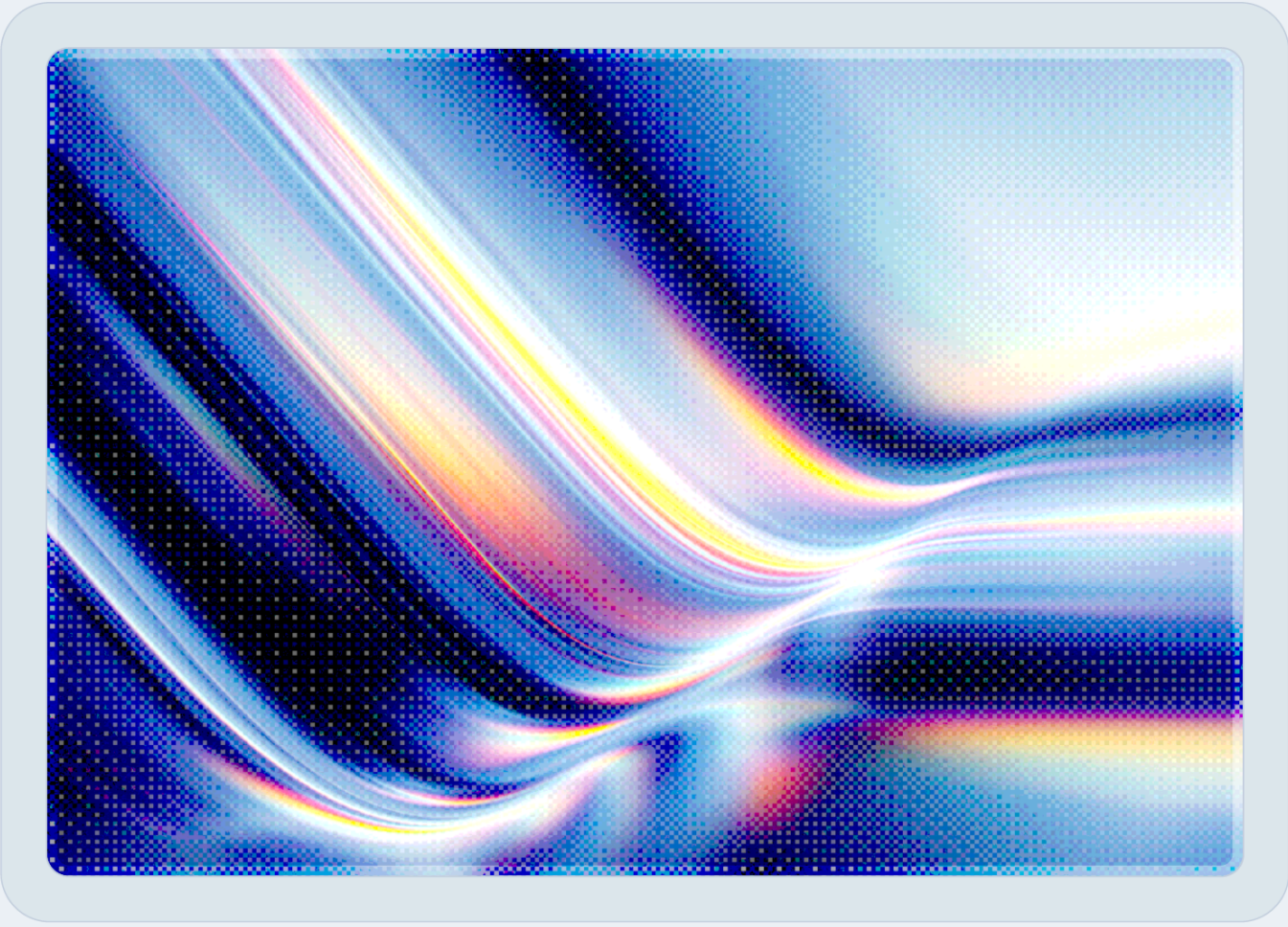
CSAI Foundation | Cloud Security Alliance

AI Package Registry Crisis: Unguarded Critical Infrastructure

How npm and PyPI Became the Soft Underbelly of the AI Development Stack

2026-06-23

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- According to Phoenix Security threat intelligence data, the first half of 2026 produced 37 distinct supply chain campaigns and 497 cataloged malicious packages across npm and PyPI — a 2.6× increase in campaign count and 4.5× increase in package volume over the entirety of the preceding year [1].
- The May 2026 "Mini Shai-Hulud" worm attack compromised 169 npm packages and 2 PyPI packages (373 malicious package-version entries total), including the official Mistral AI Python SDK, the TanStack router ecosystem, Guardrails AI, and UiPath, representing packages with more than 518 million cumulative downloads [2][3].
- A structurally new threat has emerged: the Mini Shai-Hulud campaign was the first confirmed instance of malicious packages carrying valid SLSA Build Level 3 provenance attestations, demonstrating that process integrity controls can be defeated at the build pipeline level [4].
- "Slopsquatting" — a novel attack vector exploiting AI-generated hallucinations of non-existent package names — opens an expanding attack surface that grows with every new LLM release and that current registry controls are not designed to address [5].
- The MCP package ecosystem on npm comprises 973 packages with 71% maintained by a single individual; security researchers found that 9 of 11 MCP registries failed to detect malicious uploads during testing [6].
- AI coding agents (Claude Code, Cursor, GitHub Copilot Workspace) are increasingly executing `npm install` and `pip install` autonomously in default configurations, substantially reducing the human review checkpoint that has, in conventional development workflows, provided a final opportunity to catch unfamiliar dependencies before execution [1].
- Organizations relying on SLSA attestations, trusted publishing, or mandatory 2FA as primary defenses should treat those controls as necessary but insufficient; they require runtime policy enforcement, dependency pinning, and continuous behavioral monitoring of install-time scripts.

Background

The Registries That Power AI Development

npm and PyPI are not peripheral infrastructure. They are the load-bearing columns of the modern AI development stack. Every major AI framework — LangChain, LlamaIndex, Hugging Face Transformers, OpenAI's Python SDK, Anthropic's SDK, LiteLLM, Guardrails AI, and hundreds of Model Context Protocol server implementations — is distributed through one or both registries. When a developer installs a dependency in a CI/CD pipeline or AI coding agent environment, npm or PyPI is the authoritative source from which that code arrives. For the majority of development environments using default configurations, no vetting layer exists between registry publication and code execution; organizations that have deployed private registry mirrors or mandatory SCA scanning are exceptions, and those controls are described in the Recommendations section of this note.

The scale of these registries has accelerated dramatically under AI-driven development. PyPI added approximately 130,000 new packages in 2025, nearly matching its cumulative total through 2018 [10]. npm growth has been similarly explosive, driven in part by the rapid expansion of AI-adjacent tooling and Model Context Protocol implementations. This growth creates an exponentially larger needle-in-a-haystack problem for the small teams responsible for registry security. PyPI processed over 2,000 malware reports in 2025, achieving median response times of 39 hours for removal — more than sufficient time for a self-propagating worm to cascade through an ecosystem [7].

From Human Error to Automated Exploitation

For most of the history of open source registries, supply chain attacks required human error at multiple stages: a developer had to typo a package name, or an attacker had to phish a maintainer, or both. The barriers were real, even if imperfect. Two developments in 2025 and 2026 removed those barriers in meaningful ways.

The first was the emergence of self-propagating worms capable of automating the jump from one compromised maintainer account to hundreds of poisoned packages. The original Shai-Hulud worm, which appeared in September 2025, was the first confirmed self-replicating malware observed in the npm ecosystem. It demonstrated that a single stolen maintainer token could function as a seed from which an attacker's automated tooling would publish malicious versions of additional packages, harvesting further credentials and repeating the cycle without human direction between steps [8].

The second development was the maturation of AI coding agents as first-class participants in the dependency installation workflow. Where a human developer might pause to verify an unfamiliar package, AI agents executing `npm install` or `pip install` as part of scaffolding or code generation tasks do not. They treat the registry as authoritative and execute immediately. This has substantially reduced the human review checkpoint for a growing fraction of dependency installations in 2025 and 2026 [1].

Security Analysis

The Mini Shai-Hulud Campaign: A Watershed Event

On May 11, 2026, threat actors tracked as TeamPCP deployed a worm that compromised over 160 packages across the npm and PyPI ecosystems within a five-hour window, ultimately reaching 171 packages in total [2][3]. The campaign, nicknamed Mini Shai-Hulud by researchers at Orca Security and Aikido Security, hit projects of extraordinary breadth: the complete TanStack router ecosystem (42 packages), Mistral AI's official Python SDK (`mistralai==2.4.6`), Guardrails AI (`guardrails-ai==0.10.1`), UiPath's automation tooling suite (65 packages), and OpenSearch's JavaScript client [2][4]. The cumulative download count across affected packages exceeded 518 million, making this the largest supply chain event by exposure metrics recorded to that point [2].

CVE-2026-45321 was assigned specifically to the TanStack compromise and carries a CVSS score of 9.6 [4]. For the TanStack packages, researchers established that the attacker had exploited GitHub Actions' `pull_request_target` trigger, combined with Actions cache poisoning and runtime extraction of an OIDC token from the GitHub Actions runner process [3]. This exploitation path had a significant structural consequence: the malicious packages carried valid SLSA Build Level 3 provenance attestations. SLSA Level 3 is currently the highest widely deployed integrity certification for package builds, designed to provide cryptographic assurance that a package was built by a trusted pipeline from a known source [14]. The fact that attackers were able to abuse the build pipeline itself — rather than circumventing attestation from outside — means that SLSA attestations alone cannot be treated as sufficient integrity assurance when the upstream build environment may be compromised. This does not invalidate SLSA attestations as a security control; they remain useful signals that warrant inclusion in any multi-layered defense posture. It does confirm, however, that attestations require layered verification of the build system's own integrity [3][4][14].

The malware payload was technically sophisticated. It targeted 1Password and Bitwarden vault contents alongside SSH keys, AWS and GCP credentials, Kubernetes service account tokens, GitHub tokens, and npm publishing credentials — the last of which enabled the worm to use each newly compromised maintainer account as a launchpad for further infection [2]. Exfiltration used three redundant channels: a typosquatted domain (`git-tanstack.com`), the decentralized Session messenger network, and Dune-themed GitHub repositories created using stolen tokens [2].

Slopsquatting: AI Hallucination as Attack Surface

Traditional typosquatting relies on human typographic error. Attackers register package names one character removed from a popular package (`reqeusts` for `requests`, `langchan` for `langchain`) and wait for developers to misspell a dependency. This attack vector has been understood and partially mitigated through linting tools and registry-side detection.

Slopsquatting operates differently. It exploits the tendency of large language models to hallucinate plausible but non-existent package names when generating code. A developer asks an AI coding assistant to scaffold a new project; the assistant suggests a dependency list that includes one or more packages that do not exist. Security researcher Seth Larson coined the term "slopsquatting" to describe the resulting attack vector, in which attackers pre-register those hallucinated names as malicious packages before legitimate developers discover the pattern [5].

In Trend Micro's analysis of hallucinated package names across several LLM code generation scenarios, approximately 38% closely resemble real packages (suggesting model confusion during training), 13% are typographic variants of real names, and 51% are entirely fabricated — figures that sum to approximately 100% after rounding [5]. Critically, the hallucinations are not random: the same model will hallucinate the same non-existent package name consistently across multiple code-generation sessions, allowing attackers to identify high-value targets by querying models repeatedly and cataloging recurring hallucinations. Bar Lanyado of Lasso Security documented an early example in 2024, observing that multiple AI models consistently hallucinated a Python package called `huggingface-cli` despite the genuine tool having a different install path [5].

The attack surface created by slopsquatting is not bounded by the number of existing packages — it expands with every new LLM release and every new context in which AI assistants generate code. Unlike typosquatting, which is a known and partially mitigated threat, slopsquatting has no complete defensive analog in current registry policy: registries cannot proactively detect which names will be hallucinated, and AI models do not currently validate dependencies against live registry data before recommending them.

The MCP Ecosystem: An Unguarded Frontier

The Model Context Protocol (MCP) ecosystem represents the newest and most structurally vulnerable layer of AI developer infrastructure. MCP packages — which enable AI assistants to connect to external tools, APIs, and data sources — numbered 973 on npm as of June 2026, with 71% maintained by a single individual — suggesting, though not confirming, that the large majority of packages lack any organizational security review process [6]. This concentration of single-maintainer packages creates a high-value, low-resistance target class: a single phished developer can expose every AI agent that has installed their package.

Security researchers testing MCP registries in 2026 found that 9 of 11 registries examined failed to detect deliberately uploaded malicious packages [6]. This failure rate is not surprising given the age and resourcing of MCP-specific registries relative to their growth rate, but it is significant: MCP packages are consumed directly by AI agents as capability extensions, meaning a malicious MCP package can redirect agent behavior, exfiltrate data processed by the agent, or escalate privileges within the agent's operating environment.

The TrapDoor campaign, active from May 22, 2026, illustrated the intersection of MCP exploitation with the broader AI developer environment. TrapDoor used zero-width Unicode characters to inject hidden instructions into AI coding assistant configuration files — including `.cursorrules` files and configuration analogous to `CLAUDE.md` — and submitted pull requests to major open source AI repositories including `langchain-ai/langchain`, `browser-use/browser-use`, and `langflow-ai/langflow` to distribute the poisoned configurations [9]. Separately, the WAVESHAPER.V2 backdoor malware specifically enumerated MCP configuration files for Claude Code, Cursor, Windsurf, and VS Code Continue, injecting rogue server definitions that repurposed the AI assistant into an exfiltration channel [1].

The Governance Gap: Underfunded, Overwhelmed Registries

The attack volumes documented above exist in structural tension with the governance and resourcing reality of the registries under attack. As of early 2026, reporting from The Register based on OpenSSF statements indicated that open source registries face significant funding gaps even as security costs — incident response, malware triage, attestation infrastructure, legal compliance — have risen [7]. npm is governed by GitHub (a Microsoft subsidiary) and benefits from Microsoft's infrastructure backing — resources not available to the majority of community-operated package registries. PyPI is operated by the Python Software Foundation with a small paid staff and relies substantially on volunteer labor and grant funding [10].

Neither registry currently operates a pre-publication review process, a structural limitation that the funding constraints documented by OpenSSF make unlikely to change in the near term [7]. Both operate on a publish-first, remediate-second model that made sense when the ecosystem was smaller and attacks were lower-volume. In 2026, that model is under strain it was not designed to handle. PyPI achieved a median malware response time of 39 hours in 2025 [7]. A self-propagating worm operating at the speed Mini Shai-Hulud demonstrated — over 160 packages compromised in five hours — can achieve broad propagation across the registry long before a human reviewer responds to the initial malware report, providing ample time for credential exfiltration from any end-users who install the compromised versions.

Both registries have made meaningful defensive improvements. PyPI mandated two-factor authentication for critical project maintainers and reported over 50,000 projects using trusted publishing by end of 2025, representing more than 20% of all file uploads [10]. GitHub announced mandatory 2FA for npm publishing, granular tokens limited to seven-day lifetimes, and expanded trusted publishing support for GitHub Actions and GitLab CI/CD in 2025 [11]. The Mini Shai-Hulud campaign demonstrated that these controls are necessary but insufficient when attackers can compromise the build pipeline itself and produce attestation-signed malicious packages. Process integrity controls that assume the build environment is trustworthy cannot detect an attacker who has compromised that environment.

The Linux Foundation responded to mounting pressure by launching the Sustaining Package Registries Working Group in May 2026, focused on governance enablement, collective defense, economic sustainability, and ecosystem education [12]. CISA and OpenSSF have published a joint "Principles for Package Repository Security" framework as voluntary guidance [13]. These initiatives acknowledge the structural problem without yet resolving it.

Recommendations

Immediate Actions

Organizations deploying AI development tooling should audit their AI agent configurations for signs of MCP configuration tampering, particularly reviewing MCP server definitions in tools such as Claude Code, Cursor, and Windsurf for unexpected entries. Any AI agent operating with file system or shell execution access should be reviewed for MCP configurations that were not explicitly authorized by a human administrator.

All dependencies used in AI development pipelines — including AI framework packages (LangChain, LlamaIndex, LiteLLM, Mistral AI SDK, Guardrails AI), MCP server implementations, and related tooling — should be version-pinned to known-good releases and verified against published cryptographic hashes. Lock files (`package-lock.json`, `requirements.txt` with pinned hashes, `Pipfile.lock`) should be committed to source control and treated as security artifacts requiring review on change.

CI/CD pipelines that install AI-related packages should integrate Software Composition Analysis (SCA) tooling from vendors with real-time malicious package feeds. Detection windows for registry-level response are measured in hours to days; automated pipeline-level scanning is the primary realistic detection layer for fast-moving campaigns.

Short-Term Mitigations

Organizations should establish explicit policy governing which packages may be installed in AI agent environments, implemented through network egress controls, private registry mirrors, or allowlisting at the package manager level. AI agents that autonomously execute `npm install` or `pip install` should operate under restricted network policies that prevent direct registry access without an intermediate verification step. AI coding agent tools such as Claude Code can be configured to require explicit human approval before executing install commands — a configurable behavior that security teams should enable by default in enterprise deployments.

SLSA attestations should be treated as one signal in a multi-control posture, not as sufficient assurance. Mini Shai-Hulud demonstrated that SLSA Build Level 3 can be satisfied by malicious builds when the build environment itself is compromised. Organizations relying on attestations should additionally verify the integrity of the build system configuration and monitor for anomalous CI/CD workflow changes in upstream package repositories.

Developers using AI coding assistants to generate dependency requirements should validate every suggested package against the official registry before installation. Dependency suggestions from LLMs should be treated as untrusted input until verified. Package names that are plausible but unfamiliar warrant particular scrutiny — the slopsquatting threat specifically exploits the tendency of generated code to appear authoritative.

Strategic Considerations

The structural vulnerability of npm and PyPI as unmanaged publish-first registries is not addressable through user-level controls alone. Enterprises with significant AI development activity should consider advocating through the Linux Foundation Sustaining Package Registries Working Group and the OpenSSF for adoption of publication delay windows, automated behavioral analysis of new packages, and coordinated malware sharing between registries. The current model in which each registry remediates independently creates gaps exploited by cross-registry campaigns.

The MCP package ecosystem requires governance investment before it reaches the attack surface density that has made npm and PyPI high-value targets. Security teams should evaluate MCP package usage across their AI tooling and establish internal registry or allowlisting policies for MCP servers before external MCP registries mature their security posture.

AI development environments deserve the same security treatment as production environments. Developer workstations running AI coding agents typically hold LLM API keys, cloud credentials, CI/CD tokens, and container registry access — exactly the credential classes targeted by Mini Shai-Hulud and the Mastra supply chain attack. Defense-in-depth for these environments should include endpoint detection, privileged access management for developer credential stores, and runtime monitoring of processes spawned by package install hooks.

CSA Resource Alignment

The vulnerabilities documented in this note span multiple domains addressed by CSA's AI Controls Matrix (AICM) v1.0. The AICM's AI Supply Chain Security domain directly addresses the risk of malicious or compromised dependencies entering the AI model and application development pipeline. Organizations implementing AICM controls for Orchestrated Service Providers (OSPs) — the role most enterprises occupy when deploying AI agent frameworks — should map their package registry governance practices to the supply chain controls in that implementation guide, which addresses dependency verification, artifact integrity, and third-party component risk.

CSA's MAESTRO framework for agentic AI threat modeling is directly applicable to the AI agent attack surface described here. MAESTRO's threat model for AI agents addresses the scenario in which an AI agent autonomously executes system operations — including package installation — and should be consulted when establishing policy for which operations AI agents may perform without human approval. The autonomous `npm install` behavior of contemporary AI coding agents represents a concrete instance of the agentic action scope problem MAESTRO was designed to analyze.

The Zero Trust principles in CSA's guidance apply to package installation in AI environments as they do to network access: no package should be trusted by virtue of its registry origin alone. Trust should be established through cryptographic verification of artifact integrity, behavioral analysis of install-time scripts, and network policy that limits the blast radius of a compromised dependency. CSA's STAR program provides an assessment mechanism through which organizations can evaluate the security posture of AI tooling vendors, including their supply chain practices, against these expectations.

CSA's AI Organizational Responsibilities guidance is relevant to the governance gap identified in this note. The responsibility for securing AI development pipelines does not rest with registries alone; it is shared between registry operators, framework maintainers, enterprise security teams, and the AI coding assistant vendors whose products execute package installations autonomously. Establishing clear accountability for each layer of that supply chain is a prerequisite for systemic improvement.

References

[1] Phoenix Security. "The Acceleration: How Supply Chain Attacks Went Industrial Across npm, PyPI, VS Code, and AI Agent Tooling." Phoenix Security, June 2026.

[2] Orca Security. "TanStack and 160+ npm/PyPI Packages Compromised in Supply Chain Worm Attack." Orca Security Blog, May 2026.

[3] The Hacker News. "Mini Shai-Hulud Worm Compromises TanStack, Mistral AI, Guardrails AI & More Packages." The Hacker News, May 2026.

[4] Tenable. "Mini Shai-Hulud Supply Chain Attack CVE-2026-45321 FAQ." Tenable Blog, May 2026.

[5] Trend Micro. "Slopsquatting: When AI Agents Hallucinate Malicious Packages." Trend Micro Security, 2025.

[6] Security Boulevard. "973 MCP Packages, 71% Single-Maintainer: A Practitioner's Guide to AI Developer Security." Security Boulevard, June 2026.

[7] The Register. "Open source registries underfunded as security costs rise." The Register, February 2026.

[8] SecurityWeek. "Over 100 NPM, PyPI Packages Hit in New Shai-Hulud Supply Chain Attacks." SecurityWeek, 2025–2026.

[9] The Hacker News. "TrapDoor Supply Chain Attack Spreads Credential-Stealing Malware via npm, PyPI, and CratesIO." The Hacker News, May 2026.

[10] PyPI. "PyPI in 2025: A Year in Review." Python Package Index Blog, December 2025.

[11] DevOps.com. "How GitHub Plans to Secure npm After Recent Supply Chain Attacks." DevOps.com, 2025.

[12] Linux Foundation / Open Source For You. "Linux Foundation Launches Open Source Registry Initiative Amid AI-Driven Software Supply Chain Pressure." Open Source For You, May 2026.

[13] CISA and OpenSSF. "Principles for Package Repository Security." Open Source Security Foundation, February 2024.

[14] Open Source Security Foundation. "SLSA: Supply Chain Levels for Software Artifacts." SLSA Specification, 2024.