Published by
NIGERIAN SOCIETY OF PHYSICAL SCIENCES
Available online @ https://journal.nsps.org.ng/index.php/jnsps

J. Nig. Soc. Phys. Sci. 8 (2026) 3748

Journal of the Nigerian Society of Physical Sciences

Trust inversion and its exploitation: A threat model of the Model Context Protocol for agentic AI

Muhammad Qaim Aliyu Sambo[a,*], Augustine Yusuf[b], Ifeanyi Nwokoro[c], Abdulrahman Umar Sadiq[d], Victor Legbo Yisa[e]

[a]Department of Cyber Security, Abubakar Tafawa Balewa University, Bauchi, Nigeria
[b]Department of Cyber Security, National Open University of Nigeria, Abuja, Nigeria
[c]Department of Computer Science, Rhema University, Aba, Nigeria
[d]Department of Computer Science, Abubakar Tafawa Balewa University, Bauchi, Nigeria
[e]Department of Computer Science, Dalhousie University, Halifax, Canada

Abstract

The Model Context Protocol (MCP) standardizes connections between large language model applications and external tools and data. Its capability negotiation, dynamic discovery, insertion of tool results into model context, and server-initiated sampling create composition risks when server-originated artefacts influence actions executed under client authority. We define trust inversion as a condition in which an artefact crossing from a less-trusted MCP component into the client decision context is interpreted as operational authority without provenance-bound policy re-evaluation. The term is used as an MCP-specific analytical lens, not as a replacement for confused-deputy, indirect prompt-injection, control/data-plane-confusion, or transitive-authorization concepts. We construct an architecture-level threat model using three deployment profiles—local stdio, remote Streamable HTTP, and multi-server—together with STRIDE classification, explicit assets and security objectives, and a five-by-five likelihood–impact matrix. Publicly documented demonstrations, guidance-based attack classes, and an implementation CVE are coded by actor, preconditions, attack path, affected layer, violated property, and candidate controls. The analysis groups risks into descriptor and capability drift, cross-tool trust propagation, and sampling-mediated server influence, while separating protocol, ecosystem, implementation, and deployment weaknesses. We propose immediately deployable controls, including transport authentication, least privilege, trust-zone separation, schema validation, sandboxing, and audit logging, and distinguish them from extensions requiring standardization, including signed capability manifests and re-attestation. The resulting checklist is a threat-modelling aid; its mitigation effectiveness remains to be established through prototype implementation, red-team testing, and deployment-specific evaluation.

DOI: 10.46481/jnsps.2026.3748

Keywords: Model context protocol, MCP security, Agentic AI, Trust inversion, Tool poisoning

Article history:
Received: 23 July 2026
Received in revised form: 27 August 2026
Accepted for publication: 09 September 2026
Available online: 21 September 2026

© 2026 The Author(s). Published by the Nigerian Society of Physical Sciences under the terms of the Creative Commons Attribution 4.0 International license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

Communicated by: S. Folorunso

1. Introduction

The Model Context Protocol (MCP) was released by Anthropic on 25 November 2024 and is an open standard for accessing data wherever it resides, including cloud storage, business applications, local development environments, and similar systems [1]. Anthropic stated that MCP seeks to "replace in-

*Corresponding Author Tel. No.: +234-816-262-1754.
Email address: Qaeeemsambo100@gmail.com (Muhammad Qaim Aliyu Sambo)

dividual, one-off integrations with a unified protocol, enabling AI applications to reach data through connections with MCP clients and MCP servers". In concept, this aims to increase interoperability by reducing the need for a custom connector or hook for each AI system to interact with services such as GitHub, Slack, Google Drive, databases, local files, and enterprise internal tools [1].

The same flexibility that enables MCP applications also raises new security concerns. MCP extends AI usage from neutral text creation to acting on the world through tools. Tool invocation within MCP, as detailed in the official tool specification, includes database queries, web API calls, and numerical calculations, all of which may be exposed on MCP servers for the large language model (LLM) to access. MCP tool access is also "model-controlled", implying that the LLM dynamically inspects and uses available tools according to its contextual understanding of the user's prompts and environment [2].

In this paper, trust inversion denotes an MCP composition condition in which an artefact originating in a less-trusted component—such as a server-provided tool descriptor, external resource, tool result, or sampling request—is admitted to the client-side model context and influences a decision or action executed under the client's authority without provenance-bound policy re-evaluation. The term does not replace established concepts. An indirect prompt injection is one mechanism that can supply the malicious instruction; a confused deputy describes misuse of an intermediary's legitimate authority; control/data-plane confusion describes data being interpreted as instructions; and transitive authorization describes privilege flowing across a chain. Trust inversion is used here as an architectural lens for the MCP-specific composition of those mechanisms across dynamic discovery, model-mediated tool selection, cross-tool context reuse, and server-initiated client features. The MCP specification defines protocol capabilities and normative requirements, whereas the NSA publication is external security guidance for deploying MCP-enabled automation. The advisory observes that server-to-client features alter familiar interaction patterns and recommends explicit trust boundaries, policy enforcement, isolation, and monitoring. We therefore use the specification to establish protocol behaviour and the NSA guidance to inform deployment assumptions and controls [3].

Public reports and research studies have documented or demonstrated private-repository exfiltration through a GitHub MCP workflow, WhatsApp message exfiltration through a malicious MCP server, tool-parameter abuse, tool-invocation ambiguity, rug-pull behaviour, and CVE-2025-49596 in MCP Inspector [4–8]. Such attacks expose that ensuring MCP's safety cannot solely be achieved via model alignment but necessitates architecture-level, tool-level, context-level, authorization, and run-time governance. These sources comprise proof-of-concept demonstrations, guidance, empirical research, and an implementation vulnerability; they are not all confirmed production incidents.

The manuscript makes four bounded contributions: (1) it defines trust inversion as an MCP composition condition rather than a new primitive attack; (2) it separates local stdio, remote Streamable HTTP, and multi-server deployment profiles; (3) it classifies public evidence by actor, preconditions, affected layer, violated security property, and evidence type; and (4) it maps current controls and proposed extensions to residual risks in a deployment-oriented checklist. The checklist is an analytical aid and is not claimed to have measured effectiveness.

The remainder of the paper is organised as follows. Section 2 describes MCP architecture and security-relevant features. Section 3 reviews related work and positions the trust-inversion lens against existing MCP and agent-security research. Section 4 states the threat-modelling method, scope, assets, actors, assumptions, trust boundaries, and risk-ranking procedure. Section 5 analyses the principal architectural threat classes, and Section 6 classifies the supporting public cases by evidence type. Sections 7 and 8 map controls to threats and residual risks. Section 9 presents a deployment checklist, Section 10 states the limitations, and Section 11 concludes the paper.

2. Background: MCP architecture and security-relevant features

MCP has a client-server integration architecture. A typical MCP system consists of a user interface, a large language model or agent runtime, an MCP client, an MCP server (or more than one), and the tool(s) or data that servers offer. In Anthropic's initial announcement of MCP, they stated that MCP servers offered data to, while MCP clients connected to those servers [1]. Likewise, the NSA guide suggests that an application presents requests to an LLM; the MCP client libraries convert those requests to MCP messages; MCP servers dispatch requests to tools; and the outputs return through the path, then back to the user interface [3].

MCP uses JSON-RPC messages over standard transports. In the local stdio profile, the client launches the MCP server as a subprocess and exchanges newline-delimited JSON-RPC messages through standard input and output. In the remote profile, Streamable HTTP uses an MCP endpoint that accepts HTTP POST and GET and may use Server-Sent Events for streaming. In multi-server deployments, a host maintains separate client connections, but model context can still create a logical path between servers when an output from one server influences selection or parameters for another. The threat model therefore treats each client–server connection, transport credential, tool-output insertion point, and cross-server context transition as a distinct boundary.

2.1. Tools and dynamic discovery

A server that supports tools declares the tools capability during initialisation. A client discovers tools by sending the JSON-RPC method tools/list; when listChanged has been negotiated, the server may send notifications/tools/list_changed. These identifiers are JSON-RPC method names, not HTTP URL paths. A changed list or descriptor does not itself grant additional authority. Effective privilege gain occurs only when the client accepts the change, exposes the descriptor to the model, supplies credentials or data, or permits an invocation that exceeds the previously approved policy envelope.

To support user trust and safety, the specification suggests controls involving human-intervention interfaces that provide transparency about tool invocations, visual indications of tool invocations, and tool-use confirmations [2]. However, these are implementation suggestions rather than capabilities guaranteed by the protocol. This distinction is crucial because weak approval flows, unclear tool-use cases, and user fatigue with tool confirmations can be weaponised by compromised or malicious parties.

2.2. Sampling as bidirectional authority

Sampling is a server-to-client JSON-RPC request for client-side model generation. It is available only when the client declares the sampling capability during initialisation. Tool-enabled sampling additionally requires the client to declare sampling.tools, after which a server may include a tools array and optional toolChoice in a sampling/createMessage request. The request expresses server intent; it neither directly controls the model nor authorises a tool call. The client remains responsible for policy checks, user interaction, prompt and context selection, tool availability, and the response returned to the server. Risk arises when these controls admit untrusted server instructions, sensitive context, or tool use without origin-aware approval. A server may ask the client to use tools during sampling only when the client has explicitly declared support for tool-enabled sampling [9]. Sampling is a legitimate feature, but it becomes security-sensitive when server-initiated requests can introduce untrusted instructions, expose sensitive context, or create tool-use cycles without origin validation, policy checks, and user approval.

2.3. Optional authorisation

MCP defines an optional transport-level authorisation framework for HTTP-based transports. When supported, HTTP implementations should conform to the MCP authorisation specification; stdio implementations should obtain credentials from the environment rather than use that HTTP flow. Optional protocol authorisation must not be equated with absent authentication. Security depends on the deployment profile: remote servers require transport security, origin validation, authentication, audience-restricted tokens, and scope control, while local stdio servers require process, filesystem, secret, and environment isolation. Custom transports require controls appropriate to their own security model. [10].

This optionality creates uneven deployment security. A production MCP server interacting with regulated data, enterprise repositories, or local execution environments requires strict authentication, authorization, scope control, and token lifecycle management. Yet the protocol leaves significant responsibility to implementers. The official MCP security best-practices documentation recognizes security risks such as confused deputy attacks, token passthrough, server-side request forgery, session hijacking, local MCP server compromise, and scope minimization failures [11]. Figure 1: MCP deployment profiles and security-relevant boundaries. The diagram distinguishes local stdio and remote Streamable HTTP connections, client-held credentials, per-server trust boundaries, server-to-client sampling, and the insertion of tool results into model context.

3. Related work

Peer-reviewed work now covers several portions of the MCP and agent-security landscape. Maloyan and Namiot [12] analyse specification-level weaknesses and prompt-injection scenarios. Hasan *et al.* [13] empirically assess 1,899 open-source MCP servers and report conventional vulnerabilities and MCP-specific tool poisoning. Hou *et al.* [14] examine MCP architecture, server life cycle, ecosystem threats, and governance. Huang *et al.* [15] apply STRIDE and DREAD to MCP components and experimentally study prompt injection and tool poisoning. Li and Gao [16] examine registry- and host-level attack surfaces across six registries. Song *et al.* [17] identify attack vectors that emerge beyond the protocol layer, while Halloran *et al.* [18] experimentally audit security risks in LLM systems connected through MCP.

Zhao *et al.* [19] analyse parasitic toolchain attacks against MCP and identify failures of least privilege and domain-specific execution that can allow malicious instructions from untrusted sources to propagate into critical toolchains.

Research on tool-integrated language-model agents provides complementary evidence. Greshake *et al.* [20] demonstrate indirect prompt injection in real-world LLM-integrated applications; Zhan *et al.* [21] introduce the InjecAgent benchmark of 1,054 cases across 75 attacker and user tools; Yi *et al.* [22] benchmark and evaluate defences against indirect prompt injection; and Zhan *et al.* [23] show that adaptive attacks can bypass multiple defences, exceeding 50% attack success in their experiments.

The present manuscript therefore does not claim to originate MCP threat modelling or the three underlying weakness classes. Its narrower contribution is to define trust inversion as a compositional condition, distinguish it from established security primitives, separate protocol, ecosystem, implementation, and deployment layers, and use a common evidence schema to connect server-originated artefacts to client-authorised actions and deployable controls. The value added here is the cross-layer provenance lens, evidence-type classification, deployment-profile separation, and operational checklist. These distinctions should be read as an analytical synthesis rather than as the discovery of wholly new attack primitives.

Industry studies provide further evidence. Milanta and Beurer-Kellner [5] demonstrate a GitHub MCP workflow in which malicious content in a public issue causes an agent to retrieve information from a private repository and disclose it through a public pull request. Beurer-Kellner and Fischer [6] demonstrate a WhatsApp exfiltration attack in which a malicious MCP server influences an agent connected to a legitimate WhatsApp tool. HiddenLayer [4] documents tool-parameter abuse capable of exposing privileged context, including the system prompt.

The OWASP Top 10 for LLM Applications covers broader security categories that overlap with MCP risks, including

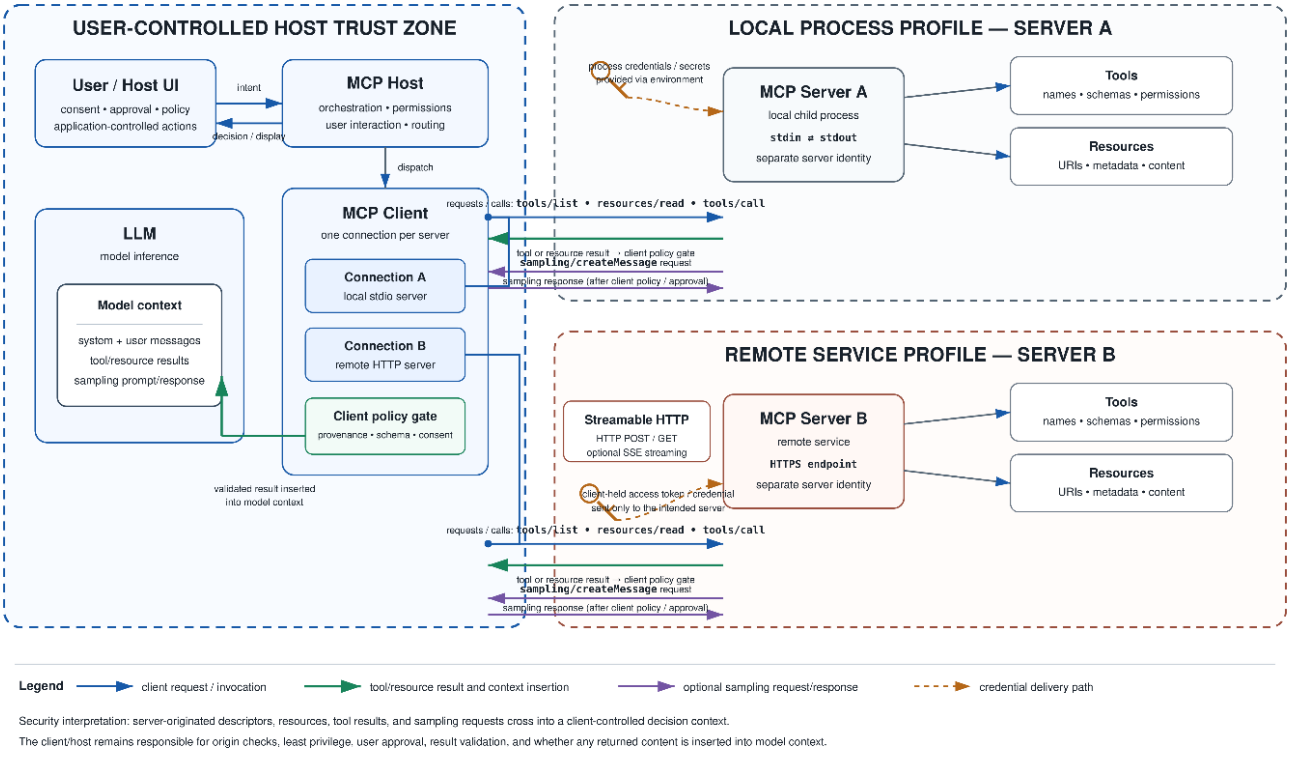

Figure 1: MCP deployment profiles and security-relevant boundaries.

prompt injection, insecure output handling, supply-chain vulnerabilities, sensitive-information disclosure, insecure plugin design, and excessive agency [24]. Trust inversion can be understood as an MCP-specific composition of several of these risks. Table 1 compares the present study with related work.

4. Threat model

This section introduces the assets, threat agents, trust boundaries, and security assumptions of MCP-enabled agent systems.

4.1. Threat-modelling method and scope

The unit of analysis is an MCP-enabled host comprising the user-facing application, the embedded language model or agent runtime, one or more MCP clients, MCP servers, exposed tools or resources, the underlying transport, and any authorisation server. The analysis uses three deployment profiles: (i) local stdio, in which the client launches the server as a subprocess; (ii) remote Streamable HTTP, in which an independently running server serves one or more clients over HTTP; and (iii) a multi-server host, in which outputs or requests from one server may influence tools connected through another server. These profiles are analysed separately because transport authentication, credential handling, process isolation, and cross-server data flow differ materially.

Threat construction follows six reproducible steps. First, assets and security objectives are identified. Second, trust boundaries and data or control flows are enumerated. Third, attacker entry points and preconditions are stated. Fourth, each scenario is classified using STRIDE: spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege. Fifth, each scenario is recorded as actor → entry point → artefact → trust boundary → client-side decision or action → affected asset. Sixth, candidate controls and residual risks are mapped to the scenario. The analysis distinguishes protocol properties from ecosystem, implementation, and deployment weaknesses.

For prioritisation, likelihood (L) and impact (I) are each scored on an ordinal scale from 1 (very low) to 5 (very high). The risk score is $R = L \times I$, where 1–4 is low, 5–9 is moderate, 10–16 is high, and 17–25 is critical. Scores are analytical judgements supported by stated preconditions and evidence type; they are not measured incident frequencies.

The evidence set is coded as one of five types: confirmed production incident, controlled proof-of-concept demonstration, empirical research experiment, guidance-derived attack class, or implementation vulnerability/CVE. For every case, the manuscript records the affected implementation or version, reproduction status, prerequisites, whether production exploitation was established, and the source. This prevents demonstrations and guidance from being described as confirmed production breaches.

Table 1: Comparison with related work.

Study	Scope/method	Primary object	Evidence	Novelty
Hou *et al.* [14]	Lifecycle and ecosystem taxonomy	Server creation, deployment, operation, maintenance	Systematic architectural study	Broader lifecycle lens; revised paper focuses on provenance crossing into client-authorised action.
Hasan *et al.* [13]	Large-scale empirical analysis	1,899 open-source MCP servers	Static and MCP-specific scanning	Empirical server quality/security study; revised paper is cross-layer threat construction.
Huang *et al.* [15]	STRIDE/DREAD and experiments	Five MCP components; prompt injection/tool poisoning	Threat modelling and controlled experiments	Closest formal threat-model overlap; revised paper must claim only the trust-inversion composition/evidence lens.
Li and Gao [16]	Registry and host attack analysis	67,057 servers across six registries	Cross-entity empirical study	Broader ecosystem entry and post-installation analysis; revised paper emphasises boundary-specific provenance and controls.
Maloyan and Namiot [12]	Protocol analysis and MCPBench	Attestation, sampling, multi-server trust	Controlled attack scenarios	Three classes overlap strongly; revised paper adds deployment profiles, evidence typing, and cross-layer control mapping.
This revised paper	Structured architecture-level threat analysis	Server-originated artefacts influencing client-held authority	Public cases classified by type; no claimed mitigation measurement	Defines trust inversion as a composition condition and separates protocol/ecosystem/implementation/deployment layers.

4.2. Protected assets

The most prominent MCP assets include user-provided prompts, system prompts, tool metadata, tool results, session tokens, OAuth bearer tokens, repository access, local runtimes, messaging history, private documentation, and logging data. These assets are not merely data: in agent-based systems, they can affect tool selection, subsequent reasoning, external agent communication, and authorisation of agentic actions. Table 2 presents the asset-to-security-concern taxonomy.

As shown in Table 2, MCP security depends on more than protecting transport channels. Server-originated descriptors, resources, tool outputs, and sampling requests can influence model decisions after crossing into the host-controlled context. The server publisher is responsible for the provenance and integrity of supplied artefacts, whereas the host and client operator remains responsible for credential scoping, policy enforcement, user approval, result validation, and context insertion. A failure at either layer can propagate across server connections and affect confidentiality, integrity, authorisation, provenance, availability, or accountability.

4.3. Threat agents

The threat actors considered in this paper include malicious MCP servers, compromised honest MCP servers, malicious external content providers, malicious or compromised clients, and browser or network attackers targeting local MCP development tools. The GitHub MCP demonstration illustrates a malicious external content provider: an attacker need not compromise the MCP server but can instead inject malicious instructions into a public GitHub issue that the agent consumes [5].

The WhatsApp MCP demonstration illustrates a malicious MCP server using rug-pull behaviour. Invariant Labs described a malicious server that initially offered a harmless tool and later changed its behaviour after the user had already approved it, exploiting weaknesses in re-approval of tool-description modifications [6].

4.4. Trust boundaries

There are multiple trust boundaries within an MCP deployment: user-to-client, client-to-model, model-to-MCP-client, MCP-client-to-server, server-to-tool, tool-output-to-model-context, and server-to-client sampling. Crucially, these are not solely data boundaries in MCP, but in many ways a message that crosses a boundary may carry some form of operational intent, either by instructing the model or a subsequent tool in what to do, or in defining for instance tool's available capabilities or state based on tool context. This is the heart of trust inversion. Figure 2: Trust-inversion attack path. A less-trusted descriptor, resource, tool result, or sampling request crosses into model context; the host applies—or fails to apply—provenance, authorisation, and user-approval policy before a model-selected tool call is executed with client-held authority. Red callouts identify the failed policy checks rather than treating all server content as automatically trusted.

Scoring note.. Likelihood (L) and impact (I) are scored on five-point ordinal scales, where 1 represents very low likelihood or

Table 2: Assets, security objectives, trust ownership, exposure surfaces, and consequences of compromise.

Asset	Security objectives	Trust owner	Exposure surface	Consequence of compromise
User prompts and conversation history	Confidentiality, integrity, authorisation, provenance	User and host/application operator	User interface, host memory or storage, model context, tool arguments, sampling prompts, logs, and remote-server requests	Disclosure of personal or business information, manipulation of user intent, unauthorised actions, or propagation of malicious instructions.
System prompts and hidden instructions	Confidentiality, integrity, authorisation, provenance	Host/application developer and operator	Host configuration, model-context assembly, debugging interfaces, logs, tool outputs, resources, and sampling exchanges	Disclosure or modification of internal policies, weakened safety boundaries, policy bypass, or persistent manipulation of model behaviour.
Tool descriptions and schemas	Integrity, provenance, authorisation, availability	MCP server publisher for supplied metadata; host/client operator for acceptance and enforcement	tools/list response, notifications/tools/list_changed, registry or installation metadata, cached descriptors, and model context	Manipulated tool selection, incorrect parameter construction, invocation of unintended operations, or unreviewed capability drift.
Tool outputs	Confidentiality, integrity, provenance, authorisation, accountability	Tool/server operator for generation; host/client operator for validation and use	tools/call responses, linked resources, client policy gates, model context, user interface, logs, and downstream systems	Indirect prompt injection, disclosure of private data, false model decisions, malicious downstream triggers, or execution of unsafe follow-up actions.

impact and 5 represents very high likelihood or impact. Risk is calculated as R=L×I, where 1–4 is Low, 5–9 is Moderate, 10–16 is High, and 17–25 is Critical. These scores are analytical judgements derived from the stated preconditions, deployment profiles, demonstrated evidence, attacker access and potential consequences. They must not be interpreted as measured incident frequencies.

Table 3 distinguishes protocol-enabled behaviour from ecosystem, implementation and deployment weaknesses. T3 concerns an optional protocol capability whose risk depends on client policy and deployment conditions, whereas T6 is a confirmed vulnerability in a particular implementation and must not be characterised as a protocol flaw. The remaining scenarios generally arise from interactions among server-originated artefacts, shared model context, ambiguous provenance and inadequate client-side enforcement. Consequently, the assigned risk scores are conditional on the stated preconditions and should be revised where the analysed deployment profile materially differs.

5. Architectural vulnerabilities

5.1. Capability amplification without strong attestation

Descriptor or capability drift occurs when a server changes an advertised tool name, description, schema, output type, or related metadata after an earlier review. The change does not itself confer authority. Effective privilege escalation occurs only if the client accepts the changed descriptor, supplies credentials or data, or permits an invocation outside the previously approved policy envelope. The threat is therefore a composition of unverified drift and a client-side enforcement failure, not the list-changed notification alone. [2, 3].

Capability amplification and rug pull attacks are closely related. Beurer-Kellner and Fischer [6] produced a ‘sleeper’ MCP server that would post an initially inoffensive tool description only to shift over time to send instructions impacting WhatsApp tool use. Descriptor level MCP literature also calls rug-pulls a general class of attack that manipulates a tool descriptor post-approval to subvert the model(s) it directs. [25].

5.2. Implicit trust propagation across chained tools

Uncontrolled trust propagation occurs when the output of one tool is used as input to another tool or model as though it were trusted. MCP exacerbates this risk because tool outputs may contain unstructured text, structured content, URLs, embedded resources, and annotations. The MCP tools specification permits tool results to contain structured or unstructured content and requires clients to validate results before providing them to the LLM [2]. Milanta and Beurer-Kellner [5] demonstrate how malicious content in a public GitHub issue can enter the agent’s context through a GitHub MCP workflow.

The threat from poor local-tool authentication is illustrated by CVE-2025-49596. According to the NVD [8], MCP Inspector versions before 0.14.1 contained a remote-code-execution vulnerability because the Inspector proxy and client lacked au-

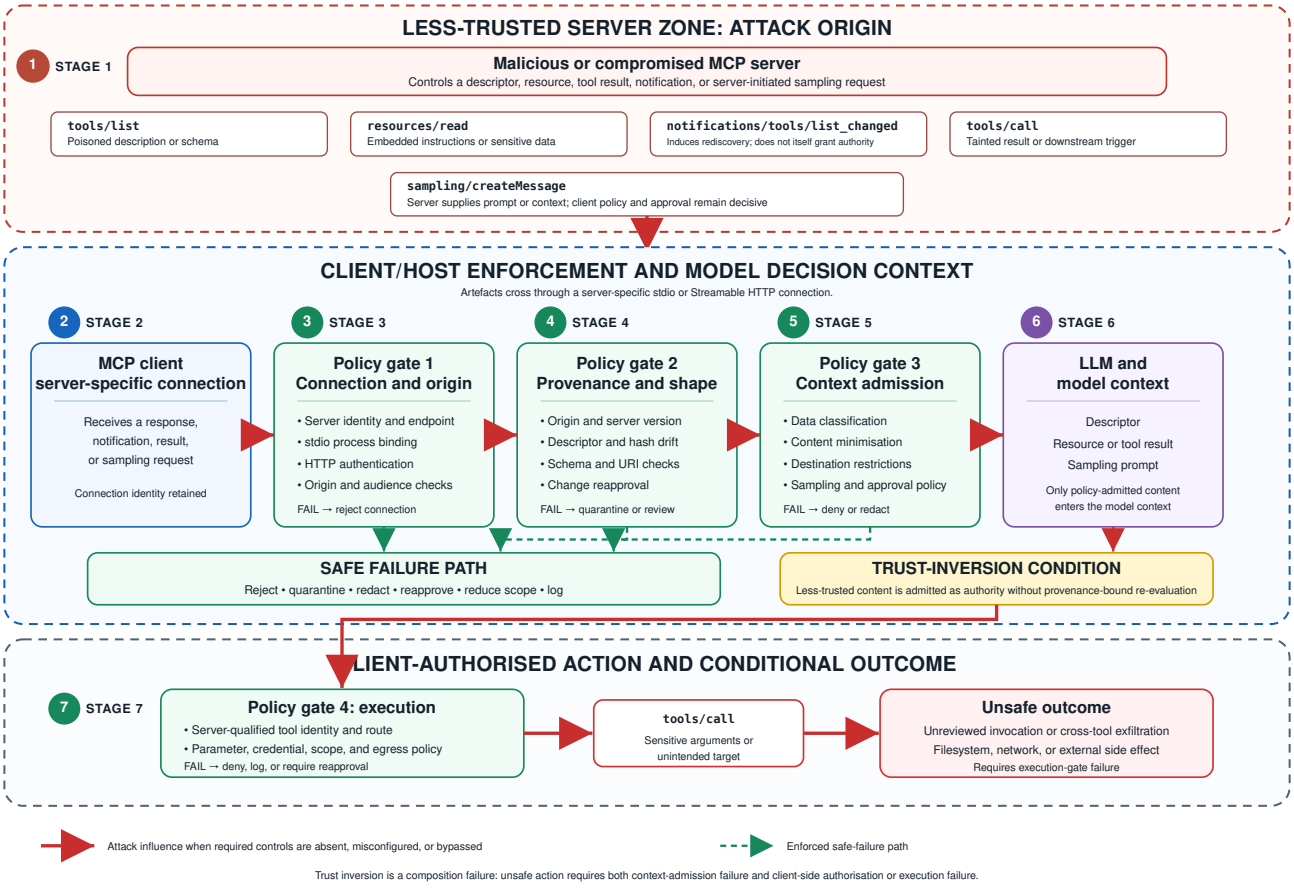

Figure 2: Trust-inversion attack path showing server-originated artefacts, policy checks, and client-authorised action.

thentication, allowing unauthenticated requests to launch arbitrary commands through stdio. Oligo Security similarly documented the browser-to-localhost attack path [7]. An exposed affected deployment could therefore allow a remote attacker to execute commands on the user's development machine.

The agent then fetched information from the privately accessible repository into its context and subsequently disclosed that repository information in a public pull request, even though the attacker had not compromised the GitHub MCP server; the malicious tool output was instead taken at face value.

5.3. Bidirectional sampling and server-originated influence

Sampling is a server-to-client JSON-RPC request for client-side model generation. It is available only when the client declares the sampling capability during initialisation. Tool-enabled sampling additionally requires the client to declare sampling.tools, after which a server may include a tools array and optional toolChoice in a sampling/createMessage request. The request expresses server intent; it does not directly control the model or authorise a tool call. The client remains responsible for policy checks, user interaction, prompt and context selection, tool availability, and the response returned to the server. Risk arises when those controls admit untrusted server instructions, sensitive context, or tool use without origin-aware approval.

The danger is not sampling per se. The danger is that sampling may not reliably be tethered to origin, context, approval, and least privilege. While the specification strongly recommends human approval and user approval mechanisms for sampling, the spec does not mandate one universal interface pattern [9]. Sampling in a secure deployment should be treated as a sensitive, elevated function, not a standard message back-and-forth. Figure 3: Policy-mediated MCP sampling sequence. The server sends sampling/createMessage only after the client has advertised sampling support. The client evaluates origin, prompt, context, user approval, and optional sampling.tools policy before invoking the model or any tool; only the approved result is returned to the server.

6. Real-world exploit mapping

Table 4 shows that MCP exploitation is not limited to conventional implementation flaws. CVE-2025-49596 is a concrete implementation vulnerability in MCP Inspector, but GitHub MCP exfiltration, WhatsApp MCP exfiltration, tool poisoning, and rug-pull attacks demonstrate a broader architectural problem: tool metadata, tool outputs, and server-

Table 3: Threat-scenario register for MCP deployment profiles.

Threat and layer	Scenario and affected asset	Evidence and risk	Controls and residual risk
T1: Descriptor drift; layer: Ecosystem/implementation	Actor/entry point: Malicious or compromised MCP server through tools/list or notifications/tools/list_changed. Preconditions: The client accepts changed tool descriptions or schemas without comparison, provenance verification, or renewed approval; descriptors are exposed to the model. Attack path: A server initially advertises a benign tool → later changes its name, description, schema, permissions, or output type → the client accepts the update → altered metadata influences tool selection or parameter construction. Asset/property: Tool descriptions, schemas, and user intent; integrity, provenance, authorisation.	Evidence: Research-demonstrated attack class and specification-derived scenario; cite the exact study and tested implementation. L: 4 — Likely: plausible where dynamic updates are accepted automatically. I: 4 — Major: may redirect tool choice or enable actions outside the reviewed policy. Risk: 16 — High.	Controls: Record descriptor hashes and versions; compare changes; require reapproval for security-relevant changes; display server origin; apply least privilege; use signed manifests only as a proposed extension. Residual risk: Moderate: stable but malicious descriptors, compromised publishers, semantic changes that pass structural comparison, or stolen signing keys may remain effective.
T2: Cross-tool data exfiltration; layer: Ecosystem/deployment	Actor/entry point: Malicious tool or server entering through a descriptor, resource, or tool result. Preconditions: Multiple tools or servers share model context; one component can access sensitive data; the client lacks cross-tool information-flow and egress controls. Attack path: A trusted tool retrieves sensitive information → its result enters model context → a malicious descriptor or result induces the model to include that information in another tool call → the receiving tool transmits or stores it externally. Asset/property: Prompts, conversation history, private files and business data; confidentiality, authorisation, provenance.	Evidence: Publicly documented proof-of-concept demonstrations; do not call them production breaches unless the cited source establishes compromise. L: 4 — Likely: demonstrated where tools share context without data-flow restrictions. I: 5 — Severe: may disclose high-value data across systems or trust zones. Risk: 20 — Critical.	Controls: Classify and label sensitive data; enforce cross-tool data-flow policy; minimise context; restrict egress; isolate servers; preview sensitive arguments; require user approval; redact protected values. Residual risk: High: legitimate data-sharing workflows, encoded content, indirect leakage, or an authorised but compromised destination can bypass simple filters.

originated requests can become control-plane inputs. Figure 4 illustrates the attack flow in which malicious content in a public GitHub issue causes an agent to retrieve private-repository data and disclose it through a public pull request.

7. Architectural countermeasures

7.1. Signed capability manifests and re-attestation

A signed capability manifest is proposed as an ecosystem or protocol extension rather than a control already required by MCP. At minimum, the manifest should bind a server identity and version to tool names, descriptor and schema hashes, output schemas, permission scopes, allowed network destinations, data classifications, and validity dates. It should identify the signing key and algorithm and carry a signature over a canonical representation. A client would verify the signature against an organisation or registry trust store, check validity and revocation status, bind the manifest to the negotiated server version, and require re-approval when a security-relevant field changes. Key rotation should permit overlapping validity, while compromise should trigger revocation and fail-closed handling for high-risk tools. For compatibility, unsigned servers may remain usable only when policy explicitly permits them and labels them as unverified; they should not be represented as attested. The proposal requires standardisation, interoperable implementations, and evaluation before its security effect can be claimed

The proposed control directly mitigates capability amplification and rug-pull attacks. It also complements emerging MCP security literature proposing manifest signing and descriptor-integrity mechanisms to mitigate tool poisoning and descriptor mutation [12, 25].

Table 3: Threat-scenario register for MCP deployment profiles (continued).

Threat and layer	Scenario and affected asset	Evidence and risk	Controls and residual risk
T3: Sampling-mediated influence; layer: Protocol/implementation	Actor/entry point: Malicious or compromised server through sampling/createMessage. Preconditions: The client has declared sampling capability; requests are processed without adequate policy checks, context separation, or user approval; tool-enabled sampling may be available. Attack path: Server submits a sampling request containing adversarial instructions or context → client forwards an approved request to the LLM → generated content influences subsequent reasoning, responses, or tool use → result is returned to the server or reused downstream. Asset/property: Model context, user intent, and client-held authority; integrity, authorisation, provenance.	Evidence: Specification-derived analytical scenario supported by prompt-injection research; not a confirmed production incident unless separately evidenced. L: 3 — Possible: requires sampling to be enabled and insufficiently governed. I: 4 — Major: can influence model output or downstream client-authorised actions. Risk: 12 — High.	Controls: Disable sampling when unnecessary; allowlist authorised servers; validate prompt and context fields; separate sampling context; require user approval; restrict tool-enabled sampling; limit returned content and sensitive context. Residual risk: Moderate: malicious meaning can remain hidden in otherwise valid requests, and adaptive prompts may evade content-based detection.
T4: Tool-parameter context exposure; layer: Implementation/deployment	Actor/entry point: Malicious server or tool developer through a tool description, schema, or invocation handler. Preconditions: The client automatically constructs arguments from model context; sensitive context is available; schema, parameter, and egress checks are weak. Attack path: A tool description or schema requests sensitive context → the model places prompts, history, identifiers, files, or secrets into tool arguments → the client invokes the tool → the remote server, downstream API, or logs receive the information. Asset/property: User prompts, conversation history, credentials and business data; confidentiality, authorisation, accountability.	Evidence: Controlled research demonstration or publicly documented proof of concept; cite the exact source and affected implementation. L: 4 — Likely: plausible in automated tool-calling workflows with broad context access. I: 4 — Major: may expose confidential data or secrets to an unintended recipient. Risk: 16 — High.	Controls: Enforce strict input schemas; reject undeclared parameters; minimise context; filter secrets; classify destinations; show argument previews; require approval for sensitive fields; redact logs. Residual risk: Moderate: some sensitive parameters may be operationally necessary, while meaning-based leakage can evade structural validation.

7.2. Explicit trust zones and MCP-aware proxies

MCP deployments should be organised into explicit trust zones. Public-data, internal-business, regulated-data, local-execution, and messaging tools should not be given an unconstrained shared context. NSA recommendations call for establishing agent-plugin, plugin-user, and model-user trust-boundary policies; grouping tools and models by data classification; and deploying filtering proxies or data loss prevention (DLP) controls at external access points [3].

MCP awareness by a proxy would result in origin enforcement, denied sampling, interrogation of tool requests, manifest validation, parameter validation, DLP policy enforcement, and event logging. Alternatively, Invariant Labs advocates real-time scanning and proxy-based scanning of MCP installations [5].

7.3. Output sanitisation and schema validation

Tool results and linked resources should be treated as untrusted inputs. Structural controls should validate the declared schema, size, media type, destination, and data classification before content is admitted to model context or a downstream tool. Prompt-injection detection may be used as a non-authoritative risk signal, but it must not be treated as a reliable security boundary. Policy enforcement should remain independent of detector output and should constrain which tools, data classes, destinations, and actions are permitted. Residual risk remains because semantically malicious content can satisfy syntactic schemas and adaptive attacks can bypass known detectors.

However, output sanitisation should not be considered a suf-

Table 3: Threat-scenario register for MCP deployment profiles (continued).

Threat and layer	Scenario and affected asset	Evidence and risk	Controls and residual risk
T5: Invocation-name/path ambiguity; layer: Ecosystem/implementation	Actor/entry point: Malicious or misconfigured server registering a duplicate, misleading, or visually similar tool identity. Preconditions: Multiple servers are aggregated; tool identifiers are not server-qualified in the host's internal routing; collision or origin checks are absent; the model selects tools using descriptive similarity. Attack path: Attacker advertises a duplicate or confusable tool name → the model selects the unintended tool → the client resolves the ambiguous identifier or route incorrectly → arguments, credentials, or actions are sent to the wrong server. Asset/property: Tool identity, routing state, credentials and user intent; integrity, provenance, authorisation.	Evidence: Analytical threat scenario; requires controlled reproduction or a directly supporting source before being presented as demonstrated. L: 3 — Possible: depends on multi-server aggregation and ambiguous routing. I: 4 — Major: may redirect sensitive data or privileged operations to an unintended component. Risk: 12 — High.	Controls: Use immutable server-qualified tool identifiers; detect collisions and Unicode confusables; display tool origin; maintain an explicit routing map; bind permissions to server identity and version; require reapproval after identity changes. Residual risk: Moderate: semantic similarity and deceptive but technically unique names may still influence model selection.
T6: MCP Inspector unauthenticated RCE; layer: Implementation/deployment	Actor/entry point: Remote or local-network attacker through an exposed vulnerable MCP Inspector service or proxy endpoint. Preconditions: An affected Inspector version is deployed; its service is reachable by an attacker; authentication, origin restrictions, network isolation, or patching is absent. Attack path: Attacker sends a crafted request to the vulnerable Inspector endpoint → attacker-controlled input reaches the affected command-execution path → operating-system commands execute with the Inspector process's privileges. Asset/property: Local execution environment, filesystem, credentials and service availability; confidentiality, integrity, availability, authorisation.	Evidence: Confirmed implementation vulnerability, CVE-2025-49596; not evidence of a protocol-level flaw. L: 4 — Likely when exposed: exploitation is feasible under the stated vulnerable deployment conditions. I: 5 — Severe: arbitrary code execution can compromise the host, secrets and connected systems. Risk: 20 — Critical.	Controls: Upgrade to the latest patched release; verify the vendor's remediation; bind development services to loopback; enforce authentication and origin checks; restrict network access; run with least privilege; sandbox the process; avoid exposing developer tools publicly. Residual risk: Moderate: patched developer tools can still retain powerful filesystem, process and network access; misconfiguration or another implementation flaw may recreate exposure.

ficient security control on its own, because research on adaptive prompt-injection techniques demonstrates that multiple defence layers can be bypassed once their underlying logic is exposed to an attacker [23]. It should therefore be supplemented by least-privilege access, tool isolation, manifest enforcement, and runtime auditing.

7.4. Session binding and token hygiene

Sensitive MCP operations should be authenticated with client identity, server identity, user identity, session nonce, timestamp and manifest version, not just by using long-lived bearer tokens. Authorization is explicitly optional by the MCP authorization spec and the NSA guidance cautions of token and session security holes, such as reuse and replay [3, 10].

For remote HTTP deployments, bind access tokens to the intended resource server and audience, use short lifetimes and least privilege, validate origins, and protect session identifiers against reuse and fixation. For local stdio deployments, bind the launched server process to an approved executable, working directory, environment, and filesystem scope. Timestamps and nonces can reduce replay where the implementation signs or otherwise authenticates the relevant message; adding unauthenticated fields alone does not create security.

The threat from weak local-tool authentication is illustrated by CVE-2025-49596. According to the NVD, MCP Inspector versions before 0.14.1 contained a remote-code-execution vulnerability because authentication was absent between the In-

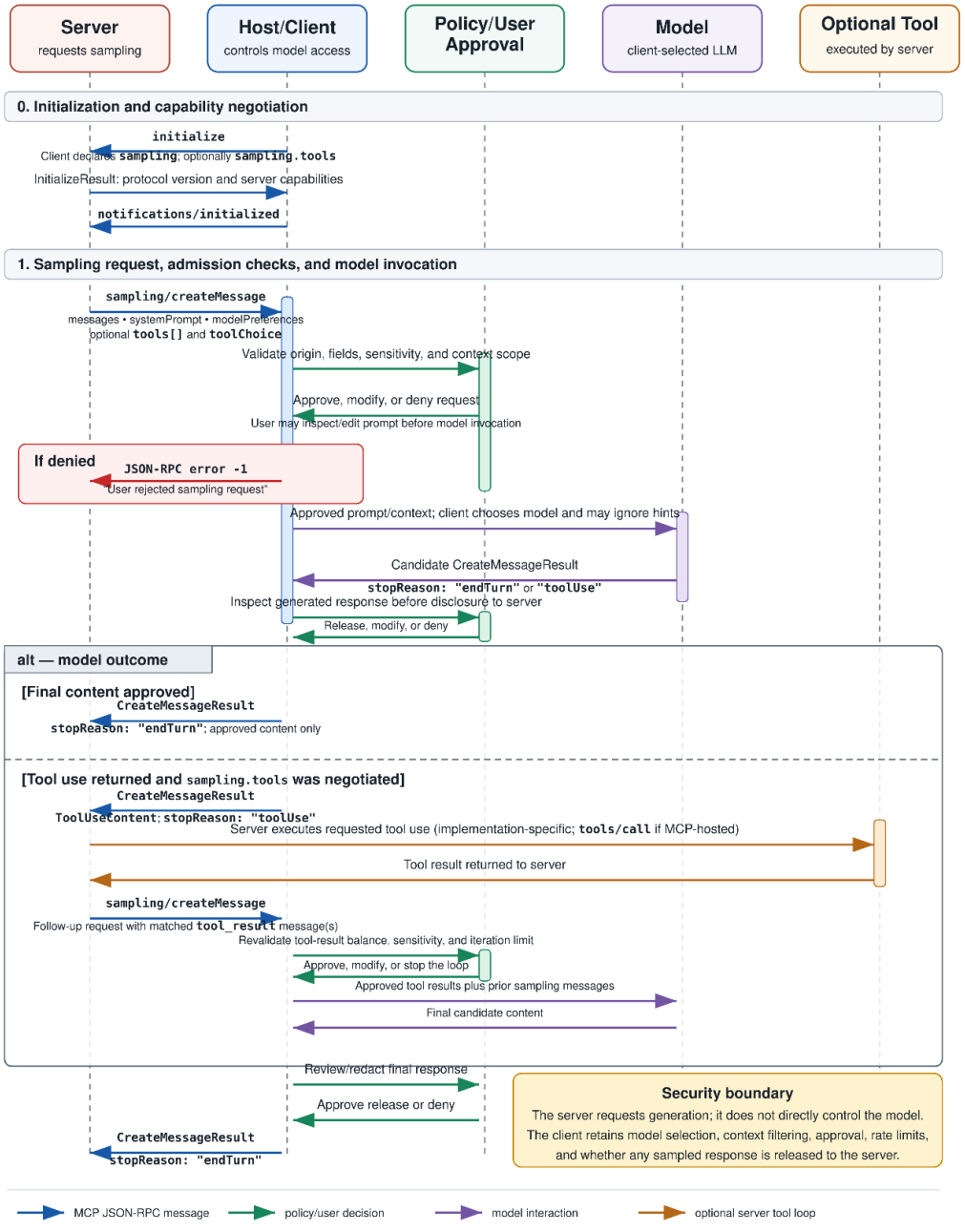

Figure 3: Policy-mediated MCP sampling sequence.

spector proxy and client, allowing unauthenticated requests to launch arbitrary commands through stdio. An affected Inspector that was reachable to an attacker could therefore permit remote code execution on the user's development machine [7, 8].

7.5. Sandboxing and least privilege

Tools that can access local or sensitive resources should execute with least privilege in a sandbox or container. Restrictions should cover permitted filesystem paths, network egress destinations, process privileges, Linux capabilities and system calls, environment variables and secrets, child-process creation, CPU and memory, and execution time. Appropriate operating-system mechanisms include AppContainer, seccomp, AppArmor, and SELinux. Sandboxing limits blast radius; it does not establish that a tool request is legitimate.

7.6. Audit logging and trace-based monitoring

MCP clients, servers, and enforcement proxies should record tool discovery and descriptor versions, tool invocations and parameters, output hashes or protected references, sampling requests, authorisation decisions, user approvals, server identity, session identifiers, manifest versions where used, policy outcomes, and timestamps. Logs should protect sensitive content, preserve integrity, and support correlation

Table 4: Real-world exploit mapping.

Exploit or disclosure	Root architectural weakness	Impact	Evidence
GitHub MCP private-repository exfiltration	Implicit trust propagation and weak data-flow isolation	Private repository data leaked into public repository workflow	Invariant Labs showed that a malicious public issue could manipulate an agent into pulling private repository data and leaking it through a public PR [5].
WhatsApp MCP message-history exfiltration	Tool-description poisoning, rug pull, and cross-server trust propagation	Chat history or contacts leaked through manipulated WhatsApp tool use	Invariant Labs demonstrated malicious MCP server behaviour against an agent also connected to WhatsApp MCP [6].
Hidden Layer tool-parameter abuse	Parameter-level context exposure and inadequate validation	Sensitive system and conversation information exposed through tool-parameter design	Hidden Layer documented MCP tool-parameter abuse and related concerns including malicious tool descriptions and naming collisions [4].
Tool invocation path confusion	Weak tool namespace and origin control	Malicious or lookalike tools may hijack execution flow	NSA guidance identifies tool invocation path confusion and naming collisions; Zhao *et al*. analyse MCP toolchain attacks [3, 19].
CVE-2025-49596 in MCP Inspector	Missing authentication between inspector client and proxy	Remote code execution through unauthenticated requests launching MCP commands over stdio	NVD describes MCP Inspector versions below 0.14.1 as vulnerable to RCE due to lack of authentication [8].
Malicious server rug pull	Capability drift after approval	Benign tool later becomes malicious without meaningful user re-approval	Invariant Labs described a sleeper server that changed behaviour after prior approval [6].

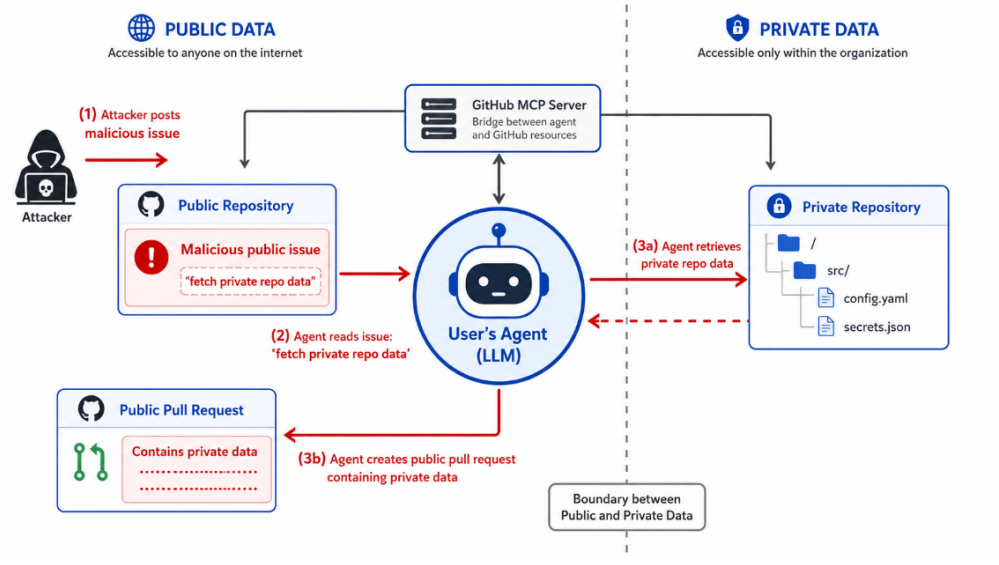

Figure 4: Exploit flow in which a malicious GitHub issue causes private-repository leakage.

across client–server connections. Central monitoring can assist anomaly detection and forensic reconstruction, but logging does not prevent an attack and must be combined with preventive controls. Figure 5 summarises how the proposed architectural countermeasures operate across MCP trust boundaries, from server-provenance verification and policy enforcement to sandboxing and audit logging.

8. Conceptual evaluation

This section conceptually compares the countermeasures proposed in this work with known attack categories. We make

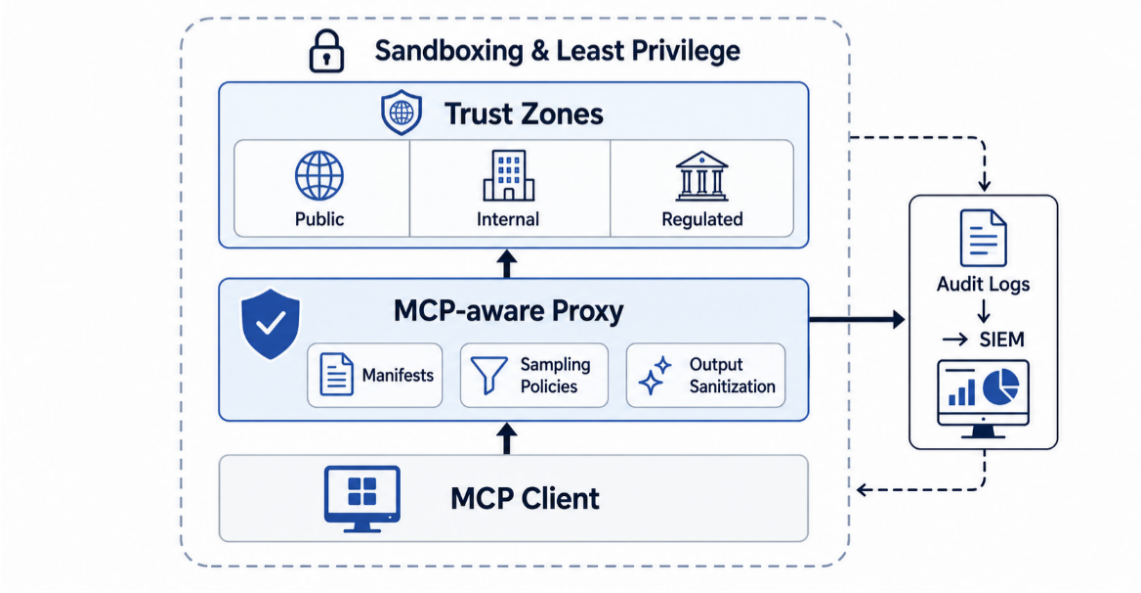

Figure 5: Architectural countermeasures overview.

no claims about measured mitigation effectiveness. Table 5 presents the attack-class taxonomy.

These comparisons indicate that MCP protection must be architectural. Prompt filtering or model alignment alone is insufficient when tools determine what can be read, written, or acted upon, including file systems, messaging services, repositories, local execution environments, and underlying API endpoints. Invariant Labs reached a similar conclusion, arguing that the harmful GitHub MCP flow likely requires an operating-system-level security posture rather than a model-only approach [5].

9. Production MCP threat-modelling checklist

Before connecting an MCP server, verify its publisher or operator, distribution source, server identity, version, transport endpoint, requested permissions, and expected tools and resources. Maintain a server-specific inventory and review any change to a tool name, description, input or output schema, permission scope, data classification, or permitted network destination before accepting it. Separate public-data, internal-data, regulated-data, messaging, repository, and local-execution tools into distinct trust zones, and bind credentials, permissions, and approval rules to the relevant server and deployment profile.

Permit server-initiated sampling only when the client has declared the required sampling capability and the requesting server, prompt, context, optional tools, and tool-use conditions satisfy client policy and any required user approval. Treat server-supplied descriptors, resources, tool results, and sampling requests as untrusted inputs until their origin, structure, size, media type, linked destinations, and data classification have been checked. Validate tool parameters against the accepted schema, minimise the content inserted into the model context, constrain cross-tool data flows and outbound destinations, and require renewed approval when a request exceeds the previously accepted policy envelope.

Use least-privilege, server-scoped credentials and request only the data and permissions required for the approved operation. Apply transport-appropriate authentication and origin checks to remote HTTP deployments. Bind local stdio servers to an approved executable, working directory, environment, filesystem scope, and network policy. Run local-execution tools with restricted filesystem, process, network, secret, CPU, and memory access, using containers or operating-system sandboxing where appropriate. Record server identity, descriptor versions, sampling requests, tool invocations and parameters, protected result hashes or references, policy decisions, user approvals, outcomes, and timestamps in access-controlled audit logs. Periodically inventory the environment for unknown, obsolete, or externally exposed servers and development interfaces, and apply the latest stable security patches. For MCP Inspector, version 0.14.1 is the minimum release that remediated CVE-2025-49596; it is not a permanent recommended target. Deploy the latest stable supported release from the maintained project line and keep MCP Inspector and similar development interfaces unavailable to untrusted networks.

10. Limitations

This study is an architecture-level threat analysis. The authors did not reproduce the cited attacks, perform penetration testing, implement the proposed signed-manifest extension, or measure mitigation effectiveness or runtime overhead. Risk ratings are structured analytical judgements rather than incident-frequency estimates. The MCP specification, SDKs, authorisation model, registry practices, and security tooling continue to evolve; consequently, the findings are version-dependent and should be re-evaluated against later protocol revisions and

Table 5: Attack-class taxonomy.

Attack class	Countermeasures and expected mitigation effect
GitHub MCP private-repository exfiltration	Main countermeasures: Trust zones, data-flow policy, output validation, audit logging Expected mitigation effect: Would block public-repository content from triggering private-repository retrieval and public leakage.
WhatsApp MCP exfiltration	Main countermeasures: Signed manifests, descriptor re-attestation, sampling restrictions, proxy enforcement Expected mitigation effect: Would block malicious descriptor changes and prevents untrusted servers from influencing trusted messaging tools.
Tool-parameter abuse	Main countermeasures: Strict schemas, unused-parameter rejection, sensitive-context blocking, logging Expected mitigation effect: Would reduce parameter channels that expose system prompts, tool history, or conversation history.
Tool invocation path confusion	Main countermeasures: Unique tool identifiers, namespace policy, manifest enforcement Expected mitigation effect: Would prevent malicious lookalike tools from silently replacing trusted tools.
MCP Inspector RCE	Main countermeasures: Authentication, origin checks, local binding, version updates, sandboxing Expected mitigation effect: Would block unauthenticated browser-to-localhost command launch and limits blast radius.
Rug-pull attacks	Main countermeasures: Descriptor hashing, signed manifests, re-approval on change Expected mitigation effect: Would convert silent capability drift into a detectable and reviewable event.

patched implementations. Future work should implement representative controls, conduct red-team and benchmark evaluation across the three deployment profiles, report false-positive and false-negative behaviour, measure operational overhead, and test residual attack paths.

11. Conclusion

MCP increases the compositional security burden of agentic applications because server-originated descriptors, external resources, tool results, and sampling requests may enter the client-side decision context and influence actions performed with client-held authority. We describe this condition as trust inversion, while recognising that its mechanisms overlap with indirect prompt injection, confused-deputy behaviour, control/data-plane confusion, and transitive authorisation. The evidence reviewed in this paper comprises controlled demonstrations, empirical research, guidance-derived attack classes, and an implementation CVE; it should not be represented uniformly as confirmed production compromise.

Immediate risk reduction should focus on deployment controls that can be applied today: explicit trust zones, transport-appropriate authentication, least-privilege credentials, origin and schema validation, policy-constrained cross-tool data flow, sandboxing, patched components, and traceable audit logs. Signed capability manifests and descriptor re-attestation are proposals that require a defined trust model, revocation and rotation procedures, standardisation, interoperable implementations, and empirical evaluation. The checklist offered here is therefore a structured threat-modelling aid rather than a proven guarantee. Future work should validate the model through prototypes, red-team testing, benchmark datasets, and deployment-specific measurement.

List of abbreviations

Abbreviation	Full meaning
AI	Artificial Intelligence
API	Application Programming Interface
CVE	Common Vulnerabilities and Exposures
DLP	Data Loss Prevention
HTTP	Hypertext Transfer Protocol
LLM	Large Language Model
MCP	Model Context Protocol
ML	Machine Learning
NIST	National Institute of Standards and Technology
NSA	National Security Agency
NVD	National Vulnerability Database
OAuth	Open Authorization
OS	Operating System
OWASP	Open Web Application Security Project
PR	Pull Request
RCE	Remote Code Execution
SDK	Software Development Kit
SELinux	Security-Enhanced Linux
SIEM	Security Information and Event Management
STDIO	Standard Input/Output
URL	Uniform Resource Locator

Data availability

The data supporting the findings of this study are available from the corresponding author upon reasonable request.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this manuscript.

Funding

The authors received no specific funding from any public, commercial, or not-for-profit organisation for this research.

References

[1] Anthropic, "Introducing the Model Context Protocol", 25 November 2024. Available online: https://www.anthropic.com/news/model-context-protocol.

[2] Model Context Protocol, "Tools", Model Context Protocol specification, version 2025-11-25, 2025. Available online: https://modelcontextprotocol.io/specification/2025-11-25/server/tools.

[3] National Security Agency, *Model Context Protocol (MCP): Security design considerations for AI-driven automation*, U/OO/6030316-26, PP-26-1834, Version 1.0, National Security Agency, Fort Meade, Maryland, USA, May 2026. Available online: https://media.defense.gov/2026/Jun/02/2003943289/-1/-1/0/CSI_MCP_SECURITY.PDF.

[4] K. Evans, T. Bonner & C. McCauley, "Exploiting MCP tool parameters", HiddenLayer, 15 May 2025. Available online: https://www.hiddenlayer.com/research/exploiting-mcp-tool-parameters.

[5] M. Milanta & L. Beurer-Kellner, "GitHub MCP exploited: Accessing private repositories via MCP", Invariant Labs, 26 May 2025. Available online: https://invariantlabs.ai/blog/mcp-github-vulnerability.

[6] L. Beurer-Kellner & M. Fischer, "WhatsApp MCP exploited: Exfiltrating your message history via MCP", Invariant Labs, 7 April 2025. Available online: https://invariantlabs.ai/blog/whatsapp-mcp-exploited.

[7] A. Lumelsky, "Critical RCE vulnerability in Anthropic MCP Inspector—CVE-2025-49596", Oligo Security, 27 June 2025. Available online: https://www.oligo.security/blog/critical-rce-vulnerability-in-anthropic-mcp-inspector-cve-2025-49596.

[8] National Institute of Standards and Technology, "CVE-2025-49596 detail", National Vulnerability Database, 2025. Available online: https://nvd.nist.gov/vuln/detail/CVE-2025-49596.

[9] Model Context Protocol, "Sampling", Model Context Protocol specification, version 2025-11-25, 2025. Available online: https://modelcontextprotocol.io/specification/2025-11-25/client/sampling.

[10] Model Context Protocol, "Authorization", Model Context Protocol specification, version 2025-11-25, 2025. Available online: https://modelcontextprotocol.io/specification/2025-11-25/basic/authorization.

[11] Model Context Protocol, "Security best practices", Model Context Protocol documentation, version 2025-11-25, 2025. Available online: https://modelcontextprotocol.io/docs/2025-11-25/tutorials/security/security_best_practices.

[12] N. G. Maloyan & D. E. Namiot, "Breaking the protocol: Security analysis of the Model Context Protocol specification and prompt injection vulnerabilities in tool-integrated LLM agents", Modern Information Technologies and IT-Education **21** (2025) 420. https://doi.org/10.25559/SITITO.021.202503.420-428.

[13] M. M. Hasan, H. Li, E. Fallahzadeh, G. K. Rajbahadur, B. Adams & A. E. Hassan, "Model Context Protocol (MCP) at first glance: Studying the security and maintainability of MCP servers", ACM Transactions on Software Engineering and Methodology (2026), advance online publication. https://doi.org/10.1145/3814959.

[14] X. Hou, Y. Zhao, S. Wang & H. Wang, "Model Context Protocol (MCP): Landscape, security threats, and future research directions", ACM Transactions on Software Engineering and Methodology (2026), advance online publication. https://doi.org/10.1145/3796519.

[15] C. Huang, X. Huang, N. P. Tran & A. Milani Fard, "Model Context Protocol threat modeling and analysis of vulnerabilities to prompt injection with tool poisoning", Journal of Cybersecurity and Privacy **6** (2026) 84. https://doi.org/10.3390/jcp6030084.

[16] X. Li & X. Gao, *A first look at the security issues in the Model Context Protocol ecosystem*, 2026 56th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Charlotte, North Carolina, USA, 22–25 June 2026, pp. 379–392. https://doi.org/10.1109/DSN69566.2026.00046.

[17] H. Song, Y. Shen, W. Luo, L. Guo, T. Chen, J. Wang, B. Li, X. Zhang & J. Chen, "Beyond the protocol: Unveiling attack vectors in the Model Context Protocol (MCP) ecosystem", IEEE Transactions on Software Engineering **52** (2026) 2410. https://doi.org/10.1109/TSE.2026.3694876.

[18] J. T. Halloran, B. Radosevich & G. Black, *MCP safety audit: LLMs with the Model Context Protocol allow major security exploits*, Proceedings of SPIE 14046, Assurance and Security for AI-Enabled Systems 2026, National Harbor, Maryland, USA, 26 April–1 May 2026, Article 140460A. https://doi.org/10.1117/12.3097390.

[19] S. Zhao, Q. Hou, Z. Zhan, Y. Wang, Y. Xie, Y. Guo, L. Chen, S. Li & Z. Xue, "Mind your server: A systematic study of parasitic toolchain attacks on the MCP ecosystem", arXiv:2509.06572 (2025). https://arxiv.org/pdf/2509.06572v1.

[20] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz & M. Fritz, *Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection*, Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, Copenhagen, Denmark, 30 November 2023, pp. 79–90. https://doi.org/10.1145/3605764.3623985.

[21] Q. Zhan, Z. Liang, Z. Ying & D. Kang, *InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents*, Findings of the Association for Computational Linguistics: ACL 2024, Bangkok, Thailand, 2024, pp. 10471–10506. https://doi.org/10.18653/v1/2024.findings-acl.624.

[22] J. Yi, Y. Xie, B. Zhu, E. Kiciman, G. Sun, X. Xie & F. Wu, *Benchmarking and defending against indirect prompt injection attacks on large language models*, Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Vol. 1, Toronto, Canada, 2025, pp. 1809–1820. https://doi.org/10.1145/3690624.3709179.

[23] Q. Zhan, R. Fang, H. S. Panchal & D. Kang, *Adaptive attacks break defenses against indirect prompt injection attacks on LLM agents*, Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, 2025, pp. 7116–7132. https://doi.org/10.18653/v1/2025.findings-naacl.395.

[24] OWASP Foundation, "OWASP Top 10 for LLM applications 2025", 17 November 2024. Available online: https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/.

[25] S. Jamshidi, A. Moradi Dakhel, K. W. Nafi & F. Khomh, "Semantic attacks on tool-augmented LLMs: Securing the Model Context Protocol against descriptor-level manipulation", arXiv:2512.06556 (2025). https://doi.org/10.48550/arXiv.2512.06556.