Detecting XLoader | A macOS ‘Malware-as-a-Service’ Info Stealer and Keylogger

sentinelone.com/blog/detecting-xloader-a-macos-malware-as-a-service-info-stealer-and-keylogger/

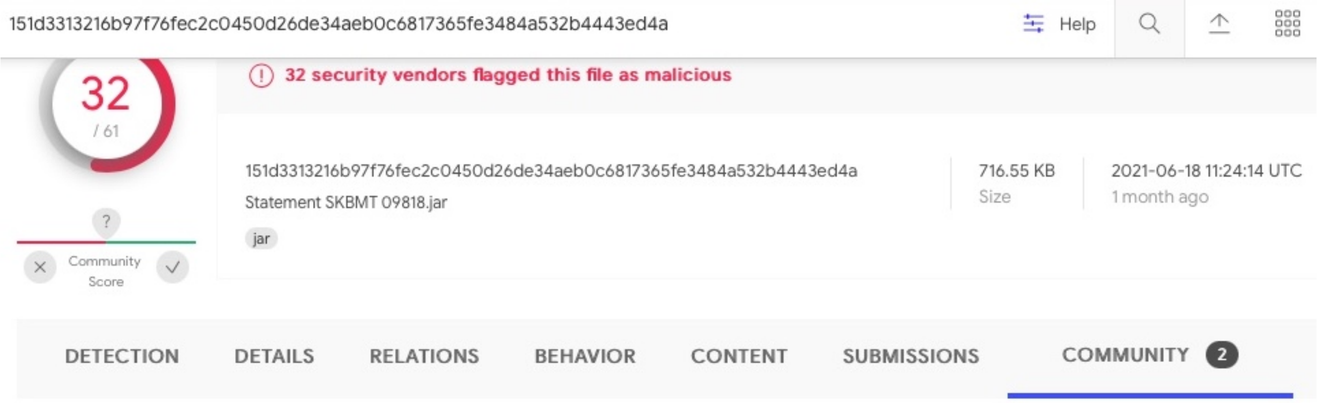
July 26, 2021

Threat actors have come to recognize the reality that today’s organizations operate fleets of devices encompassing all the major OS vendors – Apple, Microsoft, Google and many flavors of Linux – and are adapting accordingly. Threats that can be compiled on one platform but produce executables targeting many are a productivity boon to criminals, who now operate in an increasingly competitive environment trying to sell their wares.

The latest such threat to come to attention is XLoader, a Malware-as-a-Service info stealer and keylogger that researchers say was developed out of the ashes of FormBook. Unlike its Windows-only predecessor, XLoader targets both Windows and macOS. In this post, we take an initial look at the macOS version of XLoader, describe its behavior and show how XLoader can be detected on Apple’s Mac platform.

XLoader for Mac – Java Runtime For the Steal

The macOS sample we analyzed comes as both a standalone binary and as a compiled `.jar` file. The `.jar` file appears to be distributed as an attachment in a phishing lure, such as in this document `Statement SKBMT 09818.jar`.


151d3313216b97f76fec2c0450d26de34aeb0c6817365fe3484a532b4443ed4a

32 / 61

32 security vendors flagged this file as malicious

151d3313216b97f76fec2c0450d26de34aeb0c6817365fe3484a532b4443ed4a
Statement SKBMT 09818.jar
jar

716.55 KB
Size

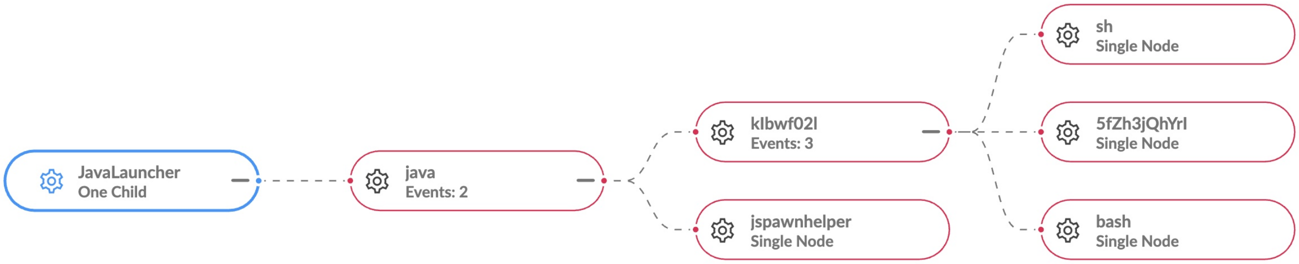
2021-06-18 11:24:14 UTC
1 month ago

XLoader is likely distributed by mail spam

Such files require the Java Runtime Environment, and for that reason the malicious `.jar` file will not execute on a macOS install out of the box, since Apple stopped shipping JRE with Macs over a decade ago.

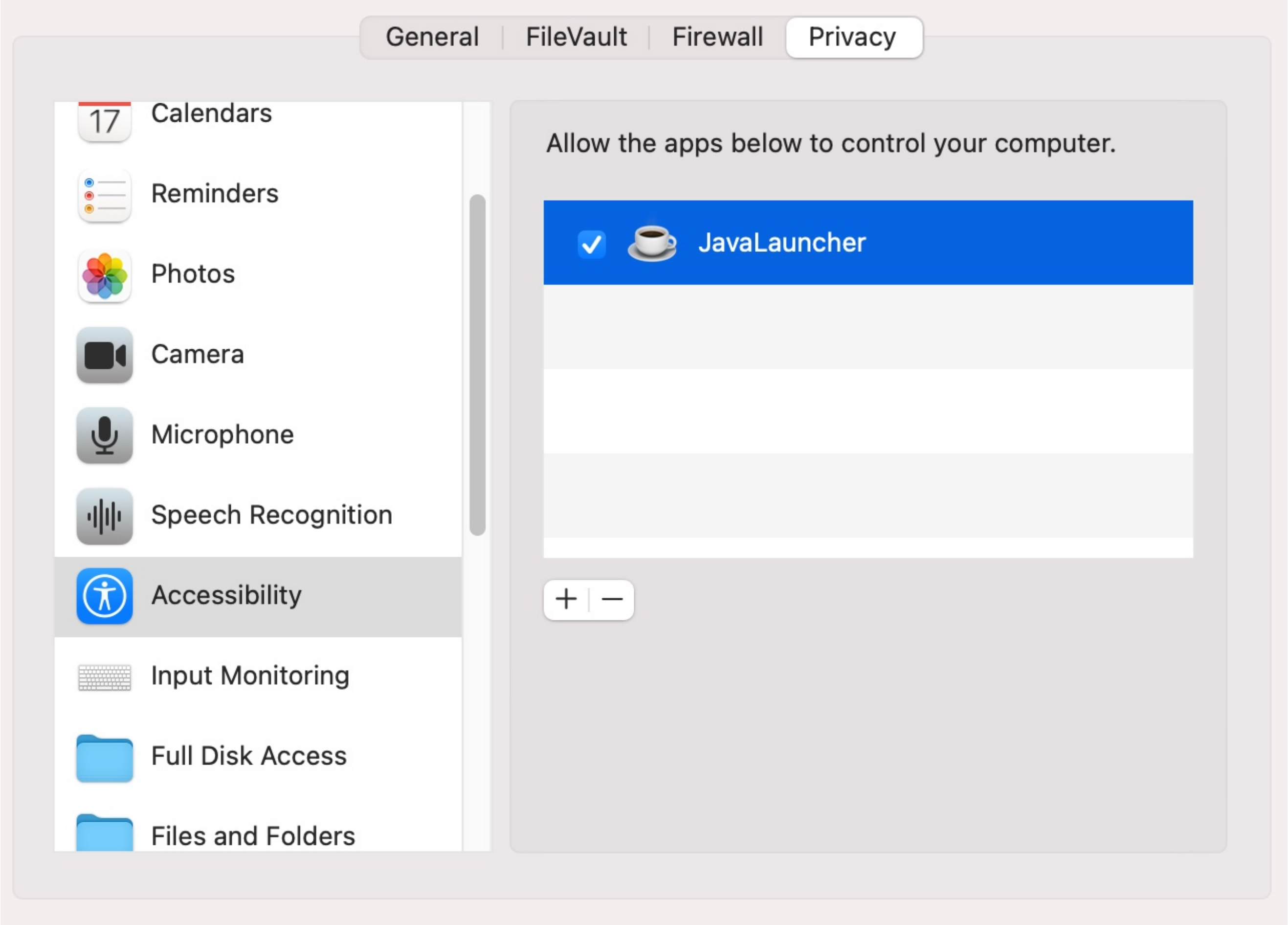
Nonetheless, Java is still a common requirement in enterprise environments and is still in use for some banking applications. As a result, many organizations will have users that either do or must install the Oracle version of Java to meet these needs. As a 3rd party plugin, the Oracle JRE is installed at `/Library/Internet Plug-Ins/JavaAppletPlugin.plugin`.

When the malware is executed as a `.jar` file, the execution chain begins with the OS-provided JavaLauncher at `/System/Library/CoreServices/JavaLauncher.app`.



XLoader’s execution chain begins with the JavaLauncher

The JavaLauncher is also populated in the Accessibility pane in **System Preferences’ Privacy** tab and a dialog is popped requesting the user to grant access for automation. As we shall see below, this is likely leveraged as part of the info stealer’s functionality.



Allow the apps below to control your computer.

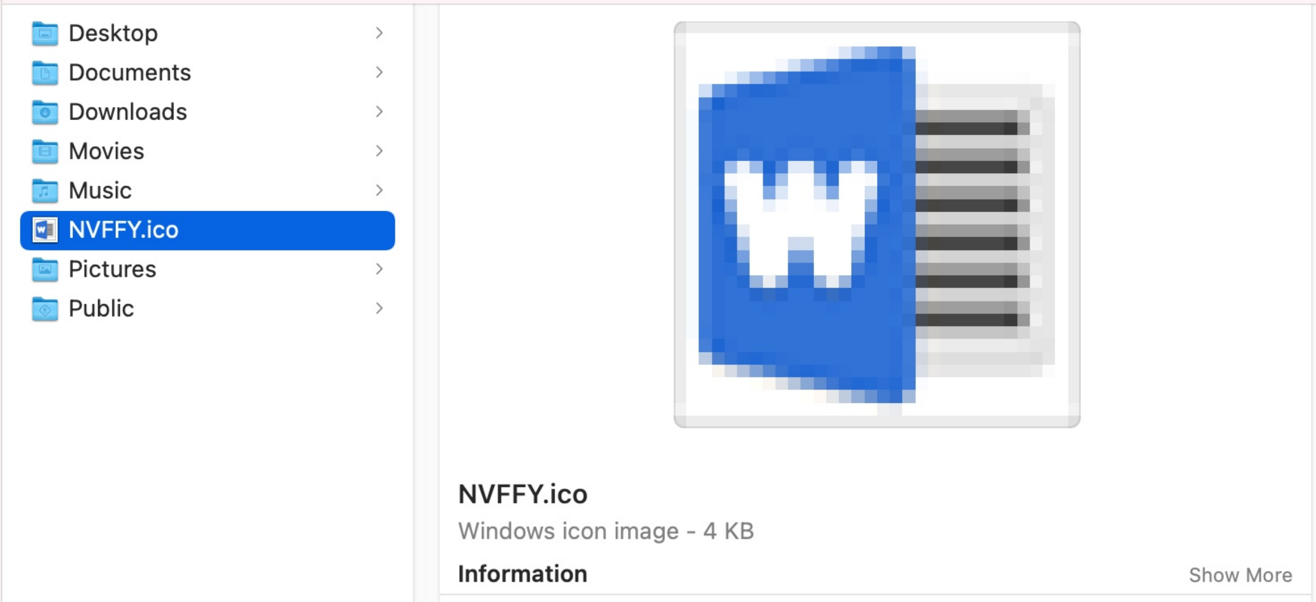
☑ JavaLauncher

The JavaLauncher requests access to control other applications

The `com.oracle.JavaInstaller` will also populate the ‘Full Disk Access’ table in the same tab. This remains unchecked by default and, at least on our test, no dialog was presented to the user to request permissions.

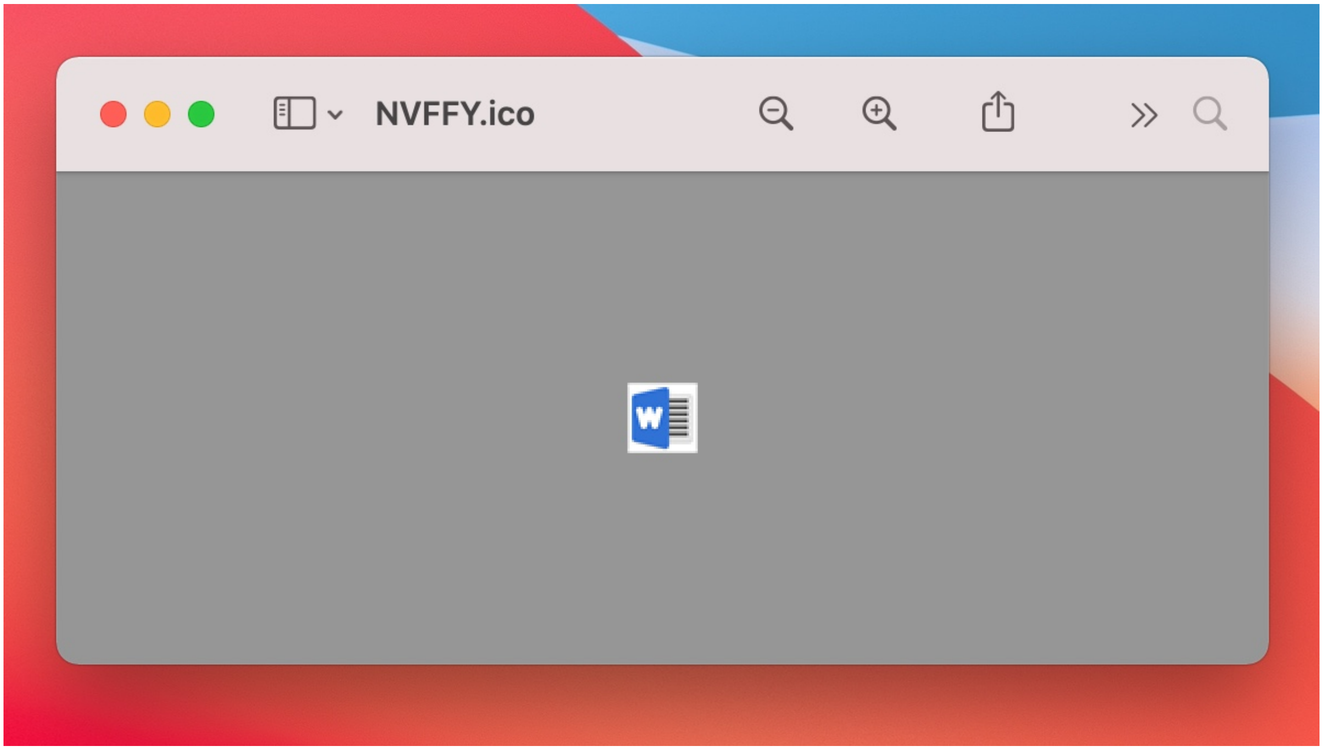
XLoader Behavior on macOS

On execution the malware drops a 32×32 pixel Windows image file in the user's home directory called `NVFFY.ico`.



A Windows icon file is dropped in the user's home folder

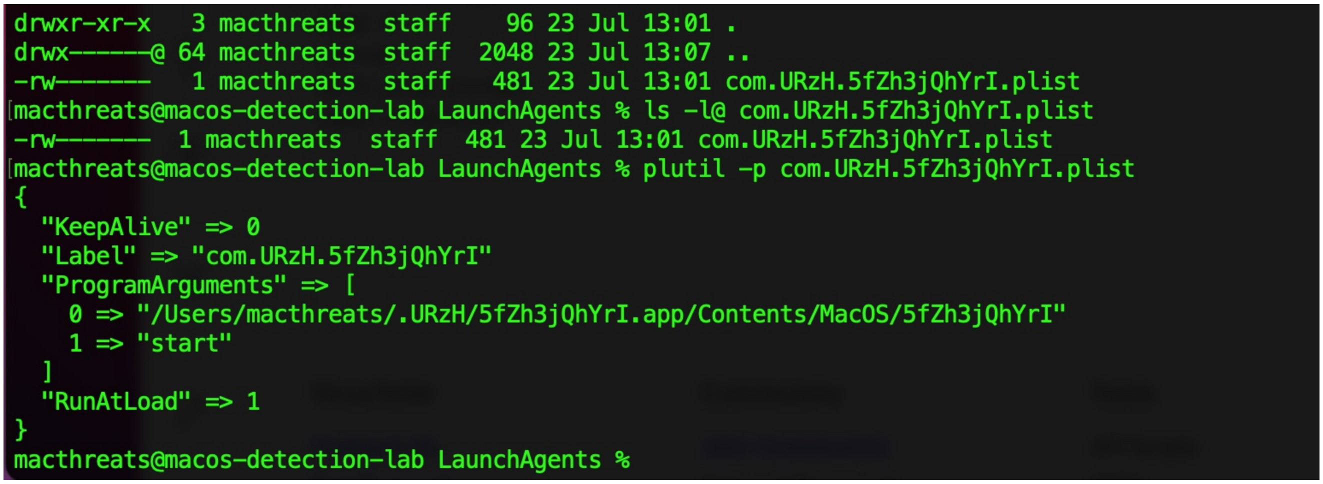
The user's default image viewer – typically the built-in Preview.app – will be launched to display this image. At this point, one could imagine that even the most unsuspecting user opening the 'Statement SKBT' file is going to think that something is amiss.



The .ico file as presented to the victim

It's unclear what the malware authors were thinking here: perhaps the sample is an early development or a test sample. Alternatively, this may be a reflection of the hazards of cross-platform malware, where the author's assumptions on the Windows platform were not fully tested on a macOS device.

In any case, no interaction is required from the user and the malware continues to drop and execute the rest of its components. This involves dropping and executing a Mach-O file in the user's Home folder. This file, `kIbwf02l`, writes a hidden application bundle, also located in the victim's Home folder, and containing a copy of itself. It then writes and loads a user LaunchAgent with a program argument pointing to the copy in the hidden app bundle. From then on, the `kIbwF02l` file appears to be redundant but is not cleaned up by the malware.

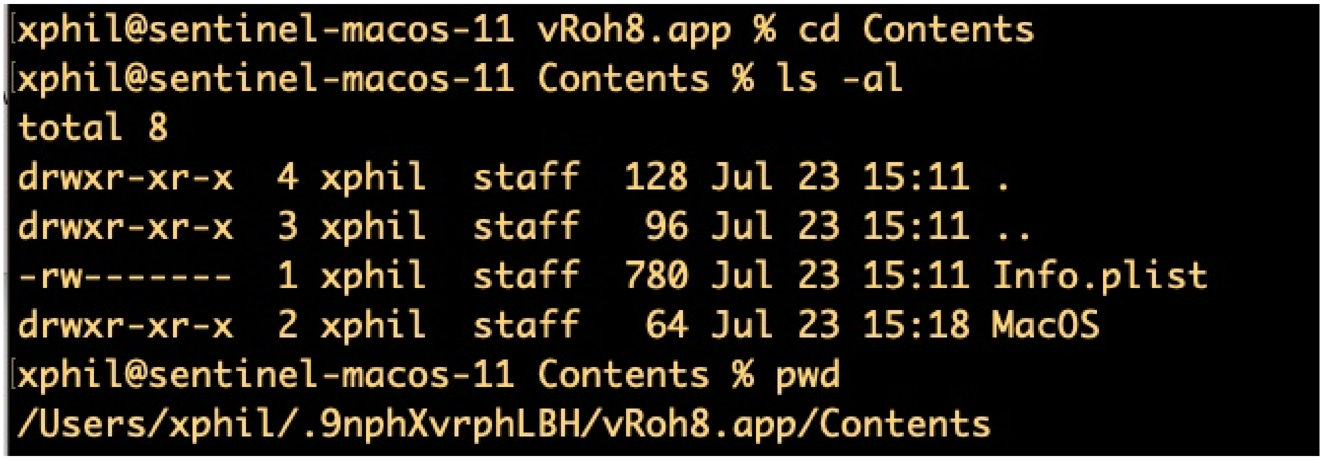


```
drwxr-xr-x   3 macthreats  staff    96 23 Jul 13:01 .
drwx------@ 64 macthreats  staff  2048 23 Jul 13:07 ..
-rw-------   1 macthreats  staff   481 23 Jul 13:01 com.URzH.5fZh3jQhYrI.plist
[macthreats@macos-detection-lab LaunchAgents % ls -l@ com.URzH.5fZh3jQhYrI.plist
-rw-------  1 macthreats  staff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

Example of an XLoader LaunchAgent

The label for the LaunchAgent and the names of the hidden app and executable are all randomized and vary from execution to execution. The binary is passed the argument `start` as a launch parameter.

The hidden application is itself a barebones bundle containing only the Info.plist and the Mach-O executable.



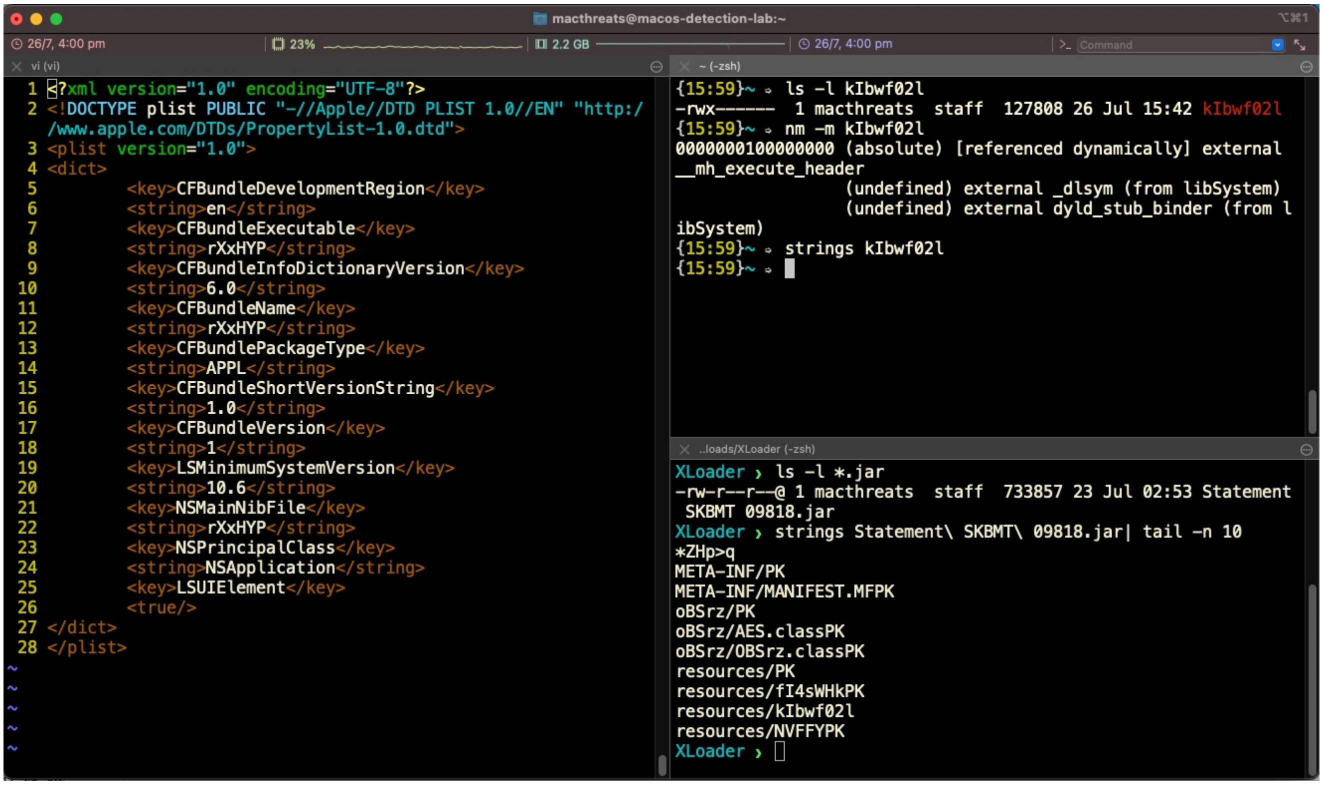
```
[xphil@sentinel-macos-11 vRoh8.app % cd Contents
[xphil@sentinel-macos-11 Contents % ls -al
total 8
drwxr-xr-x  4 xphil  staff  128 Jul 23 15:11 .
drwxr-xr-x  3 xphil  staff   96 Jul 23 15:11 ..
-rw-------  1 xphil  staff  780 Jul 23 15:11 Info.plist
drwxr-xr-x  2 xphil  staff   64 Jul 23 15:18 MacOS
[xphil@sentinel-macos-11 Contents % pwd
/Users/xphil/.9nphXvrphLBH/vRoh8.app/Contents
```

XLoader's hidden application bundle

A copy of the same executable, *sans* bundle and with the filename `kIbwf02l`, is also dropped in the User's home directory.

Analysis of the XLoader Mach-O

The compiled Mach-O executable pointed to by the persistence agent is heavily stripped and obfuscated. As the image below indicates, static analysis using tools like strings will show little, and dynamic analysis is complicated by a number of anti-debugging features.



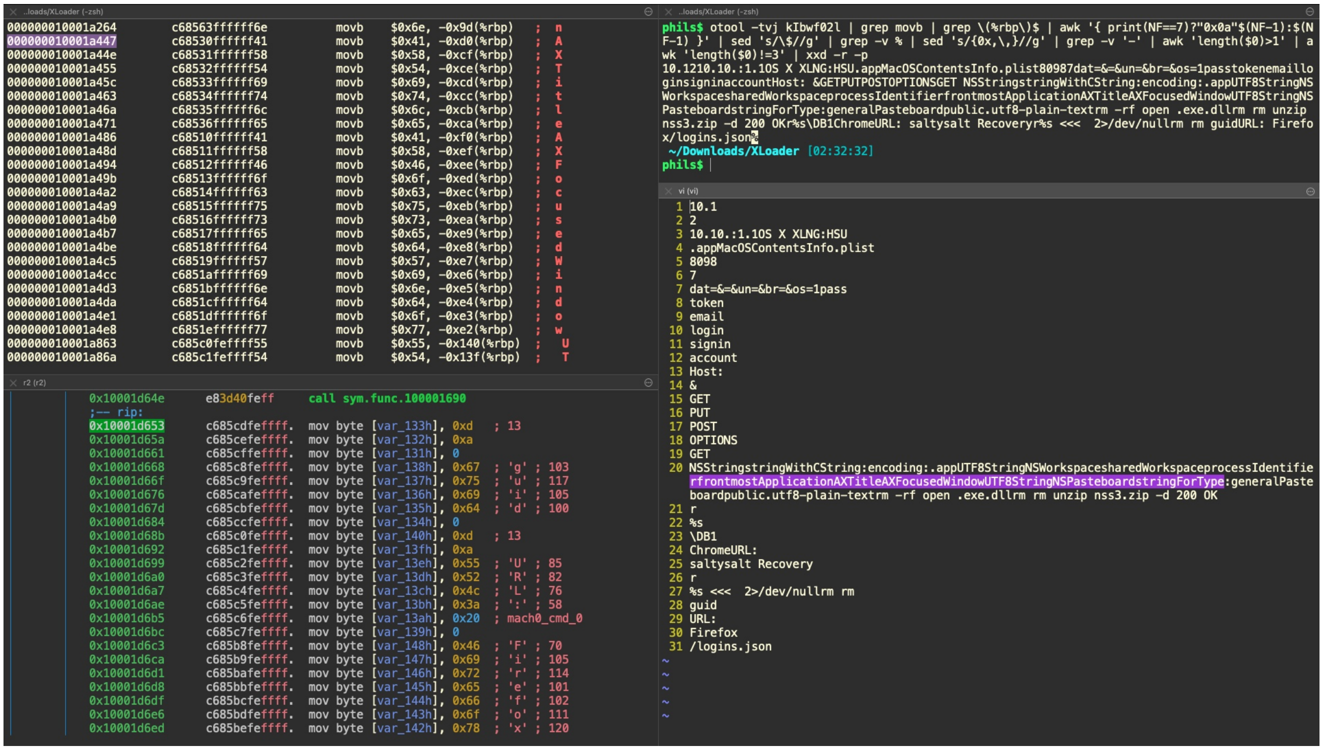
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http:/
/www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>CFBundleDevelopmentRegion</key>
        <string>en</string>
        <key>CFBundleExecutable</key>
        <string>rXxHYP</string>
        <key>CFBundleInfoDictionaryVersion</key>
        <string>6.0</string>
        <key>CFBundleName</key>
        <string>rXxHYP</string>
        <key>CFBundlePackageType</key>
        <string>APPL</string>
        <key>CFBundleShortVersionString</key>
        <string>1.0</string>
        <key>CFBundleVersion</key>
        <string>1</string>
        <key>LSMinimumSystemVersion</key>
        <string>10.6</string>
        <key>NSMainNibFile</key>
        <string>rXxHYP</string>
        <key>NSPrincipalClass</key>
        <string>NSApplication</string>
        <key>LSUIElement</key>
        <true/>
</dict>
</plist>
~
~
~
~
~
```

```
{15:59}~ ⇨ ls -l kIbwf02l
-rwx------  1 macthreats  staff  127808 26 Jul 15:42 kIbwf02l
{15:59}~ ⇨ nm -m kIbwf02l
0000000100000000 (absolute) [referenced dynamically] external
__mh_execute_header
                 (undefined) external _dlsym (from libSystem)
                 (undefined) external dyld_stub_binder (from l
ibSystem)
{15:59}~ ⇨ strings kIbwf02l
{15:59}~ ⇨ 
```

```
XLoader › ls -l *.jar
-rw-r--r--@ 1 macthreats  staff  733857 23 Jul 02:53 Statement
 SKBMT 09818.jar
XLoader › strings Statement\ SKBMT\ 09818.jar| tail -n 10
*ZHp>q
META-INF/PK
META-INF/MANIFEST.MFPK
oBSrz/PK
oBSrz/AES.classPK
oBSrz/OBSrz.classPK
resources/PK
resources/fI4sWHkPK
resources/kIbwf02l
resources/NVFFYPK
XLoader › 
```

Left: the hidden app’s Info.plist. Right: strings and symbols in the executables

For the purposes of quick triage, we extracted the stackstrings from the Mach-O using otool to get an initial idea of the info stealer’s functionality. With further processing either manually or with radare2, we can match these strings to particular functions.



```
000000010001a264        c68563ffffff6e          movb    $0x6e, -0x9d(%rbp)    ;  n
000000010001a447        c68530ffffff41          movb    $0x41, -0xd0(%rbp)    ;  A
000000010001a44e        c68531ffffff58          movb    $0x58, -0xcf(%rbp)    ;  X
000000010001a455        c68532ffffff54          movb    $0x54, -0xce(%rbp)    ;  T
000000010001a45c        c68533ffffff69          movb    $0x69, -0xcd(%rbp)    ;  i
000000010001a463        c68534ffffff74          movb    $0x74, -0xcc(%rbp)    ;  t
000000010001a46a        c68535ffffff6c          movb    $0x6c, -0xcb(%rbp)    ;  l
000000010001a471        c68536ffffff65          movb    $0x65, -0xca(%rbp)    ;  e
000000010001a486        c68510ffffff41          movb    $0x41, -0xf0(%rbp)    ;  A
000000010001a48d        c68511ffffff58          movb    $0x58, -0xef(%rbp)    ;  X
000000010001a494        c68512ffffff46          movb    $0x46, -0xee(%rbp)    ;  F
000000010001a49b        c68513ffffff6f          movb    $0x6f, -0xed(%rbp)    ;  o
000000010001a4a2        c68514ffffff63          movb    $0x63, -0xec(%rbp)    ;  c
000000010001a4a9        c68515ffffff75          movb    $0x75, -0xeb(%rbp)    ;  u
000000010001a4b0        c68516ffffff73          movb    $0x73, -0xea(%rbp)    ;  s
000000010001a4b7        c68517ffffff65          movb    $0x65, -0xe9(%rbp)    ;  e
000000010001a4be        c68518ffffff64          movb    $0x64, -0xe8(%rbp)    ;  d
000000010001a4c5        c68519ffffff57          movb    $0x57, -0xe7(%rbp)    ;  W
000000010001a4cc        c6851affffff69          movb    $0x69, -0xe6(%rbp)    ;  i
000000010001a4d3        c6851bffffff6e          movb    $0x6e, -0xe5(%rbp)    ;  n
000000010001a4da        c6851cffffff64          movb    $0x64, -0xe4(%rbp)    ;  d
000000010001a4e1        c6851dffffff6f          movb    $0x6f, -0xe3(%rbp)    ;  o
000000010001a4e8        c6851effffff77          movb    $0x77, -0xe2(%rbp)    ;  w
000000010001a863        c685c0feffff55          movb    $0x55, -0x140(%rbp)   ;   U
000000010001a86a        c685c1feffff54          movb    $0x54, -0x13f(%rbp)   ;   T
```

```
0x10001d64e      e83d40feff     call sym.func.100001690
;-- rip:
0x10001d653      c685cdfeffff.  mov byte [var_133h], 0xd   ; 13
0x10001d65a      c685cefeffff.  mov byte [var_132h], 0xa
0x10001d661      c685cffeffff.  mov byte [var_131h], 0
0x10001d668      c685c8feffff.  mov byte [var_138h], 0x67  ; 'g' ; 103
0x10001d66f      c685c9feffff.  mov byte [var_137h], 0x75  ; 'u' ; 117
0x10001d676      c685cafeffff.  mov byte [var_136h], 0x69  ; 'i' ; 105
0x10001d67d      c685cbfeffff.  mov byte [var_135h], 0x64  ; 'd' ; 100
0x10001d684      c685ccfeffff.  mov byte [var_134h], 0
0x10001d68b      c685c0feffff.  mov byte [var_140h], 0xd   ; 13
0x10001d692      c685c1feffff.  mov byte [var_13fh], 0xa
0x10001d699      c685c2feffff.  mov byte [var_13eh], 0x55  ; 'U' ; 85
0x10001d6a0      c685c3feffff.  mov byte [var_13dh], 0x52  ; 'R' ; 82
0x10001d6a7      c685c4feffff.  mov byte [var_13ch], 0x4c  ; 'L' ; 76
0x10001d6ae      c685c5feffff.  mov byte [var_13bh], 0x3a  ; ':' ; 58
0x10001d6b5      c685c6feffff.  mov byte [var_13ah], 0x20  ; mach0_cmd_0
0x10001d6bc      c685c7feffff.  mov byte [var_139h], 0
0x10001d6c3      c685b8feffff.  mov byte [var_148h], 0x46  ; 'F' ; 70
0x10001d6ca      c685b9feffff.  mov byte [var_147h], 0x69  ; 'i' ; 105
0x10001d6d1      c685bafeffff.  mov byte [var_146h], 0x72  ; 'r' ; 114
0x10001d6d8      c685bbfeffff.  mov byte [var_145h], 0x65  ; 'e' ; 101
0x10001d6df      c685bcfeffff.  mov byte [var_144h], 0x66  ; 'f' ; 102
0x10001d6e6      c685bdfeffff.  mov byte [var_143h], 0x6f  ; 'o' ; 111
0x10001d6ed      c685befeffff.  mov byte [var_142h], 0x78  ; 'x' ; 120
```

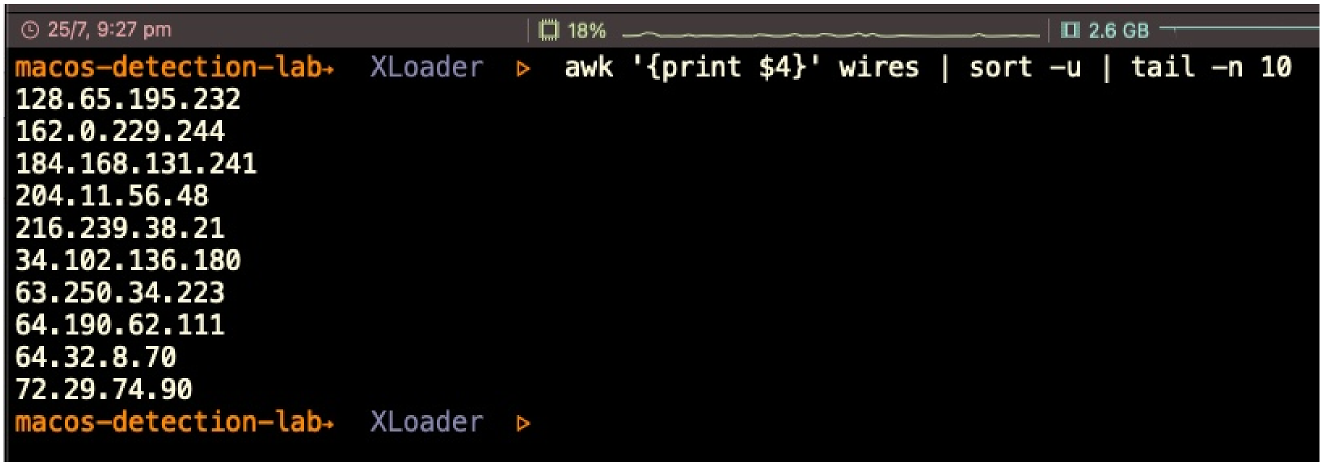
```
phils$ otool -tvj kIbwf02l | grep movb | grep \(%rbp\)$ | awk '{ print(NF==7)?"0x0a"$(NF-1):$(N
F-1) }' | sed 's/\$//g' | grep -v % | sed 's/{0x,\,}//g' | grep -v '-' | awk 'length($0)>1' | a
wk 'length($0)!=3' | xxd -r -p
10.1210.10.:1.1OS X XLNG:HSU.appMacOSContentsInfo.plist80987dat=&=&un=&br=&os=1passtokenemaillo
ginsigninaccountHost: &GETPUTPOSTOPTIONSGET NSStringstringWithCString:encoding:.appUTF8StringNS
WorkspacesharedWorkspaceprocessIdentifierfrontmostApplicationAXTitleAXFocusedWindowUTF8StringNS
PasteboardstringForType:generalPasteboardpublic.utf8-plain-textrm -rf open .exe.dllrm rm unzip
nss3.zip -d 200 OKr%s\DB1ChromeURL: saltysalt Recoveryr%s <<<  2>/dev/nullrm rm guidURL: Firefo
x/logins.json%
 ~/Downloads/XLoader [02:32:32]
phils$
```

```
10.1
2
10.10.:1.1OS X XLNG:HSU
.appMacOSContentsInfo.plist
8098
7
dat=&=&un=&br=&os=1pass
token
email
login
signin
account
Host:
&
GET
PUT
POST
OPTIONS
GET
NSStringstringWithCString:encoding:.appUTF8StringNSWorkspacesharedWorkspaceprocessIdentifie
rfrontmostApplicationAXTitleAXFocusedWindowUTF8StringNSPasteboardstringForType:generalPaste
boardpublic.utf8-plain-textrm -rf open .exe.dllrm rm unzip nss3.zip -d 200 OK
r
%s
\DB1
ChromeURL:
saltysalt Recovery
r
%s <<<  2>/dev/nullrm rm
guid
URL:
Firefox
/logins.json
~
~
~
~
~
```

Stack strings found in XLoader's macOS version

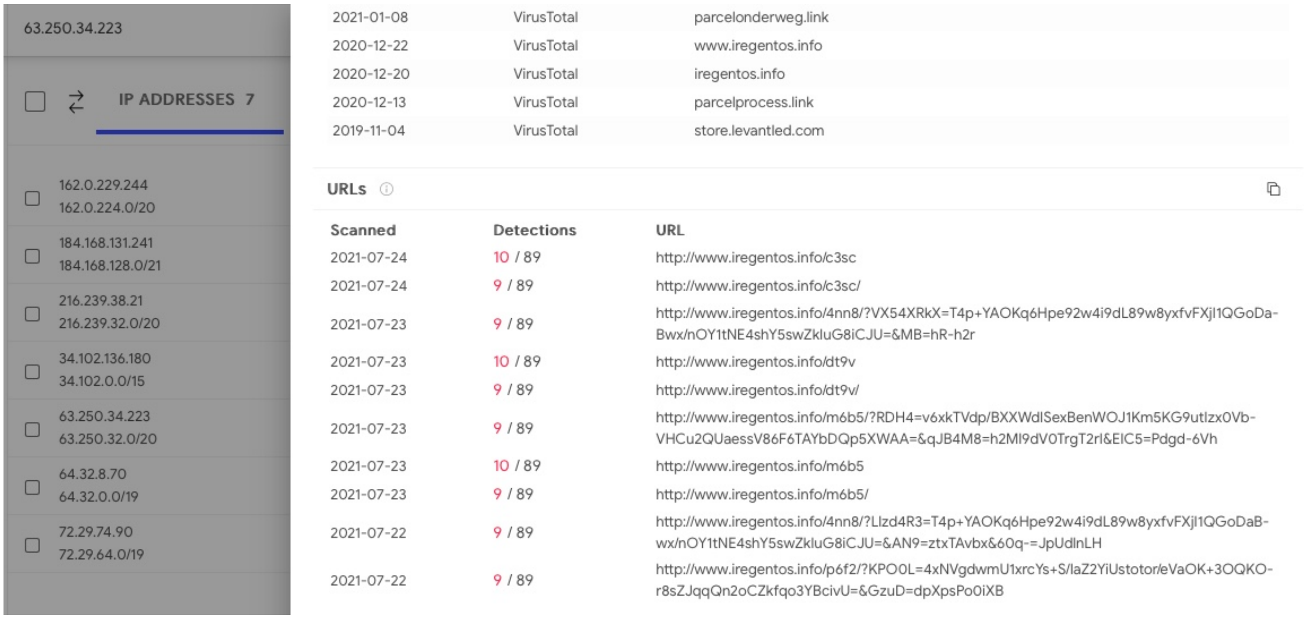
The strings here show that XLoader attempts to steal credentials from Chrome and Firefox browsers. We also see an indication that the malware calls the `NSWorkspace` API to identify the front window via the Accessibility API `AXTitleFocusedWindow` and leverages `NSPasteboard`, likely to copy information from the window of the user's currently active process. Calling Accessibility APIs requires user consent as this functionality is controlled by TCC. As noted above, the JavaLauncher has such permissions.

Other researchers have suggested that XLoader's internet traffic is laden with decoys to disguise the actual C2 used to transmit data. As we did not observe any credential stealing traffic in our test, we cannot confirm that suspicion, but XLoader's internet traffic is certainly 'noisy'. We observed the malware reaching out to a variety of known phishing and malware sites.



```
macos-detection-lab  XLoader  ▷  awk '{print $4}' wires | sort -u | tail -n 10
128.65.195.232
162.0.229.244
184.168.131.241
204.11.56.48
216.239.38.21
34.102.136.180
63.250.34.223
64.190.62.111
64.32.8.70
72.29.74.90
macos-detection-lab  XLoader  ▷
```

Some of the IP addresses contacted by the XLoader malware



63.250.34.223

□ ⇄ IP ADDRESSES 7

□	162.0.229.244 162.0.224.0/20
□	184.168.131.241 184.168.128.0/21
□	216.239.38.21 216.239.32.0/20
□	34.102.136.180 34.102.0.0/15
□	63.250.34.223 63.250.32.0/20
□	64.32.8.70 64.32.0.0/19
□	72.29.74.90 72.29.64.0/19

2021-01-08	VirusTotal	parcelonderweg.link
2020-12-22	VirusTotal	www.iregentos.info
2020-12-20	VirusTotal	iregentos.info
2020-12-13	VirusTotal	parcelprocess.link
2019-11-04	VirusTotal	store.levantled.com

URLs ⓘ

Scanned	Detections	URL
2021-07-24	10 / 89	http://www.iregentos.info/c3sc
2021-07-24	9 / 89	http://www.iregentos.info/c3sc/
2021-07-23	9 / 89	http://www.iregentos.info/4nn8/?VX54XRkX=T4p+YAOKq6Hpe92w4i9dL89w8yxfvFXjl1QGoDa-Bwx/nOY1tNE4shY5swZkluG8iCJU=&MB=hR-h2r
2021-07-23	10 / 89	http://www.iregentos.info/dt9v
2021-07-23	9 / 89	http://www.iregentos.info/dt9v/
2021-07-23	9 / 89	http://www.iregentos.info/m6b5/?RDH4=v6xkTVdp/BXXWdlSexBenWOJ1Km5KG9utlzx0Vb-VHCu2QUaessV86F6TAYbDQp5XWAA=&qJB4M8=h2Ml9dV0TrgT2rl&EIC5=Pdgd-6Vh
2021-07-23	10 / 89	http://www.iregentos.info/m6b5
2021-07-23	9 / 89	http://www.iregentos.info/m6b5/
2021-07-22	9 / 89	http://www.iregentos.info/4nn8/?Llzd4R3=T4p+YAOKq6Hpe92w4i9dL89w8yxfvFXjl1QGoDaB-wx/nOY1tNE4shY5swZkluG8iCJU=&AN9=ztxTAvbx&60q-=JpUdlnLH
2021-07-22	9 / 89	http://www.iregentos.info/p6f2/?KPO0L=4xNVgdwmU1xrcYs+S/laZ2YiUstotor/eVaOK+3OQKO-r8sZJqqQn2oCZkfqo3YBcivU=&GzuD=dpXpsPo0iXB

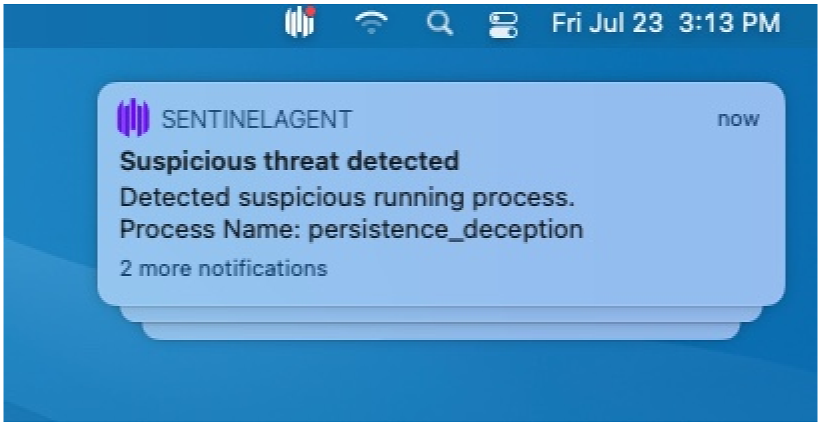
One of a number of malicious domains XLoader contacts (VirusTotal)

Detecting XLoader Infostealer on macOS

At the end of this post we provide a number of macOS-specific Indicators of Compromise to help organizations and users in general identify an XLoader infection. SentinelOne customers are protected against this malware automatically, regardless of whether it is executed via the Java Runtime Environment or by the standalone XLoader Mach-O.

In our test, we set the agent to ‘Detect-only’ policy in order to observe the malware’s behaviour. Customers are advised to always use the ‘Protect’ policy which prevents execution of malware entirely.

In ‘Detect-only’ mode, the target’s Mac device will immediately alert the user via Notifications:



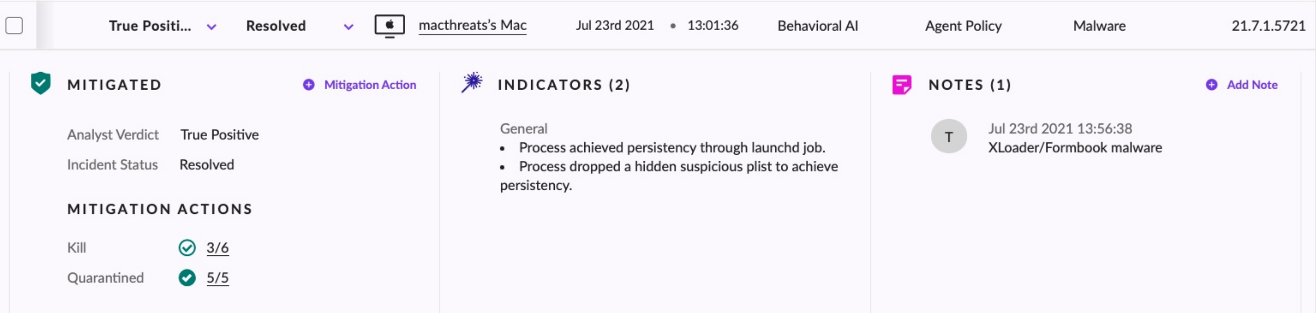
SENTINELAGENT now

Suspicious threat detected

Detected suspicious running process.
Process Name: persistence_deception

2 more notifications

Security teams and IT administrators, meanwhile, would see something similar to the following in the Management console.



True Positi... Resolved macthreats's Mac Jul 23rd 2021 • 13:01:36 Behavioral AI Agent Policy Malware 21.7.1.5721

MITIGATED

Analyst Verdict True Positive
Incident Status Resolved

MITIGATION ACTIONS

Kill ✓ 3/6
Quarantined ✓ 5/5

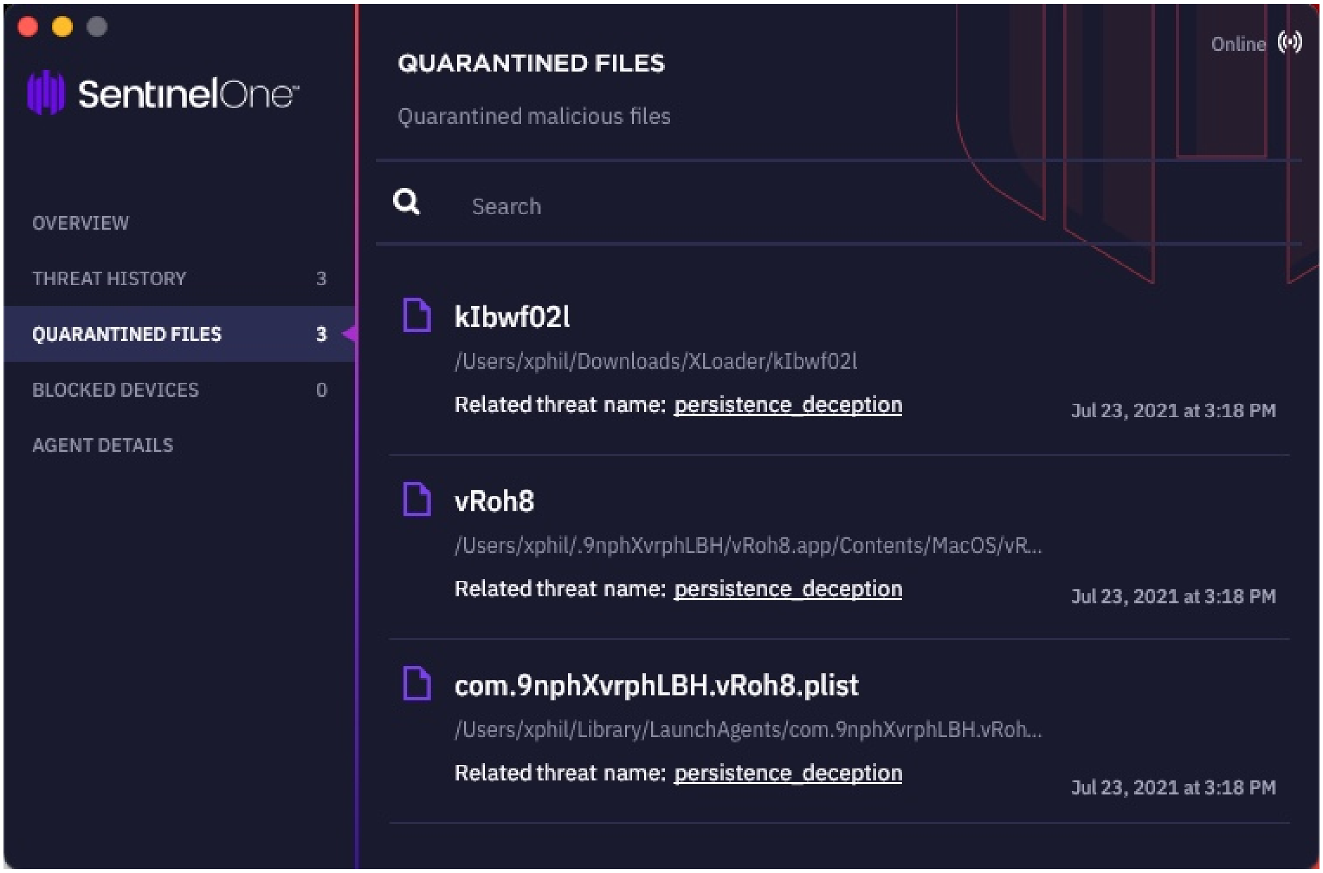
INDICATORS (2)

General
- Process achieved persistency through launchd job.
- Process dropped a hidden suspicious plist to achieve persistency.

NOTES (1)

T Jul 23rd 2021 13:56:38
XLoader/Formbook malware

After remediation, the UI (version 21.7EA) on the device indicates that the threat has been successfully killed and quarantined.



QUARANTINED FILES

Quarantined malicious files

kIbwf02l
/Users/xphil/Downloads/XLoader/kIbwf02l
Related threat name: persistence_deception Jul 23, 2021 at 3:18 PM

vRoh8
/Users/xphil/.9nphXvrphLBH/vRoh8.app/Contents/MacOS/vR...
Related threat name: persistence_deception Jul 23, 2021 at 3:18 PM

com.9nphXvrphLBH.vRoh8.plist
/Users/xphil/Library/LaunchAgents/com.9nphXvrphLBH.vRoh...
Related threat name: persistence_deception Jul 23, 2021 at 3:18 PM

Conclusion

XLoader is an interesting and somewhat unusual example in the macOS malware world. It's dependency on Java and its functionality suggests it is primarily targeting organizations where the threat actors expect Java applications to be in use. Among other things, that includes certain online banking applications, and the attractiveness from a criminal's perspective of a keylogger and info stealer in that environment can certainly be understood. It is also worth noting that the malware's minimum system requirement is 10.6 Snow Leopard (over 10 years old), so the author's are certainly casting their net wide. On the other hand, the implementation on macOS is clumsy at best and is likely to raise suspicions. No doubt the malware authors will be looking to improve on this in future iterations.

Indicators of Compromise

SHA1 Hashes

XLoader Mach-O Executable: KIbwf02l
7edead477048b47d2ac3abdc4baef12579c3c348

Suspected Phishing lure attachment: Statement SKBMT 09818.jar
b8c0167341d3639eb1ed2636a56c272dc66546fa

Example Persistence LaunchAgent: com.j85H64iPLnW.rXxHYP
cb3e7ac4e2e83335421f8bbc0cf953cb820e2e27

Contacted IPs

128.65.195.232
162.0.229.244
184.168.131.241
204.11.56.48
216.239.38.21
34.102.136.180
63.250.34.223
64.190.62.111
64.32.8.70
72.29.74.90

Interesting Strings

```
.appMacOSContentsInfo.plist
.exe.dll
/logins.json
10.:1.1OS X XLNG:
200 OK
80987dat=&=&un=&br=&os=1
DB1ChromeURL:
guidURL: Firefox
NSStringstringWithCString:encoding:
open
passtokenemailloginsigninaccountHost: &GETPUTPOSTOPTIONSGET
r%s <</dev/null
Recovery
rm -rf
rm unzip nss3.zip -d
saltysalt
UTF8StringNSPasteboardstringForType:generalPasteboardpublic.utf8-plain-text
UTF8StringNSWorkspacesharedWorkspaceprocessIdentifierfrontmostApplicationAXTitleAXFocu
```