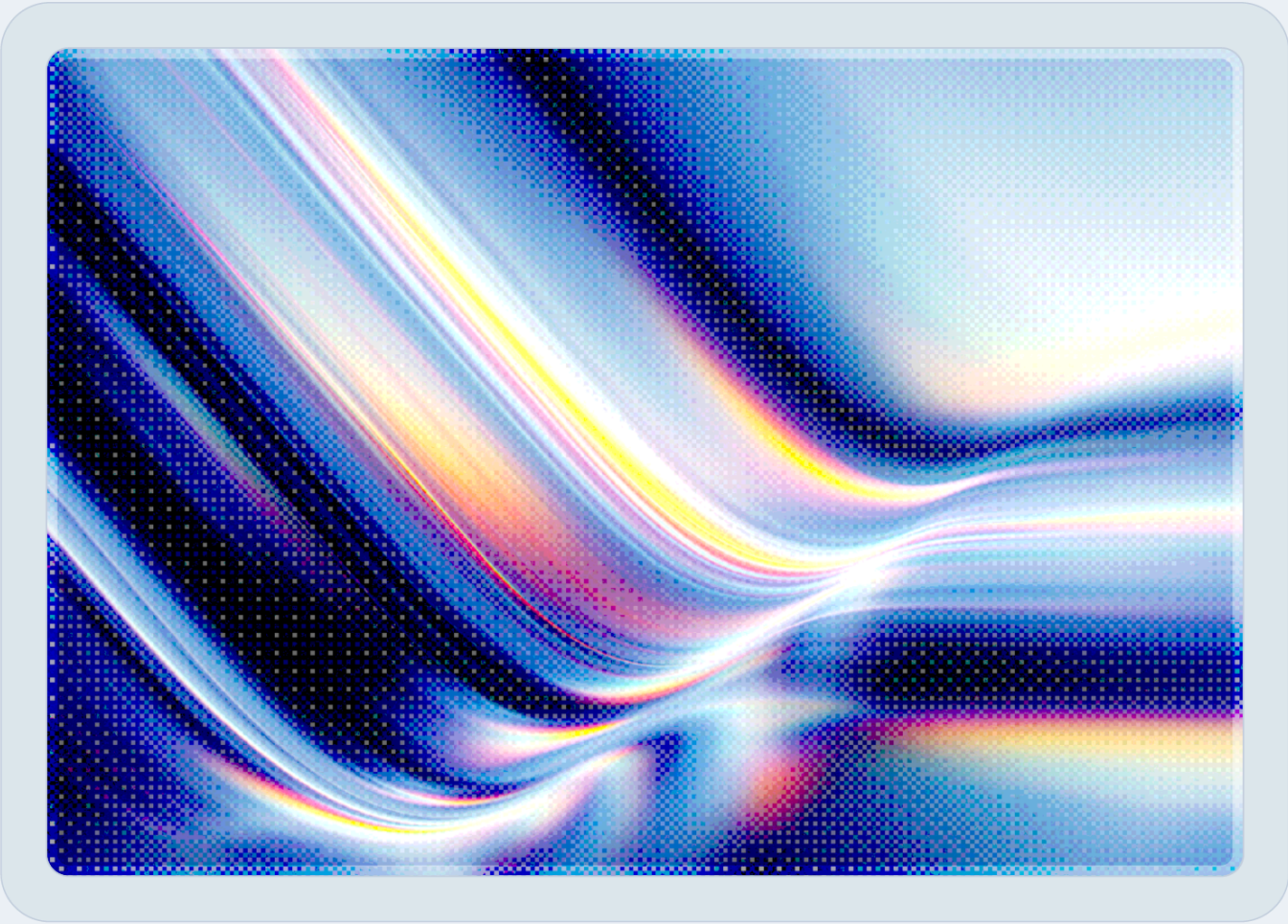
CSAI Foundation | Cloud Security Alliance

The llms.txt Trust Model Is Broken

AI Coding Agents Install Unregistered Packages Across the Fortune 500

2026-09-07

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

Independent security research disclosed in late August and early September 2026 shows that `llms.txt` files — the machine-readable documentation pages that thousands of companies now publish to guide AI agents through their products [5] — were the subject of a disclosure covering a scanned population of 6,214 domains belonging to Fortune 500 organizations, defense contractors, and major technology firms. Within that scan, a measurable share of the published files — roughly 1.5 percent — referenced software packages and domains that were never registered [1][2]. When a researcher registered a sample of those unregistered names, drawn from 120 unregistered references found across the scan, and hosted benign "phone-home" packages in their place, AI coding agents including Anthropic's Claude, OpenAI's Codex, and Nous Research's Hermes installed and executed the substituted code inside corporate networks, in one case within minutes of the name being claimed [2] [3]. The underlying failure is not a bug in any single agent but a trust model: coding agents treat vendor-published documentation as ground truth and execute the installation commands it contains without verifying that the referenced package actually exists, is owned by the vendor, or has ever been reviewed by anyone [1][2]. This is a variant of the slopsquatting problem CSA documented earlier in 2026, but it inverts the attack surface — instead of waiting for a model to hallucinate a plausible-sounding package name, an attacker can read a company's own published `llms.txt` file, identify the exact unregistered name the company's own documentation invites agents to install, and claim it with near-certainty that it will eventually be executed [4]. Because the compromise runs through an approved AI tool invoking a standard package manager against a trusted registry, it does not resemble malware to conventional endpoint or network controls, and organizations that have adopted `llms.txt` to reduce agent hallucination may have inadvertently published an attacker's target list.

Background

`llms.txt` is a proposed convention, first put forward by Answer.AI co-founder Jeremy Howard in September 2024, for a plain-text or Markdown file published at a website's root that gives AI systems a curated summary of a site's documentation, structured specifically for consumption by language models rather than human readers or search crawlers [5]. A companion file, `llms-full.txt`, typically contains the full documentation content inline so that an agent does not need to crawl the site at all.

Neither file format has been ratified by the IETF or the W3C, but adoption has grown quickly since 2024 as AI coding agents such as Claude Code, Codex, Cursor, and various open-source assistants have become common participants in software development workflows. Vendors appear to have adopted the format on the premise that giving an agent authoritative, structured text will reduce the incorrect API calls or installation instructions a model might otherwise invent — though this benefit is asserted by format proponents rather than independently measured [5].

That adoption logic — give the agent authoritative text so it stops guessing — is exactly what creates the exposure. Security researcher Alon Hertz, describing the work in a disclosure published on Medium and covered independently by Schneier on Security, Compendia Labs, and multiple security trade outlets in late August and early September 2026, scanned 6,214 live domains belonging to defense contractors, Fortune 500 companies, and major technology firms and found 8,265 `llms.txt` and `llms-full.txt` files among them, a figure GBHackers' independent account puts closer to 8,565 [1][2][3][6]. Of those files, 120 — each on a different site — contained install commands (`pip install`, `npx`, and similar) or domain references pointing to code packages or hostnames that were not registered on any public registry at the time of the scan [1][2]. Reported figures for the total scope of the finding vary modestly across outlets — 120 files containing at least one unregistered reference, 227 individual install commands pointing at unowned code, and 237 total unclaimed artifacts including bare domain references — a discrepancy that most plausibly reflects different outlets counting different units of the same underlying finding rather than disagreement about its substance [2][6]. Hertz registered a sample of the unclaimed names and domains, hosted benign packages engineered to send a callback ("phone home") the moment they were installed, and then waited. The first callback arrived from inside a Fortune 500 environment in under an hour by some accounts and in under four minutes by others; over the following days, "a few dozen more" companies triggered the beacon, and the AI agents responsible for the installs included Claude, Codex, and Hermes [1][2][6].

One case received particular attention because it involved a named, identifiable vendor rather than an anonymized "Fortune 500 company." Clerk, an authentication platform provider, published documentation instructing developers to run `npx clerk-next-fix-auth-protection` — a package Clerk had never published under its own namespace. Researchers registered the unclaimed name (tracked in coverage as MAL-2026-11069) and demonstrated that it could be installed as though it were an official Clerk artifact, illustrating that even identity infrastructure vendors are exposed to the same failure mode as any other documentation publisher [3][6]. All three AI vendors named in the disclosure — Anthropic, OpenAI, and Nous Research — declined to comment when contacted by reporters [1].

Security Analysis

The mechanism at the center of this disclosure is narrow but consequential: an AI coding agent with permission to execute shell or package-manager commands will, when it encounters an installation instruction inside a vendor's `llms.txt` file, run that instruction without independently confirming that the named package exists, is owned by the entity that published the documentation, or has a plausible publication history. The command executes with whatever privileges the agent's host session holds — frequently a developer workstation or CI/CD runner with access to source code, credentials, and internal network segments. Node's `npx`, in particular, carries elevated risk in this context because it fetches and executes a package directly without first adding it to a project's dependency manifest, so there is no lockfile entry or code review checkpoint to catch the substitution before execution [2].

This finding sits at the intersection of two problems CSA's AI Safety Initiative has already documented separately. The first is slopsquatting: large language models hallucinate plausible-sounding package names at measurable and disturbingly consistent rates. A USENIX Security 2025 study that generated 2.23 million code samples across 16 code-generating models found that 19.7% contained at least one fabricated package name, and that when researchers re-ran identical prompts ten times, 43% of the hallucinated names reappeared in every single run [7]. That consistency is what makes the attack economically viable — an attacker does not need to guess at random; they need only register the name a model reliably invents, or in this case, the name a *company's own documentation* reliably tells its agents to install.

The second, and the one this disclosure surfaces most directly, is that `llms.txt` shifts the failure mode from model hallucination to human error at the documentation layer, with an identical downstream consequence. Unregistered package names most plausibly result from ordinary documentation drift rather than attacker action — for example, a developer may have written installation instructions for a package that was never shipped, or documentation may not have been updated after a rename or deprecation. CSA has not independently confirmed the cause of any specific unregistered reference in the disclosed dataset, but no source associated with this disclosure attributes the gap to malicious intent by the publishing vendor. What converts drift into a supply chain weapon is that `llms.txt` files are explicitly designed to be read and acted on by autonomous agents with execution privileges, rather than by a human developer who might notice that a package name looks unfamiliar before running an install command. The 8,265-to-120 ratio Hertz reported — roughly 1.5% of scanned `llms.txt` files containing at least one unregistered reference — is a base rate that merits attention given how many companies have adopted the format specifically to make their products more legible to agents that install software autonomously [1][2].

The detection challenge compounds the exposure. A malicious package installed this way arrives through a chain that looks, at every layer, like sanctioned developer activity: an approved AI coding tool invokes a standard package manager, which connects to a legitimate, trusted public registry, which serves a package that was registered through the registry's normal (and typically unauthenticated) publish process. Endpoint detection tools tuned to flag unusual binaries or unexpected network destinations have little reason to intervene, because nothing about the installation pathway deviates from a developer's ordinary workflow — only the content of the package differs from what the vendor intended to publish [6]. This is structurally similar to concerns CSA's AI Safety Initiative raised in its analysis of the broader package registry ecosystem, where publish-first, review-later registry models and AI agents' autonomous dependency resolution were identified as compounding weaknesses that self-propagating supply chain campaigns had already begun to exploit at scale in the first half of 2026 [8].

Recommendations

Immediate Actions

Organizations that publish an `llms.txt` or `llms-full.txt` file should audit every install command, package reference, and domain name in that file against the corresponding public registry or DNS record within the next reporting cycle, treating the document with the same scrutiny normally reserved for a public API reference. Any referenced package that is not currently registered under the organization's own namespace should be registered defensively, even if it is not yet in active use, to close the window an attacker would otherwise exploit. Security and platform teams should also inventory which internal AI coding agents have shell or package-manager execution privileges and confirm that those agents are not permitted to run install commands sourced from external documentation without a human confirmation step.

Short-Term Mitigations

Enterprises using AI coding agents should require that any package installation the agent proposes be checked against registry metadata — registration date, publisher identity, and download history — before execution, rather than trusting the name because it appeared in ostensibly authoritative vendor documentation. Software composition analysis tooling should be configured to flag recently registered packages and packages with no prior download history, since a package registered specifically to exploit an `llms.txt` reference will typically have neither. Lockfiles should be committed to source control and verified against known-good hashes so that a substituted package cannot silently replace an

expected dependency in a subsequent build. Organizations that maintain their own `llms.txt` files should establish an ownership and review process for that document equivalent to what already exists for public API documentation, since it is now a machine-actionable artifact rather than passive reference text.

Strategic Considerations

The `llms.txt` disclosure is best understood as one instance of a broader pattern: any text that an AI agent is permitted to convert directly into an executed command becomes part of the software supply chain and needs the review, provenance, and change-control discipline that supply chain security already applies to code and dependencies. This extends past a single company's own documentation to third-party and community-maintained `llms.txt` files an agent might also consult, and to any other agent-facing text channel — README files, MCP tool descriptions, or agent skill files — that an organization allows its coding agents to treat as trusted instruction rather than untrusted external input. Extending software bill of materials (SBOM) and provenance practices to cover AI agent dependency decisions, not only human-authored ones, is a necessary long-term step, as is building organizational habits that assume vendor-published, agent-facing documentation can drift out of sync with what is actually registered and available.

CSA Resource Alignment

This disclosure extends findings CSA's AI Safety Initiative has already published on AI coding agents as unaudited actors in the software supply chain. **"AI Coding Agents: An Unaudited Supply Chain Node"** (July 2026) established the core framing this note builds on directly: that coding agents write code, select dependencies, and execute build commands with little of the scrutiny long applied to human contributors, and it specifically recommended verifying AI-suggested package names against target registries before installation — precisely the control gap the `llms.txt` disclosure demonstrates in practice, only with the vendor's own documentation supplying the unverified name instead of the model's imagination [9].

"HalluSquatting: AI Hallucinations Weaponized for Botnet Delivery" (July 2026) is the closest structural analog: it documented how the *predictability* of LLM hallucinations — the same 43% repeat rate cited above — lets an attacker pre-register a resource name before any victim agent requests it,

turning consistency into a scalable, untargeted attack vector rather than a random collision [10]. The `llms.txt` case is functionally the same play with a higher hit rate, because the vendor's documentation removes the need to predict what a model will hallucinate at all.

"AI Package Registry Crisis: Unguarded Critical Infrastructure" (June 2026) provides the registry-level context for why this attack succeeds once a malicious package is in place: npm and PyPI operate on a publish-first, review-later model in which a package can be live and installable long before any abuse review occurs, and the note's finding that a majority of MCP registries failed to detect deliberately planted malicious uploads underscores that registry-side controls should not be treated as the sole backstop [8]. Finally, this note's foundational mechanism traces to CSA's **"Slopsquatting: AI Code Hallucinations Fuel Supply Chain Attacks"** (April 2026), which first documented the hallucination-to-registration attack pattern this disclosure applies against a new class of source text [4]. Organizations mapping controls against these risks should reference the AI Supply Chain Security and Application & Interface Security domains of CSA's **AI Controls Matrix (AICM) v1.1**, which provides the control baseline for dependency verification, provenance tracking, and least-privilege execution that the recommendations above operationalize [11].

References

[1] Schneier, Bruce. "AI Coding Agents Are Installing Unknown/Untrusted Code on Corporate Networks." Schneier on Security, September 2026.

[2] "DK Dark Knight." "The llms.txt Trust Model — Dispatch." Compendia Labs, August 29, 2026.

[3] "Researcher Alon Hertz Tricked Claude, Codex and Hermes Into Running Malware." Startup Fortune, September 2026.

[4] Cloud Security Alliance AI Safety Initiative. "Slopsquatting: AI Code Hallucinations Fuel Supply Chain Attacks." CSA, April 19, 2026.

[5] "llms.txt." llmstxt.org, accessed September 2026.

[6] "Researchers Execute Code Inside Fortune 500 Companies via AI Agent llms.txt Files." GBHackers, September 2026.

[7] Spracklen, Joseph, et al. "We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs." 34th USENIX Security Symposium, 2025.

[8] Cloud Security Alliance AI Safety Initiative. "AI Package Registry Crisis: Unguarded Critical Infrastructure." CSA, June 23, 2026.

[9] Cloud Security Alliance AI Safety Initiative. "AI Coding Agents: An Unaudited Supply Chain Node." CSA, July 8, 2026.

[10] Cloud Security Alliance AI Safety Initiative. "HalluSquatting: AI Hallucinations Weaponized for Botnet Delivery." CSA, July 12, 2026.

[11] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." CSA, 2026.