INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-03-29 18:36 UTC

Infinity Stealer Combines ClickFix Social Engineering with Nuitka-Compiled Python to Target macOS Credentials and Crypto Wallets

THREAT CAMPAIGN | **HIGH** | CVSS 7.5

SCC Item ID	SCC-CAM-2026-0119
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	7.5
Affected Products	macOS (all recent versions), Chromium-based browsers, Firefox, macOS Keychain, cryptocurrency wallets (unspecified), developer .env secret stores
Published	2026-03-28
Discovery Source	Rss

Executive Summary

Infinity Stealer is a newly identified macOS infostealer that tricks users into executing a malicious Terminal command disguised as a Cloudflare CAPTCHA verification. Once run, it harvests browser credentials, macOS Keychain data, cryptocurrency wallets, and developer environment secrets (.env files). Organizations with macOS-using developers or finance personnel holding crypto assets face elevated risk of credential theft and potential supply chain exposure if secret stores are compromised.

Technical Analysis

Infinity Stealer targets macOS (all recent versions) using a two-stage attack chain. Stage one is a ClickFix social engineering lure: a fake Cloudflare CAPTCHA page instructs the victim to paste and execute a Terminal command (T1059.004, Unix Shell), bypassing traditional delivery controls by requiring user-initiated execution (MITRE T1204.002, T1566). Stage two retrieves a Python 3.11 payload (T1059.006, Python) compiled into a native macOS Mach-O binary via the Nuitka compiler (T1027.002, T1140), a technique noted in Malwarebytes reporting as a notable evasion approach on macOS. The compiled binary is harder to detect via static signature analysis than interpreted Python (CWE-506, CWE-693). Post-execution, the stealer harvests: credentials from Chromium-based browsers and Firefox (T1555.003); macOS Keychain entries (T1555.001); cryptocurrency wallet data (T1657); and developer .env secret files (T1552.001). Data is exfiltrated over the network (T1041). Sandbox evasion behavior is noted (T1497.001). No CVE is assigned. Relevant CWEs: CWE-312 (cleartext storage), CWE-693 (protection mechanism failure), CWE-506 (embedded malicious code), CWE-184

(incomplete denylist). Threat actor attribution is unknown. No vendor patch exists; this is a social engineering and detection challenge, not a patchable vulnerability. Severity is editorial, based on scope and target criticality, not CVSS-derived. Source: Malwarebytes (T3), BleepingComputer (T3). Source quality score: 0.64, treat specific technical claims as current best available, pending higher-tier confirmation. Priority score 0.382 reflects current limited prevalence reporting and unconfirmed threat actor attribution.

Action Checklist

1. Step 1: Containment. Identify macOS endpoints where users executed Terminal commands copy-pasted from a browser, particularly following a CAPTCHA prompt. Isolate any endpoint with suspicious Mach-O binaries in user-writable directories (~/Downloads, /tmp, ~/.local). Block outbound connections from endpoints exhibiting unexpected network activity from Python runtimes or unrecognized Mach-O processes pending investigation.
2. Step 2: Detection. Search EDR telemetry for: (a) Terminal process spawns initiated from a browser process or clipboard paste events; (b) Python 3.11 runtime executing from non-standard paths or launching child processes; (c) Nuitka-compiled binary indicators such as Mach-O binaries containing embedded Python runtime artifacts in user-writable directories; (d) unusual reads of ~/Library/Keychains, browser profile directories (~/Library/Application Support/Google/Chrome, ~/Library/Application Support/Firefox), and .env files in developer project directories; (e) outbound HTTP/S data transfers from Python or unknown Mach-O processes. Check DNS and proxy logs for domains associated with fake Cloudflare CAPTCHA pages. No confirmed IOC hashes or domains are available from current sources; rely on behavioral detection (process ancestry, file access patterns, network connections) rather than hash-based matching. Monitor Malwarebytes and BleepingComputer for IOC updates.
3. Step 3: Eradication. No vendor patch applies; this is a social engineering vector. Deploy or enforce a Terminal usage policy that prohibits copy-paste command execution from browser sessions for non-administrative roles. Remove any identified malicious Mach-O binaries. Rotate all credentials, Keychain entries, API keys, and .env secrets on any confirmed or suspected compromised host. Revoke and reissue tokens for any developer secret stores exposed.
4. Step 4: Recovery. After credential rotation, validate that no unauthorized sessions exist in SaaS platforms, cloud consoles, or code repositories. Monitor for reuse of harvested credentials in authentication logs (look for logins from unexpected IPs or geolocations). Re-image confirmed compromised endpoints. Verify .env file integrity and audit access logs for any pipelines or services that consumed those secrets.
5. Step 5: Post-Incident. Conduct a control gap review against MITRE T1204.002 (user execution) and T1566 (phishing). Evaluate whether security awareness training covers ClickFix-style social engineering, specifically CAPTCHA-lure Terminal execution. Assess whether developer workstations have endpoint detection capable of behavioral analysis on Mach-O binaries, not just signature matching. Review secret management practices: .env files in developer repos should be replaced with a secrets manager (e.g., HashiCorp Vault, AWS Secrets Manager) to limit blast radius.

IR / Forensic Enrichment

Triage Priority	URGENT

Escalation Criteria	Escalate immediately to CISO and legal counsel if any confirmed-compromised .env file contained secrets with access to production infrastructure, customer PII data stores, or financial systems — triggering breach notification assessment under applicable regulations (GDPR Article 33, CCPA, or state breach notification laws) — or if credential reuse is detected in authentication logs indicating active attacker presence in SaaS, cloud, or code repository environments.
Recovery Notes	After re-imaging and credential rotation, monitor authentication logs across all SaaS platforms, cloud consoles (AWS CloudTrail, GCP Audit Logs, Azure AD Sign-in Logs), and code repositories (GitHub audit log) for a minimum of 30 days for logins from unexpected IPs, geolocations, or user agents consistent with credential stuffing. Validate that all CI/CD pipelines consuming rotated secrets have been re-triggered with new credentials and that no pipeline artifacts (build outputs, Docker images, deployed configurations) were produced using compromised secrets during the exposure window. If cryptocurrency wallet keys were stored in the Keychain or .env files, treat those wallets as fully compromised and transfer remaining assets to newly generated wallets immediately, as private key theft is irreversible.
Forensic Artifacts	macOS Unified Log archive (collected via `log collect --last 72h`) — contains Terminal process spawn events showing the browser parent process (Safari, Chrome) that triggered the ClickFix paste execution, and Python/Mach-O child process chains originating from ~/Downloads or /tmp | Shell history files (~/.zsh_history, ~/.bash_history) — preserve the exact ClickFix Terminal command pasted by the user, including the curl or Python invocation used to download and execute the Nuitka-compiled Infinity Stealer binary | Nuitka-compiled Mach-O binary recovered from ~/Downloads, /tmp, or ~/.local — SHA-256 hashed for threat intelligence sharing; analyzable with `strings | grep -E 'nuitka|python|keychain|wallet|.env'` to confirm payload capabilities and C2 embedded strings | macOS Keychain access audit trail from OpenBSM (/var/audit/* if enabled) or Unified Log filtered for `securityd` and `Security` framework events — identifies which process (the Mach-O binary's PID) accessed ~/Library/Keychains/ and at what timestamp, establishing the credential harvesting timeline | Browser SQLite databases (Chrome: ~/Library/Application Support/Google/Chrome/Default/History and Login Data; Firefox: ~/Library/Application Support/Firefox/Profiles/*/places.sqlite and logins.json) — History database reveals the fake Cloudflare CAPTCHA domain visited immediately before Terminal execution; Login Data confirms whether encrypted browser-stored credentials were accessed by the stealer process

Per-Action IR Details

Step 1: Containment — Identify macOS endpoints where users have recently executed Terminal commands via copy-paste from a browser, particularly following a CAPTCHA prompt. Isolate any endpoint with suspicious Mach-O binaries in user-writable directories (~/Downloads, /tmp, ~/.local). Block outbound connections from endpoints exhibiting unexpected network activity from Python runtimes or unrecognized Mach-O processes pending investigation.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment, Eradication, and Recovery (Containment Strategy)

Controls: NIST IR-4 (Incident Handling), NIST SC-7 (Boundary Protection), NIST SI-4 (System Monitoring), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 4.5 (Implement and Manage a Firewall on End-User Devices), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory)

Compensating: For teams without EDR: run `find /Users/*/Downloads /tmp /Users/*/.local -name '*.app' -o -perm +111 -type f 2>/dev/null` on each macOS endpoint to locate unexpected executables. Use `lsof -i -n -P | grep -E 'python|Python'` to identify active outbound Python connections. Block outbound traffic from flagged PIDs immediately using `pfctl` (macOS packet filter): add a rule to /etc/pf.conf anchoring the specific process UID. Use `osquery` with the query `SELECT pid, name, path, cmdline FROM processes WHERE path LIKE '%python%' OR path LIKE '%/tmp/%'

OR path LIKE '%Downloads%';` to enumerate suspicious runtimes across fleet via osquery's distributed query.

Evidence: Before isolating: capture a full process listing with parent-child relationships using `ps -axo pid,ppid,user,comm,args > /tmp/ir_ps_snapshot.txt`; capture active network connections using `netstat -anv > /tmp/ir_netstat.txt` and `lsof -i > /tmp/ir_lsof.txt`; collect unified logs for Terminal spawn events via `log show --predicate 'process == "Terminal" OR process == "python3"' --last 24h > /tmp/ir_unified.log`; image or hash (SHA-256) any Mach-O binary found in ~/Downloads, /tmp, or ~/.local before removal using `shasum -a 256 `; export the user's shell history with `cat ~/.zsh_history > /tmp/ir_zsh_history.txt` and `cat ~/.bash_history > /tmp/ir_bash_history.txt` to capture the pasted ClickFix Terminal command.

Step 2: Detection — Search EDR telemetry for: (a) Terminal process spawns initiated from a browser process or clipboard paste events; (b) Python 3.11 runtime executing from non-standard paths or launching child processes; (c) Nuitka-compiled binary indicators such as Mach-O binaries containing embedded Python runtime artifacts in user-writable directories; (d) unusual reads of ~/Library/Keychains, browser profile directories (~/Library/Application Support/Google/Chrome, ~/Library/Application Support/Firefox), and .env files in developer project directories; (e) outbound HTTP/S data transfers from Python or unknown Mach-O processes. Check DNS and proxy logs for domains associated with fake Cloudflare CAPTCHA pages. No confirmed IOC hashes or domains are available from current sources — monitor for Malwarebytes and BleepingComputer updates.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis (Signs of an Incident, Incident Analysis)

Controls: NIST SI-4 (System Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-2 (Event Logging), NIST AU-12 (Audit Record Generation), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Without EDR: (a) Detect Terminal-browser spawn chain via macOS Unified Log: `log show --predicate 'process == "Terminal" AND (eventMessage CONTAINS "curl" OR eventMessage CONTAINS "bash" OR eventMessage CONTAINS "python")' --last 48h`. (b) Detect non-standard Python execution: `find /Users -name 'python3.11' -not -path '/usr/bin/*' -not -path '/usr/local/bin/*' 2>/dev/null`. (c) Detect Nuitka-compiled Mach-O artifacts by scanning for embedded Python runtime strings: `grep -rl '__nuitka__' /Users/*/Downloads /tmp 2>/dev/null` or use a YARA rule targeting Nuitka ELF/Mach-O markers (strings: `__nuitka_version__`, `nuitka_module_loader`). (d) Detect Keychain and browser credential access: `log show --predicate 'process != "Keychain Access" AND (eventMessage CONTAINS "Keychains" OR eventMessage CONTAINS "Chrome" OR eventMessage CONTAINS "Firefox")' --last 48h`. (e) For outbound data exfiltration detection, run Wireshark or `tcpdump -i en0 -w /tmp/ir_capture.pcap 'port 443 or port 80'` and filter for Python or unknown process connections. For DNS hunting on fake Cloudflare CAPTCHA pages, extract unique domains from proxy or DNS logs and cross-reference against newly registered domains or domains with Cloudflare-themed strings using `grep -iE 'cloudflare|captcha|verify|human' /var/log/squid/access.log`.

Evidence: Capture before analysis: macOS Unified Log export for Terminal, Python, and browser processes for the prior 72 hours (`log collect --output /tmp/ir_logarchive.logarchive --last 72h`); OpenBSM audit trail if enabled (`/var/audit/*` — records file opens, execs, and network calls at syscall level); browser history SQLite databases at `~/Library/Application Support/Google/Chrome/Default/History` and `~/Library/Application Support/Firefox/Profiles/*/places.sqlite` to identify the fake CAPTCHA page URL visited immediately before Terminal execution; `~/Library/Keychains/` directory listing and modification timestamps (`ls -la ~/Library/Keychains/`) to detect unauthorized reads; DNS query logs from network infrastructure or endpoint resolver cache (`sudo dscacheutil -cachedump -entries Host`) to surface C2 or exfiltration domains contacted by the Nuitka binary.

Step 3: Eradication — No vendor patch applies; this is a social engineering vector. Deploy or enforce a Terminal usage policy that prohibits copy-paste command execution from browser sessions for non-administrative roles. Remove any identified malicious Mach-O binaries. Rotate all credentials, Keychain entries, API keys, and .env secrets on any confirmed or suspected compromised host. Revoke and reissue tokens for any developer secret stores exposed.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication and Recovery (Eliminating Components of the Incident)

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST SI-3 (Malicious Code Protection), NIST AC-2 (Account Management), NIST IA-5 (Authenticator Management), CIS 2.3 (Address Unauthorized Software), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Remove malicious Mach-O binaries: `sudo rm -rf ~/Downloads/ /tmp/` after SHA-256 hashing for evidence. Verify removal of persistence mechanisms: check `~/Library/LaunchAgents/`, `/Library/LaunchAgents/`, and `/Library/LaunchDaemons/` for plist files referencing the malicious binary or Python paths (`ls -la ~/Library/LaunchAgents/`); remove unauthorized entries. Rotate macOS Keychain: advise users to use Keychain Access.app to audit and delete credentials, or script with `security delete-generic-password -l "`. For .env secret rotation, use a checklist: enumerate exposed keys with `grep -rE '(API_KEY|SECRET|TOKEN|PASSWORD)=' /Users//Projects/**/.env 2>/dev/null`, then revoke each identified token in the respective platform (GitHub, AWS IAM, Stripe, etc.). Enforce Terminal policy via macOS MDM profile (Jamf, Mosyle) restricting Terminal.app to admin roles, or document a written policy with acknowledgment signature for smaller teams.

Evidence: Before eradicating: preserve a forensic copy of each malicious Mach-O binary (`cp ~/Downloads/ /tmp/ir_evidence/`) and document SHA-256 hash; export the full LaunchAgents/LaunchDaemons plist inventory (`launchctl list > /tmp/ir_launchctl.txt`); record all Keychain entries accessible to the compromised user account (`security dump-keychain ~/Library/Keychains/login.keychain-db > /tmp/ir_keychain_dump.txt` — handle with care, output is sensitive); enumerate all .env files that may have been read (`find /Users//Projects -name '.env' -newer /tmp/ir_binary_timestamp 2>/dev/null`) using the malicious binary's modification time as a reference anchor for access correlation.

Step 4: Recovery — After credential rotation, validate that no unauthorized sessions exist in SaaS platforms, cloud consoles, or code repositories. Monitor for reuse of harvested credentials in authentication logs (look for logins from unexpected IPs or geolocations). Re-image confirmed compromised endpoints. Verify .env file integrity and audit access logs for any pipelines or services that consumed those secrets.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery (Restoring Systems to Normal Operations)

Controls: NIST IR-4 (Incident Handling), NIST CP-10 (System Recovery and Reconstitution), NIST AC-2 (Account Management), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-11 (Audit Record Retention), NIST SI-7 (Software, Firmware, and Information Integrity), CIS 5.3 (Disable Dormant Accounts), CIS 6.2 (Establish an Access Revoking Process), CIS 3.4 (Enforce Data Retention)

Compensating: For session validation without a CASB: manually audit active OAuth tokens and API keys in each SaaS platform (GitHub: Settings > Developer Settings > Personal Access Tokens; AWS: IAM > Users > Security Credentials; Google Workspace: Admin Console > Security > API Controls). For unexpected login detection without a SIEM: export authentication logs from each platform to CSV and filter for logins outside the user's normal country or ASN using a free GeoIP lookup script: `python3 -c "import csv,urllib.request,json; [print(r) for r in csv.DictReader(open('auth.csv'))]"` combined with ip-api.com batch API (free tier). For CI/CD pipeline secret audit: search GitHub Actions logs or GitLab CI logs for any job that consumed the rotated secret after the compromise window — flag any successful pipeline run using the old credential as potentially tainted. Re-image using a known-good MDM enrollment baseline and validate integrity of the new build by confirming MDM enrollment certificate and running `csrutil status` to verify SIP is enabled post-reimaging.

Evidence: Collect before re-imaging: full disk image using `asr` or Target Disk Mode to an external forensic drive; export all authentication logs from SaaS platforms covering the 7-day window before and after the suspected compromise date; pull CI/CD pipeline execution logs showing which secrets were used and by which jobs; capture `~/Library/Application Support/Google/Chrome/Default/Login Data` and `~/Library/Application Support/Firefox/Profiles/*/logins.json` (encrypted) as evidence that Infinity Stealer targeted these paths; document the .env file last-accessed timestamps (`stat /Users//Projects/**/.env`) to establish the exfiltration timeline.

Step 5: Post-Incident — Conduct a control gap review against MITRE T1204.002 (user execution) and T1566 (phishing). Evaluate whether security awareness training covers ClickFix-style social engineering, specifically CAPTCHA-lure Terminal execution. Assess whether developer workstations have endpoint detection capable of behavioral analysis on Mach-O binaries, not just signature matching. Review secret management practices:

.env files in developer repos should be replaced with a secrets manager (e.g., HashiCorp Vault, AWS Secrets Manager) to limit blast radius.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity (Lessons Learned, Evidence Retention)

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST IR-2 (Incident Response Training), NIST IR-3 (Incident Response Testing), NIST SI-5 (Security Alerts, Advisories, and Directives), NIST RA-3 (Risk Assessment), NIST SA-8 (Security and Privacy Engineering Principles), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 6.3 (Require MFA for Externally-Exposed Applications), CIS 6.5 (Require MFA for Administrative Access)

Compensating: Control gap assessment without a GRC platform: create a spreadsheet mapping MITRE T1204.002 (User Execution: Malicious File) and T1566.004 (Phishing: Spearphishing via Service — ClickFix variant) to current detective and preventive controls, marking each as present, partial, or absent. For behavioral detection on Mach-O binaries without commercial EDR: deploy open-source YARA rules targeting Nuitka-compiled Python artifacts (search the YARA-Rules GitHub repository for Nuitka signatures, or author a custom rule matching the strings `__nuitka__` and embedded CPython version markers); integrate with ClamAV's on-access scanning (`clamd`) for continuous monitoring on developer endpoints. For secrets management migration: pilot HashiCorp Vault Community Edition (free, self-hosted) to replace .env files in the highest-risk developer projects first; document the migration in a 30/60/90-day plan. For awareness training: add a specific ClickFix scenario — 'a website asks you to open Terminal and paste a command to prove you are human' — to the next phishing simulation cycle, using GoPhish (free, open-source) to deliver and track completion.

Evidence: For the lessons-learned record: retain all forensic images, log exports, and IR timeline documentation for a minimum of 12 months per NIST AU-11 (Audit Record Retention) or longer if regulatory obligations apply; document the specific ClickFix lure page URL (recovered from browser history) and the exact Terminal command pasted by the user as the primary attack vector evidence; preserve the Nuitka Mach-O binary samples (hashed and stored in an isolated evidence repository) for future YARA rule development and threat intelligence sharing; record detection gaps identified — specifically, whether any existing tool alerted on Keychain access or Python execution from ~/Downloads — to drive the control improvement roadmap.

Detection Guidance

Primary behavioral indicators: (1) Terminal.app or any shell process spawned with clipboard-pasted content originating from a browser process, look for parent-child process chains where a browser spawns or indirectly triggers a shell. (2) Python 3.11 runtime executing a Nuitka-compiled Mach-O binary from ~/Downloads, /tmp, or other user-writable paths, Nuitka binaries embed a Python runtime and may appear as single large executables with unusual internal structure. (3) File read events targeting ~/Library/Keychains/, browser profile credential stores (Login Data SQLite files for Chromium browsers, key4.db and logins.json for Firefox), cryptocurrency wallet config directories, and .env files in home or project directories. (4) Outbound network connections initiated by Python processes or unknown Mach-O binaries, particularly to newly registered or low-reputation domains. EDR platforms with behavioral rules (e.g., CrowdStrike, SentinelOne, Microsoft Defender for Endpoint on macOS) should alert on credential store access by non-browser processes. No confirmed IOC hashes, domains, or IPs are available from current T3 sources; rely on behavioral detection (process ancestry, file access patterns, network connections) rather than hash-based matching. Monitor Malwarebytes Threat Intelligence and BleepingComputer for IOC updates.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`Not available — no confirmed IOCs published in current sources`	Malwarebytes and BleepingComputer reporting does not include confirmed hashes, domains, or IP addresses at time of this item's sourcing. Monitor vendor threat intelligence feeds for updates.	LOW

Framework Mappings

MITRE-ATTACK

- **T1555** — Credentials from Password Stores
- **T1027.002** — Software Packing
- **T1566** — Phishing
- **T1059.004** — Unix Shell
- **T1059.006** — Python
- **T1657** — Financial Theft
- **T1555.001** — Keychain
- **T1140** — Deobfuscate/Decode Files or Information
- **T1555.003** — Credentials from Web Browsers
- **T1041** — Exfiltration Over C2 Channel
- **T1204.002** — Malicious File
- **T1552.001** — Credentials In Files
- **T1497.001** — System Checks

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
- **CM-7** — Least Functionality
- **SI-10** — Information Input Validation
- **SI-7** — Software, Firmware, and Information Integrity
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A03:2021** — Injection

CIS-V8

- **16.10**
- **6.3** — Require MFA for Externally-Exposed Applications
- **14.2** — Train Workforce Members to Recognize Social Engineering Attacks
- **15.1** — Establish and Maintain an Inventory of Service Providers
- **8.2** — Collect Audit Logs

HIPAA-SECURITY

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

SOC2-TSC

- **CC6.1** — Logical access security software, infrastructure, and architectures
- **CC9.2** — Manages risks associated with vendors and business partners

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program
- **DE.CM-01** — Networks and network services are monitored

ISO-27001-2022

- **A.5.21** — Managing information security in the ICT supply chain
- **A.5.23** — Information security for use of cloud services

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1555	Credentials from Password Stores	Credential-Access
T1027.002	Software Packing	Defense-Evasion
T1566	Phishing	Initial-Access
T1059.004	Unix Shell	Execution
T1059.006	Python	Execution
T1657	Financial Theft	Impact
T1555.001	Keychain	Credential-Access
T1140	Deobfuscate/Decode Files or Information	Defense-Evasion
T1555.003	Credentials from Web Browsers	Credential-Access
T1041	Exfiltration Over C2 Channel	Exfiltration
T1204.002	Malicious File	Execution
T1552.001	Credentials In Files	Credential-Access

Technique ID	Technique Name	Tactic
T1497.001	System Checks	Defense-Evasion

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/new-infinity-stealer...	T3
Odyssey Stealer: Advanced macOS Malware Targeting Crypto	https://wizardcyber.com/odyssey-stealer-macos-malware-analysis/	T3
Fake CleanMyMac site installs SHub Stealer and backdoors crypto ...	https://www.malwarebytes.com/blog/threat-intel/2026/03/fake-cleanmy...	T3
Odyssey Stealer: Inside a macOS Crypto-Stealing Operation - Censys	https://censys.com/blog/odyssey-stealer-inside-a-macos-crypto-steal...	T3
A malicious npm package is spreading a full RAT malware ...	https://www.facebook.com/thehackernews/posts/%EF%B8%8F-a-malicious-...	T3

Generated 2026-03-29 18:36 UTC by TJS Security Command Center