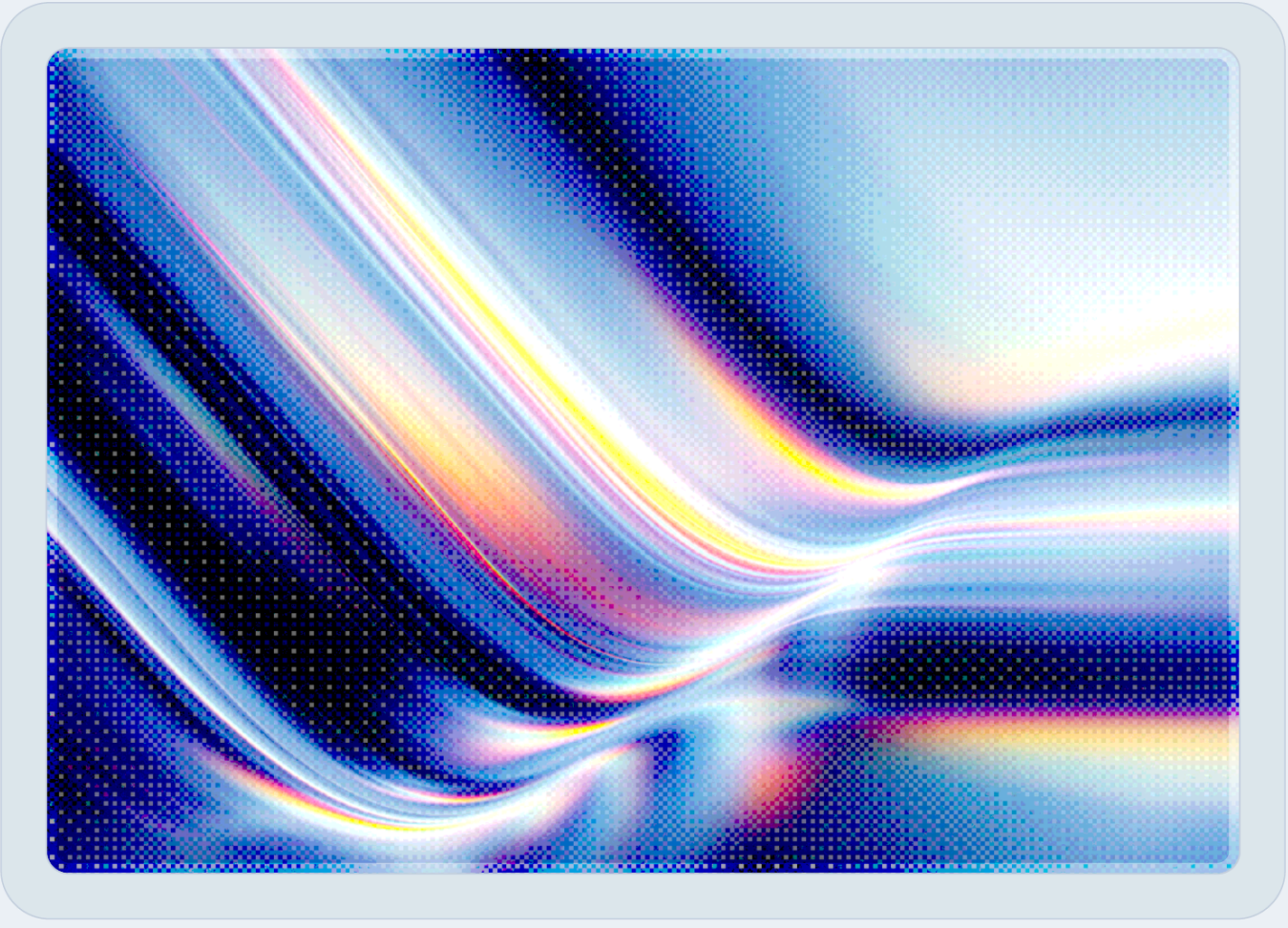
CSAI Foundation | Cloud Security Alliance

Mini Shai-Hulud: npm Worm Targets AI Developer Supply Chain

How TeamPCP's Self-Propagating Worm Forged SLSA Attestations and Weaponized AI Coding Assistants as Persistence Vectors

2026-05-17

AI-assisted Rapid Research



CSAI | CSA cloud security alliance®

Key Takeaways

- Beginning in late April 2026 and reaching peak scale in a May 11 wave, threat actor TeamPCP's self-propagating supply chain worm dubbed Mini Shai-Hulud compromised 373 malicious package-version entries across 169 npm packages and two PyPI packages with a cumulative download base exceeding 518 million [1][2][18].
- The worm became the first documented supply chain attack to forge valid SLSA Build Level 3 provenance attestations by extracting GitHub Actions OIDC tokens from runner process memory and using them to obtain legitimate Sigstore signing certificates — meaning malicious package versions appeared cryptographically verified to automated security tooling [3][4].
- Mini Shai-Hulud is also the first supply chain attack on record to weaponize AI coding agent configuration files as a persistence mechanism, writing malicious hooks into Claude Code's `.claude/settings.json` and VS Code's `.vscode/tasks.json` that survive package removal and reactivate every time a developer opens a project [5][6].
- Victims include the @tanstack namespace (~12 million weekly downloads for `@tanstack/react-router` alone), UiPath's npm estate, Mistral AI's npm and PyPI packages, Guardrails AI on PyPI, and the OpenSearch JavaScript client [2][7][8].
- Remediation order is safety-critical: the malware's persistence service triggers home directory destruction if credentials are rotated before the service is disabled [12].

Background

The Mini Shai-Hulud campaign did not emerge in isolation. Its lineage traces to a series of increasingly sophisticated attacks against the npm ecosystem beginning in August 2025, each iteration introducing new capabilities in response to detection and defensive improvements.

The chain began with the s1ngularity campaign in late August 2025, a targeted attack against Nx monorepo tooling packages that harvested 2,349 developer credentials from 1,079 systems [10]. Although s1ngularity was primarily a credential-theft operation rather than a worm, researchers assessed that the GitHub personal access tokens and npm publish tokens it harvested likely provided the access used to seed the next escalation [10]. On September 15, 2025, a threat actor launched Shai-Hulud, the

first true self-replicating worm observed in the npm ecosystem. Named by researchers after an artifact filename referencing the sandworm creature in Frank Herbert's *Dune* series, the original variant infected hundreds of packages — including widely used libraries like `ngx-bootstrap` — by combining credential harvesting with an automated dissemination loop that used each victim maintainer's existing publish rights to spread to every additional package they controlled [11].

A second wave, Shai-Hulud 2.0, appeared in early November 2025 as researchers from Palo Alto Networks Unit 42 documented a renewed compromise campaign that refined the worm's initial access techniques [12]. A related variant designated SANDWORM_MODE, first disclosed by the Socket Research Team and subsequently analyzed by Sonatype on March 2, 2026 [13], introduced adaptive targeting that allowed the worm to enumerate CI/CD pipeline structures before deciding how to propagate. Each generation of this worm family introduced capabilities that directly addressed detection and takedown techniques applied to its predecessor, suggesting the actors monitored defensive responses and adapted accordingly.

Mini Shai-Hulud, active since late April 2026 and most destructive in the May 11 wave, represents the fourth-generation evolution of this lineage. Its distinguishing characteristics — provenance attestation forgery and AI coding agent persistence — reflect a deliberate adaptation to two security controls that have gained adoption since the original 2025 incidents: software provenance verification and AI-assisted development workflows that operate with elevated filesystem access.

Security Analysis

Worm Propagation Mechanism

Mini Shai-Hulud's propagation chain begins with a package installation event. When a developer or CI/CD pipeline installs a compromised package version, the malware's `postinstall` hook executes a 2.3 MB obfuscated payload that immediately begins two parallel operations: credential harvesting and pipeline reconnaissance [5].

The credential harvesting component reads from over 100 distinct file paths, targeting GitHub personal access tokens, npm publish tokens, AWS IAM credentials, HashiCorp Vault secrets, cloud provider configuration files, cryptocurrency wallet seeds, and authentication tokens for AI tools and messaging applications [2][6]. The breadth of targeting reflects the malware authors' knowledge that modern developer environments aggregate credentials for many unrelated systems into well-known file locations.

The pipeline reconnaissance component reads GitHub Actions runner process memory directly, extracting OIDC tokens that are available only within the execution context of an active workflow job [3]. These tokens are ephemeral by design, but the worm exploits the window during which a workflow is executing — triggered, for instance, by a dependency installation step — to extract a valid OIDC token before it expires. With that token, the worm contacts Sigstore's Fulcio certificate authority to obtain a code-signing certificate, which it then uses to generate a SLSA Build Level 3 provenance attestation over the malicious package version it is about to publish [4][17].

The provenance attestation bypass exploits a design assumption rather than a technical vulnerability in the SLSA framework itself. SLSA Build Level 3 attestations certify that a package was built by a specific, verified CI/CD system. They cannot certify that the code running inside that CI/CD job was the code the maintainer intended to build — if the worm arrived via a dependency installed earlier in the pipeline, the attestation reflects the legitimate pipeline's identity while the artifact contains malicious content [4]. The result is that malicious package versions in the Mini Shai-Hulud campaign passed automated provenance verification tools that many organizations now treat as a supply chain security control.

AI Coding Agent Persistence

Unlike prior persistence mechanisms — OS-level services, cron jobs, or modified shell profiles — Mini Shai-Hulud's persistence strategy survives standard remediation playbooks by exploiting a class of configuration file that those playbooks do not address. After harvesting credentials and establishing its publish capability, the worm writes modified configuration files to two locations that AI coding assistants read automatically: `~/.claude/settings.json` for Claude Code and `.vscode/tasks.json` for VS Code [5][6]. Because these tools interpret project-level and user-level configuration files as trusted instruction sources, the malicious hooks embedded in them execute with the same permissions as legitimate tool operations and survive complete removal of the originally infected package.

In addition to AI coding assistant hooks, the worm installs an OS-level persistence service (identified in incident reports as `gh-token-monitor.service` on Linux systems) that survives reboots and monitors for credential rotation events [12]. If a developer rotates their GitHub token while this service is still active, the service is configured to destroy the home directory — a destructive tripwire that penalizes hasty remediation and coerces victims into delaying the credential rotation that would otherwise neutralize the attacker's access.

Security researchers at Phoenix Security characterized Mini Shai-Hulud as the first documented supply chain attack to weaponize AI coding agent configurations as a persistence mechanism [5]. The significance is structural: AI coding assistants now occupy a privileged position in development workflows

because they must read broadly to be useful, and the configuration files that govern their behavior had not previously been documented as an attack surface requiring hardened access controls in standard security guidance.

Affected Packages and Scale

The table below summarizes the primary affected namespaces and their exposure footprint based on reporting from StepSecurity, Corgea, and Lyrie Research as of May 13, 2026 [1][7][14].

Namespace / Package	Ecosystem	Malicious Entries	Notable Scope
@tanstack	npm	83 entries	router, start, devtools, adapter — ~12M weekly downloads for react-router [2] [7]
@squawk	npm	87 entries	SQL linting tools
@uipath	npm	66 entries	Automation platform SDK
@mistralai	npm	Multiple	mistralai, mistralai-azure, mistralai-gcp
mistralai	PyPI	1 (v2.4.6)	Official Mistral AI Python client
guardrails-ai	PyPI	1 (v0.10.1)	AI output validation framework
opensearch-project/opensearch	npm	Multiple	JavaScript search client
intercom-client, lightning	npm	Multiple	Enterprise integrations
SAP CAP ecosystem	npm	Multiple	Enterprise application development [16]

The cumulative download count across affected packages exceeded 518 million at the time of disclosure [2]. That figure represents historical downloads rather than a count of actively compromised installations, but it indicates the ecosystem surface through which the worm propagated and the number

of developer environments that may have installed a malicious version during the active campaign window (approximately April 29 through May 13, 2026).

Connection to the AI Development Ecosystem

Two aspects of this campaign are consistent with deliberate targeting of the AI application development community, or at minimum reflect an awareness of AI development workflows as an exploitation surface. First, the worm specifically targeted the npm and PyPI packages most likely to appear in AI-adjacent development stacks: Mistral AI's official client libraries, Guardrails AI (a framework for validating LLM outputs), and UiPath (widely used for agentic automation). Developers building on large language model APIs, retrieval-augmented generation pipelines, or AI agent frameworks were likely to have one or more affected packages in their dependency trees, given the direct targeting of Mistral AI, Guardrails AI, and UiPath.

Second, the AI coding assistant persistence mechanism reflects an understanding that the most valuable targets in modern AI development are developers whose tools already have broad access to cloud credentials, repository contents, and CI/CD secrets. Coding assistants like Claude Code and VS Code with Copilot operate with elevated trust in developer environments precisely because their utility depends on it. Mini Shai-Hulud exploited that elevated trust to establish a persistence layer that outlasts the initial compromise and that many standard remediation playbooks would not have addressed prior to this incident's disclosure.

Recommendations

Immediate Actions

Any development team or CI/CD environment that installed packages from the affected namespaces between April 29 and May 13, 2026 should treat the environment as potentially compromised and proceed in the following sequence, which is safety-critical because of the malware's destructive tripwire behavior.

First, audit for active persistence before rotating any credentials. Check for unauthorized system services — on Linux, look for `gh-token-monitor.service`; on macOS, check LaunchAgents for entries outside normal application paths. Disable any unauthorized persistence services before proceeding [12]. In parallel, inspect `~/.claude/settings.json`, `.vscode/tasks.json`,

and `.github/workflows/` for unexpected entries [5][6], and check all repositories the team controls for new branches named `shai-hulud` or workflow files at `.github/workflows/discussion.yaml` or `shai-hulud-workflow.yml`.

Second, after disabling persistence services, rotate all credentials that may have been present on affected systems: GitHub personal access tokens, npm publish tokens, AWS IAM access keys, HashiCorp Vault tokens, and API keys for any AI platforms. Implement DNS-level blocking for `api.masscan.cloud`, which researchers identified as a command-and-control endpoint. Teams should also audit whether any packages they publish were released from CI runs executing during the exposure window, as the worm may have used their OIDC trust to publish infected versions of downstream packages.

Short-Term Mitigations

The provenance attestation bypass demonstrates that SLSA Build Level 3 attestations are not sufficient to detect attacks that compromise the build environment itself from within. Organizations using provenance verification as a primary supply chain control should layer it with additional signals: runtime behavioral analysis during package installation, network egress monitoring from CI/CD runners, and restrictions on what processes can access runner memory during job execution. The OIDC token extraction technique can be partially mitigated by scoping OIDC token permissions to the minimum required for each workflow job and by auditing token claims in attestation metadata for unexpected audience or scope values [3].

AI coding assistant configuration files should be treated with the same access-control discipline applied to shell profiles and CI/CD configuration files. Teams should restrict write access to `~/.claude/settings.json` and equivalent VS Code settings paths, monitor these files for unexpected changes, and include them in code review scope when they appear as changed files in pull requests. Organizations that have deployed coding assistants broadly should consider whether their endpoint security tooling covers these configuration paths.

Dependency pinning — specifying exact package versions rather than semver ranges — limits exposure to worm-published updates because the malicious version must match the pinned value to execute. While pinning does not eliminate supply chain risk, post-incident analyses of the Mini Shai-Hulud campaign found that teams using lockfiles with exact version hashes were substantially less likely to have installed malicious versions during the active campaign window.

Strategic Considerations

The Mini Shai-Hulud campaign illustrates that the threat model for open-source dependency security has advanced beyond typosquatting and abandoned maintainer accounts. Supply chain worms capable of self-propagation, attestation forgery, and AI coding tool exploitation represent a category of threat that requires architectural responses, not just policy adjustments.

Organizations should evaluate whether their current dependency security posture assumes provenance attestations are sufficient proof of package integrity. This campaign demonstrates that attestations can be produced by legitimate CI infrastructure running malicious code — the provenance is real, but the artifact is compromised. Supply chain security programs should incorporate runtime behavioral baselines for package installation, treat SLSA attestations as necessary but not sufficient controls, and invest in monitoring that can detect anomalous publish events from otherwise legitimate pipelines.

The AI coding assistant persistence vector also points toward a broader architectural review. Any tool that reads project-level configuration with elevated permissions creates a persistence surface. Security architecture for AI-assisted development workflows should include explicit trust boundaries for configuration file sources, mechanisms for detecting configuration tampering, and playbooks that cover AI tool configuration files in standard incident response procedures.

Post-incident guidance from Chainguard [15] reframes the strategic challenge: the problem is not simply how to layer additional controls over public registry consumption, but whether each consumption point in the development pipeline has an explicit, auditable basis for trust. Organizations that pull from npm, PyPI, or other public registries without a defined trust policy — relying on implicit assumptions about maintainer legitimacy and publication process integrity — are operating on premises that this campaign and its predecessors have demonstrated can be systematically exploited. Establishing explicit trust criteria for registry consumption, rather than treating public package installation as a default-trusted operation, represents the architectural shift that Mini Shai-Hulud has made necessary.

CSA Resource Alignment

The Mini Shai-Hulud campaign intersects with several frameworks and guidance documents maintained by the Cloud Security Alliance.

The **MAESTRO** (Multilayer Agentic Environment Security Threat and Risk Overview) framework directly addresses the attack patterns observed here. MAESTRO's analysis of agentic AI threat layers is particularly relevant to the persistence technique that exploits AI coding assistant configurations: agents and coding assistants that interpret external configuration with broad filesystem and credential access

represent high-value lateral movement targets, and MAESTRO's guidance on agentic privilege boundaries and configuration trust applies directly to the `.claude/settings.json` and `.vscode/tasks.json` attack surface. Teams deploying AI coding assistants in development environments should apply MAESTRO's principle of minimal configuration trust scope to these tools.

The **AI Controls Matrix (AICM)**, CSA's comprehensive control framework for AI systems, covers several control domains relevant to this incident: supply chain integrity controls for AI-adjacent software, credential protection for AI platform access tokens, and monitoring requirements for AI tool configurations. Because Mistral AI and Guardrails AI were among the directly affected packages, the AICM's guidance on validating AI library provenance and auditing AI SDK versions applies directly to organizations building on these platforms.

CSA's Zero Trust guidance is applicable to the CI/CD credential extraction component. A zero trust posture applied to GitHub Actions OIDC token scopes — limiting each token to the minimum audience and permissions required by its specific job — would have reduced the worm's ability to produce valid SLSA attestations using tokens extracted from unrelated pipeline steps.

The **Cloud Controls Matrix (CCM)**, specifically its supply chain management and change management control families, addresses the organizational controls that would have limited the blast radius of this incident: package version approval workflows, artifact integrity verification, and restrictions on automated publish permissions. Organizations that map their software supply chain controls to CCM should review this incident against the Supply Chain Management (STA) control group and assess whether their implementations account for worm-class propagation patterns.

CSA's **AI Organizational Responsibilities** guidance covers the stewardship obligations of organizations that publish AI-related packages or maintain AI platform integrations — directly relevant to the Mistral AI and Guardrails AI compromises. Maintainers of AI ecosystem packages should apply this guidance when evaluating the scope of their publishing permissions, the breadth of their OIDC token grants, and their incident response obligations when their packages are used as a worm propagation vector.

References

[1] StepSecurity. "TeamPCP's Mini Shai-Hulud Is Back: A Self-Spreading Supply Chain Attack Compromises TanStack npm Packages." StepSecurity Blog, May 2026.

[2] Corgea. "Mini Shai-Hulud Supply-Chain Worm Compromises TanStack, Mistral AI, UiPath, and 160+ npm Packages." Corgea Research, May 2026.

[3] Vectra AI. "Shai-Hulud Part 2: When the Worm Forged Its Own Security Certificate." Vectra AI Blog, May 2026.

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

[5] Phoenix Security. "Mini Shai-Hulud: SAP CAP and mbt npm Packages Backdoored via Bun-Loaded Credential Stealer with Claude Code Persistence." Phoenix Security, May 2026.

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

[7] Lyrie Research. "Mini Shai-Hulud Escalates: 169 npm Packages, Mistral AI, UiPath, and Now PyPI — The Self-Spreading Supply-Chain Worm." Lyrie Research, May 13, 2026.

[8] SecurityWeek. "TanStack, Mistral AI, UiPath Hit in Fresh Supply Chain Attack." SecurityWeek, May 2026.

[9] Cyber Chief. "Mini Shai-Hulud Breach: Practical Incident Assessment & Recovery Guide for Engineering Teams." Cyber Chief, May 2026.

[10] Cloudsmith. "npm Ecosystem Update: A New Attack Vector Emerged with 's1ngularity' and 'shai-hulud' Attacks." Cloudsmith Blog, September 2025.

[11] Sysdig. "Shai-Hulud: The Novel Self-Replicating Worm Infecting Hundreds of npm Packages." Sysdig Blog, September 2025.

[12] Palo Alto Networks Unit 42. "'Shai-Hulud' Worm Compromises npm Ecosystem in Supply Chain Attack." Unit 42, November 2025.

[13] Sonatype. "SANDWORM_MODE: The Rise of Adaptive Supply Chain Worms." Sonatype Blog, March 2, 2026.

[14] Socket. "TanStack npm Packages Compromised in Ongoing Mini Shai-Hulud Supply Chain Attack." Socket Blog, May 2026.

[15] Chainguard. "Luck Isn't a Security Control: What Happened with Mini Shai-Hulud and What You Need to Do." Chainguard, May 2026.

[16] Onapsis. "Emerging Supply Chain Attack ('Mini Shai-Hulud') Targeting SAP Cloud Application Programming Ecosystem." Onapsis Blog, May 2026.

[17] Snyk. "TanStack npm Packages Hit by Mini Shai-Hulud." Snyk Blog, May 2026.

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