CSAI Foundation

Cloud Security Alliance AI Safety Initiative

Slopsquatting: AI Code Hallucinations Fuel Supply Chain Attacks

How LLM Package Fabrication Creates Exploitable Attack Surfaces in Developer Pipelines

Unofficial AI-assisted Research

2026-04-19

Key Takeaways

A new class of software supply chain attack — coined "slopsquatting" — exploits the documented tendency of large language models to produce fabricated package names [2]. When developers install these hallucinated packages, or when AI coding agents resolve them autonomously, they may receive attacker-controlled payloads instead. The threat has moved from theoretical to confirmed: real-world malicious packages exploiting this vector have accumulated tens of thousands of downloads.

Organizations should treat slopsquatting as an active supply chain risk requiring immediate action. Lockfile pinning and package hash verification should be enforced across all CI/CD pipelines, and AI agents with package management capabilities must be prohibited from installing packages without human review or an allowlist gate. Every AI-recommended dependency should be verified against the target registry, confirming publisher identity and package registration date before adoption. Projects with AI-assisted development should require a Software Bill of Materials (SBOM) for all AI-generated codebases entering production.

Background

The term "slopsquatting" was coined by Seth Larson, developer-in-residence at the Python Software Foundation, by analogy with typosquatting — the long-established practice of registering domain or package names that mimic popular ones, betting on human typographic errors [1]. Slopsquatting replaces the human typo with an AI hallucination: the language model confidently recommends a package name that never existed, and an attacker pre-registers that name on npm, PyPI, or another public registry before a legitimate package can claim it.

The scale of the underlying hallucination problem was quantified in the academic paper "We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs," presented at USENIX Security 2025 by researchers at the University of Texas at San Antonio, the University of Oklahoma, and Virginia Tech [2]. The researchers generated 2.23 million code samples using 16 popular code-generating models across Python and JavaScript. Of those samples, 440,445 — 19.7% — contained at least one hallucinated package name. Across those hallucinations, the study identified 205,474 unique fabricated package names. The breadth of this attack surface is not incidental: nearly one in five AI-assisted code suggestions points to a package that does not exist.

Hallucination rates vary considerably by model class. Open-source models produced hallucinated packages at an average rate of 21.7%, while commercial models averaged 5.2% [2]. Models in the CodeLlama family exceeded a 33% hallucination rate in some configurations. GPT-4 Turbo achieved the lowest measured rate in the study at 3.59%. These differences are significant for organizational policy, but even the best-performing commercial models hallucinate at rates sufficient to represent a meaningful, persistent risk at scale. It should be noted that these rates were measured against specific model versions studied in 2025; rates for more recent model releases may differ.

What makes slopsquatting more dangerous than its namesake is predictability. When the researchers re-ran identical prompts ten times each, 43% of hallucinated package names reappeared on every single run, and 58% reappeared on more than one [2]. By this measure, 39% of hallucinated names were unique to a single prompt run, while the remaining 61% appeared across multiple runs. This consistency means that an attacker who identifies which hallucinations a given model produces can register those names with high confidence that future developers using the same model and same prompts will receive the malicious package.

Security Analysis

Hallucination Taxonomy and Why It Is Hard to Detect

Research from USENIX Security 2025 classified hallucinated package names into three categories: conflations (38%), in which the model merges two real package names — such as combining `jscodeshift` and `react-codemod` into `react-codeshift`; typo variants (13%), which closely resemble real packages; and pure fabrications (51%), which are contextually plausible but entirely invented [2]. The distribution matters because conflation-style hallucinations are semantically convincing. A developer reviewing AI-generated code who is unfamiliar with a specific package has no obvious signal that `react-codeshift` is not a real, widely-used tool. This plausibility is precisely what makes slopsquatting viable where a random typosquatting attempt might raise suspicion.

An additional dimension of cross-registry confusion compounds the problem: researchers found that 8.7% of Python packages hallucinated by models actually exist in the npm registry [2]. A developer working in Python could unknowingly install a JavaScript-ecosystem package — or an attacker could register that name on PyPI as a payload delivery mechanism.

Confirmed Real-World Exploitation

Slopsquatting is no longer a theoretical risk. Several confirmed incidents demonstrate that attackers are actively exploiting this vector.

The `unused-imports` npm package represents one of the clearest documented cases. AI models hallucinate this name instead of the legitimate `eslint-plugin-unused-imports`. As of early February 2026, the malicious package was still available and recording approximately 233 weekly downloads despite having been security-held by npm [3].

Security researcher Bar Lanyado documented that Alibaba had copied an AI-recommended installation command for the `huggingface-cli` package into public repository documentation without apparent verification, a pattern that amplified the package's apparent legitimacy and contributed to its accumulation of over 30,000 downloads within three months [3]. This incident illustrates how institutionally trusted documentation can inadvertently propagate hallucinated package names to a wide downstream audience.

The `react-codeshift` package demonstrates how AI coding agents amplify the risk. Security researcher Charlie Eriksen discovered this conflation-style hallucinated package propagating through 237 repositories via AI-generated agent skills, accumulating daily downloads driven not by human developers manually copying code but by autonomous agents executing their own generated outputs [3].

Related Supply Chain Activity

While not a slopsquatting incident, the TeamPCP supply chain campaign that reached PyPI in March 2026 illustrates the broader convergence of supply chain attacks with AI ecosystem components. The campaign compromised two widely-used Python packages – `litellm`, a popular proxy layer for LLM API providers, and `telnyx`, a telephony SDK – through credential theft and reuse rather than AI package hallucination [4]. The targeting of `litellm`, itself a dependency in many AI development pipelines, underscores that adversaries are increasingly focusing on AI tooling infrastructure as a high-value supply chain target, regardless of the specific attack vector employed.

Agentic AI as an Accelerant

The slopsquatting threat enters a qualitatively different risk tier with the deployment of autonomous AI coding agents. Traditional slopsquatting required a human developer to review AI-generated code, notice a package recommendation, and choose to install it – providing at least an implicit checkpoint.

Agentic systems eliminate that checkpoint. As security researcher Andrew Nesbitt observed in April 2026, agents resolve packages programmatically without a human glancing at the name to notice something is off [5]. An agent tasked with building a feature will generate the code, identify its own generated dependencies, and invoke the package manager — often without any step in that sequence touching a human reviewer.

In agentic contexts, post-install scripts — one of the most common payload delivery mechanisms in malicious packages — execute in an environment that typically holds API keys, cloud provider credentials, source control tokens, and model API keys. This represents a substantially broader credential surface than an individual developer's local machine: a malicious package installed by an agent may compromise the entire pipeline, including secrets that provide access to production infrastructure and to the AI model APIs powering the agentic system itself.

The acceleration problem extends to propagation speed. In a human-driven development workflow, a single compromised package takes time to spread — a developer must notice the AI suggestion, install the package, commit it, and have that commit reviewed and merged. In an agentic workflow operating at machine speed, a hallucinated dependency can spread across a multi-agent codebase within a single task execution cycle, and the agent may have committed and pushed the dependency before any monitoring system triggers.

Interaction with Vibe Coding Practices

The cultural shift toward "vibe coding" — in which developers describe desired functionality to AI tools and accept generated implementations with minimal manual review — materially worsens the slopsquatting risk profile. The practice often proceeds on an implicit assumption that AI-generated implementations are substantially correct — an assumption that the 19.7% package hallucination rate documented by the USENIX research [2] calls into serious question for dependency management specifically. Security teams should treat vibe coding as a risk amplifier that effectively lowers the organization's bar for package verification across all AI-assisted development workflows.

Recommendations

Immediate Actions

Organizations should treat any AI-generated `import` statement, `require()` call, or package manager invocation as untrusted input subject to validation before execution. Specifically, every package name produced by an AI code assistant should be verified against the target registry before installation, confirming the package exists, was registered before the project's own start date, and has a publisher with an established track record. Commercial tooling from vendors including Socket.dev, Snyk, and Aikido can automate this check and integrate directly into IDE and CI/CD workflows. (Note: Aikido Security is also cited as a primary source for incident examples earlier in this document.)

Lockfiles should be committed to source control and verified against known-good hashes on every dependency installation in automated pipelines. Dependency trees should be regenerated from an approved lockfile rather than from scratch. This does not prevent the initial hallucinated package from entering a lockfile, but it prevents attackers from substituting a different payload in subsequent installs once a legitimate package name is claimed.

AI agents with package management capabilities must operate under an explicit allowlist, with any attempted installation of a package not on the allowlist routing to human review before execution. Organizations that cannot implement an allowlist should at minimum disable automated package installation in agentic environments until detection capabilities are in place.

Short-Term Mitigations

Development teams should establish a process for auditing AI-generated codebases before production deployment, with specific attention to dependency manifests. Software composition analysis (SCA) tooling should be configured to flag packages registered within the prior 30 to 90 days before the organization's first use, which may indicate a recently registered package exploiting AI hallucinations and warrants manual review. Package registration dates are publicly available on npm and PyPI and can be queried programmatically.

AI coding assistant configurations should be reviewed for temperature and sampling parameters. Industry reporting suggests a measurable correlation between higher temperature settings — which increase output randomness — and higher hallucination rates [1]. Where tooling allows, lowering temperature for code generation tasks is a low-cost mitigation. It is also worth testing whether prompts

that explicitly require package enumeration and verification steps reduce hallucination rates in a given model configuration, though the effect varies by model and has not been uniformly validated across tools.

Organizations should produce and maintain SBOMs for all AI-generated codebases entering production. SBOMs provide traceability when a malicious package is later identified and enable rapid impact assessment across the portfolio. SBOM generation should be a mandatory step in CI/CD pipelines for any project with AI-assisted development.

Strategic Considerations

At the organizational level, slopsquatting represents a structural challenge to the practice of delegating code generation to AI without proportionate investment in AI output verification. The economics of the threat are asymmetric: an attacker can monitor public registry activity, identify consistently hallucinated package names at scale, and register them for minimal cost. The defender must verify every AI-generated dependency. Investment in developer tooling that makes verification automatic and low-friction is more sustainable than policies that rely on manual developer vigilance.

AI coding agent procurement and deployment should include explicit evaluation criteria for supply chain safety. Agents that can install packages should be evaluated on whether they implement verification steps, whether their actions are logged for audit, and whether the vendor provides a mechanism to restrict package installation to pre-approved registries or namespaces. These criteria should be part of AI vendor risk assessment processes.

Organizations should also monitor for AI-generated documentation in public repositories — including in their own codebases and in vendor-supplied examples — that includes package installation commands. The `huggingface-cli` incident demonstrates that institutionally trusted documentation can inadvertently propagate a hallucinated package name to a wide downstream audience. Automated scanning of documentation for package manager commands (`pip install`, `npm install`, `cargo add`) against a verified registry can surface these risks before they affect downstream users.

CSA Resource Alignment

The CSA AI Controls Matrix (AICM) provides a directly applicable governance framework for slopsquatting risk management. The AICM's Supply Chain Management, Transparency, and Accountability domain — one of 18 domains spanning 243 control objectives — addresses provenance

tracking, third-party dependency validation, and software bill of materials requirements for AI systems [6]. Organizations implementing AICM controls in this domain will have a foundational structure for managing hallucinated dependency risk as part of a broader AI supply chain program.

The CSA MAESTRO framework for agentic AI threat modeling addresses supply chain compromise across multiple layers of the agentic stack, including risks to the AI agent software environment [7]. MAESTRO's recommended mitigations for this threat class — including secure dependency management, fine-grained access control, and SBOM-based provenance tracking — align directly with the technical controls described in the recommendations above. Organizations deploying AI coding agents should use MAESTRO as the threat modeling foundation for evaluating their exposure to slopsquatting and related dependency attacks.

The CSA Cloud Controls Matrix v4.1 (CCM) introduced enhanced Supply Chain Management controls covering supply chain risk management policies and Service Bill of Materials transparency, providing a complementary governance layer for organizations that have not yet adopted the AICM [8]. For organizations already operating under CCM, the supply chain controls in v4.1 offer a practical entry point for addressing AI-specific dependency risks without requiring a full transition to the AICM.

The CSA STAR program provides the assurance mechanism through which organizations can evaluate AI vendor practices on supply chain security. While existing STAR assessments may not yet include hallucination-specific criteria, security teams should proactively request vendor evidence on hallucination mitigation practices, package verification capabilities, and disclosure policies as part of AI procurement due diligence. When procuring AI coding assistants or agentic development platforms, these requests should accompany STAR-level attestation or equivalent vendor self-assessment documentation.

References

[1] Seth Larson / Infosecurity Magazine. "AI Hallucinations Create 'Slopsquatting' Supply Chain Threat." Infosecurity Magazine, 2025.

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

[3] Aikido Security. "Slopsquatting: The AI Package Hallucination Attack Already Happening." Aikido Security Blog, 2026.

[4] Datadog Security Labs. "LiteLLM and Telnyx Compromised on PyPI: Tracing the TeamPCP Supply Chain Campaign." Datadog Security Labs, March 2026.

[5] Andrew Nesbitt. "Package Security Problems for AI Agents." nesbitt.io, April 8, 2026.

[6] Cloud Security Alliance. "AI Controls Matrix." CSA, 2025.

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

[8] Cloud Security Alliance. "CCM v4.1: Strengthening Cloud Security." CSA, December 2025.