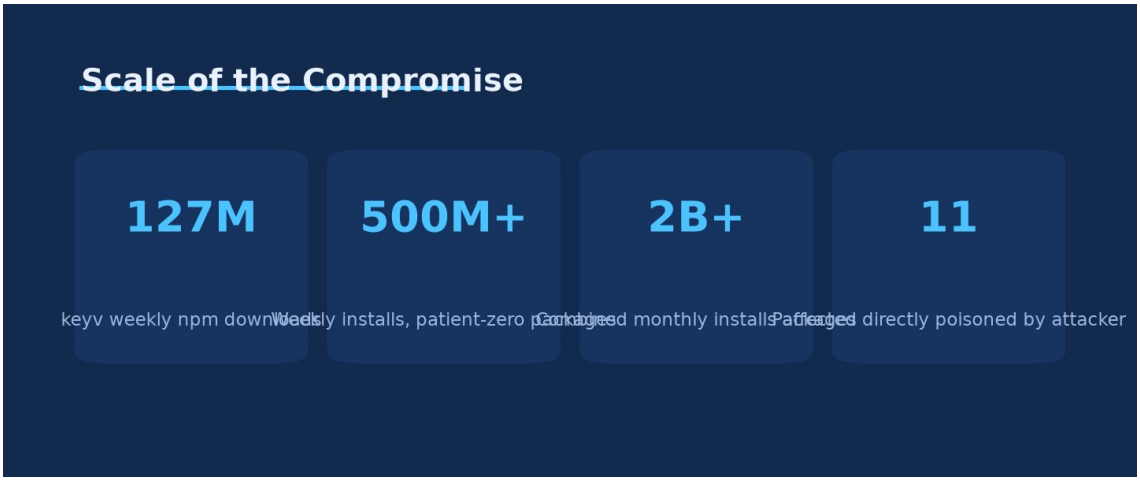
The Keyv/Cacheable npm Worm: A Confirmed "Mini Shai-Hulud" Descendant That Backdoors Claude Code and VS Code — A Defender's Technical Report

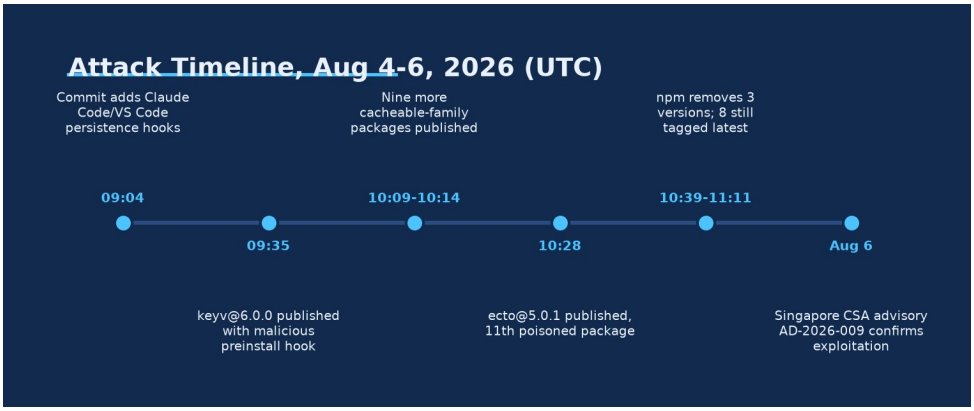
A researched, cited report · Ask Anything

Contents



Scale of the Compromise - Source: Ask Anything analysis — generated from the report's cited data

Executive summary



Attack Timeline, Aug 4-6, 2026 (UTC) - Source: Ask Anything analysis — generated from the report's cited data

On August 4, 2026, an attacker who had taken over the GitHub account of the maintainer behind keyv, cacheable, flat-cache, file-entry-cache, cache-manager, and several sibling caching libraries pushed a malicious preinstall hook straight to main and cut new releases within the hour. Independent malware-lineage analysis from Wiz concluded the payload is "a descendant of the 'Mini' Shai-Hulud malware family," sharing propagation logic, obfuscation, and dead-drop exfiltration patterns with the April–May 2026 TeamPCP/TanStack campaign, but now carrying an Ethereum smart-contract command-and-control (C2) channel and, for the first time in this lineage, persistence hooks committed directly into .claude/settings.json and .vscode/tasks.json (Wiz, 2026).

This is not a single-package incident. keyv alone carries roughly 127 million weekly (≈604 million monthly) npm downloads, and the maintainer's sibling packages add hundreds of millions more, giving the initial "patient zero" set a combined footprint north of 500 million weekly installs before any worm propagation began (Wiz, 2026; JFrog Security Research, 2026). Once the malware activated, it used harvested npm tokens to republish

trojanized versions of every other package those tokens could write to, and multiple independent vendors (Aikido, JFrog, SafeDep, Socket, Snyk) converged — with numbers that still disagree by tracker and timestamp — on a range of roughly 400 to over 2,200 poisoned package versions across several hundred package names and at least nine downstream npm organizations, with a combined dependency footprint exceeding two billion monthly installs (Datadog Security Labs, 2026; SafeDep, 2026).

The underlying weakness is not one bug but a stack of trust assumptions that each fail gracefully into the next. First, npm's lifecycle-script model still allows a package to run arbitrary code the instant it is installed, before a human ever imports it — a design decision the npm and GitHub teams only began reversing with npm 12's "scripts off by default" change in mid-2026 (GitHub Changelog, 2026-07-08). Second, npm's cryptographic provenance system (Sigstore/SLSA via GitHub Actions OIDC) verifies who built an artifact and from what workflow, but says nothing about whether the source repository itself was trustworthy at build time — so a compromised maintainer account produces a validly signed, "verified," yet fully malicious release (Snyk, 2026). Third, and most novel for this wave, the payload treats AI coding-agent and editor configuration as a second, independent execution surface: files like .claude/settings.json (a SessionStart hook) and .vscode/tasks.json (a runOn: folderOpen task) are committed straight into the source tree, so the backdoor survives npm uninstall, credential rotation, and even a full machine wipe — it simply reappears the next time anyone clones the repository and opens it in Claude Code or VS Code (hard2bit, 2026; Datadog Security Labs, "Before the first prompt," 2026).

Evidence that this is being actively exploited, not merely a proof of concept, is unusually well corroborated for a fast-moving incident: Singapore's Cyber Security Agency issued a formal advisory confirming active exploitation (CSA Singapore, AD-2026-009); npm's own registry removed some but not all malicious releases within the first two hours, with eight still tagged latest at 11:16 UTC in Snyk's snapshot (Snyk, 2026); package-scoped organizations including Deliveroo, Qlik, Picsart, OneReach, and ServiceTitan were publicly confirmed to have had scoped packages swept into the propagation (DevOps.com, 2026); and defenders can independently verify spread by searching GitHub for the hundreds of newly created public repositories carrying the description string "Shai-Hulud: Here We Go Again" that the malware itself creates as a dead-drop (Socket.dev, 2026; JFrog Security Research, 2026).

This report separates what is independently verifiable (file hashes, timestamps, package/version lists, the GitHub dead-drop pattern) from vendor interpretation (malware-family attribution, campaign scale estimates, which are still moving targets hours-to-days into the incident). No CVE or GHSA identifier had been formally assigned to the August 2026 keyv/cacheable wave at the time the primary technical write-ups (Snyk, JFrog) were published; Snyk tracked it internally as SNYK-JS-KEYV-18515941 under CWE-506 (embedded malicious code) (Snyk, 2026).

For defenders, the practical priority order is: (1) hunt for and disable the host-level persistence watcher before rotating credentials, since revocation is its trigger; (2) pin/override the eleven "patient zero" packages to their pre-compromise versions; (3) inspect any recently cloned repository for .claude/settings.json and .vscode/tasks.json before opening it in an agent or editor; and (4) treat any host or CI runner that executed the payload as compromised and rebuild rather than remediate in place (SafeDep, 2026; JFrog Security Research, 2026).

Key findings

Diverging Tracker Counts of Poisoned Packages

Tracker	Versions	Names/Orgs
SafeDep	1,684	420 names, 9 orgs
Aikido	1,381	868 packages
JFrog	1,700+	400+ packages
Other trackers (peak)	2,234	444 names

Diverging Tracker Counts of Poisoned Packages - Source: Ask Anything analysis — generated from the report's cited data

- The August 4, 2026 wave began with a compromised GitHub maintainer account (jaredwray), not a code vulnerability in keyv/cacheable itself; the initial access vector into that account remains undisclosed and unconfirmed as of this report [strong] (Chainguard, 2026).

- Independent malware analysis (Wiz) classifies the payload as a descendant of the "Mini" Shai-Hulud family, evolved with Ethereum smart-contract C2 ("EtherHiding"-style), a new RSA exfiltration key, and — new to this lineage — committed persistence hooks targeting Claude Code and VS Code [strong] (Wiz, 2026).

- Eleven packages maintained by the same account were directly poisoned between 09:35 and 10:28 UTC on August 4, 2026, with byte-identical payload files across all retrievable tarballs, indicating a single build/injection process rather than independent compromises [strong] (Snyk, 2026).

- The malicious releases carried valid npm provenance and GitHub Actions OIDC/Sigstore attestation because the underlying build pipeline itself was legitimate — provenance proved build origin, not source trustworthiness [strong] (Snyk, 2026).

- Self-propagation used harvested npm tokens to auto-publish trojanized patch releases of every package a stolen token could write to, requiring the token to have bypass_2fa === true and write permission — precisely the token class npm/GitHub began

deprecating in August 2026 [strong] (JFrog Security Research, 2026; GitHub Changelog, 2026-07-08).

- Tracker counts diverge substantially and are still moving: SafeDep reported 1,684 poisoned versions/420 names/9 orgs; Aikido reported at least 868 packages/1,381 versions; other trackers cited peaks near 2,234 versions/444 names — these are artifact counts, not confirmed compromised endpoints [moderate] (SafeDep, 2026; Aikido, 2026).

- The Claude Code hook (.claude/settings.json, SessionStart) and VS Code hook (.vscode/tasks.json, runOn: folderOpen) are committed to the git repository itself, not just installed into node_modules, so they persist across dependency reinstalls, credential rotation, and machine rebuilds until the repository history is cleaned [strong] (hard2bit, 2026; Wiz, 2026).

- VS Code's workspace-trust model prompts before running folderOpen tasks in untrusted workspaces, and Claude Code applies workspace trust to project-level settings, which limits — but by design does not eliminate — automatic execution risk for developers who accept trust prompts reflexively [moderate] (Wiz, 2026).

- A related, independently documented Claude Code hook weakness (tracked separately as CVE-2025-59536 and discussed in the CVE-2026-21852 write-up) shows that some coding-agent execution paths — including enableAllProjectMcpServers: true and environment-variable manipulation — could historically trigger before any trust-confirmation dialog appears, a distinct but reinforcing weakness from the npm supply-chain vector [strong] (Check Point Research, 2026).

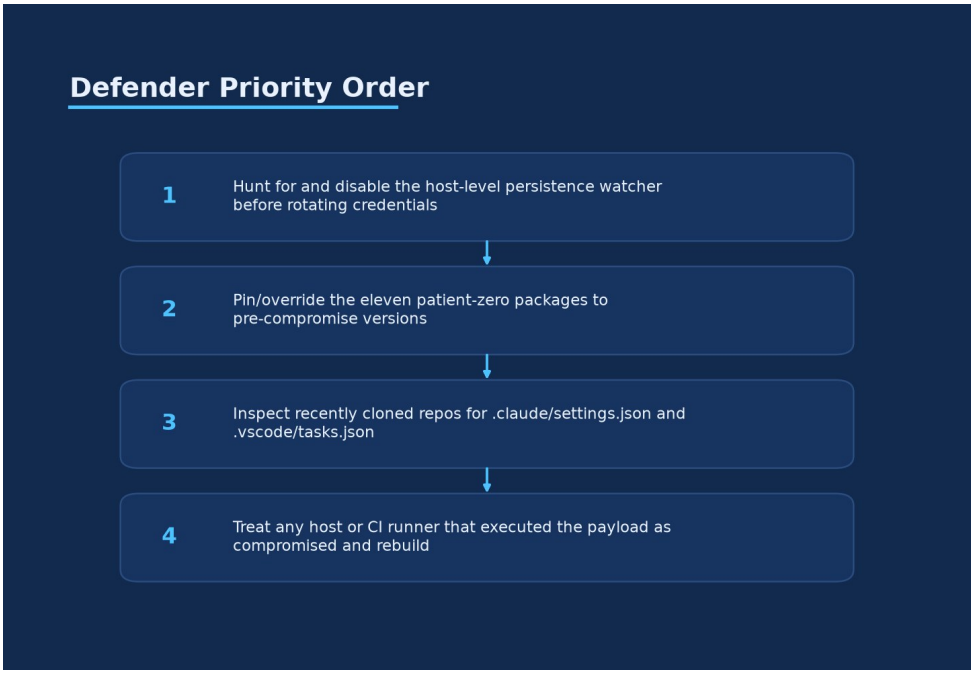
- C2 resiliency was engineered via an Ethereum mainnet smart contract (0xE1f2395ee43e45A1556EC6438a88c31B83493103) that returns HTTPS C2 domains on eth_call, making the C2 pointer effectively un-seizable through conventional domain/IP takedown [strong] (Upwind, 2026).

- Government confirmation of active exploitation exists (Singapore CSA, August 6, 2026); no U.S. CISA advisory specific to the August 2026 wave was located as of this report, though CISA's September 2025 alert covers the broader Shai-Hulud lineage [moderate] (CSA Singapore, 2026; CISA, 2025).

- Community open-source detection tooling (Cobenian's shai-hulud-detect) was updated within the incident window to cover the keyv/cacheable wave's package list and hashes, giving defenders a free, auditable scanning option [strong] (Cobenian/shai-hulud-detect, GitHub, 2026).

- No formal CVE/GHSA identifier had been assigned to the keyv/cacheable compromise at the time of the primary vendor technical analyses; Snyk's internal tracking ID (SNYK-JS-KEYV-18515941) is the closest thing to a stable reference identifier available [moderate] (Snyk, 2026).

1. What actually happened, in sequence



Defender Priority Order - Source: Ask Anything analysis — generated from the report's cited data

The timeline below is reconstructed from GitHub commit timestamps, npm publish timestamps, and researcher observation logs, primarily cross-checked between Snyk's forensic write-up and JFrog's technical analysis. All times are UTC on August 4, 2026 unless noted.

09:02–09:17

Commit ee2681a9 adds the lifecycle hook, payload files, and a test invocation to the keyv repository main branch (Snyk, 2026)

09:04

A GitHub-verified commit d8c850c7 adds the Claude Code and VS Code persistence hooks (Snyk, 2026)

09:23

Commit f97eabcd removes the earlier test invocation — read by researchers as deliberate concealment after in-repo validation (Snyk, 2026)

09:35

keyv@6.0.0 published to npm with the malicious preinstall hook active (Snyk, 2026)

10:09–10:14

Nine additional cacheable-family packages published with the same payload pattern (Snyk, 2026)

10:18–10:20

Public warning posted by researcher Charlie Eriksen (Aikido) (Wiz, 2026)

10:28

ecto@5.0.1, an eleventh package from the same account, published with an identical payload (Snyk, 2026)

10:39–11:11
npm removes three malicious versions; eight remain tagged latest in Snyk's 11:16 UTC snapshot (Snyk, 2026)

13:15–13:20 CEST
Aikido reports at least 444–868 packages across 1,381 versions compromised by community/worm spread, combined footprint over 2 billion monthly installs (Aikido, 2026)

Later Aug 4–5
SafeDep expands the registry-backed count to 1,684 poisoned versions across 420 package names tied to nine organizations; other trackers report peaks up to 2,234 versions/444 names (SafeDep, 2026)

Aug 5, 13:15 CEST
JFrog reports 400+ packages/1,700+ versions compromised, still climbing (JFrog Security Research, 2026)

Aug 6
Singapore's Cyber Security Agency issues advisory AD-2026-009 confirming active exploitation (CSA Singapore, 2026)

Two structural details matter for incident responders. First, the poisoned releases had valid npm provenance and GitHub Actions OIDC/Sigstore attestation, because they were produced by the maintainer's own (compromised) automated release workflow — this is not a spoofed-signature attack, it is a legitimately signed malicious build (Snyk, 2026). Second, restoring latest to a clean version at the registry did not fully remediate the source: the payload files were staged across all 19 packages in the keyv monorepo's git tree, so a fresh release cut later from that same tree — even by the legitimate maintainer, post-recovery — could still carry the payload unless the git history itself was scrubbed (Chainguard, 2026).

2. The underlying weakness — why this class of attack keeps working

Framing this as "an npm bug" undersells it. Four separate, individually reasonable design decisions compound into the attack surface this worm exploited.

(a) Lifecycle scripts execute before human review. npm's preinstall, install, and postinstall hooks in package.json run automatically the moment a package is installed — before a developer imports a single line of it, and often before any code review has occurred. This has been npm's default behavior since its earliest versions specifically to support native module compilation (node-gyp builds), but it means installing a dependency is equivalent to executing arbitrary code from anyone who can publish under that package name. JFrog's analysis notes that "on npm 12 or newer, preinstall lifecycle hooks do not run by default," which is true — but npm 12 only reached general availability in mid-2026, meaning the vast majority of installed developer and CI

toolchains at the time of this attack still ran scripts automatically (JFrog Security Research, 2026; GitHub Changelog, 2026-07-08).

(b) Provenance proves origin, not trustworthiness. npm's provenance system, built on Sigstore and SLSA via GitHub Actions OIDC, cryptographically attests that a specific CI workflow, at a specific commit, produced a specific artifact. It is a genuinely valuable control against typosquatting, registry-side tampering, and unsigned "who even published this" ambiguity. But as Snyk's analysis puts it, "provenance can faithfully attest a build whose source or workflow context has already been compromised" (Snyk, 2026). Once an attacker controls the maintainer's GitHub account, every subsequent commit and release inherits full cryptographic legitimacy — provenance was never designed to answer "was the human who pushed this commit actually authorized to."

(c) Token scope and 2FA-bypass paths outrun their own safeguards. The worm's self-propagation logic specifically checks harvested npm tokens for bypass_2fa === true combined with package write permission before using them to auto-publish infected releases (JFrog Security Research, 2026). These 2FA-bypass Granular Access Tokens (GATs) exist to let CI pipelines publish without an interactive 2FA prompt — a legitimate operational need — but any token with that combination of properties becomes a skeleton key the moment it's exfiltrated. npm and GitHub had already announced a phased deprecation of exactly this token class before the attack: Phase 1 (early August 2026) strips bypass-2FA tokens of the ability to perform sensitive account/package/org management actions, and Phase 2 (around January 2027) removes their ability to publish directly at all, requiring a human 2FA-approved staged publish instead (GitHub Changelog, 2026-07-08). The August 4 attack landed squarely inside the window where this exact weakness was known, publicly announced as being retired, but still live.

(d) Coding-agent and editor "project settings" is a second execution surface with weaker review norms than node_modules. This is the piece that makes this wave distinct from earlier Shai-Hulud generations. Security teams have spent years building tooling to scan node_modules and lockfiles for malicious lifecycle scripts. Almost none of that tooling inspects .claude/settings.json or .vscode/tasks.json — files developers assume are inert configuration, not executable surface. Datadog Security Labs' parallel research into coding-agent trust models describes the general class of problem bluntly: "Trusting a repository in a coding agent can allow repository-controlled code to run before you send the first prompt" (Datadog Security Labs, "Before the first prompt," 2026). In Claude Code specifically, a SessionStart hook defined in .claude/settings.json fires when the agent initializes in a directory containing that file; in VS Code, a task with runOn: "folderOpen" in .vscode/tasks.json fires when the folder is opened, subject to workspace-trust prompts. Because these files are committed to git rather than installed by npm, they survive rm -rf node_modules, npm ci, and even credential rotation — the trigger travels with the source tree to every future clone (hard2bit, 2026).

It's also worth separating this npm-worm-driven Claude Code exposure from a related but

distinct vulnerability class: independent security research (Check Point, tracked as CVE-2025-59536, with a follow-on advisory referenced as CVE-2026-21852) previously found that certain Claude Code project-file configurations — notably enableAllProjectMcpServers: true — could trigger code execution before the workspace-trust confirmation dialog appeared at all, and that environment-variable manipulation via project settings could exfiltrate API tokens (Check Point Research, 2026). The August keyv worm does not need that specific bypass — it relies on developers accepting a workspace-trust prompt they see routinely — but the existence of a documented pre-trust execution path in the same product family means defenders should not treat "workspace trust" as a complete control boundary.

3. Malware architecture (defensive-relevant detail only)

This section describes what the payload does at a level sufficient for detection and response; it deliberately omits the obfuscation cipher, C2 protocol byte layout, and any code that would materially assist reproduction.

Delivery. The preinstall hook runs node setup.mjs, a ~29.9 KB loader. The loader checks for an available Bun JavaScript runtime and, if absent, downloads Bun 1.3.13 directly from github.com/oven-sh/bun/releases — a legitimate binary from a legitimate source, chosen specifically because it is unlikely to be flagged by allow-listing tools that only look for "suspicious" executables (JFrog Security Research, 2026; Snyk, 2026).

Main payload. Bun executes a ~710–728 KB second-stage file (named Math_Symbol.js in the original packages, math_init.js in worm-propagated copies). Snyk was not able to fully decrypt this stage; JFrog's independent analysis describes its logic as: most internal strings obfuscated with "a custom position-dependent substitution cipher derived from PBKDF2-HMAC-SHA256," self-relaunching detached from the parent process with _NODE_RUNTIME_INIT=1, and holding a PID lock file to prevent concurrent execution (JFrog Security Research, 2026).

Credential harvesting scope. The collector reads several hundred filesystem glob patterns covering: npm/Yarn/PyPI/Verdaccio registry tokens; AWS/Azure/GCP/Alibaba/Tencent/Hetzner cloud configuration; Kubernetes and Helm configs, SSH keys, VPN profiles; HashiCorp Vault tokens (from six distinct source locations); Stripe and Slack API key patterns; cryptocurrency wallet files (Foundry, Solana, Monero); database connection strings and Terraform state; AI-tool credentials (OpenAI, Anthropic/Claude, Cursor, Codex, Gemini configuration); and, where reachable, /etc/shadow and /proc/self/environ (JFrog Security Research, 2026; Datadog Security Labs, 2026). On GitHub Actions runners specifically, the malware has been observed invoking sudo python3 to read the Runner.Worker process's memory via /proc/<pid>/mem in search of in-memory secret-store objects — a technique that extracts secrets the runner never wrote to disk (JFrog Security Research, 2026).

Exfiltration. Results are JSON-serialized, gzip-compressed, and encrypted with an AES-256-GCM key wrapped by an embedded RSA public key. Primary exfiltration is to a dynamic HTTPS C2 endpoint; a resilient fallback publishes the encrypted blob to a newly created public GitHub repository under the victim's own account, using Dune-novel-themed naming (sardaukar, fremen, atreides, sandworm, melange) or the repository description string "Shai-Hulud: Here We Go Again" (JFrog Security Research, 2026; Datadog Security Labs, 2026). Because these dead-drop repositories are public by design (the malware needs the attacker to be able to read them without authenticating as the victim), they are also the single most independently verifiable piece of evidence available to defenders and researchers — anyone can search GitHub for that description string.

Command and control. The C2 domain is not hardcoded. The malware calls eth_call against Ethereum mainnet contract 0xE1f2395ee43e45A1556EC6438a88c31B83493103 using function selector 0x53ed5143; the contract's stored string data decodes to one or more current C2 domains, which are then contacted over HTTPS on a /router path (Upwind, 2026; JFrog Security Research, 2026). This "EtherHiding"-style technique — storing attacker infrastructure pointers in immutable, censorship-resistant blockchain state rather than DNS — means a conventional domain seizure or registrar takedown does not stop the campaign; the operator can simply update the contract's return value (if they retain the owning key) to point at new infrastructure without touching the malware itself. Observed supporting RPC endpoints used to reach the contract include eth-mainnet.nodereal.io, go.getblock.io, and eth.llamarpc.com (Wiz, 2026). Once a C2 response containing a code field is received, it is executed via eval, giving the operator full remote code execution inside the already-compromised process (JFrog Security Research, 2026).

npm worm propagation. For each npm token the credential harvester finds, the malware validates it (requiring bypass_2fa === true and package write access), then for every package that token can write to: downloads the current tarball, injects the loader/payload pair, adds or rewrites the preinstall script, bumps the patch version, and republishes via an authenticated registry PUT. JFrog summarizes the effect precisely: "one stolen npm token with broad write access can turn every package owned by that publisher into the next wave" (JFrog Security Research, 2026). Socket.dev's telemetry measured propagation moving between organizations roughly every two to seven minutes, with a full cross-organization publishing cycle completing in about 30 minutes (Aikido, 2026).

GitHub-side propagation. Where GitHub tokens (including gh auth token-retrieved PATs and ghs_-prefixed GitHub App server-to-server tokens) are available, the malware pushes the persistence hooks and payload files as commits to up to 50 branches per repository — deliberately skipping dependabot/ and copilot/ branches, likely to avoid tripping automated dependency-bot review — using the commit message "chore: update config" and a forged Co-authored-by: claude <claude@users.noreply.github.com> trailer

designed to make the change look like ordinary AI-assisted maintenance (JFrog Security Research, 2026; Datadog Security Labs, 2026).

Host persistence (dead-man's switch). A separate, partially-observed component — gh-token-monitor — installs as a macOS LaunchAgent (com.user.gh-token-monitor) or Linux systemd user service, polling stolen GitHub tokens roughly every 60 seconds for 24 hours and executing a local handler if a token is detected as revoked (JFrog Security Research, 2026; Socket.dev, 2026). This is operationally critical: SafeDep's guidance explicitly warns that "revocation is the watcher's trigger; rotating first can run an attacker-supplied local handler," meaning incident responders should locate and disable this persistence mechanism before rotating the credentials it is watching (SafeDep, 2026).

A targeted supply-chain sub-attack. JFrog also documented a special-cased attack path aimed specifically at OpenSearch's JavaScript release workflow: when GITHUB_WORKFLOW_REF contains release-drafter.yml and GITHUB_REPOSITORY contains /opensearch-js, the malware requests a GitHub Actions OIDC token scoped to npm:registry.npmjs.org, adds malicious optionalDependencies entries, exchanges the OIDC token for npm publish credentials, and generates a Sigstore bundle through Fulcio and Rekor — producing a malicious tarball with fully valid provenance (JFrog Security Research, 2026). This shows the malware author had specific, pre-researched knowledge of at least one high-value downstream CI configuration, not just generic opportunism.

4. Affected and fixed version matrix

Directly poisoned "patient zero" packages (same maintainer account), published 09:35–10:28 UTC, August 4, 2026:

Package	Malicious version	Known-clean / rollback version	Approx. monthly downloads
keyv	6.0.0	5.6.0 (or 6.0.0-rc.1, pre-payload, if the 6.x line is required)	~604–620M
flat-cache	6.1.24	6.1.23	~580M
file-entry-cache	11.1.6 (some trackers log 11.1.7)	11.1.5	~571M
cacheable-request	13.0.20	13.0.19	~137M
@cacheable/utils	2.5.1	2.5.0	~34M
cacheable	2.5.1	2.5.0	~30M
@cacheable/memory	2.2.1	2.2.0	~28M
cache-manager	7.2.10	7.2.9	~16M
@cacheable/node-cache	3.1.2	3.1.1	~6M
@cacheable/net	2.1.1	2.1.0	~3.7K
ecto	5.0.1	5.0.0	~4.5K

Sources cross-verified across Kodem Security, 2026, Snyk, 2026, Socket.dev, 2026, and Datadog Security Labs, 2026. The file-entry-cache version discrepancy (11.1.6 vs. 11.1.7)

across sources likely reflects two rapid point releases; defenders should treat both as untrusted and pin to 11.1.5.

Secondary / community-spread wave (worm propagation to other maintainers' packages): Reported counts range from roughly 400 packages (early JFrog/Wiz snapshots) to 420–444 package names / 1,684–2,234 versions across at least nine npm organizations in later SafeDep and other tracker snapshots — the number was still climbing at the time each source was published, so treat any single figure as a lower bound at that timestamp, not a final tally (SafeDep, 2026; JFrog Security Research, 2026). Notable scoped packages/organizations publicly confirmed as swept into propagation include the @keyv/ backend-adapter family (redis, postgres, mongo, sqlite, memcache), @nebula.js/ visualization components tied to Qlik, @onereach/ (400+ packages), @servicetitan/ (500+ packages), @deliveroo/reevent, and @picsart/ai-sdk (Datadog Security Labs, 2026; DevOps.com, 2026). At least one Go module mirror, github.com/jaredwray/keyv, was also affected from v6.0.1 onward (JFrog Security Research, 2026).

Genealogy — how this wave relates to earlier Shai-Hulud generations (distinct file hashes; do not conflate):

Original Shai-Hulud

Approx. date: Sept 15, 2025

Distinguishing loader files: post-install trigger, bundle.js

Notable evolution: First worm to self-propagate via stolen npm tokens; targeted @ctrl/tinycolor and hundreds of others (ReversingLabs, 2025; CISA, 2025)

Shai-Hulud 2.0

Approx. date: Nov 2025

Distinguishing loader files: setup_bun.js, bun_environment.js

Notable evolution: Moved trigger to pre-install (guarantees execution on build servers without any human interaction); ~25,000 malicious repos, ~350 users affected (Datadog Security Labs, 2026; Unit42, 2026)

Mini Shai-Hulud / TeamPCP (TanStack)

Approx. date: May 11, 2026

Distinguishing loader files: CI cache-poisoning + OIDC runner-memory extraction

Notable evolution: First wave with fully valid SLSA provenance via GitHub Actions "pwn request" + cache poisoning; 84 malicious versions across 42 @tanstack/* packages plus Mistral AI, OpenSearch, PyPI packages, ~170 packages total; tracked as CVE-2026-45321 (CVSS 9.6) (StepSecurity, 2026; lilting.ch, 2026)

keyv/cacheable ("ChainDrop")

Approx. date: Aug 4, 2026

Distinguishing loader files: setup.mjs, Math_Symbol.js/math_init.js

Notable evolution: Ethereum smart-contract C2, RSA-4096 signed fallback, and first confirmed persistence hooks in .claude/settings.json and .vscode/tasks.json (Wiz, 2026; BleepingComputer, 2026)

Note: several outlets (StepSecurity, BleepingComputer) refer to the August wave by the alternate name "ChainDrop"; this is a vendor-naming variant for the same campaign Wiz and JFrog describe as a Mini Shai-Hulud descendant, not a separate incident.

5. Is it being exploited in the wild, and how widely? What defenders can verify themselves

The evidence for active, real-world exploitation is stronger than for a typical "researcher found a malicious package" disclosure, because it is corroborated by multiple independent parties using different telemetry, and because parts of the malware's own design (the public GitHub dead-drop) are independently checkable by any defender with a browser.

Independently checkable evidence:

- Public GitHub dead-drop repositories. The malware itself creates public repositories with the description "Shai-Hulud: Here We Go Again" to stage exfiltrated, encrypted data. Multiple vendors independently counted these — figures cited range from roughly 546 to approximately 1,300 such repositories observed on August 4–5, 2026 — and any defender can reproduce this by searching GitHub for that exact description string (Datadog Security Labs, 2026; JFrog Security Research, 2026).

- npm registry's own partial response. Snyk's timestamped snapshots show npm removing three malicious versions between 10:39 and 11:11 UTC on August 4, while eight remained tagged latest at 11:16 UTC — a registry-side action log that is itself evidence the registry operator treated this as a live, active compromise, not a theoretical risk (Snyk, 2026).

- Named downstream organizations. Packages scoped to Deliveroo, Qlik (via @nebula.js/*), Picsart, OneReach, and ServiceTitan were publicly reported as caught in the propagation — these are organization-scoped npm namespaces, which only become infected if a maintainer or CI credential tied to that real organization was compromised and used to publish (DevOps.com, 2026).

- Government advisory. Singapore's Cyber Security Agency published a dedicated advisory (AD-2026-009) on August 6, 2026, describing the campaign as an "ongoing" attack and confirming it "steals developer credentials and spreads by compromising additional packages" — official-sector confirmation independent of any single vendor's commercial incentive to overstate scale (CSA Singapore, 2026).

- Cross-vendor telemetry convergence. JFrog, Wiz, Aikido, Snyk, SafeDep, and Socket.dev — commercial competitors with independent scanning infrastructure — published overlapping (though not identical) package/version counts and identical file hashes within roughly 24–48 hours of each other, which is strong corroborating evidence that the artifacts are real and actively spreading, even though the vendors disagree on exact totals.

What is not independently verifiable, and should be treated as estimate/interpretation: the true number of developer machines or CI runners that actually executed the payload (as opposed to having a poisoned package merely present in a lockfile); the total volume or content of exfiltrated credentials; and whether any specific downstream breach (cloud account takeover, source code theft) has resulted, since none of the reviewed sources published a confirmed post-exploitation impact case study by report time. Wiz and SafeDep both explicitly flag this distinction — package/version/repository counts measure "poisoned artifacts," not "confirmed victims" (SafeDep, 2026).

Attribution remains unresolved. No named threat actor has publicly claimed the August 2026 wave. Researchers note code-lineage and technique overlap with the TeamPCP-attributed May 2026 TanStack campaign, but this is an inference from shared tooling and obfuscation style, not a confirmed common operator (Chainguard, 2026).

6. Detection guidance

File and hash indicators (August 2026 keyv/cacheable wave — do not confuse with earlier-generation hashes below):

setup.mjs (npm package copy)
SHA-256: 54dc7ea54a1317cca0e890a2770630cf7fa6c97813e0cb9d2caa93012b350668
Context: Present in poisoned tarballs (Snyk, 2026)

setup.mjs (worm-propagated / .claude, .vscode copies)
SHA-256: fd3ca4007b225fdf8de7af4345a19179d5efa8c4bb9205f88cda806e5684b1eb
Context: Later-wave and IDE-hook variant (JFrog Security Research, 2026; Socket.dev, 2026)

Math_Symbol.js / math_init.js (second stage, 727,680 bytes)
SHA-256: 9fc2570b7cef51c1b8df116d144d11ff4096357be7d2c4c6367cfc2509cf1bcc
Context: Community-spread copies renamed math_init.js (JFrog Security Research, 2026; Datadog Security Labs, 2026)

For context, the prior Shai-Hulud 2.0 generation (November 2025) used entirely different filenames and hashes — setup_bun.js (a3894003ad1d293ba96d77881ccd2071446dc3f65f434669b49b3da92421901a) and bun_environment.js (multiple observed hashes including 62ee164b9b306250c1172583f138c9614139264f889fa99614903c12755468d0) — so hash lists must be versioned by campaign wave, not merged into one undifferentiated blocklist (Datadog Security Labs, 2026).

Non-executing repository/lockfile inspection commands (safe to run; do not execute installed scripts):

```
# Flag any package.json declaring the malicious preinstall pattern, without running it
find node_modules -name package.json -print0 | xargs -0 grep -l "preinstall.*setup.mjs"

# Hunt filesystem persistence artifacts
find "$HOME" /tmp -name 'gh-token-monitor.*' -o -name 'Math_Symbol.js' -o -name 'math_init.js'
```

```
# Check a cloned repository for committed IDE/agent persistence hooks before opening it
grep -r "SessionStart" .claude/ 2>/dev/null
grep -r "runOn.*folderOpen" .vscode/ 2>/dev/null
```

(Detection pattern set adapted from Snyk, 2026 and MintMCP, 2026.)

Additional Claude Code-specific configuration review patterns worth grepping for in any recently cloned or forked repository, per independent Claude Code hook-security research (not necessarily tied to this specific worm, but the same review discipline applies):

```
grep -r "ANTHROPIC_BASE_URL" .
grep -r "enableAllProjectMcpServers.*true" .
grep -rE "SessionStart.*(curl|wget)" .
```

(MintMCP, 2026)

Network indicators:

- Domain npm-cache[.]com (Cloudflare-fronted, observed resolving to 104.21.35.216), contacted over HTTPS on path /router; also observed sibling domains pypi-get[.]com and js-mirror[.]com in related infrastructure (Wiz, 2026; JFrog Security Research, 2026).

- Outbound eth_call traffic to Ethereum RPC providers (eth-mainnet.nodereal.io, go.getblock.io, eth.llamarpc.com) from build agents or developer workstations that have no legitimate blockchain-development purpose — a strong anomaly signal precisely because it is an unusual protocol for a JavaScript build pipeline to speak (Wiz, 2026).

- User-Agent string Bun/1.3.13 originating from hosts or containers that have no declared Bun dependency (Kodem Security, 2026).

- Unexpected downloads from github.com/oven-sh/bun/releases/download/bun-v1.3.13/ on hosts that do not normally fetch release binaries during npm install (Kodem Security, 2026).

- Cloud metadata service queries (169.254.169.254, 169.254.170.2) originating from a Node.js/Bun install process rather than an application's own SDK — a classic tell for IMDS credential-scraping tooling piggybacking inside a build step (Socket.dev, 2026).

- Anomalous npm publish/OIDC token-exchange activity at registry.npmjs.org/-/npm/v1/oidc/token/exchange/package/ outside normal release cadence, or version bumps that do not correspond to any human-authored release PR (Kodem Security, 2026).

Git/GitHub behavioral signals:

- New public repositories with the exact description "Shai-Hulud: Here We Go Again" appearing under any account or organization you monitor (JFrog Security Research, 2026).

- Commits with message "chore: update config" carrying a forged Co-authored-by: claude <claude@users.noreply.github.com> trailer (JFrog Security Research, 2026).

- A surge of pushes across many branches (up to 50 per repository) in a short window, specifically excluding dependabot/ and copilot/ branches (JFrog Security Research, 2026).

- Unexpected new workflow file .github/workflows/codeql_analysis.yml appearing without a corresponding human PR (JFrog Security Research, 2026).

- Remember that a "GitHub verified" badge on a commit only confirms the signing path was valid (e.g., that GitHub Actions' bot identity or the account's configured signing key produced the signature) — it does not confirm the human behind the account authorized the change. Treat "verified" and "authorized" as separate claims during investigation (Datadog Security Labs, 2026).

Published detection tooling:

- Cobenian/shai-hulud-detect (open source, GitHub) — a filesystem/dependency scanner that cross-references installed package versions against a running list of confirmed-compromised versions (5,699+ entries spanning npm, PyPI, Composer, Crates, Go, Hex, and RubyGems as of the tool's own documentation), hashes priority files, and greps for known IOC strings; the maintainers updated it to cover the August 2026 keyv/cacheable wave, flagging it as the largest wave by install volume to date (Cobenian/shai-hulud-detect, GitHub, 2026). Note its Claude Code/VS Code coverage is described as partial — it detects a related May 2026 payload targeting ~/.claude/settings.json but does not document a dedicated signature for this wave's specific IDE hook content, so pair it with the manual grep patterns above.

- Commercial SCA/dependency-scanning platforms (Aikido, Snyk, Socket, JFrog Xray, SafeDep, Mend, Endor Labs) all published hash- and version-based detection rules within hours of disclosure and are actively maintaining live tracking pages; several (Aikido) score the finding as a maximum-severity "100/100 critical malware" issue with nightly automatic rescans (Datadog Security Labs, 2026).

- YARA/Sigma coverage. Multiple vendors (Amazon Inspector, Sysdig, ReversingLabs) maintain YARA rule libraries and Sigma detection content for the broader Shai-Hulud lineage, including rules for suspicious child-process spawning from Node/JS applications and creation of new public GitHub repositories matching the campaign's naming pattern; these are maintained as living rule sets rather than one-time signatures, so confirm your vendor's rule pack has been updated for the August 2026 wave specifically rather than assuming coverage from the original 2025 ruleset (Sysdig, 2026).

7. Interim mitigations and hardening for teams that cannot patch immediately

Ordering matters. SafeDep's guidance is explicit that credential rotation should not be the first step if the host-level watcher is still present, because triggering revocation can itself execute an attacker-supplied handler (SafeDep, 2026).

- Hunt and neutralize persistence before rotating credentials. Locate and disable

gh-token-monitor artifacts (~/.local/bin/gh-token-monitor.sh, the com.user.gh-token-monitor LaunchAgent, or the equivalent systemd user service) before invalidating any GitHub token those artifacts may be watching (JFrog Security Research, 2026; SafeDep, 2026).

- Pin and rebuild. Add explicit overrides/resolutions entries mapping every package in the version matrix above to its known-clean version, then reinstall with scripts disabled: npm install --package-lock-only --ignore-scripts, or on npm 12+, rely on the new default-off behavior while explicitly confirming no legacy allowScripts override re-enables it (Kodem Security, 2026; GitHub Changelog, 2026-07-08).

- Do not open recently cloned or forked repositories in an AI coding agent or editor without inspecting .claude/settings.json and .vscode/tasks.json first. Treat unfamiliar SessionStart hooks or runOn: folderOpen tasks as hostile until proven otherwise; treat the workspace-trust prompt as a genuine security decision rather than a habitual click-through (Chainguard, 2026; hard2bit, 2026).

- Constrain Claude Code's hook execution surface. Where supported, enable allowManagedHooksOnly: true to block project-level (repository-committed) hooks entirely, shifting hook execution to only centrally managed, IT-approved configuration rather than whatever a cloned repository happens to contain (Datadog Security Labs, "Before the first prompt," 2026). Avoid running Claude Code non-interactively (-p flag) against untrusted repositories, since non-interactive mode skips the trust prompt entirely.

- Rotate credentials from a clean, isolated system — not the potentially compromised host. Cover npm publish tokens (prioritizing any with bypass_2fa and write scope), GitHub PATs/OAuth/App tokens, cloud IAM keys (AWS/Azure/GCP), Kubernetes service-account tokens, HashiCorp Vault tokens, SSH keys, database connection strings, and any Stripe/Slack API keys that were reachable from an affected environment (JFrog Security Research, 2026; Datadog Security Labs, 2026).

- Migrate off 2FA-bypass publishing tokens now, not by the deprecation deadline. Move CI publishing to OIDC-based npm trusted publishing, or to a staged-publish workflow that requires a human 2FA approval before a package becomes public — this directly closes the propagation mechanism this worm (and its predecessors) rely on, and is arriving as a platform-enforced default in phases through January 2027 regardless (GitHub Changelog, 2026-07-08).

- Add a scanning/cooldown gate in front of your registry mirror. Route installs through a private registry proxy that applies malware scanning and a cooldown period (hours to a couple of days) before newly published versions become installable — this is a concrete, demonstrated defense-in-depth control (cited by Chainguard as having prevented exposure for its customers during this exact incident despite the upstream maintainer compromise) that does not require waiting on any single upstream maintainer's security posture (Chainguard, 2026).

- Constrain CI runner egress. Block or alert on outbound connections from build agents to non-registry domains, to Ethereum JSON-RPC endpoints, and to cloud metadata IP ranges from processes other than the expected cloud SDK — all three are unusual enough in a typical JS build pipeline to be low-noise, high-signal detections (Wiz, 2026; Socket.dev, 2026).

- Rebuild, do not remediate in place, any host or CI runner confirmed to have executed the payload. In-place cleanup cannot reliably prove the absence of a memory-resident or disk-persisted secondary implant given the malware's own dead-man's-switch and remote-eval capabilities (JFrog Security Research, 2026; Kodem Security, 2026).

8. Real-world examples defenders can independently trace

- The keyv/cacheable maintainer account itself — the confirmed origin point, with a fully reconstructed, timestamped commit-and-publish sequence available via GitHub's public commit history and npm's public registry metadata (Snyk, 2026).

- @nebula.js/* packages tied to Qlik — an organization-scoped npm namespace publicly reported as swept into the propagation, illustrating how the worm moves from an individual maintainer's compromise into enterprise-owned scopes (Datadog Security Labs, 2026).

- @deliveroo/reevent and @picsart/ai-sdk — additional organization-scoped packages named in independent reporting as caught in the community-spread wave (DevOps.com, 2026).

- The OpenSearch JavaScript release-drafter.yml targeted sub-attack — a documented, code-level special case in the malware itself proving at least one specific high-value CI target was pre-researched rather than opportunistically hit (JFrog Security Research, 2026).

- The May 2026 TanStack/TeamPCP campaign — the immediately preceding "Mini Shai-Hulud" wave, useful as a comparison case for how the OIDC/CI-cache-poisoning sub-technique works when the worm targets trusted-publishing pipelines instead of maintainer accounts directly (StepSecurity, 2026).

- The November 2025 Shai-Hulud 2.0 wave — documented at ~25,000 malicious repositories across ~350 users, useful for benchmarking how much larger (in raw download footprint, if not repository count) the August 2026 wave is by comparison (Unit42, 2026).

- Composite scenario (illustrative, not a real incident report): a mid-size SaaS engineering team running npm install in a CI pipeline that still executes lifecycle scripts by default would have had Math_Symbol.js/math_init.js execute with the CI runner's full credential set — including any long-lived cloud deployment keys stored as environment variables — within seconds of a routine dependency-lockfile update pulling in a poisoned transitive version of keyv, illustrating why transitive (not just direct) dependency auditing is

required. This composite is derived from the documented attack mechanics above, not a specific reported case.

Practical synthesis

This incident applies to essentially any organization running Node.js-based CI/CD, since keyv and its sibling packages sit as deep transitive dependencies under widely used tooling (independent reporting specifically calls out transitive inclusion via tools like ESLint's dependency chain) (Socket.dev, 2026). The decision framework for a security or platform engineering team should be:

- If your lockfile resolves any package in the version matrix above, treat every host/runner that ran an install against that lockfile since August 4, 2026, 09:35 UTC as potentially compromised, and follow the persistence-hunt-then-rotate sequence in Section 7.

- If you use Claude Code or VS Code against externally sourced or forked repositories (open source contributions, vendor SDKs, contractor repositories), treat .claude/ and .vscode/ directories as untrusted executable surface by default, not configuration — this is a durable posture change, not just an incident-specific check, given the documented pre-trust execution issues found separately in Claude Code's hook system.

- If your CI still runs lifecycle scripts by default, prioritize the npm 12 upgrade / --ignore-scripts change over almost any other single control in this report — it closes the primary delivery mechanism for this entire malware lineage, not just this one wave.

- If your organization publishes npm packages, audit whether any publishing token in active use has bypass_2fa set, and migrate to OIDC trusted publishing or staged/human-approved publishing ahead of npm's own January 2027 enforcement deadline.

- What to measure going forward: time-to-detect from malicious publish to internal lockfile flag (target: minutes, achievable with SCA tooling that ingests registry-wide malware feeds); percentage of CI pipelines with lifecycle scripts disabled; percentage of publishing tokens with 2FA-bypass scope eliminated; and mean time to rebuild (not just patch) any host confirmed to have executed a malicious payload.

Evidence & caveats

Well established: the initial compromise vector (maintainer GitHub account), the timeline of malicious commits and publishes, the file hashes and package/version list for the eleven "patient zero" packages, the existence and mechanics of the .claude/settings.json/.vscode/tasks.json persistence hooks, the Ethereum smart-contract C2 mechanism, and the npm registry's own partial-removal timeline are all corroborated by multiple independent technical write-ups with matching artifacts (hashes, timestamps).

Disputed or still in flux: the total number of poisoned package versions and affected organizations — figures from credible trackers span roughly 400 to over 2,200 versions and were still increasing at each source's publication time; treat any single number in this report as a snapshot, not a final count. The exact file-entry-cache malicious version number (11.1.6 vs. 11.1.7) is inconsistently reported across otherwise-reliable sources.

Preliminary or vendor-interpreted: the "Mini Shai-Hulud descendant" classification and the inferred (not confirmed) link to the TeamPCP actor group rest on malware-lineage and tooling-overlap analysis by Wiz and others, which is standard threat-intelligence methodology but is attribution by pattern-matching, not a confirmed common operator identity.

Missing or not yet published: a formal CVE/GHSA identifier for the August 2026 wave; any confirmed downstream breach impact case study (cloud account takeover, data theft) tied to a specific named victim organization; a public statement from npm, GitHub, or Anthropic specifically addressing this wave (as distinct from Anthropic's earlier, unrelated March 2026 Claude Code source-map leak, and as distinct from Anthropic's general Claude Code hook-security hardening work referenced in Section 2); and confirmation of whether the Ethereum C2 contract has been neutralized or is still returning live infrastructure as of this report's writing.

Not generalizable: the specific file hashes, domains, and version numbers in this report apply only to the August 2026 keyv/cacheable wave. Applying them as a blocklist against earlier or later Shai-Hulud-lineage waves will produce false negatives, since each wave has used distinct filenames and hashes (see the genealogy table in Section 4).

Sources

[1] Major Shai Hulud campaign strikes npm again, affecting keyv and 400+ packages -- JFrog Security Research (2026) -- https://research.jfrog.com/post/shai-hulud-is-back-august/

[2] keyv and cacheable npm Package Hijacked in Supply Chain Attack -- Wiz Blog (2026) -- https://www.wiz.io/blog/keyv-and-cacheable-npm-supply-chain-attack

[3] keyv npm worm: credential theft and IDE auto-run hooks -- hard2bit (2026) -- https://hard2bit.com/en/blog/npm-worm-keyv-mini-shai-hulud-hooks-claude-code-vscode/

[4] keyv npm Supply Chain Attack | IOCs and Runbook -- Kodem Security (2026) -- https://www.kodemsecurity.com/resources/keyv-supply-chain-attack-shai-hulud-npm-worm-affected-versions-iocs-and-first-hour-response-runbook

[5] Inside the keyv npm Supply Chain Compromise -- Snyk (2026) -- https://snyk.io/blog/inside-keyv-npm-compromise-preinstall-malware-trusted-provenance-ide-hooks/ (advisory ID SNYK-JS-KEYV-18515941, CWE-506)

[6] Keyv-Linked npm Worm Poisons Hundreds of Packages, Plants Claude Code and VS Code Hooks -- The Hacker News (2026) --

https://thehackernews.com/2026/08/keyv-linked-npm-worm-poisons-hundreds.html
[7] Keyv and friends compromised in npm supply chain attack -- Aikido (2026) -- https://www.aikido.dev/blog/keyv-and-friends-compromised-in-npm-supply-chain-attack
[8] Mini Shai-Hulud Strikes Again: npm Worm Compromises Hundreds of @antv Packages -- Aikido (2026) -- https://www.aikido.dev/blog/mini-shai-hulud-antv-npm-supply-chain-attack
[9] Ongoing npm Supply Chain Attack Affecting Keyv and Related Packages ("Shai-Hulud" Worm), Advisory AD-2026-009 -- Cyber Security Agency of Singapore (2026) -- https://www.csa.gov.sg/alerts-and-advisories/advisories/ad-2026-009/
[10] Cobenian/shai-hulud-detect -- GitHub open-source detection tool (2026) -- https://github.com/Cobenian/shai-hulud-detect
[11] The Shai-Hulud 2.0 npm worm: analysis, and what you need to know -- Datadog Security Labs (2026) -- https://securitylabs.datadoghq.com/articles/shai-hulud-2.0-npm-worm/
[12] The keyv and cacheable npm Supply Chain Attack: Inside the Mini Shai-Hulud Campaign -- Chainguard Unchained (2026) -- https://www.chainguard.dev/unchained/the-keyv-and-cacheable-npm-supply-chain-attack-inside-the-mini-shai-hulud-campaign
[13] npm install-time security and GAT bypass2fa deprecation -- GitHub Changelog (2026-07-08) -- https://github.blog/changelog/2026-07-08-npm-install-time-security-and-gat-bypass2fa-deprecation/
[14] "Shai-Hulud" Worm Compromises npm Ecosystem in Supply Chain Attack (Updated November 26) -- Unit 42, Palo Alto Networks (2026) -- https://unit42.paloaltonetworks.com/npm-supply-chain-attack/
[15] Popular npm Packages in the keyv and Cacheable Namespaces Compromised in Active Supply Chain -- Socket.dev (2026) -- https://socket.dev/blog/popular-npm-packages-in-the-keyv-and-cacheable-namespaces-compromised-in-active-supply-chain
[16] SANDWORM_MODE: Shai-Hulud-Style npm Worm Hijacks CI Workflow / AI Toolchain Poisoning -- Socket.dev (2026) -- https://socket.dev/blog/sandworm-mode-npm-worm-ai-toolchain-poisoning
[17] Massive ChainDrop npm supply-chain attack infects hundreds of packages -- BleepingComputer (2026) -- https://www.bleepingcomputer.com/news/security/massive-chaindrop-npm-supply-chain-attack-infects-hundreds-of-packages/
[18] Before the first prompt: Code execution paths in trusted coding-agent projects -- Datadog Security Labs (2026) -- https://securitylabs.datadoghq.com/articles/coding-agent-project-trust-code-execution-bef

ore-first-prompt/

[19] Claude Code Supply Chain Attacks: Protecting Against Malicious Repository Configs -- MintMCP Blog (2026) -- https://www.mintmcp.com/blog/claude-code-supply-chain-attacks

[20] ChainDrop npm Worm: Bun-loaded CI/CD credential harvester with Ethereum dead-drop C2 -- StepSecurity (2026) -- https://www.stepsecurity.io/blog/chaindrop-npm-worm

[21] Mini Shai-Hulud keyv/cacheable npm Compromise (No CVE Assigned): Self-Propagating Worm Steals CI, Cloud, and Developer Credentials -- Phoenix Security (2026) -- https://phoenix.security/mini-shai-hulud-keyv-cacheable-npm-supply-chain-worm/

[22] Mini Shai-Hulud: TeamPCP's Self-Propagating npm Worm Hits TanStack, OpenSearch, and Mistral AI Across 170 Packages -- Phoenix Security (2026) -- https://phoenix.security/mini-shai-hulud-teampcp-tanstack/

[23] Caught in the Hook: RCE and API Token Exfiltration Through Claude Code Project Files (CVE-2025-59536 / CVE-2026-21852) -- Check Point Research (2026) -- https://research.checkpoint.com/2026/rce-and-api-token-exfiltration-through-claude-code-project-files-cve-2025-59536/

[24] Security: hooks system lacks visibility and write-time confirmation — silent global exfiltration surface, Issue #49778 -- anthropics/claude-code, GitHub (2026) -- https://github.com/anthropics/claude-code/issues/49778

[25] keyv and cacheable npm compromise: 400+ packages -- SafeDep (2026) -- https://safedep.io/keyv-npm-supply-chain-compromise/

[26] Mini Shai-Hulud: AI Developer npm Supply Chain Worm, CSA Research Note -- Cloud Security Alliance Lab Space (2026) -- https://labs.cloudsecurityalliance.org/research/csa-research-note-shai-hulud-ai-supply-chain-20260517-csa-st/

[27] Mini Shai-Hulud hits TanStack & Mistral npm: CVE-2026-45321 (CVSS 9.6), TeamPCP campaign chain -- lilting channel (2026) -- https://lilting.ch/en/articles/mini-shai-hulud-tanstack-mistral-npm-oidc

[28] TeamPCP's Mini Shai-Hulud Is Back: A Self-Spreading Supply Chain Attack Compromises TanStack npm Packages -- StepSecurity (2026) -- https://www.stepsecurity.io/blog/mini-shai-hulud-is-back-a-self-spreading-supply-chain-attack-hits-the-npm-ecosystem

[29] Mini Shai-Hulud: The Worm Returns and Goes Public -- Akamai (2026) -- https://www.akamai.com/blog/security-research/mini-shai-hulud-worm-returns-goes-public

[30] Shai-Hulud npm supply chain attack: What you need to know -- ReversingLabs (2025) -- https://www.reversinglabs.com/blog/shai-hulud-worm-npm

[31] Widespread Supply Chain Compromise Impacting npm Ecosystem -- CISA

(2025-09-23) -- https://www.cisa.gov/news-events/alerts/2025/09/23/widespread-supply-chain-compromise-impacting-npm-ecosystem

[32] Fast-Moving Shai-Hulud Attack Infects npm Packages with 2 Billion Monthly Downloads -- DevOps.com (2026) -- https://devops.com/fast-moving-shai-hulud-attack-infects-npm-packages-with-2-billion-monthly-downloads/

[33] Keyv Supply Chain Compromise: An npm Worm That Takes Its Orders From an Ethereum Smart Contract -- Upwind (2026) -- https://www.upwind.io/feed/keyv-supply-chain-compromise-an-npm-worm-that-takes-its-orders-from-an-ethereum-smart-contract

[34] Worm compromises hundreds of popular npm packages -- Datadog Security Labs (2026) -- https://securitylabs.datadoghq.com/articles/npm-worm-compromises-popular-npm-packages/

[35] Shai-Hulud: The novel self-replicating worm infecting hundreds of NPM packages -- Sysdig (2026) -- https://www.sysdig.com/blog/shai-hulud-the-novel-self-replicating-worm-infecting-hundreds-of-npm-packages

[36] Malicious keyv Package on npm Steals CI Secrets via GitHub -- Mend.io (2026) -- https://www.mend.io/blog/keyv-malicious-packages/

[37] A New Infostealer Worm Hits npm, affecting Keyv and Cacheable -- OX Security (2026) -- https://www.ox.security/blog/a-new-infostealer-worm-hits-npm-affecting-keyv-and-cacheable/

[38] Feross Aboukhadijeh, public incident notification thread (2026) -- https://x.com/feross/status/2084648858605465801