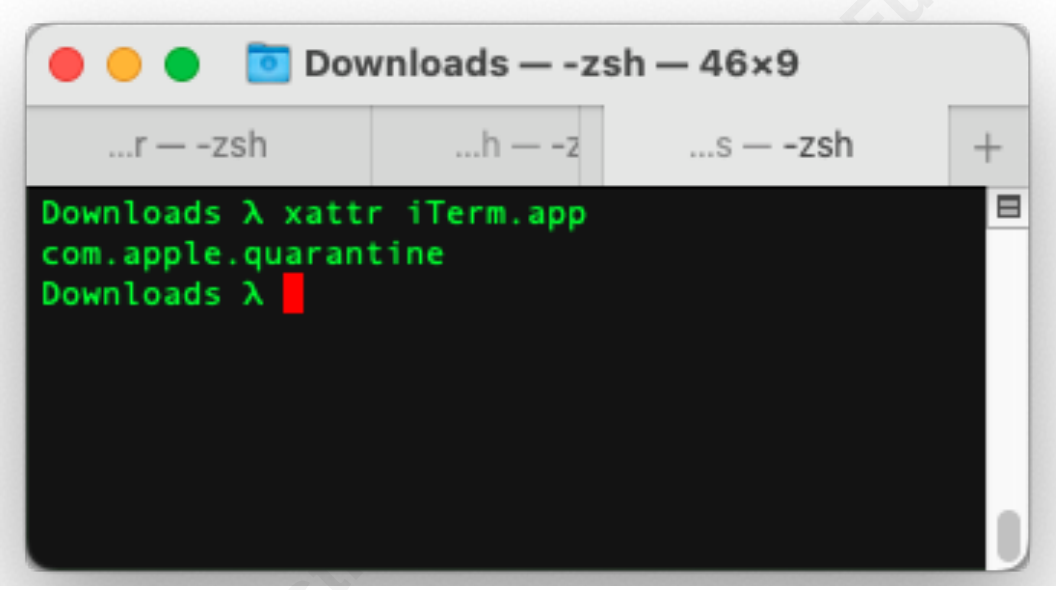

Figure 1 - Xattr Quarantine Attribute

When a user attempts to execute this quarantined content, macOS interferes by alerting the user that they downloaded this content, shows the user from where the content was downloaded and allows the user to decide whether they still wish to open it by clicking the "open" or “cancel” button.

While this was an important step toward preventing users from executing untrusted code, it did not entirely stop them from accidentally confirming malware execution. Gatekeeper provides a much more significant step in the right direction.

Initially, GateKeeper worked by checking the downloaded content's signature and ensuring that the developer signed it with a valid Apple Developer ID, however, it is trivial for anyone willing to pay the USD 99 fee to sign up for an Apple Developer account and acquire a valid Apple Developer ID.

To combat such potentially rogue developers, Apple mandated application notarization in 2019 with the release of macOS 10.15 (Catalina). Notarization requires that developers submit their code for what Apple describes as an automated review process:

Notarization gives users more confidence that the Developer ID-signed software you distribute has been checked by Apple for malicious components. Notarization

> is not App Review. The Apple notary service is an automated system that scans your software for malicious content, checks for code-signing issues, and returns the results to you quickly. If there are no issues, the notary service generates a ticket for you to staple to your software; the notary service also publishes that ticket online where Gatekeeper can find it ("Notarizing macOS software before distribution," 2022).

Along with checking that developer with a valid Developer ID signed the downloaded content, GateKeeper also ensures that Apple notarized the content and thus inspected it for malicious components. This poses the question: how does this automated notarization process work?

When Apple mandated notarization in 2019, a test was conducted. A malicious macOS application was developed. The application posed as a Zoom Updater, but it contained a payload that connected the victim machine to a red team C2 server. The USD 99 was paid, an Apple Developer Account was created, a Developer ID was created, the malware was signed, and the malware was submitted to Apple for notarization. Within seconds, the application was notarized. The notarized, malicious application could be used in red team operations for approximately two weeks. Without warning, explanation, or refund, Apple revoked the Developer ID, and the malicious application was no longer notarized.

This could suggest that the notarization process is not entirely as automated as Apple suggests, or possibly that notarization did not have the signature for the custom malware at the time. Cedric Owens indicates the reason that the test application was successfully notarized:

> I believe this is because it is simply running JXA, with the Apfell Objective C post exploitation code living on the server rather than in the app code and since the notarization check is looking solely at the app code and not server code, the app just looks like an app running JXA (Owens, 2020).

How this notarization process works is somewhat of a mystery because Apple has not documented the automated notary service process itself.

As mentioned in the introduction, an organization can use an MDM solution to completely disallow the execution of applications downloaded outside the App Store. A macOS user can accomplish this as well in the "Security & Privacy" System Preferences by selecting the "App Store" option under "Allow apps downloaded from."

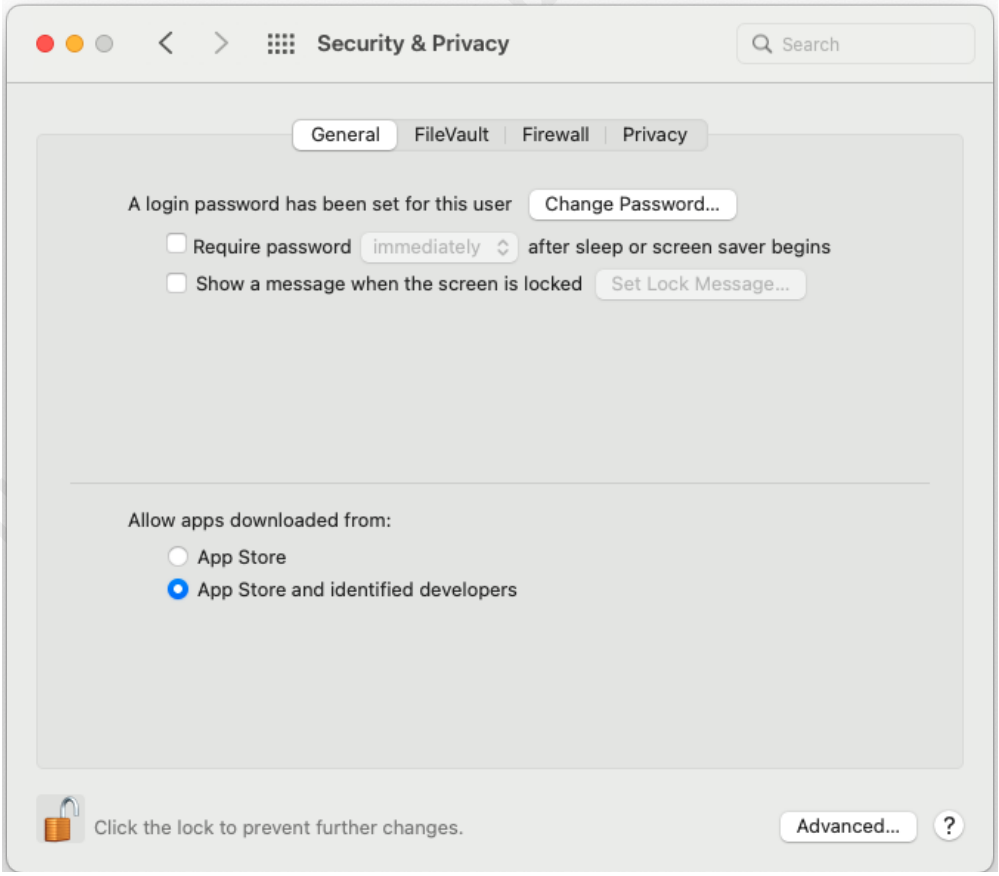

Figure 2 - System Preferences > Security & Privacy

By choosing this option, a user will be presented with a warning message when attempting to execute any application not downloaded from the App Store (quarantined content). The only options macOS presents to the user are to either click the "ok" button, which will end the process, or to click the "Show in Finder" button, which will open the finder window at the location of the application.

Antonio Piazza, antman_007@hotmail.com

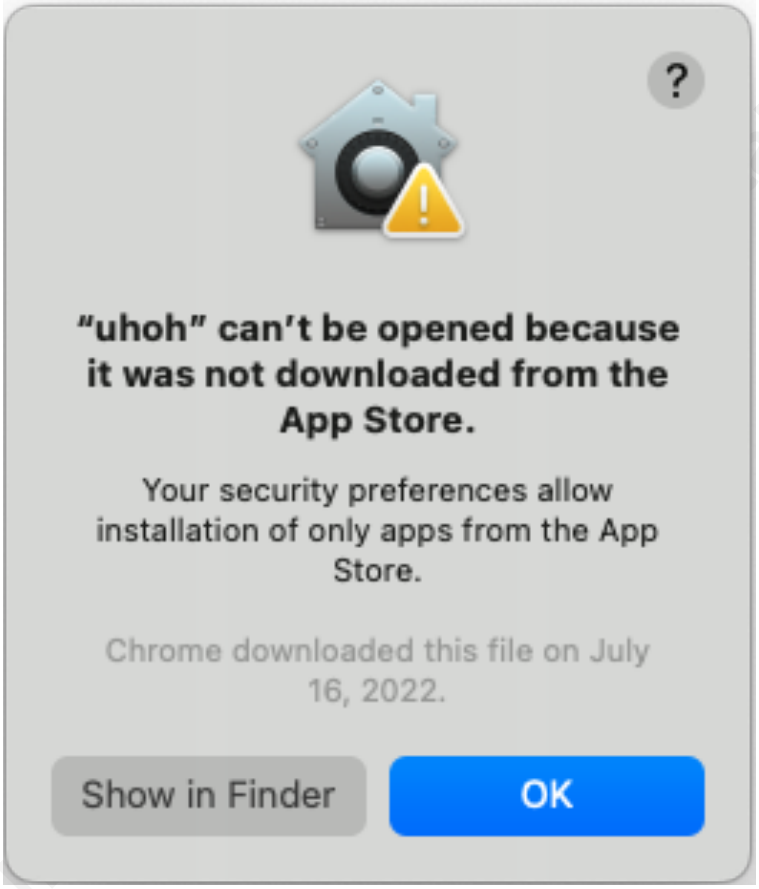

Figure 3 – Outside of App Store Download

In the System Preferences, the user may also allow the execution of apps downloaded outside of the App Store by selecting "App Store and identified developers" under "Allow apps downloaded from." When the user attempts to execute a quarantined application by double-clicking it, GateKeeper steps in and checks the application for a Developer ID and an Apple notarization. In a case where GateKeeper finds a valid signature and notarization, macOS will display the quarantine message and give the user the option to execute the application, as explained above.

Antonio Piazza, antman_007@hotmail.com

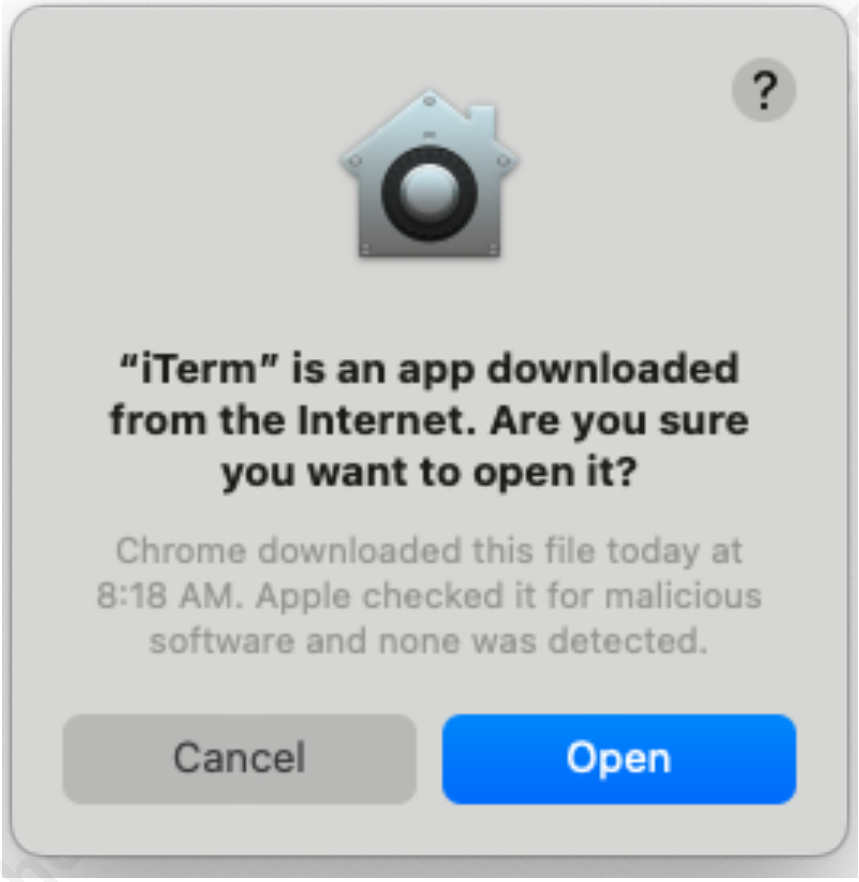

Figure 4 - Quarantine Message

However, suppose GateKeeper finds that the application is not signed or notarized. In that case, macOS will present the user with a message indicating that the developer cannot be verified ("Safely Open Apps on Your Mac," 2021). The message box allows the user to cancel the execution or move the application to the trash. The message box also presents a check box that allows the user to report the application to Apple as malware.

Antonio Piazza, antman_007@hotmail.com

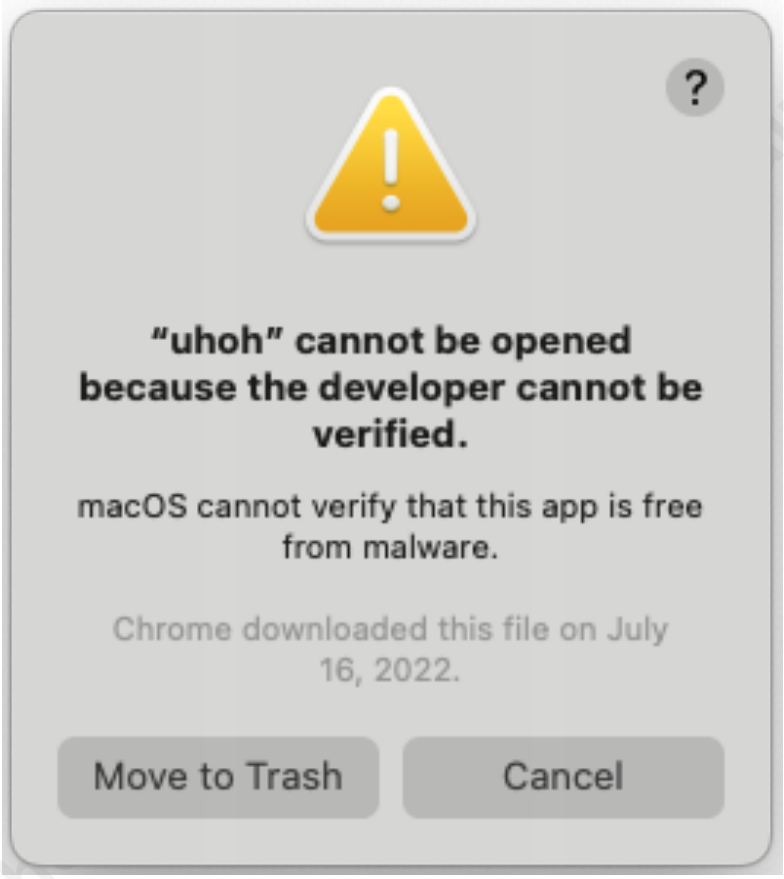

Figure 5 – Not Signed or Notarized

Despite the possible shortcoming of the notary service process, GateKeeper has come a long way in defending systems from users executing untrusted and potentially malicious code. A primary concern is not with GateKeeper's functionality, but that Apple allows users to override it easily.

3. The GateKeeper User Override

As explained above, after a user attempts to open an unverified application, macOS allows the user to cancel the execution or move the application to the trash. If the user chooses to cancel the execution, the user may now override GateKeeper by right-clicking the application and then choosing "open" from the dropdown menu. macOS

then presents the user with a new message box. This message box allows the user to choose from three options. These options are “Open,” “Move to Trash,” or “Cancel.”

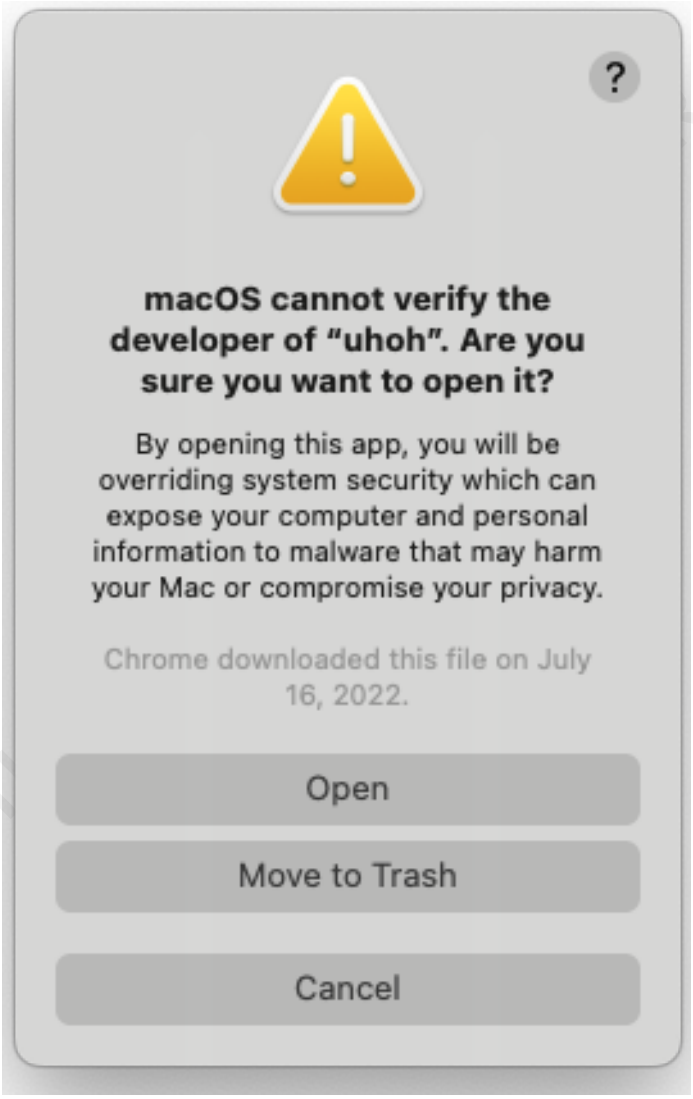

Figure 6 - GateKeeper Override

This is a convenient feature for home macOS users but potentially disastrous in an enterprise environment.

Red teams have easily socially engineered several target users to right-click and open malware in many operations. The red teams simply provide the instructions to do so within the pretext of the phishing email or on their malware-hosting website.

Antonio Piazza, antman_007@hotmail.com

This is also occurring quite frequently in the wild. According to T. Reed (personal communication, July 26, 2022), between April 28 and July 25, 2022, Malwarebytes found 2,156 detections of the 'Chropex,' macOS adware installer. This installer is unsigned, and the dmg contains an animated gif instructing the target to right-click to open.

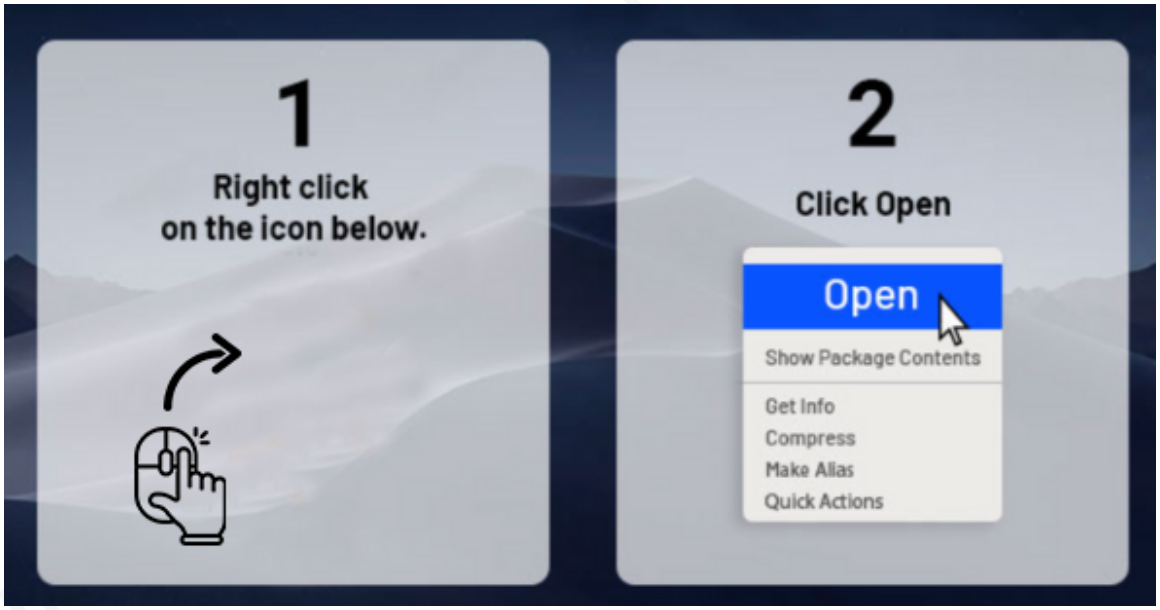

Figure 7 - Chropex Adware Installer Animated Instructions

In June of 2020, adversaries were instructing users to right-click to open the macOS malware 'Shlayer,' as well:

> After the deceptive Flash Player installer is downloaded and opened on a victim's Mac, the disk image will mount and display instructions on how to install it. The instructions tell users to first "right-click" on flashInstaller and select Open…. (Long, 2021).

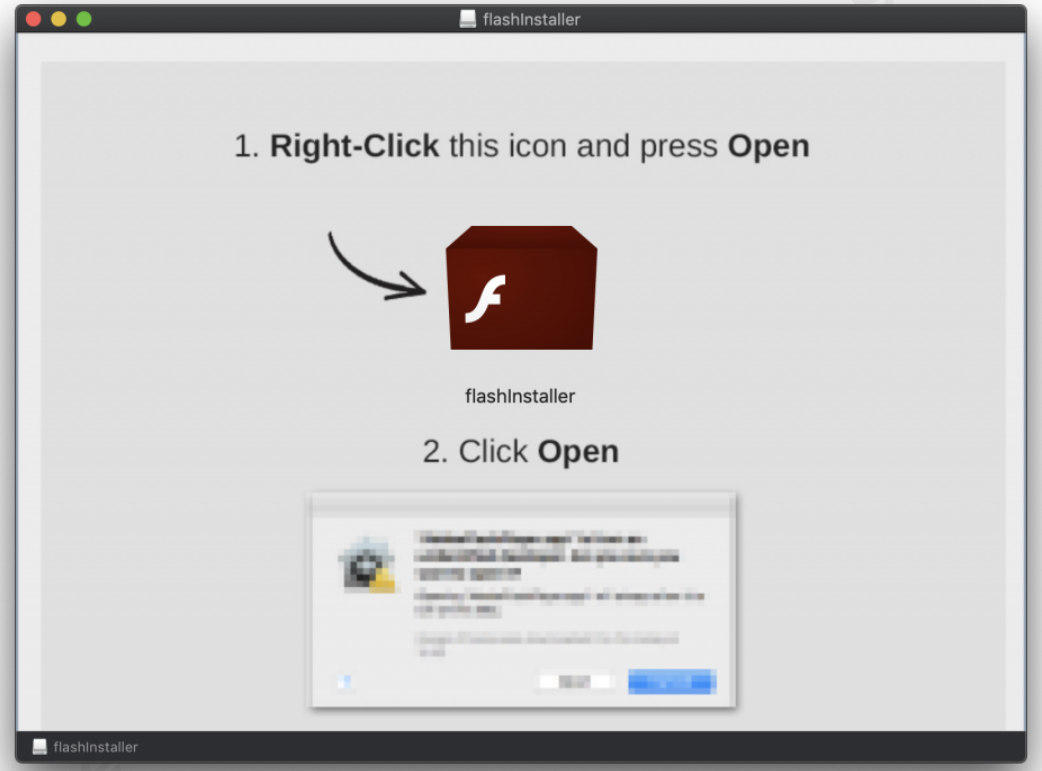

Figure 8 - Shlayer Malware instructing the victim to right-click to open

These examples show how dangerous the GateKeeper override can be to a corporate environment if left undetected. While the override is a convenient way for macOS users to run unsigned code if they wish, being able to detect this behavior on endpoints is critical to keeping environments secure.

4. The Experiments

In Searching for a possible detection for the GateKeeper override, two experiments were conducted. Both experiments were conducted on macOS versions 12.4 and 12.5 (Monterey). In one experiment, the macOS built-in log tool was used.

For the second experiment, ESF Playground, a tool developed by Jaron Bradley, macOS Detections Lead at Jamf, was used.

4.1.1 The Log Tool Experiment (Experiment 1)

To find a viable detection for the GateKeeper override, the advice of Matt Benyo, Threat Detections Developer at Jamf, was followed. Matt suggested the log tool, built-in to macOS, be used and for one to look for the "GateKeeper Denial Breadcrumb" (personal communication, January 31, 2022).

For both the control and experiment groups, the log tool was used and stdout was piped to a text file. This was done to compare the results of the control group to the experiment group.

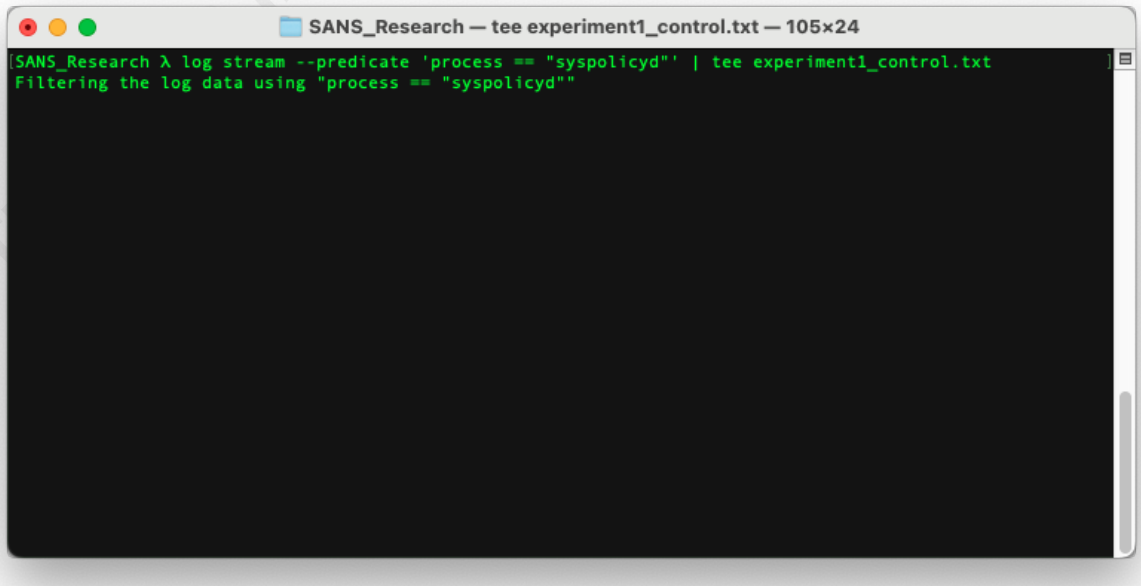

Figure 9 - macOS Log Stream Piping Stdout to a file using the syspolicyd process predicate

The log tool allows users to access system-wide log messages created by os_log, os_trace, and other logging systems. The stream function allows the user to view the log stream. By default, the command assumes system-wide streaming but can be filtered.

The filter used in this experiment is the predicate filter on the syspolicyd process. This allows everything to be filtered out except for syspolicyd messages. Otherwise, the output for the log stream would be far too noisy to find a good detection. Syspolicyd is the macOS system policy daemon.

> Jonathan Levin explains, "The system policy daemon – syspolicyd – is responsible for enforcing the GateKeeper. It is a tiny daemon residing in /usr/libexec and maintaining a SQLite3 database file in /var/db/SystemPolicy" (Levin, 2019, p. 102).

Patrick Wardle, the founder of Objective-See, suggested that to derive the needed information from the macOS log tool, one would need to enable private data in the logs (*Unified logs: How to enable Private Data*). He shared the web link, which contains directions on how to do this (personal communication, May 17, 2022). This link is in the References section.

Suppose one does not enable private information in logging. In that case, the log tool will not show the file path and name of the application the user is attempting to open via GateKeeper override. Identifying this application name is critical in building a detection for the GateKeeper override. One will need to know the name and path of the application a user opened via the GateKeeper override in order to respond to the incident.

Figure 10 - Difference between Private Data Enabled or Not in Logs

For both the control and experiments groups, while the log stream recorded syspolicyd events to a text file, a benign, unsigned application was downloaded from DropBox. The file simply opens the macOS calculator.app. The application is named uhoh.app. The uhoh.app.zip file containing the unsigned application was then unarchived.

Antonio Piazza, antman_007@hotmail.com

4.1.2 The Control Group

For the control group, while the log stream function recorded syspolicyd events to the text file, the uhoh.app application was double-clicked, then the "cancel" button was clicked after GateKeeper worked as expected.

4.1.3 The Experiment Group

For the experiment group, while the log stream function recorded syspolicyd events to the text file, the application, again was double-clicked, and then the "cancel" button was clicked. Then the application was right-clicked, and the "open" button was clicked to override GateKeeper.

4.2.1 The ESF Playground Experiment (Experiment 2)

In the second experiment conducted to find a viable detection for the GateKeeper override, ESF Playground was used. "The ESF Playground can be used to view all raw events coming from the Apple Endpoint Security Framework (ESF) and print them in JSON form" (Bradley, 2022).

ESF is Apple's Endpoint Security Framework. Apple introduced ESF to allow endpoint security tool developers to easily collect security information from macOS via an API. "Endpoint Security is a C API for monitoring system events for potentially malicious activity" (Endpoint Security. Develop system extensions that enhance user security. 2022).

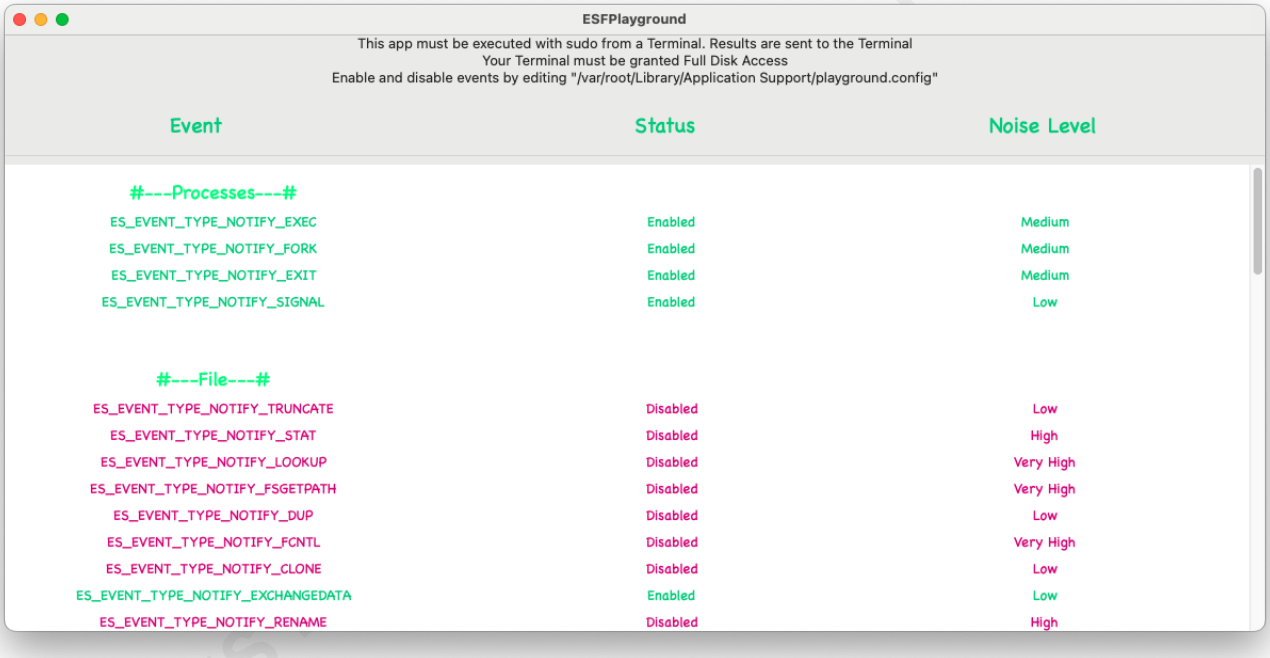

Figure 11 - ESF Playground

ESF Playground provides a configuration file that allows the user to comment out or uncomment the many different ESF event messages supplied by Apple. One can comment out an event message type by adding a "#" symbol at the beginning of the event message line in the configuration file. One can uncomment the event message by removing the "#." Commenting and uncommenting ESF event messages in ESF Playground filters out or activates the output of the desired ESF event message types to be diplayed.

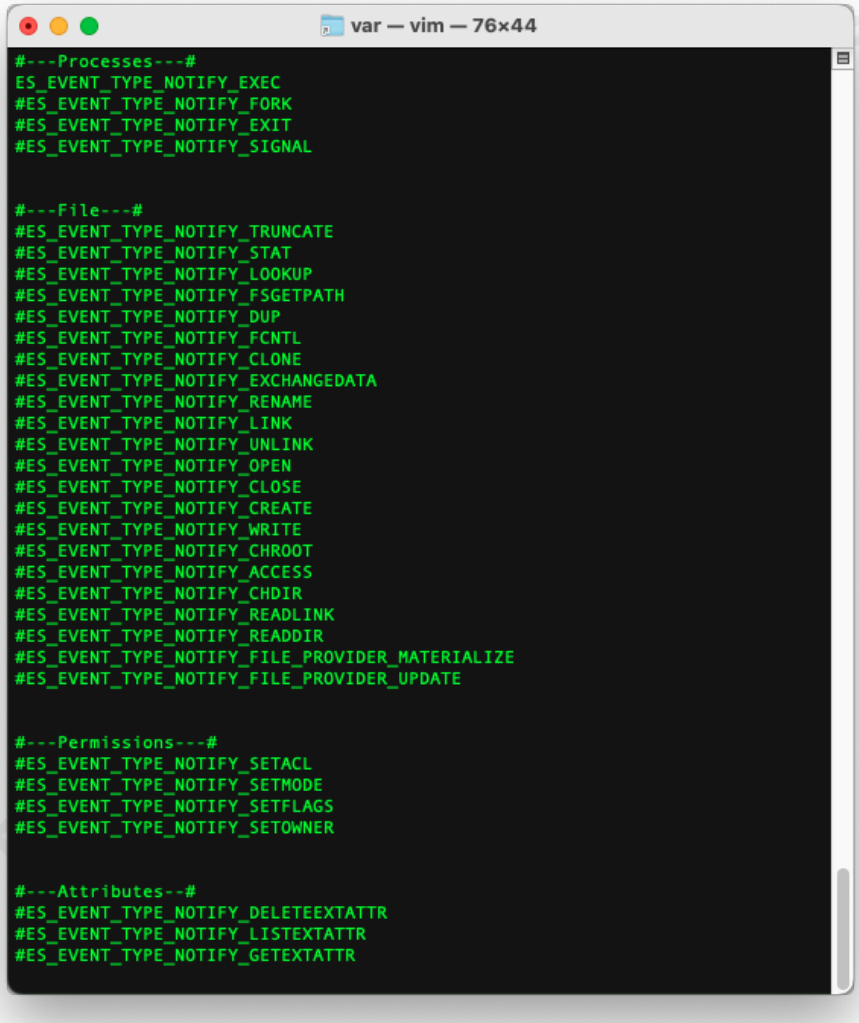

Figure 12 - ESF Playground Configuration File

For both the control and experiment groups, ESF Playground was started, and stdout was piped to a text file. Again, this allows for comparing the results of the control and experiment groups.

Antonio Piazza, antman_007@hotmail.com

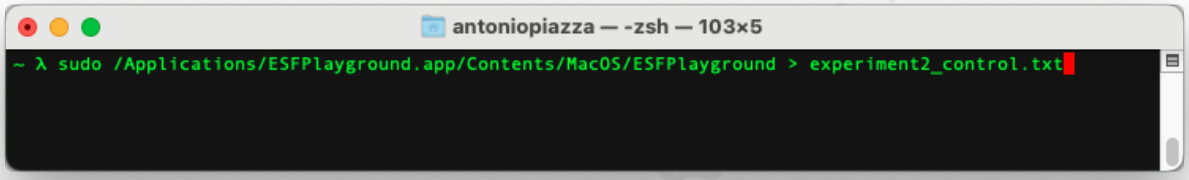

Figure 13 - Run ESF Playground and direct stdout to a txt file

For both groups, one message type at a time had to be commented out, and the experiment and control group activities ran. Running the experiment activities with more than one active ESF event type creates far too noisy an output to find a good detection.

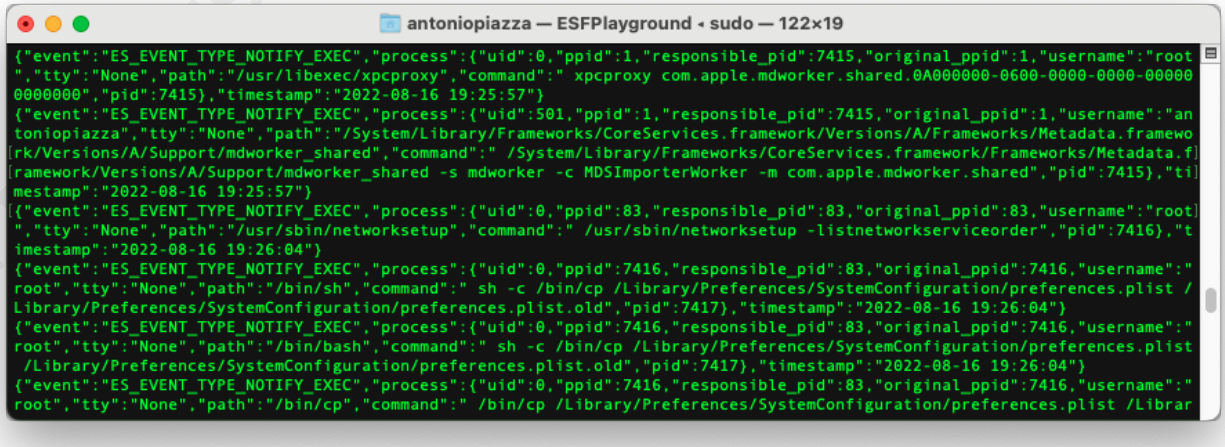

Figure 14 - ESF Playground Displaying ESF Events

For both groups, the uhoh.app.zip archived was downloaded and expanded as in Experiment 1.

4.2.2 The Control Group

Antonio Piazza, antman_007@hotmail.com

For the control group, one ESF event message at a time was uncommented while the text file recorded the output. The uhoh.app was double-clicked, and the "cancel" button was clicked after GateKeeper worked as expected.

4.2.3 The Experiment Group

For the experiment group, the application was again double-clicked, and the "cancel" button was clicked. Then, the application was right-clicked, and the "open" button was clicked to override GateKeeper.

5. The Results

To find a viable detection, differences among the output files of the control group and experiment group activities were searched for.

5.1.1 Experiment 1 Results

The first experiment proved promising for finding a good detection. The main difference between the control and experiment groups is how syspolicyd handled what it calls "the GateKeeper denial breadcrumb." In the control group, syspolicyd added the denial breadcrumb; in the experiment group, it removed it. This means that when a user right-clicks on an unsigned and unnotarized application, then clicks the "open" button, overriding GateKeeper, syspolicyd removes the denial breadcrumb.

Figure 15 - GateKeeper Breadcrumb: Adding Vs. Clearing

This difference in syspolicyd behavior is excellent for creating a log-based detection. However, looking deeper, there is another difference between the control and experiment log outputs. The experiment group log output contains a log entry containing the following test: "Clearing breadcrumb URL due to intent: /Users/antoniopiazza/Downloads/uhoh.app." This entry appears right after the user right-clicks to open, but before the user decides to open or cancel opening the application via the GateKeeper message box buttons.

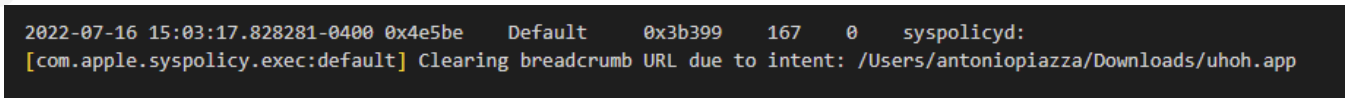

Figure 16 - Clearing Due to Intent

This log entry lets us know when a user has right-clicked to open an application, before it opens. Creating a detection for this seems to be better than the other finding in that defenders or security products have an opportunity to detect an attempted GateKeeper override before it happens. This means that there is a window of opportunity to not only detect the override but also prevent it before it happens.

This detection method is scalable for an enterprise environment. Instead of capturing the macOS logs for only one endpoint as in this experiment, one might capture

this log data for all endpoints in one's environment using an enterprise endpoint security MDM solution, such as Jamf Protect.

> With Jamf Protect, you can use the same predicate-based filter criteria that are often used with the log command to collect relevant log entries from computers and send them to a security information and event management (SIEM) solution or a third party storage solution (e.g., AWS) (Unified logging 2022).

5.1.2 Experiment 2 Results

The second experiment yielded no results indicating a possible detection. Unfortunately, there were no notable differences between the control and experiment group output of ESF events to find a good detection.

6. Conclusion

Discovering the log detection from Experiment 1 is an excellent step for users executing unsigned and possibly malicious code in their environments. Security organizations can easily forward this macOS log activity to their Security Information and Event Management (SIEM) systems. Appendix I demonstrates how to do this using the Elastic Stack SIEM. The security organization can configure their log streams on their macOS endpoints, as in Experiment 1. They can string together as many predicate filters to capture whichever processes they see fit, or they can forward all system logs by adding no predicate filtering.

An organization can also scale this via an MDM solution, such as Jamf.

While this is a working solution, it is not perfect. Many organizations do not have the resources to capture and store endpoint logs from their entire fleet. Log capture and storage can be quite resource-heavy, expensive, and far too burdensome for many organizations.

A more reasonable solution would be for Apple to create an ESF event for the syspolicyd, GateKeeper denial breadcrumb activity. Many macOS Endpoint Detection & Response (EDR) solutions are already monitoring ESF for their detections. If Apple

creates a new breadcrumb event message, EDR solutions, it would be trivial for EDR developers to include it as a detection in their products.

An EDR solution should be able to prevent the GateKeeper override if it can detect a "Clearing breadcrumb URL due to intent" ESF event if Apple creates one. An EDR can kill the process associated with the application name indicated in the event message. Even if the EDR does not have this alert built-in, many EDRs provide a mechanism to engineer custom alerts, so a detection engineer can create one for the ESF event message that the EDR should be capturing.

Without the ESF message, security analysts can use the log-based detection that they forward from the macOS endpoints to their Security Information and Event Management (SIEM) system to kick off an incident investigation. The investigators can look at the application started via the GateKeeper override as indicated in the log-based detection. The investigators can then determine if the application is malicious or not. If the application is malicious, they might use an EDR to kill the process remotely. The incident responders might also decide to quarantine the system, look for any persistence mechanism that the malware might have installed, and look for signs that the malware laterally moved to other systems on the network.

As one can see, it might be far more efficient and safer if an EDR can kill the process before the application has a chance to infect the system.

While the results of this experiment are positive and can help organizations defend themselves against users executing unsigned and possibly malicious code, Apple can help develop a better detection

Antonio Piazza, antman_007@hotmail.com

References

Apple Inc. (2021, April 30). *Safely open apps on your Mac.* Apple Support. Retrieved July 22, 2022, from https://support.apple.com/en-us/HT202491

Apple Inc. (2021, February 18). *Gatekeeper and runtime protection in macOS.* Apple Support. Retrieved July 20, 2022, from https://support.apple.com/guide/security/gatekeeper-and-runtime-protection-sec5599b66df/web

Apple Inc. (2022). *Notarizing macOS software before distribution*. Apple Developer Documentation. Retrieved July 22, 2022, from https://developer.apple.com/documentation/security/notarizing_macos_software_before_distribution

Apple, Inc. (2022). *Endpoint Security. Develop system extensions that enhance user security. Apple Developer Documentation.* Retrieved July 31, 2022, from https://developer.apple.com/documentation/endpointsecurity

Benyo, M. (2022, January 31). Personal Communication.

Bradley, J. (2022, June 8). *The ESF playground*. The Mitten Mac. Retrieved July 31, 2022, from https://themittenmac.com/the-esf-playground/

Jamf. (2022). *Unified logging*. Jamf. Retrieved August 16, 2022, from https://docs.jamf.com/jamf-protect/documentation/Unified_Logging.html

Levin, J. (2019). **Os Internals (Vol. 3, Ser. Security & Insecurity)*. Technologeeks.com.

Long, J. (2021, May 13). *New Mac malware reveals google searches can be unsafe*. The Mac Security Blog. Retrieved July 26, 2022, from https://www.intego.com/mac-security-blog/new-mac-malware-reveals-google-searches-can-be-unsafe/

Owens, C. (2020, May 18). *Launching Apfell programmatically*. Medium. Retrieved July 22, 2022, from https://medium.com/red-teaming-with-a-blue-team-mentality/launching-apfell-programmatically-c90fe54cad89

Reed, T. (2022, July 26). Personal communication.

Antonio Piazza, antman_007@hotmail.com

Unified logs: How to enable Private Data. cmdSecurity. (n.d.). Retrieved July 31, 2022, from https://www.cmdsec.com/unified-logs-enable-private-data/

Wardle, P. (2022, May 16). Personal Communication

Antonio Piazza, antman_007@hotmail.com

Appendix

To reproduce the steps of experiment 1, one must enable private data in the logs. To do this, one can follow the steps in cmdsec's blog (https://www.cmdsec.com/unified-logs-enable-private-data/). The instructions, however, do not work without error. So, the pieces that allow this to work error-free are provided.

1. Sign up for an Apple Developer Account ($99)
2. Create an application certificate
3. Download the certificate

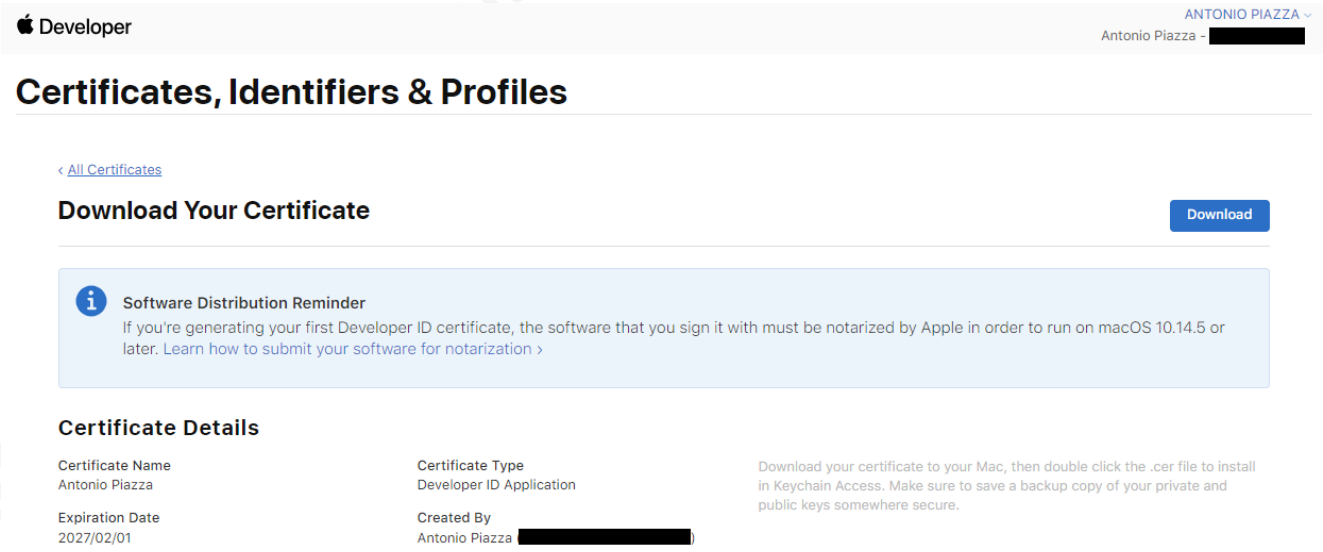

Figure 17 - Apple Developer Certificate Download Page

4. Double-click the downloaded certificate to install it

After downloading the certificate, one can follow the instructions to use it to sign the profile the blog provides.

5. Copy and paste the XML profile from the web blog
6. Save the file as profile.mobileconfig

The instructions state to use the -Z argument (hash) when running the security cms tool to sign the profile with one's Apple Developer Certificate. A bug in macOS causes an error, so one must use the -N argument (nickname) instead of -Z.

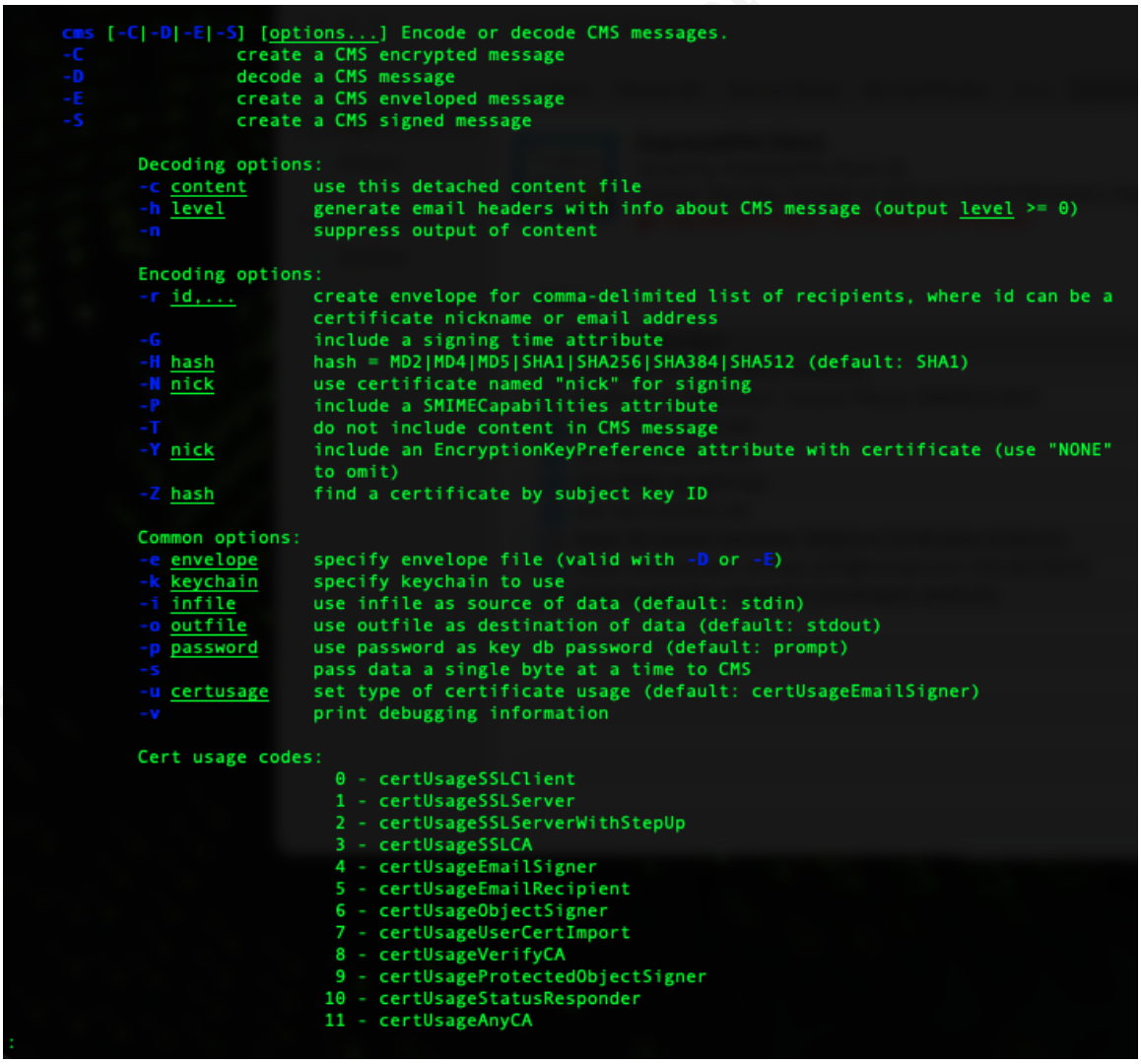

Figure 18 - Man page for the macOS "security" tool and the "cms" function

Also, there is an issue with the macOS security cms command choosing the wrong certificate even though one uses the correct name. Security cms chooses the last certificate displayed in one's keychain over one's recently installed Apple Developer Application Certificate. Removing all the certificates displayed below one's Apple Developer Application Certificate, leaving it displayed at the bottom of the keychain will

allow one to sign the profile without error. After signing the mobile profile with the developer certificate, one can reinstall the removed certificates.

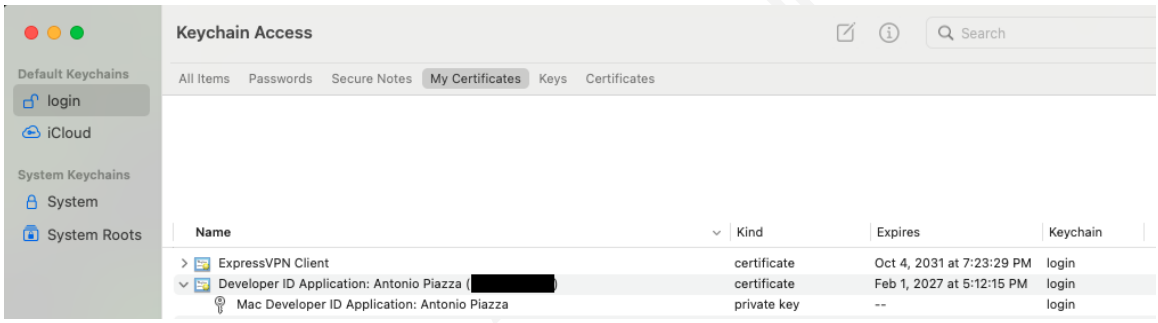

Figure 19 - Keychain Access (Developer certificate at the bottom of the list. ExpressVPN certificate reinstalled)

7. Sign the mobile configuration profile: /usr/bin/security cms -S -N "DeveloperID Application: Antonio Piazza (…)" -I /Path/to/unsigned/profile.mobileconfig" -o "output/path/to/signed/profile.mobileconfig"

```
~ λ /usr/bin/security find-identity -p codesigning -v
  1) 30981D7CC1FE65A9DF750D99F352E6F3C145AF33 "Apple Development: [redacted])"
  2) 271D7FAB238049307176D8C9A5F1831107D86FF4 "Developer ID Application: Antonio Piazza ([redacted])"
     2 valid identities found
~ λ /usr/bin/security cms -S -N "Developer ID Application: Antonio Piazza ([redacted])" -i "/Path/to/unsigned/profile.mobileconfig"
 -o "/output/Path/to/signed/profile.mobileconfig
```

Figure 20 - List codesigning certificates and sign the mobile configuration profile

8. Double-click the signed mobile config file to install it.

If one navigates to Settings > Profiles, one will see that one has installed the new mobile profile.

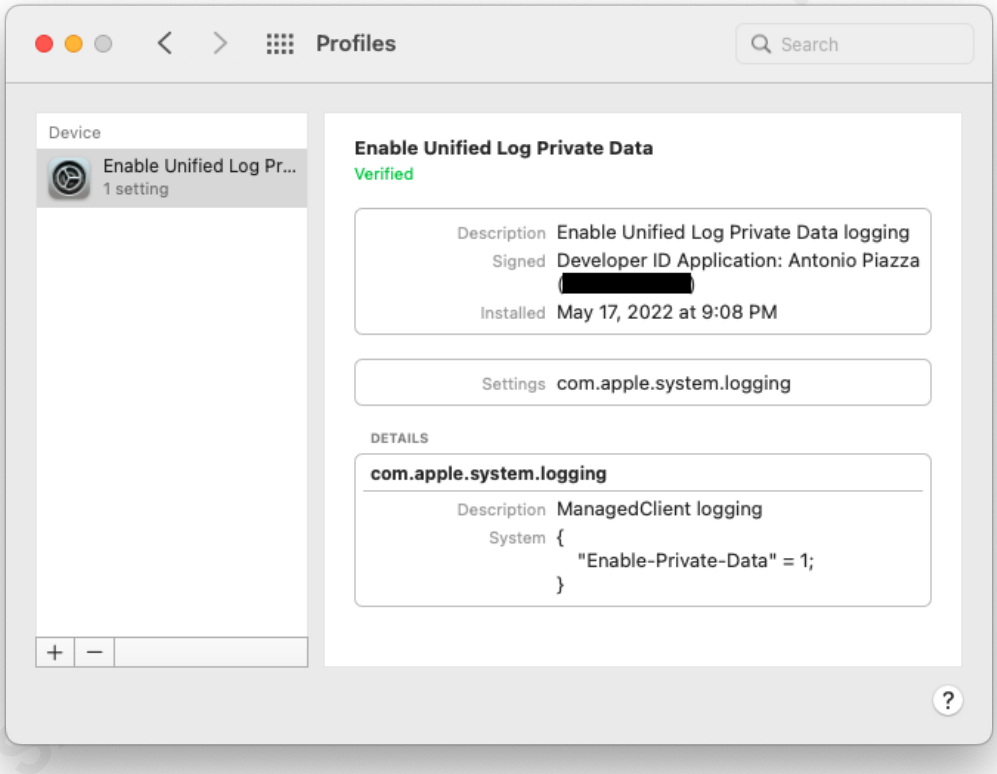

Figure 21 - Installed, signed mobile profile

After successfully installing the signed mobile profile, private information will now appear in the logs.

An Elastic, Logstash, Kibana (ELK) stack with Filebeats was used to demonstrate how to forward macOS System Logs to a SIEM. Specifically, the HELK implementation was used from Roberto Rodriguez's GitHub (https://github.com/Cyb3rWard0g/HELK).

The antman1p blog post explains how to forward macOS ESF event messages to HELK step-by-step. Still, one will need to make slight changes to these steps to forward macOS System Logs instead (https://antman1p-30185.medium.com/acting-red-seeing-blue-b04dd845c3dc).

Antonio Piazza, antman_007@hotmail.com

9. Disregard the step explaining the need to modify “HELK/docker/helk-logstash/pipeline/0098–all-filter.conf.”
10. Instead of naming your log file path using “esf” in the name, change it to something more relevant like “system_logs.log.”
11. Instead of downloading and running Appmon, as explained, run the macOS log command. If one wishes to forward all system logs, run this command: log stream >> <path/to/ system_logs.log >. One can forward less log data by using predicate filters. For instance, if one only wants to forward syspolicyd logs: log stream --predicate 'process == "syspolicyd"' >> <path/to/ system_logs.log >.

The rest of the blog post instructions remain the same. One might now find the GateKeeper bypass detection on the Kibana web interface.

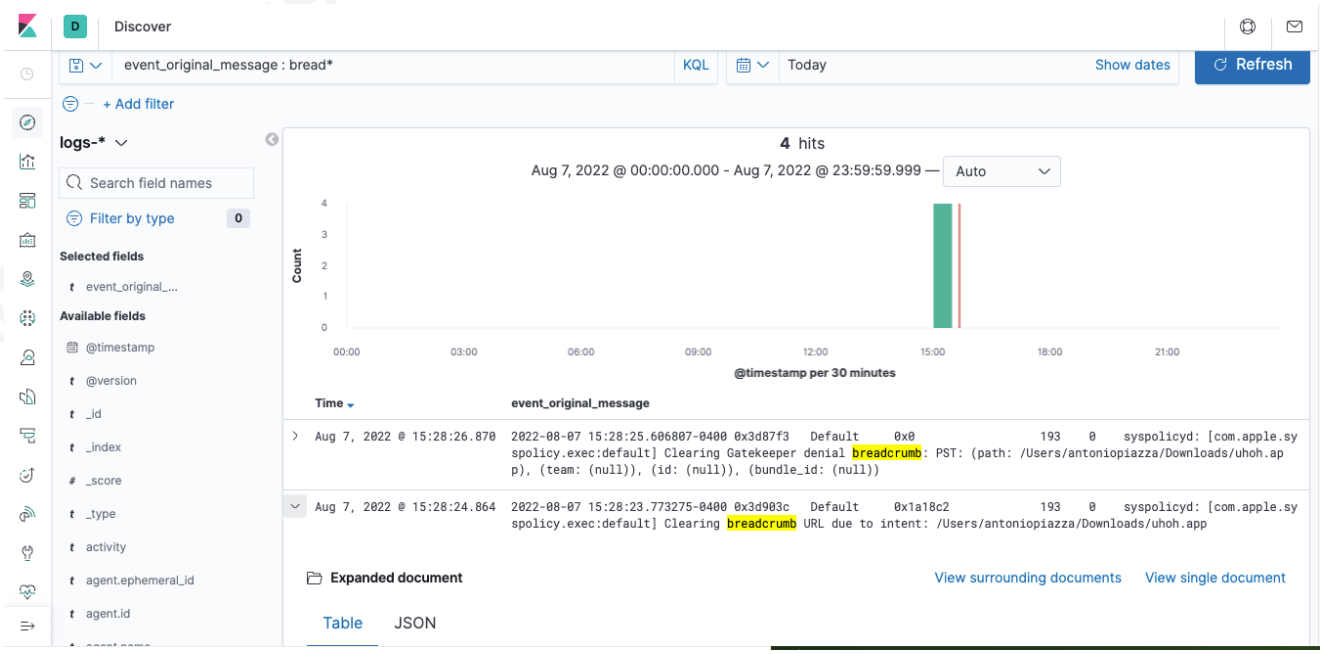

Figure 22 - GateKeeper Bypass Log Detection in Kibana