Burned by Fire(fox)

Archived: 2026-04-05 19:12:14 UTC

Burned by Fire(fox)

part i: a firefox 0day drops a macOS backdoor (osx.netwire.a)

June 20, 2019

Our research, tools, and writing, are supported by "Friends of Objective-See"
Today's blog post is brought to you by:

📝 👾 Want to play along?

I've shared the OSX.NetWire.A sample (password: infect3d)

…please don't infect yourself!

Background

A little over a week ago, I received an email from a user who stated:

> "Last week Wednesday I was hit with an as-yet-unknown Firefox 0day that somehow dropped a binary and executed it on my mac (10.14.5)
>
> Let me know if you would be interested in analysing the binary, might be something interesting in there wrt bypassing osx gatekeeper."

Of course I was intrigued! …though at the time, was wrapping up our "Objective by the Sea" v2.0 (which you can read all about here).

Now that the dust has cleared from the conference I wanted to dig into this attack, and analyze the persistent malware.

A Firefox 0day

When the user contacted me, there wasn't much information about the Firefox 0day exploit used in the attack. However now, more information is readily available!

First, I was able to obtain an email that (said user claimed) was related to the attack.

```
 Dear XXX,

My name is Neil Morris. I'm one of the Adams Prize Organizers.

Each year we update the team of independent specialists who could assess
the quality of the competing projects: http://people.ds.cam.ac.uk/nm603/awards/Adams_Prize

Our colleagues have recommended you as an experienced specialist in this
field.

We need your assistance in evaluating several projects for Adams Prize.

Looking forward to receiving your reply.

Best regards,
Neil Morris
```

Even if an attacker has a browser 0day exploit, they still have find a way to deliver it to the target.

When individuals are targeted, the delivery mechanism of choice is often an email that contains links to a malicious site (which will "throw" the exploit when the user visits said site).

Unfortunately the link (`people.ds.cam.ac.uk/nm603/awards/Adams_Prize`) currently returns a `404 Not Found` :

```
$ curl http://people.ds.cam.ac.uk/nm603/awards/Adams_Prize
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /nm603/awards/Adams_Prize was not found on this server.</p>
<hr>
<address>Apache/2.4.7 (Ubuntu) Server at people.ds.cam.ac.uk Port 80</address>
</body></html>
```

Of course it's possible that the site will only serve up ("throw") the exploit if the user browses to the site via a vulnerable version of Firefox, or perhaps from a certain IP address, etc. More than likely though, the attacker has moved on, and taken down the exploit site.

Though I don't have access to the exploit code, the user was able to provide me with persistent malware the exploit installed on the system (named `Finder.app`). Woohoo!

The remainder of this blog will cover the analysis of this malware (hash: `23017a55b3d25a2597b7148214fd8fb2372591a5`).

```
$ shasum -a 1 ~/Attack/Finder.app/Contents/MacOS/Finder
23017a55b3d25a2597b7148214fd8fb2372591a5  Finder
```

Interestingly, today a security researcher at Coinbase, Philip Martin, posted an interesting thread on twitter:

Note that the hash mentioned by Phil, `23017a55b3d25a2597b7148214fd8fb2372591a5` matches malicious file which the user sent me. Moreover the user confirmed that he was "involved with a cryptocurrency exchange until fairly recently." Thus it seems reasonable to assume we're all talking about the same Firefox 0day.

This 0day, has now been patched as CVE-2019-11707, and covered in various articles such as:

- "Mozilla patches Firefox zero-day abused in the wild"
- "Mozilla Patches Firefox Critical Flaw Under Active Attack"

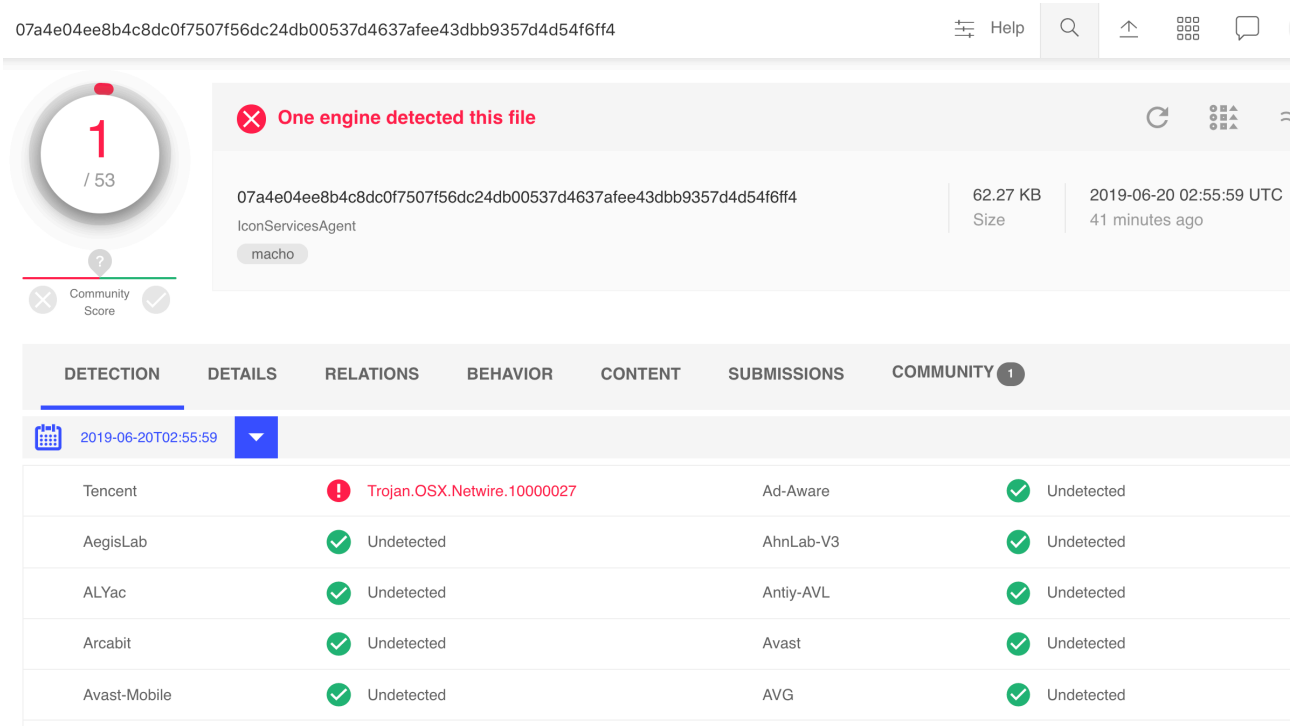
However, the details of the persistent malware used in the attack are scant (non-existent?), so let's dive into that now!

A Persistent Mac Backdoor (`OSX.NetWire.A` !?)

As noted, the infected user was kind enough to send me the malware (`Finder.app`) that the attacker persistently installed on the system (via the Firefox 0day).

Via my open-source "WhatsYourSign" utility we can quickly see it's an unsigned application:

Searching for the hash (`23017A55B3D25A2597B7148214FD8FB2372591A5`) on VirusTotal found a exact match and shows that the file was submitted on `2019-06-06` but currently is only detected by one AV engine (Tencent):



The full application bundle, Finder.app, was just submitted to VirusTotal today.

It is similarly only detected by one AV engine.

Interestingly the engine detecting the malware flags it as `OSX.Netwire`.

`OSX.Netwire` (or `OSX.Wirenet`) was first discovered in 2012(!) by Dr Web. In their writeup "Wirenet: The Password-Stealing Trojan Lands on Linux and OS X" they state that it is "the first Trojan in history to steal Linux and Mac OS X passwords."

Passwords were stolen via keylogger logic and/or directly from files on disk (i.e. saved browser logins):

```
$ strings malware/2012/OSX.Netwire

%s/Library/Opera/wand.dat
%s/.Library/Opera/wand.dat

SeaMonkey
Thunderbird

%s/Library/Application Support/Firefox

%s/signons.sqlite

NSS_Init
PK11_GetInternalKeySlot
PK11_Authenticate
NSSBase64_DecodeBuffer

select *  from moz_logins
```

But that was `OSX.Netwire` from 2012. Is this new sample really (still) `OSX.Netwire`!? I personally had not heard anything about `OSX.Netwire` since 2012, so decided to poke around.

First, via simple "string" matching, it's easy to confirm the 2012 sample and the 2019 are in someway related.

For example, note the string: `"\x03\x04\x15\x1A\r\nexit\r\n\r\nexit\n\n"` in both the 2012 sample:

```
 esi = "/bin/sh";
if(access(esi) != 0x0) {
   esi = "/bin/bash";
}

...

eax = write$UNIX2003(*0x140d0, "\x03\x04\x15\x1A\r\nexit\r\n\r\nexit\n\n", 0x15);
```

and in the 2019 sample:

```
 if(stat("/bin/sh", edi) != 0x0) {
   ebp = "/bin/bash";
}

...

write$UNIX2003(ebx, "\x03\x04\x15\x1A\r\nexit\r\n\r\nexit\n\n", 0x15)
```

…other rather unique strings are also found in both samples.

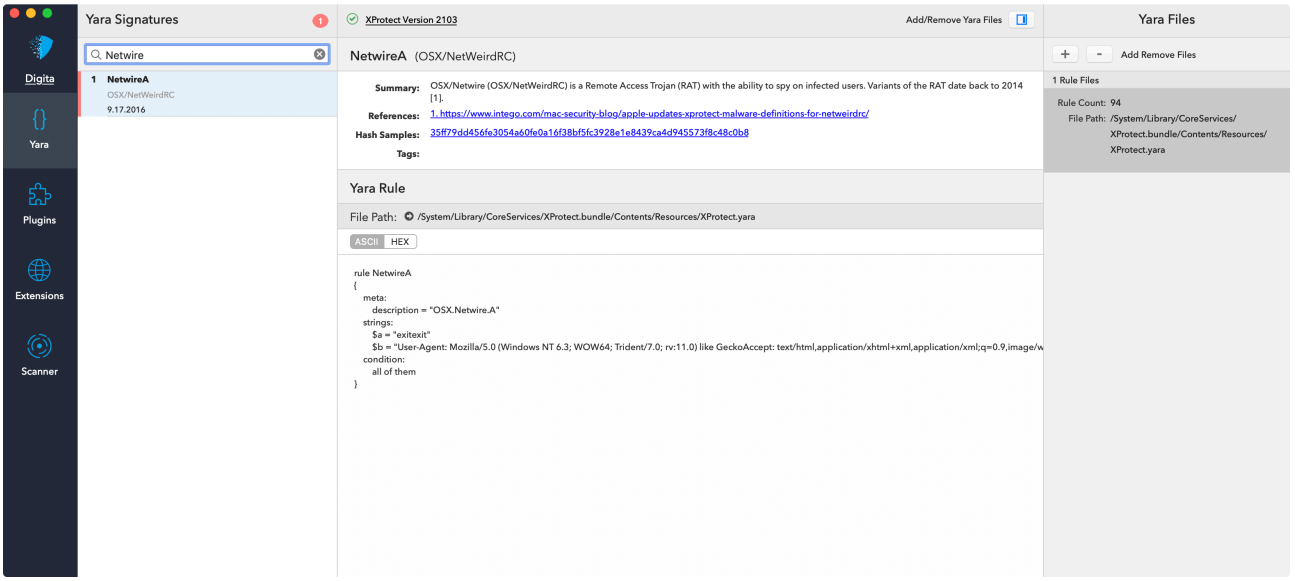
Whilst chatting about this with one my close (security researcher) friends he noted that this new sample was (already) detected by XProtect.

I was rather skeptical of this claim (as I didn't recall any recent XProtect updates for `OSX.Netwire`), but turns out he was absolutely right!

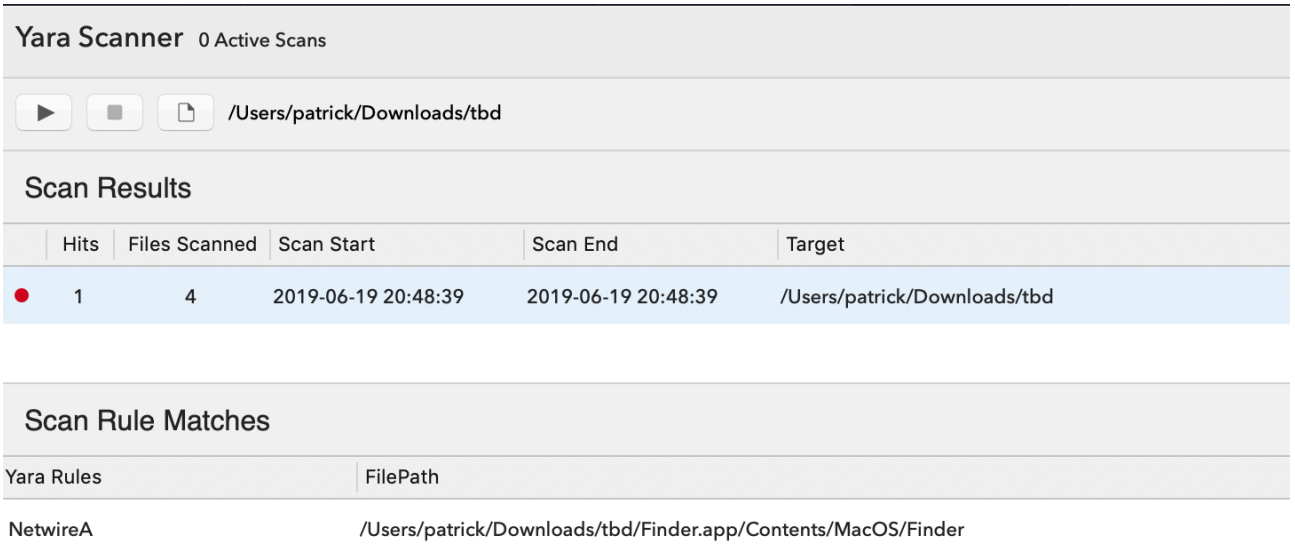
In 2016, Apple added a Yara signature to detect something called `OSX.Netwire.A`. The signature (found in `/System/Library/CoreServices/XProtect.bundle/Contents/Resources/XProtect.yara`) is presented below:

```
 rule NetwireA
{
    meta:
        description = "OSX.Netwire.A"
    strings:
        $a = "exitexit"
        $b = "User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like GeckoAccept: text/html,
    condition:
        all of them
}
```

Digita's UXProtect utility can graphically display this signature as well:



We can also utilize [UXProtect](#) to confirm this signature (from 2016) still detects the malware used in the Firefox 0day attack:



Wow kudos to Apple for writing a signature that (IMHO) rather surprisingly detects still detects this “new” threat!

Interestingly Apple’s signature does *not* detect the sample from 2012 (as it does not contain the `User-Agent: Mozilla...` string). This is first (of many) indicator that these samples, while somehow related are unsurprisingly not the same.

While the sample from 2012 and 2019 are clearly related, they are also very different. We’ll get into this more in part two of this blog post, when we dive into the capabilities of the (new) malware sample. However in short, the 2012 and 2019 samples have totally different objectives. If I had to guess, they are both written by the same author (or team), but serve unique purposes (i.e. the 2012 sample is only concerned with stealing passwords).

Before we wrap up part one of this post, let’s look at how the new sample, `OSX.NetWire.A`, persists.

A quick peek at the malware's disassembly reveals an embedded launch agent plist:

```
 memcpy(esi, "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<!DOCTYPE plist PUBLIC \"-//Apple Computer//DTD PLIST

...
eax = getenv("HOME");
eax = __snprintf_chk(&var_6014, 0x400, 0x0, 0x400, "%s/Library/LaunchAgents/", eax);

...
eax = __snprintf_chk(edi, 0x400, 0x0, 0x400, "%s%s.plist", &var_6014, 0xe5d6);
```

Seems reasonable to assume the malware will persist as launch agent.

However, it also appears to contain logic to persist as a login item (note the call to the `LSSharedFileListInsertItemURL` API):

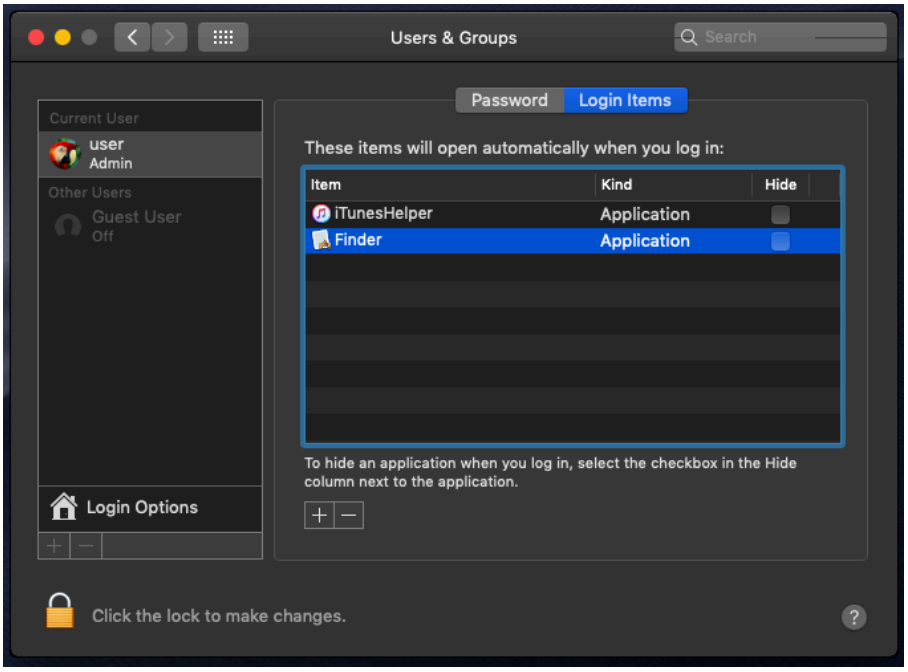
```
 eax = __snprintf_chk(&var_6014, 0x400, 0x0, 0x400, "%s%s.app", &var_748C, &var_788C);
eax = CFURLCreateFromFileSystemRepresentation(0x0, &var_6014, eax, 0x1);
...

eax = LSSharedFileListCreate(0x0, **_kLSSharedFileListSessionLoginItems, 0x0);

...
eax = LSSharedFileListInsertItemURL(eax, **_kLSSharedFileListItemLast, 0x0, 0x0, edi, 0x0, 0x0);
```

Hopping into a VM and running the malware, turns out it persists twice! First as launch agent (`com.mac.host.plist`), and then as a login item.

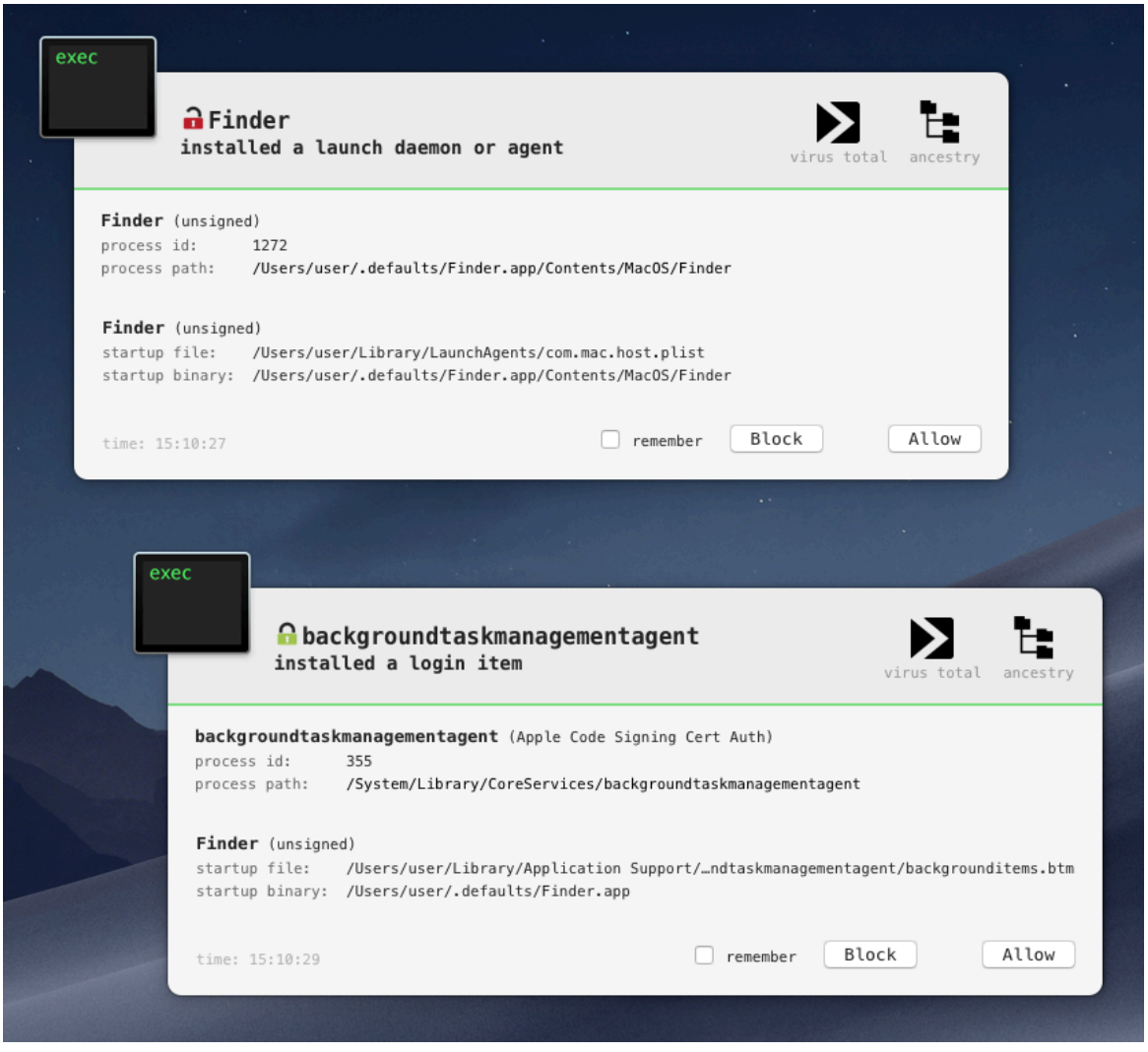
```
$ cat ~/Library/LaunchAgents/com.mac.host.plist
{
    KeepAlive = 0;
    Label = "com.mac.host";
    ProgramArguments =     (
        "/Users/user/.defaults/Finder.app/Contents/MacOS/Finder"
    );
    RunAtLoad = 1;
}
```

As the launch agent (`com.mac.host.plist`) has the `RunAtLoad` key set (to `1`), the OS will automatically launch the specified binary, `.defaults/Finder.app/Contents/MacOS/Finder` each time the user logs in.

The login item will also ensure the malware is launched. Login items however show up in the UI, clearly detracting from the malware's stealth:



Is persisting twice better than once? Not really, especially if you are running Objective-See's lovely tools such as [BlockBlock](https://objective-see.com/products/blockblock.html) which detects both persistence attempts:



For details on persisting as a login item (and the role of backgroundTaskManagementAgent), see my recent blog post: "Block Blocking Login Items"

KnockKnock can also reveal the infection (after the fact), by detecting the malware's (2x) persistence:

Maybe the malware author wanted to be extra extra sure about gaining and/or maintaining persistence!?

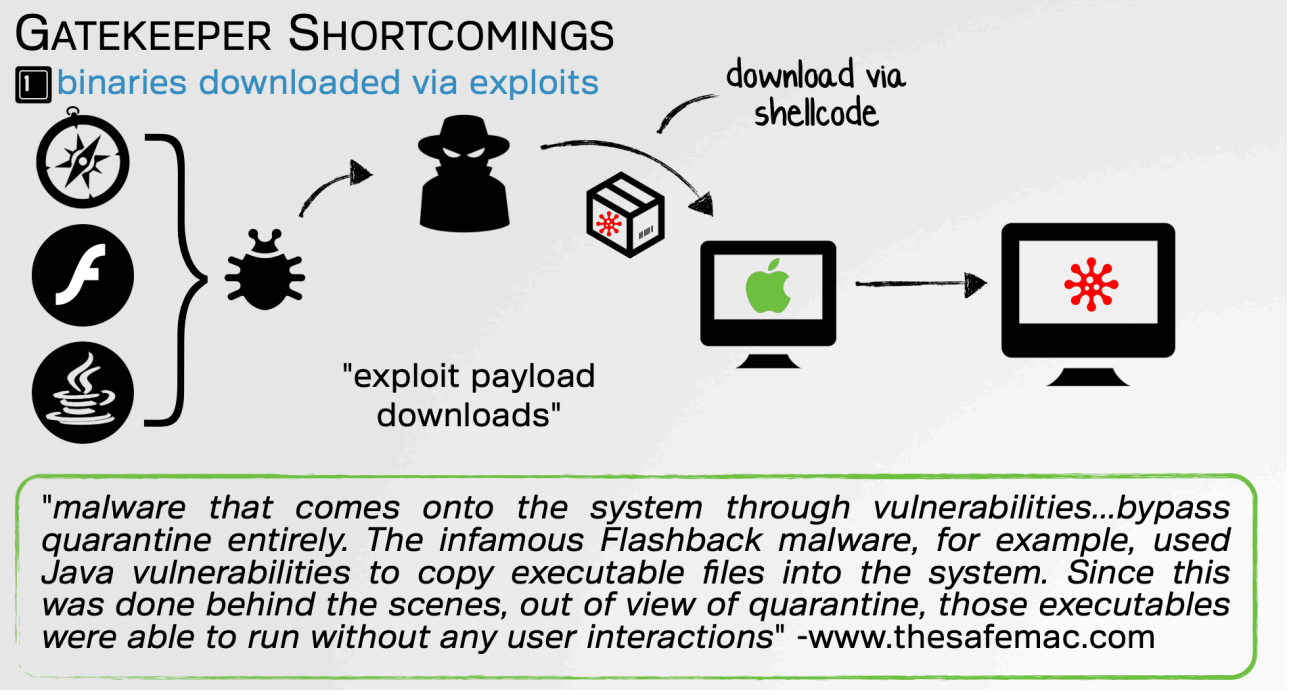
Conclusion

In today's post, we discussed the persistent payload of a rather sophisticated targeted attack again cryptocurrency exchange(s). Via a Firefox 0day, the attackers persistently deployed a macOS binary.

Rather interestingly this malware was `OSX.NetWire.A`, which bares some relation to a 2012 specimen of the same name. Similarly intriguingly we showed how a signature deployed by Apple in 2016 would detects this malware. Or would it?

Recall that in his original email, the infected user noted that the malware bypassed Gatekeeper. This is actually unsurprising as the malware was delivered by a remote 0day exploit. Gatekeeper only scans applications that have a quarantine attribute set. This is added by the application (i.e. browser) or OS only when the application is downloaded via normal means (i.e. by the user). Exploit code that downloads a payload (such as malicious application) will not set a quarantine attribute (or can remove it), thus will not trigger Gatekeeper!

I highlighted this (well known) fact in a 2016 presentation, "Gatekeeper Exposed; Come, See, Conquer":



XProtect similarly only operates on files that have the quarantine bit set.

However, this may be changing in macOS 10.15 (Catalina).

Thus, thanks to the infection vector (a Firefox 0day), neither Gatekeeper nor XProtect would protect the user. Clearly it’s never wise to leave security soley to Cupertino 😅
