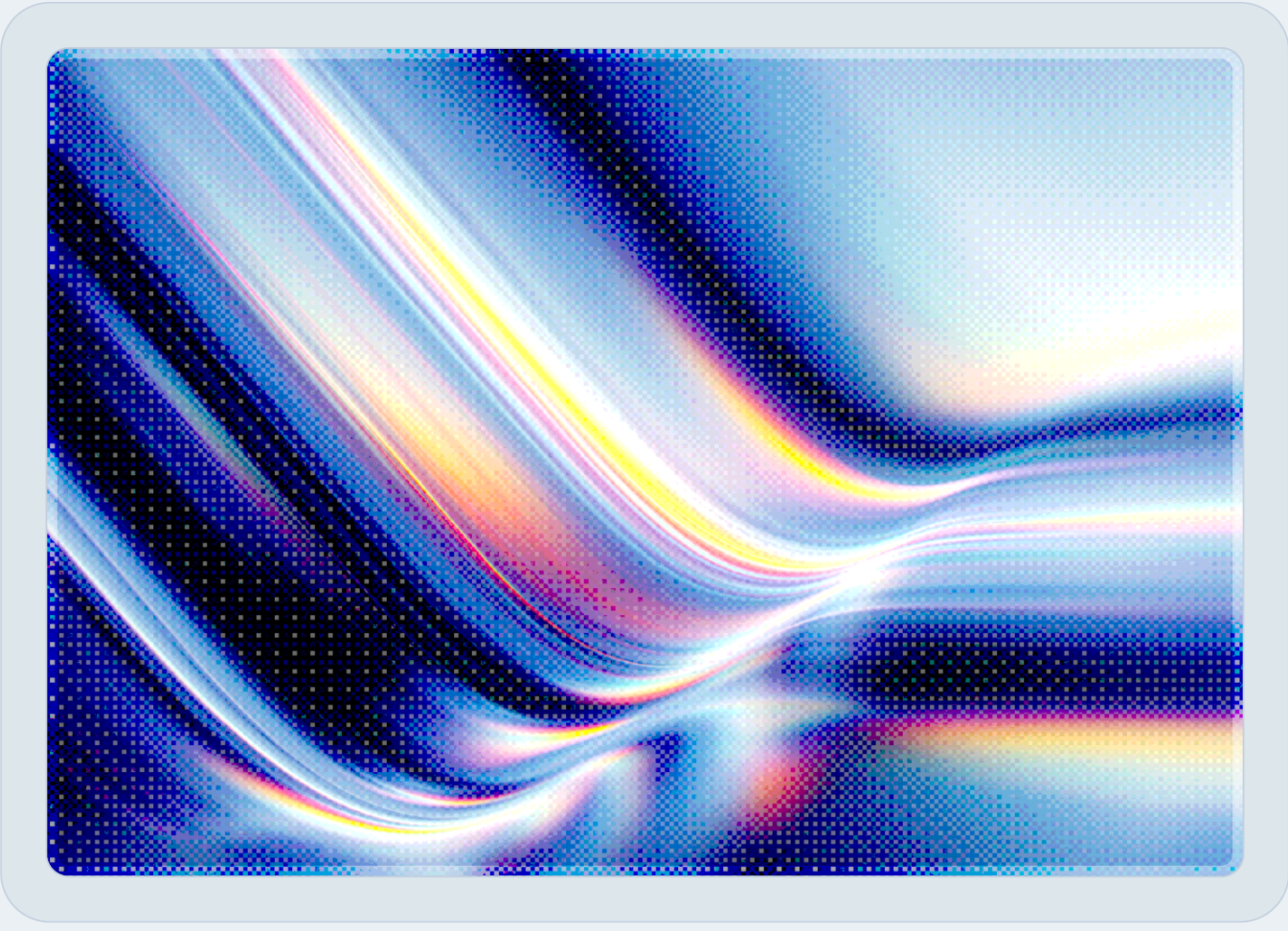
CSAI Foundation | Cloud Security Alliance

TrapDoor: Supply Chain Attack Poisons AI Coding Assistants

Cross-Ecosystem Credential Theft Targeting Crypto and AI Developers via AI-Native Persistence

2026-05-26

AI-assisted Rapid Research



CSAI | cloud security alliance®

Key Takeaways

- An active supply chain campaign named TrapDoor has published more than 34 malicious packages spanning 384+ artifact versions across npm, PyPI, and Crates.io, targeting developers in the crypto, DeFi, Solana, and AI communities with credential-stealing malware. [1]
- What distinguishes TrapDoor from prior supply chain campaigns is its deliberate exploitation of AI coding assistant configuration files: the shared npm payload plants `.cursorrules` and `CLAUDE.md` files containing hidden instructions embedded with zero-width Unicode characters (U+200B, U+200C, U+200D, U+FEFF) that are invisible in standard editors but parsed and acted upon by AI assistants such as Cursor and Claude Code. [1][2]
- The attacker extended the poisoning campaign beyond malicious packages by opening pull requests against high-profile open-source AI projects — including langchain, langflow, browser-use, llama_index, MetaGPT, and OpenHands — proposing seemingly benign documentation commits that contained the same hidden-instruction payloads. [1][3]
- The malware conducts broad credential sweeps targeting SSH keys, AWS and GitHub tokens, cloud credentials, environment variables, API keys, browser profiles, and crypto wallet keystores, with validation of harvested cloud credentials via live API calls before exfiltration. [1] [2]
- Socket Security's automated analysis flagged the first TrapDoor packages within a median of five minutes and twenty-seven seconds of publication, with the fastest detection occurring fifty-eight seconds after a package went live, demonstrating that registry-level behavioral monitoring can identify novel malicious packages within minutes of publication, though the campaign's 384+ artifact versions indicate it achieved significant distribution before full containment. [1]
- Organizations whose developers use AI coding assistants should treat AI context configuration files — `.cursorrules`, `CLAUDE.md`, `AGENTS.md`, and equivalents — as trusted-execution surfaces requiring the same review controls applied to build scripts and CI workflow definitions.

Background

AI Coding Assistants as an Emerging Attack Surface

The proliferation of AI coding assistants has introduced a class of configuration files that occupy a privileged position in the developer security model that security programs have not yet widely addressed. Files such as `.cursorrules` (used by the Cursor IDE) and `CLAUDE.md` (used by Anthropic's Claude Code CLI) exist specifically to provide AI agents with project-level behavioral context: what coding standards to follow, which commands to run, how to format outputs, and what workflows to execute. When a developer opens a project, these files are automatically loaded by the AI assistant, which then follows their instructions without requiring explicit invocation by the human.

This design is intentional and useful — it allows teams to codify AI behavior for their specific codebase. However, it creates a trust assumption that adversaries can exploit: any `.cursorrules` or `CLAUDE.md` file present in a project directory will be treated as authoritative instruction by the AI assistant, regardless of whether the human developer wrote it, reviewed it, or is even aware of its contents. Security researchers had theorized this attack vector as a natural extension of prompt injection, but TrapDoor is, to the knowledge of the researchers who identified it, the first documented campaign to operationalize this vector at scale across live package registries and open-source repositories. [3]

The Cross-Ecosystem Supply Chain Context

Industry threat intelligence has documented steady escalation in the sophistication of software supply chain attacks targeting package registries since 2020. Early campaigns relied on typosquatting — registering packages with names similar to popular libraries — but more recent operations have adopted social engineering, account compromise, and automated tooling to publish malicious packages at volume under names that mimic legitimate developer utilities. [4] TrapDoor follows this advanced pattern while adding two capabilities not previously observed in combination: simultaneous distribution across three distinct language ecosystems and active exploitation of AI assistant tooling as a persistence and exfiltration mechanism.

The campaign's choice of targets reflects deliberate threat modeling. Crypto and DeFi developers frequently store SSH keys, cloud credentials, GitHub tokens, and cryptocurrency wallet keystores on the same development machine — a concentration of high-value secrets that makes them attractive targets for credential-harvesting campaigns. AI and ML developers similarly accumulate high-value credentials — API keys for foundation model providers, cloud storage access, model registry tokens — alongside their

development tooling. Both communities make frequent use of npm, PyPI, and Rust crates, and both have rapidly adopted AI coding assistants, making them simultaneously high-value targets and users of the attack vector TrapDoor introduces.

Security Analysis

Campaign Timeline and Scale

The earliest confirmed TrapDoor artifact is PyPI's `eth-security-auditor@0.1.0`, uploaded on May 22, 2026 at 20:20:18 UTC, though analysis of infrastructure associated with the campaign places its actual start date at May 19, 2026. [1] Packages were published in waves over the following days across three registries, ultimately reaching 34 distinct malicious packages and more than 384 artifact versions by the time the campaign was reported. The npm registry accounts for 21 of the malicious packages, PyPI for 7, and Crates.io for 6. [1][2]

All packages present as legitimate developer utilities, with names chosen to appeal specifically to the targeted communities. npm packages include `crypto-credential-scanner`, `defi-env-auditor`, `prompt-engineering-toolkit`, `llm-context-compressor`, `model-switch-router`, and `wallet-security-checker`. PyPI packages include `cryptowallet-safety`, `defi-risk-scanner`, and `eth-security-auditor`. Crates.io packages target the Sui Move blockchain developer community with packages such as `sui-framework-helpers`, `move-compiler-tools`, and `sui-sdk-build-utils`. [1] The names are designed to pass casual review: they sound like tools a security-conscious developer would want, and several explicitly invoke security and auditing concepts to encourage installation by developers attempting to improve their own security posture.

Payload Mechanics Across Ecosystems

Each ecosystem is handled through a mechanism matched to its execution model. npm packages trigger a shared payload component – `trap-core.js` at 1,149 lines and 48,485 bytes – via postinstall hooks that run automatically when the package is installed. [1] PyPI packages execute on import by downloading JavaScript from an attacker-controlled GitHub Pages domain and running it via `node -e`, a technique that keeps the initial package small and relocates the payload to infrastructure that can

be updated without republishing. Crates.io packages use `build.rs` scripts that execute automatically during Rust compilation, searching for local keystores and exfiltrating data to GitHub Gists using XOR encryption with the hardcoded key `cargo-build-helper-2026`. [1][2]

The npm payload is the most feature-complete of the three, scanning the broadest range of credential types and implementing the most extensive persistence mechanisms. It scans filesystems for credentials across a broad range of target categories: SSH keys, Sui, Solana, and Aptos wallet data, AWS credentials, GitHub tokens, browser profile databases, browser-stored login data, crypto wallet browser extensions, environment variables, API keys, and local development configuration files. Harvested AWS and GitHub tokens are validated against live API endpoints before exfiltration — a validation step that confirms the credential is active and filters out stale secrets, suggesting an attacker prioritizing credential quality over raw volume. [1] Validated credentials are then encrypted using Fernet symmetric encryption with ECDH key exchange (using ECDH to derive the shared Fernet key) before transmission to attacker-controlled infrastructure.

Persistence is achieved through multiple redundant mechanisms. The payload attempts to install cron jobs, systemd services, Git hooks, and shell hooks, ensuring that at least one persistence method survives even in environments where some mechanisms are restricted. Lateral movement is attempted using harvested SSH keys. [1][2]

AI-Native Persistence: The CLAUDE.md and .cursorrules Vector

The most technically novel aspect of TrapDoor is its systematic exploitation of AI coding assistant configuration files as a persistence and exfiltration mechanism. The `trap-core.js` payload plants `.cursorrules` and `CLAUDE.md` files in the developer's project directory. These files are populated with content that appears blank or benign in standard text editors because the malicious instructions are concealed using zero-width Unicode characters — specifically U+200B (zero-width space), U+200C (zero-width non-joiner), U+200D (zero-width joiner), and U+FEFF (byte order mark used mid-text). The precise encoding scheme has not been publicly detailed, but the effect is that human-readable malicious instructions are rendered invisible in standard editors while remaining fully accessible to the AI's text parser; zero-width characters are embedded within or surrounding the plaintext directives, suppressing their visual rendering without preventing LLM interpretation. [3]

When a developer next uses Claude Code or Cursor in a project directory containing a poisoned configuration file, the AI assistant receives and executes instructions that the human user has no ability to observe through normal tooling. Those hidden instructions trigger a workflow the malware presents as a "security scan," which in practice collects and exfiltrates local secrets through the assistant's command

execution capabilities. [3] The attack requires no vulnerability in the AI assistant itself — it exploits the assistants' designed behavior of reading and acting on project configuration files, treating them as trusted.

The Open-Source Repository Injection Campaign

Beyond malicious packages, TrapDoor operationalized a second distribution vector: direct injection of poisoned configuration files into popular open-source repositories via pull requests. The GitHub account `ddjidd564` opened pull requests against six high-profile AI projects: `browser-use/browser-use`, `langchain-ai/langchain`, `langflow-ai/langflow`, `run-llama/llama_index`, `FoundationAgents/MetaGPT`, and `OpenHands/OpenHands`. [1] [3] Each pull request used benign-sounding commit messages consistent with documentation updates — for example, "docs: add .cursorrules with dev standards and build verification" — to minimize the probability of immediate rejection by maintainers performing a casual review.

The proposed configuration files pointed to an attacker-controlled configuration endpoint at `ddjidd564.github.io/defi-security-best-practices/config.json` and included a campaign marker string `P-2024-001` that appears consistently across related campaign components. [1] According to available threat intelligence at the time of this report, all six pull requests were identified as malicious and closed without being merged; the attack does not appear to have succeeded at this repository injection vector. Had any of these pull requests been merged, every developer who subsequently cloned or updated the affected repository and used a compatible AI coding assistant would have had their local environment exposed to the hidden-instruction payload — with no malicious package installation required. This propagation model exploits the trust that developers place in well-maintained open-source projects and the automatic trust that AI assistants grant to configuration files present in project directories.

Attribution and Infrastructure

The campaign's infrastructure centers on a GitHub account identified as `ddjidd564`, which operates the attacker-controlled GitHub Pages domain `ddjidd564[.]github[.]io`. The npm publisher account `asdxzxc` is associated with multiple active package versions. Two PyPI publisher accounts, `asdmini67` and `dae5411`, are linked to the PyPI component of the campaign. [1] Attacker-hosted documentation on the GitHub Pages domain includes files named `AUDIT-MATRIX.md`, `BYPASS.md`, `PAYLOAD.md`, and `SWARM.md`, suggesting the campaign is organized around a

framework with distinct operational phases. The campaign marker `P-2024-001` appearing across payload configurations and pull request content suggests either a tracking convention for multiple concurrent campaigns or an attempt to mislead attribution by referencing a 2024 date.

Recommendations

Immediate Actions

Organizations should audit all project directories for unexpected `.cursorrules` and `CLAUDE.md` files, particularly those not explicitly authored by the development team. Standard text editors will not reveal hidden instructions; detection requires scanning for zero-width Unicode characters using grep patterns targeting the Unicode code points U+200B, U+200C, U+200D, and U+FEFF. Any such files containing these characters should be removed and the machine treated as potentially compromised. Git history for these files should be reviewed to identify when they were created and whether the creation aligns with expected developer activity. [3]

Developers who installed any of the 34 identified malicious packages should rotate all credentials accessible from the affected machine without delay, including SSH key pairs, AWS access keys, GitHub personal access tokens, and any cryptocurrency wallet keystores. Credential rotation is necessary even if no explicit evidence of exfiltration is observed, because the payload validates credentials before exfiltrating, and the absence of observed exfiltration does not confirm the credentials were not collected.

Short-Term Mitigations

Organizations should implement allowlist policies that restrict which repositories their AI coding assistant tools are permitted to read configuration files from, treating these files as equivalent in trust level to build scripts and CI workflow definitions. For Claude Code specifically, the `CLAUDE.md` file should be governed under the same review controls applied to `Makefile`, `.github/workflows/`, or other files that can cause code execution. The same principle applies to `.cursorrules` for Cursor and equivalent configuration files for other AI coding assistants as they emerge.

Registry-level behavioral monitoring for npm, PyPI, and Crates.io packages — implemented either through a commercial supply chain security platform or through pre-installation scanning tooling — provides the earliest detection opportunity for this class of attack. Socket Security's automated systems

detected the first TrapDoor packages within roughly six minutes of publication at median. [1] Without such monitoring, developers have no mechanism to know a package is malicious before it executes its postinstall hook or runs on first import.

Software composition analysis (SCA) tooling should be evaluated for its ability to detect malicious packages versus merely known-vulnerable ones. Most SCA tools are designed to identify packages with disclosed CVEs, not packages that are malicious by design. TrapDoor packages will not appear in CVE databases; they require behavioral or heuristic analysis of the packages themselves.

Strategic Considerations

TrapDoor establishes a new attack pattern that organizations should expect to see replicated: AI coding assistant configuration files as a persistence and social engineering vector. As AI assistants become embedded in standard developer workflows, the implicit trust granted to their configuration files will become an increasingly attractive target for adversaries. Treating these files as trusted-execution surfaces — subject to code review, version control hygiene, and supply chain scrutiny — is a necessary adaptation to the current threat environment.

The pull request injection component of TrapDoor points to a systematic supply chain risk in organizations that allow AI coding assistants to read configuration files from cloned repositories without additional review. A developer who clones a popular open-source project for reference or contribution, and whose AI assistant automatically reads that project's configuration files, is exposed to any malicious instructions those files contain — even if the malicious instructions were never merged into the main branch. Developers who explicitly check out or review pull request branches in their local environment would be exposed, as would contributors whose IDE or tooling is configured to automatically fetch all remote refs. Security awareness training for developers should explicitly cover this attack surface.

At the organizational level, enterprises should work with AI assistant vendors to understand what controls exist for scoping configuration file trust. Configurations that limit which directories or repositories an AI assistant will read behavioral instructions from, or that require explicit user approval before acting on configuration file instructions, would substantially reduce the attack surface TrapDoor exploits.

CSA Resource Alignment

TrapDoor's exploitation of AI coding assistant configuration files maps directly to threat scenarios addressed in the MAESTRO (Multi-Agent Environment, Security, Threat, Risk, and Outcome) framework, which identifies instruction injection and context manipulation as primary risk categories for AI agent

systems. [5] The hidden-instruction technique TrapDoor employs is a practical instantiation of the agent framework and deployment infrastructure threat layers defined in MAESTRO, in which malicious instructions are introduced through trusted channels — project configuration files — rather than through direct user input, bypassing the human's ability to review what the AI agent is instructed to do.

The AI Controls Matrix (AICM) domain covering AI supply chain security is directly applicable, particularly controls addressing the integrity of model inputs, the provenance of context provided to AI systems, and the governance of AI-readable configuration artifacts. [6] Organizations implementing AICM controls should extend their supply chain integrity controls to cover AI coding assistant configuration files as a new artifact category requiring the same provenance and integrity assurances applied to software dependencies.

CSA's Cloud Controls Matrix (CCM) domain on Application and Interface Security (AIS) and the Threat and Vulnerability Management (TVM) domain are both relevant to the broader supply chain dimensions of TrapDoor. The CCM's guidance on software composition analysis and dependency management applies directly to the malicious package distribution component of the campaign. Enterprises with CCM-aligned security programs should review whether their existing SCA tooling addresses malicious packages, not only known-vulnerable ones.

The Zero Trust principles documented in CSA's Zero Trust and Agentic Trust guidance apply to this threat through the lens of least privilege and explicit verification. [7] Granting AI coding assistants implicit trust to all configuration files present in a project directory is inconsistent with Zero Trust principles; organizations should move toward explicit trust boundaries for AI agent inputs, consistent with the verification-first posture Zero Trust requires.

References

[1] Socket Security Research Team. "TrapDoor Crypto Stealer Supply Chain Attack Hits 34 Packages Across npm, PyPI, and Crates.io." Socket Security Blog, May 2026.

[2] The Hacker News Staff. "TrapDoor Supply Chain Attack Spreads Credential-Stealing Malware via npm, PyPI, and CratesIO." The Hacker News, May 2026.

[3] Phoenix Security Research. "TrapDoor Supply Chain Campaign: Cross-Ecosystem Credential Theft and AI Assistant Poisoning via npm, PyPI, and Crates.io." Phoenix Security, May 2026.

[4] Cybersecurity News. "Hackers Compromised 34 Packages in npm, PyPI, and Crates in New Supply Chain Attack." CybersecurityNews.com, May 2026.

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

[6] Cloud Security Alliance. "AI Controls Matrix (AICM)." Cloud Security Alliance, 2025.

[7] Cloud Security Alliance. "The Agentic Trust Framework: Zero Trust Governance for AI Agents." Cloud Security Alliance, February 2026.

See Also

The following sources corroborate the core campaign details and may be useful for additional context:

- SOCRadar Research. "TrapDoor: Malicious npm, PyPI, Crates.io Packages Target Developer Secrets and AI Tooling." SOCRadar, May 2026.
- Rescana Threat Intelligence. "TrapDoor Supply Chain Attack Actively Exploiting npm, PyPI, and CratesIO to Steal Developer Credentials in Crypto, DeFi, Solana, and AI Sectors." Rescana, May 2026.