INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-08 18:47 UTC

AMOS Operators Abuse macOS applescript:// URL Scheme to Bypass ClickFix Protections via Script Editor

THREAT CAMPAIGN | HIGH | CVSS 7.5

SCC Item ID	SCC-CAM-2026-0158
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	7.5
Affected Products	macOS (all versions); macOS Tahoe 26.4 (partial mitigation bypassed); macOS Script Editor (built-in); browsers with autofill/cookie storage; cryptocurrency wallet browser extensions
Published	2026-04-08T14:55:43
Discovery Source	Rss

Executive Summary

Atomic Stealer (AMOS) operators have bypassed Apple's recent ClickFix defense in macOS Tahoe 26.4 by exploiting the built-in applescript:// URL scheme to deliver credential-stealing malware via Script Editor rather than Terminal. Organizations with macOS fleets are at risk of credential theft across Keychain, browsers, and cryptocurrency wallets, regardless of whether the Tahoe 26.4 patch is applied. The business risk is significant: stolen credentials and session cookies can enable account takeover, financial fraud, and lateral movement across enterprise environments.

Technical Analysis

AMOS operators adapted their ClickFix social-engineering chain after Apple's macOS Tahoe 26.4 update introduced a Terminal-execution warning targeting prior ClickFix variants. The new delivery method triggers the macOS applescript:// URL scheme (T1059.002), which opens Script Editor pre-loaded with attacker-controlled AppleScript, bypassing the Terminal warning entirely. Lure pages impersonate Apple-branded disk cleanup utilities to trick users into clicking (T1204.001, T1566). Once executed, a Mach-O AMOS binary deploys and harvests: macOS Keychain credentials (T1555.001), browser-saved passwords, cookies, autofill data, and saved payment cards (T1555.003, T1539), and cryptocurrency wallet browser extensions (T1056.003, T1056.004). The binary uses obfuscation and deobfuscation techniques to evade static analysis (T1027, T1140). No CVE has been assigned. Relevant CWEs: CWE-494 (download of code without integrity check), CWE-693 (protection mechanism failure), CWE-829 (inclusion of functionality from untrusted control sphere). All

macOS versions are affected; macOS Tahoe 26.4 provides only a partial mitigation that this campaign explicitly bypasses. No vendor patch addressing the applescript:// vector is confirmed available as of 2026-03-04.

Action Checklist

1. Step 1: Containment. Audit macOS fleet for any unauthorized Script Editor processes launched via URL scheme. Quarantine any host where Script Editor was opened unexpectedly, especially if followed by network outbound connections or new Mach-O binary execution. Disable or restrict the applescript:// URL scheme via MDM LaunchServices policy (deploy a Configuration Profile restricting URL scheme handlers) if your organization has no legitimate use for it.
2. Step 2: Detection. Query endpoint telemetry for Script Editor (com.apple.ScriptEditor2) launches triggered by a browser process or URL handler rather than direct user invocation. Look for: (a) osquery or EDR events showing Script Editor as a child of a browser process; (b) new Mach-O binaries written to /tmp, ~/Library/Application Support, or user-writable paths immediately after a Script Editor session; (c) AppleScript execution followed by curl, osascript, or network connections to non-Apple infrastructure; (d) Keychain access events (security.framework prompts) outside normal application context. Review browser history on affected hosts for Apple-branded disk cleanup or utility lure domains.
3. Step 3: Eradication. No Apple patch addresses the applescript:// bypass as of 2026-03-04. Immediate mitigations: (a) block the applescript:// URL handler via MDM (use a LaunchServices Configuration Profile restricting URL scheme handlers); (b) deploy a web content filter or DNS block for known lure domains as IOCs are published by threat intelligence sources; (c) on confirmed-compromised hosts, remove any unknown Mach-O binaries from /tmp and user Library paths; (d) revoke and rotate any credentials stored in Keychain or browser password managers on affected systems.
4. Step 4: Recovery. After containment and eradication, validate: (a) Script Editor can no longer be triggered via applescript:// from a browser using a controlled test in an isolated VM; (b) no unknown processes persist in user LaunchAgents or LaunchDaemons directories (T1543); (c) force password rotation for all accounts whose credentials may have been stored in Keychain or browser autofill on affected hosts; (d) invalidate active browser session cookies for corporate SaaS applications on affected accounts; (e) monitor affected user accounts for anomalous login activity for a minimum of 30 days post-remediation.
5. Step 5: Post-Incident. This campaign reveals a control gap: URL scheme handlers on macOS represent an under-monitored execution path that bypasses many endpoint controls focused on Terminal and shell activity. Review your macOS MDM baseline to include URL scheme handler restrictions as a standard control. Add Script Editor launch-via-URL-handler to your macOS detection ruleset permanently. Evaluate whether cryptocurrency wallet extensions are permitted on corporate-managed browsers; if not business-critical, consider blocking via browser policy. Update security awareness training to include social-engineering lures mimicking Apple-branded utilities.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate to CISO and legal counsel immediately if forensic analysis of Keychain access logs or browser extension storage confirms exfiltration of credentials for financial accounts, corporate SSO, or cryptocurrency wallets — as this triggers breach notification assessment under applicable state privacy laws (e.g., CCPA, NY SHIELD Act) and may constitute a reportable incident under PCI-DSS if payment card credentials were stored in affected browser autofill profiles.
Recovery Notes	Post-containment recovery must prioritize credential invalidation in the order of blast radius: corporate SSO and VPN credentials first (enable forced re-authentication via IdP), followed by browser-saved passwords and session cookies for SaaS applications, and finally cryptocurrency wallet seed phrase exposure assessment. Validate that the applescript:// MDM restriction is confirmed deployed via MDM compliance reporting before returning any quarantined host to production use. Monitor affected accounts in IdP and SaaS audit logs for a minimum of 30 days post-remediation, with automated alerting on impossible travel, new device enrollment, and MFA method changes, as AMOS-stolen session cookies may have been replayed by threat actors before invalidation was enforced.
Forensic Artifacts	macOS Unified Log archive (logarchive) covering the 4-hour window around the suspected applescript:// invocation — specifically filtered for `process == 'Script Editor'`, `process == 'osascript'`, and `subsystem == 'com.apple.security'` to capture the complete execution chain from URL handler invocation through Keychain access | LaunchServices database at `/private/var/db/com.apple.LaunchServices/com.apple.launchservices.secure.plist` — records the applescript:// URL scheme handler registration state at the time of compromise, confirming whether Script Editor or a substituted handler was invoked by the browser | TCC (Transparency, Consent, and Control) database at `/Library/Application Support/com.apple.TCC/TCC.db` and `~/Library/Application Support/com.apple.TCC/TCC.db` — documents all privacy permissions (Keychain, Contacts, Full Disk Access) granted to Script Editor or the AMOS-dropped Mach-O binary during the compromise window | Browser history and downloads SQLite databases for all installed browsers (Chrome: `~/Library/Application Support/Google/Chrome/Default/History` and `Download Metadata`; Firefox: `~/Library/Application Support/Firefox/Profiles/*/places.sqlite`; Safari: `~/Library/Safari/History.db` and `~/Library/Safari/Downloads.plist`) — identifies the Apple-branded lure domain visited before applescript:// invocation and any Mach-O disk image (.dmg) download that preceded Script Editor execution | SHA-256 hashes and strings output of all Mach-O binaries found in `/tmp`, `~/Library/Application Support`, and `~/Library/LaunchAgents` with creation timestamps newer than the suspected compromise time — these binaries constitute the AMOS stealer payload and their embedded strings will contain C2 infrastructure, targeted application bundle IDs (e.g., MetaMask extension ID), and hardcoded Keychain query strings used for credential harvesting

Per-Action IR Details

Step 1: Containment — Audit macOS fleet for any unauthorized Script Editor processes launched via URL scheme. Quarantine any host where Script Editor was opened unexpectedly, especially if followed by network outbound connections or new Mach-O binary execution. Disable or restrict the applescript:// URL scheme via MDM profile if your organization has no legitimate use for it — deploy a LaunchServices policy or Content Filter profile to block the handler fleet-wide.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy

Controls: NIST IR-4 (Incident Handling), NIST SI-4 (System Monitoring), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory)

Compensating: Without MDM, manually block the applescript:// handler on each host via Terminal: run `sudo /usr/bin/defaults write com.apple.LaunchServices LSHandlers -array-add '{LSHandlerURLScheme=applescript;LSHandlerRole\:all=com.apple.TextEdit;}'` as a short-term redirect, or use `pluginkit -e ignore -i com.apple.ScriptEditor2` to suppress Script Editor as a URL handler. Use osquery with `SELECT pid, name, path, cmdline FROM processes WHERE name = 'Script Editor';` piped with parent PID lookup to identify browser-parented Script Editor instances. Network-isolate flagged hosts at the switch or Wi-Fi controller level if remote MDM quarantine is unavailable.

Evidence: Before quarantining, capture: (1) a full process tree snapshot using `sudo eslogger exec | grep -A5 'Script Editor'` or `ps auxww` with PPID resolution to confirm browser parentage; (2) a copy of ~/Library/LaunchAgents/ and /Library/LaunchAgents/ to identify AMOS persistence Mach-O drop locations; (3) contents of /tmp and ~/Library/Application Support for any newly written Mach-O binaries (check `find /tmp ~/Library/Application\ Support -newer /var/log/install.log -type f 2>/dev/null`); (4) network connection state via `sudo lsof -i -n -P` to document active C2 outbound connections at time of containment; (5) a triage image of the user's Keychain directory at ~/Library/Keychains/ before any credential rotation invalidates the forensic value.

Step 2: Detection — Query endpoint telemetry for Script Editor (com.apple.ScriptEditor2) launches triggered by a browser process or URL handler rather than direct user invocation. Look for: (a) osquery or EDR events showing Script Editor as a child of a browser process; (b) new Mach-O binaries written to /tmp, ~/Library/Application Support, or user-writable paths immediately after a Script Editor session; (c) AppleScript execution followed by curl, osascript, or network connections to non-Apple infrastructure; (d) Keychain access events (security.framework prompts) outside normal application context. Review browser history on affected hosts for Apple-branded disk cleanup or utility lure domains.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis

Controls: NIST AU-2 (Event Logging), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Without EDR, deploy osquery with this query to detect browser-parented Script Editor: `SELECT p.pid, p.name, p.cmdline, pp.name AS parent_name, pp.cmdline AS parent_cmdline FROM processes p JOIN processes pp ON p.parent = pp.pid WHERE p.name = 'Script Editor';` — schedule this every 60 seconds via osquery scheduled queries. Enable macOS Unified Log capture for the security.framework subsystem using `log stream --predicate 'subsystem == "com.apple.security"' --level debug > /tmp/keychain_audit.log` to catch Keychain access events triggered by AMOS credential harvesting. Use `mdfind -onlyin ~ 'kMDItemFSCreationDate >= $time.now(-3600) && kMDItemContentType == com.apple.application-bundle'` to surface Mach-O bundles dropped in user space within the last hour. For browser history review, parse Chrome's SQLite history DB at `~/Library/Application Support/Google/Chrome/Default/History` with `sqlite3 History 'SELECT url, title, last_visit_time FROM urls ORDER BY last_visit_time DESC LIMIT 200;'` and grep for Apple-branded lure keywords.

Evidence: Collect before analysis concludes: (1) macOS Unified Log export covering the 30-minute window around suspected Script Editor launch using `log collect --last 1h --output /tmp/unified_log.logarchive`; (2) browser history SQLite databases for all installed browsers (Chrome: `~/Library/Application Support/Google/Chrome/Default/History`; Firefox: `~/Library/Application Support/Firefox/Profiles/*/places.sqlite`; Safari: `~/Library/Safari/History.db`) to identify the lure domain visited before applescript:// invocation; (3) `/var/db/launchd.db/` and LaunchServices database at `/private/var/db/com.apple.LaunchServices/com.apple.launchservices.secure.plist` to confirm applescript:// handler registration state; (4) macOS TCC (Transparency, Consent, and Control) database at `/Library/Application Support/com.apple.TCC/TCC.db` for any permissions granted to Script Editor or dropped AMOS binaries; (5) osascript execution history from Unified Log using `log show --predicate 'process == "osascript"' --last 4h`.

Step 3: Eradication — No Apple patch addresses the applescript:// bypass as of 2026-03-04. Immediate mitigations: (a) block the applescript:// URL handler via MDM (use a Configuration Profile restricting LaunchServices URL scheme handlers); (b) deploy a web content filter or DNS block for known lure domains as IOCs are published by threat intelligence sources; (c) on confirmed-compromised hosts, remove any unknown Mach-O binaries from /tmp and user Library paths; (d) revoke and rotate any credentials stored in

Keychain or browser password managers on affected systems.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST SI-3 (Malicious Code Protection), NIST CM-7 (Least Functionality), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.3 (Address Unauthorized Software)

Compensating: For teams without MDM, distribute a shell script to all macOS endpoints that: (1) removes the applescript:// handler registration using `sudo /System/Library/Frameworks/CoreServices.framework/Frameworks/LaunchServices.framework/Support/lsregister -u /Applications/Script\ Editor.app`; (2) scans for and removes AMOS-typical dropped Mach-O binaries using a YARA rule targeting AMOS string signatures — deploy via `sudo yara -r amos_stealer.yar /tmp ~/Library/Application\ Support 2>/dev/null`; (3) flushes DNS cache with `sudo dscacheutil -flushcache && sudo killall -HUP mDNSResponder` after adding lure domains to /etc/hosts as 127.0.0.1. For credential revocation, generate a per-host list of Keychain items accessed by unknown processes using `security dump-keychain -d ~/Library/Keychains/login.keychain-db` and prioritize rotating entries for corporate SSO, email, and any cryptocurrency exchange credentials surfaced in the dump.

Evidence: Before removing malicious binaries: (1) compute SHA-256 hashes of all suspect Mach-O files using `shasum -a 256 ` and submit to VirusTotal for AMOS family confirmation; (2) capture a full strings dump (`strings -a > strings_output.txt`) to extract embedded C2 domains, hardcoded paths, and AppleScript payload fragments that will serve as additional IOCs; (3) preserve a copy of any AMOS-dropped LaunchAgent plist files from ~/Library/LaunchAgents/ — these contain the persistence mechanism and execution path referencing the dropped Mach-O; (4) export the TCC.db entries granted during the compromise window before eradication removes the access grants; (5) snapshot Keychain access log entries from Unified Log (`log show --predicate 'subsystem == "com.apple.security.sos"' --last 8h`) to document which credential categories AMOS accessed before rotating them.

Step 4: Recovery — After containment and eradication, validate: (a) Script Editor can no longer be triggered via applescript:// from a browser using a controlled test in an isolated VM; (b) no unknown processes persist in user LaunchAgents or LaunchDaemons directories (T1543); (c) force password rotation for all accounts whose credentials may have been stored in Keychain or browser autofill on affected hosts; (d) invalidate active browser session cookies for corporate SaaS applications on affected accounts; (e) monitor affected user accounts for anomalous login activity for a minimum of 30 days post-remediation.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-7 (Software, Firmware, and Information Integrity), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 5.2 (Use Unique Passwords), CIS 6.3 (Require MFA for Externally-Exposed Applications)

Compensating: For the applescript:// block validation test: on an isolated macOS VM snapshot, open Safari and navigate to a test page containing `test` — confirm Script Editor does not launch after the MDM profile or lsregister removal is applied. For LaunchAgent persistence validation, run `find ~/Library/LaunchAgents /Library/LaunchAgents /Library/LaunchDaemons -type f -name '*.plist' -newer /var/log/install.log` and inspect any recently modified plists for paths pointing to /tmp or user Library locations consistent with AMOS drop patterns. For session cookie invalidation without enterprise tooling, coordinate with SaaS administrators to force session termination per-user, or instruct affected users to manually clear all cookies and re-authenticate with newly rotated credentials under enforced MFA. Monitor IdP (Identity Provider) logs — e.g., Okta System Log or Azure AD Sign-in Logs — for the affected accounts for 30 days, alerting on impossible travel, new device registration, or MFA bypass attempts.

Evidence: During recovery validation: (1) re-run the osquery browser-parented Script Editor query post-MDM profile deployment to confirm zero hits; (2) collect a post-remediation LaunchAgents directory listing with file hashes to establish a clean baseline — store as a reference artifact; (3) pull IdP authentication logs for affected accounts covering the suspected compromise window to identify any credential use that occurred before rotation (e.g., Okta System Log filtered by actor.alternateId for affected users); (4) review corporate SaaS application access logs (e.g., Google Workspace Admin audit log, Microsoft 365 UAL) for session activity attributable to stolen cookies, particularly for any actions taken between initial compromise and cookie invalidation; (5) verify cryptocurrency wallet browser extensions (e.g., MetaMask, Phantom) were not exfiltrated by checking browser extension storage at

`~/Library/Application Support//Default/Local Extension Settings//` for unexpected access timestamps.

Step 5: Post-Incident — This campaign reveals a control gap: URL scheme handlers on macOS represent an under-monitored execution path that bypasses many endpoint controls focused on Terminal and shell activity. Review your macOS MDM baseline to include URL scheme handler restrictions as a standard control. Add Script Editor launch-via-URL-handler to your macOS detection ruleset permanently. Evaluate whether cryptocurrency wallet extensions are permitted on corporate-managed browsers — if not business-critical, consider blocking via browser policy. Update security awareness training to include social-engineering lures mimicking Apple-branded utilities.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SI-5 (Security Alerts, Advisories, and Directives), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Write a permanent Sigma rule targeting the applescript:// execution chain for use with any SIEM or log aggregator: match on process creation events where `process.name == 'Script Editor'` AND `process.parent.name IN ('Safari', 'Google Chrome', 'Firefox', 'Arc', 'Brave')` — store in your detection-as-code repo and test quarterly. For the MDM baseline update, add a LaunchServices restriction Configuration Profile payload to your Jamf/Mosyle/Kandji macOS hardening profile that maps to CIS macOS Benchmark Level 1. For cryptocurrency wallet extension governance, query all managed Chrome browsers via `google.chrome.extensions` osquery table to enumerate installed extensions and cross-reference against a prohibited list. For awareness training, create a simulation using a test applescript:// URL on an internal phishing platform (e.g., GoPhish) that mimics an Apple-branded disk cleanup prompt — measure click-through rate and use as a pre/post training metric to demonstrate control improvement to leadership.

Evidence: For the lessons-learned record: (1) compile a timeline of all Script Editor process creation events from the Unified Log archive collected during the incident, annotating each event with whether it was browser-parented or user-invoked — this establishes the detection gap baseline; (2) document the MDM profile deployment timestamp and validate via MDM enrollment records that 100% of managed macOS endpoints received the applescript:// handler restriction; (3) extract final IOC list including all lure domain FQDNs, SHA-256 hashes of AMOS Mach-O binaries, and any C2 IPs observed in lsof captures — share via ISAC or internal threat intel platform in STIX 2.1 format; (4) retain the TCC.db snapshots from compromised hosts as evidence of what Keychain categories and browser credential stores AMOS accessed — this supports scope determination for any breach notification obligations; (5) preserve browser extension storage artifacts for any cryptocurrency wallet extensions present on compromised hosts as evidence of potential financial account exposure.

Detection Guidance

Primary behavioral indicators: (1) Script Editor (com.apple.ScriptEditor2) spawned as a child of a browser process (Safari, Chrome, Firefox, Brave), this is anomalous in all enterprise contexts; (2) osascript or Script Editor execution immediately followed by curl, wget, or an outbound HTTPS connection to a non-Apple host; (3) new unsigned or ad-hoc-signed Mach-O binary written to /tmp or ~/Library paths within minutes of a Script Editor event; (4) Keychain access prompts (securityd events) outside normal application context, particularly from an unknown binary path. Log sources to query: macOS Unified Log (log show filtering for com.apple.ScriptEditor and com.apple.security.keychaind), EDR process tree telemetry, endpoint DLP or file integrity monitoring for new executables in user-writable paths. If using osquery: query processes where parent_path LIKE '%Safari%' OR '%Chrome%' OR '%Firefox%' AND name = 'Script Editor'. Browser forensics: examine browser history for Apple-branded lure pages (disk cleanup, storage optimizer themes) as a host-level IOC. No confirmed hashes or C2 infrastructure IOCs are available in sourced reporting as of 2026-03-04; monitor threat intelligence feeds for updates.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`applescript://`	URL scheme abused to launch macOS Script Editor pre-loaded with attacker-controlled AppleScript; any browser-triggered applescript:// invocation should be treated as malicious in enterprise environments	HIGH

Framework Mappings

MITRE-ATTACK

- **T1539** — Steal Web Session Cookie
- **T1140** — Deobfuscate/Decode Files or Information
- **T1555.001** — Keychain
- **T1555.003** — Credentials from Web Browsers
- **T1566** — Phishing
- **T1204.002** — Malicious File
- **T1056.003** — Web Portal Capture
- **T1056.004** — Credential API Hooking
- **T1562.001** — Disable or Modify Tools
- **T1059.002** — AppleScript
- **T1204.001** — Malicious Link
- **T1543** — Create or Modify System Process
- **T1027** — Obfuscated Files or Information

NIST-800-53R5

- **AT-2** — Literacy Training and Awareness
- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-8** — Spam Protection
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **6.3** — Require MFA for Externally-Exposed Applications
- **7.3** — Perform Automated Operating System Patch Management
- **7.4** — Perform Automated Application Patch Management

HIPAA-SECURITY

- **164.312(d)** — Person or Entity Authentication

SOC2-TSC

- **CC6.1** — Logical access security software, infrastructure, and architectures

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1539	Steal Web Session Cookie	Credential-Access
T1140	Deobfuscate/Decode Files or Information	Defense-Evasion
T1555.001	Keychain	Credential-Access
T1555.003	Credentials from Web Browsers	Credential-Access
T1566	Phishing	Initial-Access
T1204.002	Malicious File	Execution
T1056.003	Web Portal Capture	Collection
T1056.004	Credential API Hooking	Collection
T1562.001	Disable or Modify Tools	Defense-Evasion
T1059.002	AppleScript	Execution
T1204.001	Malicious Link	Execution
T1543	Create or Modify System Process	Persistence
T1027	Obfuscated Files or Information	Defense-Evasion

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/new-macos-stealer-ca...	T3
About the security content of macOS Tahoe 26.4 - Apple Support	https://support.apple.com/en-us/126794	T3
Apple's Latest macOS Tahoe 26.4 Update Hopes To Protect Against ...	https://pcper.com/2026/03/apples-latest-macos-tahoe-26-4-update-hop...	T3
About the security content of macOS Tahoe 26 - Apple Support	https://support.apple.com/en-us/125110	T3
macOS 26.x < 26.1 Multiple Vulnerabilities (125634) | Tenable®	https://www.tenable.com/plugins/nessus/272228	T3

DISCLAIMER

This intelligence report is produced by Tech Jacks Solutions Security Command Center (SCC) for informational purposes only. It does not constitute professional security advice, legal counsel, or an incident response engagement. The information herein is derived from publicly available sources and AI-assisted analysis; while every effort is made to ensure accuracy, Tech Jacks Solutions makes no warranties regarding completeness or timeliness. Organizations should conduct their own validation and consult qualified security professionals before taking action based on this report. Tech Jacks Solutions is not liable for any damages resulting from the use of this information.

Generated 2026-04-08 18:47 UTC by TJS Security Command Center