Objective-See's Blog

objective-see.com/blog/blog_0x69.html

OSX.CDDS (OSX.MacMa)

a sophisticated watering hole campaign drops a new macOS implant!

by: Patrick Wardle / November 11, 2021

📝 👾 Want to play along?
I've uploaded an OSX.CDDS sample (password: infect3d).

...please don't infect yourself!

🙋‍♂️ Updates:

- Anti-Virus engines are now detecting this threat.
- As Google referred to the attack as "MacMa", some refer to this malware as OSX.MacMa.
- Trend Micro Researchers such as @phretor and @evanslify noted that (contrary to initial reporting), the malware does not in fact leverage Data Distribution Service (DDS).
- See also "Google Caught Hackers Using a Mac Zero-Day Against Hong Kong Users".

Background

Today, Google's Threat Analysis Group (TAG), published an intriguing report titled, "Analyzing a watering hole campaign using macOS exploits."

In this report, they detailed a highly targeted attack that leveraged both iOS and macOS exploits in order to remotely infect Apple users. As they were not able to recover the full iOS exploit chain, the write-up focused almost fully on the macOS version of the attack which leveraged:

- A webkit "n-day" RCE (patched as CVE-2021-1789 in January)
- An XNU 0-day local privilege escalation (now patched as CVE-2021-30869). According to Google this exploit was, "*was presented by Pangu Lab in a public talk at zer0con21 in April 2021 and Mobile Security Conference (MOSEC) in July 2021*"

While Google's blog provided a thorough overview of the macOS exploit chain, it did not dig too much into the persistent macOS implant that would be installed upon successfully exploited systems. However they did note:

"*Notable features for this backdoor include*:

- *victim device fingerprinting*
- *screen capture*
- *file download/upload*
- *executing terminal commands*
- *audio recording*
- *keylogging*"

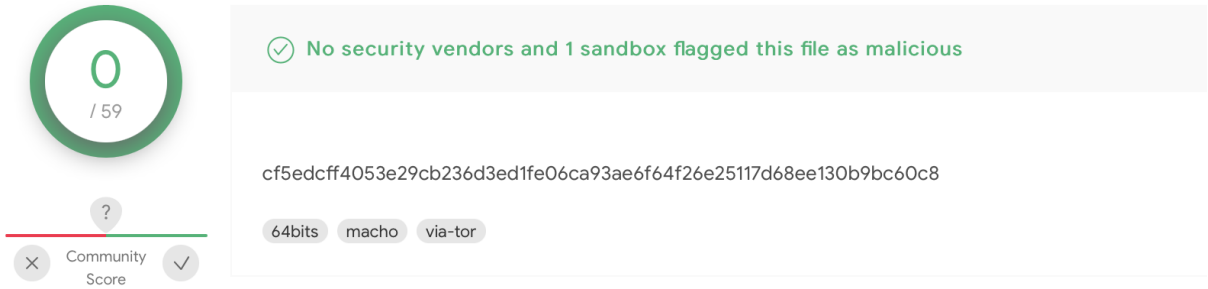
Moreover, they were kind enough to provided hashes of the implant 🤗 And, as these has also been uploaded to VirusTotal, we can grab them for analysis.

In this blog post, we'll briefly analyze this implant, `OSX.CDDS` …an implant that currently remains undetected by all of the anti-virus engines on VirusTotal.

Triage

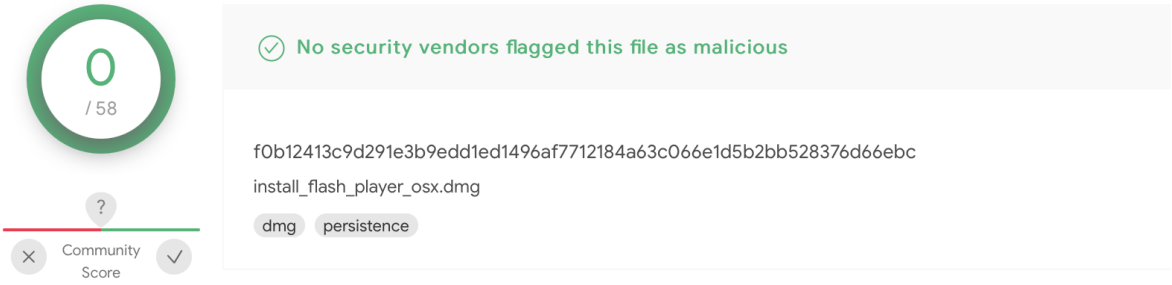
Google's report provided two hashes for `OSX.CDDS` that would be remotely installed on exploited systems:

- `cf5edcff4053e29cb236d3ed1fe06ca93ae6f64f26e25117d68ee130b9bc60c8`



cf5e... on VirusTotal

- `f0b12413c9d291e3b9edd1ed1496af7712184a63c066e1d5b2bb528376d66ebc`



f0b1... on VirusTotal

The first item, `cf5e...`, they noted was the 2021 version of the implant, while the second item, `f0b1...`, was from 2019. Note that both are currently undetected on VirusTotal.

The 2019 sample, is a disk image named `install_flash_player_osx.dmg`. If we mount it, we find application named `SafariFlashActivity`:

Contents of install_flash_player_osx.dmg

If we examine its code-signing information, we see its been signed adhoc:

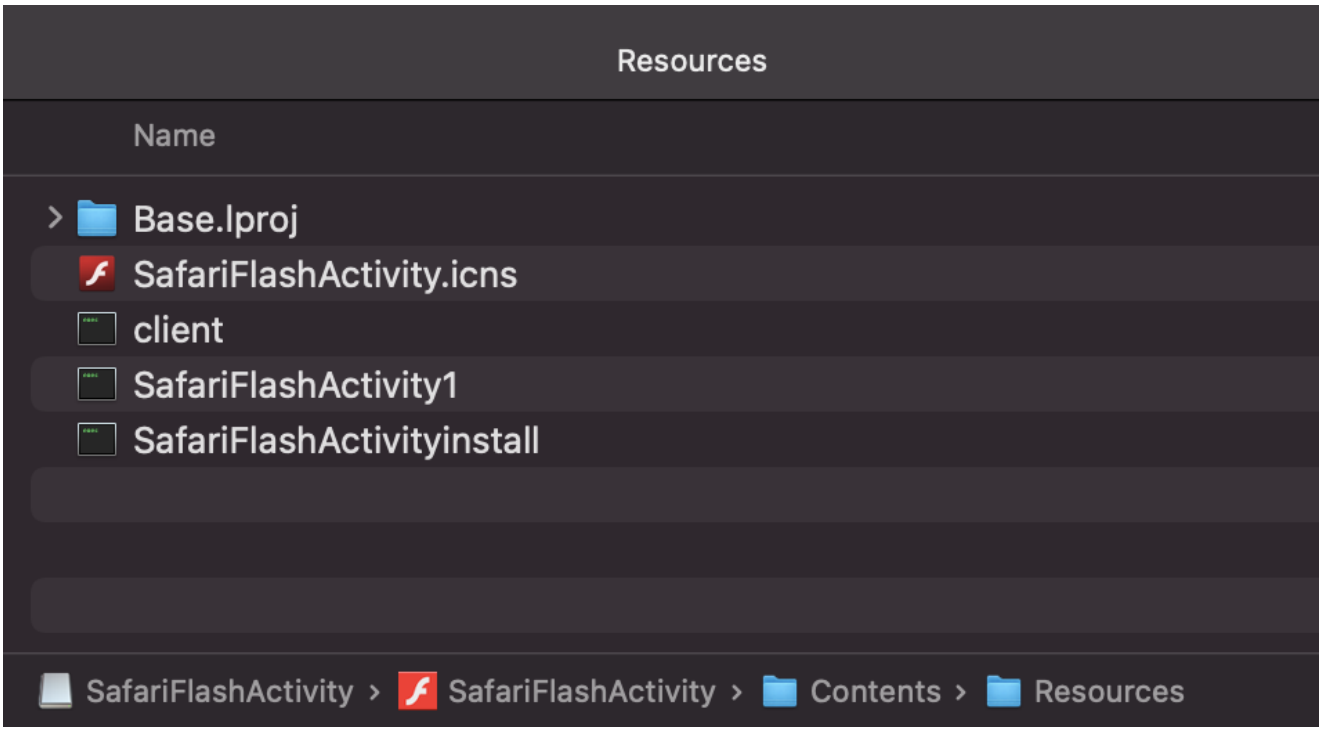
```
% codesign -dvv /Volumes/SafariFlashActivity/SafariFlashActivity.app
Executable=/Volumes/SafariFlashActivity/
           SafariFlashActivity.app/Contents/MacOS/SafariFlashActivity

Identifier=xxxxxx.preexcl-project
Format=app bundle with Mach-O thin (x86_64)
CodeDirectory v=20100 size=615 flags=0x2(adhoc) hashes=14+3 location=embedded
Signature=adhoc
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=0 size=12
```

Taking a peek at it's `Info.plist` files reveals key-value pairs such as:

- `LSMinimumSystemVersion` : `10.7`
- `CFBundleExecutable` : `SafariFlashActivity`
- `CFBundleIdentifier` : `xxxxx.SafariFlashActivity`
- `NSHumanReadableCopyright` : `Copyright © 2018年 xxxxx. All rights reserved.`
  (Note the Chinese character `年` translate to "year").

As Google's report noted that this was an older (2019) sample of the implant, we won't dig into it too much more, instead we'll focus on the 2021 sample.

Still let's note a few facts about the older sample. First, it contains various executable components in it `/Resources` section:



Additional executable components

The last item, `/Resources/SafariFlashActivityinstall` is a simple bash script:

```
% cat Resources/SafariFlashActivityinstall
#!/bin/bash
user=$(whoami)
dname=`dirname "$0"`
cd "$dname"
./client "$user"

home_dir="$(echo ~)"
launch_dir="$home_dir/Library/LaunchAgents"
launchctl unload "$launch_dir/com.UserAgent.va.plist"
launchctl load "$launch_dir/com.UserAgent.va.plist"
```

Its main goal is to execute the `client` binary (with the name of the currently logged in user), and the unload, then load a launch agent named `com.UserAgent.va.plist`.

We can observe the invocation of the `SafariFlashActivityinstall` script via a process monitor:

```
# ProcessMonitor.app/Contents/MacOS/ProcessMonitor -pretty
...
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
    "name" : "bash",
    "path" : "/bin/bash",
    "arguments" : [
      "sh",
      "-c",
      "\"/Volumes/SafariFlashActivity/SafariFlashActivity.app
          /Contents/Resources/SafariFlashActivityinstall\""
    ]
  }
}
```

…as well as the launch of the `client` binary:

```
# ProcessMonitor.app/Contents/MacOS/ProcessMonitor -pretty
...
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
    "name" : "client",
    "path" : "/Volumes/SafariFlashActivity/SafariFlashActivity.app
               /Contents/Resources/client",
    "arguments" : [
      "./client",
      "user"
    ]
  }
}
```

The malware also creates a directory named `Tools` in the `~/Library/Preferences/` directory, and saves several (embedded) custom tools into this directory, including:

- `arch` (SHA-1 `c4511ad16564eabb2c179d2e36f3f1e59a3f1346`)
  This binary invokes a function, aptly named `captureScreen`, to perform a screen capture, via Apple's Core Graphic APIs (e.g. `CGWindowListCreateImageFromArray`). It appears to then save it out to user-specified file.

- `at` (SHA-1 `77a86a6b26a6d0f15f0cb40df62c88249ba80773`)
  This binary performs a simple survey, then writes it out to `stdout`. For example, when run in a virtual machine, it produces the following:

```
% ./at
uuid=564D028C-69EF-7793-5BD9-8CC893CB8C8D
userName=user
version=Version 10.15.6 (Build 19G2021)
equipmentType=VMware7,1
mac=00:0c:29:cb:8c:8d
ip=
diskFreeSpace=11251048448/42605699072
availableMemory=2098040832/2147483648
cpu_info=Intel(R) Core(TM) i7-8559U CPU @ 2.70GHz
```

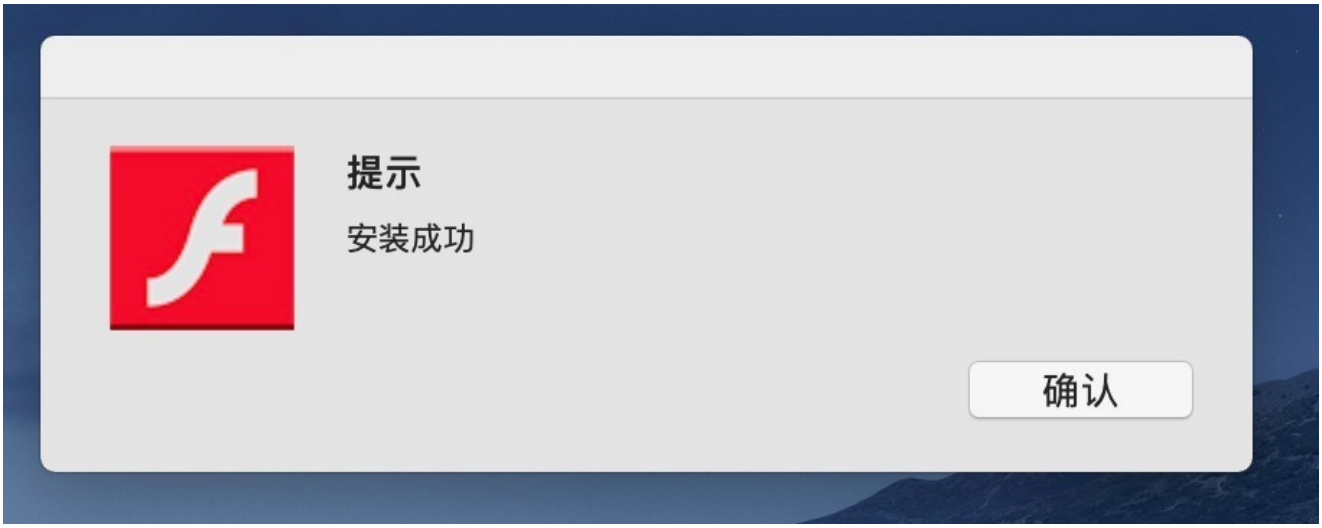
Back to the `client` binary, it then installs a persistent implant as a Launch Agent:

```
 1<?xml version="1.0" encoding="UTF-8"?>
 2<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" ...>
 3<plist version="1.0">
 4<dict>
 5  <key>Label</key>
 6  <string>com.UserAgent.va</string>
 7  <key>LimitLoadToSessionType</key>
 8  <string>Aqua</string>
 9  <key>ProgramArguments</key>
10  <array>
11      <string>/Users/user/Library/Preferences/UserAgent/lib/UserAgent</string>
12      <string>-runMode</string>
13      <string>ifneeded</string>
14  </array>
15  <key>RunAtLoad</key>
16  <true/>
17  <key>StartInterval</key>
18  <integer>600</integer>
19  <key>ThrottleInterval</key>
20  <integer>2</integer>
21  <key>WorkingDirectory</key>
22  <string>/Users/user/Library/Preferences/UserAgent/lib</string>
23</dict>
24</plist>
```

As the `RunAtLoad` key is set to `true` the specified binary, `/Users/user/Library/Preferences/UserAgent/lib/UserAgent` will be persistently executed by macOS each time the user logs in.

The UserAgent binary was originally submitted to VirusTotal on `2019-12-01`.
…and if we analyze the submission meta-data, we can see it was originally submitted to VirusTotal by a user, via one of my Objective-See tools (which integrate with VirusTotal)! How freaking cool!? 🤩

Finally, let's note that if (when) the `SafariFlashActivity` application is executed, it will simply display an error to the user:



...upon install .
This is notable for two reasons:

1. Based on the use of the Chinese language, this shows the malware is geared towards Chinese users.

2. Along with the fact that the malware is packaged up in an easily runnable application (masquerading as Flash), this indicates that this version of the malware is designed to be deployed via socially engineering methods. This is different from the 2021 version mentioned by Google that was deployed via (remote) exploitation.

   This really isn't too surprising. Sophisticated APT groups often split out their implants from their exploits. Besides allowing multiple people (or even teams) to focus on areas of expertise (e.g. writing remote macOS exploits), this also allows the two to be decoupled …meaning, the implant can be deployed in a variety of ways (social engineering, exploitation, etc. etc.).

2021

Now on to the 2021 version, (`cf5e...`). This binary seems to directly comparable the `client` (recall that was found in the `/Resources` of the application, and launched via the `SafariFlashActivityinstall` bash script when the application was launched).

In Google's report, they note that this binary (`cf5e...`) was downloaded and executed up successful exploitation. For purposes of analysis we can simply run it directly from the Terminal.

During this results in actions similar to those performed by the (older) `client` binary including:

- The creation of a directory named `Tools` in the `~/Library/Preferences/` directory, into which it drops several custom tools (named `arch`, `at`, etc.).
- The persistence of Launch Agent (or likely daemon is running as root) via the `com.UserAgent.va.plist`.
  As the `RunAtLoad` key is set to `true` the specified binary, `/Users/user/Library/Preferences/UserAgent/lib/UserAgent` will be persistently executed by macOS each time the user logs in.

Of note this, version of the malware drops a new tool named `kAgent` (`SHA-1`: `D811E97461741E93813EECD8E5349772B1C0B001`) into the `~/Library/Preferences/Tools` directory.

A quick triage of this binary reveals it's a simple keylogger that leverages Core Graphics Event Taps to intercept user keystrokes:

```
 1int sub_1000028f0(int arg0) {
 2
 3    runLoop = CFRunLoopGetCurrent();
 4    runLoopSource = CFMachPortCreateRunLoopSource(kCFAllocatorDefault, *g_eventTap,
0x0);
 5
 6    CFRunLoopAddSource(*runLoop, runLoopSource, kCFRunLoopCommonModes);
 7    CGEventTapEnable(*g_eventTap, 0x1);
 8
 9    CFRunLoopRun();
10
11    return 0x0;
12}
```

In Google's report they noted that, "*It uses a publish- subscribe model via a Data Distribution Service (DDS) framework for communicating with the C2*"

As the malware is compiled with a myriad or error and logging message, we can confirm this, but also see exactly what capabilities it supports.

Specifically we can extract strings in the format `<number>CDDS`, as these appear to show the requests the implant supports:

- "24CDDSScreenCaptureRequest"
- "28CDDSAutoScreenCaptureRequest"
- "21CDDSScreenCaptureInfo"
- "33CDDSScreenCaptureParameterRequest"
- "19CDDSFileInfoRequest"
- "12CDDSFileInfo"
- "18CDDSDirInfoRequest"
- "11CDDSDirInfo"
- "17CDDSZipDirRequest"

- “21CDDSZipDirRequestInfo”
- “22CDDSExecuteFileRequest”
- “19CDDSTerminalConnect”
- “22CDDSTerminalDisConnect”
- “17CDDSTerminalInput”
- “18CDDSTerminalOutput”
- “20CDDSUninstallRequest”
- “20CDDSClearDataRequest”
- “10CDDSCmdAck”
- “18CDDSReqMacBaseInfo”
- “15CDDSMacBaseInfo”
- “18CDDSReqMacFileList”
- “15CDDSMacFileList”
- “20CDDSMacFileListReply”
- “20CDDSReqMacSearchFile”
- “17CDDSMacSearchFile”
- “22CDDSMacSearchFileReply”
- “21CDDSStopMacSearchFile”
- “20CDDSReqMacDeleteFile”
- “20CDDSReqMacFileSystem”
- “17CDDSBaseInfoReply”
- “14CDDSReqMacTree”
- “13CDDSDriveInfo”

…based off these tasking strings, it’s clear to see the implant supports a myriad of features!

This is also why this malware is named OSX.CDDS!

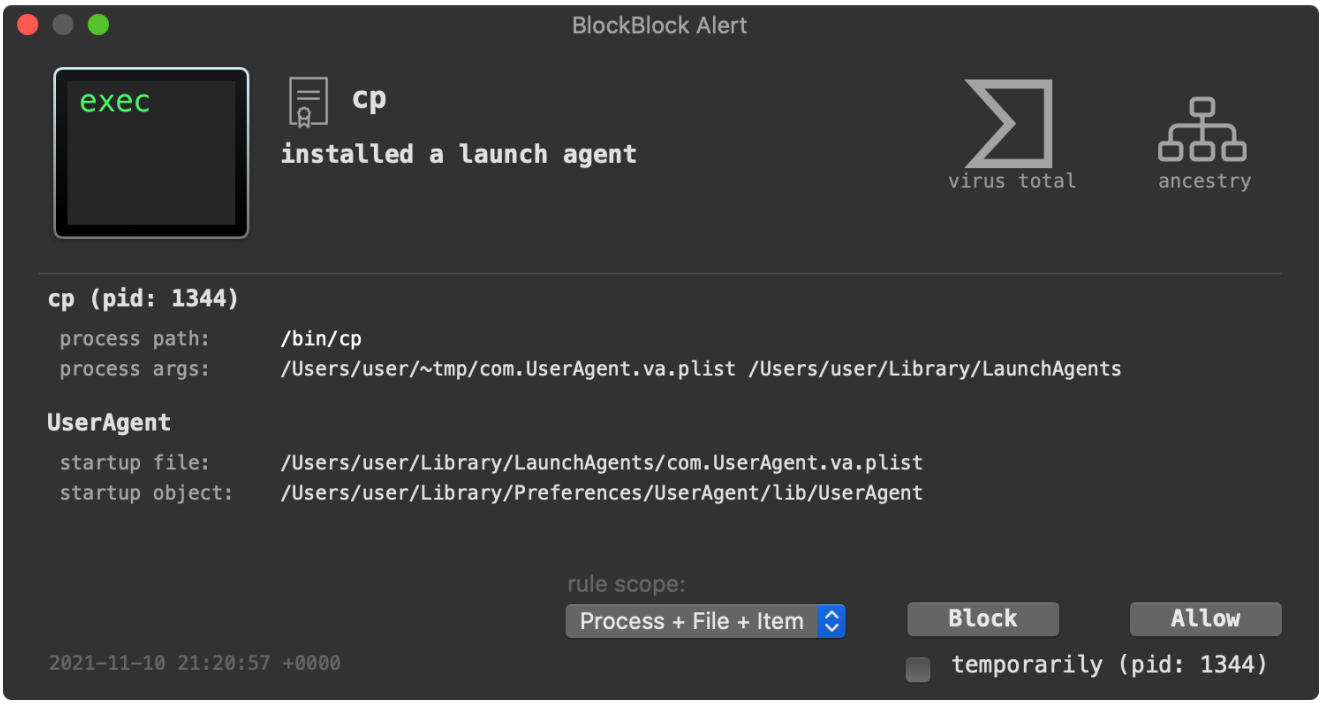
`OSX.CDDS` .vs Objective-See

Whenever a new piece of malware is uncovered I like to see how Objective-See’s free open-source tools stack up.

Good news (and no really no surprise) they are able to detect and thus thwart this new threat, even with no a priori knowledge of it! 😍

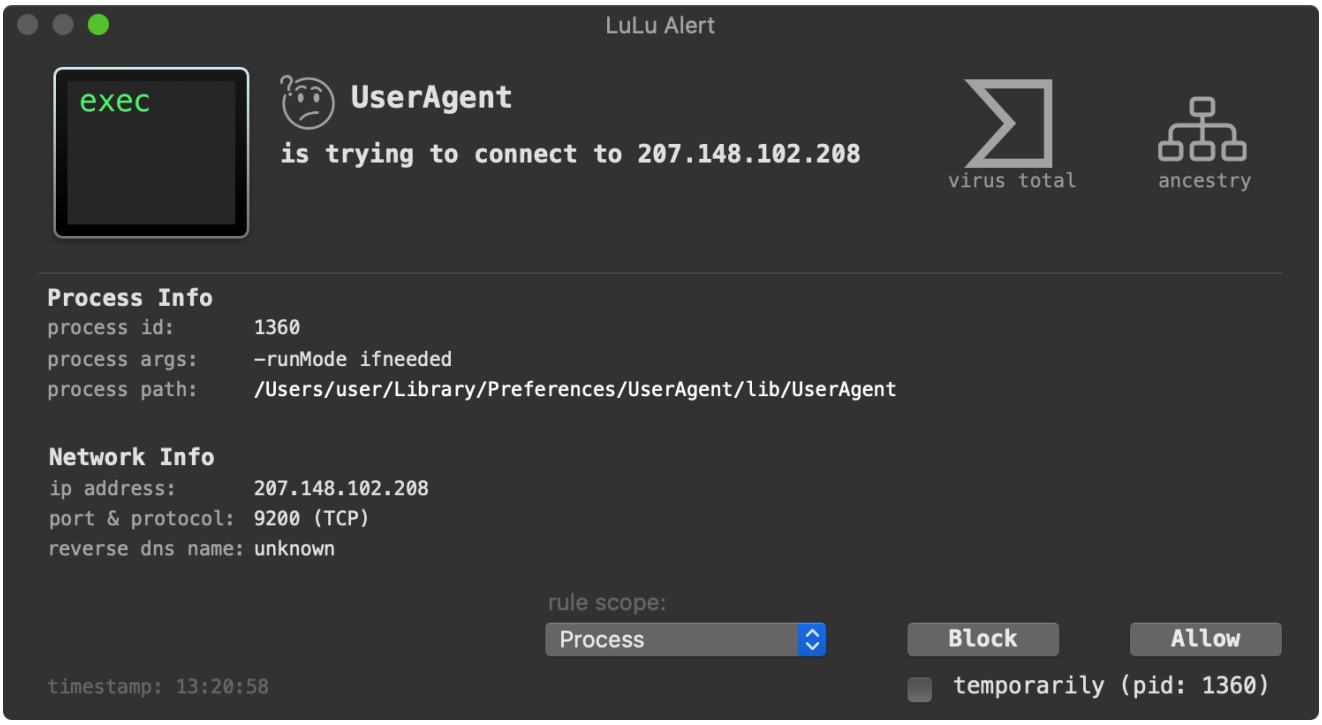
Let’s look at how!

First, BlockBlock detects OSX.CDDS’ attempt at persistence …specifically when it executes `cp` to create a launch item:



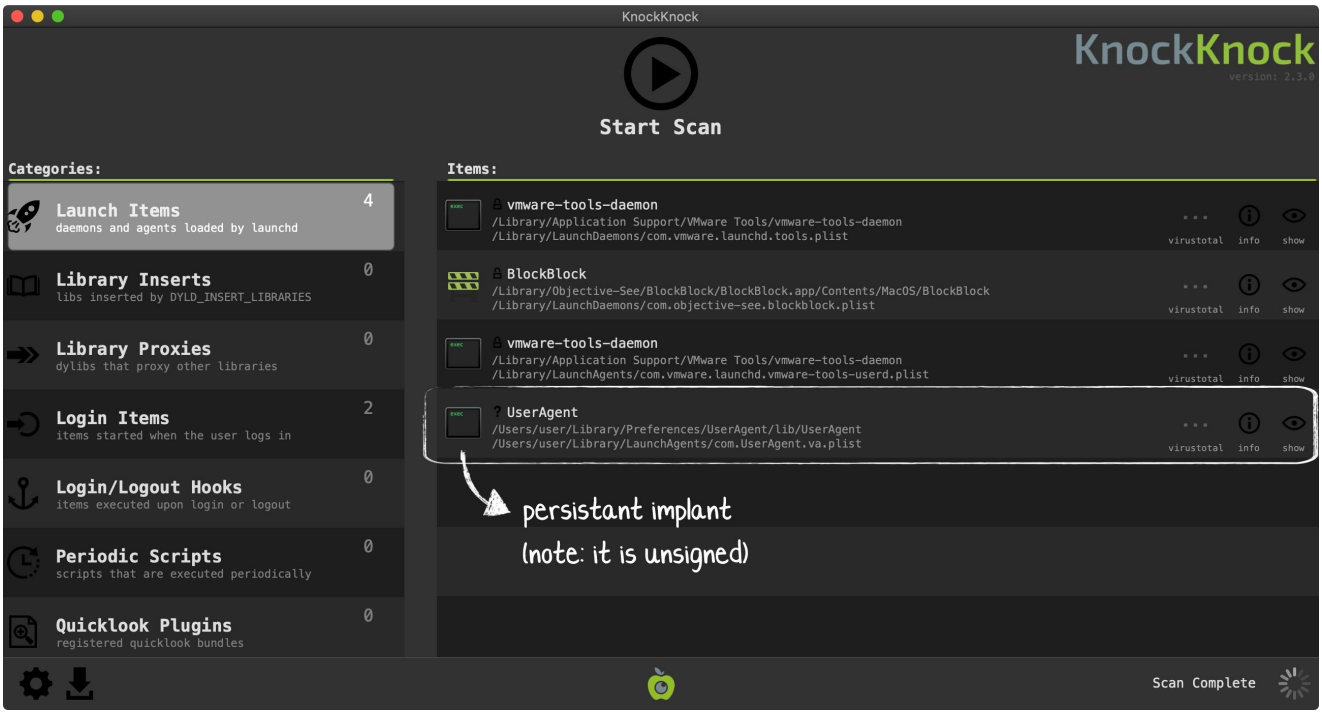
BlockBlock alert

LuLu, our free, open-source firewall detects when the implant first attempts to beacon out to its command and control server to check-in and ask for tasking:



LuLu alert

And if you're worried that you are already infected? KnockKnock can uncover the malware's persistence (after the fact):



KnockKnock detection

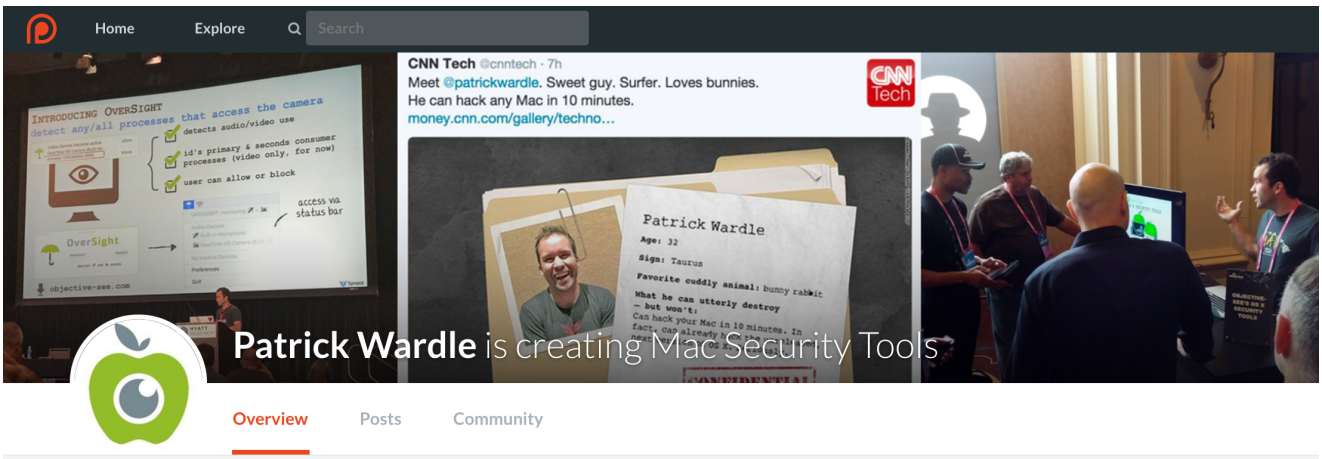
Conclusions

It's not everyday we come across a brand new fully-featured macOS implant to analyze. 🤗

Today, we triaged OSX.CDDS, an implant (whose latest version) Google detected being remotely deployed via n-day and 0-day exploits.

💕 Support Me:

Love these blog posts? You can support them via my Patreon page!



This website uses cookies to improve your experience.