CSAI Foundation

Cloud Security Alliance AI Safety Initiative

Shai-Hulud: npm Worm Targeting AI Developer Toolchains

How a Self-Propagating Supply Chain Threat Weaponizes MCP Servers and Coding Assistants

Unofficial AI-assisted Research

2026-04-25

Key Takeaways

- The Shai-Hulud worm family represents the first documented self-replicating supply chain threat operating at ecosystem scale within npm, having evolved through at least three distinct campaigns between September 2025 and April 2026 and compromising more than 796 packages with over 20 million combined weekly downloads [1][2][3].
- The latest variants specifically target AI developer tooling: malicious packages deploy fake Model Context Protocol (MCP) servers embedded with prompt-injection payloads that instruct AI coding assistants to silently harvest SSH keys, cloud credentials, and LLM API tokens without notifying the user [4].
- The April 2026 "Third Coming" campaign compromised `@bitwarden/cli` version 2026.4.0, explicitly targeting files used by Claude Code and MCP configurations, and exfiltrating encrypted data to public GitHub repositories as "dead-drop" relay points [5][6].
- The SANDWORM_MODE variant (February 2026) introduced time-gated CI detection, three-channel exfiltration including DNS tunneling, and weaponized GitHub Actions that serialize all pipeline secrets — capabilities that, taken together, represent a meaningful increase in operational sophistication [4].
- Organizations using AI coding assistants connected to npm-sourced dependencies face compounded risk: a single infected transitive dependency may simultaneously compromise developer credentials [3], CI/CD pipelines [4], and the LLM toolchain the team relies on for remediation [4][5].

Background

The Shai-Hulud worm — named after the mythological sandworms of Frank Herbert's *Dune* — was first identified by ReversingLabs researchers on September 15, 2025, with the package `rxnt-authentication` version 0.0.3 serving as the documented Patient Zero, published one day earlier [1]. It was the first documented instance of a self-replicating agent in the npm ecosystem: not a static

backdoored package that required victims to stumble upon it, but an autonomous agent that used stolen developer credentials to inject itself into other packages and publish compromised versions without any human intervention.

The worm's propagation model is designed for cascade efficiency. During package installation, a postinstall (or, in later variants, a preinstall) hook executes a bundled malicious script that searches the developer's filesystem for npm authentication tokens. Once a valid token is found, the worm authenticates to the npm registry under the victim's identity, enumerates every package that identity has publish rights over, injects itself as an obfuscated payload, and publishes new versions — all within seconds of initial infection. The cascade effect is inherent to the design: every developer who installs a package from an infected maintainer becomes a new propagation node. Unit 42, analyzing early campaign artifacts, assessed with moderate confidence that an LLM was used to generate portions of the malicious bash scripting, noting the presence of code comments and emoji decorators atypical of handwritten malware [2].

The November 2025 Shai-Hulud 2.0 campaign demonstrated how rapidly the threat matured. That wave compromised 796 unique npm packages across 1,092 affected versions and achieved estimated reach of over 20 million weekly downloads across infected packages — among them `@ctrl/tinycolor` (2.2 million weekly downloads), `ngx-bootstrap` (approximately 300,000 weekly downloads), and `ng2-file-upload` (approximately 100,000 weekly downloads) [3]. Credential exfiltration was confirmed across more than 500 GitHub users spanning 150 or more organizations, with peak exfiltration observed November 24, 2025 [3]. The 2.0 variant also introduced self-directed lateral movement: when local npm tokens were unavailable, the worm proactively searched public GitHub repositories for credentials previously exfiltrated by other infected machines, enabling compromise across otherwise-unrelated victim populations [3].

By February 2026, the SANDWORM_MODE campaign disclosed by Socket Research Team on February 20 marked a third evolutionary step, deploying 19 malicious typosquatted packages — including names impersonating Claude Code (`claud-code`, `cloude`) and the OpenClaw AI agent framework (`opencraw`) — with a multi-stage architecture explicitly designed to target AI developer tooling [4][7].

The April 23, 2026 "Third Coming" campaign brought the threat directly into the trusted supply chain. The official `@bitwarden/cli` package, version 2026.4.0, was found to contain a malicious preinstall hook that bootstrapped the Bun JavaScript runtime and deployed a heavily obfuscated payload of approximately 10 MB [5][6]. The embedded string "Shai-Hulud: The Third Coming" made explicit the actors' awareness of their own operational lineage. The Bitwarden CLI's presence as a dependency across developer and DevOps tooling stacks — combined with its 250,000+ monthly downloads [6] — made its compromise especially high-leverage compared to a typical malicious package. Code artifacts in the payload included a Russian-language bypass — logic designed to detect and avoid infecting

Russian-locale systems [6]. OX Security analysts noted that this pattern is consistent with threat actor tradecraft in nation-state-adjacent operations seeking to protect domestic development infrastructure during testing, though definitive attribution was not established [6].

Security Analysis

Self-Propagation Mechanics

What distinguishes Shai-Hulud from ordinary supply chain backdoors is its worm topology: each infection creates a new infection vector without any additional attacker action. The worm reads its own payload code to use as propagation material, enabling spread without requiring command-and-control instructions for each replication cycle [3]. The SANDWORM_MODE variant extended this model through three cascading propagation channels. Its primary mechanism mirrors the original worm — npm token theft followed by package injection and publishing. When npm access is insufficient, the worm falls back to the GitHub API, enumerating repositories, injecting itself as a dependency in `package.json` files, committing the changes, and opening pull requests configured to auto-merge. If direct push access is blocked, an SSH fallback clones repositories using harvested SSH keys, commits changes under the victim's own git identity, and pushes branches [4].

This multi-vector propagation creates an unusually durable beachhead. Even if an organization rotates its npm token promptly after detection, a concurrent GitHub API or SSH propagation path may have already seeded malicious pull requests into downstream repositories — pull requests that may be merged by automated CI systems before defenders identify and close them.

AI Toolchain as Attack Surface

The emergence of AI developer tooling as an explicit attack target represents a meaningful escalation in the threat model. SANDWORM_MODE introduced a purpose-built AI toolchain poisoning module that creates hidden directories on infected systems and deploys fake MCP servers presenting as legitimate developer utilities — offering functions named `index_project`, `lint_check`, and `scan_dependencies` to avoid suspicion [4]. Each tool description embeds a prompt-injection block instructing connected AI assistants to read SSH private keys, AWS credentials, npm tokens, and any environment variables containing the patterns `TOKEN`, `KEY`, `SECRET`, or `PASSWORD`, then pass the collected material as context to the tool call — explicitly instructing the assistant not to mention this data collection to the user [4].

The targeted AI coding assistants include Claude Code, Claude Desktop, Cursor, VS Code Continue, and Windsurf/Codeium [4]. The April 2026 Bitwarden CLI campaign took a more direct approach: rather than injecting a fake MCP server, the payload specifically targeted `~/.claude.json` and `~/.claude/mcp.json`, harvesting API keys and database credentials that might be stored in those configurations [5]. LLM API key harvesting encompassed credentials for Anthropic, OpenAI, Google, Groq, Together, Fireworks, Replicate, Mistral, and Cohere [4].

A developer whose AI coding assistant is connected to a compromised MCP server may instinctively turn to that same assistant to help investigate the suspicious package or audit their dependencies. In theory, if the assistant is operating under attacker-controlled tool definitions, it could be manipulated into producing misleading analysis or actively withholding evidence of the compromise while appearing to assist with remediation — compounding both the breach and the difficulty of detecting it.

CI/CD Pipeline Weaponization

Both SANDWORM_MODE and the Bitwarden CLI campaign specifically targeted continuous integration environments. SANDWORM_MODE implemented a time-gate bypass: when environment variables such as `GITHUB_ACTIONS`, `GITLAB_CI`, or `CIRCLECI` are detected, Stage 2 execution proceeds immediately rather than waiting for time-delay conditions to elapse, prioritizing high-value CI pipeline secrets over slower-yielding developer workstation credentials [4]. A companion malicious GitHub Action, `ci-quality/code-quality-check`, presented as a benign code quality scanning tool while implementing CI secret harvesting and automated repository modification using `GITHUB_TOKEN` [4].

The most dangerous CI injection technique involves `pull_request_target` workflow manipulation. By injecting workflows that serialize pipeline secrets using the `${{ toJSON(secrets) }}` expression, the worm gains access to secrets stored in GitHub Actions environments — including deployment keys, cloud provider credentials, and registry authentication tokens — that are available to trusted workflows but not to untrusted pull requests from forks [4]. This technique exploits a GitHub Actions security boundary — the `pull_request_target` trigger's access to parent-repository secrets — that is documented in GitHub's security guidance but frequently misconfigured in practice.

Data Exfiltration and Persistence

Exfiltrated data in the SANDWORM_MODE campaign transited a three-channel cascade designed for resilience against takedown. The primary channel used HTTPS POST to a Cloudflare Worker endpoint. If that channel was unavailable, data was double-base64-encoded and uploaded to private GitHub repositories using API calls authenticated with stolen tokens. The final fallback used DNS tunneling via base32-encoded queries with domain generation algorithm (DGA) seeding derived from the hardcoded value "sw2025" [4]. The Bitwarden CLI campaign used AES-256-GCM encryption before exfiltrating results to public GitHub repositories created under compromised developer credentials, with files named `results-TIMESTAMP-ID.json` and Dune-themed repository names such as "fremen-sandworm-441" [5][6]. Commit messages began with "LongLiveTheResistanceAgainstMachines."

Persistence on infected systems extended beyond the package ecosystem. SANDWORM_MODE planted git hooks through the global `init.templateDir` configuration, ensuring that all future repository initializations on the infected machine would inherit infected hook templates — a mechanism that survives package removal and token rotation [4]. The semantic-release configuration patching technique embedded execution of the carrier package within legitimate automated release tooling, meaning the payload would re-execute on any subsequent package publish.

The Checkmarx-branded C2 endpoint (`audit.checkmarx[.]cx/v1/telemetry`) used in the Bitwarden campaign is a notable indicator. By masquerading as application security telemetry from a known vendor, outbound traffic to this endpoint may evade network monitoring rules that allowlist security scanning traffic [5].

Recommendations

Immediate Actions

Organizations should audit all npm package lockfiles and `package.json` dependency trees for known indicators of compromise across the Shai-Hulud family. ReversingLabs and Socket Research Team have published IOC lists including package names, versions, and SHA-1 and SHA-256 hashes for known malicious artifacts [1][4]. Any system that installed `@bitwarden/cli` version 2026.4.0 should be treated as fully compromised: npm tokens, GitHub PATs, cloud provider credentials, SSH keys, and any LLM API keys stored in that environment must be rotated immediately, not sequentially.

Developers should inspect their global git configuration (`git config --global core.hooksPath` and `init.templateDir`) for unexpected values, which may indicate SANDWORM_MODE hook planting. All GitHub Actions workflow files in any repository the developer has contributed to — particularly those referencing `pull_request_target` or `${{ toJSON(secrets) }}` — should be audited for unauthorized modification.

MCP server configurations across affected AI coding assistants deserve specific review. Any MCP server registered in `~/.claude/mcp.json`, Cursor settings, or VS Code Continue configurations that cannot be traced to a known-good source should be treated as potentially malicious and removed. Particular scrutiny is warranted for servers offering tools with names like `index_project`, `lint_check`, or `scan_dependencies`.

Short-Term Mitigations

Adoption of trusted publishing mechanisms — specifically npm's OIDC-based provenance attestation — would make it detectable when a worm republishes an artifact outside the legitimate workflow, though effectiveness depends on broad ecosystem adoption and consumer-side verification. Tooling like Socket's GitHub application can provide dependency review at pull request time, blocking packages with known-malicious postinstall or preinstall hooks before they reach developer environments [4].

CI/CD secret hygiene should be audited against least-privilege principles. Repository secrets, environment secrets, and organization secrets should be scoped to only the workflows and jobs that require them. The `pull_request_target` workflow trigger should be disabled or heavily restricted in repositories that accept external contributions, as it is the primary mechanism enabling cross-fork secret exfiltration in this campaign family.

Network egress monitoring should include alerts on unexpected DNS query patterns — particularly high-volume queries to unfamiliar domains with base32-encoded subdomains — as these are indicators of the DNS tunneling fallback exfiltration channel documented in SANDWORM_MODE [4].

Strategic Considerations

The Shai-Hulud campaigns exposed a gap in how compositional analysis tools operate in practice: they assess a package's integrity at ingestion time but cannot detect a version republished hours later under a legitimate maintainer's compromised credentials. The malicious version, in many configurations, passes provenance checks because it genuinely originates from the legitimate maintainer's signing key — the

maintainer's account was simply used against their will. This is a meaningfully different threat model from a malicious actor publishing a backdoored package under a pseudonymous identity, and it requires a correspondingly different defensive posture.

This argues for defense-in-depth beyond supply chain analysis at ingestion time, including runtime behavioral monitoring of postinstall and preinstall hook execution, credential vault isolation from developer environments, and mandatory MFA with phishing-resistant methods across all package registry accounts. It also argues for organizational awareness that AI coding assistants — now deeply integrated into developer workflows — introduce new attack surfaces that adversaries are actively exploiting. Prompt injection via MCP tool descriptions is not a theoretical vulnerability; it was operationalized in the SANDWORM_MODE campaign. Organizations deploying AI coding tools in sensitive environments should treat the MCP server attack surface, by analogy, with the same scrutiny applied to third-party browser extensions: prefer known-good, audited servers over community-sourced plugins, and never grant MCP tools access to credentials or key material without explicit user confirmation per tool invocation.

CSA Resource Alignment

The Shai-Hulud worm family maps directly onto the MAESTRO framework's supply chain and cross-layer threat categories. MAESTRO describes supply chain attacks as threats that target the AI agent's dependencies, compromising software before delivery and distribution — precisely the mechanism by which SANDWORM_MODE and the Bitwarden CLI campaign operate against developer environments that include AI coding assistants [8]. Mapped to the MAESTRO framework, the MCP server injection technique falls at the intersection of Layer 2 (Data Operations) and Layer 3 (Agent Frameworks), where malicious indirect input sources — in this case, attacker-controlled tool definitions — manipulate agent behavior in ways that users cannot observe.

The prompt injection component of the AI toolchain poisoning module is a practical instance of the indirect prompt injection threat class documented in CSA's research on confused deputy attacks in AI agents [9]. In the Shai-Hulud context, the AI assistant is the confused deputy: it is trusted by the user to assist with development tasks, but malicious tool descriptions redirect its capability toward credential exfiltration without the user's knowledge or consent. CSA's AI Controls Matrix (AICM) provides a framework for addressing this attack surface through agent input validation and tool trust controls, including guidance that tool schemas and descriptions originate from verified, integrity-checked sources and that tool invocations requesting access to sensitive files be surfaced to users for explicit approval.

Organizations should apply CSA's Zero Trust principles to their MCP server ecosystem: no MCP server should be trusted by default based solely on its presence in a configuration file, and access to sensitive filesystem paths or credential stores should require explicit, per-request authorization. This is directly parallel to the Zero Trust principle of never-implicit-trust applied to network access — extended to the tool-use layer of AI agent architectures. CSA's STAR for AI program provides an assessment framework for evaluating AI toolchain risk at the organizational level, including dependency chain integrity and model context interface security.

References

[1] ReversingLabs. "FAQ: The Shai-hulud npm worm attack explained." ReversingLabs Blog, September 2025.

[2] Palo Alto Networks Unit 42. "'Shai-Hulud' Worm Compromises npm Ecosystem in Supply Chain Attack." Unit 42 Threat Research, Updated November 26, 2025.

[3] Datadog Security Labs. "The Shai-Hulud 2.0 npm worm: analysis, and what you need to know." Datadog Security Labs, November 2025.

[4] Socket Research Team. "SANDWORM_MODE: Shai-Hulud-Style npm Worm Hijacks CI Workflow and Targets AI Toolchain." Socket.dev Blog, February 20, 2026.

[5] Aikido Security. "Is Shai-Hulud Back? Compromised Bitwarden CLI Contains a Self-Propagating npm Worm." Aikido.dev Blog, April 2026.

[6] OX Security. "Shai-Hulud: The Third Coming — Bitwarden CLI Backdoored in Latest Supply Chain Campaign." OX Security Blog, April 2026.

[7] CSO Online. "Shai-Hulud-style NPM worm hits CI pipelines and AI coding tools." CSO Online, February 24, 2026.

[8] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." CSA Blog, February 2025.

[9] Cloud Security Alliance Labs. "Confused Deputy Attacks on Autonomous AI Agents." CSA Labs Research, March 2026.

