INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-03-29 18:39 UTC

PhantomRaven npm Supply Chain Campaign: 88 Malicious Packages Targeting CI/CD Tokens and Developer Credentials

THREAT CAMPAIGN | **HIGH** | CVSS 7.5

SCC Item ID	SCC-CAM-2026-0080
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	7.5
Affected Products	npm registry, Node.js environments, Babel ecosystem, GraphQL Codegen, GitHub Actions, GitLab CI, Jenkins, CircleCI
Published	2026-03-22

Executive Summary

A threat actor tracked as PhantomRaven has published 88 malicious packages to the npm registry across four campaign waves between August 2025 and February 2026, with 81 packages confirmed still available for installation as of the most recent source report (February 2026). The campaign targets CI/CD pipeline credentials, developer tokens, and environment variables for GitHub Actions, GitLab CI, Jenkins, and CircleCI, systems central to software build and delivery pipelines. Organizations with Node.js development teams or automated build infrastructure face direct risk of credential theft, pipeline compromise, and potential downstream supply chain exposure.

Technical Analysis

PhantomRaven has conducted a multi-wave npm supply chain campaign involving 88 packages that typosquat or impersonate legitimate developer tooling, including packages mimicking the Babel ecosystem and GraphQL Codegen. The campaign's primary evasion technique decouples malicious logic from the static package content via post-install lifecycle scripts (package.json 'postinstall' hooks, per GitLab threat reporting) that retrieve remote payloads or execute conditionally, bypassing registry-level static analysis. Once installed in a CI/CD build environment, the packages exfiltrate credentials and tokens stored in environment variables. Mapped CWEs include CWE-506 (Embedded Malicious Code), CWE-312 (Cleartext Storage of Sensitive Information), and CWE-494 (Download of Code Without Integrity Check). MITRE ATT&CK techniques cover the full collection-to-exfiltration chain: T1195.001 (Compromise Software Supply Chain, Development Tools), T1036.005 (Masquerading, Match Legitimate Name or Location), T1059.006/T1059.007 (Command and

Scripting Interpreter, Python/JavaScript), T1552.001 (Unsecured Credentials, Credentials in Files), T1552.004 (Unsecured Credentials, Private Keys), T1071.001 (Application Layer Protocol, Web Protocols), and T1041 (Exfiltration Over C2 Channel). No CVE has been assigned; this is a malicious package campaign, not a vulnerability in a named product. Infrastructure reuse across waves and an accelerating publication cadence indicate an organized, ongoing operation. No vendor-issued patches apply; remediation depends on package removal and credential rotation.

Action Checklist

1. Step 1, Immediate: Audit npm dependencies across all repositories and CI/CD pipeline configurations for packages typosquatting Babel, GraphQL Codegen, or other widely used developer tooling; cross-reference installed package names against the 88 packages identified in source reporting. Remove any confirmed malicious packages immediately and lock dependency versions.
2. Step 2, Detection: Search CI/CD build logs for unexpected outbound network connections initiated during npm install steps, particularly to domains or IPs not associated with known package registries or internal infrastructure. Look for postinstall script execution entries in npm debug logs and any anomalous environment variable access patterns during build runs.
3. Step 3, Assessment: Inventory all GitHub Actions, GitLab CI, Jenkins, and CircleCI secrets and tokens that were accessible in build environments where potentially affected packages were installed. Treat any token or credential present in those environments as potentially compromised pending review.
4. Step 4, Credential Rotation: Rotate all CI/CD tokens, API keys, and developer credentials that may have been exposed in affected build environments. Revoke and reissue GitHub Actions secrets, GitLab CI/CD variables, Jenkins credentials, and CircleCI environment variables where exposure cannot be ruled out. Notify affected development teams and platform owners.
5. Step 5, Long-term Controls: Enforce a private npm registry or allowlist-based dependency policy (e.g., Artifactory, Verdaccio, npm audit gates in CI pipelines) to restrict package installation to vetted sources. Implement automated software composition analysis (SCA) tooling with behavioral analysis capabilities, not solely static name-matching, as part of the CI/CD pipeline. Review and restrict postinstall script execution permissions in build environments. Establish a repeating dependency audit cadence aligned to CIS Benchmark guidance for supply chain controls.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO/security leadership immediately if any potentially malicious packages were confirmed installed in production build environments or if credential compromise cannot be ruled out; escalate to external IR firm if evidence of data exfiltration, lateral movement to source repositories, or unauthorized code commits is discovered.
Recovery Notes	After eradication, conduct a full source code audit to verify no malicious commits were injected into repositories by compromised CI/CD credentials. Perform a 90-day artifact review of all built artifacts, container images, and deployments created during the exposure window to identify potentially poisoned releases. Establish a 6-month post-incident monitoring plan with enhanced logging and SCA scanning to detect re-infection or supply chain attacks against replacement dependencies.

Forensic Artifacts	npm debug logs (~/.npm/_logs/*.log) and CI/CD pipeline build logs (GitHub Actions runner logs, GitLab job artifacts, Jenkins $JENKINS_HOME/logs/, CircleCI job logs) | package-lock.json, yarn.lock, and pnpm-lock.yaml files showing dependency resolution history | CI/CD environment variable snapshots and secrets inventory exports from GitHub, GitLab, Jenkins, CircleCI platforms | Network packet captures (PCAP files) from CI/CD runner environments during npm install execution, DNS query logs, and firewall/egress logs for outbound connections | Git commit history and `.gitlog` files showing credential-related changes, source code modifications, and pipeline configuration updates in the past 120 days

Per-Action IR Details

Step 1 — Immediate: Audit npm dependencies across all repositories and CI/CD pipeline configurations for packages typosquatting Babel, GraphQL Codegen, or other widely used developer tooling; cross-reference installed package names against the 88 packages identified in source reporting. Remove any confirmed malicious packages immediately and lock dependency versions.

NIST Phase: Preparation

Reference: NIST 800-61r3 §2.1 (Preparation — tools and resources) and §3.1 (Detection and Analysis — scope identification)

Controls: NIST 800-53 SA-3 (System Development Life Cycle), NIST 800-53 SA-10 (Developer Configuration Management), CIS Controls 6.2 (Employ Software Composition Analysis), CIS Controls 7.2 (Maintain Software Bill of Materials)

Compensating: Use `npm ls` to dump all installed packages into a CSV (name, version, resolved URL). Cross-reference manually against the PhantomRaven IOC list (available from CISA or security research reports). Automate with a bash loop: `npm ls --json | jq '.dependencies | keys[]' | while read pkg; do grep -F "$pkg" malicious_packages.txt && echo "MATCH: $pkg"; done`. Lock versions with `npm ci --prefer-offline --no-audit` after removal.

Evidence: Capture `package-lock.json` or `yarn.lock` from all repos BEFORE running npm install/update. Extract `npm ls --json` output showing full dependency tree with resolved URLs. Archive all `.npmrc` files and CI/CD pipeline YAML configs showing dependency installation steps. Preserve npm debug logs from recent CI runs: `~/.npm/_logs/*.log`.

Step 2 — Detection: Search CI/CD build logs for unexpected outbound network connections initiated during or immediately after npm install steps, particularly to unfamiliar domains or IPs. Look for postinstall script execution entries in npm debug logs and any anomalous environment variable access patterns during build runs.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2.2 (Detection and Analysis — analysis techniques) and §3.2.4 (Collect Evidence)

Controls: NIST 800-53 SI-4 (Information System Monitoring), NIST 800-53 CA-7 (Continuous Monitoring), CIS Controls 8.1 (Implement Network Segmentation), CIS Controls 8.6 (Deny Traffic by Default)

Compensating: Parse CI/CD pipeline logs (GitHub Actions: GitHub API `/repos/{owner}/{repo}/actions/runs` or runner logs; GitLab: CI job artifacts; Jenkins: `$JENKINS_HOME/logs/*.log`) for DNS queries and outbound connections. Use `grep` to search for 'postinstall', 'preinstall', 'prepare' scripts: `grep -r "postinstall\|preinstall\|prepare" package.json`. For network monitoring without SIEM, enable network packet capture during a test npm install in isolated environment and analyze with `tcpdump -i any -w npm_install.pcap 'tcp port 443 or udp port 53'` then inspect with Wireshark or tshark.

Evidence: Capture complete CI/CD build logs for all pipeline runs in the past 90 days (or since last known safe state). Extract npm debug logs: `npm install --verbose 2>&1 | tee npm_verbose.log`. Record DNS query logs from CI runner environment if available. Preserve environment variable listings from build context (GitHub: `env` output in Actions logs; Jenkins: build console output). Take network packet captures (.pcap) from isolated test runs of npm install with suspected packages.

Step 3 — Assessment: Inventory all GitHub Actions, GitLab CI, Jenkins, and CircleCI secrets and tokens that were accessible in build environments where potentially affected packages were installed. Treat any token or credential present in those environments as potentially compromised pending review.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2.3 (Determine Scope of Compromise) and §3.2.4 (Collect Evidence)

Controls: NIST 800-53 IA-4 (Identifier Management), NIST 800-53 SC-12 (Cryptographic Key Establishment and Management), CIS Controls 3.3 (Address Unauthorized Software), CIS Controls 5.1 (Establish Secure Configuration Standards)

Compensating: Manually audit secrets in each platform: GitHub (Settings > Secrets and variables > Actions); GitLab (CI/CD > Variables or Settings > CI/CD); Jenkins (Manage Jenkins > Manage Credentials); CircleCI (Project settings > Environment Variables). For each environment where a malicious package was installed, document: secret name, creation date, last rotation date, scope (org/repo/project). Use `git log --all --grep="secrets\|token\|credential" --oneline` to identify any credentials accidentally committed. Cross-reference against CI logs to determine which secrets were loaded into build environment memory.

Evidence: Export full secret/token inventory from each platform with metadata (creation date, scope, last used timestamp if available). Collect environment variable logs from build runners showing which secrets were injected into build context. Extract `$HISTORY` from CI runner shells (if preserved). Preserve GitHub Actions workflow files and GitLab CI YAML showing secret references. Document git commit history for any credential-related changes in the past 120 days.

Step 4 — Credential Rotation: Rotate all CI/CD tokens, API keys, and developer credentials that may have been exposed in affected build environments. Revoke and reissue GitHub Actions secrets, GitLab CI/CD variables, Jenkins credentials, and CircleCI environment variables where exposure cannot be ruled out. Notify affected development teams and platform owners.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.3.2 (Eradication) and §3.4.1 (Recovery — reconstitution steps)

Controls: NIST 800-53 IA-4 (Identifier Management), NIST 800-53 IA-5 (Authentication), CIS Controls 5.2 (Use Multifactor Authentication for All Administrative Access), CIS Controls 1.3 (Inventory and Control of Administrative Accounts)

Compensating: Create a credential rotation runbook: For each platform, revoke the old token (GitHub: Settings > Developer settings > Personal access tokens > Delete; GitLab: User settings > Access Tokens > Revoke; Jenkins: Manage Jenkins > Manage Credentials > Delete; CircleCI: Project settings > Environment Variables > Remove), generate a new token with the same minimal-privilege scope, update references in all CI/CD pipeline configs and local `.env` files. Use `git-secrets` or `truffleHog` to ensure old credentials are not present in git history: `truffleHog filesystem /path/to/repo --json`. Document rotation timestamp and who approved each rotation in an audit log (spreadsheet acceptable).

Evidence: Capture snapshots of old token/secret values BEFORE revocation (if policy permits; otherwise document metadata only — creation date, scope, last rotation). Record new token issuance timestamps and identifiers. Preserve notification logs showing which teams were informed and when. Document approval chain for each rotation decision. Archive old credentials in encrypted, access-controlled storage for forensic review if needed.

Step 5 — Long-term Controls: Enforce a private npm registry or allowlist-based dependency policy (e.g., Artifactory, Verdaccio, npm audit gates in CI pipelines) to restrict package installation to vetted sources. Implement automated software composition analysis (SCA) tooling with behavioral analysis capabilities — not solely static name-matching — as part of the CI/CD pipeline. Review and restrict postinstall script execution permissions in build environments. Establish a repeating dependency audit cadence aligned to CIS Benchmark guidance for supply chain controls.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.4.2 (Recovery — restore to secure baseline) and §4 (Post-Incident Activities)

Controls: NIST 800-53 SA-3 (System Development Life Cycle), NIST 800-53 SA-10 (Developer Configuration Management), NIST 800-53 SI-7 (Software, Firmware, and Information Integrity), CIS Controls 6.2 (Employ Software Composition Analysis), CIS Controls 6.3 (Maintain Software Bill of Materials), CIS Controls 13.1 (Inventory and Control of the Supply Chain)

Compensating: Deploy Verdaccio (open-source private npm registry) in Docker: `docker run -d -p 4873:4873 -v verdaccio:/verdaccio/storage verdaccio/verdaccio`. Configure `.npmrc` to use private registry: `registry=http://localhost:4873`. Implement free SCA with OWASP Dependency-Check: integrate into CI with `dependency-check --scan /path/to/project --format JSON --out reports`. Restrict postinstall execution by creating `npm-postinstall-blocker.js` that intercepts and logs postinstall attempts; add to `.npmrc`: `ignore-scripts=true` for all non-internal packages. Establish monthly dependency audit: `npm audit --json > audit_$(date +%Y%m%d).json` and compare against CIS v8 6.2 baseline. Document findings in a shared spreadsheet and track remediation.

Evidence: Preserve baseline `.npmrc`, CI/CD pipeline configs, and Verdaccio allowlist after implementation. Capture SCA scan results and dependency audit reports for each audit cycle. Document postinstall script blocking rules and exceptions (with approval trail). Archive access logs from private registry showing which packages were installed by which teams/projects. Maintain audit trail of policy changes and team acknowledgment of new supply chain controls.

Detection Guidance

Focus detection on three surfaces: package installation behavior, network activity during builds, and credential access patterns. In npm debug logs (typically ~/.npm/_logs/ or CI artifact logs), look for postinstall script execution on recently added or unfamiliar packages. In network flow or proxy logs, identify outbound HTTP/HTTPS connections originating from CI/CD worker nodes or containers during build phases, particularly to domains or IPs not associated with known package registries or internal infrastructure. In CI/CD platform audit logs (GitHub Actions workflow logs, GitLab CI job traces, Jenkins build console output), look for unexpected environment variable reads or exports outside of explicitly defined pipeline steps. Behavioral indicator: packages that execute network calls during installation rather than at runtime are a strong signal of malicious postinstall activity. For SIEM detections, correlate npm install events with outbound connections on ports 80/443 within a short time window from build agents. Source reporting (BleepingComputer, GitLab blog) may contain specific package names for direct IOC matching against installed dependency manifests, verify against your own package-lock.json and yarn.lock files.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://www.bleepingcomputer.com/news/security/new-phantomraven-npm-attack-wave-steals-dev-data-via-88-packages/`	Primary news source reporting on PhantomRaven campaign — consult for specific package names and IOC list. URL is source-reported; validate before use.	LOW
URL	`https://about.gitlab.com/blog/gitlab-discovers-widespread-npm-supply-chain-attack/`	GitLab-authored technical analysis of the campaign — likely contains package names and infrastructure indicators. URL is source-reported; validate before use.	LOW

Framework Mappings

MITRE-ATTACK

- **T1552.004** — Private Keys
- **T1071.001** — Web Protocols
- **T1036.005** — Match Legitimate Resource Name or Location
- **T1059.006** — Python
- **T1059.007** — JavaScript
- **T1552.001** — Credentials In Files
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1176** — Software Extensions
- **T1041** — Exfiltration Over C2 Channel

NIST-800-53R5

- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-4** — System Monitoring
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **2.5**
- **2.6**
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers
- **8.2** — Collect Audit Logs

HIPAA-SECURITY

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

SOC2-TSC

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

NIST-CSF-2

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

ISO-27001-2022

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1552.004	Private Keys	Credential-Access
T1071.001	Web Protocols	Command-And-Control
T1036.005	Match Legitimate Resource Name or Location	Defense-Evasion
T1059.006	Python	Execution
T1059.007	JavaScript	Execution
T1552.001	Credentials In Files	Credential-Access
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1176	Software Extensions	Persistence
T1041	Exfiltration Over C2 Channel	Exfiltration

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/new-phantomraven-npm...	T3
GitLab discovers widespread npm supply chain attack	https://about.gitlab.com/blog/gitlab-discovers-widespread-npm-suppl...	T3
Self-Replicating Worm Hits 180+ npm Packages to Steal Credentials ...	https://thehackernews.com/2025/09/40-npm-packages-compromised-in-su...	T3
Malicious NPM Package Found Targeting GitHub Actions - Veracode	https://www.veracode.com/blog/malicious-npm-package-targeting-githu...	T3
npm Under Siege: Evolving Supply Chain Threats - Cymulate	https://cymulate.com/blog/npm-under-siege-supply-chain-attacks/	T3

Generated 2026-03-29 18:39 UTC by TJS Security Command Center