INTELLIGENCE BRIEFING
Security Command Center

TLP:CLEAR
2026-03-31 06:19 UTC

Supply Chain Attack: Malicious Axios npm Packages (v1.14.1, v0.30.4) Deliver Cross-Platform RAT

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.0

SCC Item ID	SCC-CAM-2026-0129
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.0
Affected Products	Axios npm library v1.14.1 and v0.30.4
Published	2026-03-31
Discovery Source	Gemini

Executive Summary

Threat actors compromised an Axios npm maintainer account and published two trojanized versions of the Axios HTTP library (v1.14.1 and v0.30.4), each silently installing a cross-platform Remote Access Trojan on any machine that ran npm install during the exposure window. Axios receives over 50 million weekly downloads, meaning software development pipelines, CI/CD systems, and developer workstations across Windows, macOS, and Linux are potentially affected. Any organization whose developers or build systems installed either version during the compromise window should treat affected machines as fully compromised pending investigation.

Technical Analysis

An npm account takeover (CWE-287: Improper Authentication) against an Axios maintainer enabled publication of axios@1.14.1 and axios@0.30.4 to the public npm registry. Both versions introduced a malicious dependency that executed a cross-platform Remote Access Trojan (T1219: Remote Access Software) at install time via a post-install script (T1059: Command and Scripting Interpreter). The attack constitutes a supply chain compromise at the dependency injection layer (T1195.001: Compromise Software Dependencies and Development Tools; CWE-1395: Dependency on Vulnerable Third-Party Component). The RAT deletes its binary post-execution (T1070.004: File Deletion) to reduce forensic visibility, complicating post-compromise detection. The packages were also flagged as containing embedded malicious code (CWE-506: Embedded Malicious Code). The official Axios GitHub issue tracker (issue #10604) confirms the compromise. No CVE has been assigned as of 2026-03-04. Critical severity reflects full remote access capability with broad blast radius across development infrastructure. EPSS data is not applicable to this item type. The safe versions are the previously published stable releases prior to v1.14.1 and v0.30.4, specifically, v1.7.x (latest stable 1.x line) and v0.29.x or v0.28.x for the 0.x line. Developers should verify the current safe release via the official Axios GitHub repository.

Action Checklist

1. Step 1: Containment, Identify all systems (developer workstations, CI/CD runners, build containers, staging environments) that executed npm install within the exposure window and may have pulled axios@1.14.1 or axios@0.30.4. Isolate those systems from production networks immediately. Check package-lock.json, yarn.lock, and npm audit output across all active repositories and pipelines for either version. Block further installation of both versions at the npm proxy or artifact repository layer (Artifactory, Nexus, or equivalent) if one is in use.
2. Step 2: Detection, Query endpoint detection and response (EDR) telemetry on potentially affected hosts for anomalous outbound connections, process spawns originating from Node.js or npm processes, and any post-install script execution artifacts around the timeframe of axios installation. Check npm install logs and CI/CD pipeline logs for evidence of either compromised version being fetched. Because the RAT deletes its binary post-execution (T1070.004), absence of a malicious binary does not confirm a clean system. Look for network telemetry indicating C2 communication, credential access attempts, or lateral movement from affected hosts. Review DNS query logs and firewall logs for unusual external connections from build systems or developer machines.
3. Step 3: Eradication, Remove axios@1.14.1 and axios@0.30.4 from all environments. Update to the verified safe version per the official Axios GitHub repository (confirm the current safe release at https://github.com/axios/axios/releases before updating). Clear the npm cache on affected systems (npm cache clean --force). Rebuild affected containers and pipeline environments from clean base images to remove any residual persistence mechanisms.
4. Step 4: Recovery, After remediation, validate that no persistence mechanisms remain: review scheduled tasks (Windows), cron jobs (Linux/macOS), startup items, and shell profile modifications on affected hosts. Re-scan affected systems with EDR tooling post-remediation. Monitor outbound network traffic from previously affected hosts for 30 days for signs of residual C2 activity. Rotate credentials, tokens, and SSH keys stored on or accessible from affected developer machines, CI/CD runners, and build environments, as the RAT may have exfiltrated them during its execution window.
5. Step 5: Post-Incident, Analysis of this attack indicates it exploited weak npm account security controls (lack of MFA enforcement on maintainer accounts) and insufficient dependency integrity verification in build pipelines. Immediate control improvements: enforce MFA on all npm accounts with publish rights to packages your organization maintains or depends on; implement package integrity verification (npm audit signatures or equivalent) in CI/CD pipelines; evaluate use of a private npm proxy that pins approved versions and blocks unapproved new versions from reaching build systems automatically; add software composition analysis (SCA) tooling to the pipeline to alert on newly published versions of critical dependencies before they are consumed.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate to CISO and legal counsel immediately if forensic evidence (DNS logs, Sysmon network events, firewall egress logs) shows confirmed outbound C2 communication from any host with access to production secrets, customer PII, PHI, or payment card data, as this may trigger breach notification obligations under GDPR Article 33, HIPAA §164.410, or PCI DSS Requirement 12.10.4, or if any npm publish tokens accessible from affected CI/CD environments have not yet been rotated and could enable a secondary supply chain attack against your organization's own published packages.
Recovery Notes	After eradicating axios@1.14.1 and axios@0.30.4 and rebuilding affected environments from clean images, conduct a 30-day enhanced monitoring period on all previously affected hosts and CI/CD runners, specifically watching for outbound connections from node.exe or CI runner processes to non-approved external IPs, re-appearance of scheduled tasks or cron entries not present in your configuration management baseline, and any npm publish operations originating from accounts whose tokens were accessible on compromised systems. Verify that all rotated credentials (npm tokens, GitHub PATs, AWS IAM keys, SSH keypairs) have had their predecessors fully invalidated — confirm in each platform's audit log that the old credentials show no usage after rotation. Do not return previously affected CI/CD runners to production pipeline use until they have been rebuilt from a clean base image, their outbound network traffic has been verified clean for at least 72 hours, and a post-remediation EDR or osquery scan confirms no persistence artifacts remain.
Forensic Artifacts	npm cache tarball directory (~/.npm/_cacache/ on Linux/macOS, %AppData%\npm-cache_cacache\ on Windows) — the malicious axios@1.14.1 or axios@0.30.4 .tgz package may persist here even after npm removes the installed module and the RAT self-deletes, preserving the postinstall script that initiated the RAT deployment for malware analysis | Sysmon Event ID 1 (Process Create) and Event ID 3 (Network Connection) records showing node.exe or npm.cmd spawning unexpected child processes (cmd.exe, powershell.exe, sh, curl, wget) or initiating outbound TCP connections to non-npm-registry IPs during the postinstall hook execution window — the primary behavioral indicator of the RAT's T1059 execution | CI/CD pipeline build logs (GitHub Actions workflow run logs, Jenkins $JENKINS_HOME/jobs//builds//log, GitLab CI job trace) capturing the full stdout/stderr of the npm install step that fetched axios@1.14.1 or axios@0.30.4, including any postinstall script output that was not suppressed — timestamps in these logs bound the RAT's execution window precisely | Shell profile files (~/.bashrc, ~/.bash_profile, ~/.zshrc, ~/.profile) and Windows Run registry keys (HKCU\Software\Microsoft\Windows\CurrentVersion\Run, HKLM\Software\Microsoft\Windows\CurrentVersion\Run) on affected developer workstations — a cross-platform RAT establishing persistence would modify these to survive reboots, and their modification timestamps relative to the npm install event are critical forensic indicators | DNS resolver query logs or endpoint DNS cache (ipconfig /displaydns on Windows, sudo dscacheutil -cachedump on macOS, journalctl -u systemd-resolved on Linux) from affected build hosts in the window immediately following axios installation — the RAT's first C2 beacon would appear here as an anomalous domain query from a build system, and this artifact survives the RAT's binary self-deletion per T1070.004

Per-Action IR Details

Step 1: Containment — Identify all systems (developer workstations, CI/CD runners, build containers, staging environments) that executed npm install within the exposure window and may have pulled axios@1.14.1 or axios@0.30.4. Isolate those systems from production networks immediately. Check package-lock.json, yarn.lock, and npm audit output across all active repositories and pipelines for either version. Block further installation of both versions at the npm proxy or artifact repository layer (Artifactory, Nexus, or equivalent) if one is in use.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST CM-3 (Configuration Change Control), NIST SI-3 (Malicious Code Protection), CIS 2.3 (Address Unauthorized Software), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Run the following one-liner across all repos to enumerate affected lockfiles: grep -r '"axios"' --include='package-lock.json' --include='yarn.lock' . | grep -E '1\.14\.1|0\.30\.4'. On systems without EDR, immediately apply host-based firewall rules using iptables (Linux) or Windows Defender Firewall to block all outbound traffic from Node.js processes on affected hosts: iptables -A OUTPUT -p tcp -m owner --uid-owner $(id -u noderunner) -j DROP. If using Artifactory or Nexus, add axios@1.14.1 and axios@0.30.4 to the blocked artifact list immediately via the REST API or admin console. On GitHub Actions or GitLab CI, disable affected runner instances via the web UI or API before pipeline jobs resume.

Evidence: Before isolating, snapshot or export the following from each potentially affected host: (1) The resolved package-lock.json or yarn.lock entry for axios showing the exact fetched version and resolved tarball hash — this is the primary artifact tying a system to the malicious package. (2) npm's per-install log at ~/.npm/_logs/*.log (Linux/macOS) or %APPDATA%\npm-cache_logs*.log (Windows) showing the registry fetch of axios@1.14.1 or axios@0.30.4 with timestamp. (3) The npm cache tarball at ~/.npm/_cacache/ (Linux/macOS) or %AppData%\npm-cache_cacache\ (Windows) — the cached .tgz may contain the malicious postinstall script even after the binary self-deletes. (4) Network connection state at time of isolation using netstat -antp (Linux) or Get-NetTCPConnection (PowerShell) to capture any live C2 sessions before the RAT's self-deletion completes.

Step 2: Detection — Query endpoint detection and response (EDR) telemetry on potentially affected hosts for anomalous outbound connections, process spawns originating from Node.js or npm processes, and any post-install script execution artifacts around the timeframe of axios installation. Check npm install logs and CI/CD pipeline logs for evidence of either compromised version being fetched. Because the RAT self-deletes (T1070.004), absence of a malicious binary does not confirm a clean system — look for network telemetry indicating C2 communication, credential access attempts, or lateral movement from affected hosts. Review DNS query logs and firewall logs for unusual external connections from build systems or developer machines.

NIST Phase: Detection Analysis

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

Controls: NIST IR-5 (Incident Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-12 (Audit Record Generation), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: On systems without EDR, deploy Sysmon immediately using SwiftOnSecurity's config (https://github.com/SwiftOnSecurity/sysmon-config) and review existing Sysmon Event ID 1 (Process Create) for node.exe or npm spawning unexpected child processes (cmd.exe, powershell.exe, sh, bash, curl, wget) within the exposure window. Use this PowerShell command to extract relevant Sysmon events: Get-WinEvent -LogName 'Microsoft-Windows-Sysmon/Operational' | Where-Object { $_.Id -eq 1 -and $_.Message -match 'node|npm' } | Select-Object TimeCreated, Message | Export-Csv sysmon_node_spawns.csv. On Linux/macOS, parse auditd logs or use osquery with query: SELECT p.pid, p.name, p.cmdline, p.parent, pp.name AS parent_name FROM processes p JOIN processes pp ON p.parent = pp.pid WHERE pp.name IN ('node','npm') AND p.name NOT IN ('node','npm'); For network telemetry without a SIEM, capture outbound traffic from build hosts with tcpdump -i any -w axios_c2_capture.pcap and filter for non-RFC1918 destinations on unusual ports. Cross-reference against MITRE ATT&CK T1070.004 (Indicator Removal: File Deletion) by looking for short-lived file creation events in Sysmon Event ID 11 (FileCreate) followed immediately by Event ID 23 (FileDelete) in the npm temp directories.

Evidence: Forensic evidence to collect before beginning analysis sweeps: (1) Sysmon Event ID 3 (Network Connection) records from node.exe processes — the RAT's C2 beacon will appear as an outbound TCP or DNS connection from node.exe or a child process spawned during npm postinstall execution. (2) DNS query logs from the build network's resolver or endpoint DNS cache (ipconfig /displaydns on Windows; sudo dscacheutil -cachedump on macOS) showing domains queried by the affected host in the minutes following axios installation. (3) CI/CD pipeline stdout/stderr logs capturing the full npm install output including postinstall script execution — on GitHub Actions, retrieve via the Actions API; on Jenkins, from the build console log stored at $JENKINS_HOME/jobs//builds//log. (4)

Sysmon Event ID 11 (FileCreate) and Event ID 23 (FileDelete) records in npm temp directories (%TEMP%, /tmp) showing the RAT binary being written and then self-deleted per T1070.004. (5) Windows Security Event Log Event ID 4688 (Process Creation) or Linux auditd EXECVE records showing processes spawned by npm.cmd or node.exe that are not part of normal npm lifecycle scripts.

Step 3: Eradication — Remove axios@1.14.1 and axios@0.30.4 from all environments. Update to the verified safe version per the official Axios GitHub repository (confirm the current safe release at https://github.com/axios/axios/releases before updating). Clear the npm cache on affected systems (npm cache clean --force). Rebuild affected containers and pipeline environments from clean base images rather than patching in place, given the RAT's self-deletion behavior and the possibility of residual persistence mechanisms not yet documented.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST CM-3 (Configuration Change Control), NIST SI-3 (Malicious Code Protection), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Before rebuilding containers, extract and hash the malicious tarballs from the npm cache for forensic preservation: find ~/.npm/_cacache -name '*.tgz' | xargs sha256sum > npm_cache_hashes.txt, then compare against the known-good axios SHA-512 integrity hash recorded in package-lock.json. For container environments, do not attempt in-place remediation — run docker rm -f on affected containers and rebuild from a pinned clean base image with RUN npm install axios@ and no --unsafe-perm flag. Verify the rebuilt image's axios installation by running npm ls axios inside the container and confirming the resolved version and package integrity. For bare-metal developer workstations where a full rebuild is impractical, at minimum: (1) run npm cache clean --force, (2) delete node_modules and reinstall from scratch, (3) verify npm install integrity with npm audit signatures if your npm CLI version supports it (npm >= 9.x).

Evidence: Before executing eradication, preserve: (1) A full copy of the malicious axios package tarball from the npm cache directory (~/.npm/_cacache/ or %AppData%\npm-cache_cacache\) — this is the primary malware sample and must be hashed (SHA-256) and stored in an isolated evidence repository before cache clearing. (2) The postinstall script extracted from the malicious tarball's package.json — this documents the RAT installation mechanism and is needed for persistence artifact hunting in Step 4. (3) Container image layer history (docker history) showing the exact layer in which axios@1.14.1 or axios@0.30.4 was installed, as evidence of supply chain compromise scope in built artifacts. (4) The npm-shrinkwrap.json or package-lock.json from each affected environment as a timestamped record of the compromised dependency state for post-incident reporting.

Step 4: Recovery — After remediation, validate that no persistence mechanisms remain: review scheduled tasks (Windows), cron jobs (Linux/macOS), startup items, and shell profile modifications on affected hosts. Re-scan affected systems with EDR tooling post-remediation. Monitor outbound network traffic from previously affected hosts for 30 days for signs of residual C2 activity. Rotate credentials, tokens, and SSH keys stored on or accessible from affected developer machines, CI/CD runners, and build environments, as the RAT may have exfiltrated them during its execution window.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST AC-2 (Account Management), NIST IA-5 (Authenticator Management), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 5.3 (Disable Dormant Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Use the following platform-specific commands to enumerate persistence artifacts on affected hosts before declaring them clean. Windows: schtasks /query /fo LIST /v > scheduled_tasks.txt and reg query HKCU\Software\Microsoft\Windows\CurrentVersion\Run > startup_registry.txt. Linux/macOS: crontab -l -u > crontabs.txt; cat /etc/cron.* >> crontabs.txt; cat ~/.bashrc ~/.bash_profile ~/.zshrc ~/.profile > shell_profiles.txt; ls -la /etc/init.d/ /etc/systemd/system/ ~/.config/autostart/. For credential rotation prioritization, enumerate secrets accessible from affected CI/CD runners using: grep -r

'AWS_SECRET\|GITHUB_TOKEN\|NPM_TOKEN\|SSH_PRIVATE\|API_KEY' ~/.bashrc ~/.profile ~/.zshrc /etc/environment .env* ci.yml .github/workflows/ — rotate all discovered secrets immediately in their respective platforms (AWS IAM, GitHub, npm registry). For 30-day network monitoring without a SIEM, configure a lightweight Zeek or Suricata instance on a network tap or span port and write a custom rule alerting on outbound connections from previously affected host IPs to non-approved external destinations.

Evidence: Before declaring recovery complete, document: (1) Results of scheduled task and cron job enumeration commands above as a timestamped baseline proving no RAT-installed persistence entries remain. (2) Shell profile file checksums (sha256sum ~/.bashrc ~/.bash_profile ~/.zshrc) compared against a known-good baseline or git-tracked dotfile repo — the RAT may have appended a reverse shell or beacon invocation to these files. (3) SSH authorized_keys file contents (~/.ssh/authorized_keys on all affected hosts) — a cross-platform RAT with SSH key access would add an attacker-controlled public key for persistent access. (4) npm token and registry authentication state from ~/.npmrc and CI/CD environment variable stores — exfiltrated npm publish tokens would allow the attacker to repeat the supply chain compromise against your own published packages.

Step 5: Post-Incident — This attack exploited the absence of npm account security controls (MFA enforcement on maintainer accounts) and the lack of dependency integrity verification in the build pipeline. Immediate control improvements: enforce MFA on all npm accounts with publish rights to packages your organization maintains or depends on; implement package integrity verification (npm audit signatures or equivalent) in CI/CD pipelines; evaluate use of a private npm proxy that pins approved versions and blocks unapproved new versions from reaching build systems automatically; add software composition analysis (SCA) tooling to the pipeline to alert on newly published versions of critical dependencies before they are consumed.

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 SA-12 (Supply Chain Risk Management), NIST SI-7 (Software, Firmware, and Information Integrity), NIST IA-5 (Authenticator Management), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 6.3 (Require MFA for Externally-Exposed Applications)

Compensating: For teams without budget for commercial SCA tooling: (1) Add a free Dependency-Review GitHub Action (github/dependency-review-action) to all repositories — it compares dependency changes in PRs against the GitHub Advisory Database and blocks merges that introduce known-malicious or newly published versions of critical packages. (2) Implement a SBOM generation step in CI using Syft (free, Anchore): syft dir:. -o spdx-json > sbom.json — this creates a software bill of materials for each build that can be diffed to detect unexpected dependency changes. (3) For npm publish account security, enable npm's built-in 2FA enforcement: npm profile set tfa-type auth-and-writes — this prevents publish operations from npm CLI without OTP even if credentials are compromised. (4) Pin axios and all other critical dependencies to exact versions with a lockfile and add a CI step that fails if package-lock.json is modified unexpectedly: git diff --exit-code package-lock.json. (5) Subscribe to the socket.dev npm security feed or OSV.dev for real-time alerts on newly published versions of your dependency tree exhibiting anomalous behavior (new network access, new postinstall scripts, maintainer account changes).

Evidence: Post-incident documentation to produce for lessons learned and regulatory reporting: (1) Timeline of the exposure window — exact timestamps from npm registry publish records for axios@1.14.1 and axios@0.30.4 vs. first detection, used to bound the scope of potential credential exfiltration and calculate breach notification windows. (2) Inventory of all CI/CD pipeline secrets (tokens, keys, credentials) that were present in environment variables or mounted volumes on affected runners during the exposure window — required for determining regulatory notification obligations if PII-adjacent credentials were exposed. (3) MITRE ATT&CK technique coverage gap analysis: document which of T1195.002 (Compromise Software Supply Chain), T1059 (Command and Scripting Interpreter), T1070.004 (Indicator Removal: File Deletion), and T1041 (Exfiltration Over C2 Channel) were not detected by existing controls, to drive detection engineering improvements. (4) Updated npm proxy allowlist or Artifactory remote repository configuration showing axios@1.14.1 and axios@0.30.4 as blocked artifacts, as evidence of control implementation.

Detection Guidance

Primary detection challenge: the RAT deletes its binary after installation, so file-based IOC searches alone are insufficient. Focus detection on behavioral and network indicators. (1) Search CI/CD and npm install logs for any reference to axios@1.14.1 or axios@0.30.4 in the exposure window. (2) In EDR telemetry, look for Node.js or npm spawning unusual child processes during or immediately after package install, particularly shell interpreters (cmd.exe, powershell.exe, bash, sh) as child processes of node.exe or npm scripts. (3) Review network telemetry on build systems and developer machines for outbound connections to unfamiliar IPs or domains made by Node.js processes at install time. (4) Check for post-install script entries in the package.json of the malicious axios versions if you can recover the package from cache or artifact history; a legitimate axios release does not include post-install scripts that spawn system commands. (5) Audit npm cache directories for residual package tarballs from the compromised versions. (6) As of 2026-03-04, no confirmed public IOCs (C2 IPs, C2 domains, malicious hashes) have been published. This limits signature-based detection; rely on behavioral and network indicators instead. Monitor official Axios GitHub issue #10604 and OX Security's disclosure for IOC releases as analysis matures.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://www.npmjs.com/package/axios/v/1.14.1`	Malicious axios package version — npm registry listing. Do not install.	HIGH
URL	`https://www.npmjs.com/package/axios/v/0.30.4`	Malicious axios package version — npm registry listing. Do not install.	HIGH

Framework Mappings

MITRE-ATTACK

- **T1219** — Remote Access Tools
- **T1554** — Compromise Host Software Binary
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1059** — Command and Scripting Interpreter
- **T1070.004** — File Deletion

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-7** — Software, Firmware, and Information Integrity
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-8** — Identification and Authentication (Non-Organizational Users)
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A07:2021** — Identification and Authentication Failures

CIS-V8

- **6.3** — Require MFA for Externally-Exposed Applications
- **6.4** — Require MFA for Remote Network Access
- **6.5** — Require MFA for Administrative Access
- **15.1** — Establish and Maintain an Inventory of Service Providers

SOC2-TSC

- **CC6.1** — The entity implements logical access security software, infrastructure, and architectures over protected information assets
- **CC9.2** — Manages risks associated with vendors and business partners

HIPAA-SECURITY

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

NIST-CSF-2

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

ISO-27001-2022

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1219	Remote Access Tools	Command-And-Control
T1554	Compromise Host Software Binary	Persistence
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1059	Command and Scripting Interpreter	Execution
T1070.004	File Deletion	Defense-Evasion

Sources

Source	URL	Tier
gemini	https://securityboulevard.com/2026/03/poisoned-axios-npm-account-ta...	T3
New axios 1.14.1 and 0.30.4 on npm are likely malicious - Reddit	https://www.reddit.com/r/msp/comments/1s8e1af/new_axios_1141_and_03 ...	T3

Source	URL	Tier
axios@1.14.1 and axios@0.30.4 are compromised #10604 - GitHub	https://github.com/axios/axios/issues/10604	**T3**
Axios Supply Chain Attack Pushes Cross-Platform RAT via ...	https://thehackernews.com/2026/03/axios-supply-chain-attack-pushes-...	**T3**
Axios Compromised With A Malicious Dependency - OX Security	https://www.ox.security/blog/axios-compromised-with-a-malicious-dep...	**T3**

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