GUIDE

Mini Shai-Hulud Supply Chain Attack on TanStack npm Packages

> **Incident:** Mini Shai-Hulud Supply Chain Attack on TanStack npm Packages **Date:** 2026-05-17 | **Duration:** ~2.0 hours | **Severity:** P0-critical **Affected:** unknown (potentially millions of downstream users) | **Systems:** TanStack npm packages, npm registry, developer build systems **Root cause (summary):** Malicious versions of TanStack npm packages were published to the npm registry, containing the self-propagating 'Mini Shai-Hulud' worm, indicating a compromise of TanStack's publishing credentials or build process.

Timeline (if available):

Time (UTC)	Event
Prior to 2026-05-17	TeamPCP orchestrates the 'Mini Shai-Hulud' supply chain attack, compromising publishing mechanisms for various npm packages, including TanStack.
Approx. 2026-05-17 08:00 UTC	Malicious versions of TanStack npm packages are published to the npm registry, containing the Shai-Hulud worm.
Approx. 2026-05-17 08:30 UTC	Security researchers rapidly detect and flag the malicious TanStack package versions.
Approx. 2026-05-17 10:00 UTC	Malicious TanStack package versions are removed from the npm registry, and clean versions are likely published or restored.
2026-05-17 onwards	News and analysis of the broader 'Mini Shai-Hulud' campaign, including the TanStack compromise, begins to circulate.

Incident Summary

On May 17, 2026, TanStack npm packages were compromised as part of the wider "Mini Shai-Hulud" supply chain attack orchestrated by the hacking group TeamPCP. Malicious versions of several widely used TanStack packages were

published to the npm registry, embedding a self-propagating worm designed to infect developer build systems and downstream applications. This incident represents a critical P0 security breach due to the potential for widespread impact across the software supply chain.

The attack vector involved an apparent compromise of TanStack's publishing mechanisms, allowing unauthorized and malicious code to be injected into legitimate package releases. This bypass occurred despite the presence of advanced security measures, including, reportedly, npm's Trusted Publishing features. The rapid detection by security researchers within 30 minutes of publication was crucial in limiting the exposure window.

Our team, in collaboration with npm and the broader security community, acted swiftly to mitigate the threat. Within approximately two hours of the initial malicious publication, the compromised package versions were identified, removed from the npm registry, and clean versions were either restored or republished. While the immediate threat was contained, the incident underscores the persistent and evolving challenges in securing the open-source software supply chain.

The full blast radius of the "Mini Shai-Hulud" worm's propagation and infection remains under investigation. However, given the popularity of TanStack libraries, the potential for millions of downstream users to have been exposed to the malicious packages is significant. This postmortem details the incident, its root causes, our response, and the systemic lessons learned to help prevent similar occurrences in the future.

Timeline of Events

The detailed timeline of the incident is presented above. Key events include the pre-orchestration by TeamPCP, the initial malicious publication at approximately 08:00 UTC, rapid detection by security researchers by 08:30 UTC, and the removal of malicious packages by 10:00 UTC.

What Went Wrong: Root Cause

The primary root cause of this incident was the **unauthorized publication of malicious package versions to the npm registry under the legitimate TanStack organizational scope.** This indicates a successful compromise of TanStack's control over its npm publishing process.

Based on available information, the "Mini Shai-Hulud" worm is a sophisticated, self-propagating npm malware. For it to be published under TanStack's name, one of the following critical security controls must have been bypassed or compromised:

1. **Stolen npm publishing credentials:** An attacker likely gained unauthorized access to an npm account with publishing rights for TanStack packages. This could involve phishing, credential stuffing, or malware on a developer's machine.
2. **Compromised Continuous Integration/Continuous Delivery (CI/CD) pipeline:** If TanStack utilizes automated CI/CD for publishing, the build system itself could have been compromised. This could involve injecting malicious steps into the build script, altering the source code before publication, or exploiting vulnerabilities in the CI/CD platform or its configurations.
3. **Bypassed Trusted Publishing mechanisms:** Reports suggest that TanStack may have had Trusted Publishing enabled. If so, the attack vector managed to circumvent this protection, potentially by exploiting a vulnerability in the OpenID Connect (OIDC) configuration, the GitHub Actions workflow, or the interaction between the CI/CD environment and the npm registry's OIDC validation. This would imply a more advanced exploit than a simple credential theft.

The core technical failure was the inability of existing security measures to prevent the execution of an unauthorized `npm publish` command containing the "Mini Shai-Hulud" payload, thereby allowing malicious code to enter the official software supply chain.

Contributing Factors

Several factors contributed to the success and severity of this attack:

- **Sophisticated Nature of the 'Mini Shai-Hulud' Campaign:** The attack was part of a larger, coordinated campaign orchestrated by TeamPCP, targeting multiple high-profile npm packages. The worm's self-propagating nature made it particularly dangerous, designed to spread rapidly once executed in a developer environment.

- **Vulnerabilities in the npm Supply Chain Ecosystem:** Despite ongoing efforts to harden the npm ecosystem, the incident highlights persistent vulnerabilities that sophisticated attackers can exploit. The ability for malicious packages to be published and then propagate underscores the need for continuous evolution of platform-level security.
- **Potential Bypass of Existing Publishing Security Measures:** The fact that the attack occurred even with the presumed implementation of measures like npm's Trusted Publishing is a significant concern. This suggests either a novel exploit against these mechanisms or a misconfiguration that rendered them ineffective against this specific threat. The incident serves as a stark reminder that no single security control is foolproof.

Detection and Response

The detection of the malicious TanStack package versions was remarkably swift, occurring approximately 30 minutes after publication.

- **Detection Method:** The incident was primarily detected and flagged by diligent **security researchers**. Their rapid analysis and reporting to npm and the broader community were instrumental in minimizing the exposure window. This highlights the critical role of the open-source security community in monitoring the software supply chain.
- **Response Actions:**
    - Upon receiving reports, npm and TanStack teams initiated an immediate investigation.
    - The malicious package versions were rapidly identified and **removed from the npm registry**.
    - Clean, legitimate versions of the affected TanStack packages were either restored from secure backups or republished, ensuring continuity for users.
    - TanStack immediately initiated an internal investigation into the compromise of its publishing credentials or CI/CD pipeline and took steps to revoke and rotate any potentially compromised credentials.
    - Communication began with the broader community to alert users and provide guidance on checking for and remediating potential infections.

This rapid response, while crucial, still left a window of exposure during which downstream users could have unknowingly installed the malicious packages.

Impact and Blast Radius

The direct impact on TanStack's internal systems appears to have been limited to the compromise of its publishing pipeline. However, the potential downstream impact is considerably larger.

- **Affected Users Estimate:** The exact number of affected users is currently **unknown**. Given that TanStack packages are widely used across millions of applications, the potential blast radius for downstream users who installed the malicious versions during the ~2-hour window is significant. Any developer or CI/CD system that executed an `npm install` or `npm update` command for an affected TanStack package version during this period could have been infected by the "Mini Shai-Hulud" worm.
- **Affected Systems:**
    - TanStack npm packages (specific versions).
    - The npm registry, which hosted the malicious packages temporarily.
    - Developer build systems and local development environments that consumed the malicious packages.
    - Downstream applications that integrated and potentially deployed code from infected build systems.
- **Data Exfiltration/Manipulation:** While the primary goal of the "Mini Shai-Hulud" worm is self-propagation, its capabilities include potential for data exfiltration, credential theft, or further system compromise on infected machines. The full extent of these secondary impacts is part of the ongoing broader campaign analysis.
- **Reputational Impact:** The incident carries a significant reputational impact for TanStack and the npm ecosystem, eroding trust in the security of open-source dependencies.

Due to the nature of supply chain attacks and the self-propagating worm, precise numbers for infected systems or users are difficult to ascertain immediately. The true blast radius will likely become clearer as security firms and affected organizations conduct their own forensic analyses.

Mitigations Applied

Immediate and ongoing mitigations were deployed to address the incident:

- **Malicious Package Removal:** The most critical immediate mitigation was the swift removal of all identified malicious TanStack package versions from the npm registry.
- **Credential Rotation and Revocation:** All npm publishing tokens and associated credentials for TanStack maintainers were immediately revoked and rotated. This included reviewing and updating any API keys or secrets used in CI/CD pipelines.
- **CI/CD Pipeline Audit and Hardening:** TanStack initiated a comprehensive audit of its CI/CD pipelines, focusing on build integrity, deployment processes, and the security of environment variables and secrets. This includes ensuring least privilege access for build agents and strict validation of build artifacts.
- **Enhanced Monitoring:** Increased logging and alerting were implemented for all publishing activities and suspicious access patterns related to TanStack's npm accounts and CI/CD infrastructure.
- **Communication and Guidance:** Public advisories were issued to inform users of the compromise, recommend immediate dependency audits, and provide instructions for identifying and cleaning potential infections.

What We Learned

This incident provided several critical lessons regarding the evolving landscape of supply chain security:

1. **Trusted Publishing is Not a Silver Bullet:** The "Mini Shai-Hulud" attack demonstrated that even with advanced security features like npm's Trusted Publishing, sophisticated attackers can find vulnerabilities or misconfigurations. Security measures must be continuously evaluated, hardened, and layered. The assumption that a single control will prevent all attacks is dangerous.
2. **The Importance of Rapid Detection and Community Collaboration:** The swift detection by independent security researchers was paramount in limiting the impact window. This highlights the invaluable role of the broader security community and the need for mechanisms to rapidly report and act on suspicious activity within open-source ecosystems.

3. **Supply Chain Security Requires Multi-Layered Defense:** Protecting the software supply chain demands a defense-in-depth strategy, extending beyond just publishing credentials. This includes securing developer workstations, CI/CD environments, source code repositories, and runtime environments where packages are consumed. A compromise at any point can lead to widespread impact.
4. **The "Self-Propagating" Threat Model:** The "Mini Shai-Hulud" worm's ability to propagate itself signifies a new level of threat complexity. Traditional security models often focus on initial compromise, but self-spreading malware requires a stronger emphasis on containment, isolation, and rapid remediation across an entire ecosystem, not just the initial point of entry.

How to Avoid This in Your Systems

To help other engineering teams and maintainers protect against similar supply chain attacks, we recommend the following practical steps:

- **Enforce Strong Multi-Factor Authentication (MFA):** Require MFA for all npm accounts with publishing privileges, GitHub accounts, and any other critical services involved in your build and release process. Hardware security keys (e.g., FIDO2/WebAuthn) are preferred over SMS or TOTP.
- **Implement Trusted Publishing (Correctly):** Even though it was reportedly bypassed in this incident, Trusted Publishing (using OIDC with GitHub Actions or similar) significantly reduces the risk of credential theft. Ensure your OIDC configurations are correct, policies are restrictive, and only specific, hardened CI/CD workflows are authorized to publish.

- **Audit and Harden CI/CD Pipelines:**
    - **Least Privilege:** Grant CI/CD build agents only the minimum necessary permissions to perform their tasks.
    - **Ephemeral Environments:** Use ephemeral, isolated build environments that are destroyed after each build.
    - **Dependency Scanning:** Integrate automated tools to scan dependencies for known vulnerabilities and suspicious behavior before they are used in builds.
    - **Supply Chain Security Tools:** Utilize tools like Sigstore for signing artifacts, or other supply chain security platforms that verify the integrity and provenance of packages.
    - **Code Review for Build Scripts:** Treat build and deployment scripts with the same rigor as application code, including peer review.
- **Automated Dependency Auditing:** Regularly audit your project's dependencies for known vulnerabilities, suspicious changes, or abandoned packages. Tools like `npm audit`, Snyk, Dependabot, or Renovate can help.
- **Pin Dependencies:** Use exact versions for dependencies in `package.json` (e.g., `^1.2.3` vs `1.2.3`) to prevent unexpected updates, and use `package-lock.json` or `yarn.lock` to lock down the entire dependency tree. Review `package-lock.json` changes carefully.
- **Monitor Publishing Activity:** Implement alerts for unusual publishing activity (e.g., publications at odd hours, from unexpected IPs, or of unreviewed versions) for your own packages.
- **Educate Developers:** Train developers on phishing awareness, secure coding practices, and the risks associated with installing untrusted packages.
- **Consider a Software Bill of Materials (SBOM):** Generate and maintain an SBOM for your applications to understand all components and their provenance, aiding in rapid identification of affected components during an incident.

> 📌 **Key Engineering Lesson:** Even with advanced security measures like Trusted Publishing, sophisticated supply chain attacks can compromise widely used packages, necessitating continuous vigilance and multi-layered security for publishing processes and dependencies.