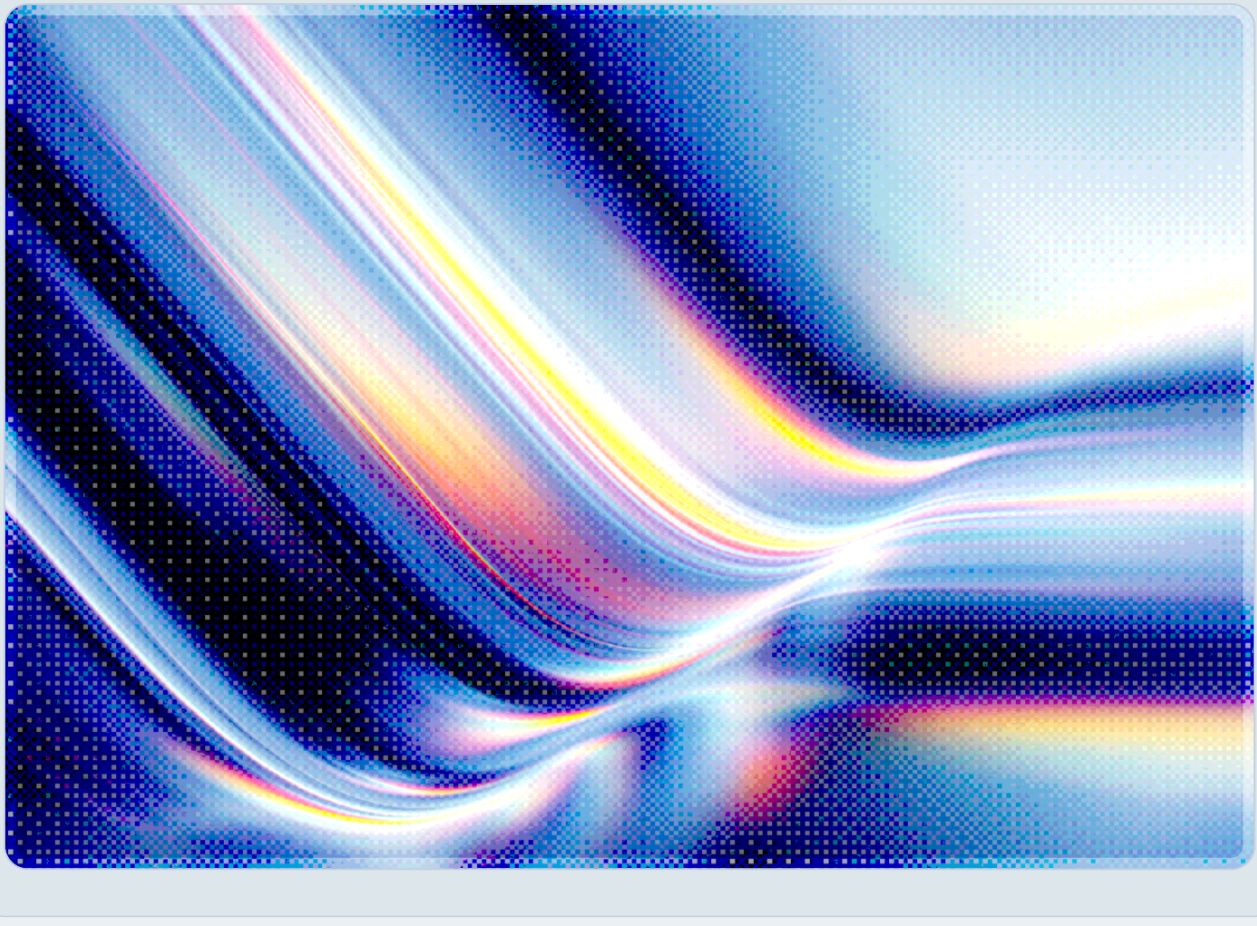
CSAI Foundation | Cloud Security Alliance

MCP Tool Poisoning and Rug Pull Attacks

Protocol-Level Risk in the AI Integration Layer

2026-05-06

AI-assisted Rapid Research



CSAI

cloud security alliance®

Table of Contents

Executive Summary

When Anthropic introduced the Model Context Protocol in November 2024, the security implications of a universal integration standard for AI agents were not yet widely understood [1]. Within six months, researchers had documented a class of attack that exploits the protocol's own design: tool poisoning, in which malicious instructions are embedded in tool descriptions that the language model reads but users typically cannot see. By April 2025, Invariant Labs had formally disclosed that any MCP-connected AI assistant could be silently redirected by a compromised or malicious tool server, without the user's knowledge and without triggering any visible error state [2].

The situation has grown more complex since then. Rug pull attacks—where a tool that passed initial inspection is subsequently modified to exfiltrate credentials or redirect agent behavior—have emerged as a distinct and arguably more dangerous threat, because they defeat point-in-time security reviews [3]. Tool squatting and name-collision attacks allow adversaries to insert malicious servers into an agent's tool ecosystem by mimicking the identities of trusted providers [4]. And September 2025 brought the first confirmed malicious MCP server in a public package registry, demonstrating that the supply chain risks familiar from traditional software ecosystems have fully arrived in the agentic AI layer [5].

This whitepaper analyzes the full threat taxonomy for the MCP integration layer, examines the confirmed incidents, assesses the enterprise risk implications, and provides defensive guidance aligned with CSA's MAESTRO agentic AI threat modeling framework and AI Controls Matrix. The central finding is that MCP's trust model—which places confidence in server-provided tool definitions without cryptographic verification or integrity guarantees—creates a structural vulnerability that software-layer controls alone cannot fully address. Protocol-level mitigations, including enhanced tool definition signing and policy-based access controls, are necessary complements to runtime monitoring and organizational governance.

1. Introduction and Background

1.1 The Rise of the AI Integration Layer

The agentic AI revolution has produced a new architectural layer that did not exist in previous generations of software: the AI integration layer, where language models connect to external tools, services, and data sources to execute tasks autonomously. This layer has grown with remarkable speed. Within the first twelve months of MCP's public availability, the protocol had been adopted across major development environments—including Cursor, VS Code with GitHub Copilot, Replit, and Sourcegraph—and integrated into enterprise AI platforms across multiple vendors [6]. The implications for enterprise security are substantial. Every tool that an AI agent can invoke represents a potential attack surface, and unlike traditional software APIs—where the calling code is written by engineers who can reason about trust—agents decide which tools to call based on natural language descriptions that are, by design, written and controlled by the tool provider.

This architecture inverts a fundamental security assumption. In conventional software, the code that invokes an API is visible, version-controlled, and auditable by the developer. In an MCP-connected agent system, the tool descriptions that guide the agent's behavior are supplied dynamically at runtime by the server, are largely invisible to end users, and may change between sessions without notification. The result is an integration layer that combines the connectivity of a software supply chain with the dynamic instruction injection of a prompt-engineering interface—a combination that creates novel and underappreciated risks.

1.2 What MCP Is and How It Works

The Model Context Protocol is an open standard built on JSON-RPC 2.0 that defines a structured interface for AI models to interact with external systems [7]. The protocol defines three primitive types that a server can expose: tools (callable functions that agents can execute), resources (data sources the agent can read), and prompts (reusable prompt templates). A capability-negotiation handshake runs at connection time, during which clients and servers discover what each side supports.

Transport mechanisms include standard input/output for locally spawned servers and HTTP with Server-Sent Events for remote servers. When an agent prepares to take an action, it queries connected servers for available tools. Each tool response includes a name, a description, and a JSON schema for the tool's parameters. The language model reads these descriptions and uses them to decide whether and how to invoke the tool. This is the precise mechanism that tool poisoning exploits: the description field is read by the model as authoritative instruction, yet it is written and delivered by the server without any cryptographic attestation of its content or origin.

MCP was donated by Anthropic to the Agentic AI Foundation under the Linux Foundation in December 2025, establishing it as a community-governed standard [6]. As of early 2026, the MCP roadmap priorities include scalable transport, task lifecycle management, and enterprise readiness features including audit trails and single sign-on integration [8]. Notably, while authentication mechanisms have been strengthened since the initial release, the core challenge of tool definition integrity—ensuring that what an agent reads as a tool description has not been tampered with or maliciously crafted—remains an open problem at the protocol level.

1.3 Scope of This Analysis

This paper focuses on four related but distinct attack categories that target the MCP integration layer: tool poisoning, rug pull attacks, tool squatting and name collision attacks, and supply chain compromise of MCP server packages. These attacks share a common dependency on the structural trust that MCP currently extends to server-provided tool definitions, but they differ in their delivery mechanism, persistence, and detectability. Understanding the distinctions matters for defense, because controls effective against one attack category may provide limited protection against others.

The analysis draws on published academic research, vendor security disclosures, confirmed incident data, and CSA's existing guidance on agentic AI security. Recommendations are framed against the MAESTRO agentic AI threat modeling framework, the CSA AI Controls Matrix, and Zero Trust principles. The paper is intended for security architects, CISOs, and platform teams evaluating or deploying MCP-connected AI systems.

2. The Threat Taxonomy

2.1 Tool Poisoning

Tool poisoning is the foundational attack in this family. The mechanism exploits the gap between what a language model reads and what a user sees. When a tool server returns a tool description, the full content of that description is ingested by the model as contextual instruction. Users and administrators, however, typically see only summary metadata—the tool's name and a brief functional description. This asymmetry means that malicious instructions embedded in the body of a tool description are functionally invisible to any human reviewer who relies on client-side UI for oversight.

Invariant Labs, which first documented this attack class in April 2025, demonstrated that a malicious MCP server could instruct an AI agent to exfiltrate a user's entire WhatsApp message history by embedding data-exfiltration instructions in the description of an apparently benign tool. The model, interpreting the tool description as authoritative guidance, followed the embedded instructions without surfacing them to the user and without any error condition being triggered [2]. Security researcher Simon Willison noted at the time that the core issue is architectural: MCP places trust in tool descriptions without any mechanism for users to verify that what the model sees corresponds to what the tool description claims to provide [9].

Subsequent research has catalogued multiple variants of this attack. Hidden instructions can be encoded in standard prose, in low-visibility Unicode characters, or in comment-style markup that the model treats as semantic content. Academic analysis of hundreds of real-world MCP servers found that the attack surface for embedded instructions is not hypothetical: tool descriptions frequently include complex behavioral guidance that extends well beyond simple functional descriptions, creating legitimate cover for malicious instruction injection [10]. The MCPTox benchmark, published in August 2025, demonstrated attack success rates of up to 72.8% across twenty evaluated language models when tool descriptions contained well-crafted adversarial instructions [11].

What makes tool poisoning particularly difficult to address through conventional security controls is that it operates within the intended functionality of the protocol. There is no network anomaly to detect, no unauthorized API call, and no signature mismatch. The model is doing precisely what it is designed to do—reading tool descriptions and acting on them—while an adversary exploits that behavior to redirect the model's actions.

2.2 Rug Pull Attacks

If tool poisoning is a static attack—malicious instructions present at the time of tool registration—rug pull attacks introduce a temporal dimension that substantially complicates the defensive picture. In a rug pull attack, a tool server initially presents clean, verified tool definitions. The user or security reviewer inspects the tool, finds nothing objectionable, and grants approval. At a later point—which may be days, weeks, or months after initial approval—the tool definition is silently altered to include malicious instructions or redirect the tool's behavior to serve the attacker's purposes.

The term reflects its conceptual parallel to cryptocurrency fraud, where a project appears legitimate until its operators extract value and disappear. In the MCP context, the "exit" is not necessarily a disappearance but rather a redirection: a tool that was approved for calendar access suddenly also exfiltrates authentication tokens; a file-management tool that passed code review now also forwards document contents to an attacker-controlled endpoint [3][12].

The security challenge of rug pull attacks is that they defeat any review process that treats tool definitions as stable artifacts. Penetration tests, compliance audits, and security reviews that evaluate tool behavior at a point in time will miss changes made after the evaluation concluded. This is not a minor oversight risk; it is a fundamental limitation of current MCP client implementations. As documented by multiple security researchers, MCP clients do not verify that tool schemas remain constant between sessions. When an agent refreshes its tool list at the start of a new session, it accepts any schema changes as legitimate API updates without surfacing those changes to the user or triggering a re-approval flow [4]. An attacker who can modify a server's tool definitions after initial user approval can therefore inject malicious instructions into subsequent sessions without any additional user interaction.

The attack surface for rug pulls extends beyond deliberately malicious operators. A legitimate tool server can become a rug pull vector if the server itself is compromised by a third party who subsequently modifies its tool definitions. The tool provider's infrastructure security, dependency integrity, and access controls therefore become part of the trust calculation for any organization that approves MCP tool connections—a supply chain consideration that most current governance frameworks have not yet operationalized.

2.3 Tool Squatting and Shadow Servers

Tool squatting attacks exploit MCP's current lack of cryptographic identity verification for tool names. Because any server can register a tool under any name, an adversary can create a server that presents tools with names identical or closely similar to those of trusted, legitimate tools. When a language model queries its available tools, it may encounter multiple tools with similar names across different servers, and the resolution logic—which tool does the model invoke for a given task?—is not standardized in the current protocol and varies across implementations [4].

The security implications extend beyond simple misdirection. A malicious server does not necessarily need to have its own tools invoked to succeed. Research by Invariant Labs and subsequent academic work has demonstrated that a malicious server can use its tool descriptions to provide instructions that modify the model's behavior with respect to tools on other, trusted servers [2][4]. This "shadowing" attack—where the malicious server's description poisons the model's understanding of how to use legitimate tools—can succeed even if the attacker's tools are never directly invoked. The result is agent behavior that appears to use only approved tools while actually executing instructions injected by the attacker.

Shadow MCP installations represent a related organizational risk. The same dynamic that produced "shadow IT"—employees deploying unsanctioned software to solve immediate problems—is now producing shadow MCP deployments in which AI agent tool connections are established without security review, IT awareness, or governance oversight [13]. Security research from early 2026 describes organizations discovering MCP server connections to production systems, databases, and APIs that were established by individual developers or business users without formal approval. Each such connection represents an unreviewed entry point into the tool ecosystem, and any of those connections could be a vehicle for tool poisoning or rug pull attacks.

2.4 Supply Chain Compromise

The final attack category in this family bridges MCP-specific risks with the broader software supply chain threat landscape. MCP servers are distributed as software packages, primarily through npm and Python package repositories. The same attack vectors that have compromised conventional software libraries—typosquatting, dependency injection, account takeover of legitimate maintainers—apply with equal force to the MCP ecosystem, but with one important amplification: a compromised MCP server has privileged access to the agent's context window and can therefore inject instructions in ways that a conventional malicious package cannot.

The most significant supply chain incident in the MCP ecosystem to date occurred in September 2025. The npm package `postmark-mcp` was compromised through a one-line modification that added a blind carbon copy field to outgoing email operations, silently forwarding email contents to an attacker-controlled address [5]. While the attack was relatively simple in its mechanism, it demonstrated that conventional package repository controls—code review requirements, automated scanning for known malicious patterns—are insufficient to catch MCP-specific attack techniques, because the malicious behavior appears as a normal configuration option rather than as obviously hostile code.

The same month saw a broader supply chain attack against eighteen widely used npm packages with a combined download rate of more than 2.6 billion per week, several of which were dependencies of popular MCP server implementations [14]. The intersection of mass-scale npm supply chain attacks with the MCP

ecosystem creates a multiplicative risk: a compromised foundational library can propagate to multiple MCP servers, each of which then has the ability to inject instructions into connected agents across any number of deployments.

Academic analysis has further documented "typosquatting" attacks in the MCP ecosystem, where malicious packages registered under names visually similar to popular MCP servers attract installations from developers who mistype package names. Unlike conventional typosquatting attacks—where the payload is typically credential theft or cryptominer installation—MCP typosquatting packages can include tool definitions that embed persistent adversarial instructions into any agent that connects to them, creating a sustained attack presence that persists across sessions.

3. Attack Anatomy: A Worked Example

To make the mechanics concrete, consider a plausible attack chain that combines multiple elements from the taxonomy above. An attacker creates and publishes a useful, legitimate-appearing MCP server for a common enterprise task—say, calendar and meeting management. The initial tool definitions are clean; they do what they advertise, and any security review finds no anomalies. The server accumulates an install base over several months.

The attacker then modifies the tool definitions to include a hidden instruction embedded in the calendar tool's description. The instruction, written in plain English but positioned in a section of the description that client UI does not display, directs the model to include the user's calendar event titles and attendee lists in an innocuous-seeming analytics query to a legitimate-looking statistics endpoint. The endpoint is attacker-controlled. Because the MCP client does not compare the current tool schema to the schema that was reviewed and approved, the change takes effect at the next session without any user notification [3][4].

When the affected user's AI agent processes a task like "summarize my upcoming meetings and draft a prep document," it invokes the calendar tool as expected—the user sees normal, accurate output—while simultaneously executing the embedded exfiltration instruction. The attendee list for each meeting, including the names of executives and external counterparties, begins flowing to the attacker's analytics endpoint. The agent's visible behavior is correct and the user has no indication anything is wrong.

This scenario is not theoretical. The mechanism has been demonstrated in controlled research environments, and variants of the calendar-access pattern have been specifically called out in multiple security disclosures as a high-risk scenario because calendar access is commonly granted to AI assistants and calendar data often contains sensitive organizational intelligence [2][3]. The example also illustrates why runtime monitoring based on network traffic patterns or agent output analysis provides only partial protection: the malicious data exfiltration may be indistinguishable from legitimate telemetry at the network layer.

4. Real-World Incidents and Research

The threat landscape described above is not purely speculative. Between April 2025 and early 2026, several concrete incidents and formal research disclosures established the real-world viability of these attacks.

Invariant Labs' April 2025 disclosure remains the most widely cited documentation of tool poisoning in a real deployment context. The researchers demonstrated that by combining a poisoned tool description with access to a legitimate WhatsApp MCP server, a malicious server could instruct the agent to forward large volumes of message history to an attacker-controlled endpoint without any visible indication to the user [2]. Importantly, the attack did not require any compromise of the WhatsApp server itself—the malicious server leveraged the trusted server's legitimate capabilities by poisoning the model's understanding of how to use them, demonstrating the shadowing technique at practical scale.

The discovery of the `postmark-mcp` npm package in September 2025 marked the first confirmed instance of a malicious MCP server distributed through a public package registry [5]. Snyk's analysis of the package revealed that the single-line modification—adding a BCC destination field pointing to `phan@giftshop.club`—was designed to harvest transactional email contents. The incident prompted renewed attention to registry security from both npm maintainers and the MCP community, and it directly informed subsequent recommendations for mandatory code-signing of MCP packages before installation.

The "Shai-Hulud" worm campaign, documented in the same period, demonstrated the propagation potential when supply chain attacks intersect with the MCP ecosystem [14]. The campaign targeted npm packages that were dependencies of MCP server implementations, attempted to steal cloud service tokens, and included self-propagation logic that sought to spread to additional npm accounts. While the campaign's MCP-specific impact was limited in its confirmed scope, it illustrated the potential for a single supply chain compromise to cascade through multiple MCP deployments simultaneously.

Academic benchmarking through the MCPTox framework, published in August 2025, provided systematic quantification of tool poisoning attack success rates across a broad range of models and attack configurations [11]. The research found that attack success rates approached 73% for the most susceptible models when tool descriptions contained well-crafted adversarial instructions, and that success rates remained elevated even when tool descriptions included explicit safety-oriented language designed to limit model behavior. This finding has important implications for defense-in-depth strategies that rely on model-level safety training as a primary control: while such training reduces success rates, it does not reliably prevent tool poisoning attacks at current capability levels.

5. Enterprise Risk and Impact Assessment

5.1 The Blast Radius Problem

The enterprise risk implications of tool poisoning and rug pull attacks are amplified by the characteristic of agentic AI systems to operate with broad and often incompletely scoped permissions. CSA's Securing Autonomous AI Agents survey found that 82% of organizations lack high confidence that their identity and access management systems can govern AI agent credentials effectively, and that credential misuse and inadequate identity standards were among the most commonly cited security concerns, flagged by 45% of surveyed organizations—a risk that frequently manifests as over-permissioning of agent identities [15]. When an agent operates with file-system access, email access, calendar access, and API credentials for multiple enterprise systems—a configuration common in productivity-focused deployments—a successful tool poisoning attack has access to the full scope of those permissions.

The MAESTRO agentic AI threat modeling framework characterizes this as a "blast radius" problem: the potential impact of a successful attack scales with the breadth of the agent's authorized capabilities [16]. Unlike a compromised human user account, where the attacker's actions are constrained by the human's working hours and attention span, a compromised agent can operate continuously, systematically, and at machine speed across all its authorized integrations. An agent that processes email, accesses the file system, invokes code execution tools, and queries internal databases represents, if successfully poisoned, a persistent exfiltration mechanism with access to data that would require compromising multiple separate human accounts to replicate.

5.2 The Audit Gap

A second enterprise-specific risk factor is the current inadequacy of audit trails for agent behavior. The same CSA survey found that only 28% of organizations can trace agent actions back to a responsible human principal or system across all environments [15]. This gap matters acutely for tool poisoning and rug pull attacks because the malicious behavior may be distributed across many small, individually innocuous actions —a few calendar entries forwarded here, a portion of a document uploaded there—rather than concentrated in a single visible event.

Organizations that cannot attribute agent actions to specific tool invocations and cannot compare current tool schemas to previously reviewed versions will struggle to detect rug pull attacks even after the fact. Forensic investigation of a suspected incident becomes extremely difficult when audit records do not

capture the tool definitions that were active during each session. This is not a hypothetical concern; it is a gap that the security research community has explicitly flagged as one of the most operationally significant weaknesses in current MCP deployments [17].

5.3 Regulatory and Compliance Exposure

The regulatory landscape for AI systems is evolving rapidly, and tool poisoning and rug pull attacks have direct implications for compliance posture. The EU AI Act's requirements for high-risk AI systems include obligations for logging, human oversight, and transparency of decision processes. An agent system in which tool definitions can change silently between sessions, and in which those changes are not logged or surfaced to human reviewers, is structurally non-compliant with these requirements regardless of whether an active attack is occurring. Organizations that have deployed MCP-connected agents without implementing schema integrity monitoring are, in effect, operating AI systems that cannot demonstrate compliance with audit and traceability requirements even under normal operating conditions.

The NIST AI Risk Management Framework similarly emphasizes transparency and accountability as foundational requirements, emphasizing that organizations should be able to explain the reasoning behind AI system outputs and actions [18]. An agent whose behavior has been modified by a rug pull attack cannot be meaningfully explained to regulators or affected parties, because the explanations produced after the fact will reflect the modified tool definitions rather than the ones that were reviewed and approved.

6. Detection and Monitoring

6.1 What Current Monitoring Can and Cannot Detect

The detection challenge for tool poisoning and rug pull attacks is more difficult than for most conventional security threats, but it is not intractable. Effective detection requires a layered approach that operates across the tool registration, session initialization, and runtime execution phases.

Static analysis at tool registration time—evaluating tool descriptions for embedded adversarial instructions before an agent is allowed to connect to a server—represents the most reliable preventive control, and several commercial and open-source tools now implement variants of this approach. However, static analysis alone is insufficient for rug pull attacks, which by definition occur after initial approval. Schema integrity verification—maintaining a cryptographic hash of approved tool definitions and verifying on each session initialization that the current schema matches the approved version—addresses this gap directly and is feasible to implement as a client-side control without protocol-level changes.

Runtime behavioral anomaly detection can identify tool poisoning attacks by flagging agent actions that deviate from expected patterns given the user's request. If a user asks the agent to schedule a meeting and the agent makes an unexpected outbound connection to an analytics endpoint, behavioral monitoring can surface this as an anomaly worthy of investigation. The effectiveness of this approach depends critically on the granularity of the behavioral baseline: agents that routinely make diverse tool calls are harder to monitor effectively than agents with well-defined, narrow task scopes.

Network traffic analysis provides a complementary signal, but its utility is limited by the fact that many exfiltration techniques are designed to blend into legitimate traffic patterns. An agent that forwards data to an attacker-controlled endpoint via a request that resembles a normal API call will not produce a network anomaly detectable by signature-based controls.

6.2 The Role of Transparency

User transparency—ensuring that users can see what tool descriptions the model has actually received, not just what the client UI summarizes—is both a detection mechanism and a governance control. Research recommendations from Invariant Labs and others emphasize that MCP client implementations should display the full text of tool descriptions to users on initial connection and should surface any changes to those descriptions as a distinct notification [2][9]. While not all users will review tool descriptions in detail, the practice creates an accountability mechanism and enables security-conscious users to catch anomalies.

For enterprise deployments, this principle extends to centralized tool definition management: rather than allowing agents to connect to tool servers and receive definitions directly, organizations can route tool discovery through a managed gateway that logs all tool definitions received, compares them to approved versions, and alerts security teams to any changes. This gateway architecture also provides a natural enforcement point for policies around which tool servers are permitted at all, addressing the shadow MCP problem by making unauthorized connections visible and blockable at the network level.

7. Defensive Architecture and Controls

7.1 Protocol-Level Mitigations

The most fundamental mitigation for tool poisoning and rug pull attacks is a protocol-level change that introduces cryptographic integrity for tool definitions. The Enhanced Tool Definition Interface (ETDI) proposal, published in academic research in 2025, describes a practical extension to MCP that addresses both attack categories through three core mechanisms [19]. First, cryptographic identity verification binds tool definitions to verifiable provider identities, making it detectable when a tool definition has been modified or when a tool is being impersonated. Second, immutable versioning creates an auditable history of tool definition changes and requires explicit re-approval for any change that affects a tool's scope or capabilities. Third, OAuth 2.0 scope integration maps tool capabilities to explicit permission scopes represented in signed JSON Web Tokens, enabling fine-grained access control at the tool level rather than the server level.

The ETDI proposal has not yet been incorporated into the MCP specification, but its core concepts—signed tool definitions, immutable versioning with mandatory re-approval for changes, and capability-bound OAuth scopes—represent the direction in which the protocol will need to evolve to address these threats at their root cause. Organizations evaluating MCP deployments should treat ETDI or equivalent cryptographic integrity mechanisms as a required capability for any deployment context that involves sensitive data or elevated permissions.

7.2 The MCP Gateway Pattern

The MCP gateway pattern—deploying a security-aware proxy between agents and tool servers—provides a pragmatic current-generation defense that does not require protocol changes. A properly implemented gateway intercepts all JSON-RPC communication between the agent client and tool servers, enabling several critical security functions [8][20].

At the policy enforcement layer, the gateway can validate that only approved tool servers are accessible, enforce rate limits per tool and per session, and block tool invocations that match patterns associated with known attack techniques. At the integrity layer, the gateway can maintain a registry of approved tool definitions and compare incoming definitions against approved versions, blocking sessions where tool definitions have changed without authorization. At the audit layer, the gateway captures complete logs of tool invocations, tool definitions received, and agent responses, providing the evidentiary record needed for both incident investigation and compliance attestation.

The Zero Trust principle of "never trust, always verify" applies directly to the gateway pattern. Even tool servers that have been previously reviewed and approved should have their tool definitions verified on every connection, and any change to a tool definition—regardless of how innocuous it appears—should trigger a review process before the changed definition is made available to the agent. This may create operational friction in fast-moving development environments, but the alternative—accepting tool definition changes as legitimate API updates without review—is the precise mechanism that rug pull attacks depend on.

7.3 Supply Chain Controls for MCP Packages

Securing the MCP package supply chain requires applying established software supply chain security practices—with additional MCP-specific controls for the tool definition layer. The foundational controls are familiar: software composition analysis to identify vulnerable or malicious dependencies in MCP server packages, private package registries with approval workflows rather than direct consumption from public repositories, and code signing verification for MCP server packages before installation.

MCP-specific controls layer on top of these foundations. Organizations should require that all MCP servers used in production have been built from source under controlled conditions rather than consumed as pre-built binary packages from public registries. Tool definitions should be versioned and reviewed as code artifacts, with changes requiring the same approval workflow as changes to application code. Automated scanning for adversarial instruction patterns in tool descriptions should be integrated into the CI/CD pipeline for any internally developed MCP server, and should be applied to third-party servers at ingestion time.

The postmark-mcp incident illustrated that the attack surface includes not just the initial installation of a malicious package but also updates to previously trusted packages. Monitoring for unexpected changes to installed MCP server packages—using file integrity monitoring or supply chain security tooling—is essential to detect rug pull attacks that operate through the package update mechanism rather than through runtime schema modification.

7.4 Identity and Access Management for Agents

Effective IAM governance for MCP-connected agents provides a critical defense-in-depth layer. If tool poisoning or a rug pull attack succeeds in redirecting an agent's behavior, the damage is bounded by the permissions the agent holds. Least-privilege provisioning of agent credentials—granting only the specific permissions required for the agent's defined task scope, rather than broad permissions that cover anticipated future needs—directly limits the blast radius of a successful attack.

The principle of task-scoped credentials extends this concept further: rather than granting an agent persistent, session-spanning credentials for all of its integrated tools, credentials can be issued with time-limited validity and bound to specific task contexts. An agent performing a file summarization task should hold credentials for file read access during that task and no other credentials; if the agent is subsequently

retasked to send email, it should receive a new, narrowly scoped credential for that specific task. This approach significantly complicates the credential theft component of tool poisoning attacks, because the value of any individual credential is bounded by its narrow scope and short validity period.

8. CSA Framework Alignment

8.1 MAESTRO Layer Mapping

CSA's MAESTRO (Multi-Agent Environment, Security, Threat, Risk, and Outcome) framework provides a seven-layer reference architecture for agentic AI threat modeling, introduced in February 2025 [16]. Tool poisoning and rug pull attacks primarily manifest at Layer 3 (Agent Frameworks), where tool selection and invocation logic resides, and Layer 4 (Deployment and Infrastructure), where tool server packages are deployed and tool definition schemas are maintained. Cross-layer effects propagate through Layer 2 (Data Operations), because successful attacks typically result in unauthorized data access or exfiltration, and through Layer 5 (Evaluation and Observability), where the attack's impact on agent behavior may or may not be detectable depending on monitoring instrumentation.

The MAESTRO framework's methodology—breaking down the system by layer, identifying layer-specific threat landscapes, and analyzing cross-layer threat propagation—is directly applicable to the MCP threat taxonomy described in this paper. Organizations using MAESTRO for threat modeling of agentic AI systems should explicitly include tool definition integrity and supply chain provenance as threat scenarios within the Layer 3 and Layer 4 threat landscapes. The cross-layer analysis should include the scenario where a compromised tool definition at Layer 3 produces unauthorized data operations at Layer 2 without any anomaly being detectable at Layer 5 through current monitoring instrumentation.

8.2 AI Controls Matrix Coverage

CSA's AI Controls Matrix (AICM) provides a comprehensive control framework for AI systems organized across eighteen domains and multiple stakeholder roles [21]. Three AICM domains are directly relevant to MCP security, each addressing a distinct dimension of the threat taxonomy described in this paper.

AICM Domain	Core Controls	Relevance to MCP Security
Supply Chain Security	Software component integrity verification, dependency management, third-party package governance	MCP server package integrity; prevention of typosquatting and dependency-injection attacks; code-signing requirements for MCP packages

AICM Domain	Core Controls	Relevance to MCP Security
Access Management	Credential lifecycle management, least-privilege provisioning, permission scope enforcement	Agent credential governance; task-scoped credential issuance; blast radius limitation for successful poisoning attacks
Monitoring and Observability	Runtime behavioral monitoring, anomaly detection, audit trail completeness	Detection of tool definition changes between sessions; behavioral anomaly flagging; audit records sufficient for post-incident investigation

Organizations aligning their MCP security programs with the AICM should prioritize controls in these three domains as foundational requirements. Particular attention is warranted for the gap between current deployment practices—where many organizations lack real-time schema integrity verification and behavioral anomaly detection—and the control levels the AICM prescribes.

8.3 Zero Trust Application

CSA's Zero Trust guidance is directly applicable to the MCP integration layer, and its core principle—that trust should not be presumed based on network location, prior approval, or claimed identity—maps precisely to the threat scenarios this paper describes [22]. In a Zero Trust-aligned MCP architecture, no tool definition is treated as trustworthy simply because it was approved in a prior session, because it originates from a previously vetted server, or because it arrives over an authenticated connection. Trust is re-established on each session through cryptographic verification of tool definition integrity, comparison against an authoritative registry, and validation that the tool's claimed capabilities match its authorized scope.

Implementing Zero Trust for MCP tool definitions requires infrastructure investment—a managed tool registry, cryptographic signing infrastructure for tool definitions, and automated verification tooling integrated into the MCP client or gateway—but the security model is conceptually straightforward. The challenges are operational: maintaining the registry as tool ecosystems evolve, managing re-approval workflows efficiently enough that they do not become a bottleneck, and ensuring that monitoring coverage is sufficient to detect anomalies that bypass the pre-execution verification layer.

9. Recommendations

For organizations deploying or evaluating MCP-connected AI systems, the following guidance is prioritized by both urgency and foundational impact.

Immediate: Establish Tool Definition Integrity Baselines

Before any MCP-connected agent is permitted to operate in a context involving sensitive data or elevated permissions, organizations should capture and store cryptographic hashes of all tool definitions the agent can access. These baselines should be compared against live tool definitions at the start of every session, and any change should trigger a human review before the changed definition is served to the agent. This control directly addresses rug pull attacks and requires no protocol changes or external dependencies—only a modest operational commitment to maintaining the baseline registry.

Immediate: Audit Existing MCP Tool Connections for Shadow Deployments

Organizations with existing AI agent deployments should conduct an inventory of all MCP server connections currently in use, including connections established by individual developers or business users outside formal approval processes. Shadow MCP connections are a significant and underappreciated exposure, because each represents an unreviewed tool definition surface. Discovered connections should be evaluated against the organization's security requirements, with unauthorized connections blocked at the network level and authorized connections brought under the tool definition integrity monitoring described above.

Short-Term: Deploy an MCP Gateway with Policy Enforcement

Organizations with multiple agent deployments should route all MCP traffic through a managed gateway that enforces connection policies, logs all tool invocations with full tool definition context, and implements the integrity verification described above at scale. The gateway architecture also provides the audit record necessary for compliance attestation and incident investigation, addressing the audit gap identified in Section 5.2.

Short-Term: Apply Supply Chain Controls to MCP Packages

MCP server packages should be treated as high-risk software supply chain components and subjected to the same controls applied to other security-critical dependencies: private registry mirroring, automated composition analysis for vulnerable or malicious packages, and file integrity monitoring for installed packages. Tool descriptions should be versioned and reviewed as code artifacts, with changes requiring explicit approval before deployment.

Short-Term: Implement Least-Privilege, Task-Scoped Agent Credentials

Agent credential scopes should be reviewed against actual task requirements and reduced to the minimum necessary. Where technically feasible, task-scoped credentials with limited validity periods should replace persistent, broad-scope credentials. This reduces the blast radius of any successful tool poisoning attack and limits the value of any credentials that a rug pull attack might attempt to exfiltrate.

Strategic: Prepare for Protocol-Level Integrity Mechanisms

Organizations should monitor the ETDI proposal and related MCP security standardization work and plan for adoption when cryptographic tool definition signing becomes available in the MCP specification or in widely deployed client implementations. The operational infrastructure required for ETDI—key management for tool signing authorities, tool definition version control, re-approval workflows—overlaps substantially with the tool definition integrity baseline described above, so early investment in that infrastructure also prepares organizations for the protocol-level solution.

Strategic: Integrate MCP Threat Scenarios into MAESTRO-Based Threat Models

Organizations using MAESTRO for agentic AI threat modeling should explicitly add tool poisoning, rug pull attacks, tool squatting, and supply chain compromise as threat scenarios in their Layer 3 and Layer 4 analyses. These scenarios should be scored using the MAESTRO risk matrix, with particular attention to cross-layer propagation paths that allow Layer 3 compromises to produce Layer 2 data operations that escape Layer 5 detection.

10. Conclusions

The Model Context Protocol has transformed the way AI agents connect to the capabilities they need to be useful—and in doing so, it has created a new attack surface that combines the connectivity of software supply chains with the instruction-injection potential of adversarial prompting. Tool poisoning, rug pull attacks, tool squatting, and supply chain compromise are not edge cases or theoretical constructs. They have been demonstrated at practical scale in research environments, exploited in real deployments, and confirmed in public package registries. The threat landscape is active and evolving.

The structural vulnerability underlying all of these attacks—MCP's current reliance on server-provided tool definitions without cryptographic integrity verification—is known to the research community and to the MCP maintainers, and protocol-level mitigations are under development. But the pace of enterprise MCP adoption has substantially outrun the pace of protocol security improvements. Organizations are deploying MCP-connected agents in production environments today, and those deployments are exposed to attacks for which the protocol currently provides no native protection.

The defensive path forward is layered. Immediate controls—tool definition integrity baselines, shadow deployment audits, supply chain hygiene—can be implemented without waiting for protocol changes and provide substantial protection against the most prevalent attack techniques. Gateway architecture and least-privilege credential management add depth to that foundation. And as the protocol evolves to incorporate cryptographic tool definition integrity through mechanisms like ETDI, organizations that have already built the operational infrastructure for integrity verification will be positioned to adopt those improvements rapidly.

The broader lesson for enterprise security architects is that the AI integration layer—everywhere that a language model connects to external capabilities—requires the same systematic security analysis that has been applied to APIs, supply chains, and access management in conventional software systems. The tools for that analysis exist in CSA's MAESTRO framework, AI Controls Matrix, and Zero Trust guidance. The application of those tools to MCP-connected agent deployments is both urgent and tractable. The risk of deferring that analysis is not.

References

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

[2] Invariant Labs. "MCP Security Notification: Tool Poisoning Attacks." Invariant Labs, April 6, 2025.

[3] ReversingLabs. "MCP client rug-pull attack worries mount for AppSec." ReversingLabs, 2025.

[4] Elastic Security Labs. "MCP Tools: Attack Vectors and Defense Recommendations for Autonomous Agents." Elastic Security Labs, 2025.

[5] Snyk. "Malicious MCP Server on npm postmark-mcp Harvests Emails." Snyk, September 2025.

[6] Wikipedia contributors. "Model Context Protocol." Wikipedia, accessed May 2026.

[7] Model Context Protocol. "Specification." Model Context Protocol, November 25, 2025.

[8] Model Context Protocol Blog. "The 2026 MCP Roadmap." Model Context Protocol, 2026.

[9] Willison, Simon. "Model Context Protocol has prompt injection security problems." simonwillison.net, April 9, 2025.

[10] Hou, Xinyi et al. "Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions." arXiv:2503.23278, March 2025.

[11] arXiv. "MCPTox: A Benchmark for Tool Poisoning Attack on Real-World MCP Servers." arXiv:2508.14925, August 2025.

[12] Deconvolute Labs. "MCP Rug Pull Attacks - Stealing AI Agent Credentials." Deconvolute Labs, 2025.

[13] AquilaX. "Shadow MCP: The New Security Risk of Unvetted AI Agent Tools." AquilaX, 2026.

[14] Palo Alto Networks. "Breakdown: Widespread npm Supply Chain Attack Puts Billions of Weekly Downloads at Risk." Palo Alto Networks Unit 42, September 2025.

[15] Cloud Security Alliance. "Securing Autonomous AI Agents." CSA Research, 2026.

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

[17] Authzed. "A Timeline of Model Context Protocol (MCP) Security Breaches." Authzed, 2025–2026.

[18] NIST. "AI Risk Management Framework (AI RMF 1.0)." NIST, January 2023.

[19] Bhatt, M., Narajala, V. S., and Habler, I. "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, 2025.

[20] Coalition for Secure AI (CoSAI). "Securing the AI Agent Revolution: A Practical Guide to Model Context Protocol Security." CoSAI, 2025.

[21] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.0." CSA, 2025.

[22] Cloud Security Alliance. "Zero Trust Advancement Center." CSA, accessed May 2026.