INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-03-31 06:19 UTC

Axios npm Supply Chain Compromise: Premeditated RAT Deployment Targets 83M Weekly Downloads

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0128
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	Axios npm package v1.14.1 and v0.30.4; plain-crypto-js v4.2.1; @shadanai/openclaw (multiple versions); @qqbrowser/openclaw-qbot v0.0.130; Node.js environments on macOS, Windows, Linux
Published	2026-03-31T02:08:00
Discovery Source	Rss

Executive Summary

Threat actors compromised the npm account of the primary Axios maintainer and published two malicious versions of the Axios package (v1.14.1 and v0.30.4), injecting a cross-platform Remote Access Trojan across both the stable and legacy release branches. Axios reportedly handles approximately 83 million weekly downloads (per The Hacker News), meaning any organization whose Node.js build pipeline pulled either affected version during the exposure window should treat those environments as fully compromised. The business risk is severe: successful exploitation grants attackers persistent remote access, credential theft capability, and the ability to destroy forensic evidence, complicating breach investigation and regulatory disclosure decisions. Note: This assessment is based on T3 sources (news and vendor blogs) only. Verification against official Axios advisories and CISA guidance is strongly recommended before operational decisions.

Technical Analysis

Attack vector: supply chain compromise via npm account takeover (account 'nrwise', nrwise@proton.me). Two malicious versions were published: v1.14.1 (stable branch) and v0.30.4 (legacy branch), both within a 39-minute window. Payloads were staged approximately 18 hours before publication, suggesting planned coordination. The dependency chain delivering the RAT ran through three intermediate packages: plain-crypto-js v4.2.1, @shadanai/openclaw (multiple versions), and @qqbrowser/openclaw-qbot v0.0.130. The RAT operates

cross-platform across macOS, Windows, and Linux Node.js environments. Post-execution, the malware actively deleted forensic artifacts (T1070.004, Indicator Removal: File Deletion), complicating post-compromise investigation. No CVE has been assigned as of this item's publication. Relevant CWEs: CWE-494 (Download of Code Without Integrity Check), CWE-506 (Embedded Malicious Code), CWE-693 (Protection Mechanism Failure), CWE-798 (Use of Hard-coded Credentials). MITRE ATT&CK techniques observed include T1195.002 (Compromise Software Supply Chain), T1105 (Ingress Tool Transfer), T1059 family (Command and Scripting Interpreter, PowerShell, AppleScript, Python), T1071 (Application Layer Protocol), T1082 (System Information Discovery), T1552.001 (Credentials in Files), T1543 (Create or Modify System Process), T1036.005 (Match Legitimate Name or Location), T1078 (Valid Accounts), and T1070.004 (File Deletion). No clean versions beyond reverting to v1.13.x (stable) or v0.29.x (legacy) are confirmed safe at this time, verify against the official Axios npm registry advisory before upgrading. All source URLs require human validation before operational reliance; source quality score is 0.56 (T3 sources only).

Action Checklist

1. Step 1: Containment. Immediately audit all Node.js build pipelines, CI/CD environments, and developer workstations for installations of axios v1.14.1 or v0.30.4, plain-crypto-js v4.2.1, @shadanai/openclaw (any version), or @qqbrowser/openclaw-qbot v0.0.130. Run 'npm list axios' and 'npm list --all' in affected project roots. Isolate any host that installed these packages during the exposure window from the network pending investigation. Treat all secrets (API keys, tokens, credentials) accessible from those environments as compromised. Immediately initiate credential rotation for all secrets accessible to the affected environments; do not wait for Step 4.
2. Step 2: Detection. Query package-lock.json, yarn.lock, and npm audit logs across repositories for the four malicious packages listed above. Search CI/CD pipeline logs for install events referencing those package names and versions. Check EDR and endpoint logs for anomalous outbound connections from Node.js processes, unexpected child process spawns (PowerShell, AppleScript, Python), and file deletion activity in temp or cache directories post-npm install. Look for persistence mechanisms: new scheduled tasks (Windows), LaunchAgents/LaunchDaemons (macOS), or systemd service entries (Linux) created around the time of a suspect install. IOC patterns include network callbacks from Node.js processes to unknown external IPs and credential file access from npm script contexts.
3. Step 3: Eradication. Remove axios v1.14.1 and v0.30.4 and all four intermediate malicious packages from all environments. Pin axios to a previously-released version (v1.13.x stable or v0.29.x legacy) pending verification against the official Axios npm registry advisory. Validate the chosen safe version hash against the npm registry before deployment. Remove any persistence artifacts (scheduled tasks, LaunchAgents, systemd services) created by the RAT. Reimage hosts where active RAT execution is confirmed rather than attempting in-place cleanup given the malware's anti-forensic behavior.
4. Step 4: Recovery. After reimaging or cleaning affected hosts, rotate all credentials, API keys, tokens, and secrets that were accessible from the compromised environment, including CI/CD secrets, cloud provider credentials, database passwords, and code signing keys. Validate that no malicious packages remain in artifact caches (npm cache, private registries, Docker layers). Restore from known-good backups where available. Monitor outbound traffic from rebuilt environments for at least 72 hours post-recovery for signs of reinfection or residual C2 activity.
5. Step 5: Post-Incident. This attack exploited the absence of npm package integrity verification and unrestricted third-party dependency resolution in CI/CD pipelines. Control improvements to implement: (1) enforce npm package integrity verification using lock files and Subresource Integrity checks; (2) implement

a private npm mirror or registry proxy that requires security review before new packages or version updates enter the pipeline; (3) apply least-privilege principles to CI/CD environments, secrets should not be accessible to npm install steps without explicit scoping; (4) adopt SLSA (Supply-chain Levels for Software Artifacts) framework guidance for build provenance; (5) configure alerts on new or updated dependency versions in package-lock.json as part of the PR review process. Map control gaps to NIST SP 800-161 (Cybersecurity Supply Chain Risk Management) and CISA's Secure Software Development Framework (SSDF) and Supply Chain Risk Management guidance.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and executive leadership immediately if forensic analysis confirms the RAT established an active C2 channel, if any credentials exfiltrated from the compromised CI/CD environment were used to access production systems or cloud infrastructure, or if code signing keys were in scope — the latter triggers mandatory breach notification assessment under applicable regulations (GDPR, state breach notification laws) given the potential for downstream supply chain impact to customers receiving signed artifacts built during the exposure window.
Recovery Notes	After rebuilding compromised hosts, validate that all Docker images, container registries, and artifact stores (Artifactory, Nexus, GitHub Packages) built during the exposure window are either deleted or rebuilt from source using a verified clean dependency tree — cached Docker layers containing node_modules from the malicious axios versions represent a reinfection vector if those images are redeployed. Monitor all rebuilt CI/CD environments and developer workstations for outbound Node.js process connections for a minimum of 72 hours using network flow logs or tcpdump captures, specifically watching for beaconing patterns (regular interval connections to non-CDN external IPs) that would indicate a residual RAT component surviving the rebuild. Do not restore from backups taken during the exposure window without first verifying that the backup predates the first recorded install of axios v1.14.1 or v0.30.4 using git history and CI pipeline logs.

Forensic Artifacts	npm debug and timing logs at ~/.npm/_logs/ (developer workstations) and /root/.npm/_logs/ (CI runners) — these record exact timestamps and resolution sequences for axios v1.14.1 and v0.30.4 installs, establishing the precise exposure window for each host | node_modules/axios/package.json and node_modules/axios/index.js on any host that installed the malicious versions — the injected RAT payload would be embedded in or loaded by these files, and their SHA-256 hashes will not match the legitimate axios package checksums in the npm registry | Scheduled tasks (Windows: schtasks /query /fo XML /v), LaunchAgent/LaunchDaemon plist files (macOS: ~/Library/LaunchAgents/, /Library/LaunchDaemons/), and systemd service unit files (Linux: /etc/systemd/system/) with creation timestamps correlating to npm install events, indicating RAT persistence established via a malicious postinstall script in axios v1.14.1, v0.30.4, or @shadanai/openclaw | CI/CD pipeline job logs (GitHub Actions run logs, Jenkins console output, GitLab CI job traces) showing the specific pipeline execution that invoked npm install and resolved to the malicious axios version — these establish which build artifacts, Docker images, and deployed applications were built with the compromised dependency and are therefore untrusted | Network flow logs or pcap captures showing outbound TCP connections originating from node or node.exe processes to external IPs during and after the exposure window — the cross-platform RAT in this campaign establishes C2 callbacks from within the Node.js process context, so connections from node to non-CDN, non-registry external IPs are high-fidelity IOCs specific to this compromise

Per-Action IR Details

Step 1: Containment — Immediately audit all Node.js build pipelines, CI/CD environments, and developer workstations for installations of axios v1.14.1 or v0.30.4, plain-crypto-js v4.2.1, @shadanai/openclaw (any version), or @qqbrowser/openclaw-qbot v0.0.130. Run 'npm list axios' and 'npm list --all' in affected project roots. Isolate any host that installed these packages during the exposure window from the network pending investigation. Treat all secrets (API keys, tokens, credentials) accessible from those environments as compromised.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST IR-5 (Incident Monitoring), NIST CM-8 (System Component Inventory), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: On each Node.js host or CI runner, execute: `npm list axios --depth=0 2>/dev/null` and `npm list --all 2>/dev/null | grep -E 'axios@1\.14\.1|axios@0\.30\.4|plain-crypto-js@4\.2\.1|@shadanai/openclaw|@qqbrowser/openclaw-qbot@0\.0\.130'`. On Windows CI agents, additionally run `Get-ChildItem -Recurse -Path C:\ -Filter package-lock.json | Select-String 'axios.*1\.14\.1|axios.*0\.30\.4'` in PowerShell. For network isolation without NAC, apply a host-based firewall block: `iptables -I OUTPUT -j DROP` (Linux) or `netsh advfirewall set allprofiles firewallpolicy blockinbound,blockoutbound` (Windows) on confirmed hosts while preserving access to the management console.

Evidence: Before isolating, capture a full snapshot of: (1) `npm list --all --json > npm_tree_$(hostname)_$(date +%s).json` to preserve the exact dependency tree showing axios v1.14.1 or v0.30.4 and their relationship to plain-crypto-js v4.2.1 and the openclaw packages; (2) the node_modules/.cache directory contents and any .npm/_npx cache entries on developer workstations which may contain extracted malicious package tarballs; (3) active network connections at time of discovery via `ss -tunap` (Linux), `netstat -anob` (Windows), or `lsof -i` (macOS) to capture any live C2 channels established by the RAT; (4) running process list with parent-child relationships (`ps auxf` on Linux/macOS, `Get-WmiObject Win32_Process | Select ProcessId,ParentProcessId,Name,CommandLine` on Windows) to identify Node.js processes that may have spawned secondary payloads.

Step 2: Detection — Query package-lock.json, yarn.lock, and npm audit logs across repositories for the four malicious packages listed above. Search CI/CD pipeline logs for install events referencing those package names and versions. Check EDR and endpoint logs for anomalous outbound connections from Node.js processes, unexpected child process spawns (PowerShell, AppleScript, Python), and file deletion activity in temp or cache directories post-npm install. Look for persistence mechanisms: new scheduled tasks (Windows), LaunchAgents/LaunchDaemons (macOS), or systemd service entries (Linux) created around the time of a suspect install. IOC patterns include network callbacks from Node.js processes to unknown external IPs and credential file access from npm script contexts.

NIST Phase: Detection Analysis

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

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

Compensating: Use osquery to hunt across all endpoints with: `SELECT pid, name, cmdline, parent FROM processes WHERE name IN ('node', 'node.exe') AND cmdline LIKE '%axios%'` and `SELECT * FROM scheduled_tasks WHERE action LIKE '%node%' OR action LIKE '%npm%'` (Windows). On Linux/macOS, use auditd or syslog to find child processes spawned by node: `grep -E 'PPID.*node|node.*exec' /var/log/audit/audit.log`. Deploy the following Sigma rule concept against syslog or Windows Event Log: detect Event ID 4688 (Process Creation) where ParentImage contains 'node.exe' and ChildImage is 'powershell.exe', 'cmd.exe', 'python.exe', or 'osascript'. For repository scanning, run `grep -rl 'axios.*1\.14\.1\|axios.*0\.30\.4\|plain-crypto-js.*4\.2\.1\|@shadanai/openclaw\|openclaw-qbot' $(find . -name 'package-lock.json' -o -name 'yarn.lock')` across all repo clones. Check LaunchAgents on macOS via `ls -la ~/Library/LaunchAgents/ /Library/LaunchAgents/ /Library/LaunchDaemons/` and correlate timestamps against npm install events in `~/.npm/_logs/`.

Evidence: Collect before any remediation: (1) all package-lock.json and yarn.lock files from affected repositories preserving original timestamps via `cp --preserve=timestamps`; (2) npm debug and timing logs from `~/.npm/_logs/` on developer machines and `/root/.npm/_logs/` on CI runners, which record exact install timestamps and package resolution sequences for axios v1.14.1 or v0.30.4; (3) CI/CD pipeline job logs (GitHub Actions logs, Jenkins build console output, GitLab CI job traces) showing the specific pipeline run that executed `npm install` pulling the malicious axios version — these establish the exposure window; (4) on Windows, export the Scheduled Tasks XML via `schtasks /query /fo XML /v > schtasks_export.xml` and check the HKCU\Software\Microsoft\Windows\CurrentVersion\Run and HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run registry hives for Node.js-related persistence entries added during the exposure window; (5) on macOS, collect plist files from LaunchAgents/LaunchDaemons directories with `sudo plutil -convert xml1` for analysis; (6) on Linux, run `systemctl list-units --type=service --state=enabled > enabled_services_$(date +%s).txt` and compare against a known-good baseline to identify systemd services created by the RAT postinstall script.

Step 3: Eradication — Remove axios v1.14.1 and v0.30.4 and all four intermediate malicious packages from all environments. Pin axios to a known-good prior version (v1.13.x stable or v0.29.x legacy) until the official Axios maintainers publish a verified clean release and npm confirms account security is restored. Validate the chosen safe version hash against the npm registry before deployment. Remove any persistence artifacts (scheduled tasks, LaunchAgents, systemd services) created by the RAT. Reimage hosts where active RAT execution is confirmed rather than attempting in-place cleanup given the malware's anti-forensic behavior.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST SI-3 (Malicious Code Protection), NIST SI-7 (Software, Firmware, and Information Integrity), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Before pinning, verify the integrity of the target safe axios version by comparing its npm registry shasum: `npm view axios@1.7.9 dist.integrity` — confirm the returned sha512 hash matches the value in your package-lock.json after `npm install axios@1.7.9`. Use `npm install axios@1.7.9 --save-exact` to enforce the pin. To

remove all four malicious packages: `npm uninstall axios plain-crypto-js @shadanai/openclaw @qqbrowser/openclaw-qbot && npm cache clean --force`. For persistence removal on Windows without EDR, use: `schtasks /delete /tn "" /f` after identifying the malicious task name from the schtasks export; remove registry run keys with `reg delete HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v "" /f`. On Linux, `systemctl disable --now && rm /etc/systemd/system/.service && systemctl daemon-reload`. For hosts with confirmed RAT execution, use a pre-built clean OS image rather than in-place removal — the cross-platform RAT's anti-forensic file deletion behavior (noted in the advisory) means in-place cleanup cannot be trusted.

Evidence: Before eradicating, preserve forensic images of: (1) the full node_modules directory tree from any confirmed-compromised environment using `tar czf node_modules_evidence_$(hostname)_$(date +%s).tar.gz node_modules/` — this preserves the malicious package files including any injected RAT payload dropped by the axios postinstall script before they are deleted; (2) memory dump of any running Node.js process that loaded the malicious axios version using WinPmem (Windows), LiME kernel module (Linux), or osxpmem (macOS) — the cross-platform RAT may reside only in memory if anti-forensic cleanup already ran on disk; (3) the npm postinstall script execution artifacts in system temp directories (`%TEMP%` on Windows, `/tmp` on Linux/macOS) including any dropped binaries, scripts, or configuration files written by the malicious axios or openclaw postinstall hooks.

Step 4: Recovery — After reimaging or cleaning affected hosts, rotate all credentials, API keys, tokens, and secrets that were accessible from the compromised environment — including CI/CD secrets, cloud provider credentials, database passwords, and code signing keys. Validate that no malicious packages remain in artifact caches (npm cache, private registries, Docker layers). Restore from known-good backups where available. Monitor outbound traffic from rebuilt environments for at least 72 hours post-recovery for signs of reinfection or residual C2 activity.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST IA-5 (Authenticator Management), NIST SI-2 (Flaw Remediation), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 5.2 (Use Unique Passwords), CIS 6.2 (Establish an Access Revoking Process), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: For credential rotation verification without a secrets manager, use: GitHub — `gh auth token` to confirm old token invalidation and generate a new one; AWS — `aws iam list-access-keys --user-name ` to enumerate and then `aws iam delete-access-key` to revoke keys that were in scope; for Docker layer inspection use `docker history --no-trunc ` and `dive ` (free tool) to identify layers built during the exposure window that may contain cached node_modules with malicious packages. To purge npm private registry or Verdaccio cache: remove the package tarballs for `axios-1.14.1.tgz`, `axios-0.30.4.tgz`, `plain-crypto-js-4.2.1.tgz` from the registry storage directory. Monitor post-rebuild outbound traffic using Wireshark or `tcpdump -i any -w recovery_monitor_$(date +%s).pcap 'src host '` and inspect for Node.js process-originated connections to non-internal, non-CDN destinations.

Evidence: Document the recovery state by capturing: (1) `npm audit --json > post_recovery_audit_$(date +%s).json` on rebuilt environments to confirm zero findings for the four malicious packages; (2) a fresh `npm list --all --json` from the rebuilt environment to confirm axios is pinned to the verified safe version; (3) outbound connection logs from rebuilt CI/CD runners for the 72-hour monitoring window, specifically filtering for TCP connections originating from node or node.exe processes to non-internal IP ranges, which would indicate residual RAT C2 beaconing or reinfection via a poisoned artifact cache.

Step 5: Post-Incident — This attack exploited the absence of npm package integrity verification and unrestricted third-party dependency resolution in CI/CD pipelines. Control improvements to implement: (1) enforce npm package integrity verification using lock files and Subresource Integrity checks; (2) implement a private npm mirror or registry proxy that requires security review before new packages or version updates enter the pipeline; (3) apply least-privilege principles to CI/CD environments — secrets should not be accessible to npm install steps without explicit scoping; (4) adopt SLSA (Supply-chain Levels for Software Artifacts) framework guidance for build provenance; (5) configure alerts on new or updated dependency versions in package-lock.json as part of the PR review process. Map control gaps to NIST SP 800-161 (Cybersecurity Supply Chain Risk Management) and CISA's Secure Software Development guidance.

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-7 (Software, Firmware, and Information Integrity), NIST SA-12 (Supply Chain Protection), NIST CM-3 (Configuration Change Control), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.3 (Address Unauthorized Software)

Compensating: Implement the following without enterprise tooling: (1) Add `.npmrc` to all project roots with `save-exact=true` and `package-lock=true` to enforce lock file integrity; add `npm ci` (clean install) instead of `npm install` in all CI pipeline definitions — `npm ci` fails if package-lock.json is absent or mismatched, preventing silent version resolution. (2) Deploy Verdaccio (free, self-hosted npm proxy) as a private registry mirror; configure it to whitelist only reviewed packages and block the four malicious package names via its `packages` ACL config. (3) Scope CI/CD secrets using environment-level restrictions in GitHub Actions (`environment: production` with required reviewers) or GitLab CI protected variables — ensure `npm install` jobs run in an isolated stage with no access to deployment secrets. (4) Add a pre-commit hook or CI step using `node -e "const lock=require('./package-lock.json'); const a=lock.packages['node_modules/axios']; if(!a||!a.integrity)process.exit(1);"` to fail builds missing integrity hashes. (5) Write a Sigma rule to alert on any PR diff that modifies package-lock.json with a new axios or transitive dependency version, routed to a Slack webhook or email via a GitHub Actions workflow.

Evidence: For the lessons-learned record, preserve: (1) the complete timeline of exposure window — first CI run that pulled axios v1.14.1 or v0.30.4 through the run that was remediated — reconstructed from CI/CD pipeline logs and package-lock.json git history via `git log --all --follow -p package-lock.json | grep -A2 'axios'`; (2) an inventory of all secrets, keys, and credentials that were in scope during the exposure window, documenting which were rotated and when, to satisfy audit requirements under NIST AU-11 (Audit Record Retention) and demonstrate completeness of the credential rotation; (3) the pre- and post-incident npm audit output files to document the control gap (no integrity enforcement) and its closure as evidence for any compliance review or supply chain risk assessment.

Detection Guidance

Primary detection focus: identify installs of axios v1.14.1, v0.30.4, plain-crypto-js v4.2.1, @shadanai/openclaw, or @qqbrowser/openclaw-qbot v0.0.130 across all Node.js environments. Audit package-lock.json and yarn.lock files in all repositories. In CI/CD logs, search for npm install output referencing those exact version strings. On endpoints, look for: (1) outbound network connections originating from node processes to non-standard external hosts, especially on non-HTTP/S ports; (2) child processes spawned by npm scripts, PowerShell (Windows, T1059.001), osascript/AppleScript (macOS, T1059.002), Python (T1059.006), that were not expected in the normal build; (3) file deletion activity in temp directories immediately following npm install (T1070.004); (4) new autorun/persistence entries (Registry Run keys on Windows, LaunchAgent plist files on macOS, systemd unit files on Linux) created within the install window. In SIEM, correlate npm install timestamps with any subsequent outbound connections from the same host. EDR behavioral rules should flag: node.exe or node spawning cmd.exe, powershell.exe, or python.exe as child processes in CI/CD build agent contexts. Note: because the malware actively deletes artifacts post-execution, absence of log evidence does not confirm a clean system, treat any confirmed install of the affected versions as requiring full compromise investigation regardless of visible indicators.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`nrwise@proton.me (account identifier)`	npm account used to publish malicious Axios versions; not a network IOC but a registry-level indicator for npm audit trails	HIGH
URL	`https://www.npmjs.com/package/axios/v/1.14.1`	Malicious Axios stable branch version on npm registry — do not install; verify current registry status before action	HIGH
URL	`https://www.npmjs.com/package/axios/v/0.30.4`	Malicious Axios legacy branch version on npm registry — do not install; verify current registry status before action	HIGH
URL	`https://www.npmjs.com/package/plain-crypto-js/v/4.2.1`	Intermediate payload delivery package in the malicious dependency chain	HIGH
URL	`https://www.npmjs.com/package/@qqbrowser/openclaw-qbot/v/0.0.130`	Intermediate payload staging package in the malicious dependency chain	HIGH
URL	`https://www.npmjs.com/package/@shadanai/openclaw`	Intermediate package in the malicious dependency chain; multiple versions flagged — treat all versions as suspect pending official clearance	HIGH

Framework Mappings

MITRE-ATTACK

- **T1059.001** — PowerShell
- **T1071.001** — Web Protocols
- **T1552.001** — Credentials In Files
- **T1059.002** — AppleScript
- **T1071** — Application Layer Protocol
- **T1082** — System Information Discovery
- **T1195.002** — Compromise Software Supply Chain
- **T1105** — Ingress Tool Transfer
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1059** — Command and Scripting Interpreter
- **T1078** — Valid Accounts
- **T1543** — Create or Modify System Process
- **T1059.006** — Python
- **T1070.004** — File Deletion
- **T1036.005** — Match Legitimate Resource Name or Location

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-7** — Software, Firmware, and Information Integrity
- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

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

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **16.10** — Apply Secure Design Principles in Application Architectures
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers

ISO-27001-2022

- **A.8.28** — Secure coding
- **A.5.21** — Managing information security in the ICT supply chain

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1059.001	PowerShell	Execution
T1071.001	Web Protocols	Command-And-Control
T1552.001	Credentials In Files	Credential-Access
T1059.002	AppleScript	Execution
T1071	Application Layer Protocol	Command-And-Control
T1082	System Information Discovery	Discovery
T1195.002	Compromise Software Supply Chain	Initial-Access
T1105	Ingress Tool Transfer	Command-And-Control
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1059	Command and Scripting Interpreter	Execution
T1078	Valid Accounts	Defense-Evasion
T1543	Create or Modify System Process	Persistence
T1059.006	Python	Execution
T1070.004	File Deletion	Defense-Evasion
T1036.005	Match Legitimate Resource Name or Location	Defense-Evasion

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/03/axios-supply-chain-attack-pushes-...	T3
Axios npm Supply Chain Attack (31 March 26) - VibeAudits.com	https://vibeaudits.com/blog/axios-npm-supply-chain-attack-31-march-...	T3
Supply chain attack hits npm package with 45,000 weekly downloads	https://www.bleepingcomputer.com/news/security/supply-chain-attack-...	T3
Popular npm package compromised with RAT in supply chain attack	https://www.scworld.com/brief/popular-npm-package-compromised-with-...	T3

Source	URL	Tier
Malicious npm Packages Harvest Crypto Keys, CI Secrets, and API ...	https://thehackernews.com/2026/02/malicious-npm-packages-harvest-cr...	**T3**

Generated 2026-03-31 06:19 UTC by TJS Security Command Center