AI-Induced Supply-Chain Compromise: A Systematic Review of Package Hallucinations and Slopsquatting Attacks

Wadhah Al-Zofi

alzofiwadhah212814@gmail.com

Changchun University of Science and Technology

Research Article

Keywords: LLM package hallucination, slopsquatting,software supply-chain security, typosquatting, dependency confusion, open-source registry policies, PRISMA systematic review

Posted Date: November 10th, 2025

DOI: https://doi.org/10.21203/rs.3.rs-8007192/v1

Additional Declarations: No competing interests reported.

AI-Induced Supply-Chain Compromise: A Systematic Review of Package Hallucinations and Slopsquatting Attacks

Wadhah Al-Zofi

Received: date / Accepted: date

Abstract The adoption of large language models (LLMs) and AI-assisted programming has accelerated software production, but it has also created a novel supply-chain vulnerability: package hallucination. When an LLM generates code, it may recommend nonexistent third-party packages that "sound" plausible. Adversaries can register these phantom names in public registries, thereby poisoning the open-source software supply chain. This attack pattern, termed *slopsquatting*, combines aspects of typosquatting and dependency confusion but is triggered by AI hallucinations rather than human error. We systematically review this emerging threat. Following PRISMA-2020 guidelines, we searched IEEE Xplore, ACM Digital Library, SpringerLink, Scopus and arXiv for publications (2018–Oct 2025) on package hallucination, typosquatting, dependency confusion, supply-chain compromise and registry policies. Twenty-one peer-reviewed papers and seven credible industry reports met the inclusion criteria. We synthesize definitions, threat models and observed incidents; report empirical evidence of hallucination prevalence across LLMs (e.g., GPT-series models hallucinate 5.2 % of packages versus 21.7 % for open-source models); and map defenses at IDE, registry, CI/CD and runtime layers. We compare slopsquatting with typosquatting and dependency confusion using a new taxonomy and highlight gaps in current safeguards. Official policies from npm, PyPI, Maven Central, RubyGems, NuGet and CRAN show varying levels of name reservation, deletion and immutability. Our review exposes an urgent need for package-existence validation within AI coding tools, stricter registry name policies and standardized provenance attestations.

Keywords LLM package hallucination · slopsquatting · software supply-chain security · typosquatting · dependency confusion · open-source registry policies · PRISMA systematic review

Wadhah Al-Zofi
School of Electronic and Information Engineering, Changchun University of Science and Technology, WeiXing Road, Changchun, 130022, Jilin, China
E-mail: alzofiwadhah212814@gmail.com

1 Introduction

Generative AI is transforming software engineering. With the advent of LLM-powered assistants such as GitHub Copilot, OpenAI Codex and Google Gemini, developers increasingly rely on auto-complete and code synthesis. One estimate suggests that up to 97 % of developers now incorporate generative AI into their workflow and that about 30 % of code is AI-generated [1]. However, these tools sometimes generate code that references packages which do not exist. *Package hallucination* occurs when an LLM suggests an import or dependency on a library that has never been published. If an attacker monitors these hallucinations and registers the suggested name in a public package registry (e.g., npm, PyPI, Maven Central), subsequent installations may fetch a malicious package. Trend Micro coined the term *slopsquatting* to describe this attack: the malicious registration of hallucinated packages to exploit AI-induced "sloppy" code suggestions [2]. Slopsquatting extends the well-known attacks of typosquatting, publishing packages with names similar to popular ones, and dependency confusion, publishing malicious packages that supersede private dependencies, but introduces unique features. Unlike typosquatting, victims may have no opportunity to notice a misspelling; the hallucination appears plausible and is presented by a trusted AI assistant. Unlike depen-

dency confusion, the package name is not necessarily used internally but is created by the LLM itself.

This systematic review aims to consolidate current knowledge on AI-induced package hallucination and slopsquatting. We begin by differentiating the new threat from established supply-chain attacks (Section 2), define a threat model and attack chain (Section 3) and explain our systematic review methodology (Section 4). Section 5 presents empirical evidence on hallucination prevalence and documented incidents. Section 6 offers a comparative taxonomy contrasting slopsquatting with typosquatting and dependency confusion. Section 7 maps existing defenses and identifies gaps, while Section 8 compares registry policies relevant to name squatting. We discuss research directions in Section 9, note limitations in Section 10 and conclude in Section 11.

2 Background and Terminology

2.1 AI-Assisted Coding and Package Hallucinations

AI-assisted coding systems typically operate in an integrated development environment (IDE) or CLI wrapper around the LLM. The user enters a prompt or partial code; the model produces a completion that may include API calls and package imports. *Package hallucination* occurs when the completion references a package that does not exist in the target registry. The Importing Phantoms study observed that LLMs suggested nonexistent packages such as `securehashlib` and that an attacker could register these names to exfiltrate secrets [3]. In *We Have a Package for You*!, a large-scale evaluation of 16 LLMs, researchers generated 2.23 million code samples across Python and JavaScript and found that 440,445 (5.2 %) of packages referenced by commercial models did not exist, whereas 21.7 % of packages referenced by open-source models were hallucinated [4]. Package hallucinations therefore present a substantial attack surface.

2.2 Typosquatting and Dependency Confusion

Typosquatting is an established supply-chain attack whereby an adversary publishes a package with a name that closely resembles a legitimate package to trick users who mistype or misread the name. The *TypoSmart* study lists several strategies, including single-character edits, prefix or suffix augmentation, and homophonic similarity [5]. Typosquatting relies on human error; vigilance and spelling checks can mitigate it. *Dependency confusion* (also called *namespace confusion*) involves attackers publishing a package to a public registry using the same name as an internal package in a victim's build system. Because some build tools prioritize public packages with higher version numbers, the malicious package is installed instead of the intended private dependency. Birsan's 2021 attack demonstrated this weakness and affected major organizations, highlighting that package names may leak via manifests or build file [6].

2.3 From Hallucination to Slopsquatting

Slopsquatting resembles typosquatting in that it exploits the lexical similarity of package names, yet it differs in origin and detection difficulty. In slopsquatting, the attacker does not approximate an existing popular name but registers a name invented by an LLM. Developers who use AI assistants may not realize the package is hallucinated and will install it intentionally, believing the AI recommendation. This differs from dependency confusion because the hallucinated package name is not used internally; rather, it arises spontaneously from an AI model. Table 1 summarizes terminology for package name attacks.

3 Threat Model and Attack Chain

We model the slopsquatting adversary as an attacker who monitors LLM outputs, either directly via model queries or indirectly via forums and code repositories, to discover hallucinated package names. The attacker has the capability to quickly register packages in public registries and to craft malicious code (e.g., backdoors, data exfiltration). We assume the victim is a developer using an AI coding assistant in an IDE or CLI. Fig. 1 depicts the attack chain. The sequence begins when a developer issues a prompt; the LLM suggests code containing a phantom `import` or dependency. If the developer attempts to install the package, the package manager fails because the package does not exist. The attacker then registers the hallucinated name in the public registry. Subsequently, any developer following the AI suggestion receives the attacker's package, and the malicious code executes during installation or runtime. Trust boundaries exist at four interception points: (1) the AI-assisted IDE where suggestions can be validated; (2) the registry where naming policies could prevent registration of hallucinated names; (3) continuous integration/continuous deployment (CI/CD) pipelines where dependency provenance can be enforced; and (4) runtime environments where behavioral monitoring may detect anomalous calls.

Table 1 Terminology crosswalk for supply-chain package-name attacks

Attack type	Trigger / name source	Primary preconditions and victims	Examples / remarks
Typosquatting	Human mis-typing or mis-reading	Popular package names; relies on user error; detection via lexical similarity	`requestss` vs. `requests`
Dependency confusion	Build tools resolving private vs. public packages; attacker publishes a higher-version package with same name	Organizations using private registries; rely on version resolution; exploit naming leakage in manifests	Birsan's 2021 attack
Package hallucination	AI model generates a nonexistent package name	LLM-assisted developers; packages never existed prior; attack window from suggestion to fix	`securehashlib` in *Importing Phantoms*
Slopsquatting	Attacker registers hallucinated name and uploads malicious code	LLM users accept AI suggestions; public registries with lax name reservation; similar to typosquatting but uses AI-generated names	Claude or Codex suggestions exploited

Fig. 1 illustrates this attack chain and highlights where detection or mitigation measures can be applied (IDE validation, registry policies, CI/CD provenance enforcement and runtime monitoring).

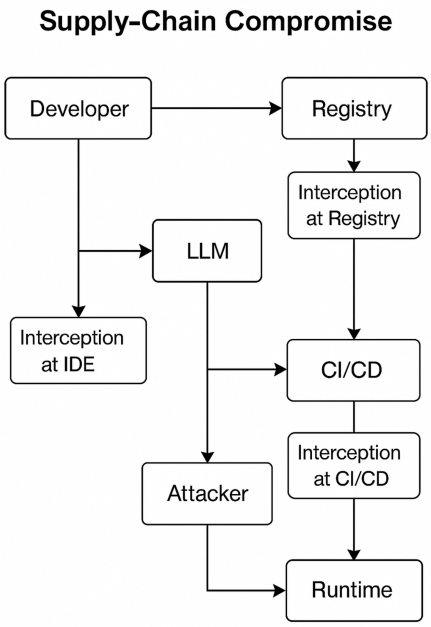

Fig. 1 Supply-Chain Compromise

4 Methods: Systematic Review Using PRISMA

4.1 Search Strategy and Selection Criteria

A systematic literature search was performed in September–October 2025. We queried IEEE Xplore, ACM Digital Library, SpringerLink, Scopus and arXiv with keywords such as **"package hallucination", "slopsquatting", "AI supply-chain attack", "typosquatting"**, and **"dependency confusion"**. Industry reports were considered when peer-reviewed literature was absent but only when published by credible organizations (e.g., Trend Micro, Kaspersky, Sonatype). The timespan was 2018 to October 2025. Inclusion criteria were: (i) studies discussing package hallucination, typosquatting, dependency confusion or related supply-chain attacks; (ii) papers presenting empirical data or documented incidents; (iii) English language. Exclusion criteria were: (i) purely anecdotal blog posts without evidence; (ii) simulation-only studies not pertaining to real package registries.

4.2 Screening and Data Extraction

Screening followed the PRISMA 2020 flow. After removing duplicates, titles and abstracts were assessed for relevance. Full-text reviews were conducted for 31 articles. Twenty-one academic papers and seven industry reports met the inclusion criteria. Data extracted included definitions, threat models, prevalence statistics, experimental methodologies, defenses and policy recommendations. Risk of bias was considered by assessing study design, sample sizes, dataset representativeness and transparency of evaluation protocols.

4.3 PRISMA Flow Diagram

Our search identified 245 records through database searching and 15 additional records from other sources. After removing duplicates, 210 records were screened by title and abstract. Following exclusion of 145 irrelevant

records, 65 full-text articles were assessed for eligibility. Thirty-seven were excluded due to lack of empirical data, leaving 21 academic papers and seven industry reports in the final synthesis. Fig. 2 shows the PRISMA 2020 flow diagram of study selection.

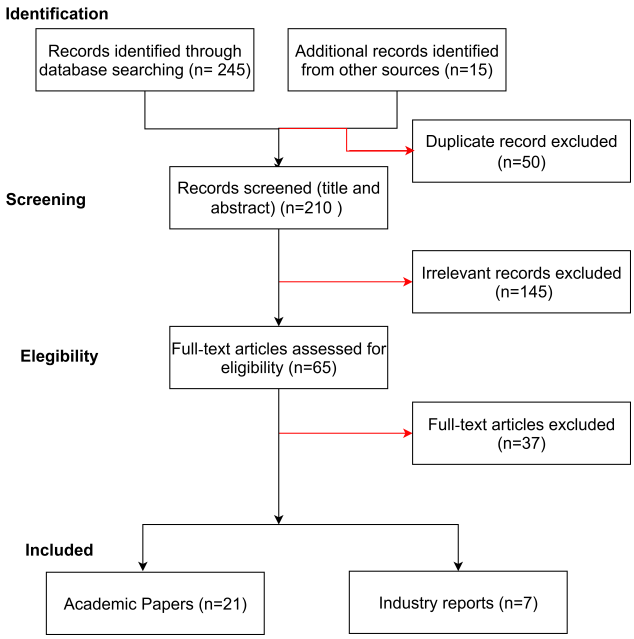

Fig. 2 PRISMA 2020 flow diagram of study selection

5 Results

5.1 Empirical Evidence of Package Hallucinations

The *We Have a Package for You*! study provides the most comprehensive evaluation to date [4]. It generated 2.23 million code samples across Python and JavaScript using 16 LLMs, both commercial (OpenAI GPT-3.5/4, Anthropic Claude 2, Google Gemini) and open-source (LLaMA, StarCoder, DeepSeek-Coder). Researchers found that commercial models hallucinated packages in 5.2 % of generated code, whereas open-source models hallucinated 21.7 %. GPT-4 Turbo had the lowest hallucination rate at 3.59 %. The study also discovered over 205,474 unique hallucinated package names. The *Importing Phantoms* paper analyzed hallucinations across multiple languages and noted that hallucination rate inversely correlated with model performance on coding benchmarks; high-performing models produced fewer hallucinations [3]. Another work, *HFuzzer*, introduced phrase-based fuzzing to trigger hallucinations and observed that certain prompt patterns (e.g., "create a secure connection using library X") reliably induced phantom names, such as `hyper-h2` [7]. UTSA's study, summarized by their university press, confirmed that GPT-series models were four times less likely to hallucinate than open-source models and that Python code exhibited fewer hallucinations than JavaScript [1].

Table 2 compiles prevalence data from peer-reviewed evaluations and credible reports published in 2024–2025. Only model–dataset pairs with explicit methodologies and sample sizes were included.

5.2 Case Studies and Documented Attacks

Evidence of real-world slopsquatting attacks remains limited but growing. Trend Micro's 2024 red-team study simulated slopsquatting by using a model that hallucinated `securehashlib`; the researchers registered the name on PyPI and found that developers using AI suggestions installed their malicious package [2]. They also reported that more capable agents (e.g., Claude Code CLI with live package search) reduced hallucination but did not eliminate it. UTSA's paper documented several hallucinated names that were subsequently registered by unknown actors before the responsible disclosure, illustrating the rapid exploitation window [1]. While no large-scale breach has yet been attributed to slopsquatting, the similarity to dependency confusion attacks suggests that exploitation is plausible. For example, Birsan's 2021 dependency confusion campaign successfully infiltrated Apple, Microsoft and PayPal by registering internal package names, demonstrating how quickly attackers can weaponize naming vulnerabilities [6].

5.3 Comparative Taxonomy of Package-Name Attacks

Fig. 3 visualizes a taxonomy comparing slopsquatting, typosquatting and dependency confusion along key dimensions. The matrix contrasts the trigger, name source, preconditions, detection difficulty and mitigation points. Slopsquatting blends aspects of both traditional attacks but introduces the unique challenge of AI-generated names. We discuss these dimensions below.

1. **Trigger and source of name:** Typosquatting is triggered by human errors; the name is a variant of a popular package. Dependency confusion arises when the same package name exists in both private and public registries. Slopsquatting is triggered by AI hallucination; the name may have no prior existence.

2. **Preconditions and trust boundary:** Typosquatting requires only that users search and install packages; preconditions are minimal. Dependency confusion requires that build tools resolve ambiguous

Table 2 LLM package-hallucination prevalence by model and dataset (2018–Oct 2025

Study / year	Model(s) and dataset(s)	Hallucination metric and result	Notes
We Have a Package for You! (USENIX Security 2025)	16 LLMs including GPT-4 Turbo, GPT-3.5, Claude 2, Gemini, StarCoder, LLaMA 2. Python and JavaScript prompts from StackOverflow and LLM-generated tasks.	Commercial models: 5.2% hallucinated packages; open-source models: 21.7%; GPT-4 Turbo lowest at **3.59%** for Python and **4.00%** for JavaScript. CodeLlama 34B exhibited **21.15%** hallucination rate in Python and **34.57%** in JavaScript. CodeLlama 7B showed **26.12%** (Python) and **35.71%** (JavaScript).	2.23 million code samples; 205k unique hallucinated names. Highlights wide variance across models and languages.
Importing Phantoms (arXiv 2025)	GPT-3.5, GPT-4, Claude 2, LLaMA; prompts across Python, Go, Rust.	Qualitative analysis; hallucination rate inversely correlated with model performance on code benchmarks; repeated phantom names such as `securehashlib`.	Provides examples but not full prevalence tables.
HFuzzer (arXiv 2025)	GPT-J, GPT-Neo; fuzzed prompts using phrase-based triggering.	Identified approximately **11.6%** hallucination rate when prompts contained security-related phrases such as "secure connection" or "network handshake"; discovered phantom names like `hyper-h2`.	Demonstrates that targeted prompts can induce hallucinations in open-source models.
UTSA / USENIX press release	Same dataset as *We Have a Package for You!*	Reports that 440,445 of 2.23 million samples (19.7%) referenced hallucinated packages; GPT-series models were four times less likely to hallucinate than open-source models; Python code produced fewer hallucinations than JavaScript.	Public summary of the USENIX 2025 study.
Trend Micro red-team study (2024)	Claude Code CLI (with live registry lookup) and Codex CLI (with automated testing) [2].	Validation features reduced hallucinations but did not eliminate them; slopsquatting demonstration using a hallucinated name that was quickly registered on PyPI.	Industry research illustrating real exploitation potential rather than prevalence metrics.

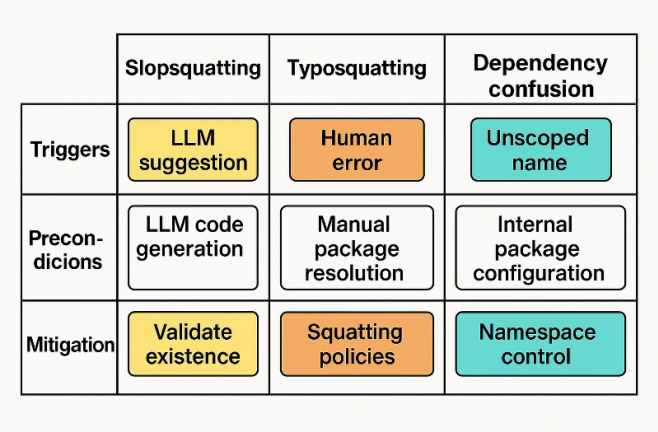

Fig. 3 Taxonomy comparing slopsquatting, typosquatting and dependency confusion

names across registries and may be mitigated by scoping or explicit package sources. Slopsquatting requires a developer to rely on an AI suggestion and a registry to accept new names without validation.

3. **Detection difficulty:** Typosquatting can be detected via lexical similarity and popularity metrics (e.g., SpellBound flags suspicious packages). Dependency confusion detection relies on verifying package provenance and version pinning. Slopsquatting detection is challenging because there is no lexical baseline; the package name is unique yet plausible. Models may hallucinate names such as `dataframe-utils` that sound legitimate.

4. **Mitigation leverage points:** For typosquatting, registries can block confusable names and implement fast take-down procedures. Dependency confusion is mitigated by namespacing (scoped packages) and private registries. Slopsquatting requires new measures: AI systems should validate package existence before suggesting imports; registries should allow developers to reserve names, and CI/CD should enforce provenance attestation.

6 Defenses and Remaining Gaps

Defending against slopsquatting requires a holistic approach across the software supply chain. Table 3 summarizes defense techniques mapped to the attack phases (IDE/AI assistant, registry, CI/CD, runtime) and highlights coverage versus gaps.

6.1 IDE and LLM Layer

Package-existence validation LLM providers can integrate registry queries into their models: before suggesting an import, the system checks whether the package exists [8]. Trend Micro's red-team tool implemented live search to ensure that packages exist before recommending them; this reduced hallucination incidence. OpenAI has begun to use package allowlists for certain languages, refusing to suggest nonexistent names [9]. However, such measures may limit the creativity of code generation and require frequent updates.

Provenance prompts and contextual warnings AI assistants could warn users when a recommended package is unknown or unverified. The model might say "I am suggesting foo but cannot confirm its existence in the registry." This transparency fosters critical thinking but depends on user attention.

Refusal policies In high-risk contexts, AI systems could be configured to refuse to invent new package names unless explicitly instructed. A refusal policy would force the user to specify dependencies or to confirm manual installation. Implementation details remain open research; prompts might instruct the model to restrict package recommendations to known lists.

6.2 Registry Policies and Controls

Package registries play a crucial role in preventing slopsquatting by controlling name registration, verifying ownership, and facilitating takedowns. Official policies vary:

npm: npm detects typosquat attacks and blocks publication of packages whose names closely resemble popular packages [10]. It encourages the use of scoped packages to avoid dependency confusion, although the registry cannot automatically detect such attacks. Package names must be unique, descriptive and non-offensive and should not be spelled similarly to existing packages. npm's unpublish policy allows a package to be removed within 72 hours if no other packages depend on it or later under strict conditions. After unpublishing, the same name cannot be reused immediately. Squatting is considered if a package has no genuine function.

PyPI: The Python Package Index implemented PEP 541 [11]. Name retention policies state that project names are persistent; they may be transferred only if a project is abandoned (no releases for 12 months, owner unreachable). Projects are removed only if they are invalid, malware, spam, illegal content, copyright/trademark violations, or name squatting (empty packages). The maintainers may pre-emptively declare certain names unavailable for security reasons. Abandoned names can be reused by another maintainer if strict criteria are met.

Maven Central: Sonatype enforces namespaces tied to domain ownership: each groupId must reflect a reverse domain name controlled by the publisher [12]. Publishers must prove domain ownership via DNS records. This namespacing reduces typosquatting and slopsquatting because an attacker cannot easily register under someone else's domain. Maven Central enforces immutability; once an artifact is published, it cannot be modified or removed. Instead, publishers must release a new version. The Central repository thus discourages deletion of malicious packages but offers no explicit mechanism to block hallucinated names outside reserved domains.

RubyGems: RubyGems' acceptable use policy prohibits content that exists only to reserve a name (name squatting) [13]. The registry operates on a first-come first-serve basis but requires that gems be functionally compatible with the build tools. Gem owners cannot delete their packages unilaterally; the RubyGems team may delete gems that violate policies or pose security risks. Name trading is prohibited. Ownership transfers require owner agreement; if owners are unreachable, the team mediates transfers.

NuGet: NuGet does not support permanent deletion of packages to avoid breaking build workflows [14]. Instead, package owners may "unlist" packages so they are hidden from search results but still downloadable by exact version. Exceptions for deletion include copyright infringement and harmful content. The policy prohibits packages used for squatting on identifiers; packages with no productive content are removed. NuGet offers ID prefix reservation: package ID prefixes can be reserved to prevent others from using them; packages matching the reserved prefix are rejected unless submitted by the owner. The prefix reservation includes visual indicators and may delegate subprefixes to other owners.

CRAN: The Comprehensive R Archive Network requires that packages be named in a way that does not conflict with any current or past CRAN or Bioconductor package [15]. Package names are persistent and generally cannot be changed; maintainers relinquish the right to use the name upon submission. CRAN may remove or modify packages without notice but usually provides notification. If maintainers are unreachable or a package is abandoned, the CRAN team may orphan it and allow another maintainer to take over.

These policies reveal that only some registries have explicit anti-squatting measures. npm and PyPI address typosquatting and name squatting; RubyGems prohibits packages that exist solely to reserve names. NuGet's prefix reservation reduces namespace confusion. Maven's domain-based groupIds implicitly mitigate name attacks. CRAN's unique names and orphaning reduce conflicts but rely on manual governance. Table 4 summarizes the policy landscape.

6.3 CI/CD and Build Security

CI/CD pipelines serve as another line of defense. **Dependency pinning** with exact version numbers prevents automatic upgrades to malicious packages and reduces exposure to dependency confusion and slopsquatting [16]. **Lockfiles** (e.g., `package-lock.json`, `Pipfile.lock`) capture dependency graphs at specific versions. **Provenance attestations** and supply-chain levels for software artifacts (SLSA) require that each build step produce verifiable metadata (e.g., using Sigstore) [17]. SLSA level 4 recommends hermetic builds and two-person review, which could detect unauthorized dependencies. Tools like in-toto can ensure that only vetted dependencies appear in the SBOM [18]. However, these measures cannot detect hallucinated names unless additional checks confirm the package's existence in the registry at build time.

6.4 Runtime Protections

Even if a malicious package is installed, runtime controls can mitigate damage. Sandboxing (e.g., container isolation, seccomp) and network access restrictions prevent unauthorized connections [19]. eBPF-based monitoring can observe system calls and block anomalous behavior [20]. Behavioral allowlists and anomaly detection can flag packages that attempt network exfiltration or privilege escalation. Nonetheless, runtime measures are reactive and cannot substitute for upstream validation.

6.5 Coverage and Gaps

Table 3 maps defense techniques to attack phases. A key gap is the lack of proactive package existence validation in AI tools; most LLMs do not integrate registry queries. Registries generally lack reserved-name systems for hallucinated names, and policies often focus on typosquatting rather than AI-generated names. CI/CD pipelines enforce provenance but do not verify that dependencies exist at the time of suggestion. Runtime protections mitigate consequences but not infiltration.

7 Registry Policy Landscape

Table 4 compares official policies of major registries relevant to slopsquatting risk. Only information from official documentation is included.

8 Discussion

8.1 Implications for AI-Assisted Development

LLM adoption is outpacing the security measures necessary to contain new risks. Package hallucination is not a theoretical curiosity; evaluations reveal that thousands of unique phantom names are generated, and 5–22 % of package references in AI-generated code are hallucinations [21]. Given that developers increasingly copy and paste AI-generated code without scrutiny, the attack surface is large. The slopsquatting attack chain is simple: an attacker monitors widely used LLMs, registers a hallucinated package, and waits for victims to install it [22]. Because hallucinated names may recur across prompts and users, attackers can prioritize names with high frequency and plausible semantics.

The threat is amplified by the openness of package registries. Most registries operate on a first-come first-serve basis, with limited pre-screening. npm blocks typosquatting but cannot detect dependency confusion or slopsquatting. PyPI requires persistence and has a formal dispute process but does not pre-validate new names. RubyGems forbids squatting but relies on manual enforcement. Maven Central's domain-based namespaces reduce risk but do not cover unscoped names outside controlled domains. NuGet's prefix reservation is the most proactive approach; it prevents untrusted parties from using reserved prefixes.

8.2 Interaction of LLM UX, Developer Behavior and Registry Governance

Several factors converge to enable slopsquatting. First, LLMs aim to be helpful and may fabricate plausible names instead of admitting uncertainty. Without explicit prompt engineering or refusal policies, they will continue to hallucinate. Second, developers often trust AI outputs and may not cross-check dependencies, especially when under time pressure. Third, registries allow instantaneous registration of new names with minimal barriers. The combination of these behaviors yields a narrow but exploitable window between hallucination and malicious registration. Strengthening any part of the chain, through AI validation, developer awareness

Table 3 Defense techniques mapped to slopsquatting attack phases

Attack phase	Defense technique	Coverage	Gaps
IDE / LLM suggestion	Live registry lookup for package existence; allowlists; refusal policies; explanatory prompts	Reduces hallucination; fosters user awareness	May limit creative suggestions; needs frequent updates; not yet widely deployed
Registry	Name reservation (e.g., npm typosquat detection, NuGet ID prefix reservation); domain-based namespaces (Maven); anti-squatting policies (RubyGems, PyPI)	Blocks typosquatting; ensures names map to domain owners; removes empty or malicious packages	No universal mechanism for hallucinated names; slow takedowns; limited enforcement across ecosystems
CI/CD	Version pinning; lockfiles; SLSA provenance; Sigstore signatures; in-toto attestations	Prevents dependency confusion and supply-chain injection; ensures reproducibility	Does not verify existence of recommended packages; may not detect malicious code within legitimate names
Runtime	Sandboxing, container isolation; syscall filtering (seccomp); eBPF monitoring; anomaly detection	Limits impact of malware; detects suspicious behavior	Reactive; may generate false positives; does not prevent installation

or registry policies, would shrink the attacker's opportunity.

8.3 Responsible Disclosure and Standardization

Researchers investigating package hallucinations have responsibly disclosed phantom names to registry maintainers to prevent exploitation. However, there is no standardized process akin to CVEs for hallucinated names. The security community should consider establishing a reporting mechanism where researchers can reserve or block hallucinated names as they are discovered [23]. Standardization efforts such as the OpenSSF's Supply-Chain Levels for Software Artifacts (SLSA), in-toto attestations and software bill of materials (SBOM) formats (SPDX, CycloneDX) provide frameworks for provenance and dependency transparency [17]. Extending these frameworks to include LLM-generated code and hallucinated dependencies is a promising avenue.

8.4 Research Gaps

Despite growing interest, many questions remain. We still lack longitudinal studies on whether slopsquatting has been exploited at scale [24]. Empirical measurements of attacker behavior, such as monitoring newly registered packages after major LLM releases, would clarify the prevalence of malicious registrations. More research is needed to design AI models that avoid hallucination without sacrificing usability [25]. Prompt engineering, retrieval-augmented generation and reinforcement learning with human feedback could reduce hallucination rates. Finally, comparative policy analysis across registries should evaluate the effectiveness of anti-squatting measures and inform best practices.

9 Limitations

This review has several limitations. First, slopsquatting is a nascent topic with limited empirical data; most observations are from controlled studies or simulations. Second, our search may have missed unpublished attacks or proprietary investigations. Third, the prevalence statistics are based on evaluations of specific models and may not generalize across all LLMs or prompts. Fourth, registry policies evolve; the policies summarized here reflect documentation accessed in October 2025 and may change thereafter. Finally, our PRISMA selection covers English-language publications; relevant studies in other languages were excluded.

10 Conclusion

AI-induced package hallucination introduces a new vector for software supply-chain compromise. Slopsquatting, the malicious registration of hallucinated packages, blends the opportunism of typosquatting with the stealth of dependency confusion. Our systematic review synthesizes definitions, prevalence evidence and defenses. We find that commercial LLMs hallucinate packages in approximately 5 % of code completions and that open-source models may hallucinate over one-fifth of referenced packages. The attack window is exacerbated by permissive registry policies and developer trust in AI suggestions. Defenses exist at multiple layers, AI assistants can validate package existence; registries can implement name reservation and squatting checks; CI/CD pipelines can enforce provenance; and runtime monitors can detect malicious behavior, but coverage remains inconsistent. We urge AI vendors to integrate package lookup and refusal policies, registries to adopt stronger

Table 4 Registry policy comparison relevant to slopsquatting risk (official documents only)

Registry	Name reservation / squatting policy	Removal / immutability	Verification / attestation	Rapid response / takedown
npm	Detects and blocks typosquat packages; encourages scoped packages for private names; package names must be unique, descriptive and not similar to existing names; squatting defined as packages with no genuine function	Registry data immutable; unpublish allowed within 72 hours if no dependents; later unpublish permitted under strict conditions; after unpublishing, name cannot be reused immediately	Supports provenance statements and ECDSA signatures; 2FA mandatory for high-impact packages	Trust and Safety team scans packages for malicious content and removes reported packages
PyPI	Name retention policy (PEP 541): names are persistent; abandoned projects may be transferred; invalid projects (malware, spam, name squatting) removed; maintainers may pre-emptively declare certain names unavailable for security reasons	Projects rarely removed; deletion reserved for invalid content; historical artifacts preserved	Provides digital attestations (trusted publishers) and encourages GPG signatures	Maintainers reachable via email; Python Software Foundation board resolves disputes
Maven Central	Namespaces tied to reverse domain names; groupId requires proof of domain ownership; names cannot be registered outside owned namespace; reduces typosquatting and slopsquatting risk	Immutability: once an artifact is published it cannot be modified or removed; new releases required for bug fixes	Requires GPG signatures and checksums; Sonatype safety rating assesses packages; ID ownership verified via DNS	Sonatype monitors for malicious components and may remove ones that violate terms; specifics not detailed
RubyGems	First-come first-serve but prohibits name squatting; packages existing only to reserve a name are disallowed; trading names is prohibited	Registry aims for immutability; gem owners cannot delete packages; RubyGems team may delete gems for policy violations or security reasons	Encourages signatures; details limited	RubyGems team mediates ownership transfers when owners unreachable; can delete gems without notice if security concern
NuGet	Package ID prefix reservation prevents others from publishing under reserved prefixes; packages matching reserved prefix are rejected unless submitted by owner; prohibits packages used to squat on identifiers or with zero productive content	Permanent deletion not supported; packages can be unlisted to hide them; exceptions for copyright infringement or harmful content allow deletion	Visual indicator shows packages from reserved prefixes; signature verification available	NuGet team removes prohibited packages; owners can report abuse
CRAN	Package names must not conflict with current or past CRAN/Bioconductor packages; maintainers may orphan packages and allow takeover if owners unreachable	Names are persistent; packages generally cannot be renamed or removed; CRAN may remove or modify packages without notice	Requires maintainers to provide contact information and license; no formal attestation mechanism	CRAN volunteers may remove packages; contact via CRAN-submissions email

anti-squatting measures and supply-chain frameworks to include LLM-generated code in provenance attestations. Future research should empirically measure slopsquatting in the wild, design hallucination-resilient AI systems and harmonize registry policies to safeguard the emerging AI-assisted software supply chain.

Author contributions W.A: Is the sole author of this manuscript. Wrote the main manuscript text, conducted the systematic literature review, prepared Tables & Figures, reviewed the research methodology and analysis sections, and approved the final manuscript.

Funding This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Data availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The author declares no competing interests.

References

1. Casta neda, A.: UTSA researchers investigate AI threats in software development. UT San Antonio Today. https://www.utsa.edu/today/2025/04/story/utsa-researchers-investigate-AI-threats.html (2025). Accessed 31 Oct 2025.
2. Park, S.: Slopsquatting: When AI Agents Hallucinate Malicious Packages. Trend Micro Research https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/slopsquatting-when-ai-agents-hallucinate-malicious-packages (2025). Accessed 31 Oct 2025.
3. Krishna, A., Galinkin, E., Derczynski, L., Martin, J.: Importing Phantoms: Measuring LLM Package Hallucination Vulnerabilities. arXiv *2501.19012*. https://doi.org/10.48550/arXiv.2501.19012 (2025).
4. Spracklen, J., Wijewickrama, R., Nazmus Sakib, A.H.M., Maiti, A., Viswanath, B., Jadliwala, M.: We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs. In: USENIX Security Symposium (USENIX Security 2025). USENIX Association. https://www.usenix.org/conference/usenixsecurity25/presentation/spracklen (2025).
5. Neupane, S., Holmes, G., Wyss, E., Davidson, D., De Carli, L.: Beyond Typosquatting: An In-depth Look at Package Confusion. In: 32nd USENIX Security Symposium (USENIX Security 2023). USENIX Association. https://www.usenix.org/system/files/usenixsecurity23-neupane.pdf (2023).
6. Birsan, A.: Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies. Medium. https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610 (2021). Accessed 31 Oct 2025.
7. Zhao, Y., Wu, M., Hu, X., Xia, X.: HFuzzer: Testing Large Language Models for Package Hallucinations via Phrase-based Fuzzing. arXiv *2509.23835*. https://arxiv.org/abs/2509.23835 (2025).
8. Arghire, I.: AI Hallucinations Create a New Software Supply Chain Threat. SecurityWeek. https://www.securityweek.com/ai-hallucinations-create-a-new-software-supply-chain-threat/ (2025). Accessed 31 Oct 2025.
9. Dunn, J.E.: Large language models hallucinating non-existent developer packages could fuel supply-chain attacks. InfoWorld. https://www.infoworld.com/article/3542884/large-language-models-hallucinating-non-existent-developer-packages-could-fuel-supply-chain-attacks.html (2024). Accessed 31 Oct 2025.
10. npm, Inc.: npm Policies. https://docs.npmjs.com/policies (2025). Accessed 31 Oct 2025.
11. Langa, Ł.: PEP 541—Package Index Name Retention. Python Enhancement Proposals. https://peps.python.org/pep-0541/ (2018). Accessed 31 Oct 2025.
12. Sonatype: Maven Central Repository Policies (Central Repository Terms/Requirements). https://central.sonatype.org/terms.html (2025). Accessed 31 Oct 2025.
13. RubyGems.org: Policies for RubyGems.org. https://rubygems.org/policies (2025). Accessed 31 Oct 2025.
14. Microsoft: NuGet.org policies—Deleting packages. Microsoft Learn. https://learn.microsoft.com/en-us/nuget/nuget-org/policies/deleting-packages (2025). Accessed 31 Oct 2025.
15. The Comprehensive R Archive Network (CRAN): CRAN Repository Policy. https://cran.r-project.org/web/packages/policies.html (2025). Accessed 31 Oct 2025.
16. FOSSA, Inc.: Understanding and Preventing Dependency Confusion Attacks. https://fossa.com/blog/dependency-confusion-understanding-preventing-attacks/ (2022). Accessed 31 Oct 2025.
17. OpenSSF: Supply-chain Levels for Software Artifacts (SLSA). https://slsa.dev/ (2025). Accessed 31 Oct 2025.
18. in-toto Project: in-toto—Providing farm-to-table guarantees for software supply chains. https://in-toto.io/ (2025). Accessed 31 Oct 2025.
19. Aqua Security: What Is a Dependency Confusion Attack? https://www.aquasec.com/cloud-native-academy/supply-chain-security/dependency-confusion/ (2024). Accessed 31 Oct 2025.
20. ActiveState: How Open Source Typosquatting Attacks Work. https://www.activestate.com/resources/quick-reads/how-open-source-typosquatting-attacks-work/ (2025). Accessed 31 Oct 2025.
21. AI Hallucinations Create "Slopsquatting" Supply Chain Threat. Infosecurity Magazine. https://www.infosecurity-magazine.com/news/ai-hallucinations-slopsquatting/ (2025). Accessed 31 Oct 2025.
22. Contrast Security: Slopsquatting—How Attackers Exploit AI-Generated Package Names. https://www.contrastsecurity.com/security-influencers/slopsquatting-attacks-how-ai-phantom-dependencies-create-security-risks (2025). Accessed 31 Oct 2025.
23. OWASP Foundation: CICD-SEC-3—Dependency Chain Abuse (OWASP Top 10 CI/CD Security Risks). https://owasp.org/www-project-top-10-ci-cd-security-risks/CICD-SEC-03-Dependency-Chain-Abuse (2025). Accessed 31 Oct 2025.
24. Kaspersky: How AI creates "slopsquatting" supply-chain risks. https://www.kaspersky.com/blog/ai-slopsquatting-supply-chain-risk/53327/ (2025). Accessed 31 Oct 2025.
25. Mend.io: The Hallucinated Package Attack—Slopsquatting Explained. https://www.mend.io/blog/the-hallucinated-package-attack-slopsquatting/ (2025). Accessed 31 Oct 2025.