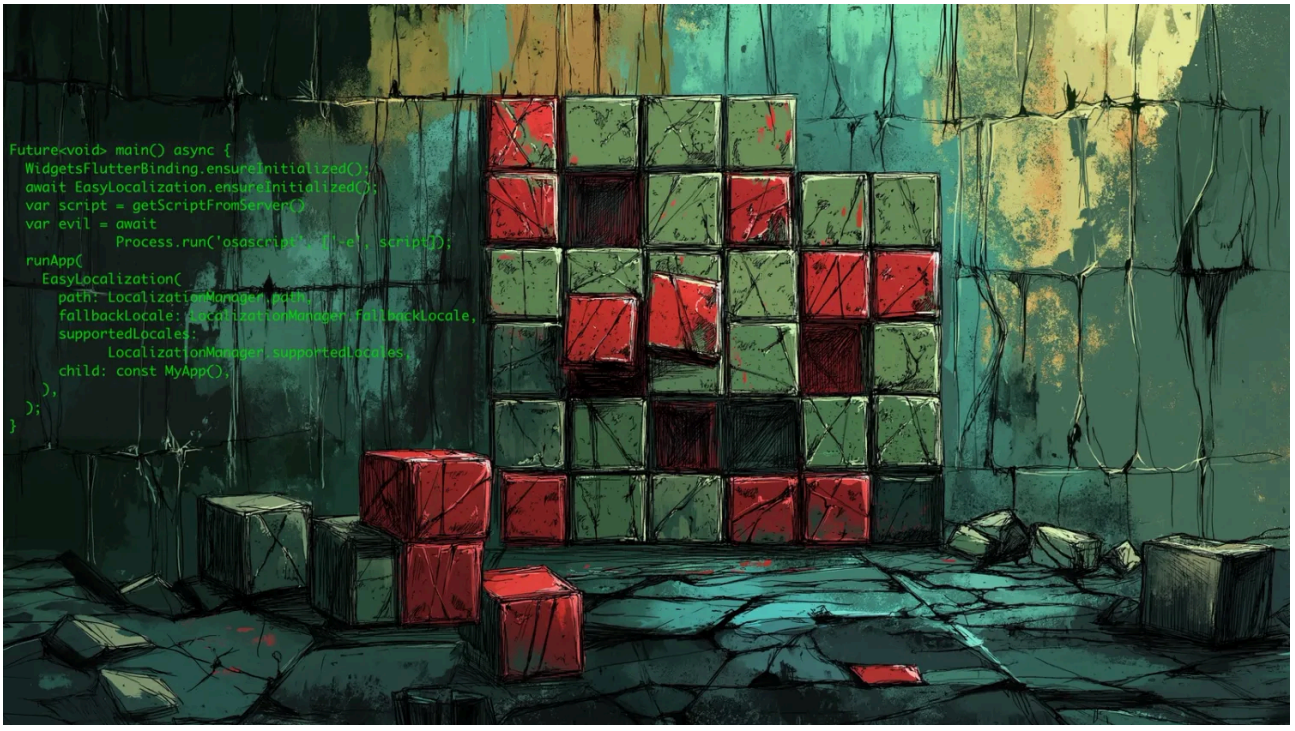
APT Actors Embed Malware within macOS Flutter Applications

By Jamf Threat Labs

Archived: 2026-04-05 20:16:52 UTC

Jamf Threat Labs discovered malware samples believed to be tied to the Democratic People's Republic of Korea (DPRK), aka North Korea, that are built using Flutter, which by design provides obfuscation to the malicious code. JTL performs a deep dive into how the malicious code works to help protect users on macOS devices.



By Ferdous Saljooki and Jaron Bradley

Introduction

In late October, Jamf Threat Labs discovered samples uploaded to VirusTotal that are reported as clean despite showing malicious intent. The domains and techniques in the malware align closely with those used in other DPRK malware and show signs that, at one point in time, the malware was signed and had even temporarily passed Apple’s notarization process. It’s unclear in this case if the malware has been used against any targets or if the attacker is preparing for a new form of delivery.

The Packaging

The discovered malware came in three forms. A Go variant, a Python variant built with Py2App and a Flutter-built application. This blog post will focus on the Flutter-built application as we find it the most interesting due to its complexity in reversing.

Flutter is a framework developed by Google that simplifies app design for cross-platform applications. If a developer is designing an app in which they want to look consistent across macOS, iOS and Android, Flutter is a viable option.

Applications built using Flutter lead to a uniquely designed app layout that provides a large amount of obscurity to the code. This is due to the fact that code written into the main app logic using the Dart programming language is contained within a dylib that is later loaded by the Flutter engine.

The image above shows the layout of a standard Flutter application with two notable files called out — a main Flutter application and a dylib file that gets assigned the name App. To make matters more confusing, this dylib is not directly loaded by the main application. Due to the complex nature in which Flutter compiles its applications, this dylib is not listed as a shared Library within the primary machO file. While there is nothing inherently malicious about this app architecture, it just happens to provide a good avenue of obfuscation by design.

The Malware

The Flutter applications that were created by the malware author are considered to be a stage one payload. We initially identified six infected applications, five of which were signed using a developer account signature. At the time of our discovery, Apple had already revoked these signatures.

One application, titled `New Updates in Crypto Exchange (2024-08-28).app` (7cb8a9db65009f780d4384d5eaba7a7a5d7197c4), was built using Flutter and developed with the Dart programming language. When executed, the victim is presented with a functional minesweeper game. The game itself appears to be a clone of a basic open-source Flutter game on GitHub which is a project designed for iOS. By cloning the project and modifying some project settings, it can easily be compiled to run on macOS.

Due to modifications made to the app, a network request is made to the domain `mbupdate[.]linkpc[.]net` upon starting the app. This caught our attention as this domain has been used by DPRK malware in the past.

Below is the GET request for the stage two malware over HTTPS.

Unfortunately, at the time of our analysis, the server was responding with a 404 error message.

As expected, due to the app architecture, the compiled Dart code makes it into the App dylib file located at the path `New Updates in Crypto Exchange (2024-08-28).app/Contents/Frameworks/App.framework/Versions/A/App` (a2cd8cf70629b5bb0ea62278be627e21645466a3).

As we see from the `nm` output below, the presence of snapshot-related symbols such as `_kDartVmSnapshotData` and `_kDartIsolateSnapshotInstructions` suggests that the application's operational logic is heavily embedded within precompiled Dart snapshots, complicating analysis and decompilation efforts.

Taking a closer look at strings, we can quickly determine some of the supported functionality. As expected, we see the domain and user-agent strings within the dylib but the presence of the `osascript` string is quite interesting as it likely indicates capabilities supporting AppleScript execution.

For testing purposes, we redirected traffic from the malicious domain within a local test environment and confirmed that the malware does indeed execute any AppleScript code returned by a valid HTTP response. Our testing showed the stage two AppleScript must be written backward in order to be successfully executed by the malware.

Below is an example of a dialog box message executed via a remote Applescript.

In the past, we have observed DPRK adapting to use native AppleScript payloads, so we suspect similar payloads may be leveraged by the attacker to compromise macOS systems.

Golang Variant

We identified a Golang variant of the malware with similar functionality, titled `New Era for Stablecoins and DeFi, CeFi (Protected).app` (0b9b61d0fffd52e6c37df37dfdffefc0e121acf7). Our friends at SentinelOne put out a recent blog post on an infection vector that uses this exact same file name, attributing it to the same threat actor.

As mentioned, this variant was previously signed and notarized by Apple but its signature has since been revoked.

Similar to the Flutter variant, the executable titled `Hello` (bc6b446bad7d76909d84e7948c369996b38966d1), makes a GET request to `hXXps://mbupdate[.]linkpc[.]net/update.php` using the user-agent `CustomUpdateUserAgent`.

It invokes `osascript` to run any AppleScript payload received in the server response.

Python Variant

The Python variant is packaged as a standalone application bundle using Py2App.

The app bundle titled `Runner.app` (ee22e7768e0f4673ab954b2dd542256749502e97) is signed ad-hoc and launches a functional Notepad application.

The boot script located at `Runner.app/Contents/Resources/__boot__.py` executes a Python script named `notepad_.py` (6f280413a40d41b8dc828250bbb8940b219940c5). This script leverages tkinter, a built-in Python library for creating GUI applications, for features like opening, editing and saving files.

However, embedded within this script is malicious logic that fetches and executes remote code. Similar to the Flutter variant, the init method sends a GET request to `hXXps://mbupdate[.]linkpc[.]net/update.php`. If a valid response is received, the content is passed to the `update()` method.

The `update()` method uses `osascript` to execute the server response as AppleScript, allowing the attacker to run arbitrary commands or payloads on the victim's system.

Conclusion

The malware discovered in this blog shows strong signs that it is likely testing for greater weaponization. This theory stems from the fact that the actor is known for putting together highly convincing social engineering campaigns from start to finish and the file names seen here do not align with the content displayed to the user

within the Flutter-built applications. This could perhaps be an attempt to see if a properly signed app with malicious code obscured within a dylib could get approved by Apple's notarization server, as well as slide under the radar of antivirus vendors.

It is not unheard of for actors to embed malware within a Flutter-based application, however, this is the first we've seen of this attacker using it to go after macOS devices. Though the question remains open if this was real malware or a test for a new way to weaponize malware, we remain vigilant in monitoring for further activity by the actor.

IOCs

Jamf Threat Labs has your six...

while Jamf solutions cover everything else!
