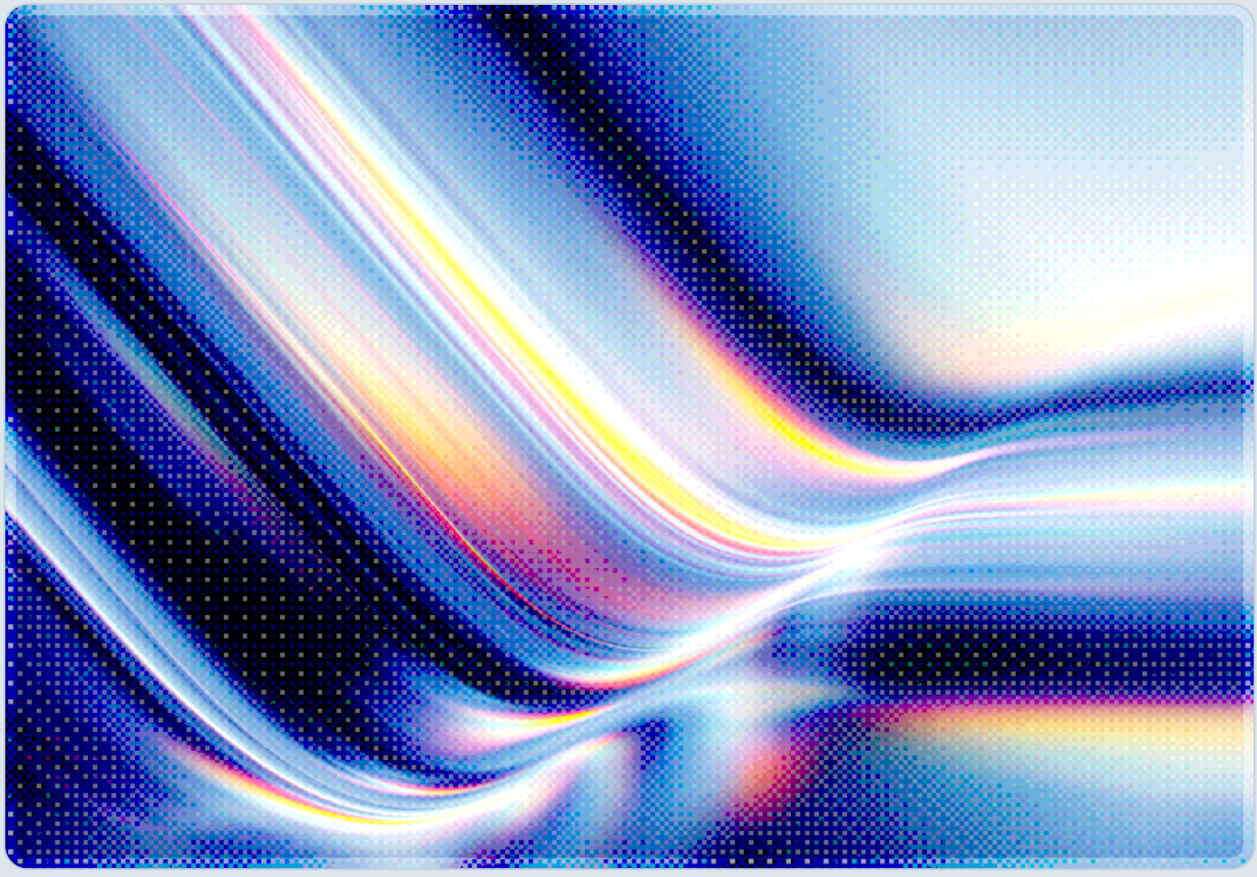
CSAI Foundation | Cloud Security Alliance

Poisoned Foundations: The AI Developer Toolchain Attack Surface

MCP Auto-Execution, Agent Skill Supply Chain Attacks, and Repository Poisoning

2026-06-30

AI-assisted Rapid Research



CSAI

cloud security alliance®

Table of Contents

Executive Summary

The tools that software developers use to build with AI have themselves become targets. Where traditional software supply chain attacks required adversaries to compromise a package registry or infiltrate a build pipeline, the AI developer toolchain introduces a more permissive threat model: natural language context is executable, configuration files can trigger remote shell commands before a user approves anything, and a single malicious skill published to a public registry can reach hundreds of thousands of developers simultaneously.

Three distinct but interconnected threat vectors define this emerging attack surface. The Model Context Protocol (MCP), which connects AI agents to tools, APIs, and data sources, prioritized extensibility and rapid adoption, with a security model that was left largely underspecified — a gap that national security agencies identified as the operational context for many of the vulnerabilities described in this paper. In parallel, the rapid growth of agent skill registries has replicated the worst historical patterns of open-source package ecosystems: low barriers to publication, minimal vetting, and a large installed base that ingests new content automatically. Meanwhile, the poisoning of source repositories — whether to manipulate model fine-tuning data or to inject adversarial instructions into agent context windows — represents a third vector that operates upstream of every other control.

This paper provides a technical analysis of each vector, examines real-world incidents that have already occurred, and offers guidance for enterprise security teams, AI developers, and platform operators. The paper also maps findings to relevant Cloud Security Alliance frameworks, including MAESTRO, the AI Controls Matrix (AICM), the Agentic Trust Framework (ATF), and the OWASP MCP Top 10, which together provide a governance and technical foundation for organizations seeking to manage these risks systematically.

Introduction and Background

The Developer as the New Attack Surface

For most of the history of enterprise security, the boundary between attacker and victim was relatively legible: attackers sought production systems, customer databases, or financial infrastructure, and defenders built perimeter and endpoint controls to protect those assets. The emergence of AI-augmented software development has complicated this model in a consequential way. Developer workstations and development

pipelines now run AI agents with broad access to code repositories, cloud credentials, local file systems, and outbound network connections. These agents operate with a degree of autonomy that traditional endpoint controls were not designed to govern.

The AI developer toolchain — encompassing coding assistants such as Claude Code, GitHub Copilot, and Cursor; agentic frameworks such as OpenClaw; and the protocol infrastructure that connects them — processes sensitive information at every stage. A developer's API keys, proprietary source code, internal documentation, and production credentials may pass through or be accessible to these tools — particularly when AI agents are configured with repository or shell access. When any component of this toolchain is compromised, the consequences extend well beyond the individual developer machine. Supply chain compromises at the toolchain layer can propagate silently to every repository the developer touches, every CI/CD pipeline they interact with, and every downstream user of the software they produce.

The MCP Adoption Curve and Its Security Gap

Anthropic introduced the Model Context Protocol in late 2024 as a standardized mechanism for connecting AI assistants to external tools, data sources, and services. The protocol's open design and the breadth of the integrations it enabled drove rapid adoption: within roughly eighteen months, MCP had accumulated more than 150 million downloads across its official SDK implementations in Python, TypeScript, Java, and Rust, with an estimated 200,000 server instances deployed across enterprise and development environments [1]. Public registry listings suggest the number of registered MCP servers grew from a few hundred in early 2025 to several thousand by mid-2026.

This pace of adoption outran the development of the protocol's security model. As the National Security Agency observed in its May 2026 advisory on MCP security design, the protocol was released with a flexible and underspecified design that gave implementers freedom but introduced significant ambiguity around safe usage [2]. The Five Eyes nations — through a joint advisory published the same month — warned that organizations adopting agentic AI services, including MCP-based architectures, should treat the security properties of these systems as fundamentally different from traditional software integrations [3]. The gap between deployment scale and security maturity is not a theoretical concern; it is the operational context in which the vulnerabilities described in this paper were discovered and exploited.

Scope of This Paper

This paper focuses on three threat vectors that share a common characteristic: they target the development and configuration layer rather than production systems, and they exploit the trust that developer tools are granted by default. The three vectors are: MCP auto-execution vulnerabilities and the protocol properties that enable them; agent skill and plugin supply chain attacks, including the compromise

of public registries; and repository poisoning attacks that corrupt the training data, context windows, and configuration files consumed by AI developer tools. The paper concludes with recommendations for enterprise security teams, platform operators, and standards bodies.

MCP Auto-Execution: The Protocol That Became an Attack Surface

Architectural Properties That Enable Attack

The Model Context Protocol was designed around a deceptively simple premise: AI agents should be able to invoke external tools as easily as calling a function. To achieve this, the protocol defines a standard interface through which a host application (typically an AI assistant or coding agent) can discover, invoke, and receive results from a server that exposes tools, resources, or prompts. The protocol's auto-discovery and auto-invocation properties — features that make MCP integrations quick to build and easy to use — are the same properties that create exploitable attack paths.

The most significant architectural departure from conventional client-server security models is what the NSA advisory describes as the inversion of the familiar interaction pattern [2]. In traditional client-server designs, a client requests data from a server, and trust flows from the client toward the server. MCP environments frequently invert this: the server defines the tools and the instructions for invoking them, and the AI client executes those instructions based on natural language context rather than explicit user commands. This means that a malicious MCP server can influence agent behavior by manipulating the tool descriptions, metadata, and responses that the agent processes — without requiring a conventional code execution vulnerability.

A second property that compounds this risk is the absence of mandatory authentication and authorization in the base protocol specification. Implementations vary considerably in whether they enforce authentication, and many public and enterprise deployments run MCP servers that accept connections or respond to tool invocations without verifying caller identity. The OWASP MCP Top 10, a community-maintained risk framework for MCP deployments, identifies insufficient authentication and authorization as MCP07:2025, noting that the multi-agent, multi-service nature of MCP ecosystems creates identity validation gaps at nearly every communication boundary [4].

Documented Vulnerabilities and CVEs

The abstract security concerns described above have materialized in concrete, exploitable vulnerabilities. OX Security researchers disclosed an architectural vulnerability in Anthropic's MCP reference SDKs that allows user-controlled input to flow directly into command execution without sanitization — a flaw present across the Python, TypeScript, Java, and Rust SDK implementations spanning the full 150 million download base [1]. The vulnerability class is straightforward: MCP tool implementations constructed command strings from protocol-supplied parameters without treating those parameters as untrusted input.

Two CVEs in Anthropic's Claude Code coding assistant illustrate how MCP and agent configuration properties combine to create developer-targeted attack paths. Check Point Research disclosed CVE-2025-59536 (CVSS 8.7), which demonstrated that repository-controlled configuration files could trigger arbitrary shell commands through Claude Code's Hooks mechanism before any user consent dialog appeared [5]. The same research identified CVE-2026-21852 (CVSS 5.3), which allowed API credential theft: Claude Code's use of the ANTHROPIC_BASE_URL environment variable to determine where to route prompts could be overridden through a malicious repository configuration file, redirecting credentials to an attacker-controlled endpoint [5]. Both vulnerabilities were exploited by the simple act of opening a repository — the kind of action a developer performs dozens of times per day without expecting security consequences.

Beyond Claude Code, CVE-2025-49596 (CVSS 9.4) demonstrated arbitrary command execution through unauthenticated MCP Inspector instances, and CVE-2026-33032 (CVSS 9.8) exposed an authentication bypass in an nginx-ui MCP endpoint that allowed attackers to restart servers and modify configuration files without credentials [6]. The pattern across these disclosures is consistent: MCP implementations frequently treat auto-execution as a convenience property and authentication as optional, creating exploitable gaps wherever the two assumptions meet.

Tool Poisoning and Rug Pull Attacks

Beyond specific CVE-class vulnerabilities, the MCP ecosystem has given rise to a class of attacks that exploit the protocol's trust model rather than implementation bugs. Tool poisoning occurs when a malicious actor publishes a tool — whether to a public registry, a shared internal server, or directly to an agent's configuration — that masquerades as a legitimate capability [7]. The malicious tool uses its description, metadata, or response data to inject instructions into the agent's reasoning context. Because the AI model processes tool descriptions as trusted natural language rather than as untrusted external data, a poisoned tool description can redirect agent behavior, exfiltrate data, or cause the agent to execute unintended commands.

The rug pull variant is operationally distinct: a tool is published with benign content, accumulates users and installations, and is then silently modified to introduce malicious behavior after trust has been established [8]. The OWASP MCP Top 10 identifies tool poisoning as MCP03:2025 and specifically calls out description

drift — the gradual or sudden alteration of a tool's described purpose after initial approval — as the mechanism that makes rug pull attacks effective [4]. Users approve a tool once during installation or first use; without automated change detection on tool metadata, no subsequent review occurs.

A third variant exploits naming collisions across servers. When an agent can connect to multiple MCP servers simultaneously, a malicious server can register a tool with the same name as a tool on a trusted server. The agent, selecting tools by name rather than by verified server identity, may invoke the malicious implementation believing it is calling the trusted one [8]. Cross-server shadowing of this kind requires no vulnerability in the agent or its infrastructure — it exploits the absence of a trusted tool namespace registry.

Attack Class	**Mechanism**	**MCP Architectural Property Exploited**	**OWASP Category**
Tool Poisoning	Malicious tool description injects instructions into agent context	Natural language tool descriptions are trusted content	MCP03:2025
Rug Pull	Tool metadata silently modified post-approval	No change detection on installed tool configurations	MCP03:2025
Server Name Collision	Malicious server registers duplicate tool name	No namespaced tool identity or server verification	MCP09:2025
Unauthenticated Execution	Tool invocations accepted without caller identity	Optional authentication in base protocol	MCP07:2025
Command Injection	Unsanitized parameters flow to shell execution	Auto-execution without input validation	MCP04:2025

Agent Skill Supply Chain: When Packages Carry Hidden Payloads

The Skill Registry Ecosystem

Agent frameworks have adopted a distribution model modeled closely on language package managers: developers define reusable capabilities — skills — in standardized files that can be shared, discovered, and installed from public registries. OpenClaw's ClawHub registry, which serves hundreds of thousands of developers worldwide, is among the largest public agent skill registries. Skills in this ecosystem typically include a SKILL.md file containing natural language instructions for the agent, configuration metadata, and optionally executable code. This design makes skills powerful and easy to create. It also makes them an ideal delivery mechanism for adversarial payloads, because the primary content the agent consumes is unstructured natural language that resists signature-based scanning.

The security properties of skill registries have lagged substantially behind their growth. A security audit of 3,984 skills from the ClawHub registry — reported by Snyk and attributed to a separate ToxicSkills study — found that approximately 36% contained at least one security flaw, 13.4% carried critical-severity issues, and 76 skills contained confirmed malicious payloads [9]. These figures emerged from an audit prompted by a specific incident; the baseline rate before active scrutiny may be higher. The analogy to early npm and PyPI security research is instructive: both ecosystems underwent similar periods of rapid growth followed by security audits that revealed widespread quality problems, with the key difference that malicious npm packages carry executable code that can be scanned, while malicious skills carry adversarial natural language instructions that cannot be detected by conventional means.

The ClawHavoc Campaign

The most consequential documented attack against an agent skill registry occurred in late January and early February 2026. The ClawHavoc campaign, tracked by multiple threat intelligence providers, involved a coordinated actor publishing 341 malicious skills to ClawHub over a short period [10]. The skills represented approximately 11.9% of the entire registry at the time of discovery, with 335 of the 341 traced to a single campaign [10][17]. Antiy Labs subsequently catalogued 1,184 historically published malicious skills associated with the campaign across ClawHub and related repositories [11]. The campaign reached an estimated 300,000 OpenClaw users [10].

The ClawHavoc skills deployed the Atomic macOS Stealer (AMOS), a credential-harvesting payload that had previously been distributed through cracked software. The attack evolved the AMOS delivery model significantly: rather than relying on users to voluntarily download and execute a malicious binary, the skills exploited AI agents as trusted intermediaries [10]. Skills masquerading as legitimate GitHub integration or

cloud configuration utilities presented fake setup prompts that requested user passwords, exploiting the user's existing trust in their AI agent to obtain credentials that AMOS would then harvest alongside browser passwords, keychain data, cryptocurrency wallet files, SSH keys, and Telegram session tokens.

The campaign demonstrated several characteristics that distinguish skill supply chain attacks from conventional software supply chain compromises. Traditional antivirus tools showed limited effectiveness against these payloads: VirusTotal failed to identify the majority of agent-targeted payloads in this category [17]. The adversarial instructions embedded in SKILL.md files were syntactically valid natural language, indistinguishable at the file level from legitimate skill content without semantic analysis. The attack also exploited the human-in-the-loop model in a novel way — the presence of a user confirmation dialog was the mechanism of compromise, not a barrier to it, because users trusted the agent presenting the dialog.

The SKILL.md Attack Surface

The CSA AI Safety Initiative published a research note on May 6, 2026 documenting the SKILL.md attack surface in detail [12]. The research describes how SKILL.md files — the natural language instruction files that define agent capabilities — represent a distinct security problem that existing controls do not address. Unlike executable code, the adversarial content in a malicious SKILL.md does not execute when scanned; it executes when read by an AI agent, at which point it enters the agent's reasoning context and can influence every subsequent action the agent takes.

A technically distinct variant of this attack class exploits Unicode Tag characters in the range U+E0000–U+E007F to embed invisible adversarial instructions within skill files [13]. These characters are rendered as whitespace or not rendered at all by most text editors, code review interfaces, and markdown viewers — but are processed as semantic content by language models. Researchers demonstrated this technique against Claude Code in early 2026, constructing a SKILL.md that appeared to reviewers as a standard GitHub integration skill while containing a hidden instruction to exfiltrate repository contents to an external server [13]. Anthropic addressed this specific technique in the Claude Code February 10, 2026 release, but the underlying class of attack applies to any agent platform that processes skill content without explicit Unicode sanitization, and no comparable patches had been publicly disclosed by other major agent platforms as of mid-2026, suggesting this remains an unaddressed exposure across much of the ecosystem.

Academic research published concurrently has begun to formalize the theoretical foundations of this attack class. A formal analysis of supply chain security for agentic AI skills modeled SKILL.md poisoning as a variant of adversarial prompt injection that operates at the supply chain layer rather than the user interaction layer, with distribution properties that make it substantially harder to remediate once it has propagated through a popular registry [14]. The distinction matters for incident response: when a malicious npm package is identified, maintainers can pull the package and issue an advisory. When a malicious skill has been ingested by thousands of agent instances as trusted context, the behavioral effects may persist in agent memory and cached context even after the original skill is removed. In enterprise deployments that collect agent outputs

for ongoing fine-tuning, there is a further risk — harder to detect and remediate — that poisoned behavior could be incorporated into subsequent model weights if the affected sessions were included in a fine-tuning corpus before the attack was identified.

Repository Poisoning: The Developer Workflow as Vector

Training Data and Fine-Tuning Contamination

The AI developer toolchain does not only depend on skills and MCP servers at runtime; it is shaped by the training and fine-tuning data that was used to build the underlying models, and in many enterprise deployments, by ongoing fine-tuning on internal codebases and documentation. Both pathways are vulnerable to poisoning attacks that operate at the data layer rather than the runtime layer, producing model behaviors that are difficult to detect because they emerge from weights rather than explicit instructions.

The OWASP LLM Top 10 (LLM04:2025) identifies data and model poisoning as a foundational risk category for deployed language model systems [15]. Industry research aggregators have catalogued multiple empirical findings that quantify this exposure, though primary sources for some figures remain difficult to trace independently through secondary publications. Available evidence suggests that 15% to 25% of scraped training datasets contain low-quality or unverifiable content that increases poisoning exposure [16]. The scale of exposure has grown substantially: open-source datasets on Hugging Face grew by more than 300% between 2022 and 2025, expanding the surface area available to adversaries seeking to contaminate future training runs [16].

The effectiveness of small-scale poisoning should not be underestimated. Research in the medical LLM domain has found that corrupting as little as 0.001% of tokens in a training dataset can increase harmful model outputs by 7 to 11 percent in that application context [16]. A separate line of research established that achieving a specified behavioral effect requires a constant number of poisoned samples rather than a percentage of the total dataset, and that this constant holds across model sizes from 600 million to 13 billion parameters [16]. This means that the barrier to effective poisoning does not scale with model size — a finding with significant implications for large-scale commercial model deployments.

Context Window and RAG Poisoning

Beyond training data, the AI developer toolchain is vulnerable to poisoning attacks that target the runtime context window rather than model weights. Retrieval-Augmented Generation (RAG) systems, which extend agent capabilities by retrieving relevant documents at inference time, are particularly exposed: if the document store from which the agent retrieves context has been compromised, the agent will reason from poisoned premises and produce outputs that reflect the adversary's intent rather than the user's [15]. This attack pathway is especially concerning in developer toolchain contexts, because developers frequently configure agents to retrieve from internal wikis, documentation repositories, or codebases that may not be subject to the same access controls as production systems.

Repository-based poisoning attacks can target the context window without touching model weights. Check Point Research's disclosure of CVE-2025-59536 in Claude Code illustrates a form of repository poisoning that operates at the configuration layer: a malicious .claude/settings.json file in a repository root contains hook definitions that execute shell commands when the agent opens the project [5]. The repository does not need to contain malicious source code; the poisoning payload is in a configuration file that the developer never reads directly. The Cline coding assistant experienced a closely related attack class in which prompt injection was delivered through GitHub Actions workflow files that the agent processed as part of its task context [9].

The "Clinejection" Pattern and CI/CD Pipeline Risks

The Cline supply chain attack documented by Snyk in early 2026 — referred to informally as "Clinejection" — demonstrated how an AI coding assistant could be turned into a supply chain attack vector by exploiting the agent's processing of repository files [9]. The attack introduced prompt injection payloads into GitHub Actions workflow files within a public repository. When the Cline agent processed these files as part of a coding or review task, the injected instructions redirected agent behavior to exfiltrate repository secrets and introduce malicious commits. The attack required no vulnerability in Cline's execution environment; it exploited the same fundamental property that makes RAG and context-aware agents useful — they read and process repository files as trusted instructions.

The implications for CI/CD pipelines are direct and serious. AI agents increasingly participate in code review, automated testing, dependency management, and deployment workflows. Each integration point where an agent reads repository content is a potential ingestion point for poisoned instructions. An adversary who can introduce a single commit — through a compromised developer account, a malicious pull request, or a vulnerable dependency — can potentially redirect the behavior of every AI agent that subsequently processes that repository. The supply chain implications compound: if an agent produces code, documentation, or configuration that is committed back to the repository, the poisoned output becomes part of the repository's history and may influence future model fine-tuning runs that use the repository as training data.

Cross-Cutting Attack Surface Analysis

The Convergence of Three Vectors

The three threat vectors described in this paper share structural properties that make them mutually reinforcing. MCP tool poisoning, agent skill supply chain attacks, and repository poisoning all exploit the same fundamental characteristic of AI developer toolchains: the boundary between trusted instructions and untrusted data has dissolved.

Code/data confusion is not new to software security — SQL injection, cross-site scripting, macro viruses, and XML External Entity attacks have exploited this boundary for decades. What AI developer toolchains introduce is a qualitatively different extension of this problem: natural language is simultaneously data and instruction, and defenses designed to address syntactic injection techniques do not transfer to semantic manipulation. A malicious SQL injection payload is detectable by pattern matching; a malicious natural language instruction is designed to be contextually appropriate and semantically coherent, making it resistant to signature-based detection at any layer of the stack. Agent context is simultaneously the product of past actions and the governance layer for future ones.

This dissolution means that the attack surface does not have a clean perimeter. A malicious MCP tool description can inject instructions that cause an agent to modify a repository. A poisoned repository file can configure an MCP server with malicious properties. A compromised skill can redirect an agent to a rogue MCP server. The attack paths are not linear; they form cycles in which compromise at any one layer can propagate to all others. Security teams accustomed to thinking about software supply chain attacks as unidirectional — from compromised upstream component to affected downstream consumers — must adapt their models to account for bidirectional and circular propagation.

Attacker Objectives and Impact Patterns

Developer toolchain attacks serve attacker objectives that differ meaningfully from conventional endpoint or network compromise. The table below summarizes the principal objective categories and the toolchain vectors most associated with each.

Attacker Objective	Primary Vector(s)	Representative Incident
Credential theft (API keys, cloud tokens)	MCP auto-execution, repository config poisoning	CVE-2026-21852 (Claude Code API key exfiltration)

Attacker Objective	Primary Vector(s)	Representative Incident
Malware delivery to developer endpoints	Agent skill supply chain	ClawHavoc / AMOS stealer campaign
Code integrity compromise	Repository poisoning, RAG context injection	Clinejection (GitHub Actions prompt injection)
Persistent access via model behavior	Training data poisoning	Theoretical; no public cases confirmed as of writing
Reconnaissance and IP exfiltration	MCP tool poisoning, SKILL.md invisible Unicode	Embrace the Red Claude Code demonstration

The Visibility Problem

A characteristic that complicates defense across all three vectors is the absence of conventional indicators of compromise. Malicious SKILL.md content is syntactically valid natural language. Poisoned MCP tool descriptions are indistinguishable from benign ones without semantic analysis against intended behavior. Repository configuration files that trigger malicious hooks look identical to legitimate CI/CD configurations at rest. Training data poisoning leaves no artifact in any repository; its effects manifest as subtle model behaviors that may not surface until the affected model has been deployed to a large user base.

This visibility problem is not merely an implementation gap that better tooling will eventually close. It reflects a fundamental property of natural language as an instruction medium: human reviewers and signature-based security tools are not well-positioned to evaluate whether a natural language string is adversarial, because adversarial natural language strings are designed to be semantically coherent and contextually appropriate. Detection approaches that have proven effective in traditional supply chain security — hash-based integrity verification, vulnerability scanning, dependency auditing — address a different threat model and must be augmented with new capabilities specifically designed for the AI toolchain context.

Organizational and Governance Implications

The Enterprise Security Team's Blind Spot

Enterprise security programs have invested substantially in software supply chain security over the last five years, driven by incidents such as the SolarWinds and XZ Utils compromises. The controls that resulted from this investment — software bill of materials (SBOM) generation, dependency audit tooling, signed commits, and build provenance attestation — were designed for a software supply chain whose fundamental unit of composition is a versioned code artifact. They do not generalize well to the AI developer toolchain, whose fundamental unit of composition is a natural language context that may be assembled at runtime from dozens of sources, none of which has a canonical hash or a cryptographic signature.

The practical consequence is that many enterprise security teams have a significant blind spot at precisely the layer where these attacks are occurring. Organizations that have implemented rigorous software supply chain controls for their production applications may have no comparable controls for the AI tools their development teams use daily. The developers who build and maintain those production applications are themselves the attack surface.

Developer Awareness and Culture

The ClawHavoc campaign and the SKILL.md attack class both relied in part on the implicit trust that developers extend to their AI agents. Developers who would scrutinize an unfamiliar shell script before executing it may not apply the same skepticism to an agent presenting a password prompt, because the agent is a trusted tool and the interaction is framed as a setup or configuration step rather than a security-relevant action. Addressing this gap requires not only technical controls but a cultural shift in how development teams think about AI tool security — specifically, the recognition that prompts, instructions, and configuration files processed by AI agents carry the same security implications as executable code.

Security awareness programs that stop at "don't click phishing links" are insufficient for the toolchain threat model. Developers need to understand the concept of prompt injection, the mechanics of skill supply chain attacks, and the specific contexts in which their AI tools process external content. As of this writing, no major security awareness training framework has incorporated prompt injection or skill supply chain concepts as standard curriculum content, leaving most existing programs without coverage of this threat model. This is a novel educational requirement that organizations building developer security culture will need to address directly.

Conclusions and Recommendations

For Enterprise Security Teams

Organizations deploying AI developer tools should begin by establishing an inventory of all MCP servers, agent frameworks, and skill registries accessible to their development teams. In the absence of formal AI asset inventory programs — which few organizations have yet established — security teams may not know which MCP integrations their developers have installed, and shadow MCP servers — private or community deployments that connect to internal systems — represent a significant unmanaged risk consistent with the OWASP MCP09:2025 category. Inventory is the prerequisite for any other control.

Once inventory exists, organizations should apply least-privilege principles to MCP server configurations, explicitly blocking tool access to file systems, internal networks, credentials, and other sensitive resources that are not required for a given integration's declared purpose. The NSA advisory recommends implementing just-in-time credentials for high-impact MCP actions — a recommendation that maps directly to the Zero Trust principle of continuous verification and the AICM control domain for access governance [2]. Organizations should also deploy change detection for installed MCP tool definitions, alerting when tool descriptions or metadata change post-installation to detect rug pull attacks before they affect users.

For agent skill adoption, organizations should establish internal skill registries with vetting requirements rather than permitting direct installation from public registries, mirroring the approach that mature organizations take with npm or pip. Skills submitted to internal registries should be reviewed for Unicode Tag characters and other invisible content encoding techniques, in addition to conventional review for malicious logic. Organizations operating on macOS should be aware that the AMOS payload delivered by ClawHavoc specifically targets macOS developer environments; platform diversity and hardened endpoint configurations are relevant mitigating factors.

For AI Platform Operators and Developers

MCP server implementers should treat all protocol-supplied parameters as untrusted input and apply input validation against well-defined schemas before any command execution. Authentication and authorization should be treated as mandatory features rather than optional configurations; the NSA advisory recommends explicit authentication mechanisms for all MCP tool invocations [2]. Servers that expose high-impact capabilities — file system access, network egress, credential management — should implement approval workflows that present users with specific, accurate descriptions of the action being requested rather than generic permission dialogs.

Agent framework developers should implement Unicode normalization and filtering on all content ingested as skill or context files, specifically stripping Unicode Tag characters (U+E0000–U+E007F) that have been used to embed invisible adversarial instructions. Integrity verification for installed skill and tool configurations — whether through cryptographic hashing, signed manifests, or trusted timestamping — would provide detection capability for rug pull attacks. Public skill registry operators should consider mandatory security scanning, publisher identity verification, and automated behavioral analysis for submissions.

For Standards Bodies and Policymakers

The OWASP MCP Top 10 provides an initial risk taxonomy that the broader security community should build upon. The framework benefits from broader empirical grounding — specifically, systematic data on the prevalence of each risk category across deployed MCP implementations. CSA's AI Controls Matrix (AICM) and the Agentic Trust Framework (ATF) provide complementary governance and control frameworks that should be extended to explicitly address the MCP and skill supply chain threat classes described in this paper.

The absence of a trusted MCP server registry — analogous to certificate transparency logs for TLS — represents a structural gap in the protocol's security architecture. The ETDI work cited in recent academic literature [18] proposes enhanced tool definitions incorporating OAuth-based signing and policy-based access control — a technical direction worth consideration by standards bodies working on protocol-level tool identity and rug pull risks.

CSA Resource Alignment

This paper's findings connect directly to several CSA frameworks and research programs that provide actionable governance and control guidance for organizations seeking to manage the AI developer toolchain attack surface.

MAESTRO (Agentic AI Threat Modeling) provides the threat modeling framework most relevant to the attack classes described here. MAESTRO's taxonomy of agentic AI threats explicitly addresses prompt injection, context manipulation, and tool misuse — the categories into which MCP auto-execution vulnerabilities, skill supply chain attacks, and repository poisoning all fall. Security teams using MAESTRO to assess their agentic AI deployments should expand their threat model scope to include the developer toolchain components described in this paper.

AI Controls Matrix (AICM) maps control requirements to five provider roles: AI Customer, Application Provider, Cloud Service Provider, Model Provider, and Orchestrated Service Provider. The threats described in this paper span all five roles, with particularly relevant controls in the supply chain security, access governance, and configuration management domains. Organizations adopting AICM should evaluate whether their control implementations address skill registries and MCP servers as supply chain components requiring the same rigor as software dependencies.

Agentic Trust Framework (ATF), stewarded by CSAI Foundation (CSA), provides a Zero Trust governance model for autonomous AI agents built on four maturity levels — Intern, Junior, Senior, and Principal — that govern the degree of autonomous action an agent is permitted to take [19]. The ATF's principle that autonomy is earned rather than granted by default is directly applicable to MCP tool invocation: tools and skills that request high-privilege actions should require higher trust maturity levels and more explicit user confirmation. ATF's compliance crosswalks to NIST SP 800-207, ISO/IEC 42001, and OWASP Agentic Top 10 provide integration pathways for organizations with existing governance programs.

STAR for AI (Security Trust Assurance and Risk) provides a mechanism for organizations to assess and disclose their AI security posture. Skill registry operators, MCP server publishers, and agent framework developers are appropriate subjects for STAR for AI Level 1 self-assessments, which would require them to document their approaches to supply chain vetting, input validation, and vulnerability disclosure. STAR for AI Level 2 certifications by qualified auditors would provide third-party assurance for enterprises making procurement decisions about AI developer toolchain components.

CSA AI Safety Initiative Research Notes on Agent Context Poisoning [12] and Hidden Unicode Instruction Injection [13] provide companion technical references for the SKILL.md attack class described in this paper, with specific detection indicators and mitigation guidance.

References

[1] OX Security. "Architectural Vulnerability in Anthropic's MCP SDKs Enables Remote Code Execution." Tom's Hardware, 2026.

[2] National Security Agency. "Model Context Protocol (MCP): Security Design Considerations for AI-Drive n Automation." NSA Cybersecurity Information Sheet, May 2026.

[3] Australian Signals Directorate, CISA, NSA, CCCS, NCSC-NZ, NCSC-UK. "Careful Adoption of Agentic AI Services." Five Eyes Joint Guidance, May 2026. (Link resolves to NSA press release announcing the joint advisory.)

[4] OWASP. "MCP Top 10: Critical Security Risks in Model Context Protocol Deployments." OWASP Foundation, 2025.

[5] Check Point Research. "Caught in the Hook: RCE and API Token Exfiltration Through Claude Code Proje ct Files." CVE-2025-59536, CVE-2026-21852, February 2026.

[6] Practical DevSecOps. "MCP Security Statistics 2026: CVEs, Vulnerabilities & Breach Data." Practical DevSecOps, 2026.

[7] Invariant Labs. "MCP Security Notification: Tool Poisoning Attacks." Invariant Labs, 2025.

[8] Elastic Security Labs. "MCP Tools: Attack Vectors and Defense Recommendations for Autonomous Age nts." Elastic, 2025.

[9] Snyk. "How 'Clinejection' Turned an AI Bot into a Supply Chain Attack." Snyk, February 2026. (References a separate ToxicSkills audit of 3,984 ClawHub skills.)

[10] PolySwarm. "The ClawHavoc Campaign." PolySwarm Threat Intelligence, 2026.

[11] CyberPress. "ClawHavoc Poisons OpenClaw's ClawHub With 1,184 Malicious Skills." CyberPress, February 2026.

[12] Cloud Security Alliance AI Safety Initiative. "Agent Context Poisoning: SKILL.md and the New AI Supply Chain Attack Surface." CSA Labs, May 6, 2026.

[13] Cloud Security Alliance AI Safety Initiative. "Hidden Unicode Instruction Injection in AI Agent Skills: Invisi ble Adversarial Payloads in Tool Descriptions, Skill Files, and MCP Servers." CSA Labs, 2026.

[14] Arxiv. "Formal Analysis and Supply Chain Security for Agentic AI Skills." arXiv:2603.00195, 2026.

[15] OWASP. "LLM04:2025 Data and Model Poisoning." OWASP Gen AI Security Project, 2025.

[16] SQ Magazine. "LLM Data Poisoning Statistics 2026: Critical Facts You Must Know." SQ Magazine, 2026.

[17] eSecurity Planet. "Hundreds of Malicious Skills Found in OpenClaw's ClawHub." eSecurity Planet, February 2026.

[18] Arxiv. "ETDI: Mitigating Tool Squatting and Rug Pull Attacks in Model Context Protocol (MCP) by using OAuth-Enhanced Tool Definitions and Policy-Based Access Control." arXiv:2506.01333, 2026.

[19] Josh Woodruff / MassiveScale.AI; CSAI Foundation (CSA). "Agentic Trust Framework (ATF) v0.9.1." April 2026. CC BY 4.0.