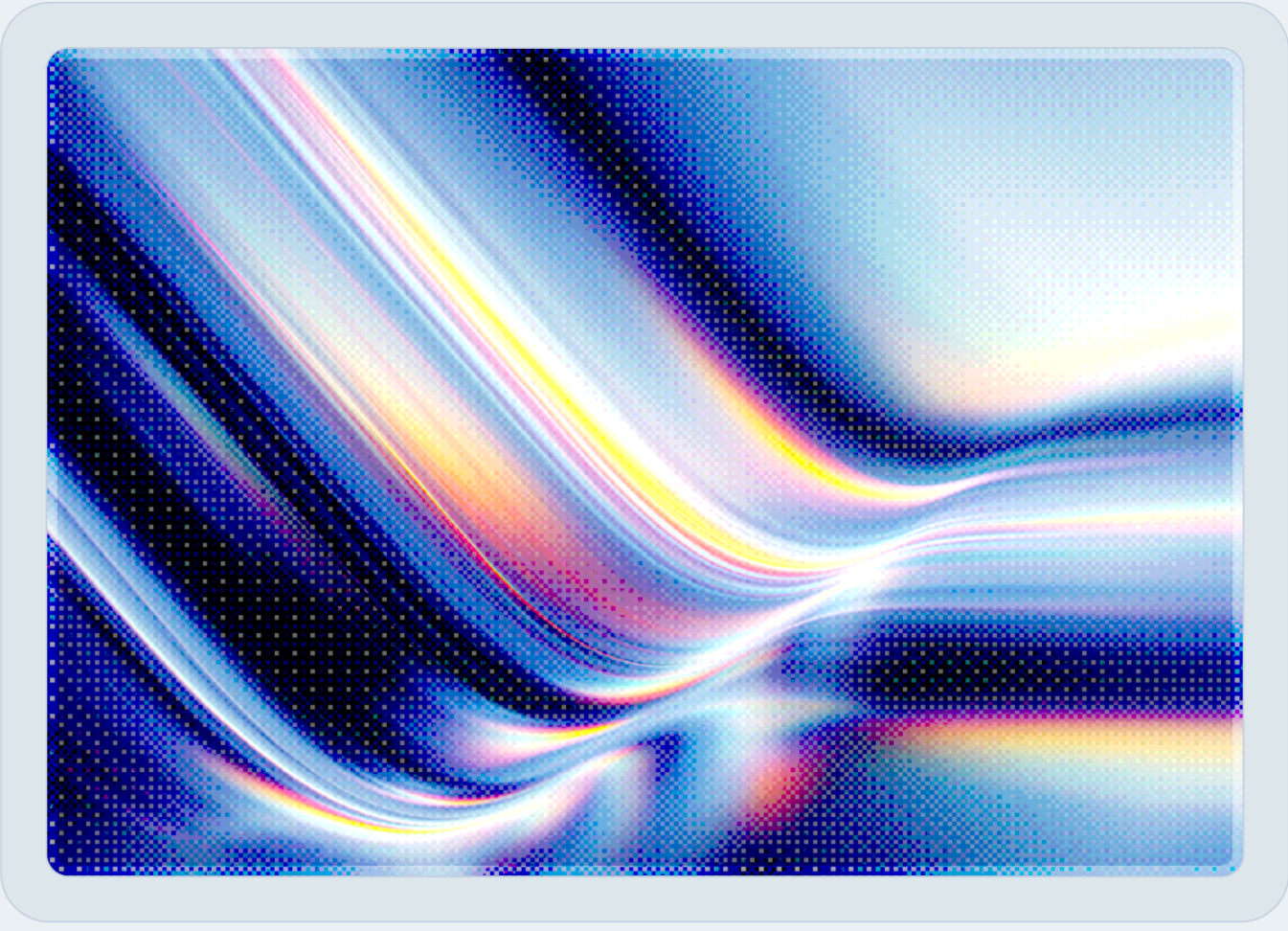
CSAI Foundation | Cloud Security Alliance

jscrambler npm Compromise: IronWorm Targets AI Dev Credentials

2026-07-13

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- On July 11, 2026, attackers who had obtained a valid publishing credential pushed five malicious versions of the jscrambler npm package (8.14.0, 8.16.0, 8.17.0, 8.18.0, 8.20.0); at least three of these — 8.14.0, 8.18.0, and 8.20.0 — have been confirmed to embed a cross-platform Rust infostealer identified as IronWorm, with the delivery mechanism evolving across the release sequence [1][2].
- IronWorm specifically hunts for credentials and configuration belonging to AI coding assistants — Claude Desktop, Cursor, Windsurf, Zed, and VS Code AI extensions — including Model Context Protocol (MCP) server tokens, alongside cloud provider keys, npm/GitHub tokens, and cryptocurrency wallets [1][2].
- Socket's automated scanning detected the first malicious release within six minutes of publication, but later versions (8.18.0, 8.20.0) moved the dropper from a `preinstall` script into the package's main code, defeating scanners and install-time controls that only inspect lifecycle scripts [2][3].
- This is not IronWorm's debut: the same Rust-based, eBPF-assisted, Tor-communicating stealer first surfaced in a June 2026 campaign that hit 36 npm packages, and researchers have observed shared commit naming with the Shai-Hulud worm lineage, suggesting continuity across a broader family of self-propagating npm supply chain campaigns [4][5].
- Organizations that installed any affected version — even transiently, in a CI pipeline or a developer's local environment — should treat all locally stored secrets, and especially AI-tool API keys and MCP credentials, as compromised and rotate them immediately.

Background

A Trusted Build Tool Becomes the Payload Delivery Mechanism

jscrambler is a JavaScript obfuscation and code-integrity tool used by development teams to protect client-side application logic from reverse engineering, and the corresponding npm package draws roughly 15,800 weekly downloads [6]. On July 11, 2026, an attacker who had compromised a legitimate maintainer's publishing credentials used that access to release five new versions of the package in rapid

succession over approximately three hours [2][3]. Because jscrambler is typically pulled into build pipelines and CI/CD systems rather than end-user applications, a compromise of the package places the attacker inside environments that concentrate durable secrets: cloud deployment credentials, source-control tokens, signing keys, and, increasingly, the API keys and local configuration files that AI coding assistants use to authenticate to model providers and to Model Context Protocol servers.

The initial malicious release, version 8.14.0, shipped a `preinstall` lifecycle hook that executed a setup script the moment the package was installed, requiring no further action from the developer and no import of the package into application code [1][7]. Socket's registry-monitoring system flagged the release within six minutes, and the maintainers subsequently revoked and rotated the compromised publishing credentials and deprecated the affected versions [2][3]. Jscrambler has stated that it identified zero confirmed downloads of the malicious releases at the time of its public disclosure, though the investigation into full exposure was still underway as of this writing, and organizations that installed the package via automated dependency updates, lockfile changes, or CI caching in the window before deprecation should not assume they were unaffected [3][7]. A clean version, 8.22.0, is available for immediate upgrade [3].

From a Single Payload to a Family of Campaigns

The malware embedded in the compromised jscrambler releases has been identified by JFrog Security Research as IronWorm, a Rust-compiled, cross-platform infostealer that the firm first documented in a June 2026 campaign affecting 36 separate npm packages [4][5]. That earlier wave targeted a defined set of 86 environment variables and roughly 20 categories of credential files, including OpenAI, Anthropic, AWS, and npm tokens, SSH keys, and Exodus cryptocurrency wallet data [5]. JFrog's July analysis of the jscrambler incident, titled "IronWorm Returns as jscrambler, Rustier Than Ever," documents substantial evolution in the malware between the two campaigns: the attacker moved from a single-platform Linux payload to a multi-platform container format bundling distinct Linux, Windows, and macOS arm64 executables, and added the ability to propagate itself by uploading new malicious packages directly to the npm registry via raw HTTP PUT requests, bypassing the npm command-line client entirely [8].

JFrog researchers also noted overlapping commit-naming conventions between IronWorm and the Shai-Hulud worm family [4], a self-propagating npm threat that has produced multiple waves of compromise since September 2025, including campaigns publicly tracked as "Shai-Hulud: The Third Coming," which compromised the `@bitwarden/cli` package in April 2026 [13], and "Mini Shai-Hulud," attributed to a group researchers call TeamPCP [9]. While JFrog stops short of asserting a direct authorship link, the pattern is consistent with a broader trend: npm supply chain attacks in 2026 have moved from isolated incidents toward reusable, increasingly automated tooling that different threat actors adapt and

redeploy against new packages. Security teams should treat the jscrambler incident not as an isolated event but as the latest data point in an active and evolving campaign family that specifically prioritizes developer and AI-tool credentials over the broader consumer-credential targeting seen in earlier infostealer generations. CSA's own prior research on this lineage, "Shai-Hulud: npm Worm Targeting AI Developer Toolchains" [13], documents that same "Third Coming" campaign in detail and is a useful next step for readers seeking deeper background on the worm family this incident extends.

Security Analysis

Execution Mechanics and Evasion of Lifecycle-Script Controls

Version 8.14.0 relied on a `preinstall` hook that executed `dist/setup.js` automatically during `npm install`, a pattern that many organizations already scan for or restrict, including through the `--ignore-scripts` flag and npm 12's July 2026 change to disable install scripts by default [1][2]. In direct response to that class of defense, the attacker evolved the payload in versions 8.18.0 and 8.20.0: rather than triggering at install time, the dropper was embedded as a self-executing function inside the package's main code and CLI entry point, so it fires whenever the package is imported or its command-line tool is invoked. This means that `npm install --ignore-scripts` — the standard mitigation organizations increasingly rely on after the last several years of npm supply chain incidents — does not stop the later variants, because execution is deferred from install time to first use [2][3]. The evolution of this payload's delivery mechanism suggests that defenses built solely around npm lifecycle-script scanning may be insufficient.

Once triggered, the loader reads a 7.8 MB binary container identified by a custom `\x1bCSI\x01` header, selects the payload matching the host operating system, extracts it to a randomly named temporary file, and launches it detached and hidden from the user [2]. On Linux, the payload uses eBPF kernel loading to obtain elevated visibility and persistence; on Windows it installs via scheduled tasks; and on macOS it persists through a LaunchAgent [1][2]. The malware protects roughly 2,400 sensitive strings inside the binary using per-string ChaCha20-Poly1305 encryption with individually stored keys and nonces, a level of obfuscation that likely serves to frustrate static analysis by security researchers and automated scanners [2].

Targeting AI Developer Credentials as a Distinct Objective

What distinguishes this campaign from a generic infostealer is the explicit targeting of AI development tool configuration alongside traditional cloud and cryptocurrency credentials. The payload searches for and exfiltrates configuration belonging to Claude Desktop (including `.claude.json` files), Cursor's MCP server credentials, and API keys stored by Windsurf, Factory, Zed, and VS Code AI integrations [1] [2]. This targeting suggests a shift in where valuable secrets now live on a developer's machine. Model Context Protocol servers frequently hold credentials that grant an AI agent read/write access to source repositories, cloud infrastructure, ticketing systems, and internal databases — meaning a stolen MCP credential can be functionally equivalent to stealing the cloud or source-control token it was configured to access, but through a control surface that most organizations have not yet included in their credential-rotation or secrets-scanning programs.

Table 1 summarizes the principal credential categories the malware targets, drawn from Socket's and JFrog's technical analyses.

Category	Specific Targets	Why It Matters
AI coding assistants	Claude Desktop config/ `.claude.json` , Cursor MCP credentials, Windsurf, Factory, Zed, VS Code AI extensions	MCP tokens often grant broad, agent-driven access to source control, cloud, and internal systems
Cloud platforms	AWS ECS/Secrets Manager, GCP metadata/Secret Manager, Azure IMDS and management credentials	Direct path to production infrastructure and data
Source/package registries	npm tokens (`.npmrc`), GitHub credentials	Enables further supply chain propagation and code tampering
Cryptocurrency wallets	MetaMask, Trust Wallet, Coinbase Wallet, Phantom, Exodus	Direct financial theft; seed phrases enable irreversible loss
Password managers	Bitwarden, 1Password vault data	Cascading compromise across all stored credentials

Category	Specific Targets	Why It Matters
Collaboration and browser sessions	Discord, Slack, Telegram tokens; Chromium and Firefox session/cookie data	Enables social-engineering follow-on and session hijacking
Security tooling	Metasploit, Sliver, Havoc framework artifacts	Suggests deliberate targeting of security researchers and red teams

Beyond exfiltration, JFrog identified an operational security flaw in the C2 design: while command-and-control communications route through a self-managed Tor Expert Bundle using layered X25519 and ChaCha20-Poly1305 encryption, bulk stolen data is instead uploaded directly to the file-sharing service temp.sh over unproxied HTTPS, and to two identified command servers at IP addresses 37.27.122.124 and 57.128.246.79 [1][8]. This bypass exposes victim IP addresses at the moment of large-scale data exfiltration despite the apparent goal of anonymizing other channel traffic, a detail defenders can use both for detection (flagging outbound connections to temp.sh from build and CI infrastructure) and for potential attribution efforts by researchers and law enforcement.

Why Security-Vendor and Build-Tool Packages Are Disproportionately Attractive Targets

CSA assesses that the choice of jscrambler as a delivery vehicle is consistent with a broader attacker calculus visible across 2026's npm incidents: packages consumed inside build pipelines and by security-conscious engineering teams likely offer higher-value footholds than consumer-facing libraries, because they run with elevated trust and typically execute inside CI/CD environments that aggregate secrets from many downstream systems. A code-obfuscation tool is installed specifically by teams protecting valuable intellectual property, and its presence in a build pipeline frequently co-locates with deployment credentials, signing keys, and increasingly, AI agent tokens with cross-system reach. On this reading, attackers compromising the publishing credentials of such tools gain a distribution channel that reaches security-mature organizations who might otherwise catch a less sophisticated attack.

Recommendations

Immediate Actions

Identify and remove affected versions across every environment. Search all lockfiles (`package-lock.json`, `yarn.lock`, `pnpm-lock.yaml`), CI build logs, container images, and developer workstations for jscrambler versions 8.14.0, 8.16.0, 8.17.0, 8.18.0, or 8.20.0. Because later versions execute on import rather than install, simply confirming the package was never explicitly installed via a compromised version number is not sufficient — check whether any dependency resolution pulled one of these versions transiently, even if a `package.json` specifies a version range rather than an exact pin. Upgrade to the confirmed-clean 8.22.0 release, or pin to the last known-good 8.13.0 if immediate upgrade testing is not feasible [1][3].

Rotate every credential reachable from an affected machine. Any system that installed a compromised version — including ephemeral CI runners — should be treated as having had its accessible secrets exposed. This includes cloud provider access keys (AWS, Azure, GCP), npm and GitHub tokens, AI provider API keys (OpenAI, Anthropic, and others configured in local AI tool settings), MCP server credentials, cryptocurrency wallet keys, and password manager master credentials where locally cached. Revoke and reissue rather than merely rotate where the provider supports full key regeneration [1][2].

Block known indicators of compromise at the network layer. Add the two identified command-and-control IP addresses, 37.27.122.124 and 57.128.246.79, to network and EDR blocklists, and alert on outbound connections to temp.sh from build servers, CI runners, and developer endpoints, since legitimate CI/CD activity has no reason to reach that service [1][8].

Short-Term Mitigations

Do not rely solely on lifecycle-script restrictions. The evolution from a `preinstall`-triggered dropper to one embedded in main-module code demonstrates that npm 12's install-scripts-disabled default and the `--ignore-scripts` flag, while valuable, are not sufficient controls on their own. Organizations should pair script restrictions with runtime monitoring on build infrastructure capable of detecting anomalous process spawning, unexpected binary extraction to temp directories, and eBPF program loading from non-administrative processes.

Extend secrets-scanning and credential-hygiene programs to cover AI tool configuration. Most existing secrets-management and endpoint data-loss-prevention tooling was built around traditional credential stores — environment variables, cloud CLI config, SSH keys — and has not yet been extended

to cover MCP server credential files and AI coding assistant configuration directories. Security teams should inventory where these files live for each AI tool in use across the organization and bring them under the same monitoring, encryption-at-rest, and access-control expectations applied to other high-value secrets.

Require registry-level publish protections for all critical dependencies. Enforce multi-factor authentication and, where supported, hardware-token-backed publishing for maintainers of any package the organization depends on directly in build pipelines, and prefer registries or organizational policies that support provenance attestation (such as npm's package provenance feature) to make compromised-publisher scenarios like this one detectable before installation.

Strategic Considerations

Treat AI agent credentials as a first-class category in supply chain risk models. The specific targeting of Claude Desktop, Cursor, and MCP credentials in this and the preceding IronWorm campaign indicates that infostealer authors have identified AI development tooling as a high-value target class in its own right, not an incidental one. Organizations building AI-assisted development workflows should model MCP server credentials with the same sensitivity as production cloud access keys, since a compromised MCP token can grant an attacker the same agentic reach the credential was provisioned to give a trusted developer or AI agent.

Plan for continued campaigns from this malware family. Given the documented links between IronWorm and the broader Shai-Hulud lineage of self-propagating npm worms, and the demonstrated rapid evolution between the June and July 2026 IronWorm variants, organizations should assume further campaigns using this or closely related tooling are likely rather than treating the jscrambler incident as a one-off event. Dependency-update and software-composition-analysis processes should incorporate near-real-time registry threat feeds (such as those from Socket, SafeDep, or equivalent vendors) rather than relying solely on periodic vulnerability scanning.

CSA Resource Alignment

The jscrambler compromise reinforces guidance CSA has already published on software supply chain transparency and on the organizational responsibilities that come with adopting AI development tools, and it surfaces a gap — credential exposure specific to AI agent tooling — that existing supply chain frameworks are only beginning to address explicitly.

CSA's "Software Transparency: Securing the Digital Supply Chain" directly addresses the underlying dynamics this incident exemplifies: the risk concentration created by widespread open-source and CI/CD dependency, and the practical controls — software composition analysis, dependency provenance, and pipeline access controls — needed to detect compromised packages before they reach production [10]. The jscrambler incident is a concrete instance of the poisoned-pipeline and malicious-package-injection scenarios that the resource catalogs as key threats to the software supply chain, and organizations implementing its recommended SBOM and component-analysis practices would likely have had a documented basis for identifying which build environments consumed the compromised versions.

CSA's "AI Organizational Responsibilities: AI Tools and Applications" guidance is the more specific fit for the incident's most novel element: the deliberate targeting of AI coding assistant credentials and MCP server tokens [11]. That guidance addresses third-party AI tool supply chain risk and vendor assessment as an organizational responsibility distinct from traditional software supply chain management, and its treatment of AI tool governance supports the recommendation above that MCP credentials be brought under the same access-control and monitoring discipline as other high-value secrets. Organizations should use this guidance as the basis for extending existing supply chain risk registers to explicitly enumerate AI development tool credentials as an asset class.

As a topical fallback where CSA has not yet published incident-specific guidance, the AI Controls Matrix (AICM) v1.1 provides applicable control language through its Threat and Vulnerability Management (TVM) domain, which covers timely remediation of compromised dependencies and monitoring of build environments for unauthorized code execution [12]. Organizations pursuing STAR or STAR-for-AI assurance should document their jscrambler exposure assessment and remediation as supporting evidence for this AICM domain. Readers seeking fuller background on the Shai-Hulud worm lineage this incident extends should also consult CSA's prior research note on that campaign family [13], which documents the April 2026 "Third Coming" campaign in detail.

References

[1] The Hacker News. "Compromised jscrambler 8.14.0 npm Release Drops Rust Infostealer During Install." The Hacker News, July 2026.

[2] Socket. "jscrambler npm Package Compromised in Supply Chain Attack." Socket, July 2026.

[3] SafeDep. "Official jscrambler npm Package Compromised Across Multiple Releases." SafeDep, July 2026.

[4] BleepingComputer. "New IronWorm malware hits 36 packages in npm supply-chain attack." BleepingComputer, June 2026.

[5] JFrog Security Research. "IronWorm: Shai-Hulud's rustier cousin." JFrog, June 2026.

[6] Cryptika Cybersecurity. "Hackers Compromised jscrambler With 15,800+ Weekly Downloads to Attack Developers." Cryptika, July 2026.

[7] TechNadu. "jscrambler npm Package Compromised in Supply Chain Attack, Targeting Wallets, AI Tools, Cloud Credentials." TechNadu, July 2026.

[8] JFrog Security Research. "IronWorm Returns as jscrambler, Rustier Than Ever." JFrog, July 2026.

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

[10] Cloud Security Alliance. "Software Transparency: Securing the Digital Supply Chain." CSA, 2022 (rev. 2025).

[11] Cloud Security Alliance. "AI Organizational Responsibilities: AI Tools and Applications." CSA, 2025.

[12] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." CSA, 2025.

[13] Cloud Security Alliance AI Safety Initiative. "Shai-Hulud: npm Worm Targeting AI Developer Toolchains." CSA Labs, April 2026.

