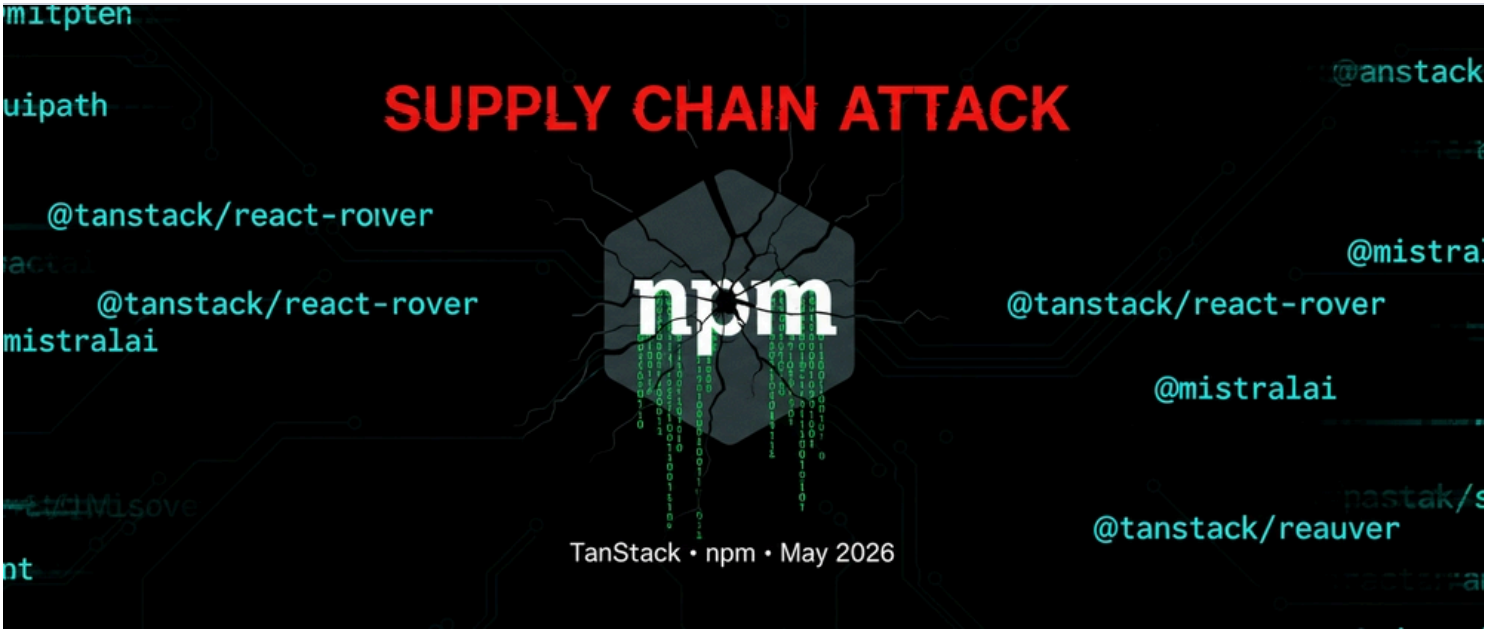

VIKAS
Posted on May 13

42 @tanstack/* Packages Were Compromised on npm: What Happened, How It Works, and What You Must Do Right Now

#javascript #security #webdev #npm

> **TL;DR:** On May 11, 2026, 84 malicious versions across 42 `@tanstack/*` packages were live on npm for ~20 minutes. No npm tokens were stolen. No accounts were compromised. The attacker hijacked TanStack's own GitHub Actions release pipeline using three chained vulnerabilities and published malware through the project's trusted OIDC identity. If your CI ran `npm install` on May 11, treat every secret it had access to as stolen.

The Router Package Powering 12 Million Projects Just Got Weaponized

`@tanstack/react-router` has over **12.7 million weekly downloads**. It's in production React apps across the world, from solo founders shipping MVPs to enterprise teams running critical infrastructure. On the evening of May 11, 2026, it became a credential-stealing worm.

This was not a theoretical supply chain risk. This was live malware, published through a **cryptographically valid, SLSA-attested release pipeline**, that ran silently in CI environments and self-propagated to every other npm package its victims maintained.

The most chilling part? Every security signal said the packages were clean. **Valid Sigstore provenance. SLSA Build Level 3 attestation. 2FA on every maintainer. OIDC trusted publishing. All green. All meaningless.**

This article covers exactly what happened, how the attack technically worked, how to know if you're affected, and what you permanently change in your workflow after this.

What Happened: The Attack Timeline

Timestamp (UTC)	Event
May 11 · ~19:10	Attacker's fork PR triggers `pull_request_target` workflow, cache poisoned
May 11 · 19:20	First malicious `@tanstack/*` publish hits npm registry
May 11 · 19:26	All 84 malicious artifacts across 42 packages published in 6 minutes
May 11 · ~19:40	External researcher `ashishkurmi` (StepSecurity) opens GitHub issue #7383 with full analysis
May 11 · 19:46	TanStack team starts war room. Socket.dev calls Tanner Linsley directly
May 11 · ~20:30	All affected versions deprecated. npm security engaged to pull tarballs

Timestamp (UTC)	Event
May 12 · 00:00+	Blast radius expands: `@mistralai`, `@uipath`, `@opensearch-project`, PyPI packages hit
May 12	TanStack publishes full postmortem at tanstack.com/blog

The window was **20 minutes** from first publish to public disclosure. For 12 million weekly downloads, 20 minutes is a long time.

The Attack Chain: Three Vulnerabilities, One Kill Path

No npm credentials were stolen. No maintainer accounts were phished. The attacker didn't need any of that. They chained three well-documented GitHub Actions weaknesses that TanStack's own workflow was already vulnerable to.

Each vulnerability alone is survivable. Together, they form a complete kill path from "random fork PR" to "published malicious packages with valid provenance."

Vulnerability 1: The Pwn Request (`pull_request_target`)

The attacker created a fork of `TanStack/router`, renamed it to `zblgg/configuration` to avoid appearing in the fork list, and opened a pull request. This triggered TanStack's `bundle-size.yml` workflow, a workflow that ran on `pull_request_target`.

`pull_request_target` is a special GitHub Actions event. Unlike `pull_request`, it runs in the context of the **base repository**, not the fork. That means it has write access to the base repo's cache, artifacts, and secrets, even when triggered by a completely untrusted fork PR.

GitHub's own security team documented this as a dangerous pattern in 2023. TanStack had it in production.

```
# TanStack's bundle-size.yml (simplified)
on:
  pull_request_target:     # ← Runs in BASE repo context. Fork code gets write a
    types: [opened, synchronize]
```

```
jobs:
  build:
    steps:
      - uses: actions/checkout@v4
        with:
          ref: ${{ github.event.pull_request.merge_commit_sha }}  # ← Checks out
      - run: pnpm install && pnpm build                           # ← Executes
```

The fork code ran inside the base repo's workflow context. That's the trust boundary violation.

Vulnerability 2: GitHub Actions Cache Poisoning

The attacker's fork code wrote a malicious pnpm store into the GitHub Actions cache. Cache keys in GitHub Actions are scoped to the branch, but `pull_request_target` runs in the base repo context, which means the attacker's cache entry was written with base repo credentials and was later readable by the base repo's release workflow.

When a legitimate TanStack maintainer merged a real PR to `main`, the release workflow ran and restored the cache, pulling the attacker's poisoned pnpm store instead of the legitimate one.

Attacker-controlled binaries were now inside the release pipeline's `node_modules`. The release workflow had no way to distinguish them from legitimate packages.

Vulnerability 3: OIDC Token Extraction from Runner Memory

With code executing inside the release workflow's runtime, the payload read the GitHub Actions runner's process memory directly:

```
# The payload read /proc/<pid>/mem to extract the OIDC token
# Specifically targeting: ACTIONS_ID_TOKEN_REQUEST_TOKEN and ACTIONS_ID_TOKEN_RE
# It deliberately skipped tokens named GITHUB_TOKEN to avoid GitHub's own secret
```

The OIDC token was extracted live from runner memory, not from environment variables, not from workflow outputs. Standard secret masking doesn't protect against memory reads.

With a valid OIDC token, the attacker called the npm OIDC trusted publisher endpoint directly and minted a publish-capable credential scoped to `@tanstack/*`. This is the

same token the legitimate release workflow uses. npm had no way to distinguish them.

84 malicious versions across 42 packages published. Every one carried a valid SLSA Build Level 3 provenance attestation because it was published through the legitimate Sigstore stack with a legitimate OIDC token.

This is the first documented npm supply chain attack where malicious packages are cryptographically indistinguishable from legitimate ones by provenance attestation.

What the Payload Does

The malware is a credential-stealing worm. Not a backdoor. Not a data exfiltrator. A **worm** that actively spreads to other packages you maintain.

Step 1: Daemonize and disappear

```
// First thing the payload checks
if (!process.env.__DAEMONIZED) {
  // Fork a fully detached child with all stdio suppressed
  // Call unref() so Node.js doesn't wait for it
  // Parent exits cleanly. Your terminal looks fine
  // Child runs silently, decoupled from your session
}
```

Your `npm install` completes normally. No errors. No warnings. A detached daemon is now running in the background.

Step 2: Credential sweep

The payload performs a systematic sweep of every credential plane common in CI environments:

Target	Method
GitHub Actions OIDC	`ACTIONS_ID_TOKEN_REQUEST_TOKEN` + `ACTIONS_ID_TOKEN_REQUEST_URL`
AWS	IMDSv2 metadata endpoint

Target	Method
GCP	Metadata server
Azure	Instance metadata
Kubernetes	Service account token at `/var/run/secrets/kubernetes.io/serviceaccount/token`
HashiCorp Vault	`VAULT_TOKEN` environment variable
npm tokens	`.npmrc` and environment
SSH keys	`~/.ssh/` sweep

Notably, the payload **deliberately skips tokens named `github_token`**, likely to avoid triggering GitHub's own secret scanning on the exfiltrated data.

Step 3: Exfiltrate over encrypted channels

Harvested credentials are sent over the **Session/Oxen messenger file-upload network**, end-to-end encrypted with no attacker-controlled C2 server in the traditional sense. Blocking by IP or domain is the only network-layer mitigation.

A secondary exfiltration channel was also identified: a typosquat domain `git-tanstack[.]com`.

Step 4: Self-propagate as a worm

```
GET https://registry.npmjs.org/-/v1/search?text=maintainer:<your-username>
```

The payload enumerates every npm package the victim maintains, then republishes each one with the same malware injection using the stolen npm OIDC token. The packages it publishes also carry valid provenance attestations.

This is how the blast radius expanded from TanStack to `@mistralai`, `@uipath`, `@opensearch-project`, and 160+ other packages within hours, all through automated self-propagation.

Confirmed Safe Packages

Not all `@tanstack/*` packages were affected. The compromise targeted the router family as the initial vector.

Confirmed clean:

- `@tanstack/query-*`
- `@tanstack/table-*`
- `@tanstack/form-*`
- `@tanstack/virtual-*`
- `@tanstack/store`

The full list of 42 compromised packages is in GHSA-g7cv-rxg3-hmpx.

Am I Affected? Check Right Now

Step 1: Check if you installed on the affected date

```
# Check npm install logs for May 11, 2026 between 19:20–19:30 UTC
# If your CI ran during this window and included any @tanstack/* package, assume

# Check your package-lock.json for affected version ranges
grep -A1 '"@tanstack/' package-lock.json | grep '"version"'
```

Step 2: Check for network IOCs in CI logs

```
# Look for outbound connections to these in your pipeline logs
# filev2.getsession.org
# seed1.getsession.org / seed2.getsession.org / seed3.getsession.org
# git-tanstack.com
```

Step 3: Check for unauthorized npm publishes

```
# Run this for every package you maintain
npm access list packages <your-org>

# Check for versions published on May 11 that you didn't author
npm view <your-package> time --json | grep "2026-05-11"
```

Step 4: Audit OIDC tokens (but read this first)

```
npm token list
# Look for tokens you don't recognize
# CRITICAL: Do NOT revoke tokens before isolating and imaging the system
# Revoking first destroys forensic evidence of which packages were republished
```

> **If you ran npm install against any `@tanstack/*` package on May 11 between 19:20 and 19:30 UTC, treat the entire install host as compromised. Every secret accessible to that process needs rotation.**

Remediation: Step by Step

1. Upgrade to clean versions

```
# Safe versions are anything published AFTER May 12, 2026
# Check tanstack.com/blog/npm-supply-chain-compromise-postmortem for the verifie

npm install @tanstack/react-router@latest   # Verify version > May 12 publish da
```

2. Lock all @tanstack/* packages and verify integrity

```
{
  "overrides": {
    "@tanstack/react-router": "1.x.x",
    "@tanstack/router": "1.x.x"
  }
}


# Verify lockfile integrity hashes match registry
npm ci --ignore-scripts
```

3. Add --ignore-scripts to CI permanently

```
# In every CI pipeline, add this now
npm ci --ignore-scripts
```

This prevents ALL `postinstall`, `preinstall`, and lifecycle scripts from running during automated builds. It eliminates the entire attack surface this class of exploit depends on.

4. If your CI ran during the affected window: full incident response

Do not attempt to clean in place. Rebuild from a known-good snapshot and rotate every credential the runner had access to:

- npm publish tokens
- AWS / GCP / Azure / cloud credentials
- GitHub personal access tokens and OIDC bindings
- Kubernetes service account tokens
- HashiCorp Vault tokens
- SSH private keys
- All `.env` and CI/CD secrets

```
# Review cloud audit logs for activity from the affected host
# AWS CloudTrail, GCP Audit Logs, Azure Activity Log
# Look for API calls originating from your CI runner IP on May 11 after 19:20 U
```

5. Block exfiltration endpoints at the network level

```
# Block these domains at firewall / DNS / hosts level
filev2.getsession.org
seed1.getsession.org
seed2.getsession.org
seed3.getsession.org
git-tanstack.com
```

Indicators of Compromise (IOCs)

Malicious npm Packages (partial list)

Package	Malicious Versions
`@tanstack/react-router`	Check GHSA-g7cv-rxg3-hmpx for full version list
`@tanstack/router`	Check GHSA-g7cv-rxg3-hmpx
`@mistralai/mistralai`	MAL-2026-3432 (OSSF)

Package	Malicious Versions
`@uipath/apollo-core`	Check Wiz advisory
`guardrails-ai` (PyPI)	Check Wiz advisory

Full 42-package list: GHSA-g7cv-rxg3-hmpx

Network IOCs

Indicator	Notes
`filev2.getsession.org`	Primary exfil: Session/Oxen messenger network
`seed1/2/3.getsession.org`	Redundant exfil endpoints
`git-tanstack[.]com`	Typosquat secondary exfil channel

Behavioral IOCs

Indicator	What It Means
Detached node process spawned during `npm install`	Daemon forked to avoid terminal association
Outbound HTTPS to `*.getsession.org` from CI runner	Credential exfiltration in progress
Unexpected `npm publish` events in pipeline logs	Worm self-propagation via stolen OIDC token
New versions of your packages published on May 11	Your packages may be compromised
400+ GitHub repos containing "Shai-Hulud: Here We Go Again"	Attribution marker left by TeamPCP

CVE

CVE-2026-45321 | CVSS 9.6 (Critical) | GHSA-g7cv-rxg3-hmpx

What This Attack Breaks About Everything We Thought Was Safe

This incident isn't just a TanStack story. It's a proof of concept that invalidates assumptions the entire ecosystem was operating on.

SLSA provenance does not mean safe

TanStack had SLSA Build Level 3 provenance on every release. The malicious packages also had SLSA Build Level 3 provenance, because they were built by the same pipeline. Provenance confirms which pipeline produced the artifact. It cannot tell you whether that pipeline was behaving as intended.

> "SLSA provenance confirms which pipeline produced the artifact, not whether the pipeline was behaving as intended. A compromised build step can produce a validly-attested but malicious package." (StepSecurity)

This is the first documented supply chain attack where provenance attestation is completely useless as a trust signal.

OIDC trusted publishing is not enough alone

OIDC publishing was designed to eliminate long-lived publish tokens. It succeeded at that. But it introduced a new assumption: that the pipeline itself cannot be compromised. This attack proved that assumption wrong.

OIDC trusted publishing needs a second layer: per-publish review gates, or provenance-source verification to detect publishes from unexpected workflow steps.

`pull_request_target` is still widespread and still dangerous

This pattern has been publicly flagged as dangerous for over three years. It still appears in thousands of popular repositories' CI configurations. Check yours today:

```
# Search your .github/workflows/ for this pattern
grep -r "pull_request_target" .github/workflows/
```

If you find it, audit carefully. `pull_request_target` is fine for trusted operations like labeling or commenting. It should never check out and execute fork code.

The worm model changes the blast radius math

Previous supply chain attacks were static. Compromise one package, infect its users. The Mini Shai-Hulud worm is dynamic. Compromise one package, infect its users, then use their credentials to infect every package they maintain, which infects their users, and so on.

The 42 initial TanStack packages became 170+ packages within hours. This is not a linear blast radius. It's exponential.

Quick Reference Checklist

```
- [ ] Checked if CI ran npm install against @tanstack/* on May 11 between 19:20
- [ ] Checked pipeline logs for outbound connections to getsession.org / git-tar
- [ ] Checked for unauthorized npm publishes from your account on May 11
- [ ] Upgraded to clean @tanstack/* versions (published after May 12)
- [ ] Added --ignore-scripts to all CI pipelines
- [ ] Committed and locked package-lock.json / pnpm-lock.yaml
- [ ] Rotated all credentials if install ran during affected window
- [ ] Blocked IOC domains at network / DNS level
- [ ] Audited .github/workflows/ for pull_request_target + checkout + run patter
- [ ] Reviewed OIDC trusted publisher bindings for unexpected workflow steps
```

Resources

- TanStack Official Postmortem
- TanStack Hardening Followup
- GitHub Advisory GHSA-g7cv-rxg3-hmpx
- StepSecurity Full Technical Analysis
- Snyk Deep Dive
- Wiz Attribution Report
- Socket Research
- GitHub Issue #7383: Original Disclosure

Sourced from TanStack's official postmortem, StepSecurity's technical analysis, Snyk, Wiz, and Socket Research. All IOCs, timeline, and technical details drawn directly from primary source investigations.

If this helped you secure your pipeline, drop a ❤️ and share it with your team. The faster it spreads, the fewer engineers get hit.

Top comments (0)

Code of Conduct • Report abuse

VIKAS

Future full-stack wizard | Ctrl+Z enthusiast | I promise my code runs on my machine | Builder of bugs disguised as features 🐛➡️✨ | Trying to make the web less broken, one div at a time 💻⏳.

LOCATION
New Delhi, India

EDUCATION
ITESM, DSEU Rajokri 🎓 | Turning diploma knowledge into building and negotiating peace with bugs 🐛

PRONOUNS
Me

WORK
Full-Stack Developer delivering scalable, production-ready web applications

JOINED
Aug 20, 2022

More from VIKAS

GitHub Copilot is Training on Your Code; Opt Out Before April 24 or Lose the Choice

#security #github #webdev #javascript

My Pre-Launch Checklist Before Handing Off Any Client We

#webdev #nextjs #javascript #productivity

An AI Found a 27-Year-Old Bug in OpenBSD- The Most Security-Hardened OS on Earth

#security #openbsd #ai #cybersecurity