SysJoker

analyzing the first (macOS) malware of 2022!

by: Patrick Wardle / January 11, 2022

📝 👾 Want to play along?

I've uploaded an macOS **SysJoker sample** (password: infect3d).

...please don't infect yourself!

Background

Earlier today (January 11th), Researchers at Intezer published an report titled, "**New SysJoker Backdoor Targets Windows, Linux, and macOS**."

In this report, they detailed a new cross-platform backdoor they named `SysJoker`. Though initially discovered on Linux, the Intezer researchers shortly thereafter also found both Windows and Mac versions:

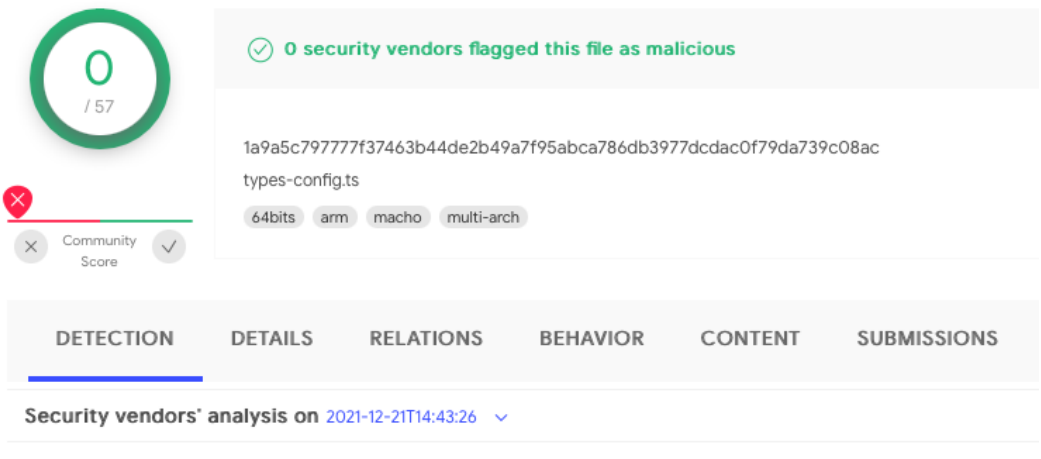
> *"SysJoker was first discovered during an active attack on a Linux-based web server of a leading educational institution. After further investigation, we found that SysJoker also has Mach-O and Windows PE versions." -Intezer*

As their report mostly focuses upon the Windows version, here we build upon their excellent research and dive into the macOS version of `SysJoker`

Triage

Intezer's report provided a hash for the macOS version of `SysJoker`:
`1a9a5c797777f37463b44de2b49a7f95abca786db3977dcdac0f79da739c08ac`.

Popping over to VirusTotal, we can grab a copy of the malicious binary, as well as noted that it was first submitted on `2021-12-21` with 0 detections:



SysJoker (macOS)... on VirusTotal

> Good news though, now anti-virus engines are **starting to flag it** as malicious.

This is the sample, we'll be diving in to today.

The file is named `types-config.ts` and based on its file extension, `.ts`, masquerades as a video file (specifically a video transport stream file):

...I'm not malware, I promise! 🤣

Using macOS' built-in `file` command we can see that in reality it's a universal ("fat") mach-O binary, containing both Intel and arm64 builds:

```
% file SysJoker/types-config.ts
SysJoker/types-config.ts: Mach-O universal binary with 2 architectures:
[x86_64:Mach-O 64-bit executable x86_64] / [arm64:Mach-O 64-bit executable arm64]

SysJoker/types-config.ts (for architecture x86_64):  Mach-O 64-bit executable x86_64
SysJoker/types-config.ts (for architecture arm64):   Mach-O 64-bit executable arm64
```

The `arm64` build ensures the malware can run natively on Apple Silicon (M1).

WhatsYourSign, my open-source utility that displays code-signing information via the UI, shows that this binary is signed, albeit by an adhoc signature:

SysJoker signed, though ad-hoc

You can also use macOS' `codesign` utility (note the `flags=0x2(adhoc)`):

```
% codesign -dvv SysJoker/types-config.ts

SysJoker/types-config.ts
Identifier=test-555549448174817ef4cf398d975b7860466eaec7
Format=Mach-O universal (x86_64 arm64)
CodeDirectory v=20400 size=1510 flags=0x2(adhoc) hashes=36+7 location=embedded
Signature=adhoc
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=0 size=12
```

Now let's run the `string` utility to extracted any embedded (ASCII) strings:

```
% strings - SysJoker/types-config.ts
updateSystem
updateMacOs
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBg...zy0eF1HqtBNbkXiQ6SSbquuvFPUepqUEjUSQIDAQAB

whoami
/Users/
/Library/MacOsServices
/Library/SystemNetwork
https://drive.google.com/uc?export=download&id=1W64PQQxrwY3XjBnv_QAeBQu-ePr537eu
/Library/LaunchAgents
/Library/LaunchAgents/com.apple.update.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>Label</key>
<string>com.apple.update</string>
        <key>LimitLoadToSessionType</key>
        <string>Aqua</string>
<key>ProgramArguments</key>
<array>
<string>
</string>
</array>
<key>KeepAlive</key>
        <dict>
           <key>SuccessfulExit</key>
           <true/>
        </dict>
        <key>RunAtLoad</key>
        <true/>
</dict>
</plist>
0123456789ABCDEFGHIJKLMNOPQRTUVWXYZabcdefghijklmnopqrstuvwxyz
.zip
unzip -o '
' -d '
chmod 0777 '
nohup '
' >/dev/null 2>&1 &
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like
Gecko) Version/14.1.2 Safari/605.1.15
response:
domain
serial=
&name=
```

```
&os=
&anti=
&ip=
&user_token=987217232
...
```

The `strings` output includes:

- Various commands, such as: `whoami`, `unzip`, `chmod` etc.
- Paths: `/Library/MacOsServices`, `/Library/LaunchAgents/com.apple.update.plist`, etc.
- An embedded URL: `https://drive.google.com/uc?...`
- An embedded launch item property list template (`com.apple.update.plist`) likely for persistence.
- Parameters for a survey? `name=`, `os=`, `ip=`, etc...
- An decryption key? `MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBg...`

Persistence

As the malware appears to be written in C++ (🤮), let's begin by using various static analysis tools observe it behavior, first focusing on it's persistence.

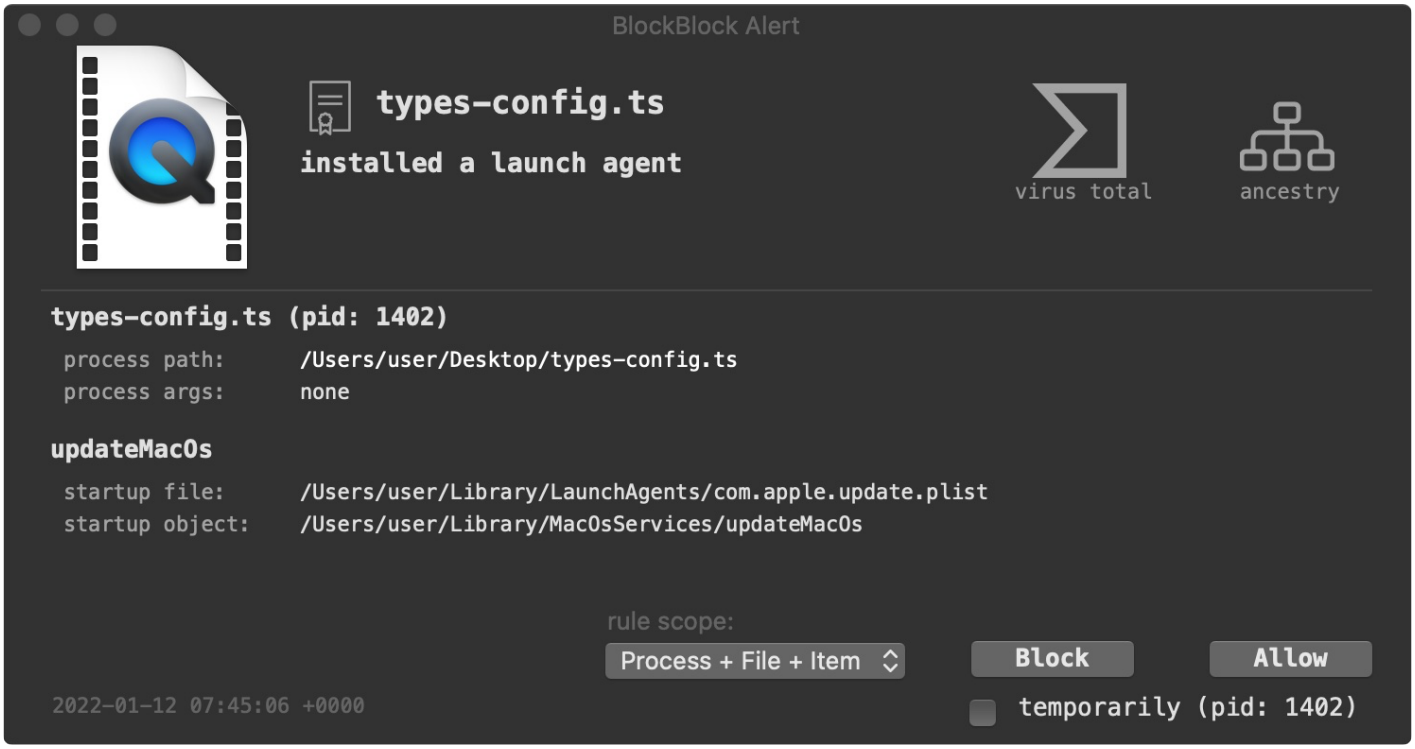
Via my **`ProcessMonitor`**, we can observe many of the malware's actions, such as the fact that when initially run, it copies itself to the user's `Library/MacOsServices/` directory, as `updateMacOs` ...and then launches this copy:

```
# ProcessMonitor.app/Contents/MacOS/ProcessMonitor -pretty
...

{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
    ...
    "arguments" : [
      "cp",
      "./types-config.ts",
      "/Users/user/Library/MacOsServices/updateMacOs"
    ],
    "path" : "/bin/cp",
    "name" : "cp",
    "pid" : 1404
  }
  ...
}

{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
    ...
    "arguments" : [
      "sh",
      "-c",
      "nohup '/Users/user/Library/MacOsServices/updateMacOs' >/dev/null 2>&1 &"
    ],
    "path" : "/bin/bash",
    "name" : "bash",
    "pid" : 1405
  }
  ...
}
```

As we have **BlockBlock** installed it then detects the malware attempting to persist:



SysJoker's persistence

If we allow the malware to persist, we can take a peek at the property list, `com.apple.update.plist` it creates:

```
% cat ~/Library/LaunchAgents/com.apple.update.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>com.apple.update</string>
        <key>LimitLoadToSessionType</key>
        <string>Aqua</string>
    <key>ProgramArguments</key>
    <array>
        <string>/Users/user/Library/MacOsServices/updateMacOs</string>
    </array>
    <key>KeepAlive</key>
    <dict>
        <key>SuccessfulExit</key>
        <true/>
    </dict>
    <key>RunAtLoad</key>
    <true/>
</dict>
</plist>
```

No surprises here. The launch agent plist (populated from the template we saw as an embedded string) points the malware's copy: `/Users/user/Library/MacOsServices/updateMacOs`. And, as the `RunAtLoad` key is set to `true`, the malware will be restarted each time the user logs in.

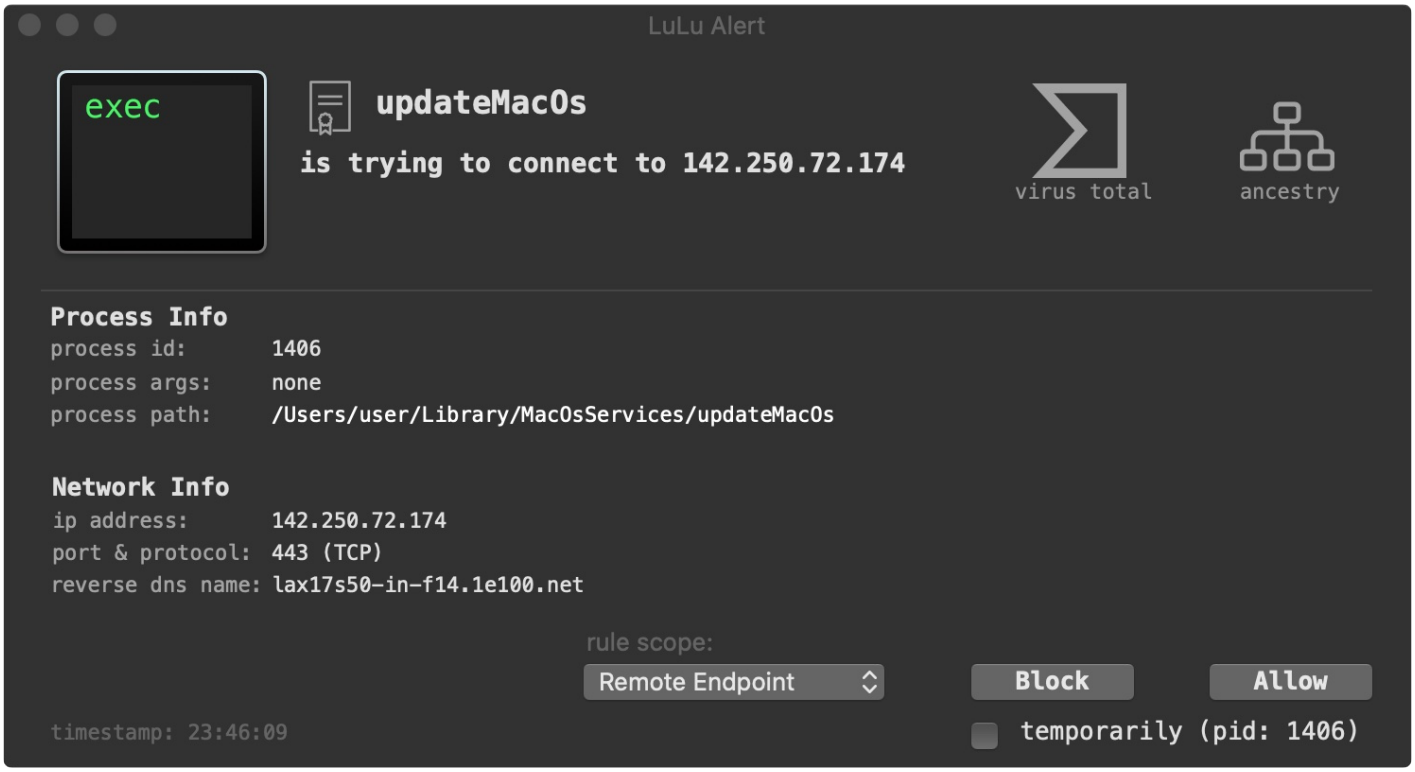
Command and Control Communications

Once the malware has persisted itself, Intezer notes that it will:

> "*generates its C2 by decoding a string retrieved from a text file hosted on Google Drive.* " -*Intezer*

From running the `strings` utility, recall we extracted what appeared to be the address of this file:
`https://drive.google.com/uc?export=download&id=1W64PQQxrwY3XjBnv_QAeBQu-ePr537eu`

As we have **LuLu**, our free macOS firewall, installed, we are alerted when the malware attempts to reach out and download this file. (Note the the IP address, `142.250.72.174` in the LuLu alert maps to URL owned by Google (e.g. `drive.google.com`)):



SysJoker's connection to Google (drive) to generate a C2 server

The malware performs it's network comms via the `curl` API. For example, in a debugger, we can observe it invoking `curl_easy_setopt` with `CURLOPT_URL` (`0x2712`), with the aforementioned Google drive URL:

```
Process 1424 stopped
* thread #1, queue = 'com.apple.main-thread'

libcurl.4.dylib`curl_easy_setopt:
->  0x7fff6543dff7 <+0>:  pushq  %rbp

Target 0: (updateMacOs) stopped.
(lldb) reg read $rsi
rsi = 0x2712

(lldb) x/s $rdx
0x7fda91c08ba0:
"https://drive.google.com/uc?export=download&id=1W64PQQxrwY3XjBnv_QAeBQu-ePr537eu"
```

As this URL is still live we can manually grab the file (`domain.txt`):

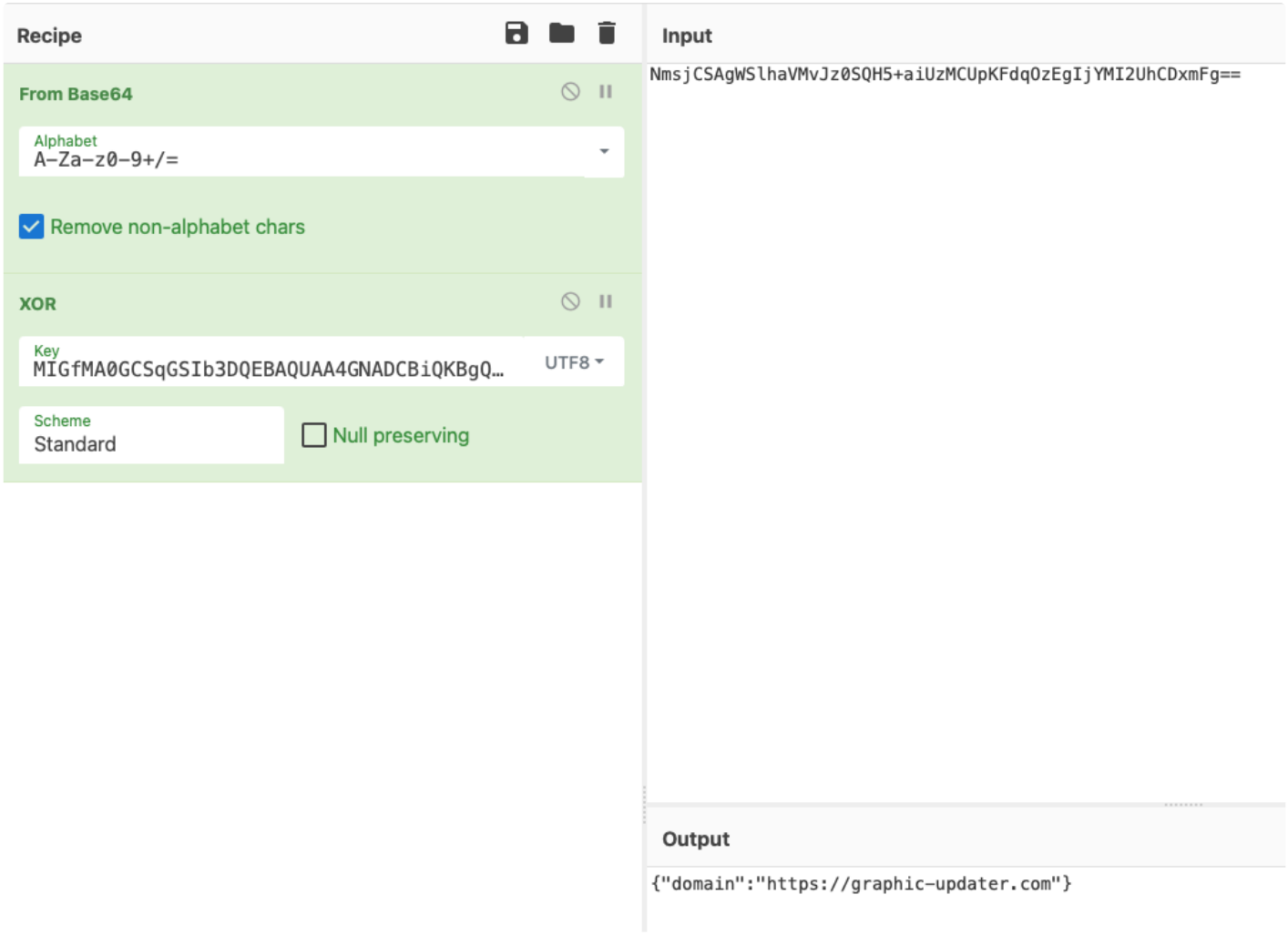
```
cat ~/Downloads/domain.txt

NmsjCSAgWSlhaVMvJz0SQH5+aiUzMCUpKFdqOzEgIjYMI2UhCDxmFg==
```

The embedded base64 encoded string `NmsjCSAgWSlhaVMvJ...mFg==` decodes to binary data, which recall, Intezer noted is used to "*generates its C2*".

As shown in their report, we can base64 decode this string, then XOR it with the embedded key,

`MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBg...zy0eF1HqtBNbkXiQ6SSbquuvFPUepqUEjUSQIDAQAB` to get decrypt the command and control server (result: `graphic-updater.com`):



Decrypting the (current) command and control server

We could have also just let the malware continue to run in the debugger and (rather lazily) uncover the server:

```
(lldb) x/s $rdx
0x7fda91cafe60: "https://graphic-updater.com/api/attach"
```

We can see (by dumping the parameters passed to other invocations of `curl_easy_setopt`) that the malware sets it's user-agent to `"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Safari/605.1.15"`. We can also dump the survey information that is sent to the C&C server on its initial checkin:

```
(lldb) x/s $rdx
0x7fda91cafef0: "serial=user_x&name=user&os=os&anti=av&ip=ip&user_token=987217232"
```

While this (brief) survey information contains the name of the logged in user (`user` on my analyis VM), though other fields seems to be unset (`ip=ip`), or hardcoded (recall `987217232` was extracted as an embedded string).

We can resolve the URL `graphic-updater.com` to `23.254.131.176`:

```
% nslookup graphic-updater.com
```

```
Server:     192.168.86.1
Address:    192.168.86.1#53

Non-authoritative answer:
Name:   graphic-updater.com
Address: 23.254.131.176
```

…which still appears to be online, and “open”?

SysJoker's command and control server?

Capabilities (Commands)

The Intezer **report** notes that all versions (Linux, Windows, and Mac) support commands named `exec` and `cmd`:

> *"[the exec] command is in charge of dropping and running an executable. SysJoker will receive a URL to a zip file, a directory for the path the file should be dropped to, and a filename that the malware should use on the extracted executable. It will download this file, unzip it and execute it."*
>
> *[the cmd] command is in charge of running a command and uploading its response to the C2." -Intezer*

Disassembling the Mac version of `SysJoker`, we find the function (at `0x0000000100005f80`) responsible for parsing the tasking from the command and control server, including the aforementioned `exec` and `cmd` commands.

First, the `exec` command:

```
int sub_100005f80(...) {

    rax = std::__1::basic_string<char, std::__1::char_traits<char>,
std::__1::allocator<char> >::compare(&var_E0, 0x0, 0xffffffffffffffff, "exe", 0x3);
    if (rax == 0x0) goto handleExec;
    ...

handleExec:

  rax = sub_100004e76(&var_60, "url");
  rax = sub_100004e76(&var_60, "dir");
  rax = sub_100004e76(&var_60, "name");
  ...

}
```

In the above disassembly you can see that if malware is tasked with the `exec` command, it will first extract the command's parameters (`url`, `dir`, `name`, etc.).

The code to then unzip the downloaded executable and execute it, appears at `sub_100003995`. This function invokes:

- `unzip -o` to unzip the executable,
- `chmod 0777` to change the permissions (on the now unzipped executable)
- `system` to execute the binary.

The function (at `0x0000000100005f80`) is also responsible for handling the `cmd` command:

```
int sub_100005f80(...) {

    ...

    rax = std::__1::basic_string<char, std::__1::char_traits<char>,
std::__1::allocator<char> >::compare(&var_E0, 0x0, 0xffffffffffffffff, "cmd", 0x3);
    if (rax == 0x0) {
    ...
    rax = sub_100004e76(&var_60, "command");
    ...

}
```

After extracting the commands parameter (`command`), it appears to invoke the `popen` API (via a helper function found at `0x00000010000256b`), to execute the command. As noted by Intezer, the results of the executed command will be uploaded to the command and control server.

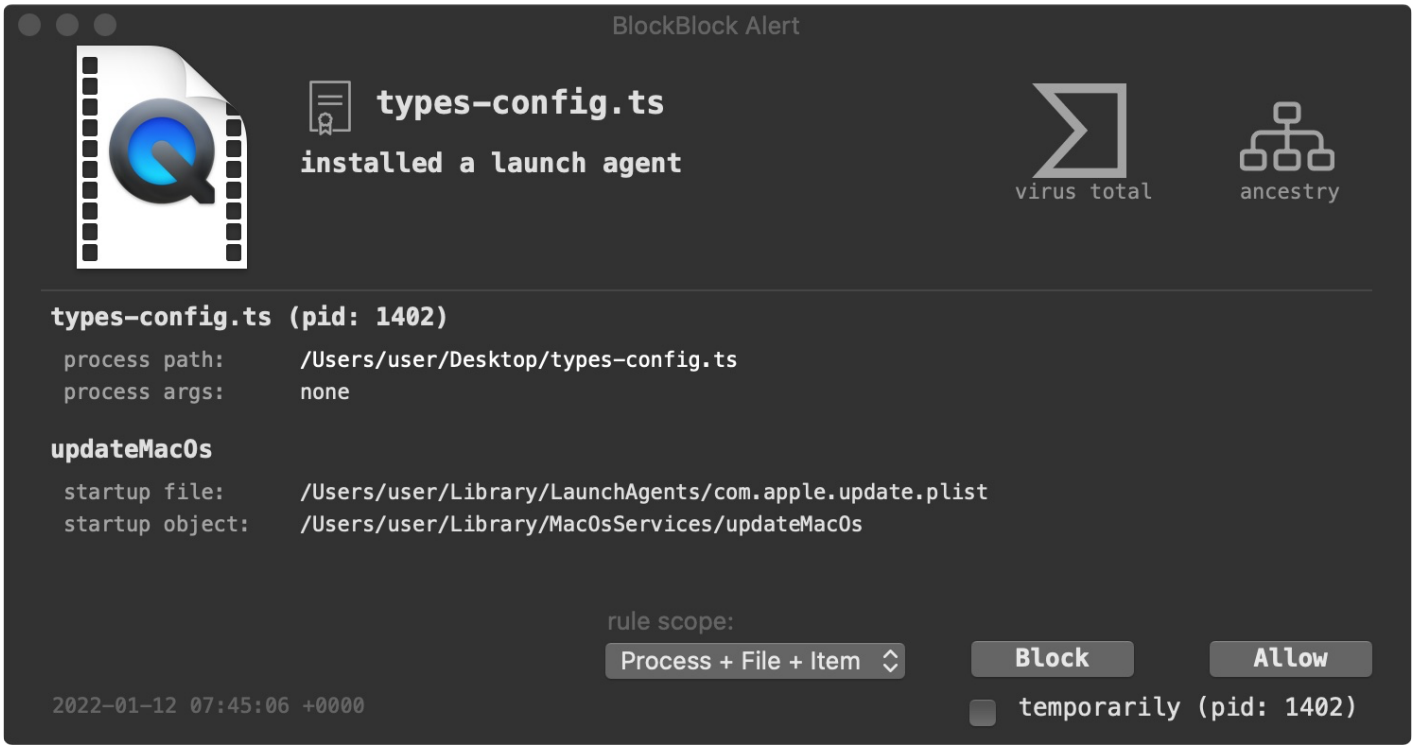
`SysJoker` vs. Objective-See

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

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

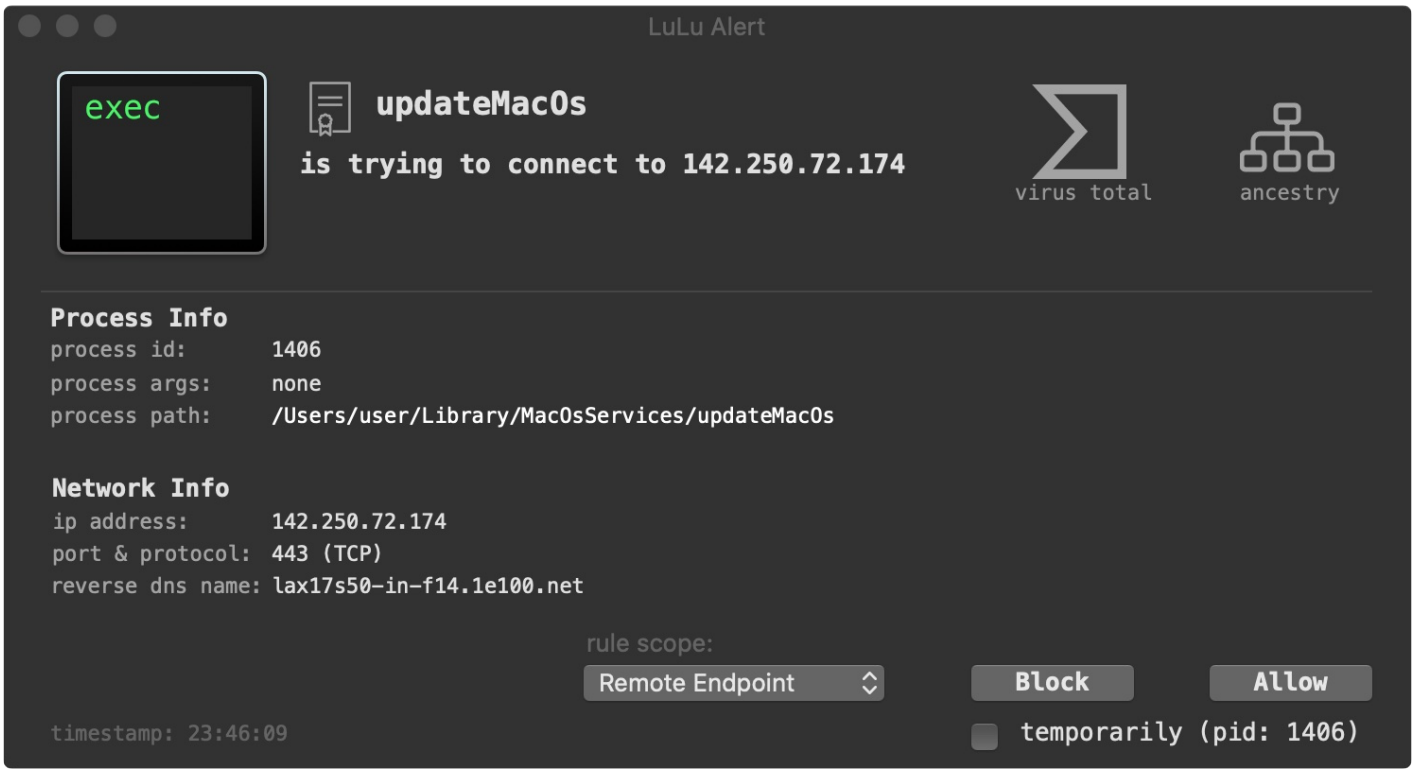
Let's look at how!

First, **BlockBlock** detects the malware's launch agent persistence (`com.apple.update.plist`):



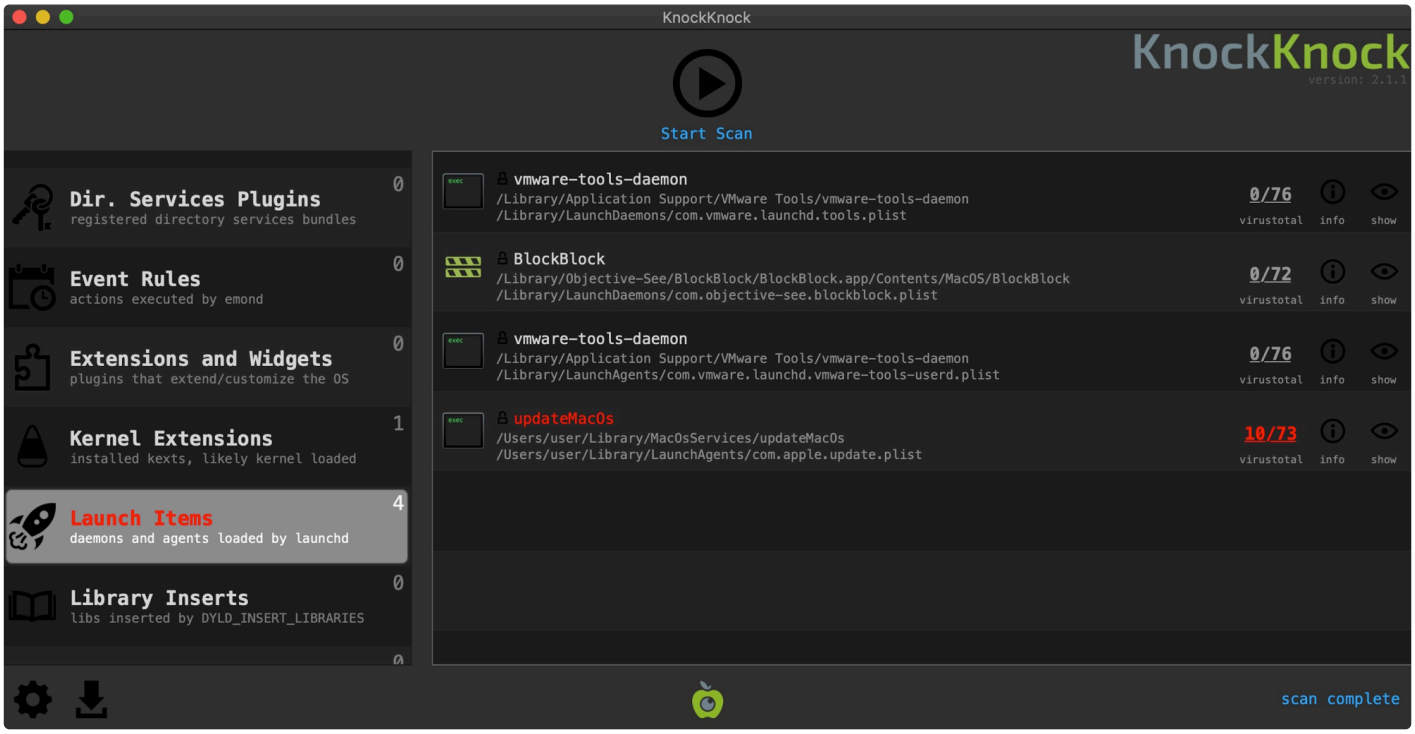
BlockBlock alert

LuLu, our free, open-source firewall detects when the malware first attempts to beacon out to grab the encrypted address of it's command and control server:



LuLu alert

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



KnockKnock detection

Conclusions

So there you have it, the first malware of 2022! Hooray? 🤗

In this blog post, we dove into the macOS version of SysJoker, a cross-platform, 1st-stage backdoor. We built upon Intezer's report, delving deeper into the macOS version, highlighting its:

- Persistence
- C&C communications
- Capabilities (commands)

Finally, we showed that if you were running Objective-See's free macOS tools that malware wouldn't have stood a chance! 😁

💞 Support Me:

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