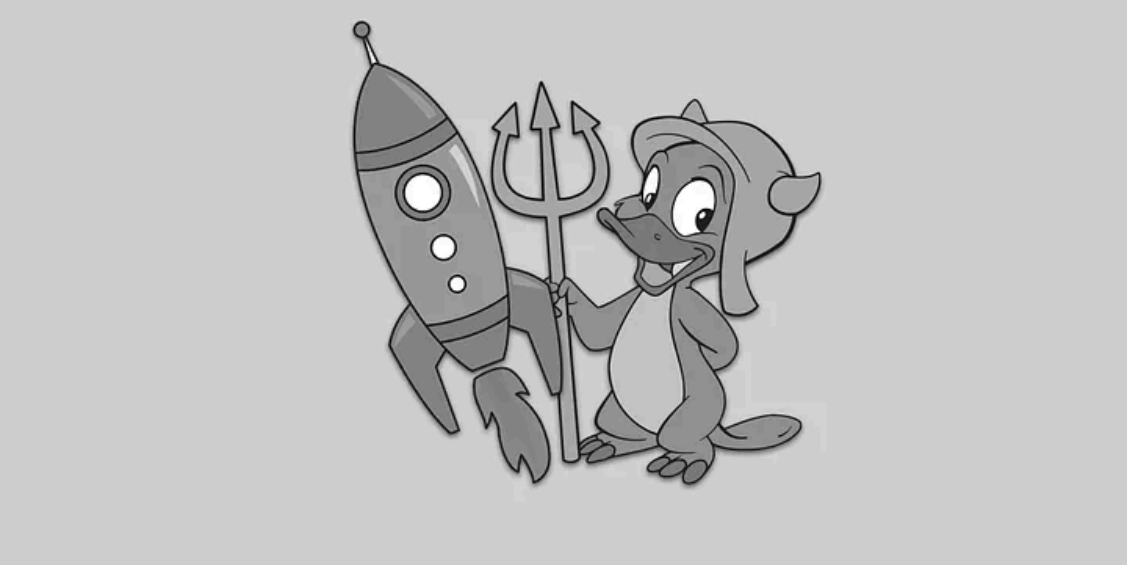
Defeating Malicious Launch Persistence

By 4n7m4n

Published: 2021-11-23 · Archived: 2026-04-05 13:31:10 UTC

Press enter or click to view image in full size



A New Mitigation Strategy for the most used macOS Persistence Technique

Why Launch Persistence Research?

This research started with me looking for suggestions for my organization on mitigating launch persistence techniques for macOS. As a red team operator, sharing knowledge and recommendations with our security organizations is crucial to making our customers safer.

I took a look at the macOS MITRE ATT&CK matrix for the persistence tactic (TA003), its launch persistence technique which MITRE calls "Create or Modify System Process" (T1543), and more specifically, the “Launch Agent” (T1543.001) and “Launch Daemon” (T1543.004) sub techniques. Collectively I am calling these two sub-techniques “launch persistence.” Patrick Wardle calls launch persistence “launch items.” Still, I want to ensure no confusion between these and “login items” or “startup items,” which are entirely different persistence mechanisms.

MITRE suggested setting group policies for mitigation to block launch persistence behavior. The problem was that I hadn't heard of any organization doing this, and I could not find any reference for this from MITRE or otherwise.

I started asking around, and no one could tell me how any organization does this. So, I decided to figure out how an enterprise could mitigate launch persistence at scale.

Upon discovering this new mitigation method, I reached out to my friend Jaron Bradley at Jamf Protect to see if he knew if my approach was possible using an MDM solution like Jamf Pro. He connected me with

, also from Jamf, who I teamed up with on this blog. Matt describes how to prepare this method using Jamf Pro in his section below.

I also reached out to

, macOS lead for MITRE ATT&CK, for some leads in the macOS community. She was a great help in tracking down some individuals with experience in macOS administration who I was able to bounce my mitigation idea off of for a sanity check. Thank you, Cat!

The first section of this blog will cover persistence, launch agents, and why it is important to mitigate the techniques. So, if you already understand this or don't want/need a refresher, you can skip to the next section.

Persistence

Persistence is a vital tactic in the adversarial kill chain. It is typically the next step after initial access and payload/malware execution. "Persistence is the means by which malware ensures that it will be automatically (re)executed by the operating system on system startup or user (re)login" (Wardle, 2020, p. 2). Most malware attempts to gain persistence. Otherwise, a system reboot would kill access to the infected system.

Launch Persistence and launchd

Launch persistence includes launch agents and launch daemons. "Launch items are the Apple recommended way to persist non-application binaries (e.g., software updates, background processes, etc)" (Wardle, 2020, p. 5).

How does this work? After system boot and during system initialization, launchd searches the `/System/Library/LaunchDaemons/` and the root `/Library/LaunchDaemons/` directories for property list files (plists) and launches the daemon that the plist requests to be running at all times. These are launch daemons.

When a user logs in, macOS starts a per-user launchd. This launchd launches the agent requested to be launched on-demand from the property list files found in `/System/Library/LaunchAgents`, the root `/Library/LaunchAgents`, and the user's `~/Library/LaunchAgents` directory. These are launch agents.

Property Lists (plists)

Property lists are very important to understand when discussing launch persistence because they describe the launch persistence to launchd. A plist is typically an XML document that contains key/value pairs. A plist can also be in JSON format, and in more rare cases a binary. We will just focus on the XML plists for this article.

To view the contents of a plist in human-readable form use the following commands:

```
plutil -p <path to plist>
```

`defaults read <path to plist>`

`cat <path to plist>`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC ...>
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>com.redteam.t1543_001</string>
  <key>ProgramArguments</key>
  <array>
  <string>touch</string>
  <string>/tmp/T1543_001.txt</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>NSUIElement</key>
  <string>1</string>
</dict>
</plist>
```

Example Property List

In terms of persistence, the most pertinent key/value pairs include:

1. **Key**: Program or Program Arguments. **Value**: Path to launch persistence's script or binary to execute.
2. **Key**: RunAtLoad. **Value**: Contains a boolean that, if set to *true*, instructs launchd to automatically launch the launch persistence. The malware will be persistently restarted by macOS at system reboot at user re-login.
3. **Key**: Label. **Value**: The job name. "This key is required for every job definition. It identifies the job and has to be unique for the launchd instance. Theoretically, it is possible for an agent to have the same label as a daemon, as daemons are loaded by the root launchd whereas agents are loaded by a user launchd, but it is not recommended by convention job names are written in **reverse domain notation**" (Soma-Zone, *What is Launchd?*). For example *com.redteam.evil*. For private agents, the domain local is a good choice: *local.cleanup.*

These three key/value pairs are all that is needed to create launch persistence.

Launch Persistence and Malware

To persist as a launch agent, malware can create a plist in the *LaunchAgents* directories, and for a launch daemon, it can do the same in the *LaunchDaemons* directory. Malware can create its persistence in the same manner as Apple recommends developers create persistence for their legitimate applications.

Apple's own Launch Agents live in the `/System/Library/LaunchAgents` directory while Launch Daemons live in `/System/Library/LaunchDaemons`. "Since the introduction of System Integrity Protection (SIP) in OS X 10.11 (El Capitan) these OS directories are now protected, therefore malware cannot modify them (i.e., they cannot create a "system" Launch Persistence). As such, malware is now constrained to creating launch persistence in the /Library or ~/Library directories" (Wardle, 2020, p. 6).

Most macOS malware persists using launch persistence. In 2020, 80% of persistent malware persisted via launch persistence. According to "The Mac Malware of 2019" report, all persistent malware persisted via launch persistence. Needless to say, this is a critical technique for organizations to mitigate.

Mitigation

To mitigate the adversarial launch persistence techniques on our home computers, we can use Patrick Wardle of Objective-See's Free and Open Sourced tool, BlockBlock. BlockBlock monitors common persistence locations and alerts whenever a persistent component is added. BlockBlock is a fantastic tool and is excellent for use on your personal mac, but it isn't scalable to the enterprise. We need something that will block an adversary from creating a plist in the launch persistence directories.

Enterprise Mitigation

What I have found in my research is that we can lock the root-level launch persistence directories (`/Library/Launch(Agents || Daemons)`) using a macOS mobile device management (MDM) solution. Assuming that the user of this managed machine does not have access to the local administrative (root) user account on this machine, the MDM admin(s) can lock these launch persistence directories using the root account. Now, only the owner of these directories (root) can unlock them. As it stands, malware, even running as root, would need to unlock the directory, and then create its launch persistence plist in the unlocked directory. I have not seen macOS malware with this bit of sophistication in the wild.

What do you mean by "locked?"

Files and directories on macOS have a flag called the `uchg` , or user immutable flag. If one sets this flag, then the file or directory is "locked." If set, this immutable bit means that the associated file or folder cannot be changed. If we set this user immutable bit on the root launch persistence directories, nothing can add to or remove from the directories. Malware cannot add a plist, which means, without additional actions, no additional launch persistence files can be added.

We can check if a file/directory has the `uchg` flag set by listing the file using the `ls` command with the `-O` option:

```
ls -O <path/to/file/dir>
```

```
$ ls -lahO ~/Library | grep LaunchAgents
drwxr-xr-x@   3 AntonioPiazza  staff  uchg    96B Nov  2 11:39 LaunchAgents
```

LauchAgents directory is immutable

We can set or unset this immutable bit using the default macOS tool, `chflags` .

To add the immutable bit:

```
$ chflags uchg /Library/LauchAgents
```

To remove it:

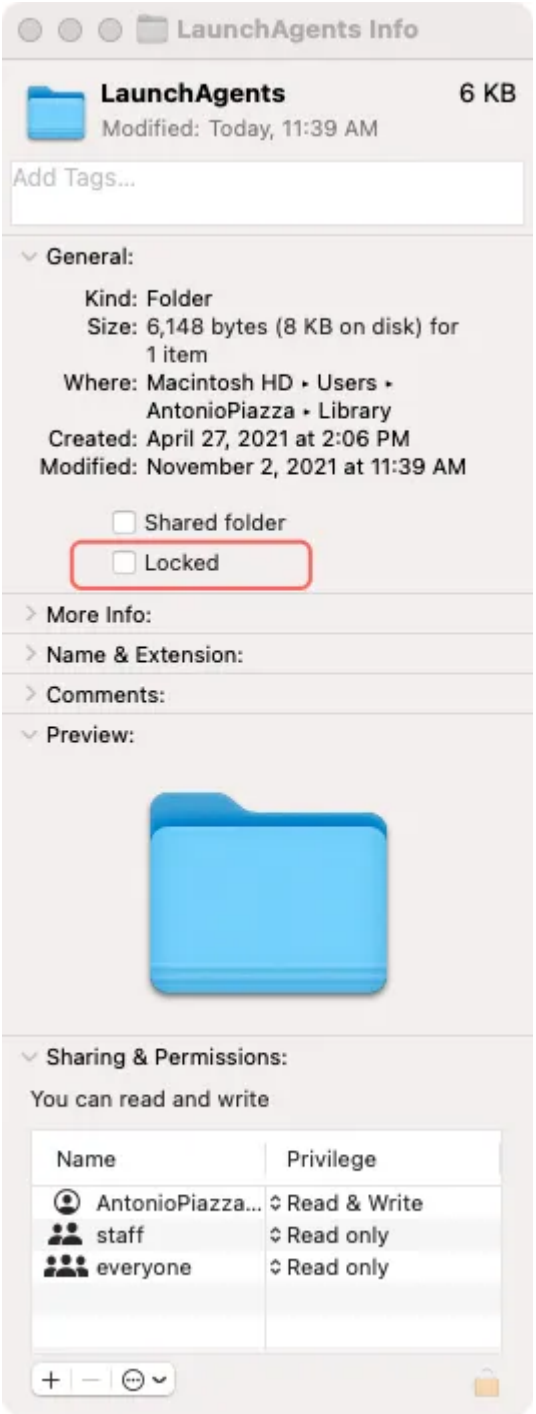
```
$ chflags nouchg /Library/LauchAgents
```

Get 4n7m4n's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

You can also change the bit using the file information window, which you can access by pressing `command + i` . You can then check or uncheck the "Locked" check box. This method is not helpful in using an MDM for scripting the locking at scale but works if you'd like to do it on your personal macOS machine.



Unlocked LaunchAgents directory via File Information

Will this break application functionality?

When I told my colleagues about this mitigation method, they expressed concerns for legitimate applications that edit the plists they have created in the launch persistence directories. If an application loses its ability to write to its plist, this mitigation method might break application functionality. I agreed and did some testing.

It turns out that with the immutable bit set, I was still able to edit and completely overwrite a plist inside these locked directories. So, even though nothing can add or remove a file in the locked launch persistence directories, one can still change the files inside. While this is excellent news for my mitigation method, this seems like immutable does not equal immutable. *I reported this as a bug to Apple on 11/8/2021 and am awaiting a response.*

Note:

informed me that one could use `chflags -R uchg /Library/LauchAgents` (`-R` option) and this will recursively lock the directory and all of its contents. However, because many services frequently update their plists this option is not ideal in that it may break service/application functionality.

What about the USER LaunchAgents directory?

So, the local user account doesn't own the root launch persistence directories. Therefore it cannot unlock the folder. Still, since the user owns the `/Users/<username>/Library/LaunchAgents` directory, they can just unlock it, negating the mitigation in userland. This user LaunchAgent directory might not be necessary for an enterprise environment. Most approved software that I've seen lately in an enterprise environment doesn't need to install persistence here, so we need a way to lock it down permanently.

Our MDM solution can create an unprivileged user account on each managed macOS system. Using this unprivileged administrative account, the MDM administrator can create the LaunchAgent directory for our real user and change ownership of the directory to the unprivileged administrative user account. The MDM administrator can then lock the directory. Now the actual user cannot unlock the directory because they, again, do not own it.

This new unprivileged administrative user can be considered a honey-user account. Defense should monitor it, and any attempts to log in to the account should create an alert and probably be regarded as malicious.

How do we do this at scale?

Applications only need to add/create a plist to the root launch persistence directories at install time. If your IT administrators manage installations, they can install approved software using an MDM solution with scripting capabilities.

To explain how an administrator can configure this in an enterprise environment, I have teamed up with

from Jamf.

Protecting Launch Directories with Jamf Pro

The following is an example of how an admin might use Jamf Pro to lock the `LaunchDaemon` and `LaunchAgent` directories across a macOS fleet. This workflow highlights key Jamf Pro features, including scripting via the Jamf Pro binary, reporting and monitoring via custom inventory attributes, and scoping actions with a dynamic Smart Computer Group.

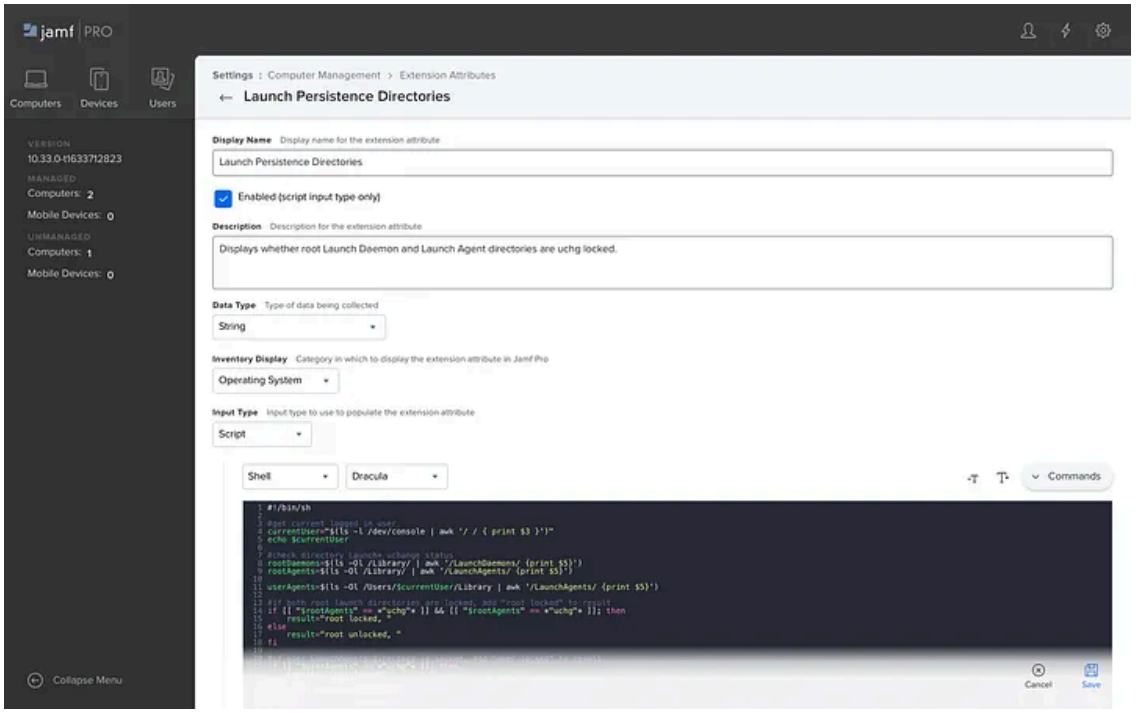
Reporting on uchg Status

The first step of the Jamf Pro workflow is to be able to determine whether the launch persistence directories have been `uchg` locked. We do this by creating a Jamf Pro Extension Attribute that is updated during the routine Jamf Pro inventory update. This custom inventory attribute checks the `uchg` status of the launch persistence directories and updates each computer's inventory record.

As Anthony demonstrated, you can see the `uchg` status by listing the file or directory using the ls command with the `-lO` option: `ls -lO <path/to/file/dir>` .

In this example, we have created a custom Extension Attribute called "Launch Persistence Directories". The Extension Attributes work by running a snippet of bash script that checks the `uchg` status of each launch persistence directory location and returns the result.

Press enter or click to view image in full size



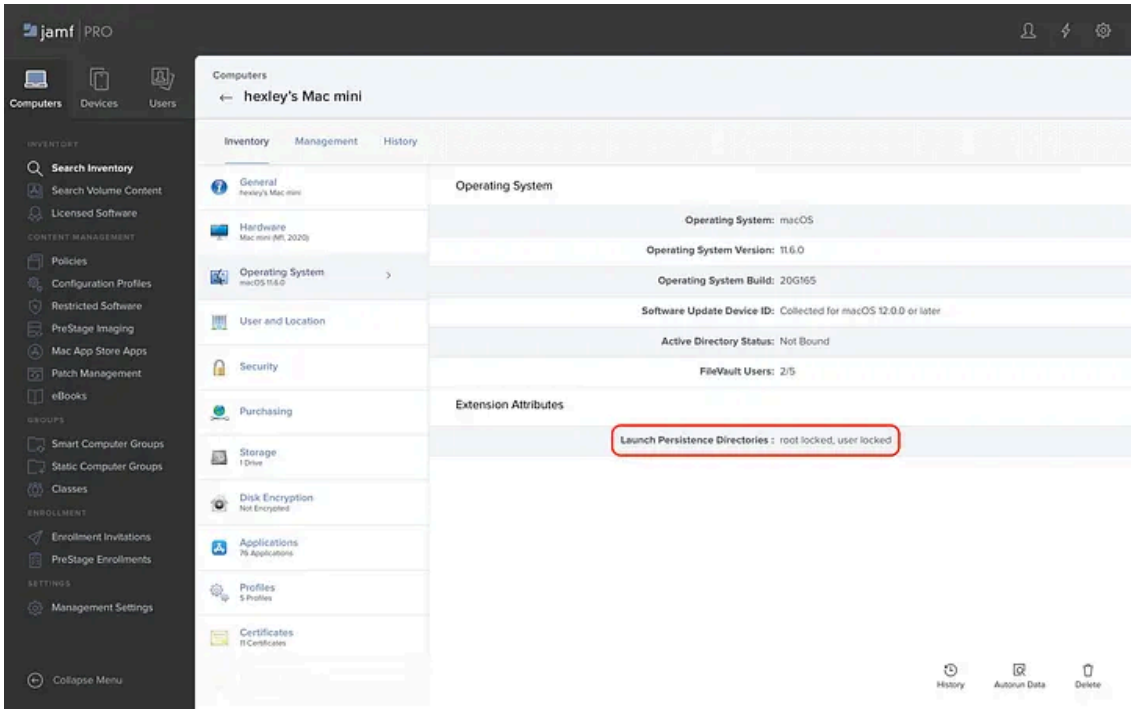
We can check the flags of a file or directory by running ls with the -Ol flag. We can then grab the `uchg` flag status by using `awk` to capture the fifth column, put that output into variables, and then test the variable values for the substring `uchg` .

Press enter or click to view image in full size

```
#!/bin/sh

#get current logged in user
currentUser="$(ls -l /dev/console | awk '/ / { print $3 }')"
echo $currentUser

#check directory Launch* uchange status
rootDaemons=$(ls -Ol /Library/ | awk '/LaunchDaemons/ {print $5}')
rootAgents=$(ls -Ol /Library/ | awk '/LaunchAgents/ {print $5}')

userAgents=$(ls -Ol /Users/$currentUser/Library | awk '/LaunchAgents/ {print $5}')

#if both root launch directories are locked, add "root locked" to result
if [[ "$rootAgents" == *"uchg"* ]] && [[ "$rootAgents" == *"uchg"* ]]; then
    result="root locked, "
else
    result="root unlocked, "
fi

#if user LaunchAgents directory is locked, add "user locked" to result
if [[ "$userAgents" == *"uchg"* ]]; then
    result+="user locked"
else
    result+="user unlocked"
fi

#return result
echo "<result>$result</result>"
```

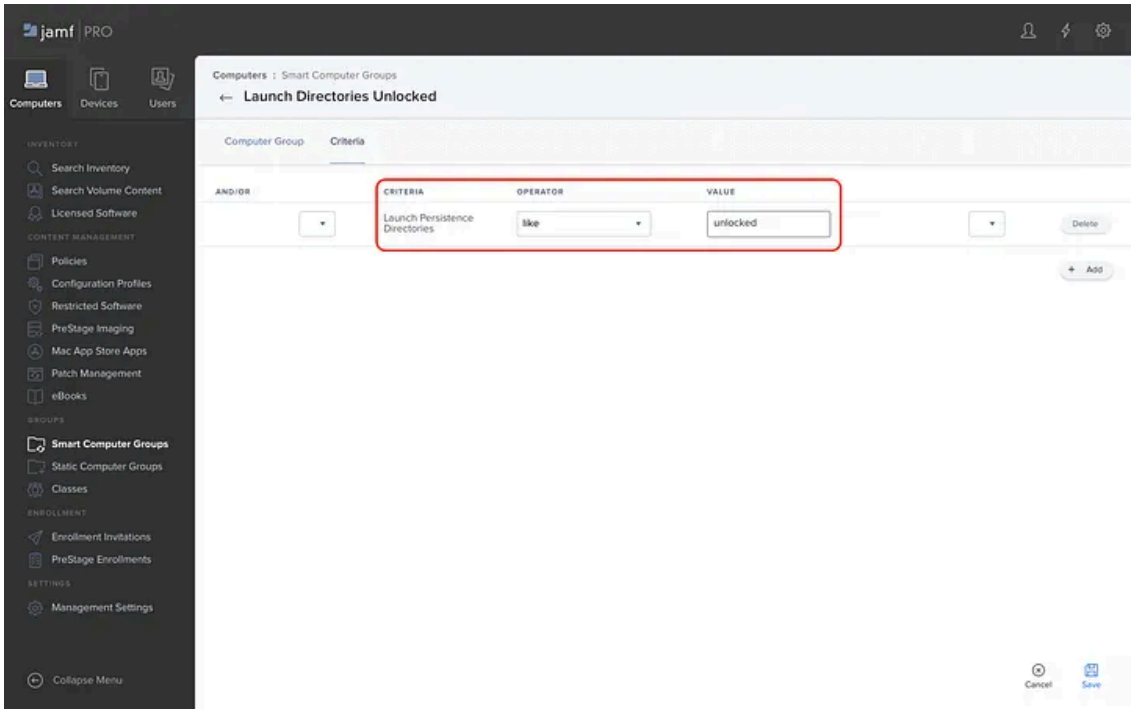
The image below is how the result of this Extension Attribute appears in the computer inventory record:

Press enter or click to view image in full size



Once we can report on the `uchg` status of the Launch directories, we can then use that attribute as the basis for a Smart Group to scope our policy. First, we give the Smart Group an identifying name. Then we define the criteria for this group as *any computer that contains the word unlocked in the Launch Persistence Directories inventory field.*

Press enter or click to view image in full size

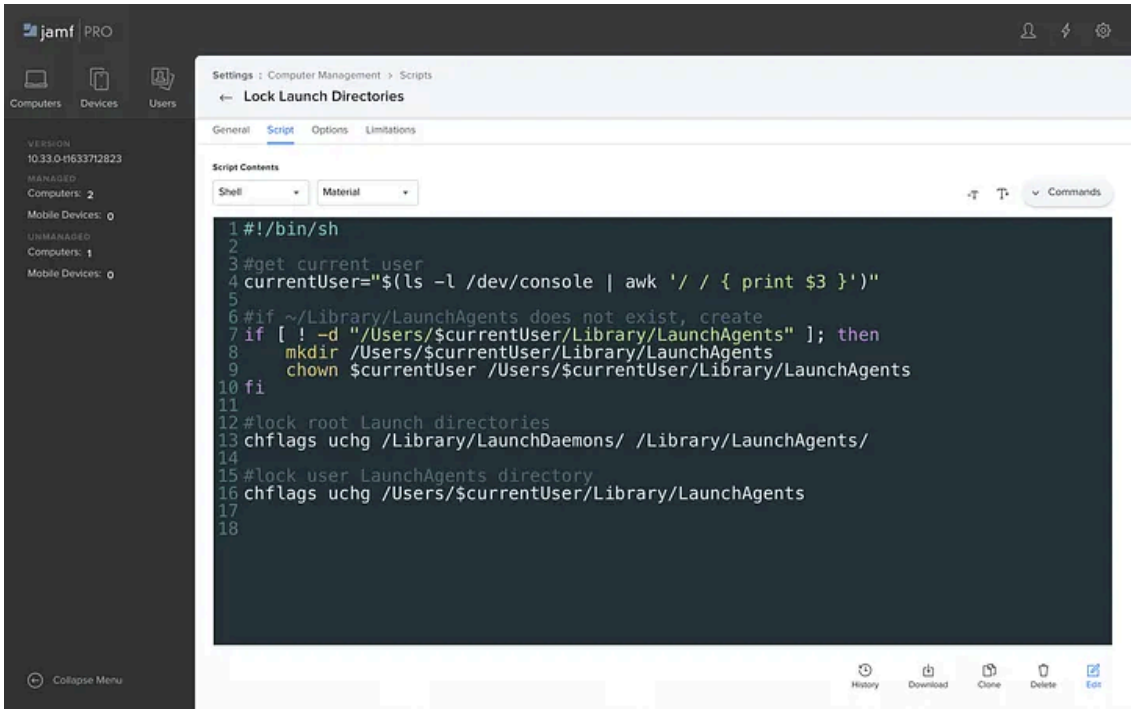


Writing the Script

Next, we need to write the script that locks the launch persistence directories. Since the `~/Library/LaunchAgents/` directory does not exist by default, in this script, we check for its existence and create it if not present. If the user is a standard user, Antonio recommends creating an unprivileged user account on each machine and changing ownership of the `~/Library/LaunchAgents` to this stand-in account. This avoids having the directory owned by root and prevents the local user from simply unlocking their directory as its owner.

Lastly, we use `chflags uchg` to lock each directory.

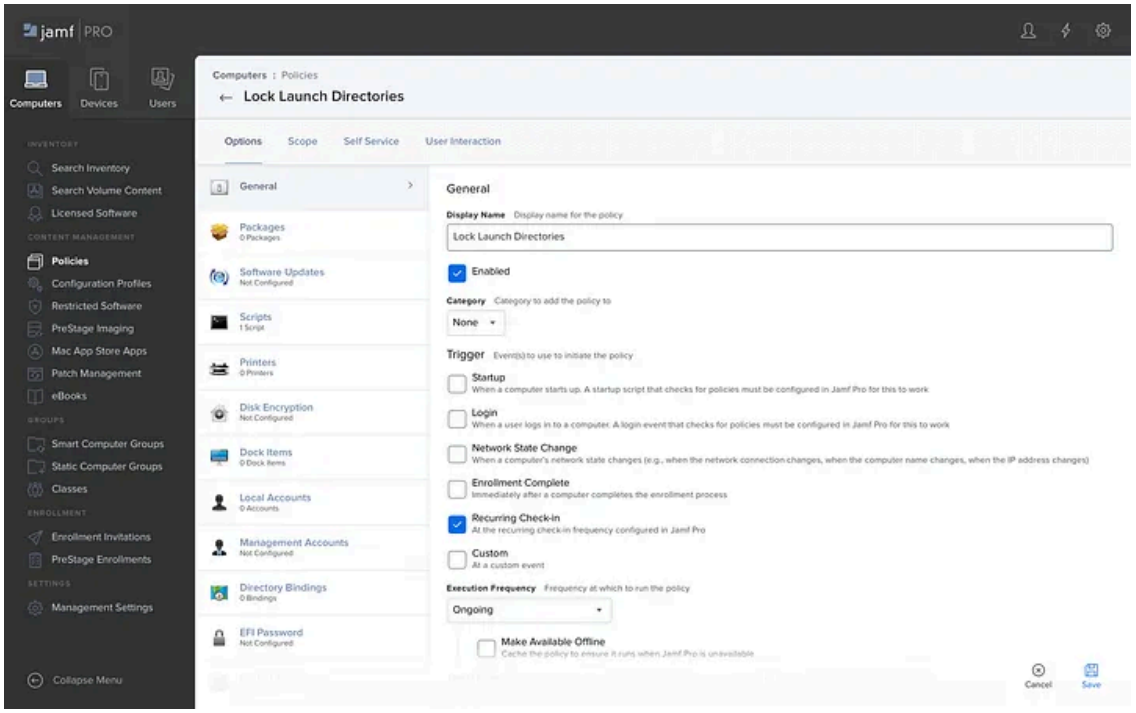
Press enter or click to view image in full size



Crafting the Policy

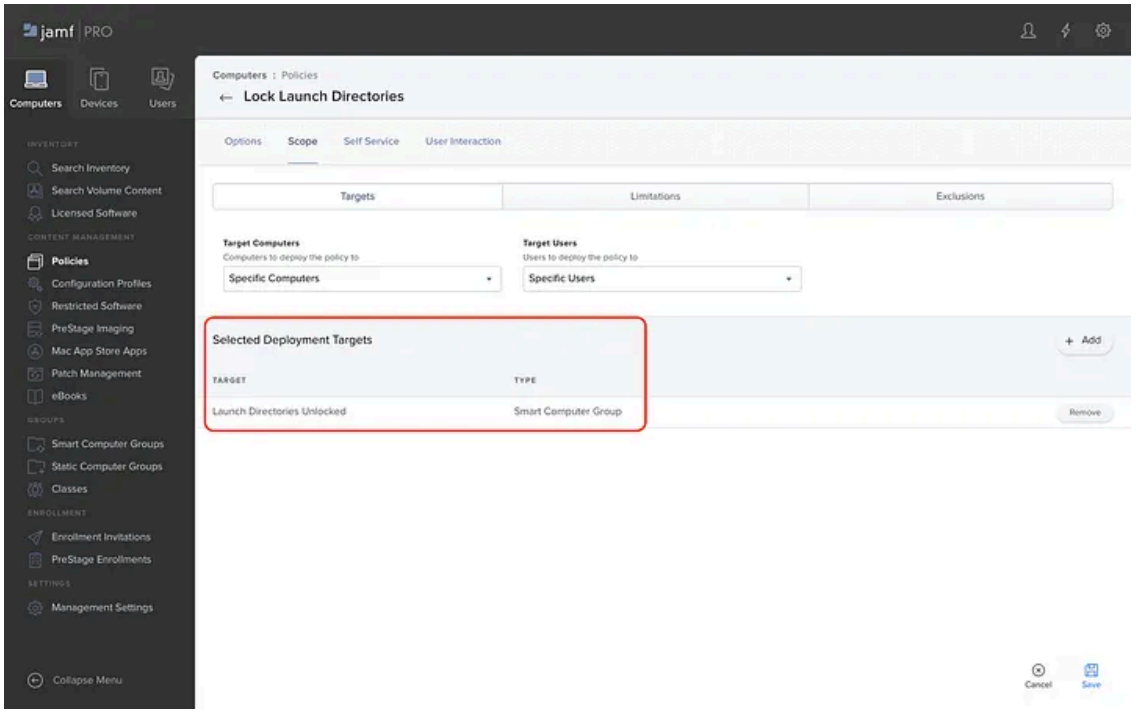
Finally, we create a policy that ties all of these elements together. After giving the policy an appropriate name, we set the policy to be triggered at the recurring check-in (every 15 minutes by default) on any computer that is in scope.

Press enter or click to view image in full size



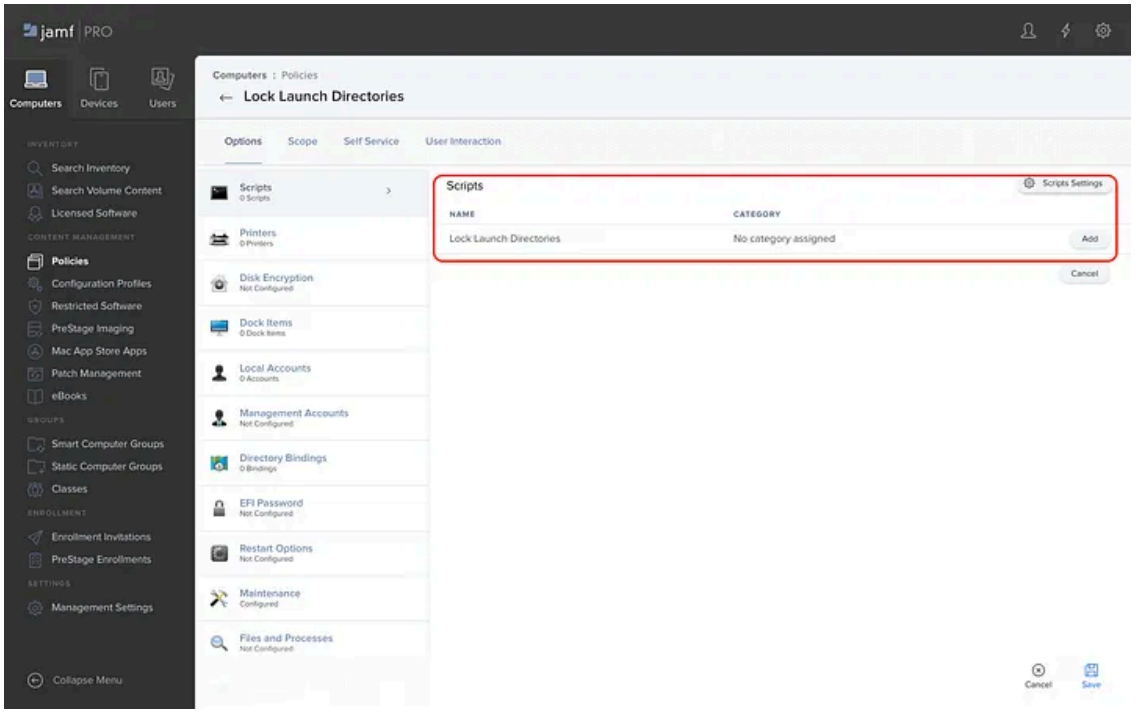
And for the scope, we set the policy to run against any computer that falls into the “Launch Directories Unlocked” Smart Group that we previously created.

Press enter or click to view image in full size



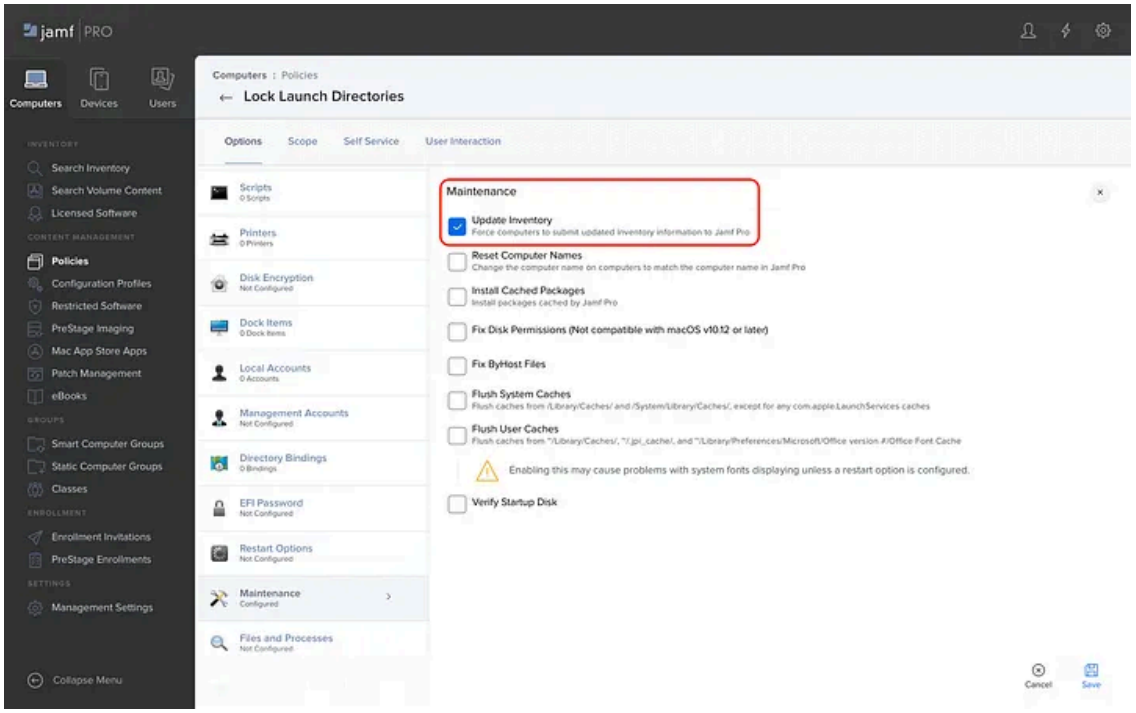
As for the Policy payloads, we will configure the Scripts payload and add our Lock Launch Directories script.

Press enter or click to view image in full size



We also add a secondary Maintenance payload to force an inventory update, to immediately reflect this change in the computer inventory record and remove the computer from the “Launch Directories Unlocked” Smart Computer Group.

Press enter or click to view image in full size



When the need arises to install a legitimate Launch Daemon or Launch Agent, an admin should run the chflags `nouchg` in a preinstall script to remove the lock and then re-enable the flag in a post-install script.

To Summarize

We can lock the root-level launch persistence directories using an MDM solution with scripting capabilities, like Jamf Pro. An administrator would only unlock it when installing approved software. They would then relock it after install.

The administrator can create a honey-user account for each managed macOS machine and change the owner of the user-level LaunchAgents directory to this honey-user. The admin can then lock the user-level directory.

I hope this mitigation strategy works for your enterprise environment. I'd love to hear from anyone who gives it a try. Contact me via my antman1p Twitter.

As always, thanks for reading, and stand by for more sauce!
