INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-03 19:10 UTC

Four coordinated npm supply chain campaigns active in May-June 2026, TTPs, IOCs, and detection notes

THREAT CAMPAIGN | HIGH | CVSS 8.1

SCC Item ID	SCC-CAM-2026-0403
Type	Threat Campaign
CVE ID	Sonatype-2026-003429
Severity	HIGH
CVSS Base Score	8.1
Affected Products	npm ecosystem, 176 packages identified across four coordinated campaigns (May-June 2026)
Published	2026-06-02
Discovery Source	Gemini

Executive Summary

Between May and June 2026, four coordinated supply chain campaigns compromised 176 packages in the npm open-source registry, the primary package manager for JavaScript and Node.js applications. Attackers used dependency confusion techniques to substitute malicious public packages for internal private ones, causing developer build systems to silently pull attacker-controlled code into software products. Any organization building software with npm that pulls private package names from public registries may have ingested malicious code during this window.

Technical Analysis

Four coordinated campaigns targeting the npm ecosystem were active May-June 2026, tracked under Sonatype identifier Sonatype-2026-003429 (no CVE assigned). The primary technique is dependency confusion (CWE-427: Uncontrolled Search Path Element), where attackers register public npm package names that mirror internal private package names. Build systems resolving packages preferentially from the public registry over internal registries pull the attacker-controlled version. A secondary weakness is inclusion of functionality from untrusted sources (CWE-829). MITRE ATT&CK techniques observed: T1195.001 (Compromise Software Dependencies and Development Tools), T1059 (Command and Scripting Interpreter, post-install script execution), T1071.001 (Application Layer Protocol, C2 over HTTP/S). A total of 176 packages were identified across the four campaigns. Sonatype Research assigned an internal severity rating of 8.1 (High) to this

campaign. No CVE or NVD record exists for this identifier. Sonatype Research and socdefenders.com are the attributed discovery sources. Full technical IOC detail (hashes, domains, IP addresses) has not been independently verified against primary authoritative sources; consult Sonatype's full advisory for campaign-specific indicators before deploying signature-based detections.

Action Checklist

1. Step 1: Containment. Audit your npm build pipeline immediately. Identify any package names in your internal registry that match names published publicly on the npm registry during May-June 2026. Temporarily pin all dependencies to verified internal versions and block unauthenticated resolution from public registry for private namespace packages. Reference NIST CM-2 (Baseline Configuration) and CIS 2.3 (Address Unauthorized Software).
2. Step 2: Detection. Review build logs, CI/CD pipeline outputs, and npm install logs from May 1-June 30, 2026 for package resolutions that sourced from registry.npmjs.org instead of your internal registry. Flag any package install that executed a postinstall or preinstall script not present in your prior approved dependency manifest. Check for anomalous outbound connections from build agents consistent with T1071.001 (HTTP/S C2). Apply NIST AU-6 (Audit Record Review and Analysis) and CIS 8.2 (Collect Audit Logs).
3. Step 3: Eradication. Remove any package identified as part of the 176 flagged packages from build caches, artifact repositories, and deployed environments. Configure your npm client to use a scoped registry redirect (via .npmrc) that forces all private package name resolutions to your internal registry and denies fallback to the public registry. Reference CIS 2.1 (Establish and Maintain a Software Inventory) and NIST SI-7 (Software, Firmware, and Information Integrity).
4. Step 4: Recovery. Re-run affected builds from clean environments using pinned, verified dependency manifests. Validate binary outputs against known-good checksums. Monitor deployed applications for anomalous outbound network activity consistent with T1071.001 for a minimum of 30 days post-remediation. Apply NIST SI-4 (System Monitoring). Optionally consult D3FEND D3-PAM (Process Analysis and Monitoring) for advanced behavioral detection techniques.
5. Step 5: Post-Incident. Implement a formal software composition analysis (SCA) tool integrated into your CI/CD pipeline to flag public registry matches against internal package namespaces before build execution. Establish a private registry scoping policy enforced via NIST CM-7 (Least Functionality) and CIS 4.6 (Securely Manage Enterprise Assets and Software). Conduct a lessons-learned review against your software supply chain risk management posture per NIST SR-3 (Supply Chain Controls and Processes).

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO and legal counsel immediately if forensic review confirms any of the 176 flagged packages executed postinstall scripts in your environment, if CI/CD secrets or cloud credentials were present as environment variables during affected builds, or if built artifacts were deployed to production — these conditions collectively constitute a potential data breach triggering regulatory notification obligations under GDPR Article 33, CCPA, or applicable state breach notification statutes.

Recovery Notes	All software artifacts (container images, deployment packages, binaries) built between May 1 and June 30, 2026 using npm must be treated as potentially backdoored and rebuilt from clean, validated dependency manifests in ephemeral environments before being re-promoted to production. Post-rebuild, monitor all production Node.js application servers for 30 days using network-layer inspection for unexpected outbound HTTPS connections from `node` processes, and run weekly file integrity checks on `node_modules/` directories against the post-rebuild SHA-256 baseline. Any CI/CD tokens, NPM_TOKEN values, cloud provider credentials (AWS, GCP, Azure), or SSH keys accessible during builds executed in the May–June 2026 window must be rotated immediately regardless of whether confirmed compromise is established, as attacker postinstall scripts in this campaign class are designed specifically to exfiltrate build-time secrets.
Forensic Artifacts	npm debug logs (~/.npm/_logs/*.log on build agents): record each resolved package URL including originating registry hostname — the presence of 'registry.npmjs.org' for packages that should resolve to your internal registry is direct evidence of dependency confusion exploitation | CI/CD build stdout archives (Jenkins build.log, GitHub Actions workflow run logs, GitLab CI job traces) for May 1–June 30, 2026: search for 'postinstall', 'preinstall', and the names of the 176 flagged packages — script execution lines confirm active malicious code execution during the build | npm cache tarballs (~/.npm/_cacache/ on Linux/macOS, %AppData%\npm-cache on Windows): contain the actual malicious package content including the postinstall script payload — extract with `npm pack` or directly decompress `.tgz` files to recover attacker commands for IOC extraction | Network flow/proxy logs from build agent IP ranges: outbound HTTPS connections to non-internal destinations initiated by node, npm, sh, curl, or wget processes during CI/CD execution windows indicate successful C2 callback from a malicious postinstall script (MITRE T1071.001) | Build agent persistence locations: crontabs (`crontab -l` for all users, /etc/cron.d/), systemd user units (~/.config/systemd/user/), ~/.bashrc, ~/.profile, and ~/.ssh/authorized_keys — dependency confusion postinstall scripts in this campaign class target these locations to establish persistence on build infrastructure for long-term access to the software supply chain

Per-Action IR Details

Step 1: Containment — Audit your npm build pipeline immediately. Identify any package names in your internal registry that match names published publicly on the npm registry during May–June 2026. Temporarily pin all dependencies to verified internal versions and block unauthenticated resolution from public registry for private namespace packages. Reference NIST CM-2 (Baseline Configuration) and CIS 2.3 (Address Unauthorized Software).

NIST Phase: Containment

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

Controls: NIST CM-2 (Baseline Configuration), NIST CM-3 (Configuration Change Control), NIST SA-12 (Supply Chain Protection), CIS 2.3 (Address Unauthorized Software), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Run `npm ls --json > dep-tree-$(date +%F).json` across all active build workspaces to snapshot current resolved dependency trees. Cross-reference package names against the 176 flagged packages using a bash script: `comm -12 <(sort internal-pkg-names.txt) <(sort flagged-176.txt)`. Immediately add `registry=https://your-internal-registry` and `@yourscope:registry=https://your-internal-registry` to all `.npmrc` files in CI/CD agent home directories to block public fallback without requiring commercial tooling. Lock Node.js dependency resolution using `npm ci` (which enforces `package-lock.json`) rather than `npm install` in all pipeline scripts.

Evidence: Before making any pipeline changes, snapshot and preserve: (1) current `.npmrc` files from all CI/CD build agents (`~/.npmrc` and project-level `.npmrc`); (2) full `package-lock.json` and `yarn.lock` files from affected repositories as of the May–June 2026 window — these record the resolved registry source URL per package and will

confirm whether resolution hit `registry.npmjs.org` vs. your internal registry; (3) npm cache directories (`~/.npm/_cacache/` on Linux/macOS, `%AppData%\npm-cache` on Windows) which retain tarballs of downloaded packages including attacker-controlled ones; (4) CI/CD pipeline execution logs from your build system (Jenkins `build.log`, GitHub Actions workflow run logs, GitLab CI job traces) covering May 1–June 30, 2026.

Step 2: Detection — Review build logs, CI/CD pipeline outputs, and npm install logs from May 1–June 30, 2026 for package resolutions that sourced from registry.npmjs.org instead of your internal registry. Flag any package install that executed a postinstall or preinstall script not present in your prior approved dependency manifest. Check for anomalous outbound connections from build agents consistent with T1071.001 (HTTP/S C2). Apply NIST AU-6 (Audit Record Review and Analysis) and CIS 8.2 (Collect Audit Logs).

NIST Phase: Detection Analysis

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

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

Compensating: On build agents running Linux: run `grep -E 'resolved.*registry\.npmjs\.org' ~/.npm/_logs/*.log` to identify packages resolved from public npm instead of your internal registry during the target window. To detect postinstall script execution, run `npm ls --json | python3 -c "import sys,json; [print(k) for k,v in json.load(sys.stdin).get('dependencies',{}).items() if 'scripts' in v]"` then diff output against your approved manifest baseline. For network telemetry without EDR, deploy Sysmon (config with NetworkConnect events enabled) on build agents and query: `Get-WinEvent -LogName 'Microsoft-Windows-Sysmon/Operational' | Where-Object {$_.Id -eq 3 -and $_.Message -notmatch 'your-internal-registry-fqdn'}` filtering on Event ID 3 (Network Connection) to surface unexpected outbound HTTPS from `node`, `npm`, or `sh` processes. On Linux build agents, use `ss -tunp` snapshots or `auditd` rules targeting `execve` syscalls from npm lifecycle script processes.

Evidence: Capture before analysis: (1) npm debug and timing logs from `~/.npm/_logs/` (filenames are timestamped ISO-8601) — these record each resolved package URL including the originating registry; (2) CI/CD build stdout/stderr archives for May 1–June 30, 2026 — search for the string `postinstall` or `preinstall` combined with package names from the 176 flagged list; (3) DNS query logs from build agent network segments for queries to `registry.npmjs.org` originating from build servers — dependency confusion attacks require the build agent to resolve the attacker's public package name via public DNS; (4) netflow or proxy logs showing outbound HTTPS (port 443) from build agent IP ranges to non-internal destinations initiated by `node` or `npm` processes — consistent with MITRE T1071.001 C2 callback from malicious postinstall scripts; (5) process execution logs (Sysmon Event ID 1 or Linux auditd `execve`) showing child processes spawned by npm lifecycle hooks — attacker postinstall scripts in this campaign class commonly exec `curl`, `wget`, `python3`, or `sh -c` to exfiltrate environment variables or download second-stage payloads.

Step 3: Eradication — Remove any package identified as part of the 176 flagged packages from build caches, artifact repositories, and deployed environments. Configure your npm client to use a scoped registry redirect (via .npmrc) that forces all private package name resolutions to your internal registry and denies fallback to the public registry. Reference CIS 2.1 (Establish and Maintain a Software Inventory) and NIST SI-7 (Software, Firmware, and Information Integrity).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST SI-2 (Flaw Remediation), NIST CM-7 (Least Functionality), NIST SA-15 (Development Process, Standards, and Tools), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.3 (Address Unauthorized Software)

Compensating: Purge npm caches on all build agents with `npm cache clean --force` then verify with `npm cache verify`. For Artifactory, Nexus, or Verdaccio internal registries, identify and delete flagged package tarballs by querying artifact metadata: in Nexus, use `curl -u admin:pass 'http://nexus/service/rest/v1/search?name=FLAGGED_PKG_NAME&repository=npm-proxy'` and DELETE via the REST API. Enforce registry isolation in `.npmrc` by setting `registry=https://your-internal-registry` at the system level

(`/etc/npmrc` on Linux) and adding per-scope overrides for all private namespaces. Use YARA rules against cached tarballs to identify malicious postinstall script patterns (e.g., base64-encoded payloads, curl/wget invocations) before deletion to preserve forensic evidence: `yara -r postinstall_malware.yar ~/.npm/_cacache/`.

Evidence: Before eradication, preserve forensic copies of: (1) the full npm cache directory (`~/.npm/_cacache/`) from each build agent — contains the actual malicious tarball content, which can be extracted and analyzed for postinstall script payload; (2) extracted `package.json` files from within the flagged cached tarballs — the `scripts.postinstall` field will contain the malicious command string used in this dependency confusion campaign; (3) any artifacts published to internal registries or artifact stores that incorporated the flagged packages as transitive dependencies — these are potentially compromised build outputs; (4) environment variable snapshots from build agents if postinstall scripts executed, as dependency confusion attacks in this class routinely exfiltrate `NODE_ENV`, `NPM_TOKEN`, `AWS_ACCESS_KEY_ID`, `CI_*` tokens, and SSH keys present in build environment.

Step 4: Recovery — Re-run affected builds from clean environments using pinned, verified dependency manifests. Validate binary outputs against known-good checksums. Monitor deployed applications for anomalous outbound network activity consistent with T1071.001 for a minimum of 30 days post-remediation. Apply NIST SI-4 (System Monitoring) and D3-SFA (System File Analysis) to verify no persistence mechanisms were dropped.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-4 (System Monitoring), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CP-10 (System Recovery and Reconstitution), NIST CA-7 (Continuous Monitoring), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Provision ephemeral build environments (fresh Docker containers or VMs with no cached state) for rebuild — do not reuse build agents that executed compromised installs without full reimaging. Generate SHA-256 checksums of all rebuilt artifacts with `sha256sum dist/* > build-manifest-$(date +%F).sha256` and compare against pre-incident known-good checksums stored in your version control system. For 30-day post-recovery monitoring without EDR, deploy osquery on production hosts with a query scheduled every 5 minutes: `SELECT pid, name, remote_address, remote_port FROM process_open_sockets WHERE remote_port IN (80, 443, 8080, 8443) AND name IN ('node', 'npm', 'sh', 'bash', 'python3', 'curl', 'wget')` — flag any production Node.js process initiating unexpected outbound connections. Use `find / -name '*.js' -newer /date-of-incident -path '*/node_modules/*' 2>/dev/null` to identify any JavaScript files written to node_modules after the incident window, which may indicate dropped persistence.

Evidence: Before restoring production traffic to rebuilt deployments, capture and validate: (1) file integrity baseline of `node_modules/` directories using `find node_modules -name 'package.json' -exec sha256sum {} \;` — compare against the pre-incident baseline to detect any files that differ from expected; (2) systemd unit files, cron jobs (`/etc/cron*`, `crontab -l` for all users), and `~/.bashrc`/`~/.profile` modifications on build agents — malicious postinstall scripts in supply chain campaigns frequently install cron-based or profile-based persistence; (3) SSH `authorized_keys` files for all user accounts on build agents and any systems the build agents have deploy access to — attacker postinstall scripts in this campaign class may have injected attacker SSH public keys; (4) outbound network connection logs from production application servers for the 30-day monitoring window, specifically filtering on `node` process connections to non-approved external IP ranges.

Step 5: Post-Incident — Implement a formal software composition analysis (SCA) tool integrated into your CI/CD pipeline to flag public registry matches against internal package namespaces before build execution. Establish a private registry scoping policy enforced via NIST CM-7 (Least Functionality) and CIS 4.6 (Securely Manage Enterprise Assets and Software). Conduct a lessons-learned review against your software supply chain risk management posture per NIST SR-3 (Supply Chain Controls and Processes).

NIST Phase: Post Incident

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

Controls: NIST CM-7 (Least Functionality), NIST SR-3 (Supply Chain Controls and Processes), NIST SR-4 (Provenance), NIST SI-2 (Flaw Remediation), NIST SA-15 (Development Process, Standards, and Tools), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 7.1 (Establish and Maintain a Vulnerability Management

Process), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Integrate OWASP Dependency-Check (free, open source) into your CI/CD pipeline as a pre-build gate: `dependency-check.sh --project MyApp --scan ./package.json --format JSON --out ./reports/` — configure it to fail the build on any package matching the Sonatype-2026-003429 advisory identifiers once published to NVD. For dependency confusion detection specifically, use `confused` (free CLI tool by visma-prodsec) to scan your `package.json` and `package-lock.json`: `confused -l npm package-lock.json` — it identifies private package names that also exist on the public registry. Enforce `.npmrc` registry scoping policy via a pre-commit hook (`husky` + a shell script) that validates all `@internal-scope` packages are pinned to your internal registry URL before any `npm install` is permitted in CI. Document and version-control the approved package manifest as a signed artifact in your repository.

Evidence: For the lessons-learned review, compile: (1) timeline reconstruction from npm install logs, CI/CD build history, and DNS logs establishing when each of the 176 flagged packages was first resolved from the public registry in your environment — this determines your actual exposure window; (2) inventory of all software artifacts (container images, deployment packages, Lambda ZIPs, etc.) built during May 1–June 30, 2026 that incorporated any of the 176 packages as direct or transitive dependencies — these constitute your blast radius; (3) list of CI/CD secrets, tokens, and credentials accessible as environment variables during the affected build window — these must be treated as fully compromised and rotated; (4) documentation of any internal package names that were squattable on the public registry (i.e., unregistered on npmjs.org) — this gap list drives your namespace reservation remediation action items.

Detection Guidance

Focus detection on three surfaces: (1) Build pipeline registry resolution logs, query for npm install events where the resolved registry URL is registry.npmjs.org and the package name matches any name also defined in your internal registry. (2) CI/CD and build agent process execution, look for postinstall or preinstall script spawns (node, sh, bash, python) that were not present in the prior approved package-lock.json or yarn.lock. This maps to T1059 and NIST AU-2 (Event Logging). (3) Network egress from build agents, alert on outbound HTTP/S connections to non-approved destinations originating from npm install processes, consistent with T1071.001. Recommended countermeasures include system file analysis of build artifacts and installed package contents, combined with access controls restricting build agent network access to approved registries only (D3FEND D3-PAM). Note: Specific IOC hashes, domains, and IP addresses have not been independently verified against primary authoritative sources. The 176 package names should be obtained directly from Sonatype Research Sonatype-2026-003429 or the socdefenders.com blog before deploying detection rules.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`registry.npmjs.org — resolution of private-namespace package names from public registry`	Dependency confusion attack vector — attacker-controlled packages published to public npm registry using internal package name collision	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1059** — Command and Scripting Interpreter

- **T1071.001** — Web Protocols

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-7** — Software, Firmware, and Information Integrity
- **SR-2** — Supply Chain Risk Management Plan

NIST-CSF-2

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

CIS-V8

- **15.1** — Establish and Maintain an Inventory of Service Providers
- **8.2** — Collect Audit Logs

ISO-27001-2022

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

SOC2-TSC

- **CC9.2** — Manages risks associated with vendors and business partners

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1059	Command and Scripting Interpreter	Execution
T1071.001	Web Protocols	Command-And-Control

Sources

Source	URL	Tier
gemini	https://socdefenders.com/blog/four-coordinated-npm-supply-chain-cam...	T3
sonatype-2026-003429 | Sonatype Research | Sonatype Guide	http://guide.sonatype.com/vulnerability/sonatype-2026-003429/sonaty...	T3
2026 Release Notes - Sonatype Nexus Repository	https://help.sonatype.com/en/nexus-repository-2026-release-notes.html	T3

Source	URL	Tier
CVE-2026-0600: Sonatype Nexus Repository 3 SSRF Vulnerability	https://www.sentinelone.com/vulnerability-database/cve-2026-0600/	T3
Sonatype 2026 Report: The Challenge of Trust at Scale	https://homeostase.pt/en/2026/02/26/sonatype-2026-report-the-challe...	T3
NVD	https://nvd.nist.gov/vuln/detail/Sonatype-2026-003429	T1

Generated 2026-06-03 19:10 UTC by TJS Security Command Center