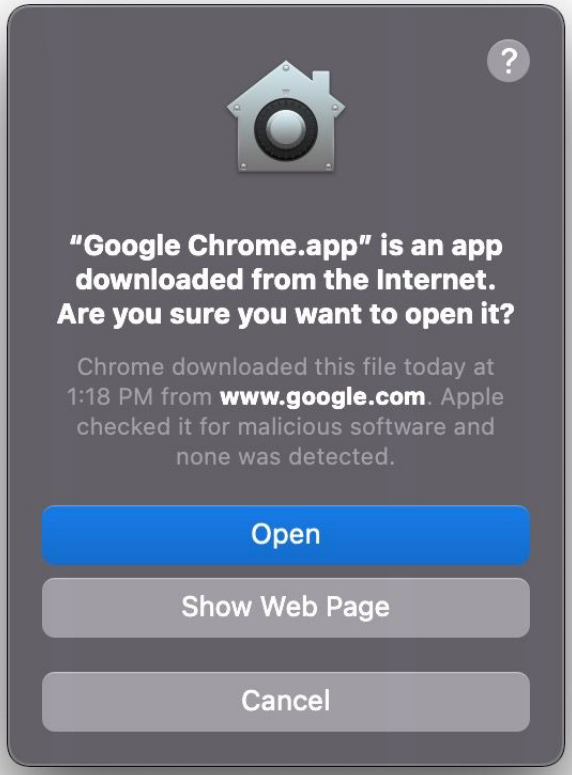
Jamf Threat Labs identifies Safari vulnerability allowing for Gatekeeper bypass

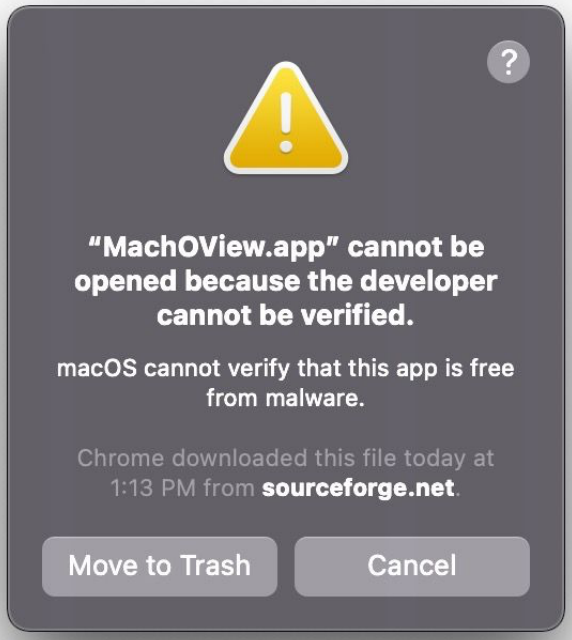
Note: *A special thanks goes out to Ferdous Saljooki who discovered the vulnerability and Jaron Bradley who provided additional research support*.

What is Gatekeeper?

Gatekeeper is a security feature built into macOS that prevents the user from executing potentially malicious and/or unwanted software. It is designed to ensure that only trusted software launches on a user's system by verifying notarization and code signing information.

Notarization is a review process in which a developer submits their software to Apple, where it is scanned to ensure it is free of malicious code. Gatekeeper also plays a role in requesting the user's explicit permission *before* executing software downloaded from the Internet and when launched for the first time.


"Google Chrome.app" is an app downloaded from the Internet. Are you sure you want to open it?
Chrome downloaded this file today at 1:18 PM from www.google.com. Apple checked it for malicious software and none was detected.
Open
Show Web Page
Cancel

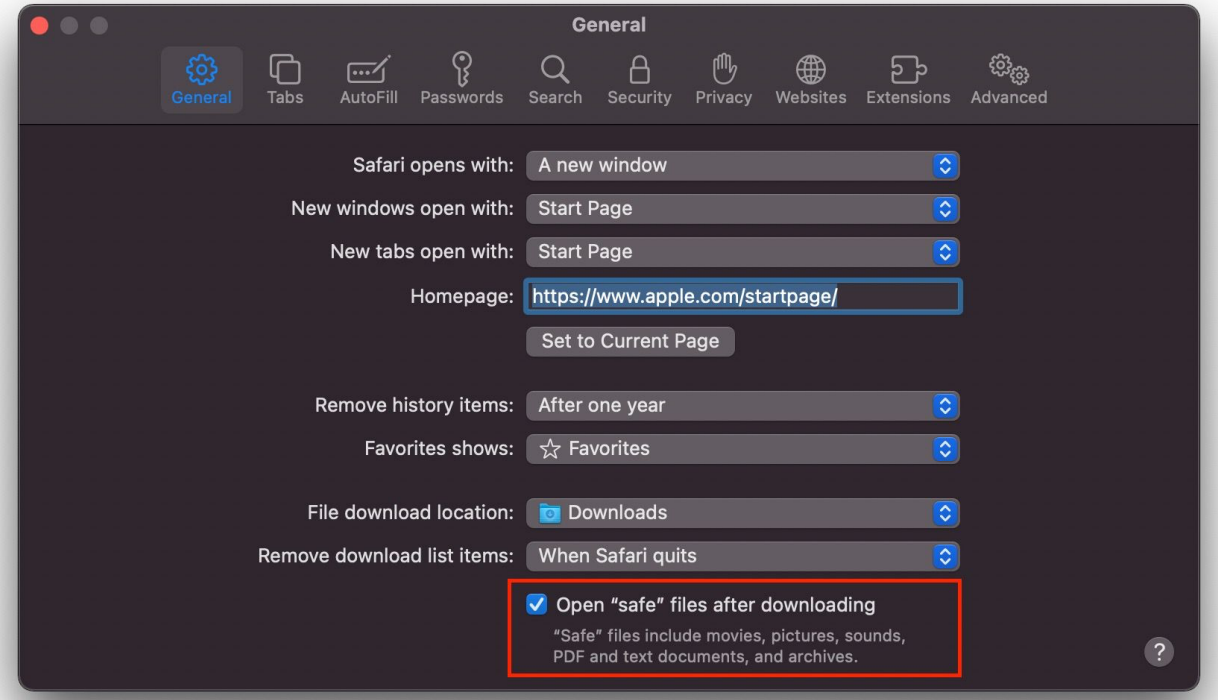

"MachOView.app" cannot be opened because the developer cannot be verified.
macOS cannot verify that this app is free from malware.
Chrome downloaded this file today at 1:13 PM from sourceforge.net.
Move to Trash
Cancel

This is all due to the fact that applications downloaded for the first time have an extended attribute placed on them titled com.apple.quarantine. After the user has chosen to run the program — regardless of the Gatekeeper prompts — the operating system will no longer prompt the user when launching this specific application in the future.

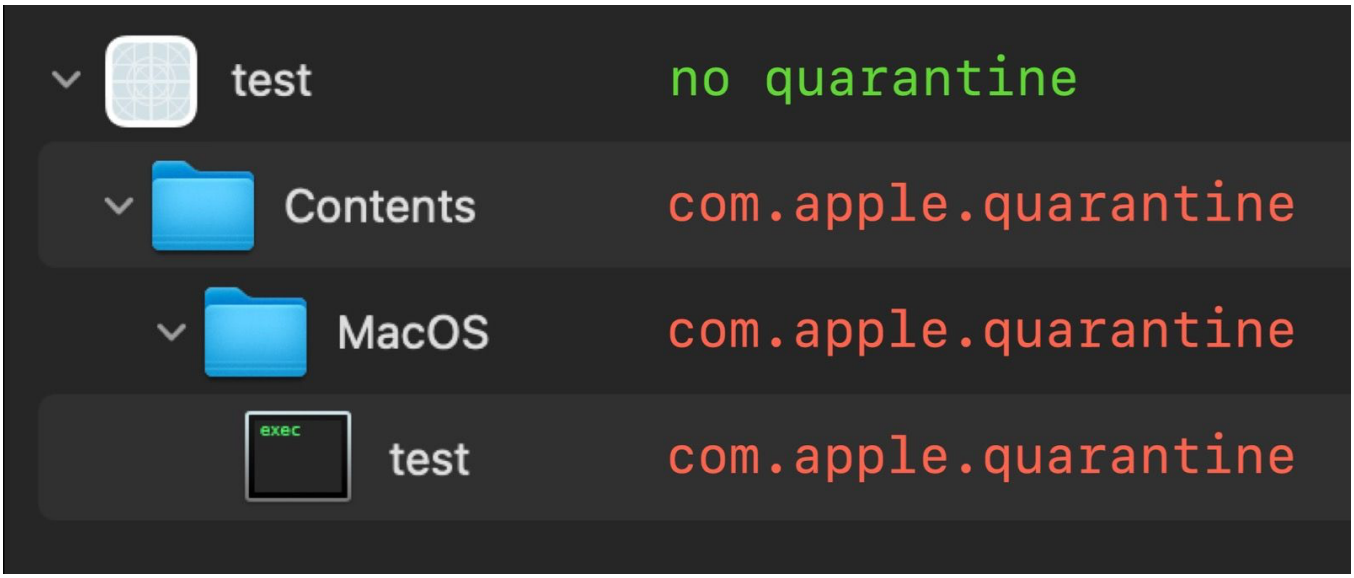
Initial Indication of Vulnerability

While investigating websites that host a large number of third-party macOS applications, we came across the popular game hosting website itch.io, where many independent game developers host their games. After downloading and opening a handful of games, we noticed that many would not trigger an alert from Gatekeeper. This made us very curious because not only were these applications being downloaded from the Internet but most were completely unsigned. Upon further testing, we noted that we would only receive the expected Gatekeeper prompts *if* the application was downloaded from a third-party web browser, such as Google Chrome.

We did note that many applications were delivered in the form of a zip file. By default, Safari has a built-in feature that is enabled that will automatically unzip applications held within a zip file after downloading them.

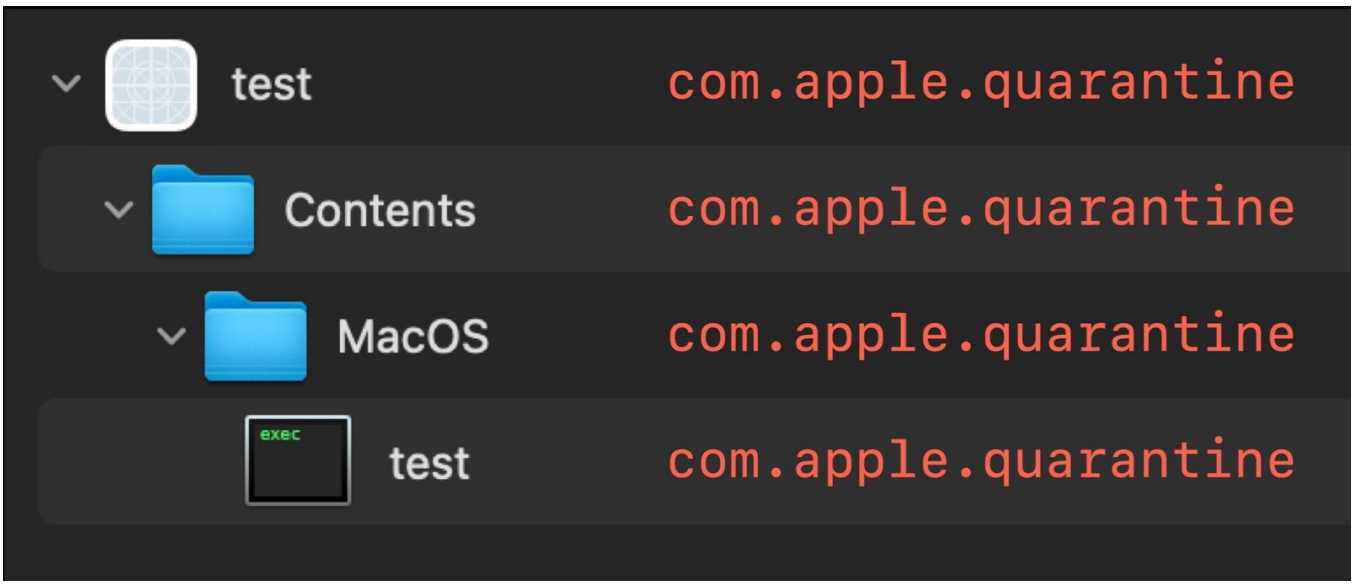


Oftentimes, when an application was automatically unzipped from this website, it did not have the `com.apple.quarantine` extended attribute applied to it. It did however have the quarantine attribute on all files under the Contents directory.



Oddly enough, if we disabled the Safari auto-extraction feature and manually unzipped the file by double-clicking it, the application would

have the extended attribute set as expected.



The process responsible for automatically unzipping downloaded applications is `com.apple.Safari.SandboxBroker`. Since the bug only seems to exist when an auto-extraction occurs, we suspected a bug in this process.

Pinpointing the Vulnerability

This of course raised the question, "What's so special about the zip files hosted on this website?" Fortunately, we were able to answer this question thanks to the great documentation that the site provides as to how games are handled. Developers are encouraged to use an open-source Go-Lang command-line tool called Butler. This tool allows them to easily manage the games which are hosted on their account. Butler analyzes an uploaded application and re-zips it before hosting it for download. We surmised the way in which it was re-zipping the application was causing the bypass to accidentally occur, having no reason to believe that the website creators were aware of the bug.

For testing purposes, we built an application and ran it through the Butler command-line tool. After trying a few different combinations, we were indeed able to skip the Gatekeeper prompts when our custom zip file was

downloaded via Safari. Upon comparing the zip file created with Butler to a zip file we created using a standard `zip` command-line utility, we noticed a subtle difference.

```
00000000: 504b 0304 0a00 0000 0000 9bb8 6f54 0000  PK..........oT..
00000010: 0000 0000 0000 0000 0000 0900 1c00 7465  ..............te
00000020: 7374 2e61 7070 2f55 5409 0003 d553 3162  st.app/UT....S1b
00000030: d553 3162 7578 0b00 0104 f501 0000 0414  .S1bux..........
00000040: 0000 0050 4b03 040a 0000 0000 00c9 996f  ...PK..........o
00000050: 5400 0000 0000 0000 0000 0000 0012 001c  T...............
00000060: 0074 6573 742e 6170 702f 436f 6e74 656e  .test.app/Conten
00000070: 7473 2f55 5409 0003 c91d 3162 d553 3162  ts/UT.....1b.S1b
00000080: 7578 0b00 0104 f501 0000 0414 0000 0050  ux.............P
00000090: 4b03 040a 0000 0000 00c9 996f 5400 0000  K..........oT...
000000a0: 0000 0000 0000 0000 0021 001c 0074 6573  .........!...tes
000000b0: 742e 6170 702f 436f 6e74 656e 7473 2f5f  t.app/Contents/_
000000c0: 436f 6465 5369 676e 6174 7572 652f 5554  CodeSignature/UT
000000d0: 0900 03c9 1d31 62d5 5331 6275 780b 0001  .....1b.S1bux...
```

```
00000000: 504b 0304 1400 0800 0000 c801 7054 0000  PK..........pT..
00000010: 0000 0000 0000 0000 0000 1200 0900 7465  ..............te
00000020: 7374 2e61 7070 2f43 6f6e 7465 6e74 732f  st.app/Contents/
00000030: 5554 0500 01c9 1d31 6250 4b07 0800 0000  UT.....1bPK.....
00000040: 0000 0000 0000 0000 0050 4b03 0414 0008  .........PK.....
00000050: 0000 00c8 0170 5400 0000 0000 0000 0000  .....pT.........
00000060: 0000 0021 0009 0074 6573 742e 6170 702f  ...!...test.app/
00000070: 436f 6e74 656e 7473 2f5f 436f 6465 5369  Contents/_CodeSi
00000080: 676e 6174 7572 652f 5554 0500 01c9 1d31  gnature/UT.....1
00000090: 6250 4b07 0800 0000 0000 0000 0000 0000  bPK.............
000000a0: 0050 4b03 0414 0008 0000 00c8 0170 5400  .PK..........pT.
000000b0: 0000 0000 0000 0000 0000 0018 0009 0074  ...............t
000000c0: 6573 742e 6170 702f 436f 6e74 656e 7473  est.app/Contents
000000d0: 2f4d 6163 4f53 2f55 5405 0001 c91d 3162  /MacOS/UT.....1b
```

What we noticed is that the root of the zip file points to two different locations. On one, it is pointed at the `test.app` folder. On the other, it is set to the `test.app/Contents` directory.

Bill of Materials

To better understand the issue at hand, it helps to have some background on the Bill of Materials (BoM). These objects are used in many different places by Apple to keep a running list of file paths when a major file

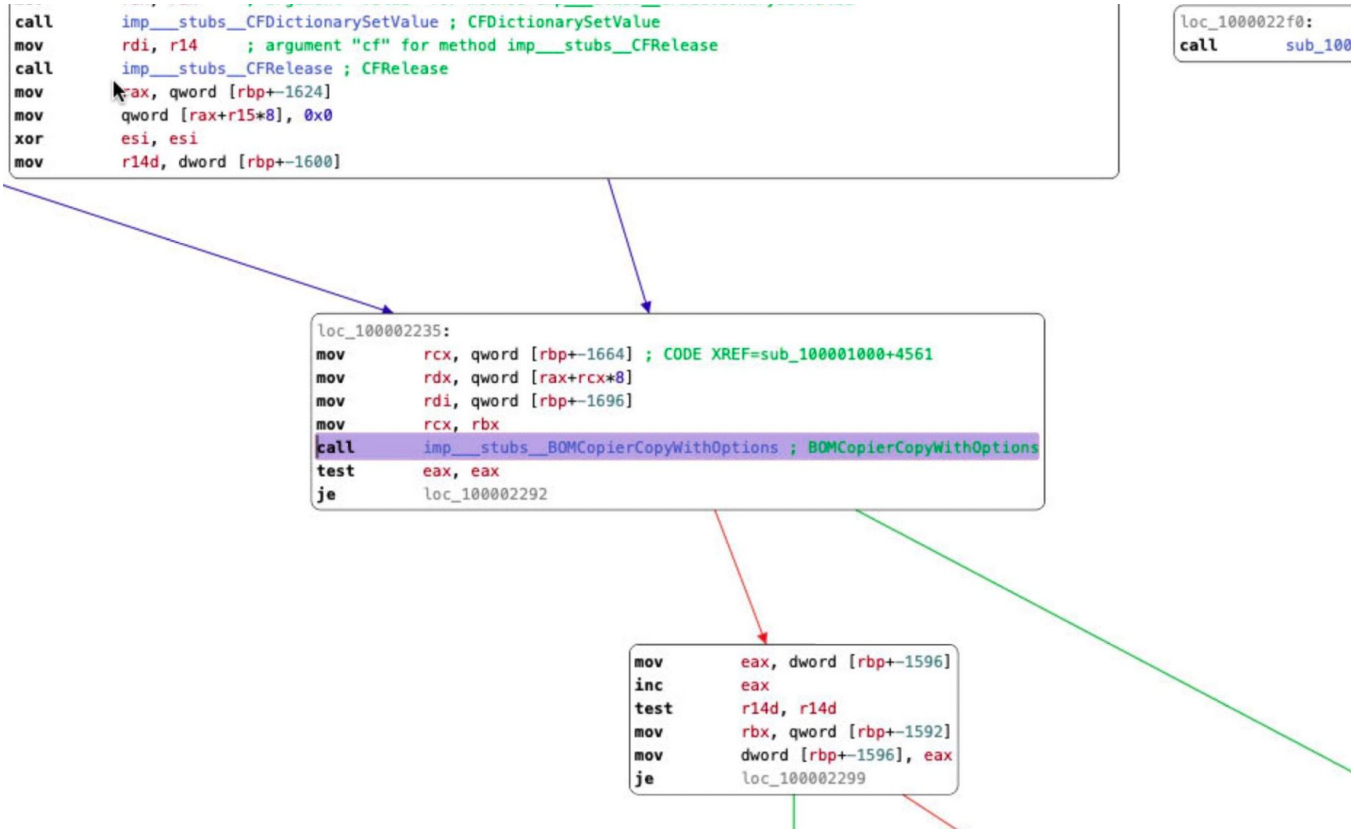
transaction occurs. For example, if an individual sends a directory to another user over AirDrop, a temporary BoM file is created on the sender's system that holds a record of all the files that were sent in that transaction. BoM items can even be saved to files in binary format (using `mkbom`). Many power users are familiar with the `/System/Library/Receipts` directory, which holds a list of BoM files. The name of the BoM file tells you what software was installed or updated. Looking inside the BoM file tells you which files were created by that software. Using the `lsbom` command-line utility, BoM files can be printed on-screen.

```
>>> lsbom /var/db/receipts/com.apple.pkg.Keynote11.bom | more
.       40755   0/0
./Applications  40775   0/80
./Applications/Keynote.app      40755   0/0
./Applications/Keynote.app/Contents     40755   0/0
./Applications/Keynote.app/Contents/Frameworks  40755   0/0
./Applications/Keynote.app/Contents/Frameworks/EquationKit.framework    40755   0/0
./Applications/Keynote.app/Contents/Frameworks/EquationKit.framework/EquationKit        120755  0/0     28      3166
./Applications/Keynote.app/Contents/Frameworks/EquationKit.framework/Resources  120755  0/0     26      3302263027
./Applications/Keynote.app/Contents/Frameworks/EquationKit.framework/Versions   40755   0/0
./Applications/Keynote.app/Contents/Frameworks/EquationKit.framework/Versions/A 40755   0/0
./Applications/Keynote.app/Contents/Frameworks/EquationKit.framework/Versions/A/EquationKit     100755  0/0     2914
./Applications/Keynote.app/Contents/Frameworks/EquationKit.framework/Versions/A/Resources       40755   0/0
./Applications/Keynote.app/Contents/Frameworks/EquationKit.framework/Versions/A/Resources/Info.plist    100644  0/0
./Applications/Keynote.app/Contents/Frameworks/EquationKit.framework/Versions/A/Resources/version.plist 100644  0/0
./Applications/Keynote.app/Contents/Frameworks/EquationKit.framework/Versions/A/_CodeSignature  40755   0/0
./Applications/Keynote.app/Contents/Frameworks/EquationKit.framework/Versions/A/_CodeSignature/CodeResources    1006
./Applications/Keynote.app/Contents/Frameworks/EquationKit.framework/Versions/Current   120755  0/0     1       1751
./Applications/Keynote.app/Contents/Frameworks/KNAnimation.framework    40755   0/0
./Applications/Keynote.app/Contents/Frameworks/KNAnimation.framework/KNAnimation        120755  0/0     28      2950
./Applications/Keynote.app/Contents/Frameworks/KNAnimation.framework/Resources  120755  0/0     26      3302263027
./Applications/Keynote.app/Contents/Frameworks/KNAnimation.framework/Versions   40755   0/0
./Applications/Keynote.app/Contents/Frameworks/KNAnimation.framework/Versions/A 40755   0/0
```

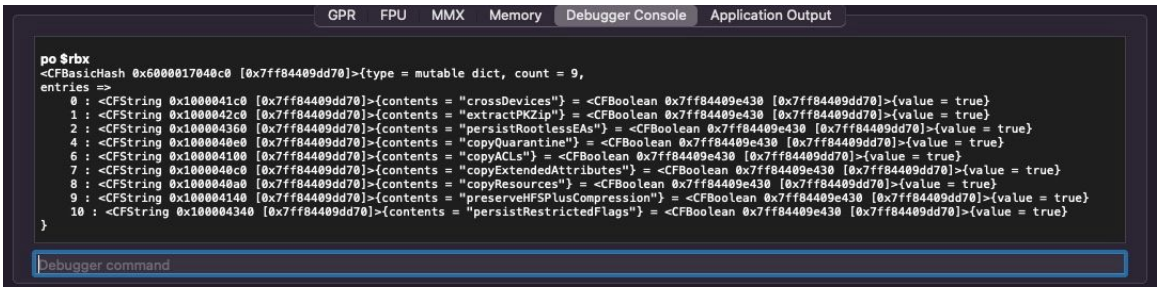
This can further be seen via the `ditto` executable which is a fantastic macOS utility for making exact copies of files. Ditto can also be used to create archives as well as extract archives. In fact, if we use a different browser to download the same test application used above (that doesn't automatically unzip applications), we can then use the ditto executable to unarchive the zip file resulting in the same lack of quarantine extended attributes.

```
→ Downloads xattr test.zip
com.apple.quarantine
→ Downloads
→ Downloads ditto -x -k test.zip extracted 2>/dev/null
→ Downloads
→ Downloads xattr extracted/test.app
→ Downloads
→ Downloads xattr extracted/test.app/Contents
com.apple.quarantine
→ Downloads
```

The reason behind this is that Safari and Ditto share the same logic calling a low-level function named `BOMCopierCopyWithOptions`, located within `/System/Library/PrivateFrameworks/Bom.framework`.



In a debugger, we can see the options that are passed to this function when `ditto` is used in unarchiving a zip file.



Most notable here is the `extractPKZip` and `copyQuarantine` options. Although there is no documentation on BoM files being used with zip files, this function seems to imply there might be some low-level functionality that uses a BoM object to keep track of files extracted from zip files. Presumably, if we can keep our application's root directory off this list of files, the extended attribute will not be applied to it.

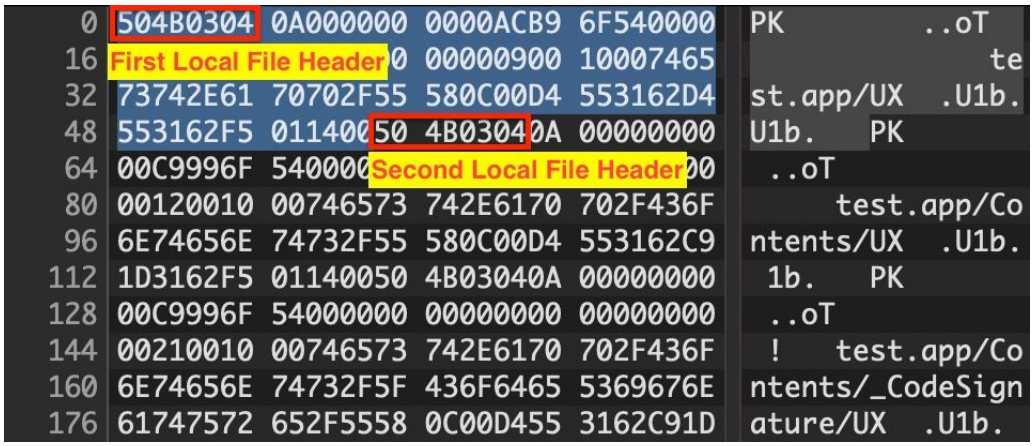
Crafting the Bypass

As mentioned earlier, if the quarantine attribute gets applied to files or folders under the test.app directory, Gatekeeper is not triggered. The application itself is all Gatekeeper looks for.

We determined that the easiest way to confuse the BoM function is by ensuring that the root of the application is not the first local file header of the zip file. For example, we took an unsigned application named `test.app` and placed it in a zip file using the following command:

```
ditto -c -k --keepParent path_to_unsigned_application zip_file
```

We then opened this zip file in a hex editor and removed the entire first Local File Header.



A simplified solution with less chance of breaking the central directory record would be to zip an application using the following command:

```
zip –r test.zip test.app/Contents
```

This causes the `test.app/Contents` directory to be the first local file header and ultimately results in a bypass of Gatekeeper once the zip file is downloaded via Safari.

Detections

Jamf Protect offers a holistic Mac endpoint security solution that provides analytics that detects anytime this vulnerability is potentially being abused. It does this by monitoring the `com.apple.Safari.SandboxBroker` process and detecting when it automatically unzips an application. It goes on to verify the extended attributes of the extracted app, effectively detecting a missing `com.apple.quarantine` attribute.

Conclusion

In our testing, we observed this detection working as far back as Safari 14.0.2 on macOS Big Sur. Apple has patched this vulnerability in Safari 15.4.

Mickey Jin (@patch1t) also stumbled upon this bug at the same time Jamf Threat Labs did, however, he discovered that the bug could be abused somewhat differently via a `gzip` archive.

In the past, we've seen malware authors implement novel ways to bypass Gatekeeper, which indicates that they see value in these types of attacks. For this reason, Jamf Threat Labs remains active in finding ways Gatekeeper can be bypassed, in order to further enhance the security

protections afforded to Jamf Protect customers.

Jamf Protect's endpoint security is purpose-built for macOS to safeguard your data, users and devices.

Contact Jamf, or your preferred representative today to request a trial and get started protecting your fleet of Apple devices.