INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-02 18:58 UTC

npm Supply Chain Crisis: Wormable Malware Commoditization and SLSA Provenance Bypass Threaten Enterprise Infrastructure

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0395
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	npm registry; @redhat-cloud-services (32 packages, compromised June 1 2026); @tanstack/* packages including react-router (~12.7M weekly downloads, May 11 2026 wave); @opensearch-project/opensearch; @uipath/* (57 packages); @mistralai/mistralai; @bitwarden/cli; GitHub Actions; CircleCI; AWS, GCP, Azure cloud environments; Kubernetes; HashiCorp Vault; Docker Hub; VS Code extensions; PyPI ecosystem
Published	2026-06-02T17:30:33+00:00
Discovery Source	Rss:T1 Threatintel

Executive Summary

Since September 2025, a sophisticated and escalating supply chain campaign has systematically compromised npm packages used by millions of enterprise development pipelines, reaching a critical point in May-June 2026. Attackers have compromised packages with a combined exposure exceeding 520 million downloads (provisional estimate pending Unit 42/Wiz corroboration), deployed wormable credential-stealing malware targeting CI/CD secrets, cloud provider credentials, and Kubernetes environments, and, most critically, demonstrated the ability to forge or obtain valid cryptographic provenance attestations (SLSA), undermining the integrity controls organizations rely on to verify software authenticity. The public release of the attack toolkit (Mini Shai-Hulud) on May 12, 2026 has lowered the barrier to entry for secondary threat actors, meaning organizations that have not audited their software supply chain face compounding risk from both the original campaign and copycat activity.

Technical Analysis

The Shai-Hulud campaign targets the npm ecosystem through compromised or malicious packages across multiple high-value namespaces. Key technical developments: (1) TanStack wave (May 11, 2026), packages including @tanstack/react-router (~12.7M weekly downloads) were weaponized, estimated cumulative exposure

of 520M downloads; (2) Mini Shai-Hulud source release (May 12, 2026), wormable malware source code made public, commoditizing attack capability; (3) Red Hat Cloud Services compromise (June 1, 2026), 32 packages under @redhat-cloud-services trojanized with valid or forged SLSA provenance attestations, directly bypassing cryptographic supply chain integrity controls. Additional affected namespaces include @opensearch-project/opensearch, @uipath/* (57 packages), @mistralai/mistralai, and @bitwarden/cli. The malware (Shai-Hulud family) exhibits wormable propagation and executes credential exfiltration targeting: GitHub Actions and CircleCI secrets (T1552.001, T1552.004), cloud provider credential stores for AWS, GCP, and Azure (T1078.004), Kubernetes service account tokens, HashiCorp Vault, and VS Code extension data (T1528). Persistence is achieved via GitHub Actions self-hosted runner registration using discussion.yaml (T1543, T1053). Shai-Hulud 2.0 includes a home-directory wipe fallback (T1485). The malware exfiltrates to code repositories (T1567.001) and uses JavaScript and Unix shell scripting (T1059.007, T1059.004). Obfuscation (T1027) is used throughout. No single CVE is assigned; relevant weaknesses: CWE-829 (inclusion from untrusted control sphere), CWE-494 (download without integrity check), CWE-426 (untrusted search path), CWE-312 (cleartext storage of sensitive information), CWE-732 (incorrect permission assignment). SLSA provenance bypass mechanism and exact download exposure figures (520M downloads, 32 @redhat-cloud-services packages) are provisional pending independent corroboration against Unit 42 and Wiz publications; treat as upper-bound estimates. Source quality score: 0.73. Microsoft Security Blog (T1 source) provides Shai-Hulud 2.0 detection and investigation guidance.

Action Checklist

1. Step 1: Containment. Immediately freeze npm dependency updates across all CI/CD pipelines. Lock package-lock.json and yarn.lock files to currently deployed, pre-compromise versions. Block installation or execution of packages from affected namespaces (@redhat-cloud-services, @tanstack/*, @uipath/*, @opensearch-project/opensearch, @mistralai/mistralai, @bitwarden/cli) until integrity is verified. Disable any self-hosted GitHub Actions runners registered after May 11, 2026, pending audit (T1543 Create or Modify System Process, T1053 Scheduled Task/Job). Reference: NIST CM-7 (Least Functionality), CIS 2.3 (Address Unauthorized Software).
2. Step 2: Detection. Audit all npm packages installed in CI/CD environments, container images, and developer workstations against known affected namespaces and version ranges. Cross-reference download exposure estimates (520M downloads, 32 @redhat-cloud-services packages) against latest Unit 42 and Wiz publications before finalizing scope of affected systems; provisional figures may underestimate or overestimate actual exposure. Query CI/CD secret access logs (GitHub Actions audit log, CircleCI audit events) for anomalous secret reads or exports since September 2025. Search for discussion.yaml or equivalent runner registration workflow files added to repositories without change control approval. Check for outbound connections to unexpected code repositories from build systems (T1567.001). Review AWS CloudTrail, GCP Audit Logs, and Azure Activity Logs for credential use from build pipeline service accounts in unusual regions or at unusual times (T1078.004). Inspect VS Code extension directories for unauthorized modifications (T1176). Run file integrity monitoring alerts against package contents using hash verification, do not rely solely on SLSA attestation given confirmed bypass (NIST SI-7, CIS 8.2, D3-SFA). Reference Microsoft Security Blog guidance for Shai-Hulud 2.0 specific detection signatures.
3. Step 3: Eradication. Replace all packages from affected namespaces with versions verified against known-good hashes from before September 2025, or with confirmed clean alternatives. Rotate all secrets exposed to compromised pipeline environments: GitHub Actions secrets, CircleCI environment variables, AWS IAM keys, GCP service account keys, Azure service principals, HashiCorp Vault tokens, and

Kubernetes service account tokens (NIST IA-5, D3-CRO). Remove any unauthorized GitHub Actions self-hosted runner registrations and audit all discussion.yaml or workflow files for injected steps. Revoke and reissue any npm publish tokens associated with affected scopes. Do not trust SLSA attestations alone for @redhat-cloud-services packages until Red Hat confirms clean re-releases, require independent hash verification (NIST SI-7, CIS 7.3, CIS 7.4).

4. Step 4: Recovery. Rebuild CI/CD pipeline environments from known-clean base images, not from cached or artifact-store copies that may have been built during the exposure window. Validate all rebuilt pipeline outputs against pre-compromise dependency manifests. Re-enable dependency updates only after implementing npm provenance verification with independent hash pinning (not SLSA alone). Monitor cloud provider credential usage for 30 days post-rotation for signs of pre-rotation credential reuse (NIST AU-6, CIS 8.2). Verify HashiCorp Vault audit logs for any unauthorized secret reads during the exposure window. Confirm GitHub Actions runner inventory matches approved registrations before re-enabling self-hosted runner workflows.

5. Step 5: Post-Incident. Implement Software Composition Analysis (SCA) tooling with hash pinning and behavioral analysis in CI/CD pipelines, do not treat attestation alone as sufficient given demonstrated SLSA bypass (NIST SA-12, CIS 2.1, CIS 2.3). Establish a dependency approval workflow requiring human review before any new or updated npm package enters production pipelines (NIST CM-3, AC-6). Given the Mini Shai-Hulud public source release, treat this threat class as persistent and expanding, update threat models to include secondary actors replicating the campaign against internal package registries and private npm mirrors. Review and test incident response playbooks for supply chain compromise scenarios. Consider adopting a private npm registry with internal vetting as a long-term control (D3-PBWSAM, NIST SC-28).

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and external IR retainer immediately if CloudTrail, GCP Audit Logs, or Azure Activity Logs show any API calls using CI/CD service account credentials from IPs outside known build infrastructure after September 1, 2025, or if any secrets with access to PII, PHI, cardholder data, or regulated cloud storage were readable by compromised pipeline environments during the exposure window — both conditions trigger breach notification assessment obligations under GDPR, HIPAA, and PCI DSS.
Recovery Notes	Rebuild all CI/CD environments from base images predating September 2025 and do not restore from any cached pipeline artifacts, container image layers, or artifact store snapshots built during the exposure window, as Shai-Hulud 2.0 embeds persistence in build outputs, not just in the pipeline environment itself. Monitor all newly rotated cloud provider credentials (AWS IAM, GCP service accounts, Azure service principals) for a minimum of 30 days for API calls inconsistent with normal build patterns — specifically, any GetSecretValue, AssumeRole, or equivalent calls from non-build-system IPs indicate pre-rotation credential capture and active threat actor reuse requiring immediate re-escalation. Given the confirmed public release of Mini Shai-Hulud source code, treat secondary campaign actors targeting internal npm mirrors and private package registries as a near-term certainty and validate private registry integrity with independent hash verification before each production deployment for at least 90 days post-incident.

Forensic Artifacts	npm install execution logs with timestamps from CI/CD build agents (GitHub Actions runner logs, CircleCI job output) — cross-reference install timestamps for packages in the seven affected namespaces against outbound network connections from the build host during the same 60-second window to identify Shai-Hulud 2.0 postinstall hook C2 exfiltration events | GitHub Actions audit log entries for `runners.created`, `secrets.access`, and `workflows.created` events since September 1, 2025 — the discussion.yaml runner registration technique produces a runners.created event from an unexpected actor identity outside the normal CI/CD service account, which is the primary forensic indicator of the runner hijacking component | HashiCorp Vault audit log (`/var/log/vault/audit.log`) showing all `secret/data/*` read operations by CI/CD AppRole tokens during the exposure window — Shai-Hulud 2.0 enumerates and exfiltrates Vault secrets during pipeline execution, producing a burst-read pattern of multiple distinct secret paths within seconds that is forensically distinguishable from normal single-secret application reads | Container image layer history (`docker history --no-trunc`) for all images built during the exposure window — the wormable malware's postinstall hook execution appears as an anomalous shell command layer invoked during the `npm install` build step, and the layer's filesystem diff will contain the malware's dropped artifacts if the hook wrote files to the image filesystem | SLSA attestation files for @redhat-cloud-services packages alongside independent SHA-256 hash verification results computed against the actual downloaded tarballs — discrepancies between attested and actual hashes constitute direct forensic evidence of the SLSA provenance bypass mechanism and should be preserved for vendor disclosure, regulatory submissions, and potential legal proceedings

Per-Action IR Details

Step 1: Containment — Immediately freeze npm dependency updates across all CI/CD pipelines. Lock package-lock.json and yarn.lock files to currently deployed, pre-compromise versions. Block installation or execution of packages from affected namespaces (@redhat-cloud-services, @tanstack/*, @uipath/*, @opensearch-project/opensearch, @mistralai/mistralai, @bitwarden/cli) until integrity is verified. Disable any self-hosted GitHub Actions runners registered after May 11, 2026, pending audit (T1543, T1053). Reference: NIST CM-7 (Least Functionality), CIS 2.3 (Address Unauthorized Software).

NIST Phase: Containment

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

Controls: NIST CM-7 (Least Functionality), NIST CM-2 (Baseline Configuration), NIST IR-4 (Incident Handling), CIS 2.3 (Address Unauthorized Software), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Use a two-step manual freeze: (1) Run `npm config set ignore-scripts true` on all build agents to block malicious postinstall hooks used by Shai-Hulud 2.0 variants. (2) Execute `git diff HEAD package-lock.json yarn.lock` across all active repositories to surface any lock file drift since May 11, 2026 — pipe output to a text file for evidence preservation. For GitHub Actions runner disablement without enterprise admin tooling, use the GitHub CLI: `gh api /repos/{owner}/{repo}/actions/runners --paginate | jq '.runners[] | select(.created_at > "2026-05-11")' ` to enumerate runners registered after the campaign start date, then disable each via `gh api -X DELETE /repos/{owner}/{repo}/actions/runners/{id}`. Two-person team split: one person owns lock file audit across repos, one owns runner enumeration.

Evidence: Before executing the freeze, snapshot current pipeline state as forensic baseline: (1) Export the full dependency tree from each affected build environment using `npm list --all --json > npm_tree_$(hostname)_$(date +%Y%m%d).json` — this captures the installed graph including transitive dependencies that may include compromised @redhat-cloud-services or @tanstack/* versions pulled in indirectly. (2) Preserve existing package-lock.json and yarn.lock files verbatim with SHA-256 hashes (`sha256sum package-lock.json`) before any freeze modification — these are your baseline evidence of what was actually installed during the exposure window (September 2025 through June 2026). (3) Capture GitHub Actions runner registration timestamps from the audit log export: Settings → Actions → Runners → export or query via API before disabling, preserving runner name, registration date, and associated

workflow files for later attribution.

Step 2: Detection — Audit all npm packages installed in CI/CD environments, container images, and developer workstations against known affected namespaces and version ranges. Query CI/CD secret access logs (GitHub Actions audit log, CircleCI audit events) for anomalous secret reads or exports since September 2025. Search for discussion.yaml or equivalent runner registration workflow files added to repositories without change control approval. Check for outbound connections to unexpected code repositories from build systems (T1567.001). Review AWS CloudTrail, GCP Audit Logs, and Azure Activity Logs for credential use from build pipeline service accounts in unusual regions or at unusual times (T1078.004). Inspect VS Code extension directories for unauthorized modifications (T1176). Run file integrity monitoring alerts against package contents using hash verification — do not rely solely on SLSA attestation given confirmed bypass (NIST SI-7, CIS 8.2, D3-SFA). Reference Microsoft Security Blog guidance for Shai-Hulud 2.0 specific detection signatures.

NIST Phase: Detection Analysis

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

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST AU-2 (Event Logging), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AC-2 (Account Management), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: For teams without SIEM: (1) Query GitHub Actions audit log via CLI — `gh api /orgs/{org}/audit-log --paginate -f phrase='action:secrets.access' | jq 'select(.@timestamp > "2025-09-01T00:00:00Z")' > secrets_audit.json` — review for any secret reads by workflow runs that installed packages from the seven affected namespaces. (2) For AWS CloudTrail without a SIEM, use the free AWS CLI one-liner: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=AssumeRole --start-time 2025-09-01 --query 'Events[?Username!=\`expected-ci-role\`]' > cloudtrail_suspicious.json` — flag any IAM role assumptions by the CI/CD service account from non-build-system IPs. (3) For package hash verification without SCA tooling, use `shasum -a 256 node_modules/@redhat-cloud-services/*/package.json` and compare against npm registry published hashes via `npm view @redhat-cloud-services/{package}@{version} dist.integrity`. (4) For VS Code extension audit on Linux/macOS: `find ~/.vscode/extensions -name '*.js' -newer /tmp/baseline_date -ls` where baseline_date is a file touched on May 10, 2026.

Evidence: Capture before any remediation or pipeline changes: (1) GitHub Actions audit log export covering September 1, 2025 through present — specifically filter for `action:secrets.access`, `action:workflows.created`, and `action:runners.created` events; the discussion.yaml runner registration technique produces a `runners.created` event outside normal workflow file paths. (2) For each container image built during the exposure window, extract the layer history (`docker history --no-trunc {image_id}`) and cross-reference `npm install` invocations with the affected namespace list — the wormable malware's postinstall hook execution will appear as a shell command layer in the image history. (3) Cloud provider credential usage: export AWS CloudTrail `GetSecretValue` and `AssumeRole` events, GCP `google.iam.credentials.v1.GenerateAccessToken` events, and Azure `microsoft.keyvault/vaults/secrets/read` operations for all CI/CD service accounts from September 2025 forward — Shai-Hulud 2.0 exfiltrates these via postinstall hooks and the call pattern (burst read of multiple secret names within seconds of an npm install) is forensically distinctive. (4) Network connection logs from build agents showing outbound HTTPS to non-registry endpoints initiated within the npm install execution window — the malware's C2 exfiltration occurs during package installation, so correlation of npm install timestamps with outbound connections to non-registry IPs is a high-confidence indicator.

Step 3: Eradication — Replace all packages from affected namespaces with versions verified against known-good hashes from before September 2025, or with confirmed clean alternatives. Rotate all secrets exposed to compromised pipeline environments: GitHub Actions secrets, CircleCI environment variables, AWS IAM keys, GCP service account keys, Azure service principals, HashiCorp Vault tokens, and Kubernetes service account tokens (NIST IA-5, D3-CRO). Remove any unauthorized GitHub Actions self-hosted runner registrations and audit all discussion.yaml or workflow files for injected steps. Revoke and reissue any npm

publish tokens associated with affected scopes. Do not trust SLSA attestations alone for @redhat-cloud-services packages until Red Hat confirms clean re-releases — require independent hash verification (NIST SI-7, CIS 7.3, CIS 7.4).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IA-5 (Authenticator Management), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-3 (Configuration Change Control), NIST AC-6 (Least Privilege), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 5.3 (Disable Dormant Accounts)

Compensating: Secret rotation without a secrets management platform: (1) AWS IAM — `aws iam list-access-keys --user-name {ci-user}` to enumerate, then `aws iam create-access-key` and `aws iam delete-access-key --access-key-id {old-key}` — complete rotation in under 5 minutes per account. (2) GitHub Actions secrets — use `gh secret set {SECRET_NAME} < new_value.txt` for each repository; enumerate all secrets requiring rotation with `gh secret list --repo {org}/{repo}`. (3) Kubernetes service account tokens — `kubectl delete secret {sa-token-secret} -n {namespace}` forces regeneration; enumerate all SA tokens in CI/CD namespaces with `kubectl get secrets --all-namespaces -o json | jq '.items[] | select(.type=="kubernetes.io/service-account-token")'`. (4) For workflow file auditing without enterprise tooling, run `git log --all --diff-filter=A --name-only --pretty=format: -- '.github/workflows/*.yaml' '.github/workflows/*.yml'` across all repositories to find newly added workflow files, then `grep -r 'runs-on: self-hosted' .github/workflows/` to identify which ones registered self-hosted runners.

Evidence: Before rotating any credential, document its current state and usage history as forensic evidence of compromise scope: (1) For each AWS IAM key associated with CI/CD pipelines, export the last-used data (`aws iam get-access-key-last-used --access-key-id {key}`) and the full CloudTrail history of API calls made with that key since September 2025 — this establishes whether the Shai-Hulud 2.0 exfiltration actually resulted in the key being operationally abused by threat actors post-theft. (2) HashiCorp Vault audit log — query for all `secret/data/*` read operations by the CI/CD AppRole or token during the exposure window; Vault's audit log at `/var/log/vault/audit.log` contains operation type, path, accessor, and timestamp — preserve this before any token revocation as it cannot be reconstructed. (3) For each workflow file identified as suspicious, capture `git log --follow -p -- .github/workflows/{filename}` to preserve the full commit history including who introduced the discussion.yaml runner registration technique and from which account — this is your attribution evidence.

Step 4: Recovery — Rebuild CI/CD pipeline environments from known-clean base images, not from cached or artifact-store copies that may have been built during the exposure window. Validate all rebuilt pipeline outputs against pre-compromise dependency manifests. Re-enable dependency updates only after implementing npm provenance verification with independent hash pinning (not SLSA alone). Monitor cloud provider credential usage for 30 days post-rotation for signs of pre-rotation credential reuse (NIST AU-6, CIS 8.2). Verify HashiCorp Vault audit logs for any unauthorized secret reads during the exposure window. Confirm GitHub Actions runner inventory matches approved registrations before re-enabling self-hosted runner workflows.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST CP-9 (System Backup), NIST CM-2 (Baseline Configuration), NIST SI-7 (Software, Firmware, and Information Integrity), CIS 8.2 (Collect Audit Logs), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: For teams rebuilding without enterprise artifact management: (1) Establish a known-clean baseline by pinning all npm dependencies to pre-September 2025 registry snapshots — use `npm pack {package}@{clean-version}` to download tarballs from npm registry and compute `sha256sum` on each, storing the manifest in a version-controlled file that becomes your internal truth source, replacing implicit trust in SLSA attestation. (2) For 30-day post-rotation monitoring of cloud credentials without a SIEM, schedule a daily cron job on any always-on Linux instance: `aws cloudtrail lookup-events --start-time $(date -d '24 hours ago' --iso-8601=seconds) --lookup-attributes AttributeKey=Username,AttributeValue={ci-service-account} | jq '.Events[] | select(.CloudTrailEvent |

fromjson | .sourceIPAddress != "{expected_build_ip}")` — alert on any off-pattern calls. (3) For GitHub Actions runner re-enablement gating, maintain a plaintext approved-runners.txt file with runner IDs and registration dates under version control, and diff against live inventory using the GitHub API before each pipeline re-enablement.

Evidence: Before declaring recovery complete, verify the absence of persistence mechanisms that would survive a pipeline rebuild: (1) Inspect Docker Hub or internal container registry for images tagged and pushed during the exposure window (September 2025 – June 2026) that were built on compromised pipelines — these images may contain embedded Shai-Hulud 2.0 artifacts in image layers and should be pulled, scanned with `docker scan` or Trivy, and quarantined if built during the window. (2) Verify that npm publish tokens used by the organization's own packages were not captured and used by attackers to inject malicious versions of your internal packages into npm registry — query `npm access ls-packages {your-org}` and cross-reference with npm registry publish history for any versions published during the exposure window that were not initiated by your team. (3) For Kubernetes environments, enumerate all cluster roles and bindings created or modified since September 2025 using `kubectl get clusterrolebindings -o json | jq '.items[] | select(.metadata.creationTimestamp > "2025-09-01T00:00:00Z")'` — the wormable component targets Kubernetes secrets and may have created persistent access via service account privilege escalation.

Step 5: Post-Incident — Implement Software Composition Analysis (SCA) tooling with hash pinning and behavioral analysis in CI/CD pipelines — do not treat attestation alone as sufficient given demonstrated SLSA bypass (NIST SA-12, CIS 2.1, CIS 2.3). Establish a dependency approval workflow requiring human review before any new or updated npm package enters production pipelines (NIST CM-3, AC-6). Given the Mini Shai-Hulud public source release, treat this threat class as persistent and expanding — update threat models to include secondary actors replicating the campaign against internal package registries and private npm mirrors. Review and test incident response playbooks for supply chain compromise scenarios. Consider adopting a private npm registry with internal vetting as a long-term control (D3-PBWSAM, NIST SC-28).

NIST Phase: Post Incident

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

Controls: NIST SA-12 (Supply Chain Protection), NIST CM-3 (Configuration Change Control), NIST AC-6 (Least Privilege), NIST SC-28 (Protection of Information at Rest), NIST RA-3 (Risk Assessment), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 2.3 (Address Unauthorized Software), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: For teams without budget for commercial SCA tools: (1) Deploy Dependency-Track (free, OWASP-hosted) as a self-hosted SCA platform — it ingests CycloneDX SBOMs generated by `cyclonedx-npm --output-file sbom.json` in your CI pipeline and flags known-malicious packages against OSS Index and NVD without per-seat licensing costs. (2) For behavioral analysis of npm packages before installation, use `npm-audit-resolver` combined with manual review of `postinstall`, `preinstall`, and `install` script fields in each package.json — the Shai-Hulud 2.0 family executes credential theft in these lifecycle hooks, so blocking `ignore-scripts=false` by default and requiring explicit human approval to enable scripts for any dependency is a zero-cost behavioral control. (3) Stand up a private Verdaccio npm registry (free, open source) as a package mirror with an internal allowlist — all CI/CD pipelines point to Verdaccio, which proxies only approved packages, eliminating direct exposure to compromised upstream namespaces. (4) For threat model updates reflecting the Mini Shai-Hulud public release, document specific detection rules as Sigma format (free) targeting postinstall hook process spawning from node.exe with outbound network connections — submit to the SigmaHQ community repository for peer review.

Evidence: For the post-incident lessons learned review, the following evidence should be compiled and preserved for the retrospective and for potential regulatory notification assessment: (1) Complete timeline of first npm install of any affected package version to date of detection, constructed from CI/CD build logs and npm install timestamps — this establishes the organization's specific exposure window, which may differ materially from the campaign's September 2025 start date depending on which packages and versions were actually used. (2) Inventory of all secrets that were readable by compromised pipeline environments during the exposure window, cross-referenced with any evidence of those secrets being used post-compromise — this is the key input for breach notification assessment if the secrets provided access to PII, PHI, or regulated data. (3) Documentation of the SLSA attestation bypass mechanism as observed in your environment — preserve any SLSA attestation files for @redhat-cloud-services packages installed

during the compromise period alongside the independent hash verification results that contradict them, as this constitutes technical evidence of the bypass that should be shared with Red Hat's security team and relevant industry working groups.

Detection Guidance

Primary detection targets are CI/CD environments, developer workstations, and cloud credential stores. Key indicators and log sources: (1) npm audit / dependency manifest review, cross-reference installed package versions in node_modules and lock files against affected namespaces and version ranges documented by Unit 42 and Wiz; flag any package where the installed hash does not match the pre-May 2026 known-good hash. (2) GitHub Actions audit log, search for 'self_hosted_runner.register' events and any workflow runs referencing discussion.yaml or equivalent runner registration files added after May 10, 2026; flag secret access events (actions.secret_access) from pipelines touching affected packages. (3) CircleCI audit events, review for environment variable reads from contexts associated with cloud credentials during build jobs that consumed affected packages. (4) Cloud provider logs, AWS CloudTrail: filter on 'GetSecretValue', 'AssumeRole', and IAM key usage from CI/CD IP ranges; GCP Audit Logs: filter on 'secretmanager.versions.access' and service account key creation; Azure Activity Logs: filter on Key Vault secret reads and service principal authentication from build system IPs, flag any activity from geographic regions inconsistent with your pipeline infrastructure. (5) Kubernetes, audit logs for service account token requests and any pod exec or API server calls from nodes running compromised build images. (6) HashiCorp Vault, audit log review for secret reads by build-system AppRoles during the September 2025 to present window. (7) VS Code, inspect extension directories for files modified after package installation from affected namespaces; use D3-SFA (System File Analysis) tooling. (8) Network egress, look for outbound HTTPS to github.com repository endpoints (raw.githubusercontent.com, api.github.com) from build systems outside of expected source control operations, consistent with T1567.001 exfiltration pattern. (9) Hash verification, do not rely on SLSA attestation for @redhat-cloud-services packages; perform independent SHA-256 hash verification of installed package tarballs against pre-June 2026 known-good values. (10) Behavioral indicator: any npm postinstall script executing shell commands, spawning child processes, or making outbound network calls is high-confidence suspicious in this campaign context (T1059.007, T1059.004, T1027).

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`raw.githubusercontent.com`	Exfiltration target — Shai-Hulud uses GitHub repository endpoints for credential exfiltration (T1567.001); anomalous POST or PUT traffic from build systems to this domain is a behavioral indicator. Note: legitimate use is common; context and volume matter.	MEDIUM

Type	Value	Context	Confidence
DOMAIN	`api.github.com`	Secondary exfiltration and C2 coordination endpoint used by Shai-Hulud runner registration technique (T1543, T1053); outbound calls from CI/CD nodes outside of expected source control operations are suspicious.	MEDIUM
URL	`https://www.wiz.io/blog/shai-hulud-npm-supply-chain-attack`	Wiz technical analysis — contains package-specific IOCs and hash values; consult for current IOC list as this field cannot reproduce the full indicator set without risk of error.	HIGH
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 T1 source — Shai-Hulud 2.0 detection signatures, investigation playbook, and defensive guidance. Treat as authoritative for behavioral IOCs.	HIGH
URL	`https://unit42.paloaltonetworks.com/monitoring-npm-supply-chain-attacks/`	Unit 42 ongoing monitoring post — use for current package version IOCs and updated campaign tracking; specific hash and version IOCs should be sourced directly from this page.	HIGH

Framework Mappings

MITRE-ATTACK

- **T1543** — Create or Modify System Process
- **T1567** — Exfiltration Over Web Service
- **T1554** — Compromise Host Software Binary
- **T1552.004** — Private Keys
- **T1078.004** — Cloud Accounts
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1059.006** — Python
- **T1485** — Data Destruction
- **T1059.004** — Unix Shell
- **T1552.001** — Credentials In Files
- **T1053** — Scheduled Task/Job
- **T1059.007** — JavaScript
- **T1027** — Obfuscated Files or Information
- **T1072** — Software Deployment Tools
- **T1567.001** — Exfiltration to Code Repository
- **T1176** — Software Extensions

- **T1528** — Steal Application Access Token

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **AC-3** — Access Enforcement
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **AC-6** — Least Privilege
- **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

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **3.3** — Configure Data Access Control Lists
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers

HIPAA-SECURITY

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

SOC2-TSC

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

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
- **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
T1543	Create or Modify System Process	Persistence
T1567	Exfiltration Over Web Service	Exfiltration
T1554	Compromise Host Software Binary	Persistence
T1552.004	Private Keys	Credential-Access
T1078.004	Cloud Accounts	Defense-Evasion
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1059.006	Python	Execution
T1485	Data Destruction	Impact
T1059.004	Unix Shell	Execution
T1552.001	Credentials In Files	Credential-Access
T1053	Scheduled Task/Job	Execution
T1059.007	JavaScript	Execution
T1027	Obfuscated Files or Information	Defense-Evasion
T1072	Software Deployment Tools	Execution
T1567.001	Exfiltration to Code Repository	Exfiltration
T1176	Software Extensions	Persistence
T1528	Steal Application Access Token	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
Shai-Hulud npm Supply Chain Attack | Wiz Blog	https://www.wiz.io/blog/shai-hulud-npm-supply-chain-attack	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 2026-06-02 18:58 UTC by TJS Security Command Center