INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-20 18:55 UTC

Shai-Hulud Worm Chains GitHub Actions Weaknesses to Compromise 520M npm/PyPI Downloads with Valid SLSA Provenance

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0345
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	npm ecosystem (169 packages including @tanstack/* 42 packages), PyPI ecosystem, @opensearch-project/opensearch, @uipath/* (57 packages), @mistralai/mistralai, @bitwarden/cli, intercom-client, GitHub Actions, Checkmarx, Docker Hub, VS Code extensions, Bun runtime
Published	2026-05-20T19:30:33+00:00
Discovery Source	Rss:T1 Threatintel

Executive Summary

A worm campaign attributed to TeamPCP compromised 169 npm packages and multiple PyPI packages, affecting an estimated 520 million cumulative downloads by injecting malicious code into projects including TanStack, OpenSearch, UiPath, and Mistral AI. The attack exploited three GitHub Actions weaknesses to move laterally across CI/CD pipelines without stolen credentials, then published poisoned packages bearing valid SLSA Build Level 3 provenance attestations, the same cryptographic trust signal organizations use to verify software supply chain integrity. The May 12, 2026 public release of the worm source code means copycat campaigns are now a near-term certainty, and any organization treating SLSA provenance as a sufficient integrity control is currently operating on a false assumption.

Technical Analysis

The Shai-Hulud campaign (TeamPCP) chains three GitHub Actions weaknesses, workflow injection via untrusted input, excessive GITHUB_TOKEN permissions, and CI/CD cross-repository trust abuse, to achieve credential-free lateral movement across repositories (T1528, T1648, T1195.002). Once inside a build pipeline, the worm modifies package contents and publishes to npm and PyPI under the victim project's identity, generating valid SLSA Build Level 3 provenance because the attestation is produced by the compromised but otherwise legitimate build pipeline. This directly subverts CWE-494 (Download of Code Without Integrity Check)

defenses. Additional TTPs include credential harvesting from workflow environment variables (T1552.001, T1552.004), obfuscated payloads (T1027), masquerading under legitimate package names (T1036.005), exfiltration to code repositories (T1567.001), and defense impairment (T1562.001). Confirmed affected packages include all 42 @tanstack/* packages, 57 @uipath/* packages, @opensearch-project/opensearch, @mistralai/mistralai, @bitwarden/cli, and intercom-client. No CVE has been assigned; the weaknesses are architectural rather than patched-version issues. CWE classification covers CWE-284 (Improper Access Control), CWE-426 (Untrusted Search Path), CWE-522 (Insufficiently Protected Credentials), CWE-829 (Inclusion of Functionality from Untrusted Control Sphere), and CWE-494. Source code was publicly released May 12, 2026, enabling copycat actors. CVSS base score assessed at 9.5 (critical). No CISA KEV entry as of configuration date.

Action Checklist

1. Step 1: Containment. Immediately audit your dependency manifests (package.json, requirements.txt, poetry.lock, yarn.lock) for all confirmed affected packages: @tanstack/* (42 packages), @uipath/* (57 packages), @opensearch-project/opensearch, @mistralai/mistralai, @bitwarden/cli, intercom-client. Pin all affected packages to the last known-clean version published before May 2026 or remove them from builds until clean versions are confirmed. Block outbound CI/CD network egress to npm and PyPI registries except through an approved internal mirror (NIST CM-7, AC-4).
2. Step 2: Detection. Query SIEM and CI/CD logs for: (a) GitHub Actions workflow runs that triggered package publish steps without a matching human-initiated release event (AU-2, AU-6); (b) GITHUB_TOKEN usage outside expected scopes, specifically 'contents: write' or 'packages: write' on repositories that do not normally publish packages; (c) new or modified workflow YAML files committed by bot accounts or external pull requests within the past 90 days; (d) package downloads or installs of the confirmed affected package list from production build agents. Check Docker Hub and VS Code extension manifests for the same package dependencies. Cross-reference installed package hashes against registry-published hashes, note that SLSA BL3 attestations alone are NOT a reliable clean signal for this campaign (CIS 8.2, NIST SI-4, AU-12).
3. Step 3: Eradication. Rotate all secrets exposed in GitHub Actions environment variables across every affected repository, including GITHUB_TOKEN, npm publish tokens, PyPI API keys, cloud provider credentials, and any secrets passed via workflow inputs (NIST IA-5, D3-CRO). Enforce least-privilege GITHUB_TOKEN permissions at the workflow level, set 'permissions: read-all' as the default and grant write scopes only on explicit steps that require them (NIST AC-6, CIS 5.4). Remove or disable any GitHub Actions workflows added or modified by external contributors since January 2026 that have not been reviewed. Rebuild all affected artifacts from source using a clean, isolated build environment after secrets rotation is complete (NIST CM-5, D3-SFA).
4. Step 4: Recovery. Re-run all affected build pipelines from reviewed source commits in an isolated environment after secrets rotation is complete. Validate output package hashes against a trusted out-of-band registry snapshot, do not rely solely on SLSA provenance attestations for this validation given the campaign's ability to generate valid BL3 attestations from a compromised pipeline. Monitor for anomalous outbound connections, new scheduled tasks, or modified system processes on hosts that installed affected package versions (T1543, T1562.001) using endpoint telemetry (NIST SI-4, AU-6). Confirm no persistence mechanisms were deployed before returning affected systems to production.
5. Step 5: Post-Incident. This campaign exposes a specific control gap: SLSA provenance attestation alone cannot distinguish a clean build from a build executed by a compromised pipeline. Update your

software supply chain trust policy to require SLSA provenance AND independent artifact hash verification against a registry snapshot AND human-reviewed release triggers for high-criticality packages. Implement GitHub Actions security hardening per CISA's Defending CI/CD Environments guidance: restrict workflow triggers from forks, require approval for external contributor workflows, and audit GITHUB_TOKEN permission scopes quarterly (NIST SA-12, CIS 7.1, CIS 7.2). Given the May 12 source code release, treat copycat activity as ongoing and maintain elevated monitoring posture for CI/CD anomalies for a minimum of 90 days.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and breach notification review immediately if forensic analysis confirms that any host where poisoned @tanstack/*, @uipath/*, @bitwarden/cli, or intercom-client versions executed has evidence of payload activity beyond the build pipeline (T1543, T1562.001, outbound C2 connections), or if npm publish tokens, PyPI API keys, or cloud provider credentials accessible to affected GitHub Actions workflows are confirmed exfiltrated — either condition may trigger breach notification obligations under GDPR, CCPA, or sector-specific regulations depending on the data accessible to compromised secrets.
Recovery Notes	Do not return any CI/CD pipeline to production until (1) all affected package versions are replaced with clean builds whose SHA-512 hashes are independently verified against pre-May-2026 registry snapshots via Sigstore/Rekor transparency logs — not solely via SLSA attestations — and (2) all GitHub Actions workflows across affected repositories have been reviewed line-by-line by a human reviewer for injected steps. Maintain elevated monitoring on all hosts where poisoned packages executed for a minimum of 90 days post-eradication, specifically hunting for T1543 persistence and T1562.001 defense evasion given the Shai-Hulud payload's known capability; given the May 12 public source code release enabling copycat variants, treat any new anomalous CI/CD publish event for the affected package namespaces (@tanstack/*, @uipath/*, etc.) as a potential re-infection requiring immediate containment.

Forensic Artifacts	GitHub Actions audit log entries (event types: workflow_run.completed, git.push targeting .github/workflows/ paths, packages.publish) for the past 90 days across all repositories in affected orgs — these logs capture the Shai-Hulud worm's cross-repository lateral movement pattern and GITHUB_TOKEN scope abuse without requiring credential exfiltration | npm package tarball SHA-512 hashes from node_modules/.package-lock.json and per-package dist-info/direct_url.json on build agents — hashes for poisoned @tanstack/*, @uipath/*, @opensearch-project/opensearch, @mistralai/mistralai, @bitwarden/cli, and intercom-client versions will diverge from registry-published dist.integrity values when compared via `npm view @ dist.integrity` | SLSA BL3 provenance attestation bundles for affected package versions retrieved from Sigstore/Rekor transparency log — these attestations will appear cryptographically valid but were generated by the Shai-Hulud-compromised pipeline; the provenance subject (build invocation URI and trigger event) will reference an automated workflow run rather than a human-initiated release, which is the forensic discriminator | Git history patches for .github/workflows/ directories in affected repositories (`git log --all --patch --follow -- .github/workflows/`) — the Shai-Hulud worm's workflow injection will appear as commits from bot accounts or external PR merges adding or modifying publish workflow steps, with the injected steps typically appearing legitimate (e.g., wrapped inside existing release jobs) to evade casual review | Process execution and network telemetry from build agents during the contamination window — specifically Sysmon Event ID 4688 (Windows) or auditd execve records (Linux) for child processes spawned by node or python during package install scripts (npm install lifecycle hooks and PyPI setup.py/post-install hooks are the payload delivery mechanism), plus any outbound TCP connections to non-registry endpoints that may represent TeamPCP C2 infrastructure

Per-Action IR Details

Step 1: Containment — Immediately audit your dependency manifests (package.json, requirements.txt, poetry.lock, yarn.lock) for all confirmed affected packages: @tanstack/* (42 packages), @uipath/* (57 packages), @opensearch-project/opensearch, @mistralai/mistralai, @bitwarden/cli, intercom-client. Pin all affected packages to the last known-clean version published before May 2026 or remove them from builds until clean versions are confirmed. Block outbound CI/CD network egress to npm and PyPI registries except through an approved internal mirror (NIST CM-7, AC-4).

NIST Phase: Containment

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

Controls: NIST CM-7 (Least Functionality) — restrict CI/CD runners from reaching public npm/PyPI registries directly, NIST AC-4 (Information Flow Enforcement) — enforce registry egress only through approved internal mirror proxy, CIS 2.2 (Ensure Authorized Software is Currently Supported) — flag affected package versions as unauthorized pending clean-version confirmation, CIS 2.3 (Address Unauthorized Software) — remove or quarantine installs of poisoned @tanstack/*, @uipath/*, @opensearch-project/opensearch, @mistralai/mistralai, @bitwarden/cli, intercom-client versions

Compensating: Run `npm ls --all --json > dep-snapshot.json` and `pip freeze > reqs-snapshot.txt` on every build agent before making changes — preserve the pre-remediation state. Use `jq` to diff dep-snapshot.json against the confirmed affected package list. For registry egress blocking on a budget: configure `/etc/hosts` or iptables OUTPUT rules on build agents to null-route registry.npmjs.org and pypi.org, then point package managers to a local Verdaccio (npm) or devpi (PyPI) instance. For lock-file hash verification without SIEM, run `npm ci --dry-run` with `--audit` and compare reported resolved hashes against pre-May-2026 registry snapshots manually captured from Sigstore/Rekor transparency log entries.

Evidence: Capture BEFORE pinning or removing packages: (1) Full dependency tree snapshots — `npm ls --all --json` and `pip show ` output for all affected packages, preserving installed version strings and resolved tarball URLs. (2) Package tarball SHA-512 hashes from node_modules/.package-lock.json and pip's direct_url.json (located at site-packages/-.dist-info/direct_url.json) — these hashes will diverge from registry-published hashes if the poisoned

tarball was installed. (3) CI/CD runner disk images or container layer snapshots before any cleanup, to preserve injected payload artifacts. (4) Network connection logs from build agents showing outbound calls to npm/PyPI registries during the compromised build window (netstat -antp output or cloud VPC flow logs filtered on destination port 443 to registry IPs).

Step 2: Detection — Query SIEM and CI/CD logs for: (a) GitHub Actions workflow runs that triggered package publish steps without a matching human-initiated release event (AU-2, AU-6); (b) GITHUB_TOKEN usage outside expected scopes — specifically 'contents: write' or 'packages: write' on repositories that do not normally publish packages; (c) new or modified workflow YAML files committed by bot accounts or external pull requests within the past 90 days; (d) package downloads or installs of the confirmed affected package list from production build agents. Check Docker Hub and VS Code extension manifests for the same package dependencies. Cross-reference installed package hashes against registry-published hashes — note that SLSA BL3 attestations alone are NOT a reliable clean signal for this campaign (CIS 8.2, NIST SI-4, AU-12).

NIST Phase: Detection Analysis

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

Controls: NIST AU-2 (Event Logging) — ensure GitHub Actions audit log events (workflow_run, push to .github/workflows) are captured and retained, NIST AU-6 (Audit Record Review, Analysis, and Reporting) — review CI/CD pipeline logs for anomalous publish events not correlated to human release triggers, NIST AU-12 (Audit Record Generation) — verify build agents generate records of package install events including resolved hashes, NIST SI-4 (System Monitoring) — monitor for GITHUB_TOKEN permission scope escalation and cross-repository workflow invocations consistent with Shai-Hulud lateral movement, CIS 8.2 (Collect Audit Logs) — confirm GitHub Actions audit logs, npm publish logs, and PyPI upload logs are being ingested

Compensating: Without SIEM: Use the GitHub REST API (`GET /orgs/{org}/audit-log?phrase=action:workflows.completed&per_page=100`) with a PAT to pull audit log events and pipe to `jq` filtering on `conclusion!=success` or `actor` matching bot accounts. For GITHUB_TOKEN scope abuse detection, use `gh api /repos/{owner}/{repo}/actions/runs --jq '.workflow_runs[] | select(.event=="push") | {id, head_commit, created_at}'` and cross-reference against git tag creation timestamps. To detect the Shai-Hulud lateral movement pattern (workflow-to-workflow privilege escalation), grep all .github/workflows/*.yml files across repos for `uses:` references pointing to workflows in recently-compromised upstream repos: `find . -path '*/.github/workflows/*.yml' -exec grep -l 'uses:.*@' {} \;`. For package hash verification without SIEM, use `cosign verify-attestation --type slsa` against Sigstore/Rekor and separately validate tarball SHA against npm registry metadata via `npm view @ dist.integrity`.

Evidence: Capture BEFORE any workflow modifications: (1) GitHub Actions audit log export for the past 90 days filtered on events: `workflow_run.completed`, `workflow_job.completed`, `git.push` to `.github/workflows/` paths — downloadable via GitHub API or GitHub Enterprise audit log streaming. (2) Full git log with patches for all .github/workflows/ directories across affected repositories: `git log --all --patch --follow -- .github/workflows/` — preserves the exact YAML diff introduced by the Shai-Hulud worm's workflow injection. (3) GITHUB_TOKEN permission grants visible in workflow run logs under the 'Set up job' step — screenshot or export the permissions block before any workflow is modified. (4) npm registry publish timestamps for affected package versions (retrievable via `npm view @tanstack/react-query time --json`) cross-referenced against release tag creation times in the upstream repos to identify the publish-without-human-release pattern. (5) SLSA provenance attestation bundles for installed package versions — retrieve via `npm view @ --json | jq .dist.attestations` — these attestations will appear valid but were generated by the compromised pipeline; preserve them as evidence of the trust-signal abuse.

Step 3: Eradication — Rotate all secrets exposed in GitHub Actions environment variables across every affected repository, including GITHUB_TOKEN, npm publish tokens, PyPI API keys, cloud provider credentials, and any secrets passed via workflow inputs (NIST IA-5, D3-CRO). Enforce least-privilege GITHUB_TOKEN permissions at the workflow level — set 'permissions: read-all' as the default and grant write scopes only on explicit steps that require them (NIST AC-6, CIS 5.4). Remove or disable any GitHub Actions workflows added or modified by external contributors since January 2026 that have not been reviewed. Rebuild all affected artifacts from source using a clean, isolated build environment after secrets rotation (NIST

CM-5, D3-SFA).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IA-5 (Authenticator Management) — invalidate and reissue all npm publish tokens, PyPI API keys, and cloud credentials that were accessible to GitHub Actions environment variables in affected repositories, NIST AC-6 (Least Privilege) — restrict GITHUB_TOKEN to minimum required scopes; default `permissions: read-all` with explicit per-step overrides, NIST CM-5 (Access Restrictions for Change) — gate artifact publication on human-approved release triggers; remove bot-accessible publish permissions, CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts) — ensure npm publish and PyPI upload tokens are scoped to dedicated service accounts, not shared org-level tokens accessible to all workflow runs

Compensating: For teams without secrets management infrastructure: Revoke npm publish tokens immediately via `npm token revoke ` (list with `npm token list --json`) and PyPI API keys via the PyPI account settings UI. For GitHub org-level secret rotation, use the GitHub CLI: `gh secret set NPM_TOKEN --org --body `. To audit GITHUB_TOKEN scope escalation across all workflow files without a dedicated tool, run: `grep -r 'permissions' .github/workflows/*.yml` and flag any file granting `contents: write` or `packages: write` that was not present before January 2026. For isolated rebuild environments without cloud infrastructure, use a fresh Docker container (`docker run --network=none --rm -v $(pwd):/src node:lts-alpine sh -c 'cd /src && npm ci && npm pack'`) with `--network=none` to prevent exfiltration during the clean build.

Evidence: Capture BEFORE rotating secrets or removing workflows: (1) Complete export of all GitHub Actions secrets names (not values) per repository via `gh api /repos/{owner}/{repo}/actions/secrets --jq '.secrets[].name'` — establishes the full scope of potentially compromised credentials. (2) Workflow YAML files added or modified since January 2026 that contain `secrets:` references or `env:` blocks with secret variables — these files may contain the Shai-Hulud worm's injected payload steps; preserve original content before deletion. (3) GitHub Actions workflow run logs for all publish jobs in the contamination window, specifically the step-level logs showing the exact commands executed by injected workflow steps — downloadable via `gh run view --log`. (4) npm access token last-used timestamps via `npm token list --json` — tokens used during the compromised window after the TeamPCP intrusion date are presumed compromised regardless of whether the secret value was directly exfiltrated.

Step 4: Recovery — Re-run all affected build pipelines from pinned, reviewed source commits in an isolated environment after secrets rotation is complete. Validate output package hashes against a trusted out-of-band registry snapshot — do not rely solely on SLSA provenance attestations for this validation given the campaign's ability to generate valid BL3 attestations from a compromised pipeline. Monitor for anomalous outbound connections, new scheduled tasks, or modified system processes on hosts that installed affected package versions (T1543, T1562.001) using endpoint telemetry (NIST SI-4, AU-6). Confirm no persistence mechanisms were deployed before returning affected systems to production.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-4 (System Monitoring) — monitor hosts that installed poisoned @tanstack/*, @uipath/*, or intercom-client versions for T1543 (Create or Modify System Process) and T1562.001 (Impair Defenses: Disable or Modify Tools) indicators, NIST AU-6 (Audit Record Review, Analysis, and Reporting) — continuously review endpoint and network logs for C2 beaconing or data exfiltration from hosts where affected packages executed in CI/CD or production Node.js/Python runtimes, CIS 7.2 (Establish and Maintain a Remediation Process) — track clean-build validation status per affected package through to confirmed production deployment, CIS 4.4 (Implement and Manage a Firewall on Servers) — block unexpected outbound connections from hosts where affected packages ran, particularly to non-standard endpoints that may represent C2 infrastructure planted by the worm payload

Compensating: For T1543/T1562.001 detection without EDR: Deploy Sysmon with SwiftOnSecurity config and monitor Event ID 7045 (new service installed) and Event ID 4698 (scheduled task created) on Windows CI/CD agents; on Linux runners use `systemctl list-units --type=service --state=running > services-baseline.txt` and `crontab -l && cat /etc/cron.*/* > cron-baseline.txt` before and after the compromised build window and diff. For anomalous outbound connection detection, run `ss -antp` or `netstat -antp` on affected hosts and compare against a known-good baseline; use Wireshark or `tcpdump -i any -w capture.pcap host ` for 24-hour network capture on hosts where poisoned

packages executed in production. For artifact hash validation without a commercial registry: compute `sha512sum` of rebuilt tarballs and compare against pre-compromise npm registry metadata fetched via `curl https://registry.npmjs.org/@tanstack/react-query/ | jq .dist.integrity`.

Evidence: Capture BEFORE returning any system to production: (1) Memory dump of any Node.js or Python process that loaded a poisoned package version during the contamination window — use `procdump -ma ` (Windows) or `gcore ` (Linux) to capture runtime state including any injected payload executing in-process. (2) File system timeline from affected build agents covering the contamination window — `find / -newer /tmp/reference-timestamp -ls > filesystem-timeline.txt` — to identify files written by the worm payload (backdoors, credential harvesters, persistence scripts). (3) Network flow records from affected hosts during the window when poisoned packages were installed and executed, specifically any DNS lookups or TCP connections to non-registry, non-CDN endpoints that could represent C2 beaconing by the Shai-Hulud payload. (4) Process execution logs (Sysmon Event ID 4688 on Windows or auditd `execve` records on Linux) filtered on processes spawned by the Node.js runtime (node.exe / node) or Python interpreter during the compromised build — to identify child process execution (T1059) triggered by the malicious package install scripts.

Step 5: Post-Incident — This campaign exposes a specific control gap: SLSA provenance attestation alone cannot distinguish a clean build from a build executed by a compromised pipeline. Update your software supply chain trust policy to require SLSA provenance AND independent artifact hash verification against a pinned registry snapshot AND human-reviewed release triggers for high-criticality packages. Implement GitHub Actions security hardening per CISA's Defending CI/CD Environments guidance: restrict workflow triggers from forks, require approval for external contributor workflows, and audit GITHUB_TOKEN permission scopes quarterly (NIST SA-12, CIS 7.1, CIS 7.2). Given the May 12 source code release, treat copycat activity as ongoing and maintain elevated monitoring posture for CI/CD anomalies for a minimum of 90 days.

NIST Phase: Post Incident

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

Controls: NIST SA-12 (Supply Chain Protection) — formalize software supply chain trust policy requiring SLSA provenance plus independent artifact hash verification; document the Shai-Hulud campaign as a case study demonstrating SLSA BL3 attestation insufficiency when the pipeline itself is compromised, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — extend vulnerability management scope to include CI/CD pipeline configurations and GitHub Actions workflow YAML as first-class managed assets subject to regular review, CIS 7.2 (Establish and Maintain a Remediation Process) — establish SLA for remediating newly identified poisoned package versions given the May 12 source code release enabling copycat campaigns, NIST AU-11 (Audit Record Retention) — retain all GitHub Actions audit logs, build logs, and package publish records for a minimum period aligned with the 90-day elevated monitoring posture, NIST AU-6 (Audit Record Review, Analysis, and Reporting) — schedule weekly CI/CD anomaly review for the 90-day post-incident window, specifically hunting for Shai-Hulud copycat patterns: new external workflow contributors, unexpected package publish events, and GITHUB_TOKEN scope expansions

Compensating: For organizations without a formal supply chain security program: Implement OpenSSF Scorecard (`scorecard --repo github.com//`) on all repositories consuming the affected packages — the 'Token-Permissions' and 'Branch-Protection' checks directly surface the GITHUB_TOKEN over-privilege pattern exploited by Shai-Hulud. For human-reviewed release gate enforcement without paid CI/CD tooling, add a required CODEOWNERS approval on `.github/workflows/` and `package.json`/`pyproject.toml` files using GitHub's native branch protection required reviewers feature. For 90-day copycat monitoring without SIEM, schedule a weekly cron job to pull the npm and PyPI package metadata for your top 50 dependencies and diff against your pinned hash baseline: `npm view @ dist.integrity` stored in a version-controlled file and compared via `git diff`.

Evidence: Capture for lessons-learned and threat intelligence sharing: (1) Complete timeline of the Shai-Hulud worm's lateral movement through your CI/CD environment — mapping which repository's compromised workflow triggered downstream repository infections — reconstructed from GitHub Actions audit logs and workflow run timestamps. (2) All SLSA BL3 provenance attestation bundles generated by compromised pipeline runs for the affected package versions — these are high-value evidence demonstrating the attestation-spoofing capability and should be submitted to the SLSA community and Sigstore maintainers. (3) Any novel C2 infrastructure, payload hashes, or injected code snippets

recovered from build agent forensics — format as STIX 2.1 indicators and share with the npm security team, PyPI security team, and GitHub Security via coordinated disclosure, referencing the TeamPCP attribution. (4) Documented mapping of which internal applications consumed affected package versions in production versus CI-only environments — this scoping evidence is required for any regulatory breach notification assessment if the worm payload exfiltrated credentials or data from production systems.

Detection Guidance

Primary detection surface is CI/CD pipeline and package registry activity, not endpoint logs. Key indicators: (1) GitHub Actions workflow runs that executed a package publish step (npm publish, twine upload, docker push) where the triggering event was a pull_request or workflow_dispatch from an external actor rather than a protected branch push or signed tag, query GitHub audit log API for 'action: workflows.completed' with publish job steps and cross-check trigger actor against repository collaborator list. (2) GITHUB_TOKEN permission grants of 'contents: write', 'packages: write', or 'id-token: write' in workflow files added or modified within the past 90 days by accounts not on your internal team roster. (3) New or modified .github/workflows/*.yml files committed via pull requests from forked repositories, this is the primary injection vector. (4) Package version bumps on affected packages (@tanstack/*, @uipath/*, @opensearch-project/opensearch, @mistralai/mistralai, @bitwarden/cli, intercom-client) published between approximately April 2026 and present that were not preceded by a corresponding GitHub release event authored by a known maintainer. (5) Outbound DNS or HTTPS from build agents to non-registry destinations during package install phases. (6) Presence of SLSA provenance attestations on packages where the build repository does not match the expected upstream source, the worm generates valid attestations from the compromised pipeline, so the attestation signer identity is the verification point, not the attestation existence. (7) On hosts that installed affected packages: unexpected child processes spawned from Node.js or Python interpreter, new cron jobs or systemd units, or outbound connections to code repository hosts (GitHub, GitLab) from non-developer systems. MITRE coverage: T1195.002, T1648, T1528, T1552.001, T1567.001.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://registry.npmjs.org/@tanstack/*`	All 42 @tanstack/* npm packages confirmed affected; verify installed versions against known-clean hashes before trusting	HIGH
URL	`https://registry.npmjs.org/@uipath/*`	57 @uipath/* npm packages confirmed affected by campaign	HIGH
URL	`https://registry.npmjs.org/@opensearch-project/opensearch`	Confirmed affected npm package	HIGH
URL	`https://registry.npmjs.org/@mistralai/mistralai`	Confirmed affected npm package	HIGH
URL	`https://registry.npmjs.org/@bitwarden/cli`	Confirmed affected npm package; elevated risk given credential manager context	HIGH

Type	Value	Context	Confidence
URL	`https://registry.npmjs.org/intercom-client`	Confirmed affected npm package	HIGH

Framework Mappings

MITRE-ATTACK

- **T1528** — Steal Application Access Token
- **T1543** — Create or Modify System Process
- **T1204.002** — Malicious File
- **T1553.002** — Code Signing
- **T1072** — Software Deployment Tools
- **T1552.001** — Credentials In Files
- **T1648** — Serverless Execution
- **T1485** — Data Destruction
- **T1562.001** — Disable or Modify Tools
- **T1027** — Obfuscated Files or Information
- **T1195.002** — Compromise Software Supply Chain
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1552.004** — Private Keys
- **T1554** — Compromise Host Software Binary
- **T1567.001** — Exfiltration to Code Repository
- **T1650** — Acquire Access
- **T1059.004** — Unix Shell
- **T1546** — Event Triggered Execution
- **T1609** — Container Administration Command
- **T1036.005** — Match Legitimate Resource Name or Location

NIST-800-53R5

- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **CM-7** — Least Functionality
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **AC-3** — Access Enforcement
- **IA-5** — Authenticator Management
- **SR-2** — Supply Chain Risk Management Plan

- **SC-13** — Cryptographic Protection

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures
- **A01:2021** — Broken Access Control
- **A04:2021** — Insecure Design
- **A07:2021** — Identification and Authentication Failures

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **6.1** — Establish an Access Granting Process
- **6.2** — Establish an Access Revoking Process
- **5.2** — Use Unique Passwords
- **6.3** — Require MFA for Externally-Exposed Applications
- **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(a)(1)** — Access Control
- **164.308(a)(5)(ii)(D)** — Password Management
- **164.312(d)** — Person or Entity Authentication

ISO-27001-2022

- **A.5.34** — Privacy and protection of personal information
- **A.5.21** — Managing information security in the ICT supply chain
- **A.8.24** — Use of cryptography

NIST-CSF-2

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1528	Steal Application Access Token	Credential-Access
T1543	Create or Modify System Process	Persistence
T1204.002	Malicious File	Execution

Technique ID	Technique Name	Tactic
T1553.002	Code Signing	Defense-Evasion
T1072	Software Deployment Tools	Execution
T1552.001	Credentials In Files	Credential-Access
T1648	Serverless Execution	Execution
T1485	Data Destruction	Impact
T1562.001	Disable or Modify Tools	Defense-Evasion
T1027	Obfuscated Files or Information	Defense-Evasion
T1195.002	Compromise Software Supply Chain	Initial-Access
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1552.004	Private Keys	Credential-Access
T1554	Compromise Host Software Binary	Persistence
T1567.001	Exfiltration to Code Repository	Exfiltration
T1650	Acquire Access	Resource-Development
T1059.004	Unix Shell	Execution
T1546	Event Triggered Execution	Privilege-Escalation
T1609	Container Administration Command	Execution
T1036.005	Match Legitimate Resource Name or Location	Defense-Evasion

Sources

Source	URL	Tier
Unit 42	https://unit42.paloaltonetworks.com/monitoring-npm-supply-chain-att...	T3
	https://www.microsoft.com/en-us/security/blog/2025/12/09/shai-hulud...	T1
	https://unit42.paloaltonetworks.com/npm-supply-chain-attack/	T3
	https://www.recordedfuture.com/research/2025-cloud-threat-hunting-d...	T3
Mass Supply Chain Attack Hits TanStack, Mistral AI npm and PyPI ...	https://safedep.io/mass-npm-supply-chain-attack-tanstack-mistral	T3

DISCLAIMER

This intelligence report is produced by Tech Jacks Solutions Security Command Center (SCC) for informational purposes only. It does not constitute professional security advice, legal counsel, or an incident response engagement. The information herein is derived from publicly available sources and AI-assisted analysis; while every effort is made to ensure accuracy, Tech Jacks Solutions makes no warranties regarding completeness or timeliness. Organizations should conduct their own validation and consult qualified security professionals before taking action based on this report. Tech Jacks Solutions is not liable for any damages resulting from the use of this information.

Generated 2026-05-20 18:55 UTC by TJS Security Command Center