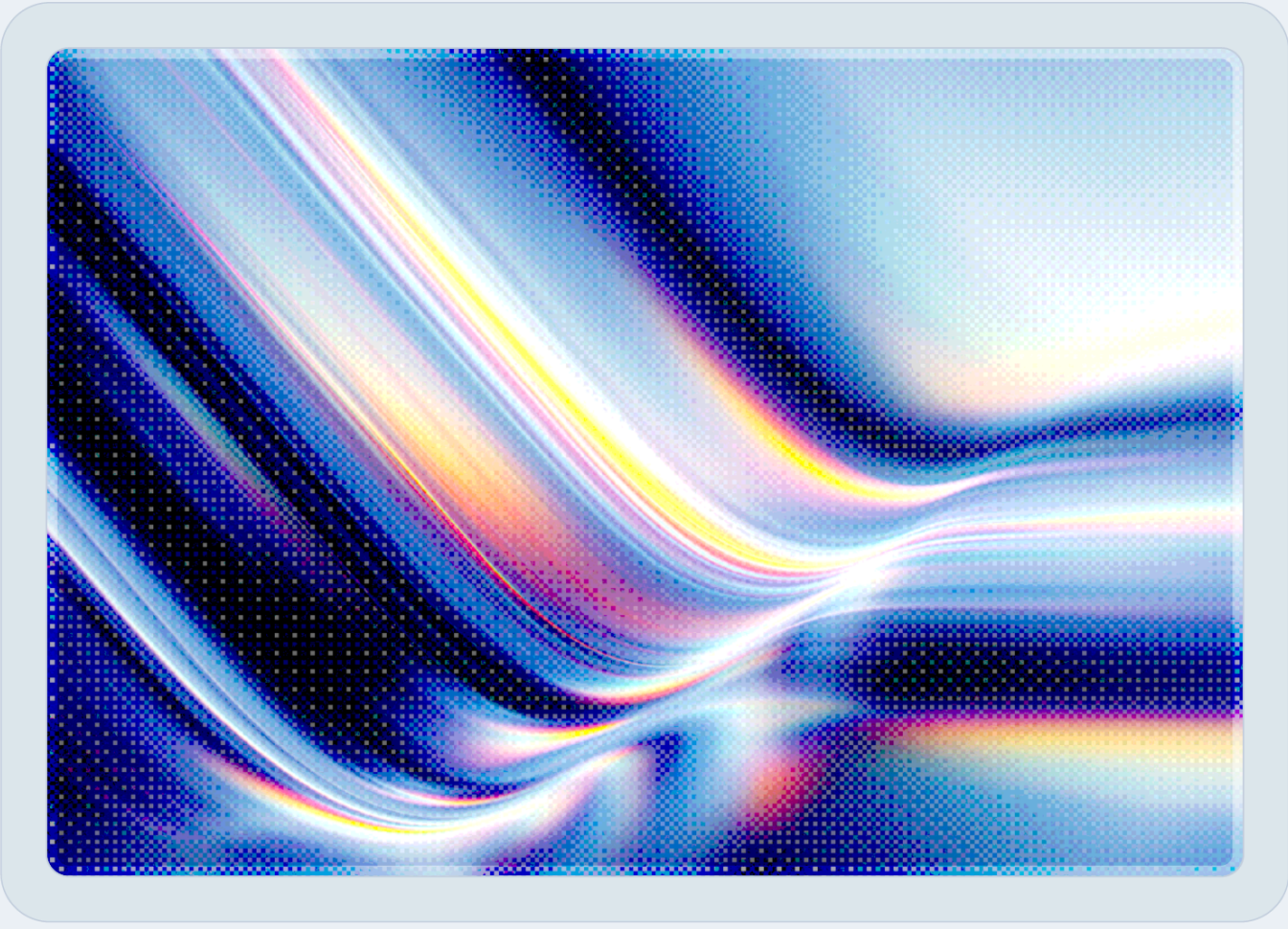
CSAI Foundation | Cloud Security Alliance

Mini Shai-Hulud: AI Developer Supply Chain Compromise

TeamPCP's Supply Chain Worm Weaponizes AI Coding Agents and CI/CD Pipelines

2026-05-16

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- Between April 29 and May 12, 2026, the threat actor TeamPCP exploited compromised developer credentials and CI/CD pipeline misconfigurations to inject malicious code into more than 170 npm and PyPI packages, including widely used AI developer libraries from TanStack, Mistral AI, and UiPath [1].
- The campaign introduced a persistence vector not previously documented in supply chain attacks of this class: injection of malicious execution hooks into Claude Code's `.claude/settings.json` and VS Code's `.vscode/tasks.json` configuration files, ensuring payload re-execution on every developer session even after the underlying infected package is removed [2].
- Attackers exploited GitHub OIDC trusted publishing and Actions cache poisoning to publish cryptographically signed malicious releases without directly stealing maintainer credentials, bypassing multi-factor authentication protections [3].
- Credential theft targeted 134 distinct file paths per compromised host, harvesting cloud provider tokens (AWS, GCP, Azure), Kubernetes service account tokens, HashiCorp Vault secrets, LLM API keys, and password vault credentials [2].
- At the time of initial reporting, no CVE, GitHub Security Advisory (GHSA), or Open Source Vulnerability (OSV) record had been assigned to any Mini Shai-Hulud-related package compromise, exposing a structural gap in vulnerability-management tooling for credential-theft-based supply chain attacks [2]; a CVE was subsequently assigned for a component of the campaign [10].

Background

Software supply chain attacks against the open-source ecosystem grew 73% year-over-year in 2025 [4], reflecting both greater attacker interest and an expanding attack surface as AI-assisted development tools proliferate and create new trust boundaries in the developer workflow. This growth has been driven

by the increasing density of developer tooling, the automated nature of modern CI/CD pipelines, and the relatively low barrier to publishing malicious package versions once any credential in a release chain is compromised.

Mini Shai-Hulud takes its name from a fictional worm species in Frank Herbert's *Dune* series—a name chosen by security researchers to describe an attack that burrows through interconnected developer ecosystems and resurfaces in unexpected locations. The designation builds on the naming convention of the September 2025 "Shai-Hulud" campaign, also attributed to TeamPCP, which first demonstrated that stolen npm tokens could propagate malware across hundreds of packages within hours [1]. The April–May 2026 campaign represents the third iteration of this worm family, and the first to deploy AI coding agent persistence mechanisms—targeting libraries central to AI application development with a degree of sophistication that prior iterations did not exhibit.

TeamPCP is a cloud-focused cybercriminal group that emerged in late 2025, specializing in automating supply chain attacks against cloud-native infrastructure including Docker, Kubernetes, and CI/CD environments [1]. Prior to Mini Shai-Hulud, the group conducted a systematic campaign against widely trusted open-source security tools: the Aqua Security vulnerability scanner Trivy was compromised on March 20, 2026 [5, 11]; the Checkmarx KICS infrastructure-as-code scanner on March 23; the AI gateway LiteLLM on March 24 [5]; the Telnyx Python SDK on March 27; and the Bitwarden CLI on April 23 [5]. Unit 42 assessed with high confidence that the same operators were responsible for all phases of the broader campaign [6]. The Vect ransomware group began publishing identified victims on April 15, 2026 with data attributed to TeamPCP-stolen credentials, confirming active monetization of harvested access within weeks of collection [6].

Security Analysis

Attack Chain

Mini Shai-Hulud proceeds through a six-stage exploitation chain that begins with credential acquisition and culminates in self-propagating worm behavior across the developer ecosystem.

Credential acquisition occurred through two distinct methods across the campaign's waves. In the earlier Trivy and KICS phases, attackers exploited incomplete credential rotation following prior compromises, using stolen GitHub Personal Access Tokens to force-push malicious commits across all version tags of target repositories. In the May 2026 TanStack campaign, the approach was considerably more sophisticated: the attackers created a repository fork, opened a pull request that triggered a permissive `pull_request_target` GitHub Actions workflow, and poisoned the GitHub Actions

cache with a malicious pnpm package store. From the compromised runner, they extracted OIDC tokens directly from the GitHub Actions runner process memory by scanning `/proc/*/mem` for live `Runner.Worker` processes, bypassing GitHub's standard secret masking controls entirely [3]. The result was publication of malicious package versions carrying cryptographically valid provenance signatures—without the attackers ever directly compromising a developer account.

Payload delivery relied on npm's lifecycle script mechanism. Attackers injected a `preinstall` hook into malicious package versions that, on installation, downloaded Bun version 1.3.13 from GitHub releases if not already present on the host [7]. Bun—a self-contained JavaScript runtime—was used for payload execution; Picus researchers assessed that Node.js-focused endpoint detection tools do not monitor Bun process activity, suggesting evasion of standard security tooling may have been an intentional design choice [7]. The primary payload, an obfuscated credential-harvesting script of approximately 11.7 megabytes, then executed under Bun, scanning 134 file paths for credentials [2].

Credential harvesting targeted an exceptionally broad set of secrets. Beyond scanning well-known credential file locations (`.npmrc`, `~/.aws/credentials`, `~/.kube/config`, `~/.config/gcloud`), the malware included embedded Python helpers that scanned the Linux `/proc` filesystem of live processes to extract GitHub Actions secrets directly from runner worker memory—circumventing log masking and environment variable concealment that CI/CD platforms deploy as standard defenses [6]. Harvested credentials spanned AWS IAM tokens, GCP project credentials, Azure Key Vault data, Kubernetes service account tokens, HashiCorp Vault secrets, npm automation tokens, LLM API keys for services including OpenAI and Anthropic, and credentials stored in password vaults such as 1Password and Bitwarden [3].

Exfiltration used three parallel channels to maximize resilience against infrastructure takedown. Primary exfiltration routed through the typosquatted domain `git-tanstack[.]com`, with backup channels over the Session decentralized messaging network (`filev2.getsession[.]org`) and through GitHub API dead drops using previously stolen tokens [8]. Data was encrypted with AES-256-CBC secured by 4096-bit RSA public keys prior to transmission [6].

Persistence was established through a mechanism not previously documented in supply chain attacks of this class: injection of malicious execution hooks into AI coding agent and editor configuration files committed to project repositories. The malware placed execution triggers in `.claude/settings.json` as SessionStart hooks and in `.vscode/tasks.json` as folderOpen triggers [2]. Because these configuration files reside inside the project directory and are frequently committed to version control, the persistence mechanism travels with the repository. A developer who remediates the infected package dependency nonetheless remains compromised on any machine where the repository is subsequently cloned, because the `.claude/settings.json` hook

fires automatically each time Claude Code opens the project. As reported by CyberScoop, citing Snyk researchers, `.claude/` directories are "typically excluded from version control via .gitignore and rarely scrutinized" [1], meaning the injected hooks routinely evade both automated scanning and manual code review.

Self-propagation closed the worm loop. Using harvested npm publishing tokens, the malware identified additional packages under each victim's control, modified their tarballs to embed a copy of the current payload, incremented patch version numbers, and republished the poisoned releases to the npm registry—all without human intervention [3]. Aikido Security identified 373 distinct malicious package-version entries across 169 npm package names during the peak of the May 2026 campaign [1, 9]. The worm additionally spoofed its commits as originating from `claude@users.noreply.github.com` with commit messages reading "chore: update dependencies," deliberately mimicking AI-assisted development workflows to blend into routine CI/CD activity [2].

AI Agent Persistence: A Novel Attack Surface

Prior supply chain campaigns established persistence through systemd services, macOS LaunchAgents, or hidden cron jobs—mechanisms that endpoint detection tools are broadly configured to monitor. The Mini Shai-Hulud approach differs from these methods in kind: it exploits the trust developers place in AI coding agent configuration files, which are not currently subject to the same scrutiny as CI/CD workflow definitions. Security researchers characterized this as a meaningful escalation in the attack surface addressed by supply chain security tooling.

Claude Code's settings file supports a hooks mechanism that executes arbitrary shell commands at defined lifecycle events including session start, pre-tool-use, and post-tool-use. These hooks are a legitimate feature designed to allow developers to automate linting, environment setup, or test execution. By injecting a malicious SessionStart hook, an attacker ensures that every time the developer opens Claude Code in the compromised project, the credential-harvesting payload executes—regardless of whether any npm package is installed, updated, or present. The payload can steal freshly rotated credentials, harvest new tokens acquired since the initial infection, and continuously re-exfiltrate the developer's environment state across an extended campaign lifetime.

The VS Code variant operates through `.vscode/tasks.json`, a configuration file that specifies tasks triggered on folder open. Both vectors share a critical characteristic: unlike npm packages, which have dedicated security tooling, registry monitoring, and an established community practice of auditing `package.json` for unexpected preinstall scripts, AI coding agent configuration files occupy a gap in

current security practice. No publicly documented static analysis or dependency scanning product is known to routinely inspect these files for malicious content, and the `.claude/` directory conventions do not yet have an equivalent to npm's lock file verification or SLSA provenance framework.

The worm's choice to disguise its commits as Claude bot activity warrants particular attention. As AI coding agents become routine participants in development workflows—committing code, updating dependencies, and managing configuration—distinguishing legitimate agent commits from adversary-spoofed ones requires verification mechanisms that relatively few organizations have deployed.

Attribution and Campaign Context

Unit 42 attributed the Mini Shai-Hulud campaign to TeamPCP with high confidence, based on shared infrastructure, code signatures, and operational patterns consistent with the group's established campaign history [6]. Notably, Unit 42 assessed with moderate confidence that an LLM was used to generate the malicious bash scripts in earlier Shai-Hulud variants, based on the inclusion of structured code comments and emoji usage atypical of human-authored attacker tooling [1]. If accurate, this indicates that TeamPCP is deploying AI-assisted tooling to accelerate and adapt its own attack development—an operational pattern likely to become more common across the threat landscape.

TeamPCP escalated target selection progressively across the campaign arc. The group moved from attacking individual security tools (Trivy, KICS, Bitwarden) with limited blast radius to compromising broadly used developer libraries (LiteLLM, Mistral AI SDK) with large downstream exposure, and finally to frontend ecosystem packages (TanStack's react-router with approximately 12 million weekly downloads) where supply chain reach is maximized [1]. Unit 42 observed that harvested npm tokens allowed automated infection of 47 additional packages across unrelated npm namespaces within 60 seconds of token acquisition [6], illustrating the compounding speed at which a single compromised developer account becomes an ecosystem-wide event.

A structural gap in vulnerability tracking complicated the community response. At the time of the CSA Labs technical analysis, no CVE, GHSA, or OSV record had been assigned to any Mini Shai-Hulud-related package compromise [2]; a CVE was subsequently assigned for a component of the campaign [10]. The core structural issue persists regardless of that subsequent assignment: Mini Shai-Hulud delivers a credential theft operation through legitimately signed package versions rather than exploiting a code vulnerability in the packages themselves, and none of the traditional vulnerability assignment criteria were satisfied for its primary attack mechanics. Organizations relying exclusively on CVE-based tooling received no automated notification for the bulk of the campaign's activity and had no standard remediation workflow to invoke.

Recommendations

Immediate Actions

Organizations that installed any affected packages during their compromise windows should treat development environments as fully compromised. All credentials accessible from developer workstations or CI/CD environments that installed affected versions require immediate rotation: cloud provider tokens (AWS, GCP, Azure), GitHub Personal Access Tokens, npm tokens, Kubernetes service account tokens, and API keys for LLM services. The `gh-token-monitor` daemon installed by some payload variants persists as a systemd service or macOS LaunchAgent polling GitHub token validity every 60 seconds [3], and must be explicitly identified and removed from all potentially affected systems before credential rotation is considered complete.

All project repositories should be audited for injected AI agent configuration files. Specifically, `.claude/settings.json` should be inspected for unexpected `hooks` entries, and `.vscode/tasks.json` should be reviewed for unexpected tasks carrying `runOn: folderOpen` triggers. Any repository containing these injected configurations should be treated as an active persistence vector regardless of whether the underlying npm dependency has been remediated—the configuration file must be cleaned and the change verified before the host is considered safe.

Short-Term Mitigations

CI/CD pipeline configuration requires hardening against the OIDC abuse techniques employed in the TanStack compromise wave. GitHub Actions workflows should be audited for overly permissive `pull_request_target` configurations, particularly any that grant secrets or write permissions to jobs triggered by fork-contributed pull requests. npm Trusted Publishing OIDC policies should be scoped to specific workflows and protected branches, preventing fork-triggered workflows from acquiring publish credentials. Package lockfiles should enforce content-hash pinning, enabling detection of unexpected package content changes between CI runs even when version numbers remain identical.

Supply chain security tooling must extend beyond CVE databases. Because Mini Shai-Hulud generated no CVE records for its core credential-theft mechanics, organizations depending solely on CVE-based vulnerability scanners received no automated notification of compromise. Registry event monitoring—tools that alert on unexpected new package versions from established maintainers—provides coverage where CVE tracking does not. Behavioral analysis tools that inspect npm packages at install time for preinstall script injection, anomalous dependencies, and obfuscated payloads offer detection earlier in the attack chain than registry-level monitoring alone.

Developer security awareness programs should explicitly address the security posture of AI coding agent configuration files. Developers should understand that `.claude/settings.json` hooks and `.vscode/tasks.json` tasks are shell command execution pathways, and that these files committed to version control should receive the same review scrutiny as any executable script or CI/CD workflow definition.

Strategic Considerations

Mini Shai-Hulud demonstrates that AI coding agents have become a meaningful attack surface in software supply chain security. The trust model implicit in tools like Claude Code—that project configuration files are safe to execute automatically—creates an assumption that adversaries can now deliberately exploit. Organizations adopting AI coding agents at scale should develop explicit policy governing what hook configurations are permitted in project repositories and who is authorized to modify them, with enforcement analogous to protected-branch policies for CI/CD workflow files.

The self-propagating nature of credential-reuse supply chain worms requires organizations to model CI/CD credential exposure as an enterprise-wide risk rather than a per-repository incident. A single developer credential compromised through one affected package can cascade to publishing access across every package that developer maintains. Incident response playbooks should account for this amplification dynamic and pre-authorize rapid mass credential rotation across all development infrastructure without requiring per-credential approval workflows that may introduce operationally significant delay during active incidents.

The vulnerability database gap exposed by this campaign warrants broader industry engagement. CSA recommends that security teams supplement CVE-based tooling with registry-level monitoring and adopt SLSA (Supply-chain Levels for Software Artifacts) provenance verification as a baseline requirement for production dependencies—enabling detection of package content changes even when no vulnerability record exists. The structural mismatch between how credential-theft supply chain attacks operate and how CVE assignment criteria are defined suggests a need for expanded advisory frameworks that can capture registry-level compromise events independent of underlying code vulnerability.

CSA Resource Alignment

The Mini Shai-Hulud campaign maps clearly to threat categories addressed in CSA's MAESTRO framework for agentic AI threat modeling. The exploitation of AI coding agent configuration files as execution and persistence vectors represents a concrete instantiation of MAESTRO's concerns around tool trust boundaries and injection of malicious instructions into AI agent operational contexts. As AI

coding agents acquire broader permissions to read, write, and execute code across developer environments, their configuration files become high-value targets that security practitioners should incorporate into existing threat models for developer tooling.

CSA's AI Controls Matrix (AICM) addresses AI supply chain security across its eighteen control domains, with direct relevance in the areas of AI infrastructure security, third-party dependency governance, and pipeline integrity controls. The AICM's shared security responsibility guidance applies to the CI/CD trust failures exploited by TeamPCP: the misconfigured OIDC policies and permissive `pull_request_target` workflows that enabled token extraction without direct maintainer credential theft represent precisely the kind of infrastructure-layer security gap the AICM is designed to surface and remediate.

CSA's Cloud Controls Matrix (CCM) addresses supply chain and third-party risk management under its Supply Chain Management, Transparency, and Accountability (STA) domain. The credential propagation mechanism central to Mini Shai-Hulud—where a single compromised token enables lateral movement across an entire package publishing graph—maps to CCM controls around access provisioning, least privilege enforcement, and credential lifecycle management in cloud-native environments.

Zero Trust principles offer a compelling architectural response to the underlying conditions that enabled this campaign. The attack relied at each stage on implicit trust: trust that packages from established maintainers are safe, trust that OIDC-signed releases reflect maintainer intent, trust that project configuration files are inert, and trust that CI/CD credentials scoped to one workflow will not be accessible to fork-triggered jobs. A Zero Trust posture applied to the software supply chain requires continuous verification of package provenance, explicit validation of configuration file contents, and credential minimization that prevents any single stolen token from propagating across the full publishing graph. CSA's Zero Trust Guidance for Achieving Operational Resilience provides applicable architectural principles for organizations seeking to implement these controls at the CI/CD and developer tooling layer.

References

[1] CyberScoop. "'Mini Shai-Hulud' malware compromises hundreds of open-source packages in sprawling supply-chain attack." CyberScoop, May 12, 2026.

[2] Cloud Security Alliance Labs. "Mini Shai-Hulud: Multi-Ecosystem Developer Supply Chain Attack." CSA Labs, May 2026.

[3] Wiz. "Mini Shai-Hulud Strikes Again: TanStack + more npm Packages Compromised." Wiz Blog, May 2026.

[4] ReversingLabs. "2026 Software Supply Chain Security Report." ReversingLabs, 2026 (registration required).

[5] Arctic Wolf. "TeamPCP Supply Chain Attack Campaign Targets Trivy, Checkmarx (KICS), and LiteLLM." Arctic Wolf, 2026.

[6] Unit 42, Palo Alto Networks. "Weaponizing the Protectors: TeamPCP's Multi-Stage Supply Chain Attack on Security Infrastructure." Palo Alto Networks, 2026.

[7] Picus Security. "Mini Shai-Hulud: The npm Supply Chain Worm Explained." Picus Security Blog, May 2026.

[8] Expel. "Mini Shai Hulud: Cross-Ecosystem Supply Chain Worm Targeting npm & PyPI." Expel Blog, May 2026.

[9] ReversingLabs. "Team PCP's Mini Shai-Hulud tears at open-source trust." ReversingLabs Blog, May 2026.

[10] The Hacker News. "Mini Shai-Hulud Worm Compromises TanStack, Mistral AI, Guardrails AI & More Packages." The Hacker News, May 2026.

[11] Microsoft Security Blog. "Guidance for detecting, investigating, and defending against the Trivy supply chain compromise." Microsoft, March 24, 2026.