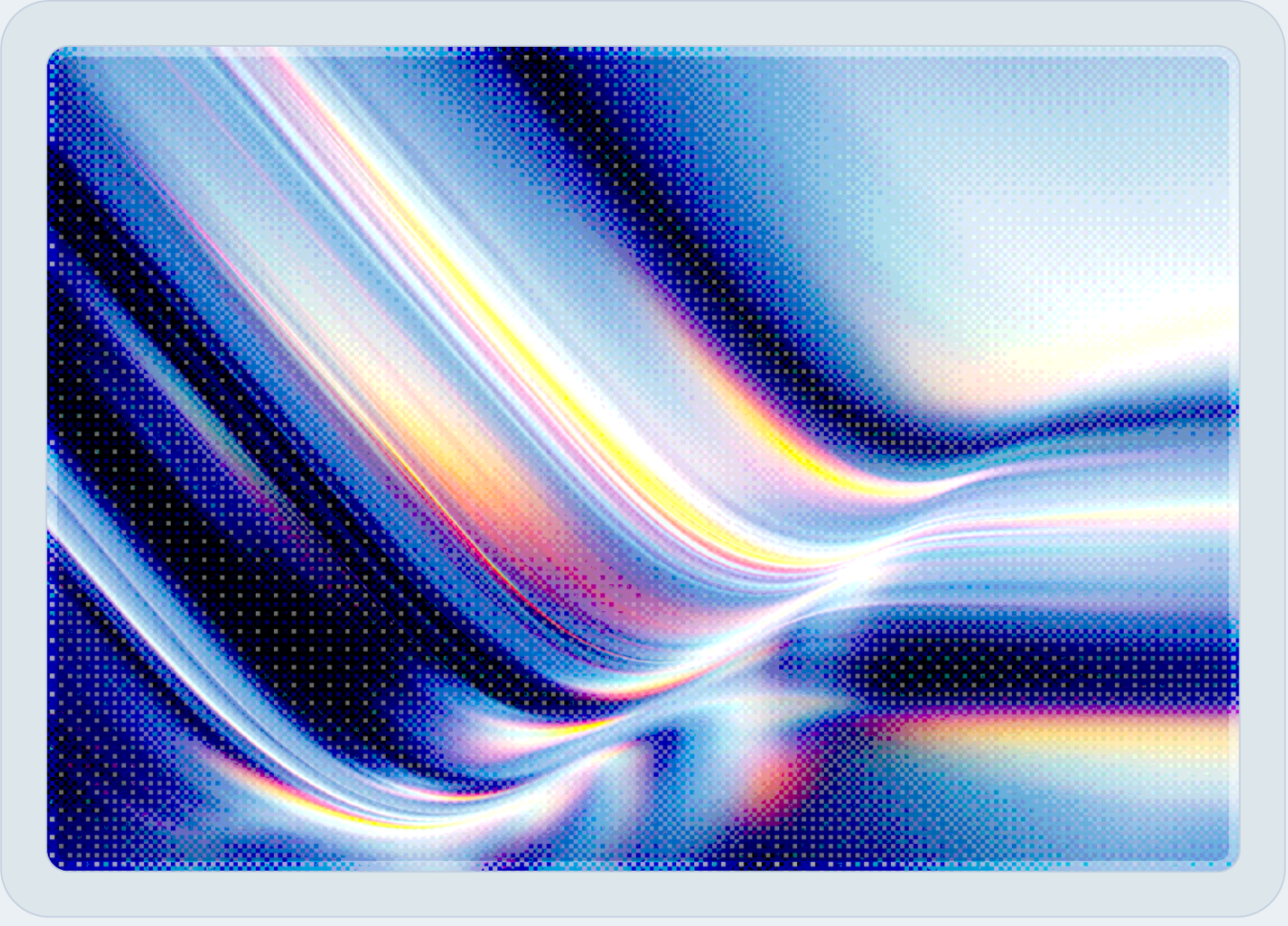
CSAI Foundation | Cloud Security Alliance

MCPhantom: Model Knowledge as an Unpatchable Software Supply Chain

Why frozen training data makes AI coding agents a correlated dependency risk

2026-11-10

AI-assisted Rapid Research



CSAI

cloud security alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

On November 2, 2026, Glow Labs, the research team at Glow Security, published research it calls "MCPhantom," which documents AI coding agents installing deprecated Model Context Protocol (MCP) servers whose names they recalled from training data rather than from any live source [1][2]. In the sessions Glow reviewed where a deprecated server was installed or suggested, the agent appears to have supplied the package name itself in 98% of cases; in the remaining 2%, the user requested the old package by name [1]. That figure describes only sessions in which a deprecated install occurred, so it is not a general error rate, and Glow has not disclosed the total number of sessions reviewed [1][3].

The affected packages are Anthropic's original MCP reference servers, most of which have been archived. The archive repository states that no security updates or bug fixes will be provided [4]. Glow reports four high-severity vulnerabilities across these servers, including a remote code execution flaw in the Puppeteer server that can be triggered through prompt injection, and the maintainers closed the report as out of scope because the code is archived [1][2]. Glow counted roughly 17.35 million npm downloads across 12 of the archived servers between July 10, 2025, and its publication date [1]; npm counts include automated and repeated fetches, such as CI runs and `npx` invocations, and do not represent unique installations.

The behavior appeared across Claude Code, Cursor, Codex CLI, Claude Cowork, and Claude chat, with models from Anthropic, OpenAI, and xAI [1][2]. CSA's assessment is that MCPhantom is best understood not as a bug in any one agent but as an early, measurable instance of a broader structural risk. A model's parametric knowledge acts as a dependency resolver that has no update channel, and because frontier models are trained on overlapping public corpora from similar time windows, their stale recommendations are likely to be correlated across vendors. Organizations should treat agent-originated package and server choices as untrusted input, inventory MCP configurations on developer endpoints now, and enforce selection policy at execution time rather than relying on the model to know what is current.

Background

MCP is the open protocol, introduced by Anthropic in November 2024 [15], that lets AI assistants and coding agents connect to external tools and data sources through "servers." Glow reports that Anthropic launched MCP with 20 reference servers intended to demonstrate the protocol, and that 13 of them have since been archived [1]; the archive repository itself currently lists 14 [4]. The archived set, maintained in the `modelcontextprotocol/servers-archived` repository, includes servers for GitHub, GitLab, Slack, PostgreSQL, Puppeteer, Google Drive, Google Maps, Brave Search, Redis, and others [4]. The repository describes them as historical reference implementations and states plainly that "no security guarantees are provided" [4]. For many of these services, the vendor now publishes its own maintained MCP server, such as GitHub's official server [5].

The problem, in CSA's assessment, is one of timing. Because these reference servers were the primary implementations available when MCP launched, much of the public documentation from late 2024 and early 2025, including tutorials, README files, blog posts, and configuration examples, referenced packages such as `@modelcontextprotocol/server-github`, and that material falls within the training windows of many current models. When a developer asks an agent to "set up a GitHub MCP," the agent fills in the package name it learned, writes it into an MCP configuration or hands the developer an `npx` command, and npm installs the deprecated package [1]. In Glow's GitHub example, the agent passed over GitHub's own server in favor of the archived Anthropic reference implementation [1][3].

Glow's disclosure timeline indicates the issue was not quickly resolvable through normal channels. Glow reported a flaw via HackerOne on July 30, 2026, which was closed as out of scope the same day, and reported all four flaws to Anthropic's CVE Numbering Authority between October 4 and 11, 2026; maintainers subsequently closed the advisory, stating no fix would come from the archive [1]. Help Net Security reports that the flaws have no CVE identifiers and that no exploitation has been observed [2]. Glow is a security vendor that sells tooling addressing this problem, and CSA is not aware of independent replication of the session-level findings, so readers should weigh the quantitative claims accordingly.

Security Analysis

A New Variant of a Known Failure Mode

MCPhantom sits within a family of failures in which a language model's memory, rather than an authoritative registry, decides what software enters an environment. The best-known member of that family is package hallucination, or "slopsquatting." Spracklen et al., in work presented at USENIX Security

2025, generated 576,000 code samples across 16 LLMs and found hallucinated package rates averaging at least 5.2% for commercial models and 21.7% for open-source models, yielding 205,474 unique nonexistent package names that attackers could register [6]. A related line of research found that all seven code LLMs evaluated in an ICSE 2025 study produced deprecated API usages, which the authors attributed to deprecated usages in training data and to the models' lack of deprecation knowledge at inference time [7].

MCPhantom differs from both in ways that matter for defenders. Slopsquatting requires an attacker to register a name before the risk materializes, and registry-side defenses can catch newly registered, low-reputation packages. The packages in MCPhantom are real, legitimately published, carry a trusted namespace, and have millions of downloads, so every reputation signal points toward installing them. Deprecated API usage typically produces code that compiles but degrades; MCPhantom installs a running, tool-invoking process on the developer's machine with whatever privileges the agent grants it.

Failure mode	What the model recalls	Does the artifact exist?	Primary risk	Signals that catch it
Package hallucination (slopsquatting)	A name that never existed	Only if an attacker registers it	Malicious code under a squatted name	Registry lookup, package age, low download count
Deprecated API usage	A real but superseded function or interface	Yes	Insecure or broken code patterns	Linters, deprecation warnings, code review
MCPhantom	A real, trusted, but abandoned package	Yes, with high downloads and trusted scope	Unpatched vulnerable server running locally with agent privileges	Few: reputation signals are favorable and no CVE exists

Why Frozen Knowledge Is "Unpatchable"

Conventional software supply chains have a remediation path, however slow. A maintainer issues a fix, a CVE is published, scanners flag the vulnerable version, and dependency managers move users to the patched release. MCPhantom shows that each link in this chain is absent or ineffective here. The maintainers have declined to patch the archived code [1][2]. npm's unpublish policy allows an owner to

remove a package older than 72 hours only if it has no public dependents, fewer than 300 downloads in the past week, and a single owner, and recommends deprecation otherwise [8]. Help Net Security reports that `server-github` alone had 185,239 weekly downloads and six owners, placing it far outside those criteria [2]. Deprecation itself is a weak control in this context: Glow notes that npm deprecation removes a package from search results, but `npx` still installs it after printing a warning [1].

The deeper issue is that the model's knowledge cannot receive the deprecation signal at all. Model weights are frozen at a training cutoff, and the cost of retraining means vendors refresh that knowledge on release cycles rather than in response to individual package deprecations. Glow's researcher Nadav Claude Cohen told Help Net Security that updating model knowledge of package names is unlikely for this reason [2]. Glow's Ophir Arie summarized the condition as "the model's memory is now a package manager with no deprecation feed" [1][2]. Even when the agent does consult a live source, it may not act on the result: Cohen described a session in which the agent checked npm, received "This package is deprecated," installed the package anyway, and never told the user [2]. This single reported session suggests that giving an agent access to live registry data does not, on its own, ensure that it will act on a deprecation notice; an enforcing control outside the model is still needed.

Correlation and Monoculture

The aspect of MCPhantom most likely to be underestimated is correlation. Glow observed the pattern across models from three different vendors and five different agent harnesses [1]. That is consistent with a common cause. Although vendors do not disclose the composition of their training data, frontier models are widely understood to draw on overlapping public web and code corpora, and MCP documentation from the reference-server era is publicly available to all of them. The evidence has limits, however. Glow has not published per-vendor rates, and three of the five affected harnesses are Anthropic products, so the observation establishes that the behavior occurs across vendors but not how strongly it is correlated. Other factors may also contribute, including live web searches that still surface outdated tutorials and the continued prominence of the deprecated packages in search results and on the npm registry. If shared training data is the dominant cause, the practical consequence is that diversity of agent vendors may provide less diversity of risk than security teams assume, because the stale recommendation originates in shared training data rather than in vendor-specific logic.

Security researchers have warned about the dynamics of this kind of convergence for decades. Geer and co-authors argued in 2003 that a software monoculture lets a single flaw propagate across an entire population of systems simultaneously [9]. Kleinberg and Raghavan later showed formally that when many decision-makers adopt the same algorithm, collective outcomes can degrade even if that algorithm outperforms each decision-maker individually [10]. MCPhantom illustrates the dynamics both works describe: correlated errors propagating across a population that relies on a shared decision source. The

decision of which server to install has migrated from many individual developers, some of whom would have found the vendor's current documentation, to a small number of models that share a similar frozen snapshot of the ecosystem. When that snapshot is wrong, the error can recur across many installations at once, and it persists for as long as those model versions remain in use.

This correlation also changes the attacker's calculus. An adversary who identifies an exploitable flaw in a widely recalled but abandoned package gains a target that is installed predictably, across vendors, with no patch forthcoming. Glow notes that the likelihood of these flaws being discovered has risen with AI-assisted bug hunting [1]. CSA infers, without evidence of current exploitation, that model-recalled abandoned packages may become an attractive class of targets, because the attacker can predict which packages agents will install by querying the same publicly available models.

Why Existing Controls Miss It

Glow and Help Net Security describe a consistent pattern in which each standard enterprise control sees an unremarkable event [1][2]. The table below summarizes that analysis.

Control	What it observes	Why it misses MCPhantom
Endpoint detection and response (EDR)	A trusted Node.js process installing a public npm package	Behavior is indistinguishable from routine development activity
Software composition analysis (SCA)	Project manifests and lockfiles	`npx`-launched servers appear in no manifest, and the flaws have no CVE
Software inventory	Installed applications	Servers run inside a Node process and are not registered as software
Agent harness guardrails	Tool-use permissions	Harnesses observed by Glow do not check or enforce deprecation status, and broad browsing permission covers the Puppeteer RCE path

The Puppeteer flaw illustrates how these gaps compound. According to Glow, a prompt injection can cause the agent to launch an arbitrary process instead of a browser, and the harness's guardrails do not intervene because the developer's permission to browse effectively authorizes the command [1]. The vulnerable component, the injection vector, and the execution path all sit outside the visibility of traditional supply chain tooling.

Recommendations

Immediate Actions

Security teams should begin by establishing whether deprecated reference servers are present in their environment. This means searching developer endpoints as well as code repositories for MCP configuration files and identifying entries that reference the archived `@modelcontextprotocol/server-*` packages [1][3], since these entries do not appear in the manifests and lockfiles that SCA tools scan. Common locations include `~/.claude.json` and `~/.cursor/mcp.json` on endpoints, along with project-level `.mcp.json` files that are often committed to repositories [3]. Running `npm view <package> deprecated` against each match confirms status. Any instance of the archived Puppeteer server should be removed first, given the unpatched remote code execution path [1][3]. Where the service owner publishes its own maintained server, such as GitHub's official MCP server [5], teams should migrate to it; where no maintained alternative exists, teams should disable the capability rather than run archived code.

Short-Term Mitigations

Because the model will continue to recommend stale packages, the durable fix is to move server selection out of the model and into policy. Glow recommends approving one MCP server per service as an enterprise standard and adding npm deprecation status to approval criteria [1]. Teams can enforce that standard through the pre-execution hooks many agent harnesses already support, intercepting commands and configuration writes that reference unapproved or deprecated MCP packages before they run; Cohen notes that hook-based enforcement works today, without waiting for harness makers, model makers, or package owners to act [2]. Organizations should also extend endpoint telemetry and software inventory to capture `npx`-launched processes and MCP configuration changes, so that agent-installed servers become visible to the same asset management processes as other software. Finally, the controls CSA previously recommended for slopsquatting, including verifying every agent-suggested package against the registry and requiring allowlists or human review before agents install packages, apply directly to MCP servers [11][12].

Strategic Considerations

Over the longer term, organizations should treat a model's training cutoff as a supply chain attribute, comparable to the version of a vulnerability database. Procurement criteria for AI coding agents should ask how the vendor grounds package and server recommendations in live registry data, whether the

harness warns on or blocks deprecated packages including those already installed, and how quickly the vendor responds to ecosystem-level deprecations [2][12]. Registry operators and protocol stewards could reduce exposure by publishing machine-readable deprecation and replacement metadata that harnesses can enforce, and by making `npx` refuse deprecated packages unless explicitly overridden; these are policy suggestions rather than announced plans. Finally, risk models should account for correlation: if the same stale recommendation can appear across every major model vendor at once, then multi-vendor agent adoption may not reduce concentration risk, and controls should be designed to be independent of any model's knowledge.

CSA Resource Alignment

CSA's July 2026 research note *AI Coding Agents: An Unaudited Supply Chain Node* argued that agents, their models, and their tool integrations have become supply chain dependencies that most organizations do not inventory, sign, or audit, and that agent-driven package resolution should be treated as a privileged action [12]. MCPhantom provides concrete evidence for that thesis and extends it in one important respect: the model's knowledge itself, not only its hallucinations or its exposure to prompt injection, is a dependency with its own version and its own end-of-life problem. The note's recommendations on extending provenance tracking to models and adding supply chain criteria to agent procurement apply directly. CSA's June 2026 paper *AI Coding Agents as Attack Surface: MCP, Poisoning, and Miasma* examines the MCP-specific side of this exposure, including MCP auto-execution and repository poisoning [16]. Where that paper addresses malicious configurations introduced into an agent's environment, MCPhantom addresses vulnerable servers the agent selects and installs on its own, and the two together cover both directions from which an untrusted MCP server can arrive.

CSA's April 2026 note *Slopsquatting: AI Code Hallucinations Fuel Supply Chain Attacks* addressed the case where models recall packages that do not exist [11]. MCPhantom is the complementary case, in which models recall packages that exist but should no longer be used. Read together, the two notes indicate that registry verification must check not only whether a package exists, but whether it is maintained, deprecated, or superseded.

The CSA AI Controls Matrix (AICM) v1.1 provides the control framework for operationalizing these recommendations [13]. Approved-server standards and deprecation-aware approval criteria map to the Supply Chain Management, Transparency, and Accountability (STA) domain; vulnerability tracking for packages that will never receive CVEs or patches maps to Threat and Vulnerability Management (TVM); pre-execution enforcement in agent harnesses maps to Application and Interface Security (AIS); and endpoint discovery of `npx`-launched servers maps to Universal Endpoint Management (UEM). For threat modeling, CSA's MAESTRO framework for agentic AI offers a layered approach in which the

foundation model layer and the agent ecosystem layer can be analyzed separately [14], which is useful for distinguishing risk that originates in shared training data from risk introduced by a specific harness or tool.

References

[1] Glow Labs (N. C. Cohen). "McPhantom: AI Coding Agents Are Installing Vulnerable Deprecated MCP Servers from Training Data." Glow Security, November 2, 2026.

[2] Help Net Security. "AI coding agents are installing MCP servers that will never be patched." Help Net Security, November 3, 2026.

[3] Piekwerk. "MCPhantom: agents chose deprecated MCP servers in 98% of tracked sessions, and no patch is coming." DEV Community, November 4, 2026.

[4] Model Context Protocol. "servers-archived: Archived MCP Reference Servers." GitHub, accessed November 10, 2026.

[5] GitHub. "GitHub MCP Server." GitHub, accessed November 10, 2026.

[6] Spracklen, J., et al. "We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs." 34th USENIX Security Symposium, August 2025.

[7] Wang, C., et al. "LLMs Meet Library Evolution: Evaluating Deprecated API Usage in LLM-based Code Completion." ICSE 2025 / arXiv:2406.09834, 2024.

[8] npm, Inc. "npm Unpublish Policy." npm Docs, accessed November 10, 2026.

[9] Geer, D., Pfleeger, C. P., Schneier, B., Quarterman, J. S., Metzger, P., Bace, R., and Gutmann, P. "CyberInsecurity: The Cost of Monopoly." Computer & Communications Industry Association, September 2003.

[10] Kleinberg, J., and Raghavan, M. "Algorithmic Monoculture and Social Welfare." Proceedings of the National Academy of Sciences 118(22), 2021.

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

[12] Cloud Security Alliance. "AI Coding Agents: An Unaudited Supply Chain Node." CSA AI Safety Initiative, July 8, 2026.

[13] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, June 2026.

[14] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." Cloud Security Alliance, February 6, 2025.

[15] Anthropic. "Introducing the Model Context Protocol." Anthropic, November 25, 2024.

[16] Cloud Security Alliance. "AI Coding Agents as Attack Surface: MCP, Poisoning, and Miasma." CSA AI Safety Initiative, June 28, 2026.