INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-23 19:02 UTC

npm Staged Publishing and Install Source Controls Close Two Persistent Supply Chain Attack Vectors

SECURITY ANALYSIS | MEDIUM | CVSS 5.0

SCC Item ID	SCC-STY-2026-0155
Type	Security Analysis
Severity	MEDIUM
CVSS Base Score	5.0
Affected Products	npm registry, GitHub, npm CLI 11.15.0+
Published	2026-05-23T12:35:10
Discovery Source	Rss

Executive Summary

GitHub has made staged publishing generally available for npm, introducing a mandatory 2FA-authenticated human approval gate before any package version becomes installable, including releases from automated CI/CD pipelines. This closes two persistent supply chain attack vectors: unauthorized automated publishing via compromised CI credentials and non-registry source substitution attacks. The controls arrive as threat group TeamPCP actively poisons open-source packages at scale, signaling that the npm ecosystem has crossed from theoretical risk to actively exploited territory.

Technical Analysis

GitHub's staged publishing feature addresses a structural weakness in the npm publishing pipeline that has existed since CI/CD automation became standard practice: once a pipeline held a valid publish token, it could push any package version to the registry without human review. This gap maps directly to MITRE T1195.001 (Supply Chain Compromise: Compromise Software Dependencies and Development Tools) and T1554 (Compromise Host Software Binary). Attackers targeting this path, whether through compromised CI credentials (T1078, Valid Accounts) or a compromised third-party integration (T1199, Trusted Relationship), could publish malicious versions that downstream consumers would install automatically.

The staged publishing gate requires a 2FA-authenticated human to approve any release before it becomes installable. This breaks the fully automated attack chain. Even if an attacker obtains CI/CD credentials, they cannot complete a malicious publish without triggering the approval workflow and requiring human action, which either exposes the attempt or blocks it entirely. This maps to CWE-287 (Improper Authentication in publishing

workflows) and addresses T1553.002 (Subvert Trust Controls: Code Signing) by ensuring that publication itself becomes a trust checkpoint, not just the artifact.

The three new install source flags in npm CLI 11.15.0+ address a separate but related vector: dependency confusion and source substitution attacks that exploit non-registry install paths. When developers specify dependencies via git URLs, file paths, or tarball URLs, there has been no native enforcement mechanism to restrict or audit those sources. The new flags give teams explicit, policy-enforceable control over which non-registry sources are permitted, directly reducing the CWE-829 (Inclusion of Functionality from Untrusted Control Sphere) and CWE-494 (Download of Code Without Integrity Check) attack surface.

The timing matters. TeamPCP has been observed conducting active, large-scale open-source package poisoning campaigns, demonstrating that supply chain compromise via the npm ecosystem is no longer a proof-of-concept scenario. Development teams that rely on npm packages, directly or transitively, face a concrete, operationally active threat. These new controls are meaningful mitigations, but they require adoption: staged publishing must be enabled by package maintainers, and the install source flags must be configured in project tooling. Neither is automatic.

Action Checklist

1. Step 1: Assess exposure, inventory all npm packages your organization publishes and all npm dependencies consumed directly or transitively in production builds; determine whether any published packages have staged publishing available and not yet enabled
2. Step 2: Enable staged publishing, for any npm packages your organization maintains, enable the staged publishing feature in npm/GitHub settings to enforce the 2FA-authenticated human approval gate before any version becomes installable (addresses NIST CM-3: Configuration Change Control and NIST SI-7: Software, Firmware, and Information Integrity)
3. Step 3: Upgrade npm CLI and configure install source flags, update to npm CLI 11.15.0+ and configure the new install source restriction flags to disallow or explicitly permit non-registry sources (git URLs, file paths, tarball URLs) in project and CI/CD configurations (addresses CIS 7.3: Perform Automated Operating System Patch Management and CIS 2.2: Ensure Authorized Software is Currently Supported)
4. Step 4: Audit CI/CD pipeline credentials, rotate npm publish tokens and audit which pipelines, service accounts, and third-party integrations hold publish access; apply least-privilege scoping so credentials cannot publish without the staged approval gate (addresses NIST AC-6: Least Privilege, D3-CRO: Credential Rotation, and CIS 5.4: Restrict Administrator Privileges to Dedicated Administrator Accounts)
5. Step 5: Update threat model for TeamPCP activity, incorporate TeamPCP's active package poisoning campaigns into your threat register against T1195.001, T1554, and T1199; establish detection baselines for unexpected package versions appearing in dependency lock files or build outputs
6. Step 6: Communicate findings, brief engineering leadership and AppSec teams on the specific exposure: if your organization publishes npm packages or consumes packages from non-registry sources, the risk is concrete and the mitigations require active opt-in, not passive updates
7. Step 7: Monitor for follow-up disclosures, track npm security advisories, GitHub changelog updates for staged publishing scope, and threat intelligence on TeamPCP campaign activity for new indicators or expansion of targeted packages

IR / Forensic Enrichment

Triage Priority	URGENT
Escalation Criteria	Escalate to immediate priority and engage AppSec leadership if `npm audit`, lock file diff analysis, or registry metadata review reveals any dependency in a production build resolved to an unexpected version or integrity hash mismatch consistent with TeamPCP substitution activity, or if any npm publish token with scope over a publicly-consumed package is found to have been used outside a known, approved pipeline run.
Recovery Notes	After enabling staged publishing and rotating publish tokens, verify recovery by confirming that a test publish attempt from the CI/CD pipeline is correctly intercepted and held in staged state pending 2FA-authenticated human approval — do not consider the control active until this end-to-end test passes. Monitor `package-lock.json` integrity hashes across all production build environments for a minimum of 30 days post-remediation, comparing each build's resolved hashes against the known-good baseline captured in Step 5, since TeamPCP campaigns may have pre-staged malicious versions that surface on the next `npm install` cycle. Retain the pre-rotation token audit output and lock file snapshots as forensic baseline records in case a downstream incident requires establishing when a potential compromise window opened or closed.
Forensic Artifacts	package-lock.json and npm-shrinkwrap.json from all production build environments: the `resolved` URL and `integrity` (sha512) fields for each dependency are the primary indicators of TeamPCP-style substitution — a compromised package will show a changed sha512 hash or a `resolved` URL pointing to a non-registry source | node_modules/.package-lock.json (npm internal lock file written during install): records the exact versions and sources actually installed during the last `npm ci` or `npm install` run, which may diverge from the committed project-level lock file if a supply chain attack modified the registry-resolved version between commits | CI/CD pipeline execution logs containing npm install/ci output: capture stdout from build runners showing 'added X packages' lines and any 'npm warn' or 'npm error' messages referencing unexpected package sources, versions, or integrity failures — these are the real-time indicators of a TeamPCP poisoning event during a build | npm token audit log from registry.npmjs.org: access via `npm token list --json` and the npm website audit log under each package's settings — records every publish event with timestamp, token ID, and source IP, enabling reconstruction of whether any unauthorized publish occurred against your org's packages before staged publishing was enabled | GitHub Actions audit log entries filtered for secrets access and workflow dispatch events: available under Organization > Settings > Audit Log, exportable via GitHub API; shows which workflows accessed `NPM_TOKEN` secrets and when, establishing the publish credential access timeline relevant to assessing pre-remediation exposure

Per-Action IR Details

Step 1: Assess exposure — inventory all npm packages your organization publishes and all npm dependencies consumed directly or transitively in production builds; determine whether any published packages have staged publishing available and not yet enabled

NIST Phase: Preparation

Reference: NIST 800-61r3 §2 — Preparation: establishing IR capability and understanding the environment before an incident occurs

Controls: NIST CM-8 (System Component Inventory), NIST RA-3 (Risk Assessment), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Run `npm ls --all --json > dep-tree.json` in each project root to capture the full transitive dependency graph. For published packages, run `npm access list packages ` to enumerate all packages your org owns and cross-reference against the npm staged publishing opt-in list at registry.npmjs.org. Use `jq` to parse and diff lock files:

`jq '.dependencies | keys' package-lock.json` to surface non-registry sources (look for 'git+', 'file:', 'http' prefixes in version strings).

Evidence: Before inventorying, snapshot current state for baseline comparison: export `package-lock.json` and `npm-shrinkwrap.json` from all production build environments, capture `npm config list --json` output per pipeline runner to document current registry and source configurations, and record the current resolved version of each package as published in the npm registry to detect future drift.

Step 2: Enable staged publishing — for any npm packages your organization maintains, enable the staged publishing feature in npm/GitHub settings to enforce the 2FA-authenticated human approval gate before any version becomes installable (addresses NIST CM-3: Configuration Change Control and NIST SI-7: Software, Firmware, and Information Integrity)

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: implementing controls to prevent further exploitation of the identified attack vector

Controls: NIST CM-3 (Configuration Change Control), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-4 (Impact Analyses), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 6.3 (Require MFA for Externally-Exposed Applications)

Compensating: For packages not yet eligible for staged publishing, enforce branch protection rules in GitHub requiring a signed review before any workflow that calls `npm publish` can execute. Add a pre-publish gate using a GitHub Actions workflow with `environment: production` and required reviewers set, so the publish job pauses at the environment approval step — this manually replicates the staged publishing gate using free GitHub features. Document the approval in a GitHub issue or PR comment for audit trail.

Evidence: Before enabling staged publishing, capture: the npm publish token currently associated with each CI/CD pipeline (`npm token list` with a token that has read:org scope), the GitHub Actions workflow YAML files that contain `npm publish` invocations to establish what automated publishing paths currently exist, and the npm audit log for each package via `npm audit --json` to record the pre-gate baseline of published versions and their integrity hashes.

Step 3: Upgrade npm CLI and configure install source flags — update to npm CLI 11.15.0+ and configure the new install source restriction flags to disallow or explicitly permit non-registry sources (git URLs, file paths, tarball URLs) in project and CI/CD configurations (addresses CIS 7.3: Perform Automated Operating System Patch Management and CIS 2.2: Ensure Authorized Software is Currently Supported)

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: removing the conditions that enabled the threat vector, specifically eliminating non-registry source substitution as an installation path

Controls: NIST SI-2 (Flaw Remediation), NIST CM-7 (Least Functionality), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: For CI/CD environments where npm CLI upgrade requires change control approval, add a pre-install lint step using `grep -rE '(git\+|file:|https?://)' package.json package-lock.json` in each pipeline to fail the build if non-registry sources are detected. Add `.npmrc` entries `save-exact=true` and configure `registry=https://registry.npmjs.org` explicitly to prevent registry substitution. For offline or air-gapped environments, pin to a verified Verdaccio proxy registry instance and block outbound npm registry traffic to all other endpoints at the firewall.

Evidence: Before upgrading, document: the current npm CLI version on all build runners (`npm --version` per runner), any `.npmrc` files present in project roots and user home directories that may specify alternate registries or allow git/file sources, and a grep of all `package-lock.json` files for `resolved` fields containing non-`https://registry.npmjs.org` URLs — these are the exact artifacts TeamPCP-style poisoning would produce to redirect resolution to attacker-controlled sources.

Step 4: Audit CI/CD pipeline credentials — rotate npm publish tokens and audit which pipelines, service accounts, and third-party integrations hold publish access; apply least-privilege scoping so credentials

cannot publish without the staged approval gate (addresses NIST AC-6: Least Privilege, D3-CRO: Credential Rotation, and CIS 5.4: Restrict Administrator Privileges to Dedicated Administrator Accounts)

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: revoking and replacing credentials that represent active publish-path attack surface before confirmed misuse occurs

Controls: NIST AC-6 (Least Privilege), NIST AC-2 (Account Management), NIST IA-5 (Authenticator Management), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Enumerate all active npm automation tokens using `npm token list --json` (requires owner-level token) and document each token's CIDR allowlist, creation date, and associated pipeline. Revoke any token with publish scope that is not bound to a specific CIDR range or that predates the staged publishing rollout. Replace with granular read-only tokens for install workflows and separate scoped publish tokens restricted to the approval-gated workflow. For GitHub Actions, audit repository secrets and organization secrets for any `NPM_TOKEN` entries using the GitHub CLI: `gh secret list --repo /`.

Evidence: Before rotating, preserve: the full output of `npm token list --json` for each package scope as a timestamped record of pre-rotation token state, the GitHub Actions audit log entries (available under Organization Settings > Audit Log, filter by `action:secrets.*` and `action:workflows.*`) showing which workflows accessed publish tokens, and any npm access log entries from the registry showing recent publish events per package — this establishes whether any unauthorized publish occurred before the gate was in place.

Step 5: Update threat model for TeamPCP activity — incorporate TeamPCP's active package poisoning campaigns into your threat register against T1195.001, T1554, and T1199; establish detection baselines for unexpected package versions appearing in dependency lock files or build outputs

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: integrating current threat actor intelligence into detection baselines to identify indicators of TeamPCP compromise before build artifacts reach production

Controls: NIST RA-3 (Risk Assessment), NIST SI-4 (System Monitoring), NIST PM-16 (Threat Awareness Program), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Create a YARA rule or bash script to diff `package-lock.json` before and after each CI run, alerting on any change to `resolved`, `integrity` (sha512 hash), or `version` fields for existing dependencies — TeamPCP poisoning would alter these values when a malicious version substitutes a legitimate one. Subscribe to the Socket.dev free tier or `npm audit` RSS feeds for packages in your dependency tree. For MITRE T1195.001 (Compromise Software Supply Chain) detection, configure a GitHub Actions workflow to run `npm audit --json` and compare resolved package hashes against the previous lock file snapshot on every pull request.

Evidence: Capture for threat modeling baseline: the current `integrity` hashes (sha512) for all top-level and critical transitive dependencies from `package-lock.json` as your known-good state, any npm security advisory history for packages in your dependency tree via `npm audit --json > audit-baseline.json`, and the current `node_modules/.package-lock.json` (the internal lock file used by the npm CLI) which records the actual resolved versions installed — divergence between this and the project-level lock file is a TeamPCP-relevant indicator.

Step 6: Communicate findings — brief engineering leadership and AppSec teams on the specific exposure: if your organization publishes npm packages or consumes packages from non-registry sources, the risk is concrete and the mitigations require active opt-in, not passive updates

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: translating threat intelligence and control gaps into organizational awareness and documented risk decisions before the next TeamPCP campaign wave

Controls: NIST IR-4 (Incident Handling), NIST PM-15 (Security and Privacy Groups and Associations), NIST AT-2 (Literacy Training and Awareness), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Produce a one-page exposure brief using the inventory output from Step 1: list each npm package your org publishes, its current staged publishing status (enabled/not enabled), and the count of dependencies resolved from non-registry sources. Frame the risk as concrete: 'TeamPCP has actively poisoned packages on the npm registry; our packages X, Y, Z are currently publishable by any pipeline holding token T without human approval.' Distribute via your existing incident ticketing system (Jira, GitHub Issues) with severity tagging so it enters the engineering backlog rather than email.

Evidence: Before the briefing, assemble supporting evidence: the token audit output from Step 4 showing publish-capable credentials, the lock file diff report from Step 5 showing any non-registry resolved sources currently in production dependencies, and any historical npm advisory matches from `npm audit` to demonstrate that packages in your dependency tree have been previously targeted — this grounds the briefing in organizational-specific data rather than abstract threat reporting.

Step 7: Monitor for follow-up disclosures — track npm security advisories, GitHub changelog updates for staged publishing scope, and threat intelligence on TeamPCP campaign activity for new indicators or expansion of targeted packages

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: establishing continuous monitoring and intelligence feeds to detect scope expansion of TeamPCP campaigns and staged publishing feature changes that affect control effectiveness

Controls: NIST SI-5 (Security Alerts, Advisories, and Directives), NIST CA-7 (Continuous Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 8.2 (Collect Audit Logs)

Compensating: Configure a GitHub Actions scheduled workflow (cron: daily) that runs `npm audit --json` against your dependency tree and posts a summary to a Slack webhook or GitHub issue, flagging any new advisories for packages in your lock file. Subscribe to the npm security advisories RSS feed at `https://github.com/advisories` filtered by `ecosystem:npm`. For TeamPCP-specific tracking, monitor the Socket.dev blog, CISA Known Exploited Vulnerabilities catalog, and the OpenSSF package-analysis project feeds — all free. Set a calendar-based review of `package-lock.json` integrity hashes every sprint cycle as a lightweight compensating control for teams without automated SCA tooling.

Evidence: Establish persistent collection of: npm registry metadata for your critical dependencies via `npm view --json` on a scheduled basis to detect unexpected new versions published between your approved upgrade cycles, GitHub Actions workflow run logs for any pipeline that calls `npm install` or `npm ci` — these logs contain the resolved package versions and will show if a dependency resolved to an unexpected version mid-campaign, and the npm access audit log for your organization's published packages (available via npm website under package settings) to detect unauthorized publish attempts that were blocked by staged publishing.

Detection Guidance

Focus detection on the software supply chain pipeline, not just runtime endpoints. Key areas to monitor:

npm Publish Anomalies (AU-2: Event Logging; AU-6: Audit Record Review): Review npm audit logs for any package publish events originating from CI/CD service accounts outside of expected release windows. Unexpected version increments, especially patch-level bumps from automated pipelines, should trigger review. If staged publishing is enabled, alert on any publish workflow that does not complete the human approval step within a defined time window.

Dependency Lock File Changes (CIS 8.2: Collect Audit Logs): Hunt for unexpected changes to package-lock.json or yarn.lock files in source control, particularly changes that alter resolved URLs, integrity hashes, or introduce non-registry sources (git:// or http:// prefixes in dependency entries). These changes should be reviewed against an approved change record.

Non-Registry Install Sources: With npm CLI 11.15.0+ flags available, audit existing projects for any dependencies currently resolved from git URLs, file paths, or tarball URLs. Flag any that are not explicitly approved. Pre-upgrade, search codebase and CI configurations for install commands that include --save with non-registry source patterns.

CI/CD Credential Anomalies (NIST AC-2: Account Management; D3-LAM: Local Account Monitoring): Monitor for npm token usage outside of expected pipeline runs, off-hours publish attempts, token use from unexpected IP ranges, or token use following a pipeline credential rotation event. Alert on any publish token used without a corresponding pipeline execution log.

Package Integrity Verification (D3-FMBV: File Magic Byte Verification; D3-SFA: System File Analysis): In build environments, verify package integrity hashes against lock file records before installation completes. Mismatches between expected and actual hashes are a strong indicator of substitution. Integrate Subresource Integrity (SRI) or equivalent hash verification into your CI/CD pipeline.

TeamPCP Indicators: Until confirmed IOCs are published, monitor threat intelligence feeds (MISP, Recorded Future, vendor threat reports) for TeamPCP-attributed package names and hashes. Flag new or updated packages in your dependency tree with sudden popularity increases or maintainer account changes.

Indicators of Compromise

Type	Value	Context	Confidence
TOOL	`Pending — refer to threat intelligence sources tracking TeamPCP for published indicators`	TeamPCP has been observed actively poisoning open-source npm packages at scale; specific package names, registry accounts, and payload hashes attributed to this campaign were not published in the available source material	LOW

Framework Mappings

MITRE-ATTACK

- **T1554** — Compromise Host Software Binary
- **T1078** — Valid Accounts
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1553.002** — Code Signing
- **T1199** — Trusted Relationship

NIST-800-53R5

- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **IA-8** — Identification and Authentication (Non-Organizational Users)
- **SI-7** — Software, Firmware, and Information Integrity

- **CM-3** — Configuration Change Control
- **AT-2** — Literacy Training and Awareness
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A07:2021** — Identification and Authentication Failures
- **A08:2021** — Software and Data Integrity 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
- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **14.2** — Train Workforce Members to Recognize Social Engineering Attacks
- **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

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain

NIST-CSF-2

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1554	Compromise Host Software Binary	Persistence
T1078	Valid Accounts	Defense-Evasion
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1553.002	Code Signing	Defense-Evasion
T1199	Trusted Relationship	Initial-Access

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/05/npm-adds-2fa-gated-publishing-and...	T3
Collection of npm package manager Security Best Practices - GitHub	https://github.com/lirantal/npm-security-best-practices	T3
Security issues related to the npm registry; "vulnerability that would ...	https://www.reddit.com/r/programming/comments/qvgw2p/security_issue...	T3
Security Vulnerability in NPM CLI · Issue #4346 - GitHub	https://github.com/npm/cli/issues/4346	T3
npm - Snyk Vulnerability Database	https://security.snyk.io/package/npm/npm	T3

Generated 2026-05-23 19:02 UTC by TJS Security Command Center