INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-27 18:54 UTC

TanStack Supply Chain Compromise: Malicious npm Package Publishes Credential-Stealing Malware

CVE VULNERABILITY | **CRITICAL** | CVSS 9.3 | **CISA KEV**

SCC Item ID	SCC-CVE-2026-0233
Type	CVE Vulnerability
CVE ID	CVE-2026-45321
Severity	CRITICAL
CVSS Base Score	9.3
EPSS Score	0.0003 (8th percentile)
KEV Status	Yes — CISA Known Exploited Vulnerability (due: 2026-06-10)
Affected Products	TanStack (npm registry packages, specific package versions unspecified in available data)
Published	2026-05-27
Discovery Source	Cisa Kev

Executive Summary

Threat actors compromised TanStack's npm publishing pipeline and released malicious package versions containing credential-stealing malware, executing a supply chain attack against any organization whose software build process consumes TanStack npm packages. CISA has added this to the Known Exploited Vulnerabilities catalog, indicating active exploitation, with a remediation deadline of June 10, 2026. Any organization with TanStack packages in its dependency tree faces risk of credential theft and downstream environment compromise through infected builds. Although the EPSS probabilistic score is low (0.027%), CISA's KEV inclusion indicates active exploitation is confirmed; probabilistic scoring should not be used to reduce remediation priority.

Technical Analysis

CVE-2026-45321 (CVSS 9.3, Critical) describes a supply chain compromise affecting TanStack npm registry packages. Adversaries published malicious package versions under the legitimate TanStack namespace, embedding credential-stealing malware targeting consumers of those packages at build or install time. Based on available CISA and NVD data, specific affected package names, version ranges, and the enabling mechanism (compromised publish credentials, CI/CD pipeline abuse, or publishing workflow flaw) remain unconfirmed in

public source material. Treat all TanStack npm packages as potentially compromised until the vendor publishes integrity verification data and confirmed clean versions. Relevant CWEs: CWE-1357 (Reliance on Insufficiently Trustworthy Component) and CWE-494 (Download of Code Without Integrity Check). MITRE ATT&CK techniques: T1195.001 (Supply Chain Compromise: Compromise Software Dependencies and Development Tools), T1078 (Valid Accounts), T1555 (Credentials from Password Stores). CISA KEV remediation due date: 2026-06-10.

Action Checklist

1. Step 1: Containment, Immediately halt all automated dependency updates and pin all TanStack packages to their currently installed versions in all repositories and CI/CD pipelines. Do not install or update TanStack packages in any environment until verified clean versions are published by TanStack maintainers. Conduct a complete audit of dependency manifests (package.json, package-lock.json, yarn.lock) across all repositories to identify all TanStack package dependencies. Reference: NIST SI-3 (Malicious Code Protection), CIS 2.3 (Ensure Authorized Software is Currently Supported), CIS 2.4 (Address Unauthorized Software).
2. Step 2: Detection, Audit build logs and npm install logs covering the period during which malicious versions were available; treat all recent TanStack package installs as potentially compromised and require forensic review. Search for anomalous outbound network connections initiated during or after npm install processes; credential-stealing malware commonly beacons to external infrastructure. Review endpoint and server logs for unexpected process spawning from Node.js or npm processes. Check for access to credential stores (OS keychains, environment variable files, .npmrc, .gitconfig, CI/CD secret stores) by build tooling. Use NIST AU-6 (Audit Record Review, Analysis, and Reporting) and CIS 8.2 (Collect Audit Logs) to ensure log coverage exists. D3FEND countermeasure: D3-SFA (System File Analysis), monitor credential files and config files for unexpected reads.
3. Step 3: Eradication, Remove all installed TanStack package versions from build environments, artifact registries, and deployed applications until TanStack publishes verified clean versions with confirmed integrity hashes. Rotate all credentials that were present in any environment where a TanStack package was installed during the suspected compromise window: npm tokens, CI/CD secrets, cloud provider keys, API keys, and developer workstation credentials. Verify no unauthorized access occurred using rotated credentials before restoring them to active use. Review access logs in affected systems (cloud provider, repository, CI/CD) for the compromise window. Apply D3-CRO (Credential Rotation) across affected systems. Reference: NIST IA-5 (Authenticator Management), CIS 5.2 (Use Unique Passwords).
4. Step 4: Recovery, Before reintroducing any TanStack dependency, verify package integrity using npm audit and subresource integrity checks against the official TanStack release advisory (monitor the TanStack GitHub and npm advisory feed for this publication). Validate that CI/CD pipeline secrets and deployment credentials have been rotated and that no unauthorized access occurred using stolen credentials. Re-enable builds only after clean-version confirmation. Monitor post-remediation builds for anomalous outbound connections. Reference: NIST SI-7 (Software, Firmware, and Information Integrity), CIS 2.4 (Address Unauthorized Software).
5. Step 5: Post-Incident, This incident exposes control gaps in software supply chain integrity verification. Implement or strengthen: (1) dependency pinning with hash verification (CIS 2.4, Address Unauthorized Software); (2) CI/CD pipeline secret isolation so npm publish tokens are scoped and rotated regularly (NIST AC-6, Least Privilege); (3) software composition analysis (SCA) tooling in CI/CD pipelines to flag newly published versions before auto-install; (4) audit logging of all package installations in build

environments (NIST AU-2, Event Logging, CIS 8.2, Collect Audit Logs); (5) review and restrict which identities hold publish rights to your own npm packages to prevent analogous outbound risk (NIST AC-5, Separation of Duties, NIST AC-2, Account Management). D3FEND countermeasure: D3-FMBV (File Magic Byte Verification), verify package integrity artifacts at install time.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and breach notification counsel immediately if cloud provider access logs, GitHub audit logs, or CI/CD platform audit trails show any API calls, repository access, or authentication events using credentials that were present in a build environment during the compromise window — confirmed credential use by threat actors triggers breach notification obligations under applicable state, federal, and international data protection regulations.
Recovery Notes	Recovery is gated on two sequential confirmations: first, TanStack maintainers must publish verified clean package versions with documented SHA-512 integrity hashes via an official GitHub security advisory and npm advisory — do not accept community-sourced version confirmations. Second, all credentials in scope during the compromise window must be provably rotated with rotation timestamps logged, and cloud/SaaS audit logs must be reviewed for the full window to rule out credential use by threat actors before builds are re-enabled. Given CISA KEV designation with a June 10, 2026 remediation deadline and active exploitation confirmed, maintain enhanced monitoring of post-recovery build environments — specifically outbound network connections from Node.js processes and access to credential files — for a minimum of 30 days post-remediation.
Forensic Artifacts	npm cache tarballs at ~/.npm/_cacache/content-v2/ (Linux) or %AppData%/npm-cache/_cacache/content-v2/ (Windows) — these contain the downloaded TanStack package files including any malicious versions and are the primary malware samples for postinstall script extraction and YARA rule development | CI/CD build console logs showing npm install stdout/stderr output including postinstall hook execution — supply chain malware in npm packages executes during the postinstall lifecycle hook, and these logs capture the execution output, any error messages from the malicious script, and the resolved package versions that were installed | Network flow logs or host-based pcap captures from build hosts filtered to the npm install execution timeframe — credential-stealing npm supply chain malware characteristically beacons to attacker-controlled HTTPS endpoints immediately after postinstall execution to exfiltrate harvested credentials from .npmrc, environment variables, and OS credential stores | File system access audit records (auditd on Linux, Windows Security Event ID 4663 on Windows) for reads against ~/.npmrc, ~/.gitconfig, /proc/*/environ, CI/CD secret environment variable files, and OS keychains during or immediately after npm install processes — these records establish which credential stores were accessed by the malicious TanStack postinstall script | Cloud provider API access logs (AWS CloudTrail, GCP Audit Logs, Azure Activity Log) scoped to the access key IDs and service account credentials that were present as environment variables in affected build pipelines during the compromise window — these logs determine whether stolen credentials were actively leveraged by threat actors for lateral movement or data access beyond the initial build environment compromise

Per-Action IR Details

Step 1: Containment — Immediately audit your dependency manifests (package.json, package-lock.json, yarn.lock) across all repositories and CI/CD pipelines for any TanStack npm packages. Freeze or pin all TanStack dependencies at their current state and block automated updates until clean versions are confirmed by TanStack maintainers. Do not install or update TanStack packages in any environment until safe versions are identified. Reference: NIST SI-3 (Malicious Code Protection), CIS 2.2 (Ensure Authorized Software is Currently Supported).

NIST Phase: Containment

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

Controls: NIST SI-3 (Malicious Code Protection), NIST CM-3 (Configuration Change Control), NIST SA-12 (Supply Chain Protection), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 2.3 (Address Unauthorized Software)

Compensating: Run `grep -r '@tanstack' --include='package*.json' --include='yarn.lock' /path/to/repos` across all repositories to enumerate exposure. Immediately set `npm config set ignore-scripts true` in all build environments to suppress postinstall hook execution — the primary mechanism by which TanStack-distributed malware executes credential theft. Use `npm pack --dry-run` on any cached TanStack package to inspect included files before any further install activity. Pin versions manually in package.json using exact version notation (no `^` or `~` prefixes) and commit the lock file.

Evidence: Before freezing dependencies, snapshot the current state: copy package.json, package-lock.json, and yarn.lock from all affected repos and CI/CD pipeline definitions to a write-once evidence store. Record the exact installed TanStack package versions and their resolved integrity hashes (`npm ls @tanstack/* --json > tanstack_inventory_$(date +%Y%m%d).json`). Preserve the npm cache directory (`~/.npm/_cacache` on Linux, `%AppData%/npm-cache` on Windows) as a forensic artifact — it contains the downloaded package tarballs including any malicious versions. Do NOT run `npm ci` or `npm install` until containment is confirmed.

Step 2: Detection — Audit build logs and npm install logs for the period during which malicious versions were available (exact window unconfirmed; treat all recent installs as suspect). Search for anomalous outbound network connections initiated during or after npm install processes — credential-stealing malware commonly beacons to external infrastructure. Review endpoint and server logs for unexpected process spawning from Node.js or npm processes. Check for access to credential stores (OS keychains, environment variable files, .npmrc, .gitconfig, CI/CD secret stores) by build tooling. Use NIST AU-6 (Audit Record Review, Analysis, and Reporting) and CIS 8.2 (Collect Audit Logs) to ensure log coverage exists. D3FEND countermeasure: D3-SFA (System File Analysis) — monitor credential files and config files for unexpected reads.

NIST Phase: Detection Analysis

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

Controls: NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-2 (Event Logging), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: On Linux CI/CD runners: query auditd logs for file reads against credential paths — `ausearch -f /root/.npmrc -f /home/*/.gitconfig -f /proc/*/environ --start recent`. On Windows build agents: query Windows Security Event Log for Event ID 4663 (file access) against `%APPDATA%\npm\` and `%USERPROFILE%\.npmrc`, and Event ID 4688 (Process Creation) filtered on `node.exe` or `npm.cmd` spawning `cmd.exe`, `powershell.exe`, or `curl.exe`. For outbound beacon detection, run `tcpdump -w tanstack_capture.pcap -i any 'port not 443 and host not registry.npmjs.org'` on build hosts during controlled reinstall testing. Deploy a Sigma rule matching `node.exe` as parent process with child `curl`, `wget`, `powershell`, or `bash` within 60 seconds of an npm install event. Review CI/CD platform logs (GitHub Actions workflow logs, Jenkins build console output, GitLab CI job traces) for unexpected curl/wget invocations in postinstall hooks.

Evidence: Capture before analysis: (1) Full npm install stdout/stderr logs from all build runs during the suspected compromise window — these will contain resolved package versions and postinstall hook execution output. (2) Network flow logs or firewall logs showing outbound connections from build hosts, filtered to the npm install execution timeframe — malicious postinstall scripts in supply chain attacks typically initiate HTTPS beaconing to attacker-controlled infrastructure immediately after execution. (3) File system access timestamps on `~/.npmrc`,

`~/.gitconfig`, CI/CD secret environment variable files, and OS credential stores (e.g., `/proc/*/environ` reads, macOS Keychain access logs via `/var/log/system.log`). (4) Process execution logs showing the full parent-child chain from the npm CLI through any spawned subprocesses during install.

Step 3: Eradication — Remove all installed TanStack package versions from build environments, artifact registries, and deployed applications until TanStack publishes verified clean versions with confirmed integrity hashes. Rotate all credentials that were present in any environment where a TanStack package was installed during the suspected compromise window: npm tokens, CI/CD secrets, cloud provider keys, API keys, and developer workstation credentials. Apply D3-CRO (Credential Rotation) across affected systems. Reference: NIST IA-5 (Authenticator Management), CIS 5.2 (Use Unique Passwords).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IA-5 (Authenticator Management), NIST AC-2 (Account Management), NIST AC-6 (Least Privilege), NIST CM-2 (Baseline Configuration), CIS 5.2 (Use Unique Passwords), CIS 5.3 (Disable Dormant Accounts)

Compensating: Purge TanStack packages from the npm cache with `npm cache clean --force` and delete node_modules directories across all affected build workspaces (`find /workspace -name 'node_modules' -type d | grep tanstack`). For internal artifact registries (Nexus, Artifactory, Verdaccio), manually delete or quarantine any cached TanStack package versions published during the compromise window. Credential rotation priority order for a 2-person team: (1) npm publish/access tokens via `npm token revoke` for all organization tokens — list with `npm token list`; (2) GitHub Actions/GitLab CI secrets via platform UI — revoke and regenerate all secrets accessible in affected pipeline environments; (3) Cloud provider IAM keys (AWS `aws iam delete-access-key`, GCP `gcloud iam service-accounts keys delete`) for any keys present as environment variables in build environments; (4) developer `.npmrc` tokens on all workstations that ran npm install during the window.

Evidence: Before eradicating, preserve: (1) Verbatim copies of the malicious package tarballs extracted from the npm cache (`~/.npm/_cacache/content-v2/`) — these are the primary malware samples needed for YARA rule development and indicator extraction. (2) A full environment variable dump from the CI/CD build context at the time of compromise (reconstruct from CI/CD platform secret audit logs if a live capture is unavailable) to enumerate which credentials were in-scope for theft. (3) Hash values (SHA-512) of all installed TanStack package files before removal, generated with `npm ls --json` and cross-referenced against the npm registry integrity field — deviations indicate tampered packages. Document the rotation event timestamps per credential for breach notification timeline construction.

Step 4: Recovery — Before reintroducing any TanStack dependency, verify package integrity using npm audit and subresource integrity checks against the official TanStack release advisory (not yet available as of this writing — monitor the TanStack GitHub and npm advisory feed). Validate that CI/CD pipeline secrets and deployment credentials have been rotated and that no unauthorized access occurred using stolen credentials. Re-enable builds only after clean-version confirmation. Monitor post-remediation builds for anomalous outbound connections. Reference: NIST SI-7 (Software, Firmware, and Information Integrity), CIS 7.3 (Perform Automated Operating System Patch Management — apply to dependency management equivalent).

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST SA-10 (Developer Configuration Management), NIST CP-10 (System Recovery and Reconstitution), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Before re-enabling any TanStack package install, verify integrity by comparing the `integrity` field from package-lock.json (sha512 hash) against the value published on the npm registry for the specific clean version: `npm view @tanstack/@ dist.integrity`. Automate this check with a pre-install script: `node -e "const p=require('./package-lock.json'); Object.entries(p.packages).filter(([k])=>k.includes('tanstack')).forEach(([k,v])=>console.log(k,v.integrity))"` and compare output against the TanStack GitHub release advisory hashes. Monitor post-recovery builds using `tcpdump`

or `ss -tulnp` on build hosts for 72 hours, alerting on any outbound connection from node.exe/npm to non-registry endpoints. Check cloud provider access logs (AWS CloudTrail, GCP Audit Logs) for any API calls made with credentials that were in-scope during the compromise window — even post-rotation, pre-rotation activity indicates credential use by threat actors.

Evidence: Before re-enabling builds, collect: (1) Audit trail of all credential rotation events with timestamps — required for breach notification scope determination if stolen credentials were used. (2) Cloud provider access logs for the compromise window (AWS CloudTrail `LookupEvents` filtered by the rotated access key IDs, GCP Admin Activity logs, Azure Activity Log) to determine whether stolen credentials were actively used by threat actors. (3) npm registry access logs if your organization uses a private registry (Verdaccio, Artifactory) — look for package publish or download events using compromised npm tokens during the window. This evidence determines whether the incident is a credential exposure or a confirmed credential compromise with downstream impact.

Step 5: Post-Incident — This incident exposes control gaps in software supply chain integrity verification. Implement or strengthen: (1) dependency pinning with hash verification (CIS 2.3 — Address Unauthorized Software); (2) CI/CD pipeline secret isolation so npm publish tokens are scoped and rotated regularly (NIST AC-6 — Least Privilege); (3) software composition analysis (SCA) tooling in CI/CD pipelines to flag newly published versions before auto-install; (4) audit logging of all package installations in build environments (NIST AU-2 — Event Logging, CIS 8.2 — Collect Audit Logs); (5) review and restrict which identities hold publish rights to your own npm packages to prevent analogous outbound risk (NIST AC-5 — Separation of Duties, NIST AC-2 — Account Management). D3FEND countermeasure: D3-FMBV (File Magic Byte Verification) — verify package integrity artifacts at install time.

NIST Phase: Post Incident

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

Controls: NIST AC-5 (Separation of Duties), NIST AC-6 (Least Privilege), NIST AC-2 (Account Management), NIST AU-2 (Event Logging), NIST SA-12 (Supply Chain Protection), NIST SI-7 (Software, Firmware, and Information Integrity), CIS 2.3 (Address Unauthorized Software), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts), CIS 8.2 (Collect Audit Logs)

Compensating: Free tooling implementations achievable by a 2-person team: (1) Add `npm ci --ignore-scripts` as the mandatory install command in all CI/CD pipelines to suppress postinstall hook execution — this alone would have blocked the credential-stealing payload in this TanStack attack. (2) Integrate `socket.dev` CLI (free tier) into the pipeline: `npx @socketsecurity/cli check` flags new package versions with supply chain risk signals before install. (3) Deploy a Sigma rule (free, community-maintained) matching npm postinstall script spawning network-capable binaries — available in the SigmaHQ repository under `process_creation` category. (4) Implement `npm audit signatures` (built into npm v8+) to verify package signatures against the npm registry's public key — `npm audit signatures` requires zero additional tooling. (5) Scope npm publish tokens to specific packages using npm granular access tokens and enforce token expiration at 90 days via `npm token create --cidr= --read-only` for CI consumers versus separate scoped publish tokens.

Evidence: Lessons-learned documentation should capture: (1) The exact TanStack package names and versions installed across the environment, mapped to which build pipelines and deployed application versions consumed them — this becomes the definitive blast radius record. (2) Whether `npm audit` or any existing SCA tooling flagged the malicious versions prior to this incident — a gap here drives the SCA tooling remediation priority. (3) Timeframe between TanStack npm advisory publication and your team's first awareness — this measures detection latency and drives investment in automated advisory feed monitoring (OSV.dev API, GitHub Dependabot alerts, or the npm security advisory RSS feed). Retain all forensic artifacts for a minimum of 12 months given CISA KEV designation, as regulatory or legal discovery is elevated for known-exploited vulnerabilities.

Detection Guidance

Primary detection focus is anomalous activity during or after npm install/build processes. Key signals to query: (1) Outbound network connections originating from node, npm, or build runner processes to non-registry IP

addresses or domains, especially short-lived connections during install phases. (2) File system access to credential stores from build processes: ~/.npmrc, ~/.gitconfig, ~/.aws/credentials, environment variable dumps, OS keychain access on developer workstations. (3) Unexpected process spawning as child processes of npm or node during package installation. (4) CI/CD audit logs showing secret or environment variable access by build steps that do not normally require them. (5) npm audit output flagging CVE-2026-45321 in installed packages. In SIEM, build a query joining process creation events (parent: node/npm) with network connection events and file access events against credential paths, flag any combination for immediate triage. D3FEND countermeasures applicable: D3-SFA (System File Analysis), D3-LAM (Local Account Monitoring). Specific IOC patterns are not confirmed in available source data; behavioral detection is the primary viable approach until the vendor or a threat research team publishes confirmed indicators.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://www.cisa.gov/known-exploited-vulnerabilities-catalog`	CISA KEV entry confirming active exploitation of CVE-2026-45321	HIGH
URL	`https://nvd.nist.gov/vuln/detail/CVE-2026-45321`	NVD vulnerability record for CVE-2026-45321	HIGH

Framework Mappings

MITRE-ATTACK

- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1078** — Valid Accounts
- **T1555** — Credentials from Password Stores

NIST-800-53R5

- **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
- **SR-2** — Supply Chain Risk Management Plan
- **IR-5** — Incident Monitoring

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **2.5** — Allowlist Authorized Software

- **2.6** — Allowlist Authorized Libraries
- **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.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program
- **DE.AE-08** — Incidents are declared when adverse events meet the defined incident criteria

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1078	Valid Accounts	Defense-Evasion
T1555	Credentials from Password Stores	Credential-Access

Sources

Source	URL	Tier
cisa_kev	https://www.cisa.gov/known-exploited-vulnerabilities-catalog	T1
CVE-2026-45321 Details - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-45321	T1
CVE-2026-45321: Tanstack arktype-adapter Auth Bypass Flaw	https://www.sentinelone.com/vulnerability-database/cve-2026-45321/	T3
CVE-2026-45321 - CVE Record	https://www.cve.org/CVERecord?id=CVE-2026-45321	T3

Source	URL	Tier
CVE-2026-45321 Common Vulnerabilities and Exposures | SUSE	https://www.suse.com/security/cve/CVE-2026-45321.html	**T3**

Generated 2026-05-27 18:54 UTC by TJS Security Command Center