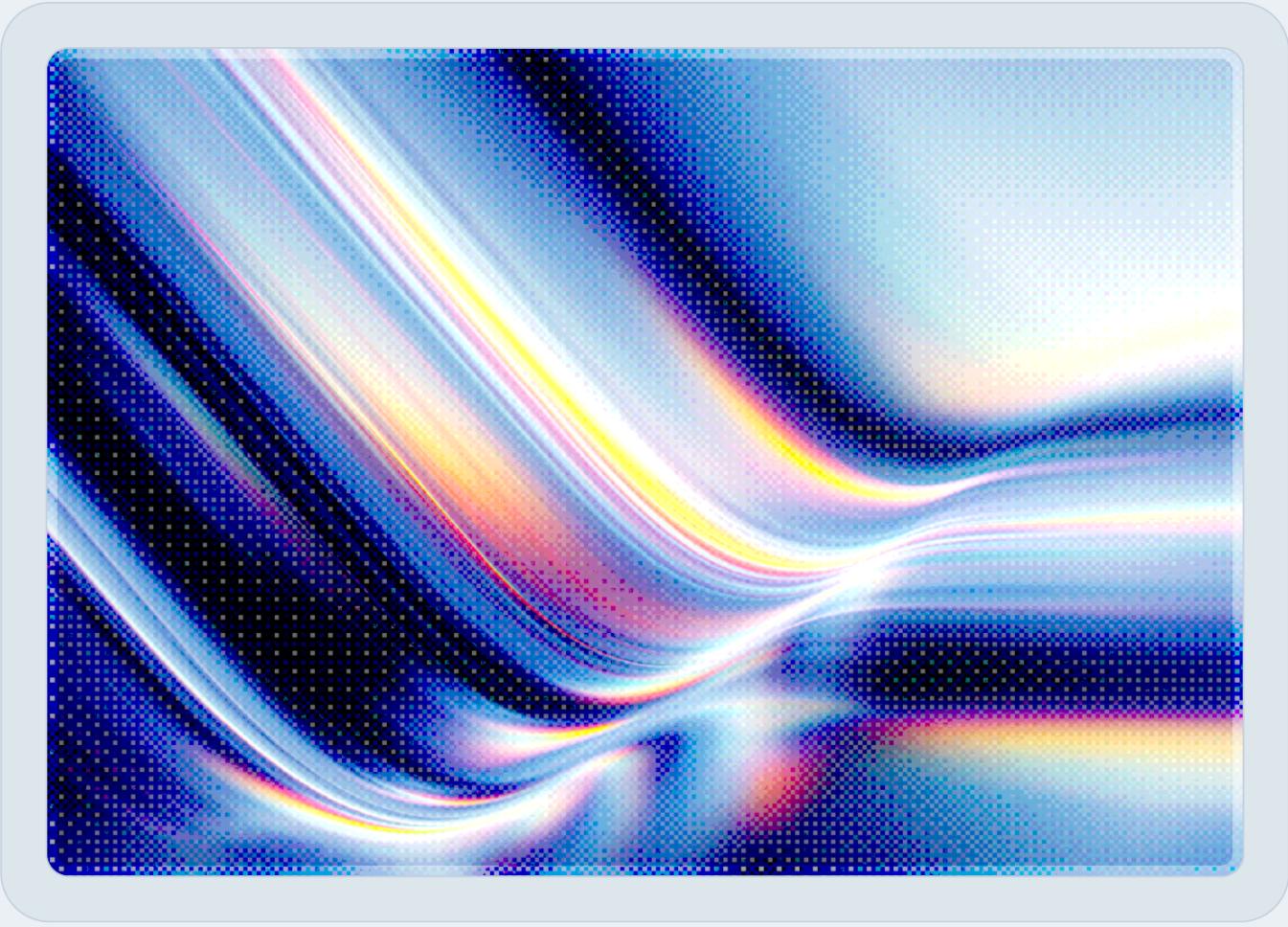
CSAI Foundation | Cloud Security Alliance

Slopsquatting: AI Hallucinations as Supply Chain Attack Vectors

How AI Coding Assistants Introduce Phantom Dependencies into Software Build Pipelines

2026-05-03

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- Large language models used for code generation hallucinate non-existent package names at measurable rates: 5.2% for commercial models and 21.7% for open-source models, according to peer-reviewed research published at USENIX Security 2025 [1].
- Slopsquatting is the practice of pre-registering these hallucinated package names in public repositories (PyPI, npm) with malicious payloads, ready to be silently installed when a developer copies AI-generated code and runs it [2].
- Because hallucinations are reproducible — 43% of hallucinated package names reappear on every repeated query, and 58% reappear on at least one of ten runs — attackers can map and occupy a predictable set of phantom namespaces rather than guessing blindly [1].
- As of this writing, no publicly confirmed mass-exploitation campaign exploiting slopsquatting has been reported, but the pre-conditions for one — abundant hallucinated names, trivially easy registry publishing, and widespread AI coding adoption — are fully in place [3].
- Immediate mitigation requires treating all AI-suggested package names as untrusted input: verify existence and provenance before installation, instrument CI/CD pipelines with software composition analysis (SCA) tools, and establish dependency allow-lists for production environments.

Background

Software supply chain attacks have become among the most damaging categories of security incidents over the past several years, with well-known campaigns demonstrating how a single compromised upstream dependency can cascade across thousands of downstream consumers. The established attacker playbook for package repository abuse has historically relied on typosquatting (registering names one character away from popular libraries), dependency confusion (exploiting private-versus-public package resolution logic), and direct compromise of legitimate package maintainer accounts. Each of these techniques requires either a human error — a mistyped package name — or active access to a trusted account.

Slopsquatting introduces a structurally different mechanism. Rather than waiting for a developer to mistype, or breaking into a maintainer's credentials, the attacker can instead harvest the names that AI coding assistants have already decided to recommend — names that exist nowhere in any registry — and register them first. The attack surface is not created by the attacker; it is created by the AI tool itself, and the attacker then occupies it.

The term "slopsquatting" was coined by Seth Larson, Developer-in-Residence at the Python Software Foundation, in April 2025 [2]. The name fuses "AI slop" — colloquial shorthand for low-quality, uncritically generated AI output — with "typosquatting," the long-established technique of registering near-identical names to legitimate packages. The term spread rapidly through the security community; its resonance reflects how precisely it names a class of vulnerability that emerges specifically from deploying generative AI in software development workflows.

Security Analysis

The Hallucination Problem at Scale

The most comprehensive empirical study on this attack surface published to date was conducted by Spracklen et al. at USENIX Security 2025 [1]. The researchers generated 576,000 code samples across Python and JavaScript using 16 widely deployed LLMs, then validated each suggested package name against the npm and PyPI registries. Their findings established two important baselines. Open-source or publicly accessible models hallucinated package names in 21.7% of generated samples; commercial models hallucinated at 5.2%. These are not edge-case or adversarially induced failures — they arise from routine coding prompts of the kind developers commonly issue across a normal working day.

The study also surfaced 205,474 unique hallucinated package names that did not exist on any public registry. More critically, the research demonstrated that hallucinations are not uniformly random. When the researchers re-ran identical prompts ten times, 58% of hallucinated package names reappeared at least once, and 43% reappeared on all ten runs. This reproducibility is what transforms an inconvenient LLM quirk into a viable attack primitive: an adversary does not need to enumerate all possible hallucinations, only the stable ones, which can be discovered through systematic probing of a target AI tool.

The Attack Lifecycle

A slopsquatting campaign follows a straightforward sequence that requires no novel technical capability. An attacker selects a programming language and a set of common coding tasks, then queries target AI assistants — commercial or open-source — to collect the package names those tools reliably hallucinate. The attacker registers those names on PyPI or npm, publishing packages containing malicious payloads: credential stealers, reverse shells, data exfiltration stubs, or supply chain staging code. Once the packages are live, the attacker does nothing further. The AI tool does the distribution work: any developer who asks that assistant for help with the same task will receive the same hallucinated package name in the generated code, and if the developer installs it without verification, the payload executes.

The intersection with dependency confusion adds a secondary attack vector that Andrew Nesbitt documented in December 2025 [4]. Organizations that maintain internal package registries may configure their package managers to fall back to public registries for names not found internally. A hallucinated name that does not resolve in the internal registry therefore falls through to the public registry, where an attacker has pre-registered a malicious version. This means slopsquatting is not limited to developers who manually copy-paste AI suggestions; automated build systems with public registry fallback may silently resolve and install hallucinated packages without any human interaction at the point of failure.

Why Standard Defenses Fall Short

Most traditional software composition analysis tools are oriented toward detecting known-bad packages — packages with recorded CVEs, packages with known malware signatures, packages whose maintainer accounts were previously compromised; behavioral analysis tools extend this coverage, though they are not yet universally deployed. A newly registered slopsquatting package, created the day after an attacker identified a stable hallucination, carries no CVE, no malware signature in threat intelligence feeds, and no suspicious maintainer history. It is, in every traditional SCA sense, "clean." Only its recency, low download count, sparse metadata, and the absence of a verifiable source repository distinguish it from a legitimate new library — none of which are checks that standard CI/CD pipeline configurations perform by default, though organizations with mature supply chain programs may have added them.

This gap is compounded by the autonomous execution patterns now common in AI-assisted development. Agentic coding tools — those that not only suggest code but execute commands, create files, and run package managers on behalf of the developer — may install suggested packages without surfacing the package name to the developer for review at all. In many agentic workflows, the human verification step that could catch a suspicious package name is bypassed unless the organization has

explicitly configured approval gates for package installation commands. The model hallucinates a name, the agent runs `pip install <hallucinated-name>`, and the malicious payload executes before the developer has seen any indication that the package did not previously exist.

Threat Model Positioning

Slopsquatting occupies a position in the threat model that is distinct from both prompt injection and model poisoning, though it interacts with both. It does not require the attacker to manipulate the model's weights or corrupt its training data. It does not require injecting adversarial instructions into the model's context at runtime. It exploits a stable behavioral property of the model — its tendency to confabulate plausible-sounding but non-existent package names — and then operates entirely within the legitimate, expected behavior of the developer's build toolchain. This makes it resilient to defenses like output filtering, jailbreak detection, and adversarial prompt mitigation — controls oriented toward the AI layer — while remaining invisible to CVE-based supply chain scanners.

Recommendations

Immediate Actions

Enterprises and development teams should verify all AI-suggested package names before installation, treating them as untrusted external input regardless of the AI tool's apparent confidence. The verification checklist should include confirming that the package exists on the intended registry, examining its creation date (newly created packages warrant heightened scrutiny), reviewing download counts and maintainer activity, and confirming the presence of a linked source repository with coherent commit history. For Python environments, PyPI's JSON API at `https://pypi.org/pypi/<packagename>/json` provides this metadata programmatically and can be integrated into pre-install hooks. For JavaScript environments, `npm view <packagename>` surfaces equivalent information.

Development teams should also integrate behavioral analysis tooling — such as Socket.dev — into their CI/CD pipelines as a complement to CVE-based scanners. Unlike traditional SCA, behavioral analysis flags packages that establish network connections, access the filesystem in unexpected ways, or use code obfuscation — patterns characteristic of malicious packages regardless of whether they have a recorded CVE history [5].

Short-Term Mitigations

Organizations should establish and enforce a dependency allow-list for production builds, accepting only packages that have been explicitly reviewed and approved. Any AI-suggested package that is not already on the allow-list should trigger a human review workflow before it reaches production. This is not a novel supply chain control, but it becomes newly important when AI tools can generate package suggestions that developers accept without the same skepticism they would apply to a cold internet search result.

Private registry configurations deserve review wherever public-registry fallback is enabled. Package managers should be configured to fail explicitly — rather than fall through to public registries — when a requested package name is absent from the internal registry. This eliminates the dependency confusion amplification vector that makes slopsquatting exploitable even in nominally air-gapped internal build environments.

Pinning dependencies using hashes rather than version ranges provides a complementary layer of defense. For dependencies already in the lockfile, hash pinning prevents silent substitution by a newly published malicious version — including a version pushed through a compromised maintainer account. This does not protect against the initial installation of a hallucinated package name, which is why allow-listing and pre-install verification are the primary controls for slopsquatting. Software Bill of Materials (SBOM) generation, aligned with existing CSA guidance on software transparency [6], supports visibility into what is actually installed and enables faster response if a previously clean package is later identified as malicious — provided the SBOM feeds into active monitoring and response workflows.

Strategic Considerations

The deeper issue that slopsquatting surfaces is the implicit trust model that has emerged around AI coding assistants. When a developer accepts a code suggestion from an AI tool, they frequently accept not just the logic but the package dependencies embedded in that code — dependencies the developer may have no independent knowledge of. The training of the AI model is not a guarantee of the continued existence or legitimacy of any package name it learned to associate with a given task.

Security teams should work with their engineering counterparts to establish organizational norms for AI tool output verification, distinguishing between code logic (which developers are generally accustomed to reviewing) and package dependencies (where verification practices have not yet caught up to the new risk surface). Enterprises deploying agentic development tools — those that execute commands autonomously — should configure those tools to require explicit human approval before running any package installation command, preventing the autonomous install path that bypasses developer review entirely.

On a longer time horizon, AI tool vendors have a role in addressing hallucination rates directly. The Spracklen et al. research demonstrated that mitigation strategies applied during inference can significantly reduce package hallucination rates without degrading overall code quality [1]. Organizations procuring AI coding tools should include hallucination rate and mitigation disclosure as evaluation criteria in vendor assessments, and should monitor vendor commitments to reducing hallucination frequency over successive model versions.

CSA Resource Alignment

The slopsquatting threat is directly relevant to several CSA frameworks and research initiatives. The MAESTRO framework for agentic AI threat modeling provides the most precise mapping. Within MAESTRO's seven-layer architecture, slopsquatting operates at Layer 3 (Agent Frameworks), where an AI coding assistant constructs and executes code that includes hallucinated dependencies, and at Layer 4 (Deployment and Infrastructure), where build pipelines instantiate those dependencies into running environments [7]. MAESTRO's cross-layer threat identification methodology is particularly applicable here, as the attack exploits a gap between the AI reasoning layer and the infrastructure execution layer — a gap that neither layer's controls are individually designed to close.

The AI Controls Matrix (AICM) v1.0 addresses AI supply chain security as a distinct domain, and the supply chain integrity controls within AICM are directly applicable to the pipeline hardening steps described in the recommendations above. Organizations implementing AICM controls should extend their scope to cover AI-generated build artifacts and AI-suggested dependencies, not merely the AI models themselves [8].

CSA's Software Transparency and Securing the Digital Supply Chain guidance establishes the SBOM and verification practices that form the strategic foundation of a slopsquatting defense. The alignment between existing supply chain security controls and the mitigations described here is high: slopsquatting does not require a wholly new control domain, but it does require organizations to apply existing controls to a new class of inputs — AI-generated package references — that their current security programs may not yet treat as in-scope.

References

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

[2] Socket Security. "The Rise of Slopsquatting: How AI Hallucinations Are Fueling a New Class of Supply Chain Attacks." Socket.dev, April 8, 2025.

[3] Infosecurity Magazine. "AI Hallucinations Create 'Slopsquatting' Supply Chain Threat." Infosecurity Magazine, 2025.

[4] Nesbitt, Andrew. "Slopsquatting meets Dependency Confusion." nesbitt.io, December 10, 2025.

[5] Cloudsmith. "Typosquatting & Slopsquatting: Detecting and Defending Against Malicious Packages." Cloudsmith, 2025.

[6] Cloud Security Alliance. "Software Transparency: Securing the Digital Supply Chain." CSA, October 2022.

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

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

