INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-19 06:43 UTC

TeamPCP's Mini Shai-Hulud Worm Poisons 323 npm Packages, Forges SLSA Provenance, and Goes Open Source on BreachForums

THREAT CAMPAIGN | CRITICAL | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0333
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	npm packages: @antv/g2, @antv/g6, @antv/x6, @antv/l7, @antv/s2, @antv/f2, @antv/g, @antv/g2plot, @antv/graphin, @antv/data-set, echarts-for-react (~1.1M weekly downloads), timeago.js, size-sensor, canvas-nest.js; 323 total packages (639 malicious versions); downstream exposure to GitHub Actions, AWS, Google Cloud, Microsoft Azure, Docker, Kubernetes, HashiCorp Vault, Stripe, TanStack, Mistral AI, Guardrails AI
Published	2026-05-19T00:54:17
Discovery Source	Rss

Executive Summary

A threat actor called TeamPCP hijacked a trusted npm package maintainer account and injected malicious code into 639 versions of 323 packages, including widely used data visualization libraries with millions of weekly downloads. The malware self-replicates across the npm ecosystem, steals credentials from cloud platforms, CI/CD pipelines, and payment systems, and forges software supply chain verification signatures, meaning infected packages appeared legitimate even to teams using provenance checks. TeamPCP then published the full attack toolkit on a criminal forum, enabling copycat attackers to repeat the campaign against organizations with JavaScript build pipelines.

Technical Analysis

TeamPCP compromised the 'atool' npm maintainer account and used it to publish 639 malicious versions across 323 packages. Primary targets include the @antv visualization ecosystem (@antv/g2, @antv/g6, @antv/x6, @antv/l7, @antv/s2, @antv/f2, @antv/g, @antv/g2plot, @antv/graphin, @antv/data-set), echarts-for-react (~1.1M weekly downloads), timeago.js, size-sensor, and canvas-nest.js. The worm is self-replicating (CWE-506) and deploys a credential stealer (CWE-522) targeting 20+ credential types: AWS, GCP, and Azure secrets; GitHub Actions tokens; Docker and Kubernetes credentials; HashiCorp Vault tokens;

and Stripe payment keys. The campaign's most significant technical escalation is SLSA provenance forgery: the worm abuses stolen OIDC tokens to generate counterfeit supply chain attestations (CWE-346, CWE-295), defeating provenance verification controls organizations rely on to confirm package integrity. Obfuscation is present. The threat actor subsequently open-sourced the worm on BreachForums (T1650), dramatically lowering the barrier for copycat campaigns. MITRE techniques include T1195.001 (Supply Chain Compromise: Compromise Software Dependencies), T1552/T1552.001 (Unsecured Credentials: Credentials in Files), T1528 (Steal Application Access Token), T1553 (Subvert Trust Controls), T1611 (Escape to Host), T1587.001 (Develop Capabilities: Malware), and T1059 (Command and Scripting Interpreter). No CVE is assigned; no vendor patch is available. Remediation requires auditing installed package versions against the malicious version list and rotating all credentials accessible from affected build environments.

Action Checklist

1. Step 1: Containment, Immediately audit all package.json, package-lock.json, and yarn.lock files across every build environment and CI/CD pipeline for any of the 323 affected packages. Cross-reference installed versions against the malicious version list published by StepSecurity (https://www.stepsecurity.io/blog/shai-hulud-here-we-go-again-mass-npm-supply-chain-attack-hits-the-antv-ecosystem). [VERIFY THIS URL BEFORE PUBLICATION.] Isolate any build runner, container, or pipeline that installed a flagged version from production networks and secrets stores pending credential rotation.
2. Step 2: Detection, Query CI/CD pipeline logs, container runtime logs, and endpoint telemetry for outbound connections from build environments to unexpected external IPs or domains. Hunt for process execution anomalies in npm install or postinstall hooks (T1059). Check for OIDC token issuance events in GitHub Actions audit logs that do not correspond to known workflows; stolen OIDC tokens were used to forge SLSA attestations (T1553, T1528). Search SIEM for credential access patterns matching T1552.001: file reads of ~/.aws/credentials, environment variable enumeration, and Vault token reads from build agents. Monitor StepSecurity and CISA for updated IOC releases; initial hash and C2 indicators may not be available for 24-48 hours post-disclosure.
3. Step 3: Eradication, Pin all @antv packages and other affected dependencies to known-clean versions predating the compromise window. Remove malicious versions from local npm caches and private registries. Do not rely solely on SLSA provenance attestations to verify cleanliness; the worm forges these. Verify package integrity using independent hash comparison against pre-compromise release hashes where available. Re-evaluate any pipeline that grants OIDC token access to npm publish workflows.
4. Step 4: Recovery, Rotate all credentials accessible from any build environment that ran a flagged package version. This includes AWS IAM keys, GCP service account credentials, Azure service principals, GitHub Actions secrets, Docker registry credentials, Kubernetes service account tokens, HashiCorp Vault tokens, and Stripe API keys. Redeploy affected services from clean build artifacts only. Audit cloud provider access logs (AWS CloudTrail, GCP Audit Logs, Azure Activity Log) for anomalous API calls in the window following any flagged install event. Validate that no new IAM roles, API keys, or access policies were created by unauthorized actors.
5. Step 5: Post-Incident, This campaign exposed a critical gap: SLSA provenance verification alone is insufficient when OIDC token theft enables attestation forgery. Implement additional controls: enforced dependency pinning with hash verification (not just provenance), private registry mirroring with pre-ingestion scanning, and least-privilege OIDC token scoping in GitHub Actions. Treat worm source code availability on BreachForums as a persistent threat indicator: assess your npm dependency surface for exposure to copycat campaigns using the same techniques. Review cloud access logs if SAP assets

are in your JavaScript build pipeline scope.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO and legal counsel immediately if AWS CloudTrail, GCP Audit Logs, or Azure Activity Log confirm unauthorized API calls using credentials stolen from build environments, if Stripe API keys show unauthorized charge activity, if Kubernetes service account tokens were used to access production namespaces, or if any system processing PII or PHI was built using a flagged package version — triggering breach notification obligations under GDPR, CCPA, or HIPAA depending on jurisdiction.
Recovery Notes	Redeploy all services from build artifacts produced exclusively by pipelines verified clean post-eradication; do not reuse any artifact built during the TeamPCP compromise window regardless of SLSA attestation status, as attestations are untrustworthy for the affected period. Monitor AWS CloudTrail, GCP Audit Logs, Azure Activity Log, and GitHub audit logs continuously for 30 days post-rotation for re-use of rotated credentials or creation of new unauthorized IAM entities — TeamPCP or copycat actors with the BreachForums-published worm may attempt persistence using credentials exfiltrated before rotation. Treat any new npm package version for the 323 affected packages as suspect until the npm security team confirms the maintainer accounts have been fully recovered and re-secured.
Forensic Artifacts	npm cache tarballs at ~/.npm/_cacache/ (Linux/macOS) or %AppData%\npm-cache (Windows) containing the malicious @antv and downstream package versions with injected postinstall hook code — primary evidence of the Mini Shai-Hulud worm's injection mechanism and self-replication logic | GitHub Actions audit log OIDC token issuance records — specifically actions.create_workflow_run and oidc.create_registration events from the compromise window that document TeamPCP's SLSA attestation forgery via stolen OIDC tokens (T1528, T1553) | Build agent process execution logs showing node/npm child processes spawned during postinstall hook execution — on Linux runners via auditd records for execve syscalls with parent process npm, on Windows runners via Event ID 4688 (Process Creation) with ParentProcessName containing npm or node | Cloud provider credential access logs: AWS CloudTrail GetSecretValue and AssumeRoleWithWebIdentity events, GCP cloudaudit.googleapis.com/data_access for storage and Secret Manager reads, HashiCorp Vault audit log secret read operations — all filtered to build agent source IPs during the npm install execution window, documenting which secrets the worm's credential harvesting payload successfully accessed | node_modules/@antv/*/package.json and corresponding dist/index.js files from any build environment that installed flagged versions — the worm modifies these files in place and the diff between the malicious installed version and the known-clean pre-compromise release hash constitutes direct evidence of the injection

Per-Action IR Details

Step 1: Containment — Immediately audit all package.json, package-lock.json, and yarn.lock files across every build environment and CI/CD pipeline for any of the 323 affected packages. Cross-reference installed versions against the malicious version list published by StepSecurity (https://www.stepsecurity.io/blog/shai-hulud-here-we-go-again-mass-npm-supply-chain-attack-hits-the-antv-ecosystem). Isolate any build runner, container, or pipeline that installed a flagged version from production networks and secrets stores pending credential rotation.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: isolate affected systems to prevent further credential exfiltration and worm self-replication across the npm ecosystem

Controls: NIST IR-4 (Incident Handling), NIST CM-7 (Least Functionality) — restrict build runners from outbound internet access during investigation, NIST SI-7 (Software, Firmware, and Information Integrity) — verify package integrity against pre-compromise hashes, CIS 2.1 (Establish and Maintain a Software Inventory) — authoritative package inventory is prerequisite for rapid cross-referencing, CIS 2.3 (Address Unauthorized Software) — flagged malicious package versions constitute unauthorized software requiring immediate removal from build environments

Compensating: Run this one-liner across all repos to enumerate @antv and other affected packages: `find . -name 'package-lock.json' | xargs grep -l '@antv\|echarts-for-react\|timeago.js\|size-sensor\|canvas-nest' 2>/dev/null`. Then extract installed versions with `jq '.packages | to_entries[] | select(.key | test("@antv|echarts-for-react|timeago.js|size-sensor|canvas-nest")) | {pkg: .key, ver: .value.version}' package-lock.json`. For GitHub Actions, enumerate workflow YAML files with `grep -r 'npm install\|npm ci' .github/workflows/` to identify all affected pipeline files. Isolate build runners at the network level using host-based firewall rules: `ufw deny out from to any` or equivalent iptables rule.

Evidence: Before isolating any build runner or container, preserve: (1) full filesystem snapshot of the node_modules directory tree — the Mini Shai-Hulud worm modifies package files in place and evidence of the injected postinstall hook code will be present in node_modules/@antv/*/package.json and the corresponding index.js or dist files; (2) running process list from the build agent at time of detection (`ps auxf` on Linux, `Get-Process` on Windows) to capture any child processes spawned by the malicious postinstall hook; (3) network connection state (`ss -tupn` or `netstat -ano`) to capture any active outbound connections initiated by the npm postinstall execution; (4) environment variable dump (`printenv > /evidence/env-dump.txt`) from the build runner — the worm targets env vars for credential harvesting and the captured state documents what secrets were exposed.

Step 2: Detection — Query CI/CD pipeline logs, container runtime logs, and endpoint telemetry for outbound connections from build environments to unexpected external IPs or domains. Hunt for process execution anomalies in npm install or postinstall hooks (T1059). Check for OIDC token issuance events in GitHub Actions audit logs that do not correspond to known workflows — stolen OIDC tokens were used to forge SLSA attestations (T1553, T1528). Search SIEM for credential access patterns matching T1552.001: file reads of ~/.aws/credentials, environment variable enumeration, and Vault token reads from build agents. IOC patterns are not fully published at this time; monitor StepSecurity and The Hacker News for updated IOC releases.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: correlate pipeline execution logs, OIDC audit events, and cloud provider logs to determine scope of credential exposure and attestation forgery

Controls: NIST IR-5 (Incident Monitoring) — track and document all pipeline executions that installed affected package versions, NIST AU-6 (Audit Record Review, Analysis, and Reporting) — review GitHub Actions audit logs for anomalous OIDC token issuance events tied to TeamPCP's SLSA attestation forgery technique, NIST AU-12 (Audit Record Generation) — ensure cloud provider audit logging was active during the compromise window for AWS CloudTrail, GCP Audit Logs, and Azure Activity Log, NIST SI-4 (System Monitoring) — monitor build environments for T1059 process execution from npm postinstall hooks spawning unexpected child processes, CIS 8.2 (Collect Audit Logs) — validate that CI/CD pipeline logs, container runtime logs, and cloud provider logs are retained and queryable for the full compromise window

Compensating: For teams without SIEM: (1) Extract GitHub Actions audit log via API — `gh api /orgs/{org}/audit-log --paginate --jq '.[] | select(.action == "oidc.create_registration" or .action == "oidc.update_registration")' > oidc-audit.json` — flag any OIDC registrations not matching known workflow file SHAs. (2) Hunt postinstall hook execution in runner logs by grepping GitHub Actions job logs for lines containing 'postinstall' or 'node scripts/' within the time window of any flagged npm install. (3) Use osquery to detect credential file access on build agents: `SELECT * FROM process_open_files WHERE path LIKE '%/.aws/credentials' OR path LIKE '%/.config/gcloud%';` (4) For outbound connection detection without EDR, enable auditd on Linux runners with `auditctl -a always,exit -F arch=b64 -S connect -k npm-outbound` and review `/var/log/audit/audit.log` for connections initiated by node processes during npm install windows. MITRE ATT&CK T1528 (Steal Application Access Token) and T1553 (Subvert Trust Controls) are

the primary technique signatures to hunt.

Evidence: Capture before analysis is complete: (1) GitHub Actions audit log for the full organization covering 90 days prior to detection — specifically filter for `actions.create_workflow_run`, `oidc.create_registration`, and any `npm.publish` events that could indicate the worm's self-replication via stolen OIDC tokens; (2) npm audit log from any private registry (Verdaccio, Artifactory, GitHub Packages) showing which internal packages were re-published during the compromise window — the worm self-replicates by injecting into packages and republishing; (3) AWS CloudTrail `GetSecretValue`, `AssumeRoleWithWebIdentity`, and `CreateAccessKey` events from build agent IAM roles in the window following any flagged npm install; (4) Container runtime logs (Docker daemon log at `/var/log/docker.log` or `journalctl -u docker`) for any containers that executed npm install with flagged package versions, preserving the full stdout/stderr of the postinstall hook execution.

Step 3: Eradication — Pin all @antv packages and other affected dependencies to known-clean versions predating the compromise window. Remove malicious versions from local npm caches and private registries. Do not rely solely on SLSA provenance attestations to verify cleanliness — the worm forges these. Verify package integrity using independent hash comparison against pre-compromise release hashes where available. Re-evaluate any pipeline that grants OIDC token access to npm publish workflows.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: remove malicious package versions from all caches and registries, close the OIDC token scope vector that enabled SLSA attestation forgery, and restore dependency integrity through hash-verified pinning

Controls: NIST SI-2 (Flaw Remediation) — pin affected @antv and downstream packages to verified pre-compromise versions and validate with independent hash comparison, NIST SI-7 (Software, Firmware, and Information Integrity) — do not accept SLSA provenance attestations as integrity proof given TeamPCP's demonstrated OIDC-based forgery capability; require cryptographic hash verification against pre-compromise release hashes, NIST CM-7 (Least Functionality) — revoke or restrict OIDC token scopes in GitHub Actions workflows to prevent npm publish permissions from being abused in future worm self-replication cycles, CIS 7.3 (Perform Automated Operating System Patch Management) — extend automated patch management discipline to npm dependency pinning with hash verification in lockfiles, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — update vulnerability management process to include supply chain package integrity verification as a mandatory gate

Compensating: Clear npm cache on all build agents: `npm cache clean --force` then verify cache is empty with `npm cache verify`. For private registry cleanup (Verdaccio): identify and unpublish malicious versions with `npm unpublish @antv/g2@ --registry http://your-registry`. Perform independent hash verification without relying on SLSA: download the known-clean tarball from npm, compute `sha512sum .tgz`, and compare against the `integrity` field in your package-lock.json — if they match without going through npm's provenance chain, the package is clean. To restrict OIDC scope in GitHub Actions, edit workflow YAML to replace broad `id-token: write` permission with explicit resource-scoped tokens and remove `npm publish` from any OIDC-enabled workflow that does not require it. Use `npm ls --all` after re-install to verify the full dependency tree resolves to pinned clean versions only.

Evidence: Before purging caches and registries: (1) Extract and archive the malicious package tarballs from the local npm cache (located at `~/.npm/_cacache/` on Linux/macOS or `%AppData%\npm-cache` on Windows) — these contain the injected postinstall hook code and are primary forensic evidence of the worm's injection mechanism; (2) Dump the full content of any private registry's storage for affected package versions before unpublishing — for Verdaccio this is at `./storage/@antv//.tgz`; (3) Export the complete GitHub Actions workflow permission configuration (`gh api /repos/{owner}/{repo}/actions/permissions`) for all repositories where flagged packages were installed, documenting which workflows had `id-token: write` scope that enabled OIDC token access.

Step 4: Recovery — Rotate all credentials accessible from any build environment that ran a flagged package version. This includes AWS IAM keys, GCP service account credentials, Azure service principals, GitHub Actions secrets, Docker registry credentials, Kubernetes service account tokens, HashiCorp Vault tokens, and Stripe API keys. Redeploy affected services from clean build artifacts only. Audit cloud provider access logs (AWS CloudTrail, GCP Audit Logs, Azure Activity Log) for anomalous API calls in the window following any flagged install event. Validate that no new IAM roles, API keys, or access policies were created by

unauthorized actors.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery: restore systems from clean build artifacts, rotate all secrets exposed to build environments that executed malicious @antv or dependent package versions, and verify no persistent access was established by TeamPCP via stolen cloud credentials

Controls: NIST IR-4 (Incident Handling) — execute recovery in coordination with cloud provider security teams if anomalous API activity is confirmed in CloudTrail, GCP Audit Logs, or Azure Activity Log, NIST AC-2 (Account Management) — audit and disable any IAM users, service accounts, or API keys created during the compromise window across AWS, GCP, Azure, and GitHub, NIST IA-5 (Authenticator Management) — rotate all credentials confirmed exposed: AWS IAM keys, GCP service account keys, Azure service principal secrets, HashiCorp Vault tokens, Stripe API keys, Docker registry credentials, and Kubernetes service account tokens, NIST CP-9 (System Backup) — rebuild and redeploy affected services exclusively from build artifacts produced by clean pipelines post-eradication; do not reuse any artifact built during the compromise window, CIS 6.2 (Establish an Access Revoking Process) — apply documented access revocation process to all credentials and tokens accessible from build environments where flagged package versions executed

Compensating: Credential rotation by platform for a 2-person team: AWS — `aws iam list-access-keys --user-name ` then `aws iam delete-access-key` for each key, followed by `aws iam create-access-key`; audit for rogue keys with `aws iam generate-credential-report && aws iam get-credential-report`. GCP — `gcloud iam service-accounts keys list --iam-account=@.iam.gserviceaccount.com` then delete all keys and generate new. GitHub Actions — rotate secrets via `gh secret set ` for each affected repository. HashiCorp Vault — revoke all tokens issued to build agents with `vault token revoke -accessor ` and rotate root AppRole secret IDs. For post-rotation anomaly detection without SIEM, run a targeted CloudTrail query using AWS CLI: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=CreateRole --start-time ` to detect unauthorized IAM role creation by TeamPCP using stolen credentials.

Evidence: Before rotating credentials, preserve audit records that document unauthorized use: (1) AWS CloudTrail — export all events for the build agent's IAM role ARN covering the period from the earliest flagged npm install to present, specifically filtering for `AssumeRole`, `CreateUser`, `CreateAccessKey`, `PutRolePolicy`, and `GetSecretValue` events; (2) GCP Audit Logs — export `cloudaudit.googleapis.com/activity` logs for the affected service account, filtering for `iam.serviceAccounts.actAs`, `storage.objects.get`, and any `create` operations; (3) Azure Activity Log — export all operations by the affected service principal for the compromise window, filtering for `roleAssignments/write` and `secrets/write` operations; (4) HashiCorp Vault audit log (`/var/log/vault/audit.log`) for all token lookup, secret read, and auth operations from build agent source IPs during the compromise window — Vault's structured JSON audit log makes this directly greppable by `client_token` or `remote_address`.

Step 5: Post-Incident — This campaign exposed a critical gap: SLSA provenance verification alone is insufficient when OIDC token theft enables attestation forgery. Implement additional controls — enforced dependency pinning with hash verification (not just provenance), private registry mirroring with pre-ingestion scanning, and least-privilege OIDC token scoping in GitHub Actions. Treat worm source code availability on BreachForums as a persistent threat indicator: assess your npm dependency surface for exposure to copycat campaigns using the same techniques. Review SAP system access logs if SAP assets are in scope.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: conduct lessons-learned review focused on SLSA attestation forgery gap, update detection playbooks for npm supply chain worm behavior, and share indicators with community — worm source code on BreachForums indicates imminent copycat campaigns requiring updated detection posture

Controls: NIST IR-4 (Incident Handling) — update incident handling procedures to treat forged SLSA provenance as an active threat vector, not a verification control, NIST IR-8 (Incident Response Plan) — revise IR plan to include npm supply chain compromise as a named scenario with specific detection signatures for postinstall hook abuse and OIDC token theft, NIST SI-2 (Flaw Remediation) — institutionalize hash-pinned dependency management as a flaw remediation control; document that SLSA provenance verification was defeated by TeamPCP via OIDC token theft and is insufficient as a sole integrity control, NIST RA-3 (Risk Assessment) — re-assess software supply chain risk incorporating the BreachForums publication of the Mini Shai-Hulud worm source code as a threat multiplier enabling

copycat campaigns, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — update vulnerability management process to include continuous npm dependency surface monitoring and private registry pre-ingestion scanning as mandatory controls, CIS 7.2 (Establish and Maintain a Remediation Process) — document hash-pinned dependency pinning and least-privilege OIDC scoping as required remediations in the updated supply chain risk remediation process

Compensating: For a 2-person team to implement durable post-incident controls without enterprise tooling: (1) Enforce hash pinning by committing `package-lock.json` with `npm ci` (not `npm install`) in all CI pipelines — `npm ci` enforces exact lockfile versions and fails on mismatch. (2) Stand up a local Verdaccio private registry mirror (`npx verdaccio`) with an upstream proxy to npmjs.com and configure a pre-ingestion YARA scan using a rule targeting the Mini Shai-Hulud postinstall hook pattern — write a YARA rule matching the worm's known injection strings once IOCs are published by StepSecurity. (3) Restrict GitHub Actions OIDC scope by adding `permissions: id-token: none` to all workflow jobs that do not explicitly require token issuance, and audit with `grep -r 'id-token: write' .github/workflows/`. (4) Subscribe to the GitHub Advisory Database RSS feed and StepSecurity's blog for copycat campaign indicators — the BreachForums publication of the worm source means new variants targeting additional maintainer accounts are a near-term threat. (5) Use `npm audit signatures` (available in npm v8.9+) to verify package signatures against the npm public key — note this does NOT defeat TeamPCP's OIDC forgery but will catch unsigned packages from copycat actors who lack OIDC access.

Evidence: Post-incident documentation to preserve for lessons-learned and potential regulatory reporting: (1) Complete timeline artifact mapping all flagged package versions installed across all build environments, correlated with cloud provider API events — this documents the blast radius for breach notification assessment; (2) Preserved copy of the malicious postinstall hook code extracted from forensic npm cache captures in Step 3 — this is primary evidence for threat intelligence sharing with CISA and npm security team; (3) GitHub Actions audit log entries showing OIDC token issuance events that were used for SLSA attestation forgery — document the specific workflow run IDs and token subjects to inform the lessons-learned on attestation verification gaps; (4) Record of all credentials rotated, systems rebuilt, and policy changes implemented — required for IR-8 plan update and any regulatory breach notification filing if PII or payment data was accessible from the compromised build environments.

Detection Guidance

Primary detection surfaces: CI/CD pipeline logs (GitHub Actions audit logs for anomalous OIDC token issuance), npm install/postinstall hook execution logs, and outbound network connections from build agents. Hunt for: (1) postinstall script execution in @antv/* or echarts-for-react packages that spawns child processes or makes network calls; (2) environment variable enumeration from build runners, particularly reads of AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, GOOGLE_APPLICATION_CREDENTIALS, AZURE_CLIENT_SECRET, VAULT_TOKEN, STRIPE_SECRET_KEY, and GITHUB_TOKEN; (3) outbound DNS or HTTP requests from build environments to domains not in your approved egress list, particularly during or immediately after npm install; (4) OIDC token requests in GitHub Actions audit logs not tied to known publish workflows; flag any oidc:write permission usage that wasn't in a known release job. SLSA attestation forgery means provenance checks will return false-positive clean results for malicious packages; do not use SLSA verification as a sole clearance gate. Monitor StepSecurity, CISA, and primary threat vendors for updated IOC releases; initial hash and C2 indicators may not be available for 24-48 hours post-disclosure.

Indicators of Compromise

Type	Value	Context	Confidence
HASH	`not published at time of writing`	Malicious package version hashes for 639 affected npm versions — not confirmed in available sources; monitor StepSecurity blog for release	LOW
DOMAIN	`not published at time of writing`	C2 or exfiltration infrastructure used by the worm — not confirmed in available sources at time of writing	LOW

Framework Mappings

MITRE-ATTACK

- **T1552.001** — Credentials In Files
- **T1528** — Steal Application Access Token
- **T1552** — Unsecured Credentials
- **T1567.001** — Exfiltration to Code Repository
- **T1176** — Software Extensions
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1611** — Escape to Host
- **T1587.001** — Malware
- **T1553** — Subvert Trust Controls
- **T1027** — Obfuscated Files or Information
- **T1078** — Valid Accounts
- **T1059** — Command and Scripting Interpreter
- **T1650** — Acquire Access

NIST-800-53R5

- **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
- **CM-7** — Least Functionality
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **SC-8** — Transmission Confidentiality and Integrity
- **SC-17** — Public Key Infrastructure Certificates
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures
- **A04:2021** — Insecure Design
- **A07:2021** — Identification and Authentication Failures
- **A02:2021** — Cryptographic Failures

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **5.2** — Use Unique Passwords
- **3.10** — Encrypt Sensitive Data in Transit
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers

HIPAA-SECURITY

- **164.308(a)(5)(ii)(D)** — Password Management
- **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.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
T1552.001	Credentials In Files	Credential-Access
T1528	Steal Application Access Token	Credential-Access
T1552	Unsecured Credentials	Credential-Access
T1567.001	Exfiltration to Code Repository	Exfiltration
T1176	Software Extensions	Persistence

Technique ID	Technique Name	Tactic
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1611	Escape to Host	Privilege-Escalation
T1587.001	Malware	Resource-Development
T1553	Subvert Trust Controls	Defense-Evasion
T1027	Obfuscated Files or Information	Defense-Evasion
T1078	Valid Accounts	Defense-Evasion
T1059	Command and Scripting Interpreter	Execution
T1650	Acquire Access	Resource-Development

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/05/mini-shai-hulud-pushes-malicious-...	T3
Shai-Hulud: Here We Go Again. Mass npm Supply Chain Attack Hits ...	https://www.stepsecurity.io/blog/shai-hulud-here-we-go-again-mass-n...	T3
AntV - G6 Graph Visualization Framework in JavaScript	https://g6.antv.antgroup.com/en	T3
@antv - npmx	https://npmx.dev/@antv	T3
antv/f2 vs @antv/g2 vs @antv/g6 vs @antv/l7 vs @antv/x6 - npm trends	https://npmtrends.com/@antv/f2-vs-@antv/g2-vs-@antv/g6-vs-@antv/l7-...	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 2026-05-19 06:43 UTC by TJS Security Command Center