# Security Analysis of Agentic Communication Protocols: Model Context Protocol (MCP) and Agent-to-Agent (A2A)

Zhang Yiyue*, Minseok Kim*, Hyungjoon Koo**

Department of Computer Science and Engineering, Sungkyunkwan University (Graduate student*, Professor**)

**Abstract**

Recent advancements in AI systems allow autonomous agents to communicate using structured protocols. This paper analyzes two emerging communication protocols for such systems: Model Context Protocol (MCP) that offers a standardized framework for models to discover and interact with external APIs and data sources and Agent-to-Agent (A2A) that facilitates secure collaboration between multi-agents. Examining the architecture and security models of both protocols, we uncover potential attack vectors (risks) and propose mitigation strategies to support secure and interoperable designs for multi-agent AI systems.

## I. Introduction

Autonomous AI agents have evolved from simple text generators to systems capable of executing real-world tasks through external tool integration [1]. This evolution necessitates robust communication protocols and security frameworks. The Model Context Protocol (MCP) establishes a standardized mechanism for AI models to dynamically discover and orchestrate external APIs and data sources, reducing integration complexity while enhancing system adaptability [2-4]. Complementing MCP, the Agent-to-Agent Protocol (A2A) provides a JSON-RPC-based framework that facilitates negotiation, delegation, and task coordination between autonomous agents with enterprise-grade authentication [5,6].

These protocols form synergistic layers in the AI ecosystem: MCP abstracts tool invocation at the model level, while A2A governs inter-agent collaboration. As both protocols operate across distributed environments, they introduce novel attack surfaces that raise significant security implications. This paper presents a systematic analysis of the architectural design, interaction patterns, and security models of MCP and A2A, with emphasis on identifying key vulnerabilities and proposing architectural safeguards to inform future protocol standardization efforts.

## II. Agent Communication Protocols

**MCP.** The Model Context Protocol enables AI models to dynamically invoke external tools and data sources during inference. Foundational research, like the work by Hou et al. [18], has laid out the basic components, how MCP works, and some early security concerns. As illustrated in Figure 1, when a user submits a query to the MCP Host, it forwards the request to the embedded Large Language Model (LLM). The LLM evaluates whether external tool use is necessary; if not, it generates a direct response. Otherwise, it issuses a request to the MCP Server, which accesses either local files or remote APIs. Results are returned to the LLM for final

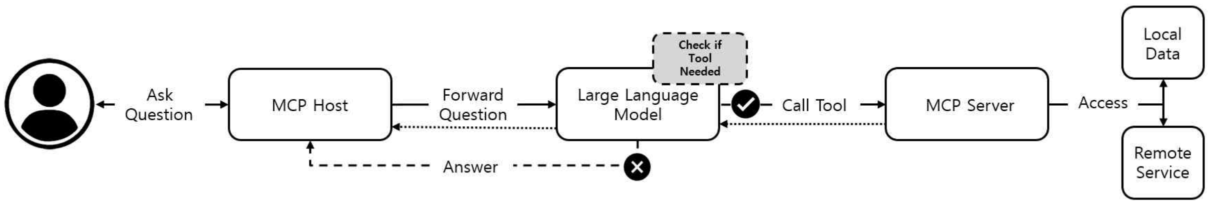

Figure 1. MCP Architecture and Request Flow. The LLM evaluates user queries to determine tool requirements. When tools are necessary, the MCP Server mediates access to local and remote resources, returning structured data for LLM integration. Dashed arrows indicate direct response paths where external tool invocation is bypassed.

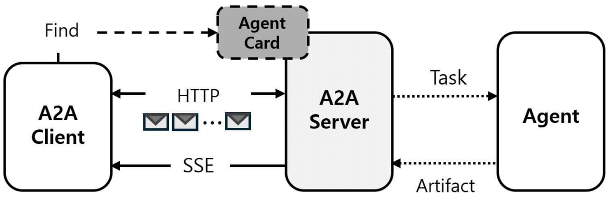

Figure 2. Overview of the A2A workflow. Clients dynamically discover agent capabilities via agent cards, initiate tasks using HTTP requests, and receive task progress or completion notifications via Server-Sent Events (SSE). Tasks are processed by backend Agents, producing Artifacts returned securely to clients through the A2A Server.

reasoning, and the answer is routed back to the user via the host.

**A2A.** The Agent-to-Agent Protocol facilitates inter-agent communication through a task-based JSON-RPC framework. Figure 2 demonstrates how agents advertise capabilities via machine-readable agent cards for dynamic service discovery. Clients initiate tasks with authenticated HTTP requests and receive updates through Server-Sent Events (SSE). Each task object contains a unique identifier, session metadata, status indicators, timestamps, optional execution logs, and resultant artifacts. A2A implements end-to-end security at the transport layer, ensuring only authorized agents can create, access, or modify task information while maintaining agent encapsulation.

**Motivating Example.** Consider an A2A agent leveraging MCP to dynamically access business intelligence APIs, process relevant data, and generate analytical reports. The completed artifact is then returned to the client via the A2A protocol. Similarly, an MCP client can function as an A2A participant, utilizing other agents as tool providers. Through compatible JSON-RPC formats, MCP and A2A unify model-tool interactions and agent-agent collaboration, enabling secure orchestration of complex autonomous workflows.

## III. Security Analysis

While MCP and A2A enhance AI system adaptability, they simultaneously introduce security vulnerabilities that necessitate comprehensive risk assessment. Table 1 summarizes key protocol-specific risks and corresponding mitigation strategies.

**Delegation Misuse.** While inapplicable to MCP, delegation misuse constitutes a fundamental concern in A2A implementations where multi-agent workflows require controlled privilege transfer. Inadequate token propagation controls between parent and child agents enable excessive privilege sharing or confused-deputy attacks. Mitigation requires implementing cryptographically bound token delegation using mechanisms like Demonstration of Proof-of-Possession (DPoP), ensuring each agent receives independently validated, scope-limited tokens with verifiable delegation lineage [9].

**Misconfigured Authentication.** MCP servers functioning as OAuth 2.0 authorization

| Attack Vector | Mitigation | |
|---|---|---|
| | MCP | A2A |
| Delegation Misuse | Not applicable | Use DPoP-bound tokens scoped per agent and task |
| Misconfigured Authentication | Delegating authentication to hardened identity providers, encrypt tokens, enforcing strict lifetimes, and adopting resource oriented authorization models | |
| Excessive Permissions | Enforce least-privilege principles through scoped permissions, attribute-based access controls (ABAC), and automated permission auditing | |
| Prompt Injection | Sanitize the prompts, deploy prompt shielding and execution environment segregation | |
| Vulnerability Propagation | Monitor CVEs, revoke access for vulnerable components, apply trust scoring and periodic reauthorization | |
| Token theft | Bind tokens (to mTLS/DPoP where applicable), rotate keys, shorten lifetimes, monitor usage | |

Table 1. Potential attack surface and mitigations

servers pose a threat misconfigured authentication flows and token theft, potentially enabling unauthorized resource access [9]. Though A2A typically delegates authentication, implementations using external credentials face similar vulnerabilities. Effective mitigation in both protocols includes delegating authentication to hardened identity providers, implementing envelope encryption for tokens, enforcing strict expiration policies, and adopting resource-oriented authorization models that limit attack surfaces [11].

**Excessive Permissions.** Granting MCP servers overly broad access scopes [13] creates significant security exposure. Compromised servers with excessive privileges enable systematic data exfiltration or unauthorized modification of critical systems. In A2A, permission mismanagement occurs when tokens lack appropriate scoping or context binding. Mitigation necessitates strict enforcement of least-privilege principles through fine-grained API permissions, attribute-based access controls, and automated permission auditing to detect and remediate privilege creep before exploitation.

**Prompt Injection.** Tool-poisoning attacks occur when malicious instructions embedded in MCP manifests trigger unintended behaviors including data exfiltration or unauthorized command execution [16]. Though A2A primarily employs structured RPCs, prompt injection vulnerabilities persist when agent metadata or inputs undergo language model processing. Protection requires schema validation with type enforcement, contextual input sanitization, AI-specific shielding mechanisms for detecting semantic injection patterns, and execution environment segregation between description processing and tool invocation.

**Vulnerability Propagation.** Both MCP and A2A risk propagating protocol-level and software supply chain vulnerabilities, especially when agents or services integrate third-party tools or unverified dependencies. In MCP, this occurs when outdated or insecure tool registries are referenced during dynamic discovery. In A2A, this risk intensifies as agents form complex, dynamic dependency chains where compromised components affect entire networks. Recommended countermeasures include integrated CVE monitoring with automated compliance verification, token revocation mechanisms for vulnerable agents [10], formal trust scoring frameworks for service registries, and lease-based authorization requiring periodic security revalidation and continuous integrity assessment mechanisms.

**Token Theft.** Both protocols face exposure risks through improper token handling, with A2A systems particularly vulnerable due to frequent token exchanges across distributed componentsn and highly interconnected agent workflows. Comprehensive protection requires robust, transport-layer security binding (mTLS/DPoP), context-specific token lifetimes, cryptographic key rotation, anomaly detection for suspicious usage patterns, and centralized token transparency logging that enables real-time, fine-grained forensic analysis while facilitating rapid and secure revocation of compromised credentials [5].

## Ⅳ. Discussion and Future Work

This study focuses on the architectural and protocol-level security of MCP and A2A. While the analysis highlights key risks and mitigation strategies, several limitations remain. The discussion is largely based on protocol specifications, public documentation, academic website and official blogs without validation through large-scale empirical deployment. Additionally, the analysis primarily considers expected agent behavior and does not cover more complex threat models, such as adversarial agents or compromised components within a network. Compatibility and trust management between MCP and A2A implementations also remain areas that require further exploration.

Future work may include formal methods to analyze protocol behavior under adversarial conditions, as well as empirical studies on how vulnerabilities may propagate in practical multi-agent deployments. Developing reference implementations and security setup could help assess real-world security performance and support consistent adoption.

## Ⅴ. Conclusion

In this paper, we identify potential attack vectors in MCP and A2A protocols and propose corresponding mitigation strategies for each protocol defenses to support secure, scalable AI agent collaboration.

## [References]

[1] Reuters, “AI agents: greater capabilities and enhanced risks,” Reuters, Apr. 22 2025.

[2] Anthropic,,“Anthropic Introduces Model Context Protocol,” Anthropic Blog, Nov. 25 2024.

[3] “Model Context Protocol (MCP),” GitHub, 2025.

[4] “Model Context Protocol (MCP) Specification,” modelcontextprotocol.io, Mar. 26 2025.

[5] Google Cloud AI, “Announcing the Agent2Agent Protocol (A2A),” Google Developers Blog, Apr. 2025.

[6] Minyang Chen, “Google’s A2A Protocol: Seamless Agent Collaboration Explained,” Medium, Apr. 2025.

[7] Anna Gutowska, “What Are AI Agents?,” IBM Think, 2025.

[8] Resonance Security, “From Tools to Agents: The Evolution and Impact of Autonomous AI Systems,” Medium, Jan. 2025.

[9] D. Fett et al., “OAuth 2.0 Demonstrating Proof-of-Possession (DPoP),” RFC 9449, IETF, Sept. 2023.

[10] NVIDIA Developer Blog, “Applying Generative AI for CVE Analysis at an Enterprise Scale,” NVIDIA, Jun. 2024.

[11] AWS Compute Blog, “Consuming private Amazon API Gateway APIs using mutual TLS,” AWS, 2023.

[12] Billgist, “How Amazon Web Services Facilitate Remote Work,” Billgist, Nov. 2024.

[13] Docker Docs, “Seccomp security profiles for Docker,” Docker, 2024.

[14] Kubernetes Docs, “Restrict a Container’s Access to Resources with AppArmor,” Kubernetes, 2024.

[15] JSON-RPC Working Group, “JSON -RPC 2.0 Specification,” jsonrpc.org, 2024.

[16] AxialCorps, “Validating JSON-RPC Messages,” AxialCorps Blog, 2014.

[17] Cloud Native Now, “SIEM for Containerized Environments,” CloudNativeNow, 2023.

[18] X. Hou, Y. Zhao, S. Wang, and H. Wang, “Model context protocol (mcp): Landscape, security threats, and future research directions,” arXiv preprint arXiv:2503.23278, 2025. [Online]. Available: https://arxiv.org/abs/2503.23278