INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-03 06:51 UTC

SLSA Provenance Weaponized via Credential-Free CI/CD Injection: Shai-Hulud npm Campaign Evolution

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0398
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	npm registry, PyPI, @bitwarden/cli, @tanstack/* packages (including @tanstack/react-router ~12.7M weekly downloads), @redhat-cloud-services namespace (32 packages, ~80K weekly downloads), @opensearch-project/opensearch, @mistralai/mistralai, @uipath/* (57 packages), GitHub Actions, CircleCI, AWS/GCP/Azure, Kubernetes, HashiCorp Vault, VS Code extensions, Docker Hub
Published	2026-06-02T17:30:33+00:00
Discovery Source	Rss:T1 Threatintel

Executive Summary

A sophisticated supply chain campaign known as Shai-Hulud has evolved to compromise CI/CD pipelines without stolen credentials, producing malicious npm packages that carry valid, signed software provenance attestations - the integrity signals organizations rely on to verify package safety. Confirmed affected packages include widely-used open-source libraries spanning TanStack (@tanstack/react-router and related packages at ~12.7M weekly downloads as of June 2026), @redhat-cloud-services (32 packages, ~80K weekly downloads), and enterprise tooling spanning AWS, GCP, Azure, Kubernetes, and HashiCorp Vault environments (corroborated by StepSecurity and Unit 42). The public release of the Mini Shai-Hulud source code on May 12, 2026 has democratized the attack technique, meaning any organization consuming npm or PyPI packages through automated pipelines faces elevated risk regardless of vendor namespace or signed provenance status. Important: The June 2026 Red Hat namespace compromise and Miasma payload characteristics are medium-confidence pending independent verification and should not yet drive production remediation timelines, though defensive audits should proceed immediately.

Technical Analysis

The Shai-Hulud campaign has pivoted from credential-phishing-based package hijacking to pipeline configuration injection, enabling attackers to introduce malicious build artifacts into CI/CD pipelines that then generate legitimate SLSA provenance attestations for the resulting artifacts. This directly undermines SLSA's integrity model: the provenance is technically valid because the compromised pipeline produced it. Affected packages include @tanstack/react-router and related @tanstack/* packages (~12.7M weekly downloads), 32 @redhat-cloud-services namespace packages (~80K weekly downloads), @bitwarden/cli, @opensearch-project/opensearch, @mistralai/mistralai, @uipath/* (57 packages), and potentially VS Code extensions and Docker Hub images. Secondary propagation vectors include GitHub Actions and CircleCI pipeline injection, scheduled task abuse (T1053), and software deployment tool exploitation (T1072). Relevant CWEs: CWE-506 (Embedded Malicious Code), CWE-345 (Insufficient Verification of Data Authenticity), CWE-494 (Download of Code Without Integrity Check), CWE-829 (Inclusion of Functionality from Untrusted Control Sphere), CWE-312 (Cleartext Storage of Sensitive Information), CWE-284 (Improper Access Control). MITRE techniques include T1195.002 (Compromise Software Supply Chain), T1554 (Compromise Client Software Binary), T1553.002 (Code Signing), T1528 (Steal Application Access Token), T1552.001 (Credentials in Files), T1552.004 (Private Keys), T1059/T1059.004 (Command and Scripting Interpreter), T1567 (Exfiltration Over Web Service), T1027 (Obfuscated Files), T1611 (Escape to Host), T1485 (Data Destruction), and T1036.003 (Masquerading). No CVE assigned. No CISA KEV entry. Corroborated by StepSecurity and Unit 42; Red Hat namespace and Miasma payload specifics remain medium-confidence pending primary source verification.

Action Checklist

1. Step 1: Containment. Immediately audit your npm and PyPI dependency trees for any direct or transitive dependencies on @tanstack/* (especially @tanstack/react-router), @redhat-cloud-services/* (all 32 packages), @bitwarden/cli, @opensearch-project/opensearch, @mistralai/mistralai, and @uipath/* packages. Identify the last package version published prior to May 1, 2026 (before the confirmed attack window) for each affected namespace, then audit that baseline version's transitive dependencies to rule out compromise via supply chain path. Once a known-good baseline is confirmed, lock package versions in package.json and requirements.txt to those pinned hashes using 'npm ci' with lockfile integrity enforcement (CIS 2.1, Software Inventory; NIST CM-2 baseline configuration). Block automated pipeline promotion of any package version published after May 12, 2026 from the affected namespaces until integrity can be independently verified.
2. Step 2: Detection. Query CI/CD pipeline logs for unexpected outbound connections, credential access patterns, or new environment variable reads during build stages (NIST AU-6, Audit Record Review; CIS 8.2, Collect Audit Logs). Specifically look for: shell spawning from within Node.js build processes (T1059.004), access to cloud metadata endpoints (169.254.169.254 or equivalent) from build runners, unexpected npm publish events in pipeline logs, and any SLSA provenance attestation generated after a pipeline config change that lacks a corresponding reviewed pull request (this detection requires integration of SLSA attestation verification into your CI/CD audit pipeline as a prerequisite control). Cross-reference pipeline execution timestamps against GitHub Actions workflow modification history. Alert on file system analysis signals: modifications to CI config files (.github/workflows/*.yml, .circleci/config.yml) not tied to approved commits.
3. Step 3: Eradication. Roll back to the last known-good lockfile state predating May 1, 2026 for all affected packages, or replace affected dependencies with verified alternatives where available. Rotate all secrets, API tokens, and credentials accessible to compromised or potentially compromised build runners,

including AWS/GCP/Azure service account keys, Kubernetes service account tokens, and HashiCorp Vault tokens (NIST IA-5 Authenticator Management). Revoke and reissue any code-signing certificates or npm publish tokens associated with affected pipelines. Re-examine all VS Code extensions and Docker Hub images built in potentially affected pipelines for unexpected outbound network activity or credential access patterns during runtime.

4. Step 4: Recovery. After rotating credentials and restoring clean lockfiles, re-run full dependency audits using 'npm audit' and an independent SCA tool against the restored state. Validate that SLSA provenance attestations for newly built artifacts trace to pipeline configurations that have been independently reviewed post-remediation; do not treat provenance alone as integrity proof given this campaign (NIST SI-7, Software, Firmware, and Information Integrity). Monitor build pipeline outputs for 30 days post-remediation for recurrence, with alerting on any new outbound network connections from build runners.

5. Step 5: Post-Incident. Document the control gap: SLSA provenance attestation is a pipeline-integrity signal, not an artifact-integrity guarantee when the pipeline itself can be compromised. Update your software supply chain security policy to require out-of-band build pipeline integrity verification, separation of duties between pipeline configuration and pipeline execution (NIST AC-5, Separation of Duties), and mandatory human review of all CI/CD config changes before merge (CIS 4.6, Securely Manage Enterprise Assets and Software). Evaluate adoption of SLSA Build Level 3 controls and Sigstore Cosign with transparency log verification as layered controls. Conduct a tabletop exercise simulating a credential-free pipeline injection scenario against your specific CI/CD stack.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to executive leadership, legal, and external IR retainer immediately if forensic evidence confirms that build runner OIDC tokens were used to access AWS/GCP/Azure production environments, Kubernetes clusters, or HashiCorp Vault instances beyond the CI/CD boundary, or if any affected package (@tanstack/react-router with ~12.7M weekly downloads, @bitwarden/cli, or @uipath/* enterprise tooling) was deployed to a production environment that processes PII, PHI, or financial data — the latter triggers mandatory breach notification assessment under GDPR Article 33, HIPAA §164.400, and applicable state breach notification statutes.
Recovery Notes	Recovery is not complete at lockfile restore — because Shai-Hulud produces packages with valid SLSA provenance attestations, any artifact built during the compromise window must be treated as potentially tainted regardless of its attestation status and should be rebuilt from source in a freshly provisioned, isolated build environment with a clean runner image. Monitor all build runner service account identities (AWS IAM roles, GCP service accounts, Azure managed identities) for 30 days post-rotation using CloudTrail/Audit Log alerting on any API calls that do not match the expected CI/CD action baseline, specifically flagging sts:AssumeRole, iam:CreateAccessKey, or equivalent privilege escalation calls. Re-evaluate recovery completion only after a full dependency audit on the restored state passes both 'npm audit' and an independent SCA scan with zero high/critical findings, and after Sigstore Rekor confirms no unauthorized attestations exist for your pipeline OIDC identity post-remediation.

Forensic Artifacts	npm registry publish audit log entries for @tanstack/react-router, @redhat-cloud-services/*, @bitwarden/cli, @mistralai/mistralai, and @uipath/* versions published after May 12, 2026 — each entry includes the npm automation token ID and IP address used for publish, which can be correlated against known build runner egress IPs to confirm whether a legitimate or hijacked pipeline identity performed the publish | Sigstore Rekor transparency log entries for SLSA provenance attestations tied to the compromised pipeline's GitHub Actions OIDC subject (format: 'https://github.com///.github/workflows/.yml@refs/heads/main') — entries with attestation timestamps that postdate a workflow YAML modification without a corresponding reviewed PR are primary evidence of Shai-Hulud's credential-free injection mechanic | GitHub Actions workflow run logs and the .github/workflows/*.yml git blame history showing unauthorized modifications, specifically looking for injected steps that call 'actions/github-script' or raw curl/wget commands targeting external endpoints or reading $ACTIONS_ID_TOKEN_REQUEST_TOKEN and $ACTIONS_ID_TOKEN_REQUEST_URL environment variables | Cloud provider metadata endpoint access logs — on AWS, VPC Flow Logs showing connections from build runner subnets to 169.254.169.254 on TCP/80 outside of expected IMDSv2 initialization windows; on GCP, similar flows to metadata.google.internal; these connections during npm install or build stages are high-fidelity indicators that a malicious postinstall script in a compromised @tanstack/* or @redhat-cloud-services/* package attempted to harvest cloud credentials | node_modules/.package-lock.json and the npm cache tarballs (~/.npm/_cacache/content-v2/) for the specific compromised package versions — the cached tarballs preserve the exact malicious package content as it was downloaded, including any obfuscated postinstall scripts in package.json 'scripts.postinstall' or 'scripts.install' fields, and can be statically analyzed with 'node --inspect' or YARA rules targeting common Shai-Hulud payload patterns (base64-encoded eval blocks, dynamic require() of network modules, process.env enumeration)

Per-Action IR Details

Step 1: Containment — Immediately audit your npm and PyPI dependency trees for any direct or transitive dependencies on @tanstack/* (especially @tanstack/react-router), @redhat-cloud-services/* (all 32 packages), @bitwarden/cli, @opensearch-project/opensearch, @mistralai/mistralai, and @uipath/* packages. Lock package versions in package.json and requirements.txt to known-good pinned hashes using 'npm ci' with lockfile integrity enforcement (CIS 2.1 — Software Inventory; NIST CM-2 baseline configuration). Block automated pipeline promotion of any package version published after May 12, 2026 from the affected namespaces until integrity can be independently verified.

NIST Phase: Containment

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

Controls: NIST CM-2 (Baseline Configuration), NIST CM-8 (System Component Inventory), NIST SI-2 (Flaw Remediation), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Run 'npm ls --all 2>/dev/null | grep -E "@tanstack|@redhat-cloud-services|@bitwarden/cli|@opensearch-project/opensend|@mistralai|@uipath"' recursively across all repos to surface transitive hits. For Python, use 'pip-tree' or 'pipdeptree | grep -E "opensearch|mistral"'. Pin lockfile hashes immediately using 'npm ci --ignore-scripts' (the --ignore-scripts flag is critical to prevent malicious postinstall hooks from executing during the audit itself). For PyPI, add hash pinning via 'pip-compile --generate-hashes'. Use the free Phylum CLI (phylum.io) or Socket.dev CLI to scan package manifests for known-malicious package SHAs against their threat feeds at no cost.

Evidence: Before locking lockfiles, snapshot the current state: copy package-lock.json, yarn.lock, and requirements.txt with file hashes (sha256sum) to an isolated evidence directory. Capture 'npm ls --json > npm_tree_pre_containment.json' and diff against the last git-committed lockfile to identify which transitive @tanstack/*

or @redhat-cloud-services/* versions were silently upgraded post-May 12, 2026. Preserve the node_modules/.package-lock.json and the npm cache (~/.npm/_cacache) — the cached tarballs may contain the malicious payload and serve as artifact evidence. For PyPI, preserve ~/.cache/pip and the installed dist-info directories under site-packages for affected packages.

Step 2: Detection — Query CI/CD pipeline logs for unexpected outbound connections, credential access patterns, or new environment variable reads during build stages (NIST AU-6 — Audit Record Review; CIS 8.2 — Collect Audit Logs). Specifically look for: shell spawning from within Node.js build processes (T1059.004), access to cloud metadata endpoints (169.254.169.254 or equivalent) from build runners, unexpected npm publish events in pipeline logs, and any SLSA provenance attestation generated after a pipeline config change that lacks a corresponding reviewed pull request. Cross-reference pipeline execution timestamps against GitHub Actions workflow modification history. Alert on D3-SFA (System File Analysis) signals: modifications to CI config files (.github/workflows/*.yml, .circleci/config.yml) not tied to approved commits.

NIST Phase: Detection Analysis

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

Controls: NIST AU-2 (Event Logging), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-12 (Audit Record Generation), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs), MITRE ATT&CK T1059.004 (Command and Scripting Interpreter: Unix Shell), MITRE ATT&CK T1552.001 (Unsecured Credentials: Credentials in Files), MITRE ATT&CK T1195.002 (Supply Chain Compromise: Compromise Software Supply Chain)

Compensating: Without a SIEM, query GitHub Actions audit logs directly via the GitHub API: 'gh api /orgs/{org}/audit-log --paginate --jq ".[] | select(.action == \"workflows.prepared_workflow_job\")" | grep -E "(push|workflow_dispatch)"' filtering for workflow runs post-May 12, 2026. For build runner outbound connections, deploy a temporary Wireshark or tcpdump capture on the runner host filtering for '169.254.169.254' or unexpected external IPs during npm install: 'tcpdump -w build_traffic.pcap host 169.254.169.254 or not net 10.0.0.0/8'. Use this free Sigma rule category as a detection template — search for process trees where node.exe or npm spawns bash/sh/cmd as a child process, queryable via osquery: 'SELECT pid, parent, name, cmdline FROM processes WHERE name IN ("bash","sh","cmd.exe") AND parent IN (SELECT pid FROM processes WHERE name LIKE "%node%");'. Review all .github/workflows/*.yml git blame history for changes between May 1–12, 2026 using 'git log --all --diff-filter=M -- .github/workflows/*.yml'.

Evidence: Collect GitHub Actions workflow run logs (Settings > Actions > Workflow Runs) for all runs post-May 12, 2026 involving affected package namespaces — download full job logs as .zip archives before they age out (GitHub retains for 90 days). Capture the git history of .github/workflows/ and .circleci/config.yml via 'git log --follow -p .github/workflows/' to identify unsigned or unreviewed modifications. Extract SLSA provenance attestation JSON from affected package versions using 'npm view @tanstack/react-router@ dist.attestations' and verify the rekorURL transparency log entry timestamp against the pipeline run timestamp — a mismatch or missing log entry is a high-fidelity indicator of tampering. Preserve CloudTrail/GCP Audit Logs/Azure Activity Logs for the build runner's IAM identity for the 72-hour window around each suspect build.

Step 3: Eradication — Roll back to the last known-good lockfile state predating May 12, 2026 for all affected packages, or replace affected dependencies with verified alternatives where available. Rotate all secrets, API tokens, and credentials accessible to compromised or potentially compromised build runners, including AWS/GCP/Azure service account keys, Kubernetes service account tokens, and HashiCorp Vault tokens (D3-CRO — Credential Rotation; NIST IA-5 Authenticator Management). Revoke and reissue any code-signing certificates or npm publish tokens associated with affected pipelines. Re-examine all VS Code extensions and Docker Hub images built in potentially affected pipelines for embedded payload indicators.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IA-5 (Authenticator Management), NIST CM-2 (Baseline Configuration), NIST CM-7 (Least Functionality), NIST SI-2 (Flaw Remediation), NIST AC-6 (Least Privilege), CIS 5.3 (Disable Dormant Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Credential rotation without a PAM tool: use cloud-provider CLIs in sequence — AWS: 'aws iam list-access-keys --user-name ' then 'aws iam delete-access-key' and 'aws iam create-access-key'; GCP: 'gcloud iam service-accounts keys list' then 'gcloud iam service-accounts keys delete'; Kubernetes: 'kubectl get secrets -A | grep service-account' then delete and recreate affected tokens. For HashiCorp Vault, revoke all tokens issued to affected AppRoles: 'vault token revoke -accessor ' for each build runner's accessor. For npm publish tokens specifically, run 'npm token list' and revoke all automation tokens created before or during the suspect window via 'npm token revoke '. Scan Docker Hub images built from affected pipelines using the free Trivy scanner: 'trivy image : --scanners secret,vuln' to surface embedded credentials or malicious layers. For VS Code extensions (.vsix files), unzip and grep the extracted JS bundles: 'unzip -p extension.vsix | strings | grep -E "(curl|wget|base64|eval|exec|spawn)"'.

Evidence: Before rotating credentials, enumerate and document all active tokens and their last-use timestamps as evidence of potential exfiltration windows: 'aws iam get-access-key-last-used --access-key-id ' and equivalent GCP/Azure commands. Preserve Kubernetes audit logs from the API server for service account token usage during the May 12, 2026 window — specifically look for token requests to the TokenRequest API (audit.k8s.io Event kind, verb=create, resource=serviceaccounts/token) from non-standard workloads. Capture the full layer history of suspect Docker images ('docker history --no-trunc ') and extract each layer for static analysis before revoking pipeline access. Archive npm publish audit events from the npm registry for affected package namespaces via the npm audit log endpoint to establish which pipeline identity performed malicious publishes.

Step 4: Recovery — After rotating credentials and restoring clean lockfiles, re-run full dependency audits using 'npm audit' and an independent SCA tool against the restored state. Validate that SLSA provenance attestations for newly built artifacts trace to pipeline configurations that have been independently reviewed post-remediation — do not treat provenance alone as integrity proof given this campaign (NIST SI-7 — Software, Firmware, and Information Integrity; D3-FMBV — File Magic Byte Verification for artifact binary validation). Monitor build pipeline outputs for 30 days post-remediation for recurrence, with alerting on any new outbound network connections from build runners (D3-LAM — Local Account Monitoring for build service accounts).

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST SI-2 (Flaw Remediation), NIST CM-3 (Configuration Change Control), NIST CA-7 (Continuous Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: For independent SCA without enterprise tooling, run both 'npm audit --audit-level=high' and the free Socket.dev CLI ('npx socket scan .') against the restored lockfile — Socket specifically flags Shai-Hulud-style install script abuse and typosquatting patterns that 'npm audit' misses. To validate SLSA provenance is not Shai-Hulud-tainted, verify each new build's attestation against the Sigstore transparency log using: 'cosign verify-attestation --type slsaprovenance --certificate-oidc-issuer https://token.actions.githubusercontent.com' and confirm the workflow ref matches your reviewed pipeline config SHA exactly. Set up a free GitHub Actions workflow job that runs 'tcpdump -i any -w /tmp/build_net.pcap &' alongside npm install steps and uploads the pcap as an artifact — this provides ongoing network visibility for the 30-day watch period without a SIEM.

Evidence: Before declaring recovery complete, perform a binary diff between the newly built artifacts and the last known-good artifacts using 'sha256sum' or 'diffoscope' to confirm byte-level equivalence after the lockfile restore. Query the Sigstore Rekor transparency log directly for entries tied to your pipeline's OIDC identity during the compromise window: 'rekor-cli search --email ' to identify any attestations signed by your pipeline that you did not authorize. Preserve build runner network flow logs (VPC Flow Logs for AWS, VPC Flow Logs for GCP, NSG Flow Logs for Azure) for the full 30-day monitoring period as evidence of recurrence or residual C2 activity — specifically flag any outbound connections to non-CDN IPs on ports 443/80 from runner subnets that lack a corresponding package download justification.

Step 5: Post-Incident — Document the control gap: SLSA provenance attestation is a pipeline-integrity signal, not an artifact-integrity guarantee when the pipeline itself can be compromised. Update your software supply

chain security policy to require out-of-band build pipeline integrity verification, separation of duties between pipeline configuration and pipeline execution (NIST AC-5 — Separation of Duties), and mandatory human review of all CI/CD config changes before merge (CIS 4.6 — Securely Manage Enterprise Assets and Software). Evaluate adoption of SLSA Build Level 3 controls and Sigstore Cosign with transparency log verification as layered controls. Conduct a tabletop exercise simulating a credential-free pipeline injection scenario against your specific CI/CD stack.

NIST Phase: Post Incident

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

Controls: NIST AC-5 (Separation of Duties), NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SA-11 (Developer Testing and Evaluation), NIST SA-15 (Development Process, Standards, and Tools), NIST CM-3 (Configuration Change Control), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: For the tabletop exercise without a red team budget, use the publicly documented Shai-Hulud TTP pattern as your scenario inject: an attacker gains write access to a GitHub Actions workflow YAML via a dependency confusion or typosquatting vector, modifies it to exfiltrate ACTIONS_ID_TOKEN_REQUEST_TOKEN during a scheduled workflow run, and uses that OIDC token to generate a signed SLSA provenance attestation for a malicious package version — walk your team through detection, isolation, and rotation steps using only the free tools identified in prior steps. To implement separation of duties for pipeline config without enterprise tooling, enforce GitHub branch protection rules requiring at least one non-author code owner review on all changes to '.github/workflows/' using a CODEOWNERS file entry: '/.github/workflows/ @security-team'. Implement the free OpenSSF Scorecard GitHub Action ('ossf/scorecard-action') to continuously score your pipeline's supply chain hygiene against the SLSA and Sigstore criteria and surface regressions via PR comments.

Evidence: Compile the lessons-learned report with the following Shai-Hulud-specific artifacts attached: (1) the git diff of any .github/workflows/*.yml changes made during the compromise window with reviewer history showing the control gap, (2) the npm provenance attestation JSON for each malicious package version as evidence of the SLSA trust-signal abuse, (3) the timeline correlation between pipeline OIDC token issuance events and malicious npm publish timestamps from the registry audit log, and (4) the network pcap excerpts showing metadata endpoint access or exfiltration from build runners. These four artifact categories directly evidence the credential-free injection mechanic and SLSA weaponization — they are the primary inputs for the policy update and the tabletop scenario design.

Detection Guidance

Note: At time of analysis, no network IOCs (C2 domains, attacker IPs, malware hashes) are confirmed public. Detection must focus on behavioral signals and package metadata anomalies. Primary detection targets are CI/CD pipeline logs, npm registry publish logs, and network egress from build runners. Specific indicators and queries: (1) GitHub Actions: search workflow run logs for unexpected calls to 'curl', 'wget', 'nc', or base64-decoded payloads within build steps; query via GitHub Audit Log API filtering on 'workflow_run' events with unexpected external domains. (2) CircleCI: review job output logs for environment variable enumeration commands ('env', 'printenv') not present in approved workflow definitions. (3) npm registry: flag any package version published from @tanstack/*, @redhat-cloud-services/*, @bitwarden/cli, @opensearch-project/opensearch, @mistralai/mistralai, or @uipath/* after May 12, 2026 without a corresponding reviewed release PR; cross-reference npm publish timestamps against repository tag history. (4) Network: alert on outbound DNS or HTTP from build runner IPs to domains not in an approved allowlist, especially during non-interactive (automated) build windows; look for cloud metadata endpoint access (T1552.001). (5) SLSA provenance: treat any provenance attestation where the referenced workflow SHA does not match a human-reviewed commit in your VCS as an integrity warning - valid provenance from a tampered pipeline is the core attack mechanic here. (6) Endpoint/container: monitor for new scheduled tasks or cron entries created during or after build execution (T1053); alert on container escape indicators consistent with

T1611 if Kubernetes-based build runners are in scope. Behavioral IOCs to hunt: unusual npm publish activity from CI service accounts outside release windows; secrets or private key files read by build processes during non-signing stages (T1552.004); code obfuscation artifacts in published package dist directories (T1027).

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://www.stepsecurity.io/blog/mini-shai-hulud-is-back-a-self-spreading-supply-chain-attack-hits-the-npm-ecosystem`	StepSecurity analysis of Mini Shai-Hulud self-spreading npm supply chain attack — corroborating source for TanStack compromise and campaign lineage	MEDIUM
URL	`https://unit42.paloaltonetworks.com/monitoring-npm-supply-chain-attacks/`	Unit 42 npm supply chain attack monitoring report — corroborating source for campaign attribution and affected package scope	MEDIUM
URL	`https://unit42.paloaltonetworks.com/npm-supply-chain-attack/`	Unit 42 npm supply chain attack technical analysis — secondary corroborating source	MEDIUM
URL	`https://www.microsoft.com/en-us/security/blog/2025/12/09/shai-hulud-2-0-guidance-for-detecting-investigating-and-defending-against-the-supply-chain-attack/`	Microsoft Security Blog — Shai-Hulud 2.0 detection and defense guidance; listed as T1 source in feed data. Note: URL contains a 2025 date stamp — verify this resolves to current, relevant content before use as primary reference.	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1567** — Exfiltration Over Web Service
- **T1611** — Escape to Host
- **T1528** — Steal Application Access Token
- **T1059.004** — Unix Shell
- **T1485** — Data Destruction
- **T1553.002** — Code Signing
- **T1078** — Valid Accounts
- **T1552.001** — Credentials In Files
- **T1195.002** — Compromise Software Supply Chain
- **T1554** — Compromise Host Software Binary
- **T1176** — Software Extensions
- **T1059** — Command and Scripting Interpreter
- **T1027** — Obfuscated Files or Information

- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1072** — Software Deployment Tools
- **T1053** — Scheduled Task/Job
- **T1036.003** — Rename Legitimate Utilities
- **T1552.004** — Private Keys

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **AC-3** — Access Enforcement
- **SC-8** — Transmission Confidentiality and Integrity
- **SC-17** — Public Key Infrastructure Certificates
- **CM-3** — Configuration Change Control
- **AT-2** — Literacy Training and Awareness
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A02:2021** — Cryptographic Failures
- **A07:2021** — Identification and Authentication Failures
- **A08:2021** — Software and Data Integrity Failures
- **A01:2021** — Broken Access Control

CIS-V8

- **3.10** — Encrypt Sensitive Data in Transit
- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **6.1** — Establish an Access Granting Process
- **6.2** — Establish an Access Revoking Process
- **6.3** — Require MFA for Externally-Exposed Applications
- **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(a)(1)** — Access Control
- **164.312(d)** — Person or Entity Authentication
- **164.308(a)(5)(i)** — Security Awareness and Training

ISO-27001-2022

- **A.5.34** — Privacy and protection of personal information
- **A.5.21** — Managing information security in the ICT supply chain
- **A.5.23** — Information security for use of cloud services

NIST-CSF-2

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1567	Exfiltration Over Web Service	Exfiltration
T1611	Escape to Host	Privilege-Escalation
T1528	Steal Application Access Token	Credential-Access
T1059.004	Unix Shell	Execution
T1485	Data Destruction	Impact
T1553.002	Code Signing	Defense-Evasion
T1078	Valid Accounts	Defense-Evasion
T1552.001	Credentials In Files	Credential-Access
T1195.002	Compromise Software Supply Chain	Initial-Access
T1554	Compromise Host Software Binary	Persistence
T1176	Software Extensions	Persistence
T1059	Command and Scripting Interpreter	Execution
T1027	Obfuscated Files or Information	Defense-Evasion
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access

Technique ID	Technique Name	Tactic
T1072	Software Deployment Tools	Execution
T1053	Scheduled Task/Job	Execution
T1036.003	Rename Legitimate Utilities	Defense-Evasion
T1552.004	Private Keys	Credential-Access

Sources

Source	URL	Tier
Unit 42	https://unit42.paloaltonetworks.com/monitoring-npm-supply-chain-att...	T3
	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
A Self-Spreading Supply Chain Attack Compromises TanStack npm ...	https://www.stepsecurity.io/blog/mini-shai-hulud-is-back-a-self-spr...	T3

Generated 2026-06-03 06:51 UTC by TJS Security Command Center