INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-19 18:48 UTC

Shai-Hulud Wave 3: Forged Provenance, P2P Exfiltration, and IDE Backdoors Mark a New Threshold in npm Supply Chain Attacks

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0339
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	npm ecosystem: @antv/g2, @antv/g6, @antv/x6, @antv/l7, @antv/g2plot, @antv/graphin, echarts-for-react, timeago.js, size-sensor, canvas-nest.js, jest-canvas-mock (323 packages, 639 malicious versions); Development environments: VS Code, Claude Code; CI/CD: GitHub Actions, GitLab CI, Jenkins, Azure DevOps, CircleCI; Infrastructure: Kubernetes, Docker, HashiCorp Vault; Platforms: GitHub, Vercel, Netlify
Published	2026-05-19T10:30:22
Discovery Source	Rss

Executive Summary

On May 19, 2026, attackers injected malicious code into 639 versions of 323 npm packages within a single hour, targeting widely-used data visualization libraries from the AntV ecosystem and other high-download packages. The campaign introduces forged software provenance certificates, self-spreading worm behavior, and backdoors that persist inside developer IDE extensions after packages are cleaned up, meaning standard remediation may leave environments compromised. Any organization whose developers installed affected packages may have exposed source code, cloud credentials, Kubernetes secrets, and SSH keys to attacker-controlled infrastructure.

Technical Analysis

The Shai-Hulud Wave 3 campaign executed on 2026-05-19, pushing 639 malicious versions across 323 npm packages in under 60 minutes (per BleepingComputer, Aikido, and StepSecurity reporting as of 2026-05-20). Confirmed affected packages include @antv/g2, @antv/g6, @antv/x6, @antv/l7, @antv/g2plot, @antv/graphin, echarts-for-react, timeago.js, size-sensor, canvas-nest.js, and jest-canvas-mock.

Five novel capabilities distinguish this wave from prior Shai-Hulud activity:

1. Forged Sigstore/SLSA provenance attestations (CWE-295, CWE-494): Malicious versions carried fabricated attestation signatures designed to pass standard supply chain verification tooling, undermining attestation-based controls in CI/CD pipelines.

2. Self-propagating worm (CWE-506, T1195.001, T1554): Malware harvested npm tokens from compromised developer environments and used them to republish infected versions, enabling autonomous lateral spread across the registry without further attacker interaction.

3. IDE-layer persistence (T1176, T1547): Backdoors were implanted in VS Code and Claude Code extensions. These persist after npm package removal, surviving standard incident response cleanup steps.

4. Credential harvesting (CWE-522, CWE-312, T1552.001, T1552.004, T1528, T1550.001): Targeted artifacts include GitHub tokens, AWS/GCP/Azure credentials, Kubernetes secrets, HashiCorp Vault tokens, Docker credentials, and SSH private keys.

5. P2P exfiltration via Session Protocol encrypted channels (T1041, T1567, T1027): Stolen data exits over peer-to-peer encrypted channels, bypassing perimeter egress monitoring and network-based DLP.

Additional MITRE techniques observed: T1059.007 (JavaScript execution), T1053 (scheduled tasks for persistence), T1078/T1078.001 (valid account abuse), T1567.001 (exfiltration to code repository), T1650 (acquire access).

No CVE IDs assigned. No vendor-issued patches; remediation requires version pinning, audit of installed extensions, and credential rotation. As of 2026-05-20, no official security advisory has been published by npm or GitHub Security. This assessment relies on T3 threat intelligence sources (BleepingComputer, Aikido, StepSecurity); verification from npm, GitHub Security, or CISA is pending. Confidence in TTP and affected package list is high based on cross-source corroboration, but readers should monitor official channels for updated guidance.

CWE references: CWE-494 (Download of Code Without Integrity Check), CWE-295 (Improper Certificate Validation), CWE-829 (Inclusion of Functionality from Untrusted Control Sphere), CWE-312 (Cleartext Storage of Sensitive Information), CWE-506 (Embedded Malicious Code), CWE-522 (Insufficiently Protected Credentials).

Action Checklist

1. Containment: Immediately pin @antv/g2, @antv/g6, @antv/x6, @antv/l7, @antv/g2plot, @antv/graphin, echarts-for-react, timeago.js, size-sensor, canvas-nest.js, and jest-canvas-mock to versions published before 2026-05-19 in all CI/CD pipelines (GitHub Actions, GitLab CI, Jenkins, Azure DevOps, CircleCI). Do not allow npm install to pull versions from 2026-05-19 onward for these packages. Isolate any build system or developer workstation that installed affected versions since 2026-05-19.

2. Detection: Audit installed VS Code and Claude Code extensions on all developer machines for unrecognized or recently modified extensions; malicious IDE backdoors survive package cleanup. Review npm audit logs and CI/CD build logs for installs of the affected packages between 2026-05-18 and present. Check for outbound connections to Session Protocol (Session messenger) P2P infrastructure, this traffic will appear as encrypted UDP/TCP to unfamiliar endpoints and will not match standard npm registry or CDN patterns. Search secrets management systems (HashiCorp Vault, Kubernetes secrets stores) for access events originating from developer workstation IPs or CI/CD runner IPs during the exposure window.

3. Eradication: Remove all affected package versions from lockfiles and caches. Pin to versions published before 2026-05-19 by checking the npm registry listing for each package. Verify lockfile integrity hashes (package-lock.json integrity field or yarn.lock sha512) match the pre-2026-05-19 version from npm's public registry. Do not use cached or mirror-sourced versions. Remove and reinstall VS Code and Claude Code extensions from scratch on any machine that installed affected packages; do not attempt to audit extensions in place. Revoke and rotate all credentials accessible from affected environments: GitHub tokens, AWS/GCP/Azure IAM keys, Kubernetes service account tokens, HashiCorp Vault tokens, Docker registry credentials, and SSH keys. Treat npm tokens on affected developer machines as compromised and rotate immediately to stop worm self-propagation.

4. Recovery: After credential rotation, audit cloud provider access logs (AWS CloudTrail, GCP Audit Logs, Azure Activity Log) for anomalous API calls using the old credentials during the exposure window. Verify Kubernetes RBAC and HashiCorp Vault audit logs for unauthorized secret access. Re-run your full dependency tree build from a clean environment with lockfile integrity verification. Monitor CI/CD pipeline outputs for unexpected new package versions being published to npm, the worm uses stolen tokens to republish, so watch your own package namespaces for unauthorized version bumps.

5. Post-Incident: This attack defeated attestation-based supply chain defenses via forged Sigstore/SLSA signatures; review whether your pipeline's provenance verification can detect forged attestations or whether it relies solely on signature presence. Implement IDE extension allowlisting policies for developer workstations, unmanaged extension installation was the persistence vector here. Evaluate adoption of lockfile integrity enforcement (npm ci over npm install) and private registry proxies with content inspection. Map this incident to SLSA threat model gaps and NIST SP 800-161r1 C-SCRM controls for future supply chain risk posture review.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal, and external IR retainer immediately if HashiCorp Vault audit logs, Kubernetes RBAC logs, or cloud provider audit logs confirm unauthorized secret access or API calls using compromised credentials, or if npm registry evidence shows the worm successfully published malicious versions under your organization's package namespaces — either condition indicates active data exfiltration or downstream customer impact triggering breach notification assessment.
Recovery Notes	After credential rotation and environment rebuild, maintain elevated monitoring on your organization's npm package namespaces and cloud provider IAM activity for a minimum of 30 days, as the Shai-Hulud worm's use of stolen tokens for self-propagation means secondary infections in connected developer environments may not surface immediately. Verify that all rebuilt CI/CD pipelines produce deterministic builds using `npm ci` with lockfile integrity checks and that no pipeline stages have write access to npm registry credentials except a dedicated, gated publish job. Confirm with all affected developers that VS Code and Claude Code extensions have been fully wiped and reinstalled — do not accept self-reported audits, as the IDE backdoor is specifically designed to survive cleanup attempts that stop short of full extension directory deletion.

Forensic Artifacts	VS Code extension directory forensic copy (~/.vscode/extensions/ on Linux/macOS, %USERPROFILE%\.vscode\extensions\ on Windows): extensions installed or modified after 2026-05-19 are the primary persistence artifacts for the IDE backdoor component; preserve full directory with timestamps before any remediation | npm cache and lockfile artifacts (~/.npm/_cacache/, package-lock.json, yarn.lock, .yarn/cache/): contain cryptographic integrity hashes of the exact malicious package versions fetched, enabling confirmation of which of the 639 compromised versions were installed on each system | HashiCorp Vault audit log (/var/log/vault/audit.log or Vault audit backend output): JSON-structured log of every secret read/write operation with requestor token accessor, remote IP, and timestamp — primary evidence for determining which secrets were accessed from compromised developer or CI runner identities during the exposure window | Network packet capture or firewall flow logs showing outbound UDP/TCP from build machines to non-npm-registry endpoints: Session Protocol P2P exfiltration traffic produces sustained encrypted flows to rotating IP addresses outside npm registry ASNs (Fastly AS54113, Cloudflare AS13335); these flows are the forensic signature of active exfiltration by the malicious package payload | AWS CloudTrail / GCP Audit Logs / Azure Activity Log exports for the exposure window (2026-05-19 through credential rotation): API calls using compromised IAM keys, GitHub tokens, or service account credentials are the authoritative record of post-exploitation lateral movement and data access, and are required for breach scope determination and regulatory notification decisions

Per-Action IR Details

Containment — Immediately audit your npm dependency tree for any of the 11 confirmed affected packages: @antv/g2, @antv/g6, @antv/x6, @antv/l7, @antv/g2plot, @antv/graphin, echarts-for-react, timeago.js, size-sensor, canvas-nest.js, jest-canvas-mock. Block installation of new package versions from these namespaces in your CI/CD pipelines (GitHub Actions, GitLab CI, Jenkins, Azure DevOps, CircleCI) until provenance is confirmed clean. Isolate any build systems or developer workstations that installed affected versions since 2026-05-19.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: isolate affected systems, prevent further spread before eradication begins

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST CM-3 (Configuration Change Control), CIS 2.3 (Address Unauthorized Software), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Run `npm ls --all 2>/dev/null | grep -E '@antv/(g2|g6|x6|l7|g2plot|graphin)|echarts-for-react|timeago.js|size-sensor|canvas-nest.js|jest-canvas-mock'` across all developer workstations via a pushed shell script or Ansible ad-hoc task. For CI/CD blocking without a paid registry proxy, add a pre-install step to GitHub Actions or GitLab CI that fails the build if any of the 11 package names appear in `package-lock.json` or `yarn.lock` with a version published after 2026-05-18: use `node -e "const l=require('./package-lock.json'); Object.keys(l.packages).forEach(p=>{if(//.test(p)) process.exit(1)})"`. Network-isolate affected build runners by removing their outbound internet routing rules at the host firewall level using `iptables -I OUTPUT -j DROP` until the pipeline is patched.

Evidence: Before isolating, snapshot the full resolved dependency tree with `npm ls --all --json > dep-tree-$(hostname)-$(date +%Y%m%d%H%M%S).json` to preserve which malicious versions were present. Capture `~/.npm/_cacache/` directory listing and `package-lock.json` / `yarn.lock` from every affected project — these record the exact resolved version hashes that were installed and will confirm whether compromised versions were fetched. Preserve CI/CD runner job logs from GitHub Actions (`~/.npm/_logs/`, runner `_diag/` directory) or GitLab CI job artifacts that show `npm install` output with resolved version strings for the 11 packages between 2026-05-19T00:00Z and present.

Detection — Audit installed VS Code and Claude Code extensions on all developer machines for unrecognized or recently modified extensions; malicious IDE backdoors survive package cleanup. Review npm audit logs and CI/CD build logs for installs of the affected packages between 2026-05-18 and present. Check for outbound connections to Session Protocol (Session messenger) P2P infrastructure — this traffic will appear as encrypted UDP/TCP to unfamiliar endpoints and will not match standard npm registry or CDN patterns. Search secrets management systems (HashiCorp Vault, Kubernetes secrets stores) for access events originating from developer workstation IPs or CI/CD runner IPs during the exposure window.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: correlate indicators across log sources, identify scope of compromise, and declare incident when criteria are met

Controls: NIST SI-4 (System Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-12 (Audit Record Generation), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: Enumerate VS Code extensions on all developer machines with `code --list-extensions --show-versions > vscode-extensions-$(hostname)-$(date +%Y%m%d).txt`; diff output against a known-good baseline or flag any extension with a `lastModifiedDate` (found in `~/.vscode/extensions//package.json`) after 2026-05-18. For Claude Code, inspect `~/.claude/extensions/` or the equivalent platform directory for recently written files using `find ~/.claude -newer /tmp/ref_date -type f`. Detect Session Protocol P2P exfiltration traffic without a SIEM by running `tcpdump -nn -i any 'not (dst net 104.16.0.0/12 or dst net 151.101.0.0/16 or dst net 8.8.8.0/24) and (udp or tcp) and not port 443' -w session-p2p-$(date +%Y%m%d).pcap` on developer workstation network segments; Session Protocol uses onion-routed encrypted UDP that will produce sustained traffic to rotating IPs not in npm/CDN ASNs. Query HashiCorp Vault audit log (`vault audit list`; default log at `/var/log/vault/audit.log`) for `auth.accessor` values tied to developer or CI runner tokens during the exposure window using `jq 'select(.time >= "2026-05-19") | select(.request.remote_address | test(""))' audit.log`.

Evidence: Collect VS Code extension directory timestamps (`ls -la ~/.vscode/extensions/`) and the `package.json` of each extension installed or modified after 2026-05-18 — the backdoor will present as a new or silently updated extension. Extract npm install history from `~/.npm/_logs/*.log` filtering for the 11 package names. Pull HashiCorp Vault audit logs and Kubernetes API server audit logs (`/var/log/kubernetes/audit.log` or via `kubectl get events`) for secret read operations (`secrets/data/*` in Vault; `get`/`list` on `secrets` resource in k8s) from workstation or runner IPs after 2026-05-19. Capture a full packet capture or firewall flow logs showing outbound UDP/TCP to non-registry endpoints from build machines during the exposure window to identify Session Protocol C2 beaconing patterns.

Eradication — Remove all affected package versions from lockfiles and caches; pin to clean versions verified against known-good checksums from the npm registry, confirming version publish timestamps predate 2026-05-19. Remove and reinstall VS Code and Claude Code extensions from scratch on any machine that installed affected packages — do not attempt to audit extensions in place. Revoke and rotate all credentials accessible from affected environments: GitHub tokens, AWS/GCP/Azure IAM keys, Kubernetes service account tokens, HashiCorp Vault tokens, Docker registry credentials, and SSH keys. Treat npm tokens on affected developer machines as compromised and rotate immediately to stop worm self-propagation.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: remove threat artifacts from all affected systems, verify removal, and address root causes before recovery begins

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST IA-5 (Authenticator Management), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Verify clean package checksums by running `npm view @ dist.integrity` for each pinned version and comparing against the `integrity` field in your `package-lock.json` — mismatches confirm tampered packages. Purge the npm cache entirely with `npm cache clean --force` and delete `node_modules` before reinstalling. For VS Code, fully remove the extensions directory (`rm -rf ~/.vscode/extensions`) and re-provision from a curated list using `cat approved-extensions.txt | xargs -I{} code --install-extension {}` — do not use 'Update All' from within a potentially backdoored instance. Revoke GitHub tokens via `gh auth token` and the GitHub API (`DELETE

/applications/{client_id}/token`); rotate Kubernetes service account tokens by deleting and recreating the secret (`kubectl delete secret -n `); cycle HashiCorp Vault tokens with `vault token revoke -accessor ` for each exposed accessor found in audit logs.

Evidence: Before wiping `~/.vscode/extensions/`, take a forensic copy of all extension directories modified after 2026-05-18 — the backdoor source files will contain the malicious payload and may reveal C2 addresses or exfiltration logic for threat intelligence. Preserve `~/.npmrc` and `/etc/npmrc` showing any token values before rotation, storing them in a secured evidence repository (not version control). Document all Kubernetes secrets and Vault paths that were readable by compromised service accounts by running `vault token capabilities ` against each suspected path before revocation — this scopes the data-at-risk for breach notification assessment. Capture the process list and loaded module list (`ps auxf`, `lsmod` on Linux; `Get-Process` on Windows) from affected machines before reimaging to detect any persistence mechanisms beyond IDE extensions.

Recovery — After credential rotation, audit cloud provider access logs (AWS CloudTrail, GCP Audit Logs, Azure Activity Log) for anomalous API calls using the old credentials during the exposure window. Verify Kubernetes RBAC and HashiCorp Vault audit logs for unauthorized secret access. Re-run your full dependency tree build from a clean environment with lockfile integrity verification. Monitor CI/CD pipeline outputs for unexpected new package versions being published to npm — the worm uses stolen tokens to republish, so watch your own package namespaces for unauthorized version bumps.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery: restore systems to normal operation, verify integrity of restored systems, and monitor for recurrence

Controls: NIST IR-4 (Incident Handling), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-11 (Audit Record Retention), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-3 (Configuration Change Control), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory)

Compensating: Query AWS CloudTrail for API calls by the compromised IAM keys using `aws cloudtrail lookup-events --lookup-attributes AttributeKey=Username,AttributeValue= --start-time 2026-05-19 --end-time ` filtering for `CreateUser`, `PutRolePolicy`, `GetSecretValue`, `AssumeRole`, and `s3:GetObject` event names that indicate lateral movement or data access. Monitor your npm organization namespaces for unauthorized publishes by polling `https://registry.npmjs.org/-/v1/search?text=maintainer:&updated_after=` or subscribing to npm's webhook for your organization — the Shai-Hulud worm publishes new malicious versions using stolen tokens within minutes of credential theft, so polling frequency should be every 5 minutes during the active recovery window. Re-run `npm ci` (not `npm install`) from a clean ephemeral build container using a pinned base image with no pre-existing `node_modules` or npm cache, and verify the resulting `node_modules/.package-lock.json` integrity hashes match your approved lockfile.

Evidence: Preserve AWS CloudTrail, GCP Audit Logs, and Azure Activity Log exports for the full exposure window (2026-05-19 through credential rotation date) before log retention windows expire — these are the primary evidence source for determining whether stolen IAM credentials were used for data exfiltration or privilege escalation. Export Kubernetes audit logs showing all `get`/`list`/`watch` operations on `secrets` resources by compromised service account identities. Retain npm registry publish history for your organization's packages (retrievable via `npm info time --json`) to document whether the worm successfully republished any of your packages using stolen tokens — this is critical evidence for downstream customer notification decisions.

Post-Incident — This attack defeated attestation-based supply chain defenses via forged Sigstore/SLSA signatures; review whether your pipeline's provenance verification can detect forged attestations or whether it relies solely on signature presence. Implement IDE extension allowlisting policies for developer workstations — unmanaged extension installation was the persistence vector here. Evaluate adoption of lockfile integrity enforcement (npm ci over npm install) and private registry proxies with content inspection. Map this incident to SLSA threat model gaps and NIST SP 800-161r1 C-SCRM controls for future supply chain risk posture review.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: lessons learned meeting, evidence retention, capability improvement, and threat intelligence sharing

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SI-7 (Software, Firmware, and Information Integrity), NIST SA-12 (Supply Chain Protection), NIST RA-3 (Risk Assessment), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: For IDE extension allowlisting without an MDM/endpoint management platform, enforce VS Code extension policy via a workspace `.vscode/extensions.json` with `unwantedRecommendations` listing all non-approved extension IDs, and deploy a Git pre-commit hook that fails if `.vscode/extensions.json` is modified without a code review approval. Implement Sigstore cosign signature transparency log verification in CI/CD by adding `cosign verify --certificate-identity-regexp= --certificate-oidc-issuer=https://token.actions.githubusercontent.com ` as a mandatory pipeline gate — this validates the certificate identity chain, not just signature presence, which is the gap the Shai-Hulud campaign exploited. Stand up a local Verdaccio private registry proxy (`npx verdaccio`) configured to cache and content-inspect packages against a YARA ruleset targeting the malicious payload patterns identified in this campaign before forwarding to the public registry.

Evidence: Retain all forensic artifacts, build logs, and cloud audit exports for a minimum of 12 months to support potential regulatory breach notification timelines and any downstream customer disclosure requirements. Document the exact Sigstore/SLSA attestation artifacts that were accepted by your pipeline for the compromised package versions — these forged certificates are primary evidence for reporting to Sigstore's transparency log monitors and for improving attestation verification logic. Produce a post-incident timeline mapping the 639 malicious version publish timestamps (all within a one-hour window on 2026-05-19) against your CI/CD build job history to precisely bound the exposure window for each affected system and support accurate breach scope determination.

Detection Guidance

Key detection signals for Shai-Hulud Wave 3:

1. npm install logs: Search CI/CD build logs and local npm cache logs for installs of @antv/g2, @antv/g6, @antv/x6, @antv/l7, @antv/g2plot, @antv/graphin, echarts-for-react, timeago.js, size-sensor, canvas-nest.js, or jest-canvas-mock with version publish timestamps on or after 2026-05-19.

2. IDE extension audit: On developer workstations, enumerate VS Code extensions (code --list-extensions) and Claude Code extensions; flag any extension installed or modified after 2026-05-18 that is not on your approved list. Unrecognized extensions with filesystem or network permissions are high-confidence indicators.

3. Network indicators: Look for outbound encrypted traffic to Session Protocol P2P nodes, this will not match known npm, GitHub, or CDN egress patterns. Flag unusual encrypted UDP traffic from developer machines or CI/CD runners, particularly to IPs outside your standard vendor ranges. Standard DLP and egress filters will not catch this, inspect at the DNS query and connection destination level.

4. Credential use anomalies: Alert on GitHub token use from IPs other than the token owner's known workstation or your CI/CD runner range. Flag Kubernetes secret reads and HashiCorp Vault token use originating from build environments outside normal job execution windows.

5. npm publish events: If your organization publishes npm packages, monitor for unexpected version publishes to your namespaces, the worm uses harvested tokens to self-propagate by republishing infected versions.

6. MITRE-mapped behavioral signals: T1176 (new browser/IDE extension installs on developer hosts), T1554 (modification of software in the software supply chain, watch for lockfile changes not tied to a PR), T1552.004 (SSH private key reads from non-interactive processes), T1567.001 (git push or API calls to GitHub from unexpected processes).

Note: Forged Sigstore/SLSA attestations mean standard provenance verification tools may return false-clean results. Do not rely solely on attestation checks for these packages.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`registry.npmjs.org (malicious versions)`	Malicious package versions were published to the official npm registry for 11 confirmed packages including @antv/g2, @antv/g6, @antv/x6. Block or audit installs of versions with publish timestamps on or after 2026-05-19.	HIGH
URL	`https://www.npmjs.com/package/@antv/g2 (versions >= 2026-05-19)`	Confirmed affected package namespace. Treat any version published on or after 2026-05-19 as suspect until npm security team confirms clean versions.	HIGH
URL	`https://www.npmjs.com/package/@antv/g6 (versions >= 2026-05-19)`	Confirmed affected package namespace.	HIGH
URL	`https://www.npmjs.com/package/echarts-for-react (versions >= 2026-05-19)`	Confirmed affected package namespace.	HIGH
URL	`Session Protocol P2P network (encrypted exfiltration channel)`	Stolen credentials and secrets were exfiltrated over Session Protocol encrypted P2P channels. Flag outbound encrypted traffic to unfamiliar endpoints not matching npm/GitHub/CDN ranges from developer machines or CI/CD runners.	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1176** — Software Extensions
- **T1554** — Compromise Host Software Binary
- **T1550.001** — Application Access Token
- **T1547** — Boot or Logon Autostart Execution
- **T1552.004** — Private Keys
- **T1053** — Scheduled Task/Job
- **T1041** — Exfiltration Over C2 Channel
- **T1528** — Steal Application Access Token
- **T1567.001** — Exfiltration to Code Repository

- **T1567** — Exfiltration Over Web Service
- **T1078.001** — Default Accounts
- **T1027** — Obfuscated Files or Information
- **T1650** — Acquire Access
- **T1078** — Valid Accounts
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1552.001** — Credentials In Files
- **T1059.007** — JavaScript

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **AC-3** — Access Enforcement
- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **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
- **SC-13** — Cryptographic Protection

OWASP-TOP10-2021

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

CIS-V8

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

- **8.2** — Collect Audit Logs

HIPAA-SECURITY

- **164.308(a)(5)(ii)(D)** — Password Management
- **164.312(d)** — Person or Entity Authentication
- **164.312(e)(1)** — Transmission Security

SOC2-TSC

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

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program
- **DE.CM-01** — Networks and network services are monitored

ISO-27001-2022

- **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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1176	Software Extensions	Persistence
T1554	Compromise Host Software Binary	Persistence
T1550.001	Application Access Token	Defense-Evasion
T1547	Boot or Logon Autostart Execution	Persistence
T1552.004	Private Keys	Credential-Access
T1053	Scheduled Task/Job	Execution
T1041	Exfiltration Over C2 Channel	Exfiltration
T1528	Steal Application Access Token	Credential-Access
T1567.001	Exfiltration to Code Repository	Exfiltration
T1567	Exfiltration Over Web Service	Exfiltration
T1078.001	Default Accounts	Defense-Evasion
T1027	Obfuscated Files or Information	Defense-Evasion
T1650	Acquire Access	Resource-Development

Technique ID	Technique Name	Tactic
T1078	Valid Accounts	Defense-Evasion
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1552.001	Credentials In Files	Credential-Access
T1059.007	JavaScript	Execution

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/new-shai-hulud-malwa...	T3
Mini Shai-Hulud Strikes Again: npm Worm Compromises Hundreds ...	https://www.aikido.dev/blog/mini-shai-hulud-antv-npm-supply-chain-a...	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
G2: The Concise and Progressive Visualization Grammar - GitHub	https://github.com/antvis/g2	T3
@antv/g6 CDN by jsDelivr - A CDN for npm and GitHub	https://www.jsdelivr.com/package/npm/@antv/g6	T3

Generated 2026-05-19 18:48 UTC by TJS Security Command Center