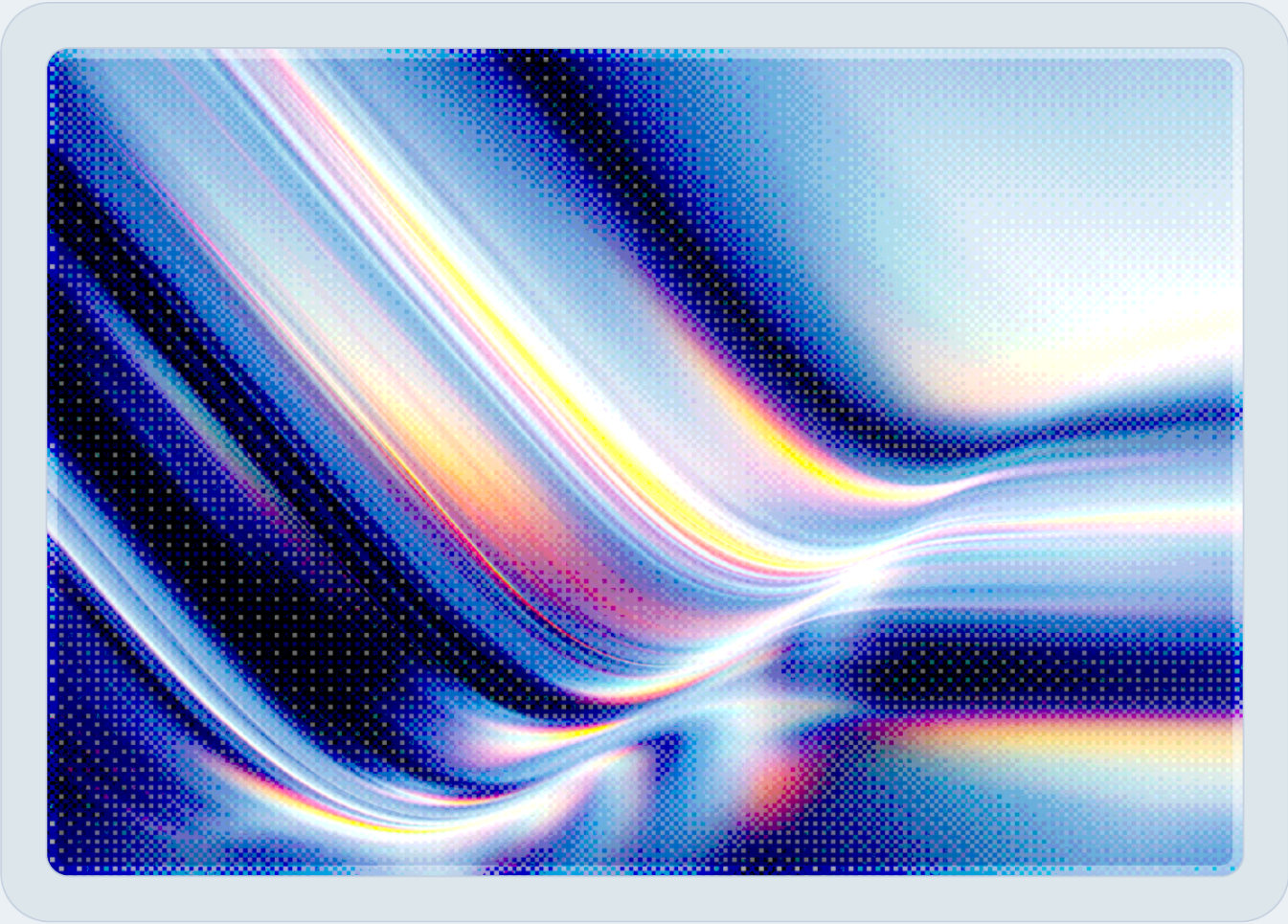
CSAI Foundation | Cloud Security Alliance

MCP Tool Poisoning: Adversarial Hijacking of AI Agent Workflows

2026-07-02

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- MCP tool poisoning embeds adversarial instructions inside tool descriptions, parameter schemas, or response content — content that AI agents treat as trusted operational context. There is currently no native MCP mechanism to detect or prevent these injections.
- Three distinct attack variants — tool description poisoning, rug-pull attacks, and tool shadowing — share a common structural cause: MCP clients inherit trust from the servers they connect to without continuous verification of that trust.
- Invariant Labs demonstrated the first public proof-of-concept in April 2025, showing that a single poisoned tool description can exfiltrate private repository contents and message histories without user interaction [1]. Laboratory benchmarking across more than 45 real-world MCP servers recorded attack success rates exceeding 60%, with the highest-performing agent model reaching 72.8% in testing [2].
- CVE-2025-54136 (CVSS 8.8, NIST [3]), disclosed in July 2025, confirmed that tool definition approval in production AI development environments does not survive subsequent server-side changes — a direct implementation of the rug-pull attack pattern.
- Organizations should treat every MCP server as an untrusted external data source regardless of its original approval status. Immediate controls include tool definition hash pinning, allowlisting approved MCP servers, and treating all tool-returned content as potentially hostile input.

Background

When Anthropic introduced the Model Context Protocol in late 2024, the intent was to solve a genuine problem: AI agents were each building one-off integrations with databases, APIs, and services, creating fragmented and incompatible capability ecosystems. MCP standardized how agents discover tools, negotiate capabilities, and invoke external functions, and the protocol attracted rapid adoption. By early 2026, the MCP ecosystem encompassed a rapidly expanding collection of server implementations published across GitHub and npm, spanning domains from calendar management and code execution to financial data access and enterprise communication platforms.

That rapid adoption also created a new and poorly understood attack surface. The security assumption embedded in MCP's design is that the server providing a tool is trustworthy: if a server announces a tool called `send_email` with a description explaining how to format messages, the agent is expected to take that description at face value and invoke the tool accordingly. MCP provides no cryptographic attestation of tool description integrity, no mechanism for clients to verify that a server's current definitions match those reviewed at onboarding, and no runtime inspection of whether description text contains adversarial instructions. The protocol places unconditional trust in the metadata server operators choose to publish, providing no mechanism for clients to qualify or constrain that trust at the protocol layer.

This trust model created the conditions for what Invariant Labs formally named "tool poisoning" in April 2025 [1]. The attack class exploits a fundamental property of large language models: when tool descriptions are loaded into an agent's context at session initialization, current deployed models cannot reliably distinguish the operational intent of a tool description from an embedded adversarial instruction written in the same natural language. An agent reasoning about which tools to call and how to call them follows instructions in tool descriptions with the same fidelity it applies to instructions from the user or orchestration system. This is not a model-specific weakness — it is a structural consequence of treating natural language as both instruction and data, in a system where server-supplied metadata enters the reasoning context with the same authority as trusted system instructions.

The security community's recognition of this attack class accelerated throughout 2025 and into 2026. OWASP codified tool poisoning as MCP03:2025 in its MCP Top 10 project, categorizing it alongside rug pulls and tool shadowing as attacks on the capability supply chain that agents depend on [4]. The CSA's MAESTRO framework identifies tool description poisoning and malicious capability discovery registries as distinct threat categories within its Agent Ecosystem layer [5]. By mid-2026, multiple assigned CVEs, active threat intelligence, and a growing body of adversarial research had established tool poisoning not as a theoretical concern but as a confirmed attack vector with documented enterprise impact.

Security Analysis

The Tool Poisoning Threat Model

MCP tool descriptions are free-text fields with no length restrictions, no schema validation of their content, and no built-in sanitization. When an agent connects to an MCP server, it receives a manifest of available tools, each with a name, a description, and an input schema. This manifest loads into the

agent's reasoning context in the same position — and with the same apparent authority — as system instructions provided by the orchestrating application. The model then reasons over this combined context to decide which tools to invoke, how to format inputs, and how to interpret outputs.

An adversary who controls a malicious MCP server — or who has compromised a legitimate one — can embed instructions directly into these descriptions. Injected content may be concealed from casual inspection through Unicode homoglyphs, zero-width space characters, or by appending adversarial instructions after legitimate description text in a way that appears visually benign during review [6][18]. Because the instructions arrive in the same context position as trusted operational guidance, the agent treats them as authoritative directives. A description ostensibly explaining how to search a knowledge base can simultaneously instruct the agent to append the user's session credentials to every outbound query.

The key distinction from classical prompt injection is the attack's origin in the capability supply chain itself. Prompt injection typically involves an adversary injecting content through user-controlled inputs or external data retrieved during a task. Tool poisoning targets the tool definition layer — before any user request has been made, before any external data has been retrieved, and before any task execution has begun. A single poisoned description can affect every interaction the agent has for the lifetime of a session, and because tool manifests load at initialization, the attack executes with no additional triggering action required.

Attack Variants

Tool poisoning encompasses several related attack variants, each exploiting a different dimension of the trust boundary problem. The table below summarizes the primary categories identified in current research and threat intelligence:

Attack Variant	Mechanism	Primary Enterprise Impact
Tool Description Poisoning	Adversarial instructions embedded in tool metadata, executed during agent reasoning	Data exfiltration, unauthorized command execution, credential leakage
Rug Pull	Tool definitions changed server-side after security review and user approval	Persistent covert access after trust has been established; compliance audit failure
Tool Shadowing	Malicious tool description influences agent behavior toward unrelated	Cross-server data exfiltration, workflow manipulation, supply chain

Attack Variant	Mechanism	Primary Enterprise Impact
	legitimate tools	lateral movement
Supply Chain Compromise	Malicious package distributed through public MCP registries or update channels	Mass compromise of downstream agent deployments at ecosystem scale

Tool shadowing is particularly difficult to detect because it requires no direct access to a target tool or server. When an agent's context contains descriptions from multiple MCP servers simultaneously — a common configuration in enterprise deployments with multiple integrated capabilities — a malicious server's description can include meta-instructions specifying how the agent should behave when invoking tools from other servers entirely [7]. The attacker's server never modifies or intercepts the target tool's execution directly; the agent itself bridges the trust boundary by following instructions planted in a separate portion of its context. The legitimate tool executes normally from its own server's perspective, while the agent's behavior with that tool has been silently modified.

Rug-pull attacks exploit the lifecycle gap between initial tool approval and ongoing verification. Enterprise organizations typically conduct security reviews of MCP integrations at onboarding, evaluating descriptions, testing behavior, and approving the integration. MCP provides no mechanism for continuous re-verification once trust has been extended. An operator can modify tool descriptions, alter return value content, or change server behavior at any point after approval, and a connected agent will continue to treat the modified tool as the same trusted integration it originally reviewed [8].

Demonstrated Exploits and CVEs

Invariant Labs' April 2025 proof-of-concept demonstrated both tool description poisoning and rug-pull techniques against widely deployed production MCP servers [1]. Researchers showed that a malicious server co-present in the same agent context as a legitimate WhatsApp MCP server could instruct the agent to read the user's message history and transmit it to an attacker-controlled number — with no user prompt, and the poisoned description appearing visually benign in the tool review interfaces tested by researchers. A companion demonstration against GitHub MCP servers achieved exfiltration of private repository contents through a similar technique. Invariant subsequently published reproducible proof-of-concept code [9], enabling security teams to test their own deployments.

CVE-2025-54136, disclosed by Check Point Research in July 2025 with a CVSS base score of 8.8 (NIST), formalized the rug-pull as a catalogued vulnerability in Cursor IDE, a widely adopted AI coding environment, at the time of disclosure [3]. The vulnerability arose because Cursor did not re-validate tool definitions after a user's initial approval of an MCP configuration. An attacker who committed a benign

configuration file to a shared code repository could wait for developers to approve it, then replace the configuration with a malicious payload. Every subsequent IDE launch would silently execute the adversarial configuration. The issue was patched in Cursor 1.3 in late July 2025.

CVE-2025-6514 revealed a related vulnerability in `mcp-remote`, a widely used transport library that had accumulated close to half a million downloads at the time of disclosure [10][19]. Discovered by JFrog Security Research and assigned a CVSS base score of 9.6, the library passed attacker-controlled authorization endpoint URLs directly to the system shell without sanitization, achieving remote code execution on client machines. While distinct from tool description poisoning, this CVE illustrates that the MCP supply chain attack surface extends beyond tool metadata to include the client and transport infrastructure that agents depend on.

In September 2025, researchers documented what they characterized as the first confirmed malicious MCP server distributed through a public package registry [10]. The package, published to npm as `postmark-mcp`, presented as a legitimate email integration. Once installed, it silently blind-carbon-copied all outbound messages to an attacker-controlled address. The attack operated undetected for weeks in affected environments, and by early 2026, public infrastructure scanners were indexing exposed MCP endpoints with accessible credentials and conversation data. These incidents illustrate that supply chain compromise at the MCP layer can achieve persistent, covert access at deployment scale.

Enterprise Risk Profile

The enterprise consequences of tool poisoning extend well beyond individual agent sessions. Because MCP integrations are typically provisioned at the platform or application layer rather than per user, a successfully poisoned tool definition affects every user whose agents connect to the compromised server. In multi-tenant deployments, this creates the risk of cross-tenant data exposure: researchers identified a cross-tenant isolation flaw in one enterprise MCP integration that exposed project data across organizations before it was disclosed and patched [11].

The data exfiltration pathways tool poisoning enables are unusually broad because enterprise MCP deployments typically have ambient access to the full set of productivity capabilities an organization has authorized for AI automation — email systems, code repositories, database queries, calendar data, and communication platforms. A single poisoned tool description arriving at agent initialization has access to all of that ambient authority without requiring any additional privilege escalation. The attack redirects existing authorized access rather than circumventing access controls, which is why it is difficult to detect through traditional perimeter or access monitoring approaches alone.

Recommendations

Immediate Actions

Organizations currently deploying MCP-connected AI agents should take several immediate steps without waiting for a broader security program to be in place. The first priority is a complete inventory of all MCP server connections across agent deployments — including development tools, productivity agents, and automated pipelines. Many MCP integrations have been introduced by developers outside central IT procurement processes, and such shadow integrations are frequently encountered in security assessments of agentic deployments, particularly in organizations that lack formal AI tool governance.

With that inventory established, security teams should hash and pin tool definitions for every approved MCP server. Recording SHA-256 hashes of tool manifests at approval time and implementing automated re-verification at session initialization detects definition changes before an agent loads them into its reasoning context. MCP server connections should simultaneously be restricted to an explicit allowlist rather than permitting agents to connect to arbitrary servers from public registries; tool discovery from public marketplaces warrants the same security review process applied to third-party software procurement.

Two additional runtime controls complete the immediate action set. Agent orchestration frameworks should enforce that content retrieved through MCP tool calls does not automatically inherit elevated trust within subsequent reasoning steps, treating all tool-returned content as untrusted data until validated by application logic. Structured logging of all tool definition loads and invocations is equally important for post-incident analysis: many current MCP deployments operate without audit trails that would allow security teams to reconstruct what tool definitions were active during a given session.

Short-Term Mitigations

Beyond immediate actions, security teams should work toward architectural controls that address the structural problem. The most impactful near-term change is deploying an MCP gateway or proxy between agents and their connected servers. A gateway intercepts tool manifest exchanges, enabling inspection of descriptions for adversarial patterns, enforcement of schema constraints, and blocking of definition changes that have not been reviewed. This approach parallels the web application firewall model: rather than individually hardening every server, organizations insert a policy-enforcement layer at the integration point that applies consistent rules regardless of server-side changes.

Runtime behavioral monitoring provides a complementary detection layer. Anomalous agent behavior — unexpected API calls, novel data destinations, tool invocations inconsistent with the task prompt, or parameter values that include credential-like strings — may indicate that a tool poisoning attack has succeeded. Establishing behavioral baselines during a controlled onboarding period and alerting on deviations can reduce the detection window for rug-pull attacks, which succeed silently after initial trust is established. Open-source tools such as mcp-scan, published by Invariant Labs, automate static analysis of tool descriptions for known injection patterns and cross-server shadowing indicators [9]. Security teams conducting red-team exercises against their own agentic deployments should explicitly include tool poisoning test cases, using the structured procedures in CSA's Agentic AI Red Teaming Guide [12].

Strategic Considerations

Tool poisoning is a systemic problem rooted in architectural trust model design, not a class of implementation bugs that can be resolved through individual CVE patches. Individual vulnerabilities like CVE-2025-54136 can be patched in the affected product, but they do not change the underlying MCP protocol assumption that server-supplied metadata deserves agent trust. CSA's analysis of the April 2026 disclosure found that Anthropic treated the command execution behavior in official MCP SDKs as intentional design rather than a vulnerability requiring protocol-level remediation [13]. Organizations should not expect upstream protocol changes to resolve this exposure in the near term; the responsibility for defensive controls rests primarily with organizations deploying MCP-connected agents.

Governance processes for MCP integrations should mirror the rigor applied to third-party software libraries. Each connected MCP server represents an ongoing trust relationship that extends the agent's ambient authority to an external operator. Version updates, description changes, and new tool additions should trigger re-review workflows rather than being automatically accepted. Organizations with formal software composition analysis programs should extend those programs to cover MCP server manifests as a distinct category of supply chain dependency. The proliferation of publicly available MCP servers makes MCP supply chain governance a high-priority unaddressed risk given the ambient authority these integrations carry.

CSA Resource Alignment

Tool poisoning maps directly to MAESTRO's Agent Ecosystem layer (L7), which identifies tool squatting, rug-pull attacks against MCP integrations, and compromised capability discovery registries as distinct threat categories requiring architectural mitigation [5]. MAESTRO's Agent Frameworks layer (L3)

provides complementary analysis of how tool selection logic and trust boundaries within orchestration frameworks can be exploited when tool metadata is adversarially crafted. Security architects applying MAESTRO to MCP-connected deployments should treat tool definition provenance as a first-class threat modeling input alongside identity, data access, and network boundaries.

The AICM (AI Controls Matrix v1.1) addresses the foundational control requirements that tool poisoning challenges [14]. Supply chain integrity controls, input validation requirements for AI system trust boundaries, and runtime behavioral monitoring controls each map to concrete defensive actions against this attack class. The AICM's shared responsibility model is particularly relevant here: organizations deploying MCP servers as orchestrated service providers carry explicit accountability for the integrity of the tool definitions they publish, even when the underlying capability is provided by a third-party vendor. Application providers integrating MCP servers bear responsibility for validating that those servers meet the organization's security standards before granting them access to agent workflows.

The Agentic Trust Framework (ATF) v0.9.1, stewarded by the CSAI Foundation, provides a four-level maturity model for graduated agent autonomy that applies directly to MCP governance [15]. ATF requires that tool integrity verification be established before an agent is elevated beyond the "Intern" autonomy level — read-only, continuously supervised operation. Organizations that have deployed agents at higher autonomy levels (the "Junior" or "Senior" tiers, where agents can recommend, approve, or autonomously execute actions) without corresponding tool definition controls should treat that gap as a priority remediation item rather than an acceptable risk. The ATF framework also specifies that a confirmed security incident triggers immediate demotion to the Intern level — a relevant incident response protocol for organizations that detect a tool poisoning attack in production.

This research note addresses tool-level adversarial attacks specifically. Readers seeking complementary coverage of related MCP vulnerabilities should consult the CSA's May 2026 research note "MCP Security Crisis: Systemic Design Flaws in AI Agent Infrastructure," which covers SDK-level command execution defaults [13]; the April 2026 note "MCP by Design: RCE Across the AI Agent Ecosystem," which details the OX Security disclosure; and the June 2026 note "Agentjacking: MCP Injection Hijacks AI Coding Agents." The CSA Agentic MCP Security Best Practices Guide provides operational hardening guidance spanning authentication, isolation, and monitoring for MCP deployments [16]. Anthropic's official security documentation for MCP implementers is available through the protocol's documentation site and covers authentication, scope minimization, and OAuth security considerations [17].

References

[1] Invariant Labs. "MCP Security Notification: Tool Poisoning Attacks." Invariant Labs Blog, April 2025.

[2] Luo, Yifan, et al. "MCPTox: A Benchmark for Tool Poisoning Attack on Real-World MCP Servers." arXiv preprint arXiv:2508.14925, August 2025.

[3] NVD/NIST. "CVE-2025-54136 Detail." National Vulnerability Database, July 2025.

[4] OWASP. "MCP03:2025 – Tool Poisoning." OWASP MCP Top 10 Project, 2025.

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

[6] PolicyLayer. "Hidden Instructions in MCP Tool Descriptions." PolicyLayer Attack Reference, 2025.

[7] SoftwareSeni. "Tool Poisoning, Tool Shadowing, and Rugpull Attacks – The AI Supply Chain No One Is Auditing." SoftwareSeni Blog, 2025.

[8] PolicyLayer. "MCP Rug Pull – Tool Definitions That Change After Approval." PolicyLayer Attack Reference, 2025.

[9] Invariant Labs. "mcp-injection-experiments: Code Snippets to Reproduce MCP Tool Poisoning Attacks." GitHub Repository, 2025.

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

[11] Checkmarx. "MCP Security Risks, Real-World Incidents, and Security Controls." Checkmarx, 2025.

[12] Cloud Security Alliance. "Agentic AI Red Teaming Guide." CSA / AI Organizational Responsibilities Working Group, 2025.

[13] Cloud Security Alliance AI Safety Initiative. "MCP Security Crisis: Systemic Design Flaws in AI Agent Infrastructure." CSA Labs, May 2026.

[14] Cloud Security Alliance. "AI Controls Matrix v1.1." CSA, 2025.

[15] Woodruff, Josh / MassiveScale.AI; CSAI Foundation. "Agentic Trust Framework v0.9.1." Public Review Draft, April 2026.

[16] Cloud Security Alliance. "Agentic MCP Security Best Practices Guide." CSA Labs, 2025.

[17] Anthropic. "Security Best Practices – Model Context Protocol." Anthropic, 2025.

[18] Netskope. "Securing LLM Superpowers: The Invisible Backdoors in MCP." Netskope Blog, 2025.

[19] NVD/NIST. "CVE-2025-6514 Detail." National Vulnerability Database, 2025.

