How Malware Persists on macOS

By Phil Stokes

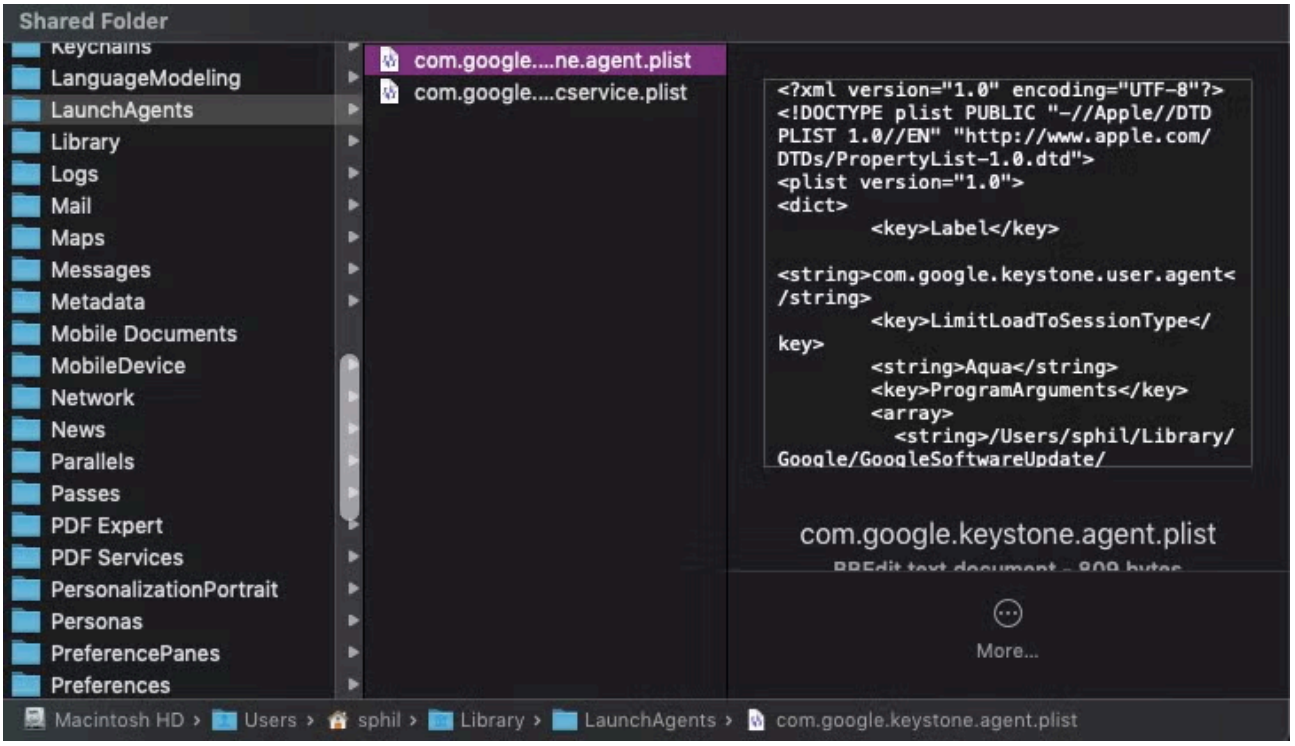
Published: 2019-06-17 · Archived: 2026-04-05 13:06:27 UTC

Whether it's a cryptominer looking for low-risk money-making opportunities, adware hijacking browser sessions to inject unwanted search results, or malware designed to spy on a user, steal data or traverse an enterprise network, there's one thing all threats have in common: the need for a persistent presence on the endpoint. On Apple's macOS platform, attackers have a number of different ways to persist from one login or reboot to another.

In this post, we review macOS malware persistence techniques seen in the wild as well as highlighting other persistence mechanisms attackers could use if defenders leave the door open. Has your IT team and security solution got them all covered? Let's take a look.

How To Persist Using a LaunchAgent

By far the most common way malware persists on macOS is via a LaunchAgent. Each user on a Mac can have a LaunchAgents folder in their own Library folder to specify code that should be run every time that user logs in. In addition, a LaunchAgents folder exists at the computer level which can run code for all users that log in. There is also a LaunchAgents folder reserved for the System's own use. However, since this folder is now managed by macOS itself (since 10.11), malware is locked out of this location by default so long as System Integrity Protection has not been disabled or bypassed.



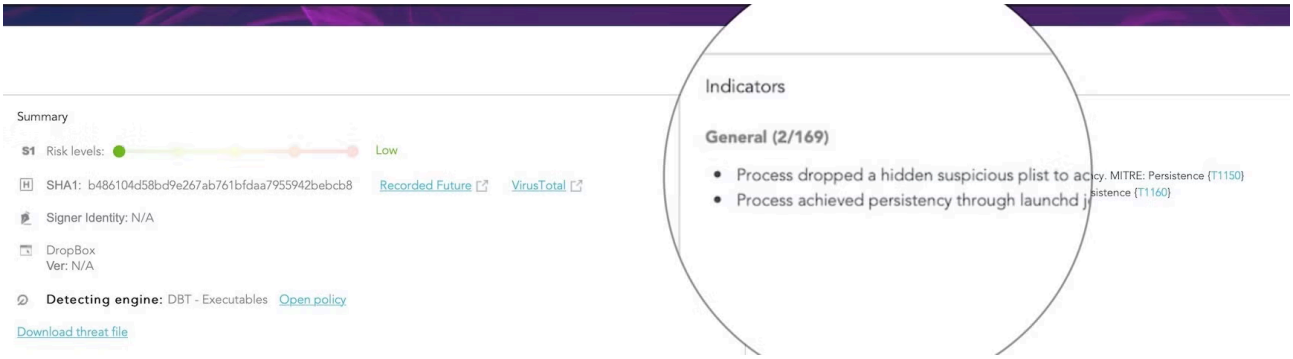
LaunchAgents take the form of property list files, which can either specify a file to execute or can contain their own commands to execute directly.

```
2018-09-06 13:04:32.089154+0700 stringDecoder[43969:12705281]
Decoded: <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>Label</key>
	<string>%@</string>
	<key>KeepAlive</key>
	<false/>
	<key>RunAtLoad</key>
	<true/>
	<key>StartInterval</key>
	<integer>%d</integer>
	<key>ExitTimeOut</key>
	<integer>0</integer>
	<key>ProgramArguments</key>
	<array>
	%@
	</array>
</dict>
</plist>
```

Since user LaunchAgents require no privileges to install, these are by far the easiest and most common form of persistence seen in the wild. Unfortunately, Apple took the controversial step of hiding the parent Library folder from users by default all the way back in OSX 10.7 Lion, making it easier for threat actors to hide these agents from unsavvy users.

Users can unhide this library in a couple of different ways for manual checks, but enterprise security solutions should monitor the contents of this folder and block or alert on malicious processes that write to this location, as shown here in this example from the SentinelOne console. The threat is autonomously blocked and the IT team is

alerted to the IOCs, with reference to Mitre Att&ck framework, and convenient links to RecordedFuture and VirusTotal detections.



Persistence By LaunchDaemon

LaunchDaemons only exist at the computer and system level, and technically are reserved for persistent code that does not interact with the user – perfect for malware. The bar is raised for attackers as writing a daemon to `/Library/LaunchDaemons` requires administrator level privileges. However, since most Mac users are also admin users and habitually provide authorisation for software to install components whenever asked, the bar is not all that high and is regularly cleared by infections we see in the wild. In this image, the computer has been infected by 3 separate, malicious LaunchDaemons.

```
/Library/LaunchDaemons:

        com.adobe.agsservice.plist
                --> Program Arguments: /Library/Application Support/Adobe/AdobeGCClient/AGSService

        com.IvCL8.plist
                -> Program: /Library/SIBaJ/WQ9EI

        com.KreberisecDaemon.plist
                --> Program Arguments: /Library/Application Support/com.KreberisecDaemon/Kreberisec
                --> Program Arguments: r

        com.apple.installer.osmessagetracing.plist
                --> Program Arguments: /System/Library/PrivateFrameworks/OSInstaller.framework/Resources/OSMessageTracer

        com.KreberisecP.plist
                --> Program Arguments: /var/root/.Kreberisec/KreberisecDaemon

        com.adobe.acc.installer.v2.plist
                -> Program: /Library/PrivilegedHelperTools/com.adobe.acc.installer.v2
                --> Program Arguments: /Library/PrivilegedHelperTools/com.adobe.acc.installer.v2

        com.adobe.fpsaud.plist
                --> Program Arguments: /Library/Application Support/Adobe/Flash Player Install Manager/fpsaud

```

Because LaunchDaemons run on startup and for every user even before a user logs in, it is essential that your security software is aware of what daemons are running and when any new daemons are written. As with System LaunchAgents, the System LaunchDaemons are protected by SIP so the primary location to monitor is `/Library/LaunchDaemons`.

Don’t just assume labels you recognize are benign either. Some legitimate LaunchDaemons point to unsigned code that could itself be replaced by something malicious. For example, the popular networking program Wireshark uses a LaunchDaemon,

`/Library/LaunchDaemons/org.wireshark.ChmodBPF.plist`

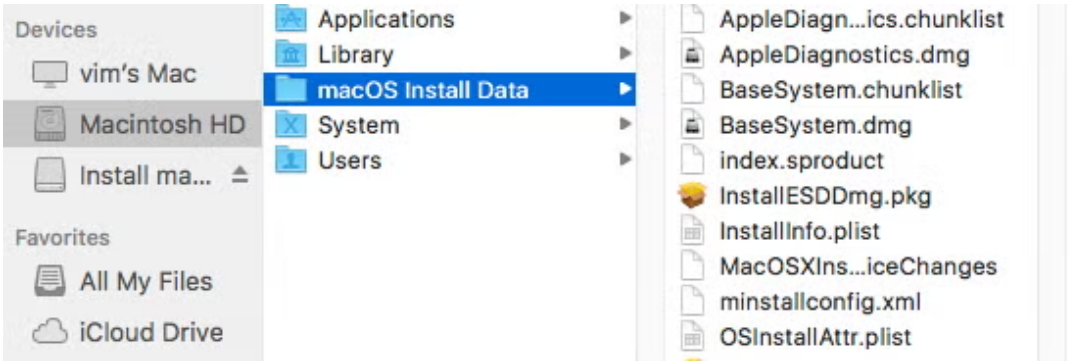
that executes unsigned code at the path:

`/Library/Application Support/Wireshark/ChmodBPF/ChmodBPF`

Even Apple itself uses a LaunchDaemon that isn't always cleaned up immediately such as

`/Library/LaunchDaemons/com.apple.installer.cleanupinstaller.plist`

This points to an executable in the `/macOS Install Data` folder that could be replaced by malicious code.



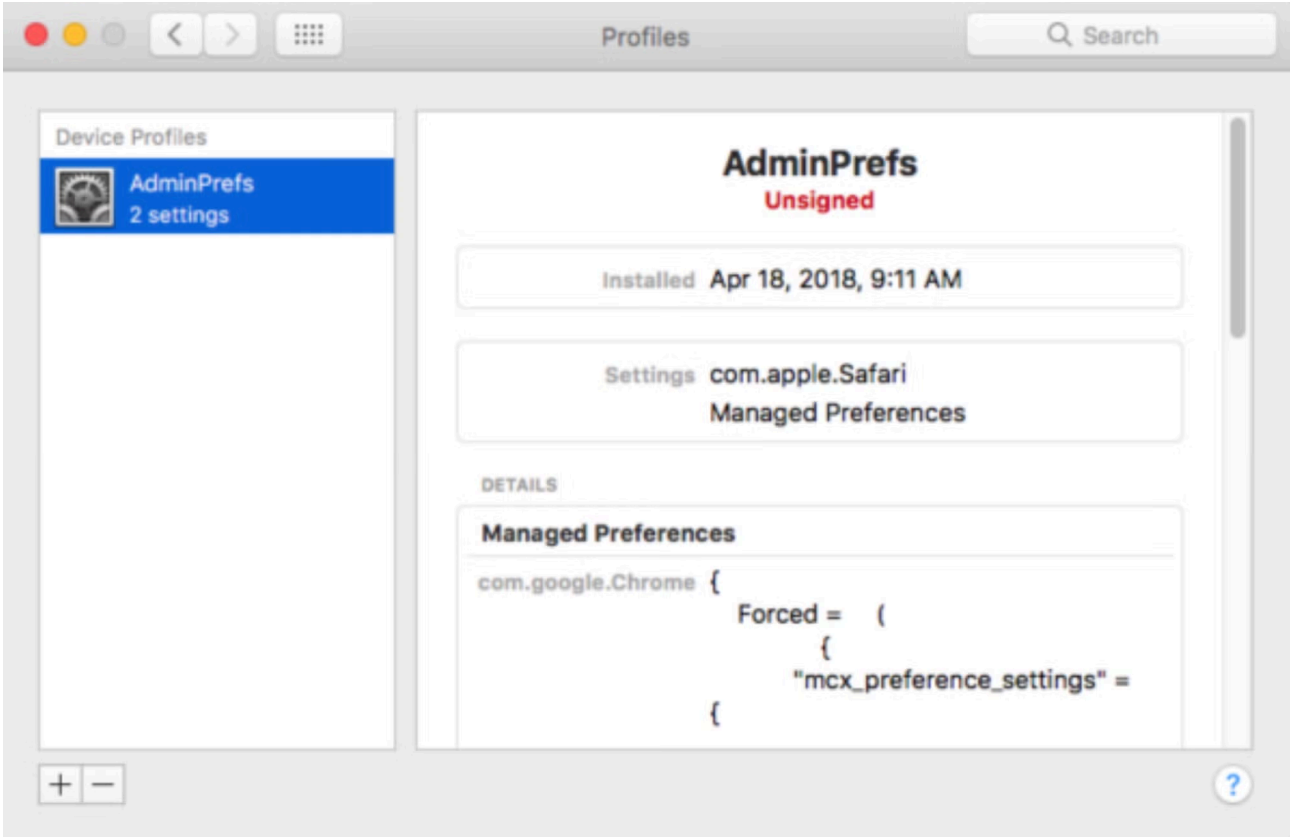
Remember that with privileges, an attacker can either modify the program arguments of these property plists or the executables that they point to in order to achieve stealthy persistence. Since these programs will run with root privileges, it's important that you or your security solution isn't just blanket whitelisting code because it looks like it comes from a legitimate vendor.

Persistence with Profiles

Profiles are intended for organizational use to allow IT admins to manage machines for their users, but their potential for misuse has already been spotted by malware authors. As profiles can be distributed via email or a website, tricking users into inadvertently installing them is just another element of social engineering.

```
NAME
     profiles -- Profiles Tool

SYNOPSIS
     profiles [[-I | -R | -i] [-F file_path_to_profile | -]] [[-L]
              [-U username]] [[-r] [-p profile_id] [-u uuid]
              [-o output_file_path] [-Y shortname]] [-PHDdCchfvxVzYeN]

DESCRIPTION
     profiles allows you to install, remove or list configuration profiles, or
     to install provisioning profiles.  Some commands may only work with ele-
     vated privileges, or for the current user.

     -I  Install a configuration profile for a particular user from a profile
         file.

     -i  Install a provisioning profile from a profile file.

     -V  Verify a provisioning profile from a profile file.

     -R  Remove a configuration profile for a particular user from a profile
```

Configuration profiles can force a user to use certain browser settings, DNS proxy settings, or VPN settings. Many other payloads are possible which make them ripe for abuse.



Profiles can be viewed by users in System Preferences Profiles pane and by administrators by enumerating the `/Library/Managed Preferences` folder. Be aware that neither the pane nor folder will be present on a system where profiles have never been installed.

```
Profiles:
_computerlevel[1] attribute: name: AdminPrefs
_computerlevel[1] attribute: configurationDescription:
_computerlevel[1] attribute: installationDate: 2019-02-11 01:07:00 +0000
_computerlevel[1] attribute: organization:
_computerlevel[1] attribute: profileIdentifier: com.myshopcoupon.safari
_computerlevel[1] attribute: profileUUID: b1e288a3-28e5-46c6-896b-102e5958a337
_computerlevel[1] attribute: profileType: Configuration
_computerlevel[1] attribute: removalDisallowed: TRUE
_computerlevel[1] attribute: version: 1
_computerlevel[1] attribute: containsComputerItems: TRUE
_computerlevel[1] attribute: internaldata: TRUE
_computerlevel[1] payload count = 1
_computerlevel[1]           payload[1] name                  = (null)
_computerlevel[1]           payload[1] description           = (null)
_computerlevel[1]           payload[1] type                  = com.apple.Safari
_computerlevel[1]           payload[1] organization          = (null)
_computerlevel[1]           payload[1] identifier            = com.myshopcoupon.safari.AdminPrefs.9f0375b6-53ad-47d8-af34-94e1e4b45f33
_computerlevel[1]           payload[1] uuid                  = 828236d5-112b-4b99-a1ab-ee040ab12a3f
_computerlevel[2] attribute: name: AdminPrefs
_computerlevel[2] attribute: configurationDescription:
_computerlevel[2] attribute: installationDate: 2019-02-11 01:07:00 +0000
_computerlevel[2] attribute: organization:
_computerlevel[2] attribute: profileIdentifier: com.myshopcoupon.chrome
_computerlevel[2] attribute: profileUUID: cfc06391-6f38-41c4-9a2f-0b9b2ce32c63
_computerlevel[2] attribute: profileType: Configuration
_computerlevel[2] attribute: removalDisallowed: TRUE
_computerlevel[2] attribute: version: 1
_computerlevel[2] attribute: containsComputerItems: TRUE
_computerlevel[2] attribute: internaldata: TRUE
_computerlevel[2] payload count = 1
_computerlevel[2]           payload[1] name                  = (null)
_computerlevel[2]           payload[1] description           = (null)
_computerlevel[2]           payload[1] type                  = com.apple.ManagedClient.preferences
_computerlevel[2]           payload[1] organization          = (null)
_computerlevel[2]           payload[1] identifier            = com.myshopcoupon.chrome.ChromeAdmin.90f31854-3a27-4ef5-b3df-847c5b40b128
_computerlevel[2]           payload[1] uuid                  = c5302b59-3160-4bd7-b0c8-ad29ed5d5011
There are 2 configuration profiles installed

```

Cron Still Persists on macOS

The venerable old `cron` job has not been overlooked by malware authors. Although Apple has announced that new `cron` jobs will require user interaction to install in 10.15 Catalina, it's unlikely that this will do much to hinder attackers using it as a persistence method. As we've noted before, user prompts are not an effective security measure when the user has already been tricked into installing the malicious software under the guise of something else. There's overwhelming evidence to suggest that users escape 'death by dialog' by simply clicking everything without paying attention to what the dialog alert actually says.

Malicious `cron` jobs are used by AdLoad and Mughthesec malware, among others, to achieve persistence.

```
User Crontab:

        36 */2 * * * /Users/User1/Library/Application\ Support/CBE60FF0-C978-4DA3-BCF2-415305E55B89/4A5672E2-E47F-4F68-BB0D-652AE594E47E h >/dev/null 2>&1
41 * * * * /Users/User1/Library/unfinical.jt/unfinical.jt cr
~
~
```

Kexts for Persistence

Kernel extensions are widely used by legitimate software for persistent behavior, and we've seen them also used by so-called PUP software like MacKeeper. An open-source keylogger, logkext, has also been around for some years, but in general kexts are not a favoured trick among malware authors as they are comparatively difficult to create, lack stealth, and can be easily removed.

How to Find Persistent Login Items

Changes made by Apple to Login Items have, on the other hand, resulted in more attractive opportunities for malware persistence. Once upon a time, Login Items were easily enumerated through the System Preferences utility, but a newer mechanism makes it possible for any installed application to launch itself at login time simply by including a Login Item in its own bundle. While the intention of this mechanism is for legitimate developers to offer control of the login item through the app's user interface, unscrupulous developers of commodity adware and PUP software have been abusing this as a persistence trick as it's very difficult for users to reliably enumerate which applications actually contain a bundled login item.

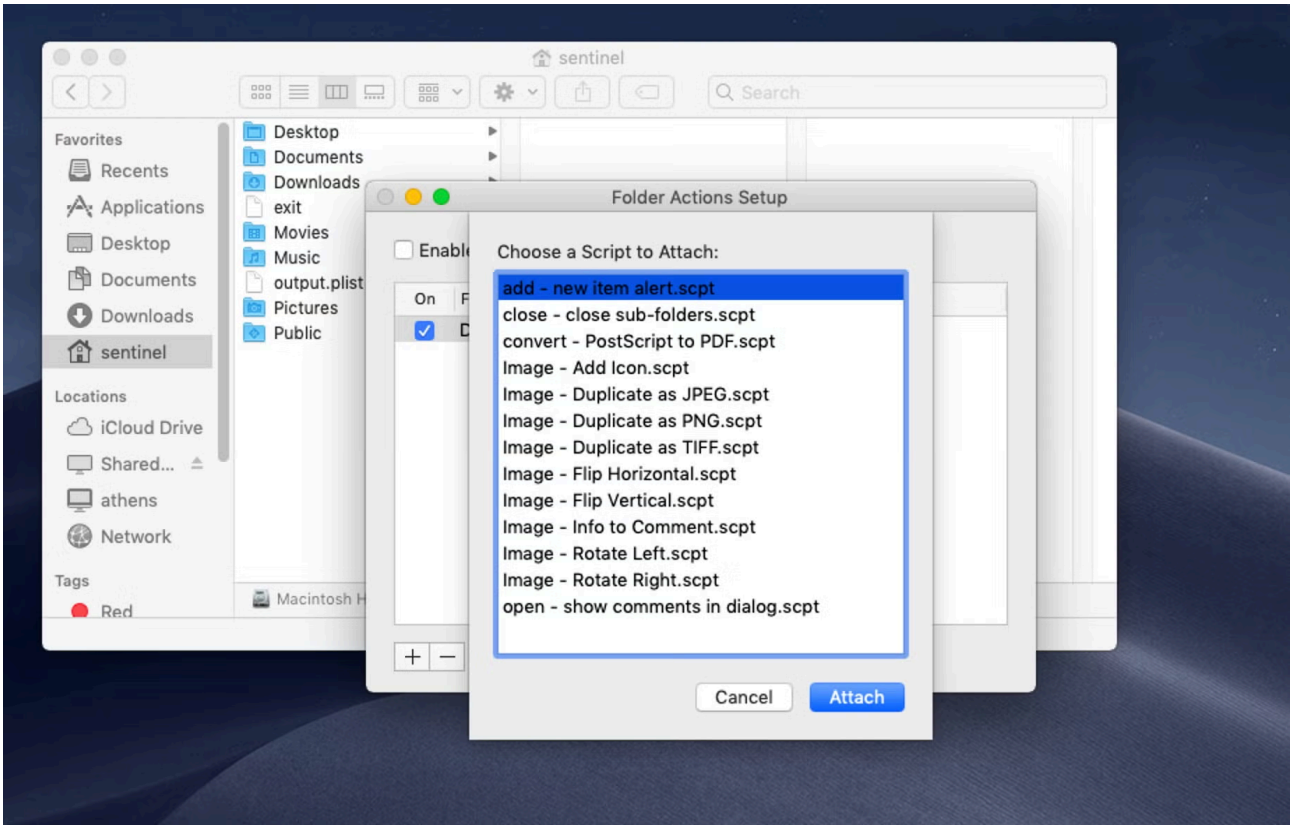
While it's not a simple matter for users to enumerate all the Login Items, admins can do so with a little extra work by parsing the following file, if it exists:

```
~/Library/Application Support/com.apple.backgroundtaskmanagementagent/backgrounditems.btm
```

A method of doing so was first written up by security researcher Patrick Wardle, but that still requires some programming skill to implement. A more user-friendly AppleScript version that can be cut and pasted into the macOS Script Editor utility and run more conveniently is available here.

AppleScript & Friends

While on the subject of AppleScript, Apple's most useful "swiss army knife" tool somewhat unsurprisingly also has some persistence mechanisms to offer. The first leverages Folder Actions and allows an attacker to execute code that could even be read into memory remotely every time a particular folder is written to. This remarkably clever way of enabling a fileless malware attack by re-purposing an old macOS convenience-tool was first written up by Cody Thomas.



Admins with security solutions that do not have behavioral AI detection should monitor processes executing with `osascript` and `ScriptMonitor` in the command arguments to watch out for this kind of threat.

An even more wily trick leverages Mail rules, either local or iCloud-based, to achieve persistence by triggering code after sending the victim an email with a specially-crafted subject line. This method is particularly stealthy and will evade many detection tools.

Defenders can manually check for the presence of suspicious Mail rules by parsing the `ubiquitous_SyncedRules.plist` file and the `SyncedRules.plist` file for iCloud and local Mail rules, respectively. A quick bash script such as

```
grep -A1 "AppleScript" ~/Library/Mail/V6/MailData/SyncedRules.plist
```

will enumerate any Mail rules that are calling AppleScripts. If any are found, those will then need to be examined closely to ensure they are not malicious.

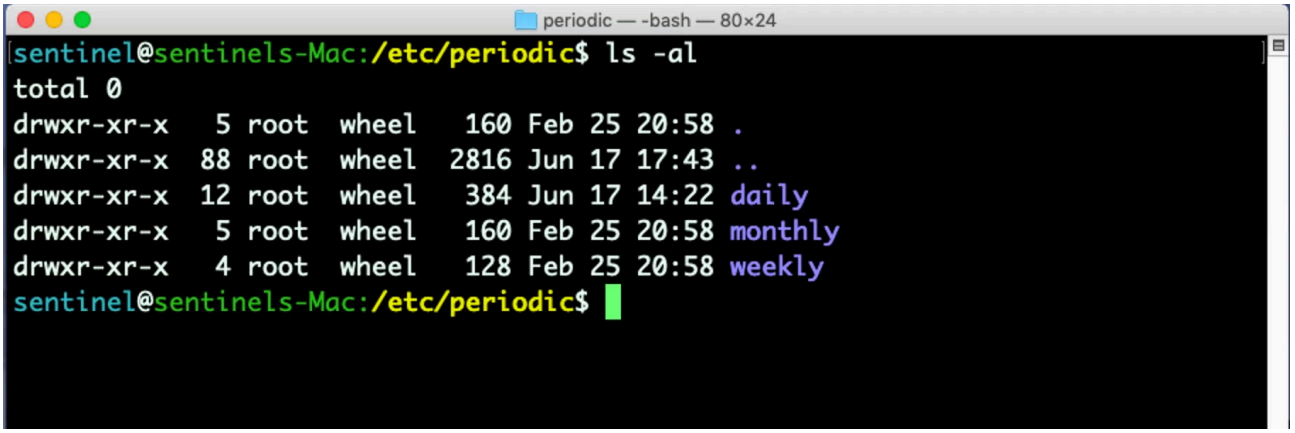
Also Ran: Forgotten Persistence Tricks

For those who remember them, `rc.common` and `launchd.conf` no longer work on macOS, and support for StartupItems also appears to have been removed after 10.9 Mavericks.

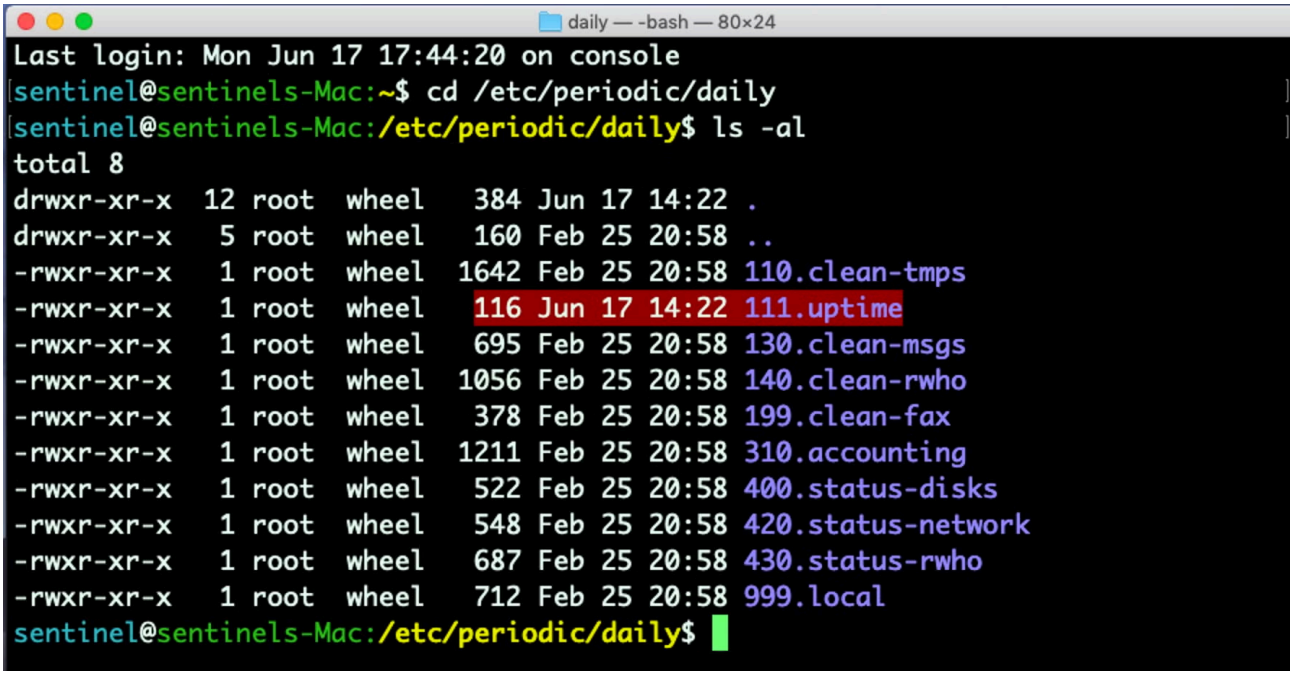
Even so, other old “nix tricks” do still work, and while we’ve yet to see any of the following persistence mechanisms used in the wild, they are worth keeping an eye on. These tricks include using `periodics`, `loginhooks`, `at` jobs, and the `emond` service.

Periodics As a Means of Persistence

Periodics are system scripts that are generally used for maintenance and run on a daily, weekly and monthly schedule. Periodics live in similarly titled subfolders within `etc/periodic` folder.



Listing the contents of each of the subfolders should reveal the standard set of periodics, unless your admins are using their own custom periodic scripts. If not, anything additional found in there should be treated as suspicious and inspected. Notice the unusual “uptime” script here, which will run on a daily basis without user interaction or notification.



Also, be sure to check both `/etc/defaults/periodic.conf` and `/etc/periodic.conf` for system and local overrides to the default `periodic` configuration.

LoginHooks and LogoutHooks

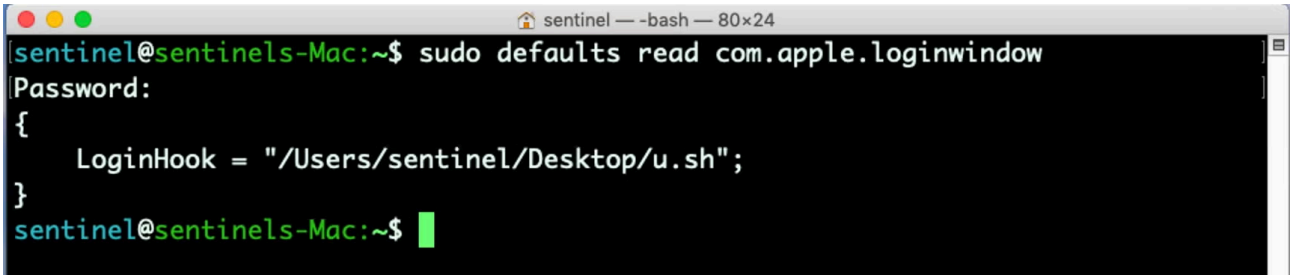
LoginHooks and LogoutHooks have been around for years and are rarely used these days, but are still a perfectly viable way of running a persistence script on macOS Mojave. As the names suggest, these mechanisms run code

when the user either logs in or logs out.

It's a simple matter to write these hooks, but fortunately it's also quite easy to check for their existence. The following command should return a result that doesn't have either LoginHook or LogoutHook values:

```
sudo defaults read com.apple.loginwindow
```

If, on the other hand, it reveals a command or path to a script, then consider those worthy of investigation.



At Jobs: Run Once, Persist Forever

A much less well-known mechanism is `at` jobs. While these only run once and are not enabled by default, they are a sneaky way to run some code on restart. The single-use isn't really a problem, since the `at` job can simply be re-written each time the persistence mechanism fires, and these jobs are very unlikely to be noticed by most users or indeed many less-experienced admins.

You can check whether any `at` jobs are scheduled by enumerating the `/var/at/jobs` directory. Jobs are prefixed with the letter `a` and have a hex-style name.

Emond – The Forgotten Event Monitor

Sometime around OSX 10.5 Leopard, Apple introduced a logging mechanism called `emond`. It appears `emond` was never fully developed and development may have been abandoned by Apple for other mechanisms, but it remains available even on macOS 10.14 Mojave.

In 2016, James Reynolds provided the most comprehensive analysis to-date of `emond` and its capabilities. Reynolds was not interested in `emond` from a security angle, but rather was documenting a little-known daemon

from the angle of an admin wanting to implement their own log scanner. Reynolds concludes his analysis with an interesting comment, though:

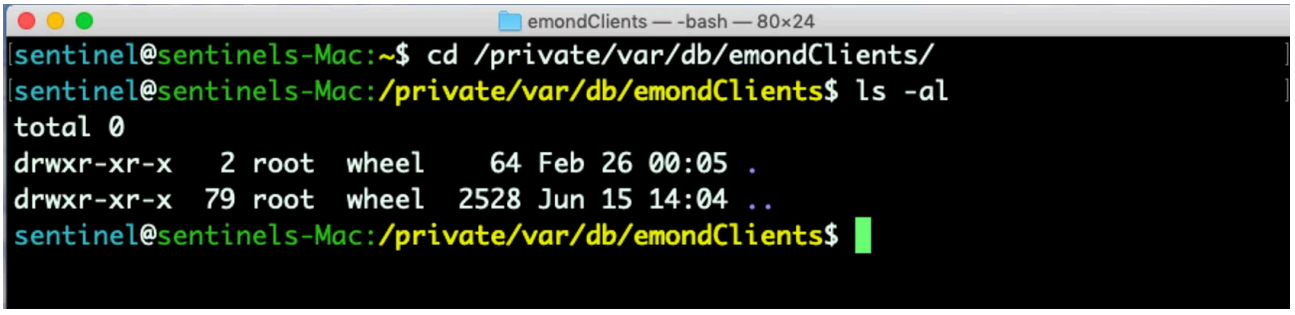
> Considering how easy it is to log, run a command, or send an email in a perl script, I can't see why I'd want to use emond instead of a script.

This little-known service may not be much use to a Mac admin, but to a threat actor one very good reason would be to use it as a persistence mechanism that most macOS admins probably wouldn't know to look for.

Detecting malicious use of `emond` shouldn't be difficult, as the System LaunchDaemon for the service looks for scripts to run in only one place:

`/private/var/db/emondClients`

Admins can easily check to see if a threat actor has placed anything in that location.



As `emond` is almost certainly not used in your environment for any legitimate reason, anything found in the `emondClient` directory should be treated as suspicious.

Conclusion

As the above mechanisms show, there are plenty of ways for attackers to persist on macOS. While some of the older ways are now defunct, the onus is still very much on defenders to keep an eye on the many possible avenues that code execution can survive a reboot. To lessen that burden, it's recommended that you move to a next-gen behavioral solution that can autonomously detect and block malware from achieving persistence. If you are not already protected by SentinelOne's activeEDR solution, contact us for a free demo and see how it can work for you in practice.

Source: https://www.sentinelone.com/blog/how-malware-persists-on-macos/