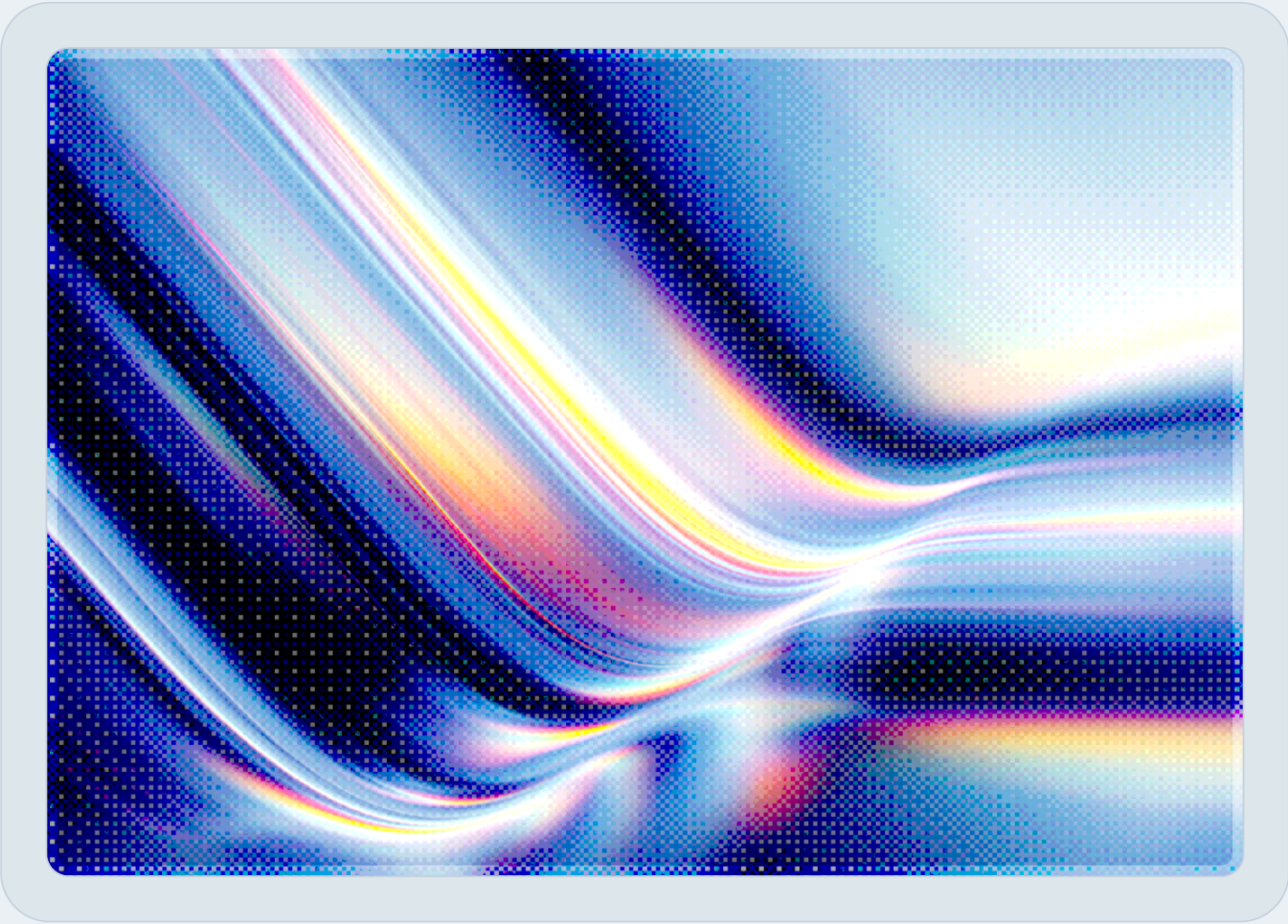
CSAI Foundation | Cloud Security Alliance

AI Coding Agents: An Unaudited Supply Chain Node

2026-07-08

AI-assisted Rapid Research



CSAI

cloud security alliance®

Key Takeaways

AI coding agents have inserted themselves into the software supply chain as an actor that writes code, selects dependencies, and executes build commands with little of the scrutiny traditionally applied to human contributors or third-party packages. At Google, 75% of new code is now AI-generated and approved by engineers, up from 50% a year earlier [12], and a separate study spanning 4.2 million developers found AI-authored code accounted for 26.9% of production commits by February 2026, up from 22% the prior quarter [13]. That volume raises the question of whether the controls organizations use to verify what ends up in a build have kept pace — a gap the incidents below suggest is real, even without survey data quantifying it. Attackers have already adapted: hallucinated package names invented by AI agents are being pre-registered and weaponized in a technique researchers call slopsquatting [3], and prompt injection delivered through something as ordinary as a GitHub issue title has been used to hijack an AI agent's CI/CD privileges and push a malicious package to production [4]. The common thread is that the agent itself, along with the models, tool integrations, and autonomous decisions behind it, has become a supply chain dependency that the incidents above suggest organizations are not yet inventorying, signing, or auditing the way they do a third-party library. Closing that gap requires extending existing software transparency and provenance practices, including SBOM generation and lineage tracking, to cover the agents and models that now author a growing share of enterprise code, and it requires treating agent-driven package resolution and CI/CD execution as privileged actions rather than routine developer convenience [5].

Background

Software development has moved through several distinct eras, each of which shortened the distance between an idea and a deployed application. Waterfall development optimized for predictability at the cost of speed; Agile and DevOps introduced iterative sprints and continuous deployment that compressed delivery cycles from years to hours; and the current era, sometimes called "vibe coding," lets a developer describe an objective in natural language and receive a working application through iterative conversation with an AI agent [2]. Morey Haber, Chief Security Advisor at BeyondTrust, has described this progression as approaching "the speed of thought," with the caveat that security has not kept pace: AI-generated code can appear functional while carrying vulnerabilities, licensing problems, or excessive privileges that would have been caught by the friction of a slower process [2]. Haber compares the

resulting dynamic to shadow IT, in that the same agents that make individual developers dramatically more productive also expand the organization's risk surface in ways security teams cannot see until something breaks.

The Model Context Protocol, an open standard for connecting AI agents to external tools and data sources, is a useful marker of how quickly this shift has become structural rather than experimental. Launched roughly 20 months before mid-2026 [1], MCP is now embedded in build pipelines at organizations that had no equivalent integration point a year and a half earlier — a shift this note treats as introducing a new class of dependency in its own right. Industry analysts have started to treat this as a distinct market problem: Gartner published its inaugural Magic Quadrant for Software Supply Chain Security in June 2026, evaluating 18 vendors on their ability to protect software delivery pipelines from upstream risk that now explicitly includes AI-generated and AI-assisted code alongside traditional open source dependencies [6]. That a stand-alone analyst category exists at all signals that at least part of the vendor and analyst community no longer treats AI-authored code as an edge case to be handled by existing tooling.

None of this displaces the supply chain risks organizations were already managing. The industry is still absorbing Shai-Hulud, a self-propagating npm worm first disclosed by CISA in September 2025 that harvested credentials from compromised build environments and used them to republish additional malicious packages without any human operator in the loop; a second wave in late November 2025 compromised hundreds of additional npm packages and touched more than 27,000 repositories [7][8]. Incidents like SolarWinds, Log4Shell, and the XZ Utils backdoor already demonstrated that a single compromised dependency, build system, or maintainer account can cascade across thousands of downstream consumers [1]. What has changed is that the list of things capable of introducing that kind of compromised dependency now includes the AI agent itself — and the incidents in this note indicate that provenance and access models have not caught up.

Security Analysis

The Agent as an Unaudited Actor

Traditional software supply chain security assumes a human developer makes deliberate, reviewable decisions about which dependencies to add and which commands to run, and that automated tooling operates within a narrow, well-understood scope. AI coding agents break both assumptions. An agent tasked with implementing a feature will typically write code, identify what it believes the code depends on, and invoke a package manager to install those dependencies, often without a human reviewing the specific packages before they land in the environment [1][9]. The provenance question this raises

extends beyond the code artifact a human eventually merges: it now has to account for the model that generated the code, the agent framework and tool integrations that carried out the installation, and the prompts or context that shaped both. As one industry analysis put it, "the risk lives less in the code a team writes and more in everything that produces it" [1]. Attackers have taken notice, planting prompts and documentation designed to steer what an agent writes or installs, a manipulation surface that did not exist when a human was the only actor making dependency decisions.

Hallucinated Dependencies and Slopsquatting

The clearest expression of this new risk is slopsquatting, in which attackers register the fabricated package names that large language models predictably invent and wait for an agent, or a developer copying an agent's suggestion, to install them. A CSA AI Safety Initiative research note published in April 2026 analyzed 2.23 million AI-generated code samples and found that 19.7% contained a fabricated package name, totaling 205,474 unique hallucinations; open-source models hallucinated at an average rate of 21.7%, compared with 5.2% for commercial models, and some open configurations exceeded 33% [3]. Critically, these hallucinations are not random noise that would be hard for an attacker to exploit: 43% of hallucinated package names reappeared across all ten runs of an identical prompt in the CSA study, which suggests an attacker needs only to query a popular model a handful of times to identify which fabricated names are worth registering [3].

Several incidents illustrate how this plays out in practice. In February 2026, a malicious npm package named unused-imports exploited confusion with the legitimate eslint-plugin-unused-imports and accumulated roughly 233 weekly downloads before detection [3]. An AI-recommended installation command for a package called huggingface-cli was copied into official documentation published by Alibaba and drove more than 30,000 downloads over three months, despite referring to a package that behaved differently than the one developers intended [3]. Perhaps most illustrative is react-codeshift: a name that a model invented by conflating two real tools, jscodeshift and react-codemod, first appeared in a single commit that dumped 47 AI-generated "Agent Skills" into a public repository in October 2025, then propagated through 237 forked repositories as developers and agents copied the skill files without verification; by January 2026, a security researcher registered the name to preempt an attacker, but the package was already receiving a steady trickle of real installation attempts from autonomous agents faithfully following the hallucinated instructions [3][10]. In March 2026, the same underlying trust gap was exploited more directly when the TeamPCP campaign used stolen credentials to compromise the legitimate PyPI packages litellm and telnyx, packages that sit inside the tooling many organizations use to run their own AI agents [3].

Agents as CI/CD Attack Vectors

A second class of risk emerges when AI agents are granted execution privileges inside build and deployment pipelines. The Clinejection incident, documented in a CSA AI Safety Initiative research note published in March 2026, showed how an agent's own automation could be turned against the pipeline it was meant to help secure. The Cline coding assistant, used by more than 5 million developers, had added an AI-powered issue-triage workflow that read the text of incoming GitHub issues and granted the agent broad tool access, including Bash, Write, and Edit, to any user who could open an issue [4]. Security researcher Adnan Khan disclosed on February 9, 2026 that an attacker could embed instructions in an issue title to make the agent execute arbitrary commands, and that this could be chained with a GitHub Actions cache-poisoning technique to persist a compromise across otherwise unrelated workflows [4] [9]. Eight days later, on February 17, 2026, an unknown actor used this exact chain in production: a still-valid npm token was used to publish a modified version of the Cline package that silently installed an unrelated agent tool on any developer machine that updated during an eight-hour window, reaching an estimated 4,000 developers before the legitimate maintainers shipped a fix [4]. The CSA research note further found that the underlying configuration pattern, granting broad tool access to an AI agent triggered by unauthenticated external input, was present at five Fortune 500 companies and at Google, suggesting the Cline incident was not an isolated misconfiguration but a common way organizations had been standing up agentic automation [4].

Scale Compounds the Problem

Both classes of risk are amplified by the sheer volume of code now moving through this pathway. When AI agents author roughly a quarter to three-quarters of new code at leading organizations, even a low per-instance hallucination or misconfiguration rate translates into a large absolute number of exposures [12][13]. Review capacity has not been shown to scale at the same rate, which is the operating assumption behind the recommendations below. The following table summarizes the incidents discussed above and the specific supply chain mechanism each one exploited.

Incident	Date	Vector	Mechanism
Shai-Hulud npm worm	Sept. 2025 (v2 Nov. 2025)	Compromised maintainer credentials	Self-propagating worm harvests build credentials, republishes malicious packages [7][8]
react-codeshift	Oct. 2025–Jan. 2026	Hallucinated package name	AI-generated "Agent Skills" instructed agents to install a package that did not yet exist [3][10]

Incident	Date	Vector	Mechanism
Clinejection	Feb. 2026	Prompt injection + cache poisoning	Malicious GitHub issue title hijacked an AI triage agent's CI/CD privileges [4][9]
TeamPCP campaign	March 2026	Stolen credentials	Legitimate AI-tooling packages (litellm, telnyx) compromised and republished [3]

Recommendations

Immediate Actions

Organizations should verify every AI-suggested package name against the target registry before installation rather than trusting an agent's dependency resolution outright, and should confirm that a package's registration date, download history, and publisher identity are consistent with a legitimate project before allowing it into a build [3]. Any AI workflow that interpolates unsanitized external input, such as the text of a GitHub issue or pull request, into an agent's prompt should have its tool permissions restricted immediately, removing broad access to Bash, Write, or Edit unless that access is demonstrably required, and replacing wildcard user permissions with scoped access limited to authenticated collaborators [4]. Lockfiles should be committed to source control and verified against known-good hashes on every build; pinned, hash-verified dependencies would have limited exposure for developers who did not explicitly pull the malicious Clinejection release during its eight-hour window [4].

Short-Term Mitigations

Security teams should configure software composition analysis tooling to flag any package registered shortly before its first use inside the organization, a pattern consistent with an attacker pre-registering a name an agent is likely to hallucinate [3]. AI-generated codebases destined for production should receive a dedicated audit pass before deployment, and organizations should begin producing genuine software bills of materials for that code rather than treating SBOM generation as a practice reserved for traditionally authored software [3][5]. Long-lived personal access tokens used by CI/CD automation

should be replaced with short-lived, narrowly scoped credentials, and cache isolation should be enforced between workflows that operate at different privilege levels so that a compromise in one pipeline cannot silently propagate to another [4].

Strategic Considerations

Over the medium term, organizations should extend their existing lineage and provenance tracking to treat the AI models, agent frameworks, and tool integrations used in development as dependencies in their own right, not merely as productivity tools sitting outside the supply chain boundary [1][5]. Procurement processes for AI coding agents should include supply chain safety criteria, such as how a vendor's agent handles package resolution, what guardrails exist against executing unreviewed commands, and whether the vendor has a track record of responding to disclosed prompt injection or cache-poisoning issues [3][4]. Prompt injection should be formally modeled as a threat vector in CI/CD security reviews and incorporated into tabletop exercises, and vulnerability triage should prioritize exploitability over raw finding volume, since AI-accelerated code generation and AI-accelerated vulnerability discovery are both increasing the number of findings competing for a security team's limited attention [1][4].

CSA Resource Alignment

This research note builds directly on two CSA AI Safety Initiative research notes that examined the specific incidents analyzed here. "Slopsquatting: AI Code Hallucinations Fuel Supply Chain Attacks" quantified the hallucination rates behind the unused-imports, huggingface-cli, and react-codeshift incidents and established the immediate, short-term, and strategic mitigations for package-level risk that this note extends to the broader question of the agent as a supply chain actor [3]. "Clinejection: Prompt Injection in GitHub Issue Titles Enables CI/CD Cache Poisoning and Supply Chain Compromise" documented how an AI agent's own CI/CD tool access became the attack vector, and its finding that the underlying misconfiguration pattern appeared at multiple Fortune 500 companies supports this note's conclusion that agent-driven pipeline execution needs to be governed as a privileged capability rather than a developer convenience [4].

The CSA CISO Community's "The 'AI Vulnerability Storm': Building a 'Mythos-Ready' Security Program" situates these package- and pipeline-level risks within a broader shift in the attacker-defender balance, explicitly naming the "agentic supply chain," including MCP servers, plugins, and skills, as a governance gap alongside its recommendations to generate real SBOMs, minimize unnecessary software, and define scope boundaries and blast-radius limits for AI agents before they reach production [11]. Those

recommendations map directly onto the strategic considerations in this note. Finally, "Securing the Software Supply Chain: Transparency in the Age of the Software Driven Society" provides the SBOM, software composition analysis, and CI/CD security foundations that organizations already have in place; the task ahead is extending those existing practices to cover AI agents and the code they generate rather than building an entirely new discipline [5]. Organizations working from CSA's AI Controls Matrix (AICM) v1.1 should map the risks in this note primarily to the Threat & Vulnerability Management (TVM) and Application & Interface Security (AIS) domains, alongside Governance, Risk & Compliance (GRC) controls for the procurement and oversight recommendations above.

References

[1] The Hacker News. "What Changes When Your Software Supply Chain Includes AI Writing Your Code?." The Hacker News, July 2026.

[2] Haber, Morey J. "Software Is Now Written at the Speed of Thought. Security Isn't.." BleepingComputer, July 6, 2026.

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

[4] Cloud Security Alliance AI Safety Initiative. "Clinejection: Prompt Injection in GitHub Issue Titles Enables CI/CD Cache Poisoning and Supply Chain Compromise." Cloud Security Alliance, March 10, 2026.

[5] Hughes, Chris. "Securing the Software Supply Chain: Transparency in the Age of the Software Driven Society." Cloud Security Alliance, 2025.

[6] ReversingLabs. "2026 Gartner Magic Quadrant for Software Supply Chain Security: 5 Takeaways." ReversingLabs, June 2026.

[7] CISA. "Widespread Supply Chain Compromise Impacting npm Ecosystem." Cybersecurity and Infrastructure Security Agency, September 23, 2025.

[8] The Hacker News. "Second Sha1-Hulud Wave Affects 25,000+ Repositories via npm Preinstall Credential Theft." The Hacker News, November 2025.

[9] Snyk. "How "Clinejection" Turned an AI Bot into a Supply Chain Attack." Snyk Blog, 2026.

[10] Aikido Security. "Agent Skills Are Spreading Hallucinated npx Commands." Aikido Blog, 2026.

[11] Evron, Gadi, Rich Mogull, and Robert T. Lee. "The "AI Vulnerability Storm": Building a "Mythos-Ready" Security Program." Cloud Security Alliance, April 12, 2026.

[12] Swartz, Jon. "Google CEO Says 75% of New Code is AI-Generated." DevOps.com, April 29, 2026.

[13] ShiftMag. "93% of Developers Use AI. Why Is Productivity Only 10%?." ShiftMag, February 18, 2026.

