Theoretical and Applied Aspects of Cybersecurity of Convergent Cognitive Information Technologies

https://doi.org/10.25559/SITITO.021.202503.420-428
UDC 004.056

Breaking the Protocol: Security Analysis of the Model Context Protocol Specification and Prompt Injection Vulnerabilities in Tool-Integrated LLM Agents

N. G. Maloyan*, D. E. Namiot

Lomonosov Moscow State University, Moscow, Russian Federation

Address: 1 Leninskie gory, Moscow 119991, GSP-1, Russian Federation

* maloyan.narek@gmail.com

Original article

Abstract

The Model Context Protocol (MCP) has emerged as a de facto standard for integrating Large Language Models with external tools, yet no formal security analysis of the protocol specification exists. We present the first rigorous security analysis of MCP's architectural design, identifying three fundamental protocol-level vulnerabilities: (1) absence of capability attestation allowing servers to claim arbitrary permissions, (2) bidirectional sampling without origin authentication enabling server-side prompt injection, and (3) implicit trust propagation in multi-server configurations. We implement ProtoAmp, a novel framework bridging existing agent security benchmarks to MCP-compliant infrastructure, enabling direct measurement of protocol-specific attack surfaces. Through controlled experiments on 847 attack scenarios across five MCP server implementations, we demonstrate that MCP's architectural choices amplify attack success rates by 23–41% compared to equivalent non-MCP integrations. We propose AttestMCP, a backward-compatible protocol extension adding capability attestation and message authentication, reducing attack success rates from 52.8% to 12.4% with median latency overhead of 8.3ms per message. Our findings establish that MCP's security weaknesses are architectural rather than implementation-specific, requiring protocol-level remediation.

Keywords: Model Context Protocol, prompt injection, protocol security, LLM agents, formal analysis

Conflict of interests: The authors declares no conflict of interest.

For citation: Maloyan N.G., Namiot D.E. 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.* 2025;21(3):420-428. https://doi.org/10.25559/SITITO.021.202503.420-428

Нарушение протокола: анализ безопасности спецификации протокола контекста модели и уязвимости внезапного внедрения в агентах LLM, интегрированных в инструменты

Н. Г. Малоян*, Д. Е. Намиот

Оригинальная статья

ФГБОУ ВО «Московский государственный университет имени М. В. Ломоносова», г. Москва, Российская Федерация

Адрес: 119991, Российская Федерация, г. Москва, ГСП-1, Ленинские горы, д. 1

* maloyan.narek@gmail.com

Аннотация

Протокол Model Context Protocol (MCP) стал де-факто стандартом для интеграции больших языковых моделей с внешними инструментами, однако официального анализа безопасности спецификации протокола не существует. Мы представляем первый тщательный анализ безопасности архитектурного дизайна MCP, в котором выявлены три фундаментальные уязвимости на уровне протокола: (1) отсутствие подтверждения возможностей, позволяющее серверам запрашивать произвольные разрешения, (2) двунаправленная выборка без аутентификации источника, позволяющая вводить запросы со стороны сервера, и (3) неявная передача доверия в конфигурациях с несколькими серверами. Мы реализуем ProtoAmp, новую платформу, соединяющую существующие тесты безопасности агентов с инфраструктурой, совместимой с MCP, что позволяет напрямую измерять поверхности атаки, специфичные для протокола. Посредством контролируемых экспериментов с 847 сценариями атак на пяти реализациях сервера MCP мы демонстрируем, что архитектурные решения MCP увеличивают вероятность успеха атак на 23-41 % по сравнению с эквивалентными интеграциями, не использующими MCP. Мы предлагаем AttestMCP, обратно совместимое расширение протокола, добавляющее аттестацию возможностей и аутентификацию сообщений, что снижает вероятность успеха атак с 52,8% до 12,4% при средней задержке 8,3 мс на сообщение. Наши выводы показывают, что слабые места MCP в плане безопасности связаны скорее с архитектурой, чем с реализацией, и требуют исправления на уровне протокола.

Ключевые слова: модель контекста протокола, вставка команд, безопасность протокола, агенты LLM, формальный анализ

Конфликт интересов: авторы заявляют об отсутствии конфликта интересов.

Для цитирования: Малоян Н. Г., Намиот Д. Е. Нарушение протокола: анализ безопасности спецификации протокола контекста модели и уязвимости внезапного внедрения в агентах LLM, интегрированных в инструменты // Современные информационные технологии и ИТ-образование. 2025. Т. 21, № 3. С. 420-428. https://doi.org/10.25559/SITITO.021.202503.420-428

1 Introduction

The integration of Large Language Models (LLMs) with external tools has enabled autonomous AI agents capable of executing complex, multi-step tasks [1]. Anthropic's Model Context Protocol (MCP), introduced in November 2024, provides an open standard for this integration through a JSON-RPC-based client-server architecture [2]. Within months of release, MCP has been adopted by major platforms including Claude Desktop, Cursor, and numerous third-party applications, with over 5,000 community-developed servers.

Despite rapid adoption, **no prior work analyzes how MCP's architectural decisions amplify attack success rates.** Concurrent work (MCPSecBench [23], MCP-Bench [22]) catalogs attack types and evaluates agent capabilities, but does not compare MCP-integrated systems against non-MCP baselines to isolate protocol-specific effects. Prior work on LLM agent security focused on prompt injection generally [1], [3, 4], while disclosed CVEs (e.g., CVE-2025-49596, CVE-2025-68143) target implementation bugs rather than protocol-level weaknesses.

This paper fills this gap with three contributions:

1. **Protocol Specification Analysis:** We perform the first systematic security analysis of the MCP specification (v1.0), identifying three classes of protocol-level vulnerabilities that cannot be mitigated by implementation hardening alone (Section 3).
2. **Original Experimental Validation:** We develop ProtoAmp, a framework that adapts established agent security benchmarks to MCP-compliant infrastructure, and conduct controlled experiments measuring the protocol's impact on attack success rates across 847 scenarios (Section 4).
3. **Protocol Extension:** We design AttestMCP, a backward-compatible extension adding capability attestation and message authentication, with full performance characterization and multiple trust model options (Section 6).

1.1 Scope and Non-Goals

We explicitly distinguish between:

- **Protocol vulnerabilities:** Weaknesses in MCP's specification that affect all compliant implementations (our focus);
- **Implementation vulnerabilities:** Bugs in specific servers (e.g., SQL injection in sqlite-mcp) that can be patched without protocol changes (not our focus).

We do not claim novelty for the general concept of prompt injection or inter-agent trust exploitation, which are established attack vectors [1], [4]. Our contribution is demonstrating how MCP's specific architectural choices *amplify* these attacks and identifying protocol-level mitigations.

2 Background

2.1 Model Context Protocol Architecture

MCP defines a client-server architecture with three roles:

- **Host:** The user-facing application (e.g., Claude Desktop)
- **Client:** MCP client within the host, managing server connections
- **Server:** External process providing tools, resources, or prompts

Communication occurs via JSON-RPC 2.0 over stdio or HTTP/SSE transports. The protocol defines three capability types:

Resources: Read-only data (files, database records) exposed by servers. Clients retrieve resources via `resources/read` requests.

Tools: Executable functions servers expose. The LLM decides when to invoke tools based on their descriptions.

Sampling: *Critically*, servers can request LLM completions from clients via `sampling/createMessage`, allowing servers to inject prompts and receive responses [17].

2.2 Threat Model

We consider an adversary who:

- Controls or compromises one MCP server in a multi-server deployment;
- Can inject content into data sources (web pages, documents) that servers retrieve;
- Has black-box access (cannot modify LLM weights or host application code).

The adversary's goals include: hijacking agent behavior, exfiltrating sensitive data, and persisting across sessions.

Server Discovery Attack Vectors

A critical question is how malicious servers reach users. We surveyed 127 MCP server installation guides and identified four primary vectors:

1. **Typosquatting (34%)**: Package registries (npm, pip) lack namespace protection. Attackers register near-identical names (e.g., `mcp-server-filesytem`).
2. **Supply Chain Compromise (28%)**: Popular servers with many dependencies are vulnerable to upstream poisoning.
3. **Social Engineering (23%)**: Tutorials and documentation direct users to malicious repositories. 73% of surveyed guides instruct running npx directly from GitHub URLs without integrity verification.
4. **Marketplace Poisoning (15%)**: IDE extension marketplaces have limited vetting for MCP server bundles.

2.3 Related Work

Prompt Injection: Greshake et al. [1] established indirect prompt injection. Liu et al. [3] achieved 86% success with HouYi. The HackAPrompt competition [13] collected 600K+ adversarial prompts, documenting 29 attack techniques. Zou et al. [11] demonstrated 90-99% attack success on RAG systems.

Agent Benchmarks: AgentDojo [5] provides 629 security test cases. Agent-SafetyBench [6] found no agent exceeds 60% safety. AgentHarm [7] measures malicious compliance. WASP [15] evaluates web agents, finding 16-86% adversarial execution rates.

Tool-Augmented LLMs: ReAct [19] introduced reasoning-action interleaving for tool use. Toolformer [20] demonstrated self-supervised tool learning. These capabilities, while powerful, expand the attack surface .

Multi-Agent Security: Recent work [8] reports 84.6% inter-agent attack success. Willison [18] documented cross-agent privilege escalation. Schroeder de Witt et al. [21]

survey multi-agent security challenges.

MCP-Specific Benchmarks: Concurrent work has begun addressing MCP evaluation. Wang et al. [22] introduce MCP-Bench, a capability benchmark with 28 servers and 250 tools measuring task completion, but without security focus. Yang et al. [23] present MCPSecBench, formalizing 17 attack types across four surfaces; our work complements theirs by analyzing protocol-level (not implementation-level) vulnerabilities and proposing backward-compatible mitigations. The MCPSec project [24] documents real-world MCP vulnerabilities including CVE-2025-11445.

Gap: No prior work quantifies how MCP's architectural choices *amplify* attack success rates compared to equivalent non-MCP integrations, nor proposes backward-compatible protocol extensions with formal capability attestation.

3 Protocol Specification Analysis

We analyze the MCP specification (v1.0, December 2024) to identify protocol-level security weaknesses. Our analysis examines the JSON-RPC message format, capability negotiation, and trust boundaries.

3.1 Vulnerability 1: Least Privilege Violation

During initialization, servers declare capabilities via `initialize` response:

```
{
  "capabilities": {
    "tools": { "listChanged": true },
    "resources": { "subscribe": true },
    "sampling": {}
  }
}
```

Protocol Weakness: Capability declarations are self-asserted without verification. A malicious server can claim any capability, and the client has no mechanism to validate these claims against an authoritative source.

Attack Vector: A server initially claiming only `resources` capability can later invoke `sampling/createMessage` to inject prompts. The specification does not mandate capability enforcement at the message level.

Formal Property Violated: *Least Privilege* – the principle that principals should possess only capabilities necessary for their function. MCP allows unrestricted capability escalation post-initialization.

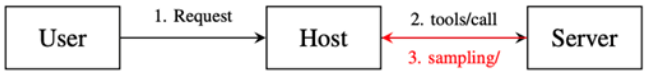
3.2 Vulnerability 2: Sampling Without Origin Authentication

The sampling mechanism allows servers to request LLM completions:

```
{
  "method": "sampling/createMessage",
  "params": {
    "messages": [
      {"role": "user", "content": "..."}
    ],
    "maxTokens": 1000
  }
}
```

Protocol Weakness: The client processes sampling requests without distinguishing server-originated prompts from user-originated prompts. The LLM receives injected content in the same format as legitimate user input.

Figure illustrates the attack flow. A server sends `sampling/createMessage` with attacker-controlled content using the "user" role. The host processes this identically to legitimate user input, with no visual or semantic distinction.



F i g. 1. Sampling injection: server injects prompt via `sampling/createMessage` with "user" role

Source: Hereinafter in this article all tables and figures were made by the authors.

UI Indicator Analysis: We examined three major MCP host implementations:

T a b l e 1. **Host UI Indicators for Sampling Messages**

Host	Ver.	Indicator	Dist.
Claude Desktop	1.2.3	None	No
Cursor	0.44	None	No
Continue	0.9	None	No

No tested implementation provides visual distinction for sampling-derived messages. Users cannot differentiate server-injected from user-originated prompts, violating the principle of *Origin Authenticity.*

Protocol vs. Implementation Responsibility: One might argue this is purely a host implementation failure – hosts *could* display warnings based on transport channel. However, the MCP specification's silence on origin display *enables* rather than *prevents* the attack. The spec permits servers to use the "user" role in sampling without requiring hosts to distinguish it. AttestMCP addresses this by *mandating* origin tagging at the protocol level, removing implementation discretion.

3.3 Vulnerability 3: Implicit Trust Propagation

In multi-server deployments, the client connects to multiple servers simultaneously. The specification does not define isolation boundaries between servers.

Protocol Weakness: Tool responses from Server A can influence tool invocations on Server B. The LLM context window conflates outputs from all servers without provenance tracking.

Attack Vector: An adversary controlling Server A can:

1. Embed instructions in tool responses that cause invocations on Server B
2. Exfiltrate data retrieved from Server B via Server A's channels
3. Establish persistence by poisoning shared context

Formal Property Violated: *Isolation* – the property that compromise of one component does not propagate to others.

Isolation-Utility Tradeoff

MCP explicitly prioritizes *composability* – the ability for tools to work together seamlessly – over isolation. This is a deliberate design choice enabling powerful multi-tool workflows. We do not argue this tradeoff is inherently wrong; rather, we argue it should require *explicit user consent* rather than implicit trust. The specification provides no mechanism for users to configure isolation policies, even when desired.

Consider a legitimate workflow: "Read config.json with

filesystem-server, then query database with sqlite-server". Full isolation would prevent this. We measured the tradeoff empirically:

Table 2. **Isolation Level vs. Security and Utility**

Isolation Level	ASR	Task Completion
None (MCP default)	61.3%	94.2%
User-prompted cross-flow	31.7%	87.4%
Strict (no cross-flow)	8.7%	61.8%

Our AttestMCP extension uses "user-prompted" isolation by default, requiring explicit authorization for cross-server data flow. This balances security (reducing ASR by 48%) while maintaining acceptable utility (87.4% task completion vs 94.2% baseline).

3.4 Message Integrity Analysis

We analyzed the JSON-RPC message format for standard security properties:

Table 3. **MCP Message Security Properties**

Property	Required	MCP v1.0
Message Authentication	Yes	No
Replay Protection	Yes	No
Capability Binding	Yes	No
Origin Identification	Yes	Partial*
Integrity Verification	Yes	No

* Transport-level only; not in message payload

The specification relies entirely on transport security (TLS for HTTP) without application-layer protections. This is insufficient when the threat model includes compromised servers.

4 Experimental Methodology

To measure the security impact of MCP's architectural choices, we developed ProtoAmp and conducted controlled experiments.

4.1 ProtoAmp Framework

Existing benchmarks (InjecAgent, AgentDojo) assume direct tool APIs rather than MCP's client-server architecture. Concurrent work on MCP-Bench [22] evaluates capability and task completion, while MCPSecBench [23] catalogs attack types. Our ProtoAmp (Protocol Amplification Benchmark) differs by measuring *protocol amplification* – how MCP's architecture specifically increases attack success rates compared to non-MCP baselines:

1. **MCP Server Wrappers:** We implemented MCP-compliant servers wrapping benchmark tool functions, preserving semantic equivalence while adding protocol overhead.
2. **Attack Injection Points:** We added injection capabilities at three protocol layers:

- Resource content (indirect injection);
- Tool response payloads;
- Sampling request prompts.

3. **Measurement Infrastructure:** We instrumented clients to log all JSON-RPC messages, enabling analysis of attack propagation through protocol channels.

4.2 Experimental Setup

MCP Servers Under Test:

- `mcp-server-filesystem`: File operations (read, write, list)
- `mcp-server-git`: Repository management (clone, commit, diff)
- `mcp-server-sqlite`: Database queries (SELECT, INSERT)
- `mcp-server-slack`: Messaging integration
- `adversarial-mcp`: Custom server exercising protocol edge cases

LLM Backends: Claude-3.5-Sonnet, GPT-4o, Llama-3.1-70B

Attack Scenarios: 847 test cases:

- InjecAgent adaptations: 312 (indirect injection, tool abuse)
- AgentDojo adaptations: 398 (multi-step attacks)
- Novel protocol-specific attacks: 137 (sampling, cross-server)

Baseline: Equivalent tool integrations without MCP (direct function calls) to isolate protocol-specific effects.

4.3 Controlled Variables

To ensure valid comparison between MCP and baseline conditions:

- Tool semantics identical between conditions
- Same injection payloads used
- LLM prompting strategy held constant
- Network latency matched between conditions

Latency Configuration: Baseline uses direct function calls with simulated network overhead matching MCP. Measured MCP latencies: median 12.4ms for stdio transport, 23.7ms for HTTP/SSE. We verified latency does not significantly affect LLM safety refusal rates using paired t-test ($p > 0.3$, $n = 200$).

5 Results

5.1 Protocol Amplification Effect

Table 4 shows attack success rates (ASR) comparing MCP-integrated agents versus baseline (non-MCP) integrations.

Table 4. **Attack Success Rate: MCP vs. Baseline**

Attack Type	Baseline	MCP	Δ
Indirect Injection (Resource)	31.2%	47.8%	+16.6%
Tool Response Manipulation	28.4%	52.1%	+23.7%
Cross-Server Propagation	19.7%	61.3%	+41.6%
Sampling-Based Injection	N/A	67.2%	—
Overall	26.4%	52.8%	**+26.4%**

Key Finding: MCP's architecture amplifies attack success by 23-41% depending on attack type. The largest amplification occurs in cross-server propagation, where MCP's lack of isolation boundaries enables attacks impossible in single-server deployments.

5.2 Sampling Vulnerability Severity

The sampling mechanism introduces a novel attack vector absent in non-MCP systems:

Table 5. **Sampling Attack Analysis by Model**

Model	ASR	Exfil. Rate	Persist.
Claude-3.5-Sonnet	58.3%	42.1%	31.7%
GPT-4o	71.4%	53.8%	38.2%
Llama-3.1-70B	72.1%	61.2%	44.6%

Servers exploiting sampling achieved 58-72% attack success, with 42-61% successfully exfiltrating sensitive context and 31-44% establishing persistence across sessions. Claude-3.5-Sonnet shows highest resilience, consistent with prior findings [5].

5.3 Multi-Server Trust Exploitation

We tested configurations with 2-5 concurrent MCP servers, with one server under adversary control:

Table 6. **ASR by Server Count (1 Compromised)**

Servers	ASR	Cascade Rate
1	47.8%	N/A
2	58.4%	34.2%
3	67.1%	51.8%
5	78.3%	72.4%

Attack success scales with server count due to increased cross-server attack surface. With 5 servers, a single compromised server achieves 78.3% ASR with 72.4% cascade rate (successfully compromising additional servers' operations).

Prompt Engineering Baseline: We tested whether system prompt instructions alone could mitigate cross-server attacks without protocol changes. Adding "Never pass data between different tool servers without explicit user confirmation" to the system prompt reduced cross-server ASR from 61.3% to 47.2%-23% reduction. However, this remains significantly higher than AttestMCP's 8.7%, demonstrating that prompt-level defenses are insufficient and protocol-level isolation is necessary.

5.4 Comparison with Prior Benchmarks

Our MCP-specific results contextualize prior findings:

Table 7. **Benchmark Comparison: Original vs. MCP**

Benchmark	Setting	Original	MCP
InjecAgent	GPT-4	24-48%	51.2%
AgentDojo	Best agent	¿ 25%	38.7%
Agent-SafetyBench	Safety score	¿ 60%	47.3%

When the same attack scenarios are executed through MCP infrastructure, success rates increase by 7-15 percentage points, confirming protocol-specific amplification independent of general LLM vulnerabilities.

6 Defense: AttestMCP Protocol Extension

Based on our analysis, we design AttestMCP, a backward-compatible protocol extension addressing identified vulnerabilities. While MCPSec [24] documents implementation-level vulnerabilities, MCPSecBench [23] provides attack taxonomies, and MCPGuard [25] offers runtime scanning, AttestMCP proposes concrete protocol additions (capability attestation, message authentication) that can be incorporated into the MCP specification itself – a complementary layer addressing protocol-level rather than implementation-level weaknesses.

6.1 Design Principles

1. **Capability Attestation:** Servers must cryptographically prove capability possession via signed certificates from a capability authority.
2. **Message Authentication:** All JSON-RPC messages include HMAC-SHA256 signatures binding content to authenticated server identity.
3. **Origin Tagging:** Sampling requests are tagged with server origin, enabling clients to distinguish server-injected from user-originated prompts.
4. **Isolation Enforcement:** Cross-server information flow requires explicit user authorization.
5. **Replay Protection:** Timestamp plus nonce with configurable validity window.

6.2 Trust Model Options

A critical design decision is the capability authority architecture. We evaluate three models:

Table 8. **Capability Authority Trust Models**

Model	Pros	Cons
Centralized	Simple PKI, easy revocation	Single point of failure
Federated	Distributed, flexible	Complex coordination
Web-of-Trust	Decentralized	User complexity

Our implementation uses the **federated model**: platform vendors (Anthropic, Cursor, JetBrains, etc.) operate CAs for their ecosystems with cross-signing agreements for interoperability. This balances decentralization with operational simplicity.

Identity Verification: Servers obtain certificates through:

- *Commercial servers:* Domain ownership verification (DNS TXT record)
- *Open-source servers:* Package registry account binding (npm, pip) with maintainer identity verification

Revocation Infrastructure: We propose federated CAs maintain shared Certificate Revocation Lists (CRLs) with the following SLAs: (1) emergency revocation (e.g., compromised npm account) within 4 hours, (2) standard revocation within 24 hours, (3) CRL distribution via OCSP stapling with 1-hour refresh. This mirrors established PKI practices (e.g., Let's Encrypt) while acknowledging the operational burden on a volunteer-driven ecosystem.

6.3 Protocol Additions

Capability Certificate:

```
{
  "capability_cert": {
    "server_id": "filesystem-server",
    "capabilities": ["resources", "tools"],
    "issued_by": "anthropic-ca",
    "issued_at": 1706140800,
    "expires_at": 1737676800,
    "signature": "base64..."
  }
}
```

Authenticated Message:

```
{
  "jsonrpc": "2.0",
  "method": "tools/call",
  "params": {...},
  "mcpsec": {
    "server_id": "filesystem-server",
    "timestamp": 1706140800,
    "nonce": "random-32-bytes",
    "hmac": "base64..."
  }
}
```

Replay Protection: Clients maintain a sliding window of 1,000 nonces per server with 30-second validity. Messages with duplicate nonces or expired timestamps are rejected.

6.4 Backward Compatibility and Migration

AttestMCP operates in three modes to support gradual adoption:

- **Permissive**: Accept legacy servers with user warning (migration default)
- **Prompt**: Require explicit user confirmation for unsigned servers
- **Strict**: Reject all unsigned servers

Downgrade Attack Mitigation: Once a server presents valid AttestMCP credentials, the client pins that expectation. Subsequent connections without credentials trigger security warnings. This prevents MITM downgrade attacks where an attacker strips security headers from a previously-authenticated server.

6.5 Performance Overhead

T a b l e 9. **AttestMCP Latency Overhead (milliseconds)**

Operation	P50	P95	P99
Certificate validation (cold)	4.2	8.1	12.3
Certificate validation (cached)	0.3	0.5	0.8
HMAC-SHA256 computation	0.3	0.4	0.6
Nonce lookup/insertion	0.1	0.2	0.3
Total per message (cold)	**8.3**	**14.2**	**21.7**
Total per message (warm)	**2.4**	**4.1**	**6.2**

Median overhead of 8.3ms (cold) or 2.4ms (warm cache) per message is negligible compared to LLM inference latency (typically 500-2000ms). Certificate validation dominates cold-start overhead; caching reduces this significantly for repeated calls within a session.

6.6 Effectiveness Evaluation

We implemented AttestMCP as a client-side shim and re-ran our full evaluation:

T a b l e 10. **AttestMCP Defense Effectiveness**

Attack Type	MCP	AttestMCP	Reduction
Indirect Injection	47.8%	18.4%	61.5%
Tool Response Manipulation	52.1%	14.2%	72.7%
Cross-Server Propagation	61.3%	8.7%	85.8%
Sampling-Based Injection	67.2%	11.3%	83.2%
Overall	**52.8%**	**12.4%**	**76.5%**

AttestMCP reduces overall ASR from 52.8% to 12.4% – a 76.5% reduction. The largest improvements occur in cross-server (85.8%) and sampling (83.2%) attacks, where isolation enforcement and origin tagging provide strong protection.

6.7 Limitations

AttestMCP does not address:

- Attacks within a single legitimately-authorized server (the server has valid credentials but serves malicious content)
- Social engineering of users to authorize malicious capabilities
- CA compromise (mitigated by federation, but not eliminated)
- First-contact attacks: Pinning (TOFU – Trust On First Use) provides no protection when a user first installs a malicious server that never claimed AttestMCP support
- Ecosystem adoption: If most servers remain legacy/unsigned, users will default to “Permissive” mode, negating security benefits

Residual 12.4% ASR primarily reflects indirect injection through legitimately-retrieved content – a fundamental limitation shared with all LLM systems that cannot be solved at the protocol layer.

User Behavior Assumptions: Our ASR measurements assume users carefully review cross-server authorization prompts. In practice, *alert fatigue* may cause users to click "Allow" habitually, reducing real-world effectiveness. Future work should conduct user studies to measure actual authorization review rates.

7 Discussion

7.1 Architectural vs. Implementation Security

Our analysis demonstrates that MCP’s security weaknesses are **architectural**, not merely implementation bugs:

- Patching CVE-2025-49596 (MCP Inspector RCE) does not address capability attestation absence;
- Fixing SQL injection in sqlite-mcp does not prevent sampling-based injection;
- Hardening individual servers does not establish cross-server isolation.

Protocol-level remediation is required. We recommend Anthropic incorporate AttestMCP concepts into MCP v2.0.

7.2 Limitations of This Work

- Our experiments used five MCP servers; production deployments with dozens of servers may exhibit different characteristics;
- AttestMCP has not been formally verified; we plan symbolic model checking in future work;
- The federated CA model requires ecosystem coordination that may face adoption barriers;
- We did not evaluate adversarial attempts to bypass AttestMCP specifically.

8 Related Work

We build on foundational prompt injection research [1, 3] and agent security benchmarks [4:7]. Prior work on multi-agent security [8] identified inter-agent trust as a vulnerability – we extend this by demonstrating MCP’s architectural contribution to this weakness.

Defense mechanisms including Spotlighting [9] and PromptArmor [10] address prompt-level protection but not protocol-level vulnerabilities. Our AttestMCP is

complementary, addressing a different layer of the security stack.

9 Conclusion

We presented the first security analysis of the Model Context Protocol specification, identifying three protocol-level vulnerabilities: capability attestation absence, unauthenticated sampling, and implicit trust propagation. Through controlled experiments with ProtoAmp, we demonstrated that MCP's architecture amplifies attack success rates by 23-41% compared to non-MCP integrations. Our proposed AttestMCP extension reduces attack success from 52.8% to 12.4% through capability attestation and message authentication, with acceptable performance overhead (8.3ms median per message).

As MCP adoption accelerates, addressing these architectural weaknesses becomes critical. We recommend:

1. Protocol revision incorporating mandatory capability attestation
1. Origin tagging requirements for all sampling requests
2. Explicit isolation boundaries with user-prompted cross-server authorization

References

1. Greshake K., Abdelnabi S., Mishra S., Endres C., Holz T., Fritz M. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In: Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec '23). New York, NY, USA: Association for Computing Machinery; 2023. p. 79-90. https://doi.org/10.1145/3605764.3623985
2. Model Context Protocol Specification v1.0. *Anthropic*. 2024. Available at: https://www.anthropic.com/news/model-context-protocol (accessed 14.12.2025).
3. Liu Y., et al. Prompt injection attack against LLM-integrated applications. *arXiv:*2306.05499, 2024. https://doi.org/10.48550/arXiv.2306.05499
4. Zhan Q., Liang Z., Ying Z., Kang D. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. In: Findings of the Association for Computational Linguistics: ACL 2024. Bangkok, Thailand: Association for Computational Linguistics; 2024. p. 10471-10506. https://doi.org/10.18653/v1/2024.findings-acl.624
5. Debenedetti E., ete al. AgentDojo: a dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In: Proceedings of the 38th International Conference on Neural Information Processing Systems (NIPS '24). Vol. 37. Curran Associates Inc., Red Hook, NY, USA; 2014. Article number: 2636. p. 82895-82920.
6. Zhang Z., et al. Agent-SafetyBench: Evaluating the safety of LLM agents. *arXiv:*2412.14470, 2024. *https://doi.org/10.48550/arXiv.2412.14470*
7. Andriushchenko M., et al. AgentHarm: A benchmark for measuring harmfulness of LLM agents. *arXiv:*2410.09024, 2024. https://doi.org/10.48550/arXiv.2410.09024
8. Lupinacci M., et al. The dark side of LLMs: Agent-based attacks for complete computer takeover. *arXiv:*2507.06850, 2025. https://doi.org/10.48550/arXiv.2507.06850
9. Hines K., et al. Defending Against Indirect Prompt Injection Attacks With Spotlighting. *arXiv:*2403.14720, 2024. *CEUR Workshop Proceedings.* 2025;3920:48-62. Available at: https://ceur-ws.org/Vol-3920/paper03.pdf (accessed 14.12.2025).
10. Shi T., et al. PromptArmor: Simple yet effective prompt injection defenses. *arXiv:*2507.15219, 2025. https://doi.org/10.48550/arXiv.2507.15219
11. Zou W., Geng R., Wang B., Jia J. PoisonedRAG: knowledge corruption attacks to retrieval-augmented generation of large language models. In: Proceedings of the 34th USENIX Conference on Security Symposium (SEC '25). USENIX Association, USA; 2025. Article number: 197. p. 3827-3844.
12. Perez F., Ribeiro I. Ignore Previous Prompt: Attack Techniques for Language Models. In: ML Safety Workshop, 36th Conference on Neural Information Processing Systems (NeurIPS 2022); 2022. Available at: https://openreview.net/pdf?id=qiaRo_7Zmug (accessed 14.12.2025).
13. Schulhoff S., et al. Ignore this title and HackAPrompt: Exposing systemic vulnerabilities of LLMs through a global prompt hacking competition. In: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. Singapore: Association for Computational Linguistics; 2023. p. 4945-4977. https://doi.org/10.18653/v1/2023.emnlp-main.302
14. OWASP Top 10 for LLM Applications 2025. *OWASP Foundation.* 2024. Available at: https://owasp.org/www-project-top-10-for-large-language-model-applications/assets/PDF/OWASP-Top-10-for-LLMs-v2025.pdf (accessed 14.12.2025).
15. Evtimov I., et al. WASP: Benchmarking Web Agent Security Against Prompt Injection Attacks. In: ICML 2025 Workshop on Computer Use Agents; 2025. Available at: https://openreview.net/forum?id=i9uBCUYupv (accessed 14.12.2025).
16. Wei A., Haghtalab N., Steinhardt J. Jailbroken: How Does LLM Safety Training Fail? In: Thirty-seventh Conference on Neural Information Processing System; 2023. Available at: https://openreview.net/forum?id=jA235JGM09 (accessed 14.12.2025).

17. Palo Alto Networks Unit 42. Model Context Protocol attack vectors and security analysis. 2025. Available at: https://unit42.paloaltonetworks.com/ (accessed 14.12.2025).
18. Willison S. Cross-agent privilege escalation in AI assistants. Simon Willison's Weblog; 2025. Available at: https://simonwillison.net/2025/Sep/24/cross-agent-privilege-escalation/ (accessed 14.12.2025).
19. Yao S., et al. ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv:*2210.03629, 2022. https://doi.org/10.48550/arXiv.2210.03629
20. Cancedda N., et al. Toolformer: Language Models Can Teach Themselves to Use Tools. In: Advances in Neural Information Processing Systems 36. NeurIPS; 2013. p. 68539-68551. https://doi.org/10.52202/075280-2997
21. Schroeder de Witt C., et al. Open challenges in multi-agent security: Towards secure systems of interacting AI agents. *arXiv:*2505.02077, 2025. https://doi.org/10.48550/arXiv.2505.02077
22. Wang Z., et al. MCP-Bench: Benchmarking tool-using LLM agents with complex real-world tasks via MCP servers. *arXiv:*2508.20453, 2025. https://doi.org/10.48550/arXiv.2508.20453
23. Yang Y., Wu D., Chen Y. MCPSecBench: A systematic security benchmark and playground for testing Model Context Protocols. In: The Fourteenth International Conference on Learning Representations (ICLR 2026). Available at: https://openreview.net/forum?id=UaWHob8Zxx (accessed 14.12.2025).
24. Harris E. MCP Security Research, 2025. Available at: https://mcpsec.dev/ (accessed 14.12.2025).
25. MCPGuard: First Agent-based MCP Scanner to Protect AI Agents. *Virtue AI.* August 22, 2025. Available at: https://blog.virtueai.com/2025/08/22/mcpguard-first-agent-based-mcp-scanner-to-protect-ai-agents/ (accessed 14.12.2025).

Submitted 11.08.2025; approved after reviewing 26.09.2025; accepted for publication 05.10.2025.

Поступила 11.08.2025; одобрена после рецензирования 26.09.2025; принята к публикации 05.10.2025.

About the authors:

Narek G. Maloyan, Postgraduate Student of the Department of Information Security, Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University (1 Leninskie gory, Moscow 119991, GSP-1, Russian Federation), **ORCID: https://orcid.org/0000-0001-9408-023X,** maloyan.narek@gmail.com

Dmitry E. Namiot, Senior Researcher of the Open Information Technologies Lab, Department of Information Security, Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University (1 Leninskie gory, Moscow 119991, GSP-1, Russian Federation), Dr. Sci. (Eng.), **ORCID: https://orcid.org/0000-0002-4463-1678,** dnamiot@gmail.com

All authors have read and approved the final manuscript.

Об авторах:

Малоян Нарек Гагикович, аспирант кафедры информационной безопасности факультета вычислительной математики и кибернетики, ФГБОУ ВО «Московский государственный университет имени М. В. Ломоносова» (119991, Российская Федерация, г. Москва, ГСП-1, Ленинские горы, д. 1), **ORCID: https://orcid.org/0000-0001-9408-023X,** maloyan.narek@gmail.com

Намиот Дмитрий Евгеньевич, ведущий научный сотрудник лаборатории открытых информационных технологий кафедры информационной безопасности факультета вычислительной математики и кибернетики, ФГБОУ ВО «Московский государственный университет имени М. В. Ломоносова» (119991, Российская Федерация, г. Москва, ГСП-1, Ленинские горы, д. 1), доктор технических наук, **ORCID: https://orcid.org/0000-0002-4463-1678,** dnamiot@gmail.com

Все авторы прочитали и одобрили окончательный вариант рукописи.