INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-03-29 18:34 UTC

GhostLoader Supply Chain Campaign Targets Developers with Fake npm Packages, Sudo Phishing, and Dual-Revenue Theft Model

THREAT CAMPAIGN | CRITICAL | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0093
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	npm registry, Node.js ecosystem, macOS, GitHub, Telegram, Binance Smart Chain (BSC), OpenClaw AI agent platform
Published	2026-03-24
Discovery Source	Rss

Executive Summary

A multi-wave software supply chain campaign, tracked by ReversingLabs and JFrog, is distributing malicious npm packages that install GhostLoader, a macOS remote access trojan, on developer machines. Attackers steal sudo credentials, browser-stored passwords, and cryptocurrency wallet data, then exfiltrate via Telegram bots while managing affiliates through Binance Smart Chain smart contracts. Organizations with macOS developers are at risk if they have installed npm packages from the malicious publisher account 'mikilanjillo' or trojanized GitHub repositories associated with this campaign, or if they use OpenClaw AI agent tooling known to be targeted by this campaign.

Technical Analysis

GhostLoader is a macOS RAT delivered through malicious npm packages published under the account 'mikilanjillo' and trojanized GitHub repositories. The campaign employs typosquatting and dependency confusion (T1195.001, T1036.005) to achieve initial access. During installation, fake UI progress indicators (T1056.002) mask malicious activity while the installer displays a fake macOS system authentication dialog designed to mimic native OS credential prompts, a GUI-based input capture technique (CWE-522). Once elevated access is obtained, GhostLoader is dropped and persisted on the macOS host. The malware exfiltrates credentials from browser credential stores (T1555.003, T1555), private keys and secrets from files (T1552.001, T1552.004), and cryptocurrency wallet data via Telegram bot C2 channels (T1071.001, T1041). Payloads are obfuscated and encoded (T1027, T1140), delivered via ingress tool transfer (T1105), and execute JavaScript

and Python-based malicious code (T1059.007, T1059.006). The campaign operates a dual-revenue model: primary revenue from cryptocurrency wallet and credential theft; secondary revenue from affiliate commission fees managed through BSC smart contracts (T1098), decoupling infrastructure from operator control and making takedown significantly harder. The campaign explicitly targets AI developer workflows, including the OpenClaw AI agent platform. Relevant CWEs: CWE-312 (cleartext credential storage), CWE-506 (embedded malicious code), CWE-522 (insufficiently protected credentials), CWE-494 (download of code without integrity check). No CVE assigned. No patch available, campaign-based threat.

Action Checklist

1. Step 1, Immediate: Audit npm dependencies installed in the last 90 days against the known malicious publisher account 'mikilanjillo'; remove and quarantine any packages from that account. Block the account name and associated package names at your registry proxy or Artifactory instance.
2. Step 2, Immediate: Isolate any macOS developer machines that: (a) displayed sudo credential prompts during npm install, (b) have suspicious LaunchAgent/LaunchDaemon entries created after recent package installations, or (c) show Telegram API connections in network logs. Treat confirmed cases as compromised; investigate unconfirmed cases with forensic tools before escalation.
3. Step 3, Detection: Search endpoint logs for unexpected outbound connections to Telegram API endpoints (api.telegram.org) and unexpected or obfuscated Binance Smart Chain RPC connections from developer workstations. Exclude known legitimate blockchain development libraries (e.g., ethers.js, web3.js) operating in expected contexts. Flag any processes spawned by npm or node that initiate unexpected network connections.
4. Step 4, Detection: Query browser credential stores and macOS Keychain access logs for reads by unexpected processes. Review bash and zsh history on developer machines for sudo commands not initiated by a known user action.
5. Step 5, Assessment: Inventory all macOS developer systems with npm, Node.js, or OpenClaw tooling installed. Prioritize systems used in CI/CD pipelines or with access to production secrets, signing keys, or cryptocurrency wallets.
6. Step 6, Communication: Notify development teams to halt installation of npm packages from unverified sources and to treat any recent sudo prompts during package install as a potential compromise indicator. Escalate affected system owners to incident response. Retrieve comprehensive IOC lists (package names, file hashes, network indicators) directly from ReversingLabs and JFrog published research before operationalizing detection rules.
7. Step 7, Long-term: Enforce package integrity verification (npm audit signatures, lockfile pinning, private registry mirroring). Implement controls that prevent npm install scripts from spawning GUI dialogs or prompting for elevated credentials. Evaluate secrets scanning across developer endpoints and CI/CD systems.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate to external incident response firm or law enforcement if evidence shows exfiltration of production secrets, code signing keys, or cryptocurrency wallet credentials; or if any developer system has confirmed persistence mechanism (cron job, launchd agent) installed by GhostLoader.
Recovery Notes	Post-containment: rebuild all affected developer machines from known-good macOS image with hardened configuration (System Integrity Protection enabled, no postinstall script execution, non-root npm user). Rotate all developer credentials (SSH keys, GitHub tokens, AWS/GCP service account keys, npm auth tokens) that may have been accessed. Perform forensic imaging of isolated systems and preserve for 90 days pending legal hold. Review and revoke any code commits pushed from affected systems in past 90 days; re-verify commit signatures and audit published artifacts in artifact repositories.
Forensic Artifacts	~/.bash_history, ~/.zsh_history, ~/.zsh_history_extended (shell command history with timestamps) | ~/.npm/_logs/* (npm install progress and error logs covering entire execution) | /var/log/system.log (macOS unified logging: sudo, Keychain, security events, network sockets) | ~/Library/Keychains/login.keychain-db (macOS Keychain credential store; requires forensic dump) | package-lock.json, npm-shrinkwrap.json, .npmrc (npm dependency manifest and configuration showing package sources and versions) | tcpdump/pcap of developer machine network traffic during suspected installation window (captures Telegram API and BSC RPC outbound connections) | ~/.npm (npm cache directory: contains package metadata and install logs) | /var/log/auth.log (macOS authentication and sudo execution audit logs) | GitHub Actions workflow logs and CI/CD pipeline build artifacts (exported via API) showing package installations | Keychain access audit logs via 'log show --predicate "sender == Security"' (identifies unauthorized credential access attempts)

Per-Action IR Details

Step 1 — Immediate: Audit npm dependencies installed in the last 90 days against the known malicious publisher account 'mikilanjillo'; remove and quarantine any packages from that account. Block the account name and associated package names at your registry proxy or Artifactory instance.

NIST Phase: Preparation

Reference: NIST 800-61r3 §2.2 (preparation: tools and processes)

Controls: NIST 800-53 SI-4 (information system monitoring), NIST 800-53 RA-5 (vulnerability scanning), CIS 6.1 (inventory and control of enterprise software)

Compensating: Run `npm ls --all > npm_audit_baseline.json` on each developer machine and grep for 'mikilanjillo' by publisher field. Cross-reference against known-bad package list maintained in Git. Use curl + npm registry API to bulk-check: `curl -s 'https://registry.npmjs.org/-/v1/search?text=publisher:mikilanjillo&size=100'` and log output. For Artifactory-free teams: use local npm cache audit via `npm cache verify` and manual spot-check of node_modules/.package-lock.json for mikilanjillo entries.

Evidence: Capture npm shrinkwrap.json, package-lock.json, and full `npm ls` output before removal. Preserve node_modules directory structure (tar/zip) showing installation timestamps. Document .npmrc proxy configuration and any custom registry tokens used. Export npm registry cache from ~/.npm if present.

Step 2 — Immediate: Isolate any macOS developer machines that prompted for sudo credentials during a package installation or displayed unusual progress UI during npm install or GitHub repo setup. Treat those systems as compromised pending forensic review.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.2.2 (containment: short-term and long-term)

Controls: NIST 800-53 IR-4 (incident handling), NIST 800-53 AC-6 (least privilege), CIS 5.3 (access control: restrict sudo access)

Compensating: Interview developers directly: 'Did npm install or postinstall scripts ever ask for your password?' Correlate with system.log entries showing sudo prompts during npm processes (search `log show --predicate 'process == "sudo"' --last 90d`). On isolated systems, run `sudo last` to review all sudo session history with timestamps; cross-reference against npm install activity in bash_history. Without EDR, use manual timeline correlation: export npm install times from .npm/logs/* and compare against system.log timestamp for unexpected privilege escalation.

Evidence: Before isolation: snapshot /var/log/system.log for sudo execution timestamps. Export ~/.bash_history, ~/.zsh_history, and ~/.npm/_logs/* directory (contains install progress logs with timestamps). Capture screenshot of any unusual sudo prompts if still visible. Preserve Keychain access logs via `log show --level debug --predicate 'process == "Keychain"' --last 90d` before network disconnect. Take forensic image of entire /Users/*/Library/Caches/npm directory.

Step 3 — Detection: Search endpoint logs for unexpected outbound connections to Telegram API endpoints (api.telegram.org) and Binance Smart Chain RPC nodes from developer workstations. Flag any processes spawned by npm or node that initiate network connections.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2.1 (detection and analysis: analysis)

Controls: NIST 800-53 SI-4 (information system monitoring), NIST 800-53 CA-7 (continuous monitoring), CIS 8.1 (network monitoring)

Compensating: Query DNS logs for api.telegram.org and Binance BSC RPC node resolutions (e.g., bsc-dataseed.binance.org, bsc-dataseed1.binance.org). Without network TAP: use macOS `log show --predicate 'sender == "mDNSResponder"' --last 90d` to extract DNS queries. Examine /var/log/system.log for network socket creation by npm/node processes: `grep -i 'node\|npm' /var/log/system.log | grep -i 'socket\|connect'`. On older macOS: use `lsof -i -P -n` captured to baseline file, then compare against current state for unexpected listening sockets. Check ~/.npm/_logs/* for network-related errors during install that may indicate exfiltration.

Evidence: Capture DNS query logs (macOS: query logs from mDNSResponder via syslog export). Export full tcpdump/pcap from developer machine network interface during suspected installation window: `tcpdump -i en0 -w suspect_npm_install.pcap 'tcp port 443 or udp port 443 or tcp port 80'`. Preserve /var/log/system.log covering entire 90-day window. Extract DNS cache: `log show --last 90d --predicate 'process == "mDNSResponder"' > dns_history.txt`. Capture all active network connections at time of isolation via `netstat -an > netstat_baseline.txt` and `lsof -i -P -n > lsof_baseline.txt`.

Step 4 — Detection: Query browser credential stores and macOS Keychain access logs for reads by unexpected processes. Review bash and zsh history on developer machines for sudo commands not initiated by a known user action.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2.1 (detection and analysis: triage and prioritization)

Controls: NIST 800-53 IA-4 (identifier management), NIST 800-53 AC-2 (account management), CIS 6.2 (vulnerability scanning: credentials discovery)

Compensating: Export Keychain entries manually: `security dump-keychain -d login.keychain > keychain_dump.txt` (requires interactive session). Cross-reference against /var/log/system.log for 'SecKeychainItemCopyContent' or 'Keychain' entries showing non-user processes accessing credentials. For bash/zsh: search shell histories for `sudo` invocations without corresponding user interaction (TTY logs or screen session markers). Compare timestamps of sudo commands in history against user login records via `last -f /var/log/wtmp`. Use `process_monitor.sh` script (GitHub: osquery/osquery) to log all process execution with parent PID; correlate npm/node parent processes spawning credential access.

Evidence: Export ~/Library/Keychains/login.keychain-db using `security dump-keychain` and preserve binary. Capture Keychain access audit logs (if enabled): `log show --predicate 'sender == "Security"' --last 90d > keychain_access.log`. Export ~/.bash_history, ~/.zsh_history, and ~/.zsh_history_extended (if present) with file modification timestamps. Preserve system.log entries showing all sudo executions: `grep 'sudo:' /var/log/system.log > sudo_audit.log`. Screenshot or capture active Keychain items at isolation time via Keychain Access.app.

Step 5 — Assessment: Inventory all macOS developer systems with npm, Node.js, or OpenClaw tooling installed. Prioritize systems used in CI/CD pipelines or with access to production secrets, signing keys, or cryptocurrency wallets.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2.1 (detection and analysis: scope and impact determination)

Controls: NIST 800-53 CM-8 (information system component inventory), NIST 800-53 RA-3 (risk assessment), CIS 1.1 (asset inventory: hardware and software)

Compensating: Query MDM/inventory tool (Jamf, Intune, osquery). If unavailable: manually query each macOS system via SSH with `which npm node && npm list -g --depth=0 && find ~ -name '.openai*' -o -name '.openclawrc*' 2>/dev/null`. Cross-reference against GitHub Actions runner configuration files (.github/workflows/*.yml) to identify CI/CD machines. Search Slack/Teams channels and email for 'npm install', 'postinstall', or 'Node.js' mentions in past 90 days to identify affected user base. Check credential manager (1Password, LastPass, Vault) audit logs for access from developer machines.

Evidence: Maintain running inventory spreadsheet: system hostname, IP, macOS version, npm version, Node.js version, installed global packages, GitHub runner role, access to secrets manager, access to code signing keys, cryptocurrency wallet presence. Preserve each system's last 90 days of shell history, npm cache logs, and GitHub Actions workflow runs (via API export). Capture AWS/GCP service account key locations and permissions via `find ~ -name '*key*.json' -o -name '*.pem'`. Document network access to artifact repositories and package managers (Artifactory, Nexus).

Step 6 — Communication: Notify development teams to halt installation of npm packages from unverified sources and to treat any recent sudo prompts during package install as a potential compromise indicator. Escalate affected system owners to incident response.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.2.2 (containment: communication and coordination)

Controls: NIST 800-53 IR-1 (incident response policy), NIST 800-53 IR-6 (incident reporting), CIS 17.1 (security incident management program)

Compensating: Draft incident notification email template including: (1) list of specific malicious package names and 'mikilanjillo' account; (2) exact symptoms ('sudo prompt during npm install'; 'unusual UI dialogs'); (3) immediate action ('stop all npm installs; verify package sources via package-lock.json'); (4) escalation path (direct to IR team with contact info). Post notification in Slack #security channel pinned message. Schedule all-hands async video explaining threat in non-technical terms (10 min max). Create internal wiki page with FAQ, remediation steps, and forensic contact info. Send targeted email to CI/CD pipeline owners and DevOps team separately with elevated priority.

Evidence: Before communication: finalize list of confirmed affected systems and developers. Prepare evidence summary document for distribution (sanitized for disclosure). Record communication timestamp and recipient list. Capture Slack/email delivery confirmation. Monitor incident response ticket queue for inbound reports of additional affected systems post-notification.

Step 7 — Long-term: Enforce package integrity verification (npm audit signatures, lockfile pinning, private registry mirroring). Implement controls that prevent npm install scripts from spawning GUI dialogs or prompting for elevated credentials. Evaluate secrets scanning across developer endpoints and CI/CD systems.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.4 (post-incident activities: lessons learned and prevention)

Controls: NIST 800-53 SI-7 (information system monitoring: software, firmware, and information integrity), NIST 800-53 CM-5 (access restrictions for change), CIS 6.3 (secure software development practices)

Compensating: Mandate use of npm-audit-signature verification in CI pipelines via npm config set audit-level moderate. Enable npm lockfile integrity via `npm ci` instead of `npm install` (respects package-lock.json). Deploy private npm registry mirror (Verdaccio free tier or Nexus OSS) and restrict public registry access via firewall rules. Disable postinstall scripts by default: `npm config set ignore-scripts true` in .npmrc. Block sudo prompts during CI runs

by enforcing non-root execution contexts and pre-approving required privileges (e.g., brew install via sudo in Homebrew.bundle). Implement mandatory secrets scanning: use git-secrets, TruffleHog, or OWASP Dependency-Check on developer machines and in pre-commit hooks. Audit ~/.npm/rc, ~/.npmrc, and GitHub Actions secrets vault quarterly. Enforce code review of all package.json changes.

Evidence: Document baseline controls in security policy wiki with rationale. Create pre/post-implementation metrics: npm audit failure rate, lockfile compliance, postinstall script execution count, secrets detected. Maintain change log of registry proxy rules (Artifactory/Nexus) blocking specific publishers. Store secrets scanning reports (monthly). Archive all CI/CD pipeline logs showing npm install success/failure rates post-remediation.

Detection Guidance

Behavioral indicators: (1) npm or node processes initiating outbound HTTPS to api.telegram.org, not expected in normal development workflows; (2) macOS GUI dialogs requesting sudo credentials launched as a child process of npm install or a shell script; (3) node or Python processes reading from ~/Library/Keychains/, ~/Library/Application Support/Google/Chrome/Default/Login Data, or common crypto wallet paths (e.g., ~/.electrum, ~/.exodus); (4) unexpected cron jobs, LaunchAgents, or LaunchDaemons created in /Library/LaunchAgents/ or ~/Library/LaunchAgents/ following package installation. Log sources: macOS Unified Log (process spawn events), EDR telemetry (file access patterns, network connections by process), npm audit logs, GitHub Actions runner logs for dependency installation steps. Query pattern (EDR/SIEM, adapt to platform): process_name IN ('node','npm','python3') AND network_destination CONTAINS 'api.telegram.org'. IOC note: Specific package names, hashes, and IP/domain IOCs were not embedded in the structured source data provided. Retrieve comprehensive IOC lists directly from ReversingLabs and JFrog published campaign research before operationalizing detection rules. Validate current IOC lists against vendor research for this campaign.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`api.telegram.org`	Telegram bot C2 channel used for credential and wallet data exfiltration — flag outbound connections from developer workstations	HIGH
URL	`npm publisher account: mikilanjillo`	Identified npm publisher account used to distribute malicious packages in this campaign — block and audit any packages from this publisher	HIGH
DOMAIN	`Binance Smart Chain RPC nodes (public BSC endpoints)`	Used to store and retrieve affiliate configuration via smart contracts — unexpected BSC RPC calls from developer hosts warrant investigation	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1555.003** — Credentials from Web Browsers
- **T1552.004** — Private Keys
- **T1027** — Obfuscated Files or Information
- **T1140** — Deobfuscate/Decode Files or Information
- **T1105** — Ingress Tool Transfer
- **T1204.002** — Malicious File
- **T1566** — Phishing
- **T1098** — Account Manipulation
- **T1555** — Credentials from Password Stores
- **T1056.002** — GUI Input Capture
- **T1059.007** — JavaScript
- **T1071.001** — Web Protocols
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1059.006** — Python
- **T1552.001** — Credentials In Files
- **T1041** — Exfiltration Over C2 Channel
- **T1036.005** — Match Legitimate Resource Name or Location

NIST-800-53R5

- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **AT-2** — Literacy Training and Awareness
- **SI-8** — Spam Protection
- **IA-5** — Authenticator Management
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A04:2021** — Insecure Design
- **A07:2021** — Identification and Authentication Failures
- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **5.2**
- **2.5**
- **2.6**
- **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

HIPAA-SECURITY

- **164.308(a)(5)(ii)(D)** — Password Management
- **164.312(d)** — Person or Entity Authentication
- **164.308(a)(5)(i)** — Security Awareness and Training

SOC2-TSC

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

ISO-27001-2022

- **A.5.34** — Privacy and protection of personal information
- **A.5.21** — Managing information security in the ICT supply chain

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1555.003	Credentials from Web Browsers	Credential-Access
T1552.004	Private Keys	Credential-Access
T1027	Obfuscated Files or Information	Defense-Evasion
T1140	Deobfuscate/Decode Files or Information	Defense-Evasion
T1105	Ingress Tool Transfer	Command-And-Control
T1204.002	Malicious File	Execution
T1566	Phishing	Initial-Access
T1098	Account Manipulation	Persistence
T1555	Credentials from Password Stores	Credential-Access
T1056.002	GUI Input Capture	Collection
T1059.007	JavaScript	Execution
T1071.001	Web Protocols	Command-And-Control
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access

Technique ID	Technique Name	Tactic
T1059.006	Python	Execution
T1552.001	Credentials In Files	Credential-Access
T1041	Exfiltration Over C2 Channel	Exfiltration
T1036.005	Match Legitimate Resource Name or Location	Defense-Evasion

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/03/ghost-campaign-uses-7-npm-package...	T3
New Malware Targets macOS Crypto Wallets - Binance	https://www.binance.com/en/square/post/304490160578577	T3
Compromised npm package silently installs OpenClaw on ...	https://www.csoonline.com/article/4135449/compromised-npm-package-s...	T3
OpenClaw AI Attack via GitHub Issue Title | Rohit Tamma posted on ...	https://www.linkedin.com/posts/rohittamma_cybersecurity-infosec-app...	T3
OpenClaw Security Crisis: The npm Nightmare Returns in AI Agent Era	https://stable-learn.com/en/openclaw-security-supply-chain-attack/	T3

Generated 2026-03-29 18:34 UTC by TJS Security Command Center