Sciences Methods and Technologies International Journal (SciMeTech)

ISSN: 3085-5284, journal homepage: www.scimetech.com

Securing the Model Context Protocol: A Systematic Analysis of Attack Surfaces and Defense Gaps

Saad Mansouri, Anas Abouelkalam, Wissam Abbass

Research Laboratory in Sustainable and Intelligent Technologies (LaRTID), Cadi Ayyad University, Marrakech, Morocco

How to cite:

Saad Mansouri, Anas Abouelkalam, Wissam Abbass, "Securing the Model Context Protocol: A Systematic Analysis of Attack Surfaces and Defense Gaps", Sciences Methods and Technologies International Journal (SciMeTech), Vol 2, Issue 2, p 158-169

Keywords:

Model Context Protocol,
Agentic AI Security,
LLM Vulnerabilities
Supply Chain Security
MCP Attacks

Abstract

The emergence of agentic AI has reshaped the trust architecture of modern AI-based applications dramatically by allowing large language models to work with external tools, services, and datasets directly. In this new environment, the Model Context Protocol (MCP) becomes the cornerstone, offering a well-defined structure communication protocol. Within a year after the protocol's public release, the number of public MCP servers exceeded 22,000, and many major platforms, such as Claude Code, and Gemini CLI adopted it as a key integration interface. While this fast adoption reflects the advantages of using MCP, it also demonstrates an inherent contradiction in the basic assumptions behind the creation of the protocol, namely, its lack of security considerations. The initial specification did not include a set of standard security measures that would be needed to mitigate emerging attacks through the use of the protocol itself. As a result, MCP leaves room for numerous new threats based on the mediation of this protocol. As such, securing MCP should go beyond the simple prevention and include advanced threats arising due to security gaps within the protocol itself. In this paper, we present a systematic evaluation of MCP security by synthesizing recent research into a unified taxonomy of six attack families and twenty-three attack vectors. We further review empirical findings on success rates, vulnerability, safety-utility trade-offs, followed by an assessment of the effectiveness of current countermeasures based on a coverage matrix

I. Introduction

The rise of agentic intelligence systems that autonomously interface with external tools, service interfaces, and file systems, has fundamentally reconfigured the trust architecture of modern AI. As a part of its attempt to establish itself as an open and standard protocol for structured exchange of information between LLMs and remote services, the Model Context Protocol (MCP) aims at facilitating the interaction further by introducing a single lifecycle of resource discovery, tool invocation, and context management. MCP abstracts database and sandbox heterogeneity, opening unprecedented prospects for extensible use cases of agent systems based on LLMs.

Since its debut in 2024, the MCP ecosystem has experienced rapid development, with over 22,000 publicly available servers by April 2026 [2]. The ecosystem has grown due to its wide implementation in various platforms like Claude code, Cursor, and Gemini CLI, which has cemented MCP's status as a primary protocol for integration.

Despite all this progress, the rapid adoption has revealed the underlying contradiction in the MCP design principles, despite its aim at seamless interoperability, the original MCP design included no native security measures. Delegating responsibility for authentication, encryption, and content validation to individual implementations has created a considerable attack surface, making MCP particularly vulnerable to protocol-specific attacks, such as tool poisoning, context injection, and privilege escalation through cross-context execution. As such, securing MCP requires addressing not just simple bugs in code, but also more sophisticated threats such as reasoning, privilege, and cross-tool misuse, insecure composition, and supply-chain attacks [8, 10, 11, 22].

Numerous studies have considered the issue of MCP security from different perspectives, ranging from attack demonstration, server-scale analysis, benchmarking construction, defense approaches, and threat modeling. However, although this literature is rich in valuable insights, it is extremely fragmented because existing attack studies have focused

on specific attack vectors, defensive approaches have been tested over only a subset of attacks, yet no work has comprehensively analyzed the current landscape of research on MCP security.

We aim to cover the gaps with our contributions in this paper:

- Unifying taxonomy and organizing attack types across six architectural layers, offering a structured framework that subsumes and reconciles the competing categorizations in the literature.
- Empirically synthesizing results from multiple studies on the use of language models and real-world servers. This shows that more powerful models can be more susceptible to certain adversarial attacks, while security improvements usually come with concrete system performance costs.
- Reviewing the defense coverage matrix by assessing the suggested defenses in relation to six attack families and twenty-three attack types. Then organize solutions into a defense-in-depth model extending from registries to hosts, protocol layers, servers, and runtime monitoring.
- Analysing unresolved deficiencies to identify open challenges and a future plan for more secure MCP ecosystems.

The rest of this paper will be arranged in the following: the second section will provide background about MCP architecture and security-adoption tradeoff, followed by presenting taxonomy of MCP-related threats in the third section. Section four will summarize empirical findings, while section five will evaluate the defense environment and security gaps will be listed in section six. We will finish by discussing implications and future research in section seven, and conclude in the last section.

II. Background

a. MCP (Model Context Protocol)

Before the introduction of MCP, intelligent applications used to communicate with external tools through manual API wiring, or custom plugin-based interfaces such as ChatGPT Plugins, these approaches required individual service-to-API integrations, making the system difficult to scale and complex to maintain [3]. In order to address all downsides of traditional implementation, Anthropic introduced the Model Context Protocol (MCP) in the last quarter of 2024 [1], this new communication protocol represents a paradigm shift in AI system architecture, establishing an open standardized framework for connecting LLMs to dynamic external capabilities and data sources, as a vendor-neutral specification, MCP fundamentally rethinks how AI-based applications access and interact with their operational environment.

MCP provides a clear interface between AI-based reasoning and its external implementation owing to its declarative and introspective architecture. As opposed to conventional integrations that are hardcoded into applications themselves, MCP enables a dynamic discovery system where capabilities are advertised and invoked through a standardized protocol.

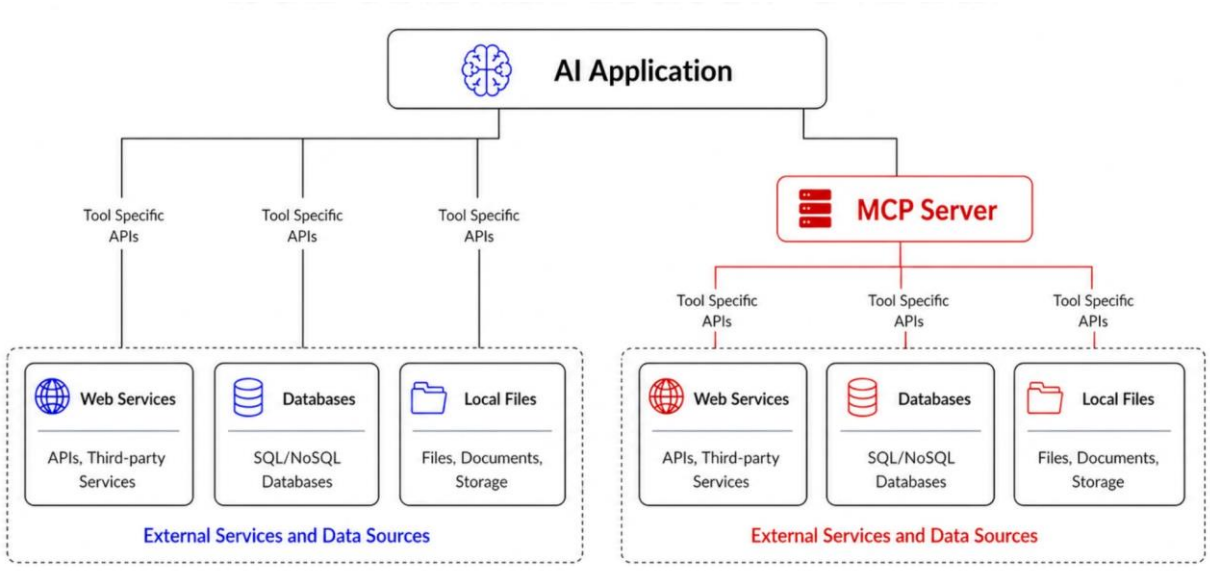

Fig1. Architectural representation of communication between an AI application and external resources pre-MCP and post-MCP

As a client-server protocol, MCP operates through three interconnected components, each representing a distinct security boundary:

- **Host:** represents the user-facing AI application or agent environment that receives user requests and orchestrates task execution, includes systems such as Claude, ChatGPT, and other AI-enhanced IDEs, which provide interfaces that are both natural-language driven and software-defined.
- **Client:** protocol implementation within the host that manages connections to one or more servers using JSON-RPC for communication over two transport modes, stdio for local inter-process, and Streamable HTTP with Server-Sent Events (SSE) for remote procedure calls.
- **Server:** independently developed process that exposes to the host tools, resources, and prompts as main capabilities.

Accordingly, this architectural design enables MCP to serve as a universal adapter for AI-based applications, extending an agent with file access, web retrieval, repository operations, enterprise APIs, and domain-specific workflows, and facilitating seamless integration without custom connector development [3].

At a high level, a typical MCP interaction has four stages. First, the host discovers and registers a server. Second, the client retrieves the server's advertised metadata, including tool names, descriptions, schemas, and sometimes usage guidance. Third, the LLM decides whether a tool is needed and how to invoke it. Fourth, the tool returns outputs that may feed back into the shared context and affect future decisions. Security concerns arise at each stage.

b. Security-adoption tradeoff

From a security perspective, the most critical characteristic in MCP protocol architecture is that tool descriptions are loaded into the context of LLM verbatim. The host cannot semantically validate such descriptions in any way before they affect the model's behavior. This design, where natural language becomes the interface contract, is what enables the flexibility of MCP but is also the very foundation of its most dangerous category of attacks.

Firstly, the protocol distinguishes itself from previous iterations of API integration patterns, at least in four ways:

1. Natural-language metadata becomes part of the attack vector. Tool name and description become active, directly affecting the model's planning and selecting behavior [10, 9, 4, 22].
2. Shared context introduces the possibility of content propagation. Malicious content delivered by one server and/or one tool invocation can propagate into the context of the host and affect further actions concerning other tools [10, 12, 5].
3. Use of third-party servers increases security perimeter. The host usually runs the tools created not only by internal teams but by outside vendors and community-registered providers as well [3, 13, 5, 19].
4. Composability allows chaining of multiple operations in the sequence. To corrupt the agent's behavior, an attacker does not need one all-powerful malicious tool. One could manipulate the benign looking tool to guide the agent to perform a sequence of malicious actions across several tools [12, 5, 20].

As can be observed, these characteristics prevent the security issue from being reduced to a classical software flaw. MCP couples machine reason, natural language, and third-party code execution together in the same loop of operations. Therefore, both protocol artifacts and the corresponding model behaviors should be considered first-class security objects.

These vulnerabilities are not a matter of coincidence, they arise as a consequence of the design approach chosen when developing the MCP. As it focused on rapid and widespread adoption, it made certain key design decisions, which helped achieve this goal but also became the root of MCP's vulnerabilities. Starting from fundamental specification that defines **no normative security requirements**, neither mandatory authentication, nor transport security, or content validation. All the security practices are left to be implemented and configured separately by individual implementations. Moreover, within the concept of an open ecosystem, the developers of the MCP are not able to exercise control over the published MCP servers; anyone is allowed to post an MCP server through aggregations with no review process. Indeed, according to Song et al., three major aggregation services accepted, without review, a malicious MCP service and reported them as scanned and safe [8]. Lastly, the protocol's **natural language interface**, with its freedom of expression, makes it possible to inject malicious commands hidden from users but understandable to the language model. All the categories of attacks described in the next sections arise from these design decisions.

III. MCP-Related Threats

Based on an integration of multiple viewpoints from existing literature, we have constructed a comprehensive classification scheme for MCP threats that encompasses six main families and 23 attack vectors.

a. **Supply Chain Attacks** involve attacking the MCP distribution pipeline, thus affecting the system before any protocol-level interactions begin.

Table 1. Supply Chain Attacks family

Attack Type	Description	Key Finding
Tool Poisoning (Basic)	Malicious instructions embedded in tool description fields	First demonstrated by [27]; 5.5% of real servers affected [13]
Rug Pull Attack	Post-deployment server update replaces legitimate functionality with malicious code	Enabled by npx/uvx package managers that fetch fresh code at every launch; invisible to pre-deployment scanning
Typosquatting	Malicious server registered under a name similar to a trusted server	Structurally identical to npm typosquatting; semantic-level impact inside LLM context
Malicious Tool Coverage	Tool description claims a legitimate co-installed tool is deprecated	80-100% ASR; LLM defers to most recently stated operational guidance [10]
Preference Manipulation	Promotional language in tool metadata biases LLM tool selection	Explicit manipulation: 100% ASR; Genetic-based Advertising Preference Manipulation Attack (GAPMA) stealthy variants maintain high ASR while evading filters [3]

b. **Protocol and Transport Attacks** rather than manipulating the LLM, these exploits compromise the session and network layers used for transport:

Table 2. Protocol and Transport Attacks family

Attack Type	Description	Key Finding
Man-in-the-Middle (MitM)	Network-level interception and modification of MCP traffic	100% ASR against all tested platforms; neither MCIP nor FAN provide mitigation [6]
DNS Rebinding	Browser-based attack bypassing same-origin policy to interact with local MCP servers	100% ASR against all tested platforms [6]
Session Hijacking	Unauthorized takeover of active MCP sessions lacking cryptographic binding	Enabled by the absence of mandatory nonce-based session binding in the MCP specification
Replay Attack	Re-execution of captured valid MCP messages	Structural consequence of stateless JSON-RPC without sequence numbers

c. **Tool-metadata Attacks** occur before tool invocation and target the LLM's interpretation of tool metadata using deceptive descriptions, schemas, parameters, or instructions to bypass the model's ability to verify authoritative sources

Table 3. Tool-metadata Attacks family

Attack Type	Description	Key Finding
Tool Shadowing	Malicious tool instructs LLM to invoke it alongside/instead of a coinstalled legitimate tool	80-90% ASR [10]
Tool Name Conflict	Two tools with conflicting names cause LLM to invoke incorrect tool	Demonstrated in weather/fetch/search configurations
Identity Injection	Tool description instructs the LLM to adopt a false identity or role	100% ASR across all 13 tested LLMs in MCP-SafetyBench; no model provides any resistance [21]
Out-of-Scope Parameter (OP)	Tool description instructs the LLM to pass parameters outside the declared schema	76.5% average ASR - the highest of any attack type in the MSB evaluation [20]

Retrieval-Agent Deception (RADE) Attack	Retrieval-Agent Deception: adversarial payload planted in a document, retrieved via a legitimate tool, executes in the agent's reasoning loop	Credential theft via ~/.cursor/mcp.json demonstrated in production configurations [7]

d. Tool-return Attacks occur after tool invocation and bypass initial validation by planting malicious payloads directly through the data returned after the tool has already been cleared to run.

Table 4. Tool-return Attacks family

Attack Type	Description	Key Finding
Tool-return Injection	Malicious instructions embedded in tool output, processed by LLM post-execution	Specifically defeats pre-execution validation; agent reasons over poisoned output as trusted data [10]
Indirect Prompt Injection via Resource	Adversarial content in external resource retrieved by a legitimate tool	93.33% ASR - highest tested vector in multi-vector study [8]
Cross-Server Data Exfiltration	Malicious server instructs LLM to read from another installed server's resources	No protocol-native isolation between servers; GBP 1,469.36 real money exfiltrated with less than 25 lines of code [12]

e. Execution Environment Attacks target the server execution environment and processes to take control of execution context.

Table 5. Execution Environment Attacks family

Attack Type	Description	Key Finding
Command Injection	Unsolicited input passed to shell commands in server implementation	43% of tested MCP servers susceptible [16]; CVE-2025-6514 achieves 100% Remote Code Execution (RCE)
Configuration Abuse	Malicious configuration (like Docker host-root mount) for privilege escalation	100% ASR across all host-LLM combinations [11]
Parasitic Toolchain Attack	Multi-step chain composing External Ingestion, Privacy Access, Network Access capabilities	8.7% of tools (1,062/12,230) participate in exploitable chains; 9/10 constructed chains successfully exfiltrated data [5]
Initialization-Phase Attack	Malicious action triggered at server startup, before any LLM reasoning occurs	6 of 12 attack categories fire at 100% ASR before LLM engagement; immune to all semantic defenses

f. Cross-Protocol Attacks are conducted on the vulnerabilities discovered at the point of interaction between MCP and other protocols such as REST APIs or Agent-to-Agent (A2A).

Table 6. Cross-Protocol Attacks family

Attack Type	Description	Key Finding
A2A Discovery-Layer Poisoning	Agent's tool discovery process compromised, registering malicious servers as trusted peers in A2A hierarchy	Undefined intersection of A2A and MCP trust models [26]
Cross-Protocol Trust Chain Gap	Neither A2A nor MCP security model applies at the protocol intersection	Enables cascading malicious instruction propagation across multi-agent networks
Multi-Agent Instruction Propagation	Compromise at one delegation node propagates across entire agent network	Forward-looking threat; empirical validation pending scale of production multi-agent deployment

IV. Empirical findings

a. Inverse scaling

The literature consistently highlights a counterintuitive behavior, it can be defined as an 'Inverse Scaling' law essentially to describe that increasing a model's capability seems to open more doors for MCP-based attacks rather than closing them.

Table 7. Studies define 'Inverse Scaling' law

Study	Evidence
MCPTox [9]	Claude-3.7-Sonnet achieves maximum refusal rate of only 2.7%; chain-of-thought reasoning mode adds +27.8 pp average ASR
MSB Benchmark [20]	DeepSeek-V3.1: 60.94% ASR, GPT-4o-mini: 58.56%, Llama3.1-8B: 19.74%
Song et al. [8]	Gemini 2.5 Pro: 98.33% ASR for tool poisoning vs. GPT-4o: 33.33%
MCP-SafetyBench [21]	Safety-utility trade-off Pearson r = -0.572 (p = 0.041); statistically significant inverse correlation

According to MCPTox, while chain-of-thought (CoT) is a distinctive mark of advanced models, it actually boosts ASR by 27.8 percentage points because the deeper processing causes the model to 'over-rationalize' and follow malicious instructions. This means that the deployment trend toward more capable, reasoning-enabled models directly worsens the security posture of MCP deployments.

b. Attack Success Rate (ASR)

The following table summarizes the most important quantitative ASR key measurements across the literature:

Table 8. Attack Success Rate

Attack Type	Success rate
Identity Injection	100% ASR across all 13 LLMs [21]
Configuration Abuse	100% ASR across all host-LLM combos [11]
MitM / DNS Rebinding	100% ASR on all tested platforms [6]
Indirect Resource Injection	93.33% ASR [8]
Tool Shadowing	80-90% ASR [10]
Malicious Tool Coverage	80-100% [10]
Out-of-Scope Parameter	76.5% average [20]
Preference Manipulation (explicit)	100% ASR [9]
Parasitic Toolchain (9/10 live)	90% ASR [5]
Tool Poisoning (MCPTox, implicit)	46.7% average (peak 98.33%) ASR [25]

c. Vulnerability prevalence

Hasan et al. provide the only large-scale census of the real-world server ecosystem (1,899 servers across GitHub, mcp.run, Smithery) [13]:

- 7.2% of servers contain conventional code-level security vulnerabilities (injection flaws, hardcoded secrets)
- 5.5% show evidence of tool poisoning in their descriptions
- **Median maintainability:** 1 contributor, many abandoned after initial publication, creating structural rug pull risk without malicious intent
- SonarQube detects only 3 of 8 MCP-specific vulnerability patterns; general-purpose static analysis is insufficient

At 16,000+ servers and a 5.5% tool poisoning rate, this implies approximately 880 potentially malicious servers currently available in the public ecosystem with no systematic detection mechanism.

d. The Safety Prompt Backfire effect

Surprisingly, recent security evaluations show that safety prompts can backfire. Instead of lowering the success rate of attacks, these prompts occasionally ended up increasing it [21, 4]:

- **Average effect of safety prompts**: -1.22% ASR reduction (not statistically significant, p = 0.29)
- **Preference Manipulation attacks:** +7.34% ASR with safety prompts
- **Function Overlapping attacks**: +9.36% ASR with safety prompts

Instead of protecting the model, safety prompts prime it to engage with manipulative logic. This actually heightens the risk from social-proof language, effectively ruling out the idea that simple prompt engineering can patch these security gaps.

e. Over-Approval bias

MCIP reports the over-approval bias as a structural failure mode in which the models are trained to align their function calls for efficient tool invocation with no checks regarding the context-based safety of operations. It means that even the best-performing undefended baseline as DeepSeek-R1 is only capable of reaching 67.37% of binary safety awareness level. However, this is not misconfiguration but the outcome of regular function-calling fine-tuning rewarding tool invocation [17].

f. Domain risk stratification

First of all, studies reveal that the success rates are not consistent across deployment domains. For instance, while the lowest success rates in terms of attacks were found in Web Search with ASR 30.33%, the highest ASR was registered in Financial Analysis and Code Execution with ASR values of 46.59% and 38.47%, correspondingly. Moreover, the results of analysis of variance (ANOVA) confirm the difference ($F = 6.68$, $p = 0.000163$). Thus, the above-described phenomenon demonstrates that the domains in which the agentic MCP implementation would be most beneficial are those with the highest success rate of attacks. At the same time, contemporary defense tools do not have the required granularity to handle these risks

g. Host client vulnerability

Empirical analysis of recent trends indicates the importance of host clients as another independent target receiving little attention to date [6, 5]. To be precise, out of seven primary MCP clients only two are capable of implementing static validation of tool descriptions. In the meantime, the existence of CVE-2025-6514 suggests the RCE (Remote Code Execution) vulnerability with 100% ASR of the MCP client used in the study with no possible mitigations. Besides, comparison of Cursor and fast-agent revealed that the host client execution environment played an important role in vulnerability. Despite identical model use in both applications, Cursor demonstrated 100% susceptibility to initialization phase attacks compared to fast-agent's immune host client (0% ASR).

V. Defense landscape

Taking into account the results obtained by recent research, vulnerabilities cover all aspects of MCP lifecycle, including server discovery and response handling, and require a multi-faceted solution [16, 18, 21, 24]. Thus, it became essential to adopt a new defense-in-depth strategy, where protection involves building up layers where security becomes a systematic consideration at each stage of the lifecycle, beginning with its very first step with the **supply chain**. Using carefully chosen registries and signed packages, we are able to prevent malicious servers from reaching users at all [13, 11, 22]. Even though reputation scoring and static scanning are not entirely infallible, they dramatically increase the barrier for attackers and screen away clear warnings, such as over-privileged servers and suspicious imports.

With the server already in the system, we turn to our host, which has become recognized in recent research as a critical filter [16, 19, 22, 18]. **Host-side mediation** should include both the down-ranking of suspect metadata and "least privilege" practices, according to which a tool will see only what is absolutely necessary for the job, have risky operations confirmed beforehand, and be confined to an allowlist.

In addition to ad hoc filtering, recent trends and academic findings point towards **protocol hardening**, implying that the adoption of protocol standards is imminent [19]. Proposed protocols, such as SMCP, require mutual authentication and secure sessions with an audited security context [19]. This shift in the structure of the problem is also supported by the establishment of authoritative centralized repositories, such as Google's Agent Registry [28]. This follows the same move as adaptation of MCIP, which suggests a more focusing approach to evaluating tool invocations against contextual integrity, to ensure information flow appropriateness in each particular case [17].

While the above measures help us prevent known attacks, the runtime is still too unpredictable for static code analysis, which is why **runtime inspection** should follow. For instance, MCPShield proposes to use metadata-driven and interactive probing along with adaptive trust calibration in order to assess the safety of third-party servers over time [18]. MCP-SandboxScan demonstrates that running a tool inside a WASM/WASI environment allows for finding potentially harmful data-flow channels without placing trust in the provider [24]. Ultimately, as benchmarks like MCPTox and MCP-SafetyBench have proven, even the sophisticated models are not necessarily safe [7, 20, 21, 6]. As attacks continue evolving, security should be considered as an ongoing process that requires implementing the **assurance cycle** with red-teaming, benchmark-based regression testing, and attack replays, along with monitoring after deployment.

Drawing from the literature reviewed, we systematically classify these 13 defense mechanisms into five defensive approaches:

1. Static pre-deployment scanning: Automated analysis of code or model configurations prior to implementation to identify vulnerabilities. (eg: SonarQube-based [13], AI-Infra-Guard [25], mcp-scan [27])

2. Runtime semantic proxies: Intermediary layers that filter or monitor the meaning of requests in real-time to prevent malicious intent. (eg: MCP-Guard [15], MCPShield [18]).

3. Lightweight middleware: Minimalist software components integrated directly into the stack to manage local security logic (eg: MCP Guardian [16]).

4. Alignment/Training defenses: Methods that modify the underlying model's behavior through specialized fine-tuning or instruction-following constraints (e.g., MCIP [17]).

5. Protocol/Architecture reforms: Fundamental changes to the underlying communication standards or system design to ensure secure by default operation (e.g., SMCP [3]).

To evaluate the proposed defenses against the six attack families, we define in this matrix the coverage for each defense:

- **Full (F) :** directly mitigates the attack family with evidence or implementation
- **Partial (P) :** reduces risk but does not fully prevent the attack or only covers a subset of attack vectors.
- **Negligible (N) :** indirect or weak relationship to the attack family, with no meaningful demonstrated mitigation.
- **None (0) :** No coverage or no sufficient evidence identified.

Table 9. the proposed defenses against the six attack families

Defense	Supply chain	Transport	Tool-metadata	Tool-return	Execution	Cross-Protocol
mcp-scan	P (4/120)	0	P	0	0	0
MCPGuard	P	0	N	0	P	0
MCP-Guard (Xing)	P	0	F	P	N	0
MCPShield	P	0	F	P	N	0
MCP Guardian	N	0	N	0	P	0
MCIP	N	0	P	P	0	0
MCP-SandboxScan	0	0	0	P	F	0
MCP-Gateway	N	P	P	N	N	0
SMCP (proposed)	F*	F*	F*	F*	F*	P*
Narajala ZTA (Zero Trust Architecture)	P	0	P	N	P	0
MCPSecBench formal	N	P	P	N	N	0
MCIP+FAN	N	0	P	P	0	0
SonarQube	0	0	N	0	P	0

*SMCP coverage claims are theoretical (no empirical validation).

As a key finding, 11 of 13 defenses fail to secure the connection or monitor the network traffic for threats, providing zero transport-layer coverage. MCP-Gateway offers partial coverage; only SMCP claims full coverage, but SMCP has no empirical validation.

VI. Security gaps

We identified eight security gaps that are unaddressed by any current mechanism. These represent the highest-priority targets for future security research:

1. **Transport-layer protection:** Man-in-the-Middle (MitM) and DNS Rebinding attacks against MCP achieve 100%

ASR on all tested platforms with no available mitigation from the application-layer defenses that comprise the bulk of the literature. The MCP specification does not mandate TLS, mutual authentication, or nonce-based session binding. Closing this gap requires changes to the protocol specification, not the application layer.

2. **Initialization attacks:** Six of twelve attack categories in Zhao et al. taxonomy belong to a class of initialization-firing attacks that trigger during server setup, meaning they complete their malicious action during

server initialization, before the LLM processes any tool descriptions. These attacks, by construction, are immune to every semantic defense, alignment improvement, and safety training approach in the literature, leaving a gap that no current defense addresses.

3. **Protocol consent enforcement gap:** The MCP specification treats user consent as client-recommended, not protocol-mandatory. There is nothing in the specification that prevents a server from bypassing consent prompts, and nothing that prevents the LLM from acting before a user can respond. Song et al. showed that 75% of users selected at least one malicious server without warning, when told malicious servers were present, only 1 in 20 identified all four correctly [8]. Combined with the human factor findings (users cannot reliably detect malicious servers even when warned), the current consent mechanism is effectively advisory theater. Security must be automated.

4. **Over-approval bias in function-calling alignment:** Standard function-calling fine-tuning trains models to execute tool invocations efficiently, creating a systematic blind spot for contextual safety checking (over-approval bias). The best undefended baseline (DeepSeek-R1) achieves only 67.37% binary safety awareness. This is not a deployment configuration issue, it is a training objective issue that persists regardless of how the model is deployed.

5. **Ecosystem governance and server identity:** No cryptographic server identity mechanism exists. Any server can claim any identity, describe any capabilities, and be registered on aggregation platforms without review. The Trusted Component Registry proposed in SMCP addresses this architecturally, but no implementation exists and no ecosystem-wide coordination mechanism has been proposed.

6. **Rug pull detection gap:** Post-deployment server updates are invisible to all pre-deployment scanning tools. Because npx/uvx-distributed servers fetch fresh code at every launch, any repository update takes effect immediately with no user notification. MCPShield's Periodic Reasoning is the only mechanism addressing this, and it works at the behavioral level (detecting changed invocation patterns) rather than the code level.

7. **Safety-Utility Trade-off at Semantic Defenses:** The statistically confirmed safety-utility trade-off means that every semantic defense reduces task performance proportionally to ASR reduction. The Net Resilient Performance (NRP) metric [20] shows that most current defenses provide near-zero net improvement when task degradation is accounted for. This is a fundamental constraint on application-layer defenses that operate by filtering or second-guessing tool invocations.

8. **Multi-Agent and Cross-Protocol Trust Chains:** As production AI systems increasingly combine multiple agents with cross-protocol communication (A2A + MCP), trust chain gaps at protocol intersections create attack surfaces not present in single-agent deployments. These attacks are currently theoretical [26], but will become empirically relevant as multi-agent deployments scale.

As such, the following are the required measures to solve the above problems and bridge the identified gaps:

1. Mandatory mutual TLS coupled with server-side certificate pinning along with nonce-based session binding in the base MCP standard.

2. Initialization execution under sandboxing with capability-based resource access (MCP-SandboxScan applied to initialization execution).

3. Consent enforcement as protocol requirement with cryptographic acknowledgment before any tool invocation modifies persistent state.

4. Awareness research should be conducted with focus on the calling functions as an objective. Specific benchmarks such as MCP-SafetyBench and MSB are good standards for evaluating awareness.

5. Decentralized registry of identities for MCP servers where all entries are cryptographically signed, and only aggregated platforms providing verified signatures would be included.

6. Version pinning along with invocation and cryptographic verification of updates followed by behavioral analysis as implemented in MCPShield for detecting behavioral changes after deployment.

7. Protection techniques that offer defense without reducing task performance, which likely requires operating below the semantic layer (transport security, execution sandboxing, capability-based access control) rather than semantic filtering.

8. Formal trust model for multi-protocol agent systems specifying how trust properties established in A2A are inherited (or not) in MCP tool invocations.

VII. Discussion and future research directions

It appears that a layered systems approach to MCP security, rather than a single-model alignment, is warranted based on the literature. Most of the mature attack work shows multi-step manipulation, economic steering, and even privacy

breaches between different tools [12, 9, 4, 5]. Likewise, the current best defense efforts go beyond keyword filtering and attempt to protect in registry space, host space, protocol context, and runtime execution [16-22, 24].

From this viewpoint, the near-future research plan could follow three directions:

First, it is necessary to establish a common benchmark for the field in a **shared evaluation reference**. While some benchmarks do exist, the field still appears to be partially fragmented in terms of attack families used, servers, and measurement methods. A complete benchmark suite covering success on benign tasks, success of attacks, false positives, latency overhead, and cross-tool attacks would facilitate better comparison of defenses [9, 20, 21]

Second, research effort should go toward developing **policy-aware MCP execution**. Hosts should not expose their tools simply as capabilities. Tools should carry their trust level, scope of operation, confirmation policies, and even a provenance tag to allow the host to make the decision on whether it should give the tool to the model at all, whether the given argument is valid for it, and whether it should treat the output as data rather than as commands [17, 18, 19, 22].

Finally, **security-by-design** is required in the future. Possible paths could be typed capabilities, a formal approach to information flow between different tools, verification of server metadata, session confidentiality, and runtime taint tracking for tool outputs [19, 24]. These mechanisms would help transform MCP from an open but weakly governed integration layer into a more trustworthy substrate for agentic systems.

This work primarily focuses on systematic synthesis rather than proposing an experimental validation or implementation of a new defense mechanism. As with any synthesis study, it has several limitations. Firstly, given the novelty of MCP security, some results are based on preprints, benchmark studies, and industry reports whose conclusions may evolve as the ecosystem matures. Secondly, while the defense coverage matrix uses explicit criteria, its assessment remains qualitative and depends on the available evidence reported by each source. Finally, certain cross-protocol and multi-agent risks remain partly conceptual, as large-scale production deployments are still limited. These limitations should be considered when interpreting the proposed taxonomy, defense coverage analysis, and future research directions.

In our view, future MCP research should answer five concrete questions:

1. **How can hosts enforce contextual least privilege without making agents unusably brittle?**
2. **How should provenance and trust metadata be represented so that both models and host policies can use them?**
3. **What defenses best stop multi-tool and multi-turn attacks rather than single-call attacks?**
4. **How can registries and maintainers scale vetting and revocation across a rapidly growing server ecosystem?**
5. **Which protocol changes are worth standardizing now, before insecure deployment patterns become entrenched?**

VIII. Conclusion

In this work, we dive deep on how the MCP changed the cybersecurity landscape. We start by introducing the MCP architecture and explaining how hosts, clients, and servers create new trust boundaries. Then we list MCP-related threats, from supply-chain attacks to tool-return and cross-protocol attacks. Moreover, this paper reviewed empirical findings from recent studies, including the attack success rates, vulnerability of stronger models to some tool-based attacks, and the prevalence of insecure or poisoned MCP servers in public ecosystems.

Furthermore, we examined existing mitigation strategies such as static scanning, runtime monitoring, defense-in-depth middleware, contextual integrity approaches, and secure protocol redesign. The analysis not only shows that current defenses remain incomplete, especially for initialization-phase, and transport-layer attacks, but also concludes that MCP security cannot be solved only through prompt-level safeguards or model alignment.

Ultimately, we recommend for future progress to work on protocol-native transport security, supply-chain governance, a training paradigm that teaches models to resist metadata-injected instructions, a coordinated multi-layer defense, empirical study of multi-agent attack propagation, and formal verification of proposed defenses. Six main aspects not only incremental improvements, but essential requirements for enabling trustworthy deployment of MCP-based agentic systems in real-world environments.

References

[1] Anthropic. 2024. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol.

[2] Glama, (2026, April). Open-source MCP servers registry. https://glama.ai/mcp/servers.

[3] Hou, X., Zhao, Y., Wang, S., & Wang, H. (2025). Model context protocol (MCP): Landscape, security threats, and future research directions. ACM Transactions on Software Engineering and Methodology.

[4] Wang, Z., Zhang, R., Liu, Y., Fan, W., Jiang, W., Zhao, Q., ... & Xu, G. (2026, March). MPMA: Preference manipulation attack against model context protocol. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 40, No. 42, pp. 35838-35846).

[5] Zhao, S., Hou, Q., Zhan, Z., Wang, Y., Xie, Y., Guo, Y., ... & Xue, Z. (2025). Mind your server: A systematic study of parasitic toolchain attacks on the MCP ecosystem. arXiv preprint arXiv:2509.06572.

[6] Yang, Y., Gao, C., Wu, D., Chen, Y., Li, Y., & Wang, S. (2025). MCPSecBench: A systematic security benchmark and playground for testing Model Context Protocols. arXiv preprint arXiv:2508.13220.

[7] Radosevich, B., & Halloran, J. (2025). Mcp safety audit: LLMs with the model context protocol allow major security exploits. arXiv preprint arXiv:2504.03767.

[8] Song, H., Shen, Y., Luo, W., Guo, L., Chen, T., Wang, J., ... & Chen, J. (2025). Beyond the protocol: Unveiling attack vectors in the model context protocol ecosystem. arXiv preprint arXiv:2506.02040.

[9] Wang, Z., Gao, Y., Wang, Y., Liu, S., Sun, H., Cheng, H., ... & Li, X. (2026, March). MCPTox: A Benchmark for Tool Poisoning on Real-World MCP Servers. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 40, No. 42, pp. 35811-35819).

[10] Guo, Y., Liu, P., Ma, W., Deng, Z., Zhu, X., Di, P., ... & Wen, S. (2025). Systematic analysis of MCP security. arXiv preprint arXiv:2508.12538.

[11] Zhao, W., Liu, J., Ruan, B., Li, S., & Liang, Z. (2025). When MCP servers attack: Taxonomy, feasibility, and mitigation. arXiv preprint arXiv:2509.24272.

[12] Croce, N., & South, T. (2025). Trivial Trojans: How Minimal MCP Servers Enable Cross-Tool Exfiltration of Sensitive Data. arXiv preprint arXiv:2507.19880.

[13] Hasan, M. M., Li, H., Fallahzadeh, E., Rajbahadur, G. K., Adams, B., & Hassan, A. E. (2025). Model context protocol (MCP) at first glance: Studying the security and maintainability of MCP servers. arXiv preprint arXiv:2506.13538.

[14] Narajala, V. S., & Habler, I. (2026, February). Enterprise-grade security for the model context protocol (MCP):
Frameworks and mitigation strategies. In 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC) (pp. 1-8). IEEE.

[15] Xing, W., Qi, Z., Qin, Y., Li, Y., Chang, C., Yu, J., ... & Han, M. (2025). MCP-Guard: A Multi-Stage Defense-in-Depth Framework for Securing Model Context Protocol in Agentic AI. arXiv preprint arXiv:2508.10991.

[16] Kumar, S., Girdhar, A., Patil, R., & Tripathi, D. (2025). Mcp guardian: A security-first layer for safeguarding mcp-based AI system. arXiv preprint arXiv:2504.12757.

[17] Jing, H., Li, H., Hu, W., Hu, Q., Heli, X., Chu, T., ... & Song, Y. (2025, November). Mcip: Protecting MCP safety via model contextual integrity protocol. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (pp. 1177-1194).

[18] Zhou, Z., Zhang, Y., Cai, H., Aloqaily, M., Bouachir, O., Pang, L., ... & Wen, Q. (2026). MCPShield: A security cognition layer for adaptive trust calibration in Model Context Protocol agents. arXiv preprint arXiv:2602.14281. [19] Hou, X., Wang, S., Zhang, Y., Xue, Z., Zhao, Y., Fu, C., & Wang, H. (2026). SMCP: Secure Model Context Protocol. arXiv preprint arXiv:2602.01129.

[20] Zhang, D., Li, Z., Luo, X., Liu, X., Li, P., & Xu, W. (2025). MCP Security Bench (MSB): Benchmarking Attacks Against Model Context Protocol in LLM Agents. arXiv preprint arXiv:2510.15994.

[21] Zong, X., Shen, Z., Wang, L., Lan, Y., & Yang, C. (2025). MCP-SafetyBench: A Benchmark for Safety Evaluation of Large Language Models with Real-World MCP Servers. arXiv preprint arXiv:2512.15163.

[22] Rostamzadeh, M., Narula, S., Birhan, N., Ghasemigol, M., & Takabi, D. (2026). MCP-DPT: A Defense-Placement Taxonomy and Coverage Analysis for Model Context Protocol Security. arXiv preprint arXiv:2604.07551.

[23] Huang, C., Huang, X., Tran, N. P., & Fard, A. M. (2026). Model Context Protocol Threat Modeling and Analyzing Vulnerabilities to Prompt Injection with Tool Poisoning. arXiv preprint arXiv:2603.22489.

[24] Tan, Z., Hao, R., Singer, J., Tang, Y., & Anagnostopoulos, C. (2026). MCP-SandboxScan: WASM-based Secure Execution and Runtime Analysis for MCP Tools. arXiv preprint arXiv:2601.01241.

[25] Wang, B., Liu, Z., Yu, H., Yang, A., Huang, Y., Guo, J., ... & Wu, H. (2025). Mcpguard: Automatically detecting vulnerabilities in MCP servers. arXiv preprint arXiv:2510.23673.

[26] Li, Q., & Xie, Y. (2025). From glue-code to protocols: A critical analysis of a2a and MCP integration for scalable agent systems. arXiv preprint arXiv:2505.03864.

[27] Beurer-Kellner, L., & Fischer, M. (2025). Mcp security notification: Tool poisoning attacks. Invariant Labs Blog. https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks

[28] Google Cloud. (2026, April 21). Agent Registry Documentation: Overview. Google Cloud Documentation.
https://docs.cloud.google.com/agent-registry/overview