macOS Gatekeeper Bypass (2021 Edition)

By Cedric Owens

Published: 2021-05-04 · Archived: 2026-04-06 00:26:41 UTC

Abusing the bug in syspolicyd

Preparation: I first ensured that Gatekeeper was enabled and even set it to the most restrictive level on the target macOS host. Next I stood up a new host with Mythic C2 (https://github.com/its-a-feature/Mythic), created a new unsigned Mythic Poseidon golang macho payload, and hosted the payload at http://192.168.1.191:8000/bad-unsigned-macho. Later I will point my shell script downloader to pull this unsigned macho payload down.

Normally, macOS apps tend to have a directory structure as follows:

[Name.app]/Contents/MacOS/Name (this is usually a macho with the same name as the app that is executed when the app is run; since it is a macho, gatekeeper checks to ensure that it has been signed and notarized if it contains the quarantine attribute — "com.apple.quarantine"). And macOS will append the quarantine attribute to any files downloaded from browsers (macOS also appends this attribute to files downloaded via other methods such AirDrop, but for the sake of this article I will not go into those here).

So this begs the question: **what if a script is placed in the Contents/MacOS/ directory instead of a macho, since scripts are not checked by Gatekeeper? That is essentially what this technique does.** Steps are below:

Build A Simple Shell Script Downloader:

Build a simple shell script that uses curl to download a payload (ex: unsigned macho binary, jxa payload, python, etc.) to /tmp , sets the executable bit, and executes that payload. I chose curl because it is known that files pulled down by curl do not have the quarantine attribute appended and I chose the /tmp directory because /tmp is not protected by TCC. An example shell script downloader (down.sh) is below:

The example shell script above pulls a malicious unsigned macho binary down to tmp, sets the executable bit, and runs it backgrounded. Then it displays a message to the user thanking them for installing the fake "IT macOS Provisioner". Again, since curl is used the unsigned macho that is pulled down will not get the quarantine attribute appended (i.e., won't be checked by gatekeeper) and since /tmp is not protected by TCC we can drop payloads there. So now we have our shell script downloader payload ready to use.

Turn This Shell Script Into A Double-Clickable "App":

Next, we will need to put this shell script into the proper format. I did some digging and found that mac admins have actually been running shell scripts for years inside of app directories to make them "double clickable" so they would not need to remember long command line strings when they needed to do a simple task (ex: run a headless browser). In my searching, I found a script named appify had been written 11 years ago that basically turned shell scripts into "apps" that could be double clicked: https://gist.github.com/mathiasbynens/674099. I use the word "apps" loosely since as I note later below this directory structure is missing some key components of what a real app directory structure contains. So I downloaded appify, pointed it at my malicious shell script above and it worked as expected and put my shell script into the proper macOS app directory structure so it could be double-clicked and run like an app. The example below is me running appify (renamed as masquerade.sh) to put my malicious shell script (down.sh) into an app named "FakeApp":

Press enter or click to view image in full size

Example of putting the script into the macOS directory structure, changing it to a clickable "app"

Alternatively, I could have done the same thing manually from Terminal by running:

- ***mkdir -p "FakeApp.app/Contents/MacOS"***
- ***cp down.sh FakeApp.app/Contents/MacOS/FakeApp***

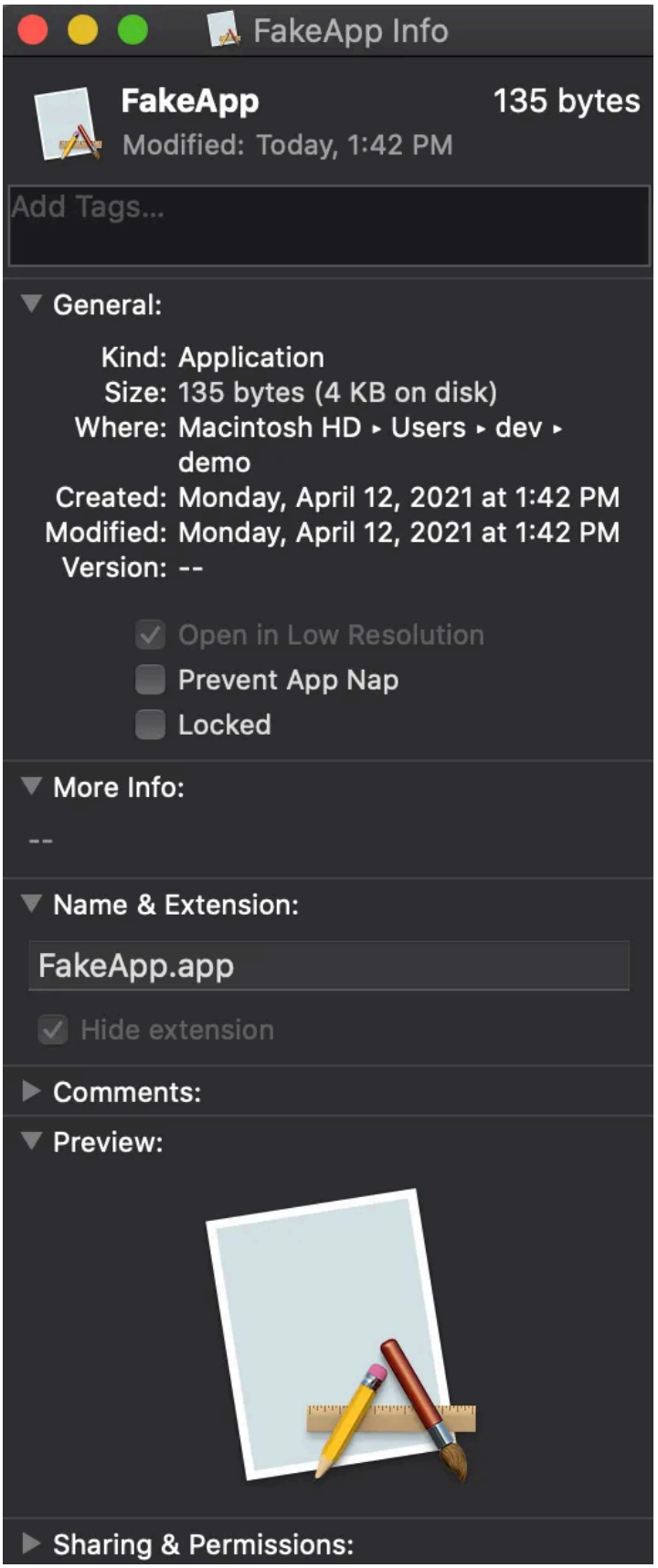
So now we have our malicious shell script disguised as an app:

FakeApp.app/Contents/MacOS/FakeApp (note: this is the shell script downloader down.sh)

Something of note: Apple stated that Info.plist is a required Application Bundle component in their Bundle Structure article here:
https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFBundles/BundleTypes/BundleTypes.html#//apple_r

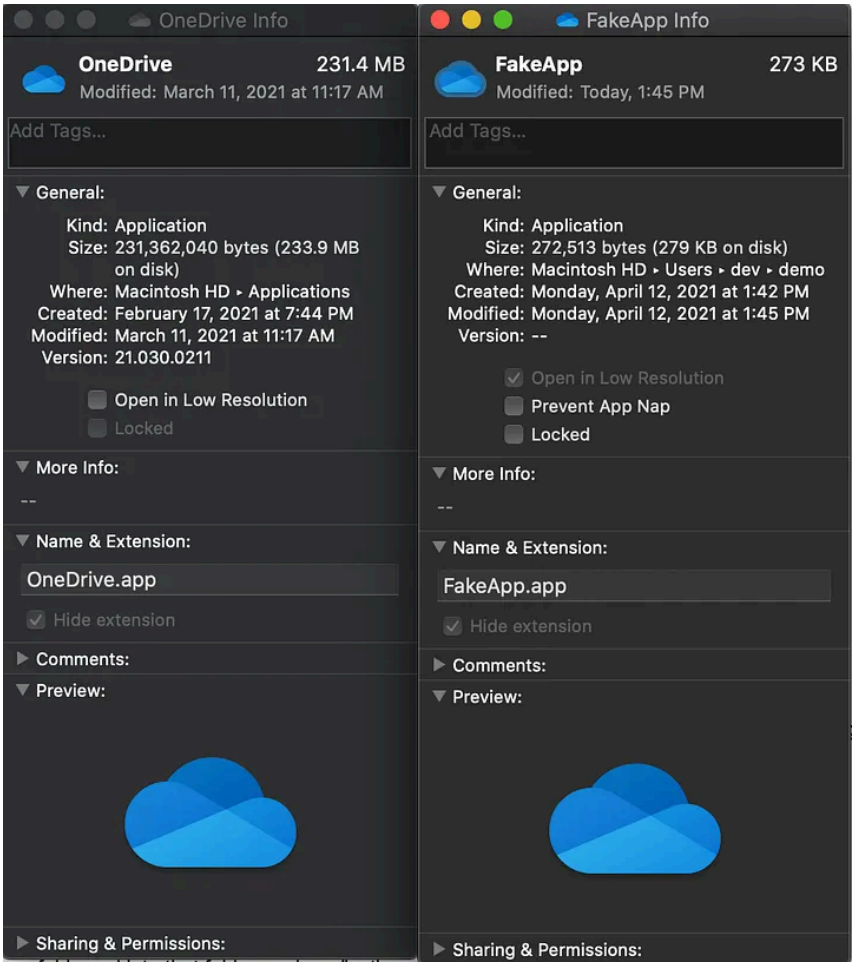
CH101-SW1. However, as you can see in our fake app above, Info.plist is not required for macOS to recognize an app (just the [Name.app]/Contents/MacOS/[Name] app directory structure where Name can be a macho OR a script).

macOS recognizes this as a valid app:



Next, you can pick the app icon from a target app in the /Applications directory. Once you have identified the icon you want to copy, you **Right Click -> Get Info -> Click the icon image -> Edit -> Copy**. This will copy the target icon to the clipboard. Then you can **Right Click on your new FakeApp -> Get Info -> Click the icon -> Edit -> Paste**. This will paste the icon onto your fake app to make it look even more appealing:

Press enter or click to view image in full size



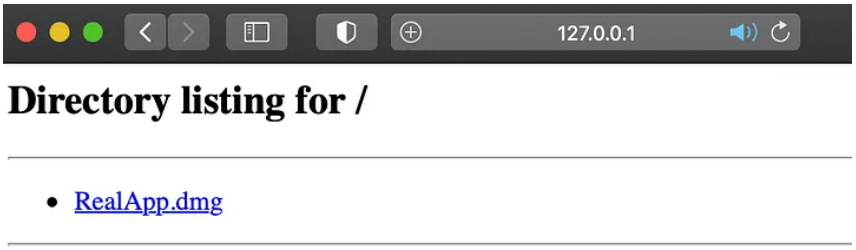
Now that your app is ready to go, you can simply put the app inside of a .dmg (or zip), host it, and share the link. To put inside of a dmg:

1. create a new directory and cp -r FakeApp.app into that new directory

2. **Disk Utility -> File -> New Image -> Image From Folder -> select the folder you created in #1 above**. This will create a new .dmg that contains your fake app.

Now you can host the .dmg and share the link. At this point, you can test what would happen on a target victim machine by placing the .dmg you just created into a folder, cd into that folder, and run “python -m SimpleHTTPServer 80”. In a browser you can then navigate to http://127.0.0.1 and download the .dmg (we want to simulate how a targeted user may download this payload).

Press enter or click to view image in full size



example of hosted payload inside .dmg

Once downloaded, the .dmg (and everything in it) will have the com.apple.quarantine attribute set since it was downloaded via a browser. You can verify this by running ***"xattr ~/Downloads/RealApp.dmg".***

showing that the quarantine attribute is set

Now you can see what would happen on the victim machine:

- double click the downloaded .dmg

- double click the fake app inside of the .dmg

- the shell script payload will run, you will receive a callback on your Mythic C2 server, and you'll see the fake message to the user thanking them for installing the "IT Provisioner Script". Notice that gatekeeper did not prompt or block this

- you will notice that you have un-sandboxed access at this stage (i.e., you will have access to non-TCC protected folders such as some folders inside of the user's home directory and /tmp plus any directories that the user has given Terminal access to). **So even worst case scenario where the user has not granted Terminal full disk access or access to any folders, this payload would still be capable of accessing sensitive files in the user's home directory (ex: ~/.ssh, ~/.aws, etc.).**

To recap, this payload bypassed the following macOS protections:

- **Gatekeeper:** I view this payload as a hybrid app and script. It's an app in the sense that you can double click it and macOS views it as an app when you right click -> Get Info on the payload. Yet it's also shell script in that shell scripts are not checked by Gatekeeper even if the quarantine attribute is present. The bug in syspolicyd allowed the unique combination of properties present in this payload to completely bypass Gatekeeper.
- **App Transport Security (ATS):** ATS usually requires certain Info.plist entries to allow macOS apps to connect to servers. However, this payload does not even have an Info.plist and ATS is not invoked, even though macOS treats the payload as an app.
