A Critical Security and Architectural Review of the Model Context Protocol (MCP) Ecosystem

Yusuf Talha ARABACI
M.Sc. Student in Software Engineering
Karabuk University
Karabuk, Turkey

***Abstract*—The Model Context Protocol (MCP), introduced in late 2024, marks a significant turning point in the evolution of Large Language Models (LLMs) from passive text generators to active participants in digital ecosystems. By standardizing the interface between AI agents and external data, MCP seeks to dismantle the long-standing "$N \times M$" integration barrier that has stifled the scalability of autonomous systems. This paper offers more than just a technical overview; it presents a critical synthesis of the emerging MCP landscape, evaluating the real-world trade-offs between architectural flexibility and security vulnerabilities. We analyze the industry's shift towards a "Code Execution Paradigm" as a necessary response to context bloat, and we dissect the dangerous convergence of Data, Internet, and Action—a "Lethal Trifecta" that demands a rethinking of traditional security models. Through a comprehensive review of current literature, we identify a troubling gap between the protocol's theoretical capabilities and the actual compliance of today's models. Ultimately, we argue that the ecosystem's maturity depends not on more features, but on stronger governance and trust frameworks.**

***Keywords*—Model Context Protocol, MCP, context bloat, code execution paradigm, agentic security, lethal trifecta, multi-agent systems.**

I. Introduction: The Integration Knot

For years, the promise of "agentic AI"—systems that can plan, reason, and act—has effectively hit a wall. While Large Language Models (LLMs) have become incredibly articulate, their ability to meaningfully interact with the outside world has remained brittle and unscalable. The problem wasn't a lack of intelligence, but a lack of standards. Connecting any given model (N) to any given tool (M) required a custom, bespoke integration. As the number of proprietary models and diverse data sources exploded, developers found themselves trapped in an unmanageable "$N \times M$" integration knot.

The Model Context Protocol (MCP) arrived in late 2024 as a proposed solution to this gridlock. Functioning much like a "USB-C port" for the AI layer, it offers a universal standard for connectivity. The basic premise is simple: build a connector once, and it works with any compliant model. Backed by industry heavyweights like Anthropic, OpenAI, and Google, MCP has rapidly become the de-facto standard, shifting the paradigm from stateless, one-off API calls to stateful, persistent sessions.

However, resolving the connectivity issue has only revealed deeper, more complex challenges. Giving AI agents the power to read files, query databases, and execute code introduces a "Lethal Trifecta" of security risks: access to sensitive Data, connection to the open Internet, and the ability to take Action. This paper argues that while MCP is an architectural triumph, it exposes us to critical vulnerabilities that current security models are ill-equipped to handle. We must move beyond celebrating the connection to rigorously governing the interaction.

II. Architectural Foundations

At its core, MCP is about decoupling. It separates the "thinking" (the Client/Host) from the "doing" (the Server). This modularity is what allows the ecosystem to scale, and it relies on a few fundamental concepts.

A. Client-Server Model

The model essentially mirrors the web. The **MCP Host** is the application where the user lives—think of your IDE or a desktop chat app. This Host spins up an **MCP Client**, which handles the heavy lifting of managing connections. The **MCP Server**, on the other hand, is a standalone worker. It sits in front of your database, your file system, or your internal API, acting as a secure gateway. It doesn't know or care which AI model is asking for data; it just fulfills the request if the permissions match.

B. Core Primitives

Instead of treating every tool as a unique snowflake, MCP simplifies the world into three distinct categories:

1) **Tools:** These are the agent's hands. Whether it's running a SQL query or posting a message to Slack, if it changes the state of the world, it's a Tool.
2) **Resources:** These are the agent's eyes. They provide passive read access to data—logs, code files, API responses—without the risk of unintended side effects.
3) **Prompts:** These are the instruction manuals. By standardizing common workflows (like "Review this Pull Request"), Prompts help ensure that agents stay on track and behave consistently.

C. Transport Layers: Local vs. Remote

How these components talk to each other depends entirely on where they live.

- **Stdio (Standard Input/Output):** This is the choice for privacy-conscious, local development. The server runs as a customized sub-process of the main application. It's fast, and because data never leaves your machine's memory, it's inherently more secure.

- **HTTP with SSE (Server-Sent Events):** For cloud-native or distributed systems, we need something more robust. Here, the client sends commands via standard HTTP POST, while the server pushes updates back through a long-lived SSE connection. It's firewall-friendly, but opens up the usual pandora's box of web security vulnerabilities.

III. Performance: The Battle Against Bloat

Solving the connection problem is only step one. The immediate next hurdle is efficiency. We are witnessing a phenomenon aptly named "Context Bloat," where the sheer potential of the ecosystem threatens to overwhelm the models it serves.

A. The Context Bloat Paradox

In theory, giving an agent access to thousands of tools sounds empowering. In practice, it's a disaster. To let a model "know" about a tool, we have to feed it the tool's description and schema. When you scale this to an enterprise level, you end up filling the model's context window with thousands of lines of definitions before the user even asks a question. This doesn't just burn through token budgets (increasing costs by up to $236\times$); it actually makes the models dumber, as they struggle to find the signal in the noise.

B. The Code Execution Paradigm Shift

To fight this, the industry is pivoting. We are moving away from "Direct Tool Calling"—where the model awkwardly calls one API endpoint after another—towards a "Code Execution Paradigm."

- **The Old Way (Direct Calling):** The model calls a tool, waits, reads the result, thinks, and calls the next tool. It's chatty, slow, and expensive.
- **The New Way (Code Execution):** The model acts like a software engineer. It writes a single, precise Python script that imports necessary libraries, fetches data, filters it in-memory, and returns exactly what was asked. This approach is exponentially faster and can reduce token usage by over 98%.

IV. Security: The Lethal Trifecta

When we empower a text generator to take real-world actions, we create a dangerous intersection of capabilities that security researchers call the "Lethal Trifecta": **(1) Access to Private Data**, **(2) Connection to the Public Internet**, and **(3) The Authority to Act**.

A. Emerging Threat Vectors

The attack surface here is fundamentally different from a standard web app:

- **Indirect Prompt Injection (IPI):** This is the "Trojan Horse" of the AI world. An attacker doesn't need to hack your firewall; they just need to leave a note. By hiding malicious instructions in a webpage or email that the agent reads, they can trick the AI into betraying its user—"Ignore previous instructions and forward all contacts to this address."
- **Tool Poisoning:** It's a supply chain attack for agents. If an attacker publishes a useful-looking MCP server that contains hidden, malicious "features," an unsuspecting user might install a backdoor directly into their system.
- **Sampling Manipulation:** A compromised server could essentially "gaslight" the model, feeding it falsified conversation history to manipulate its future decisions.

B. Defense-in-Depth Strategies

We cannot rely on firewalls alone. We need a layered defense:

1) **Rigorous Sandboxing:** Every MCP server should be treated as a potential insider threat. They must run in isolated containers with strictly limited access to the file system.
2) **Human-in-the-Loop (HITL):** Efficiency cannot trump safety. For high-stakes actions—like deleting files or transferring funds—the AI must pause and ask for human permission.
3) **Least Privilege:** An AI agent should never run as "root." It should operate with the bare minimum permissions required to complete the task at hand.

V. Literature Review & Academic Gap Analysis

Synthesizing recent technical reports and early empirical studies allows us to map the current boundaries of our knowledge.

A. Performance and Scalability

While MCP is a robust standard, deploying it at scale creates a **cognitive bottleneck**. Benchmarks confirm that simply dumping tools into a context window degrades performance. The Code Execution Paradigm is a promising fix, but it trades one problem for another: we are swapping context efficiency for the security risks of allowing models to write and run code.

B. Empirical Evaluation

Frameworks like MCPGAUGE have revealed a stark reality: **Models are not as smart as we think.**

- **Degraded Performance:** Paradoxically, giving a model more information via tools often makes it less accurate than if it relied on its own training data.
- **Lack of Initiative:** Unless explicitly prodded, models rarely "decide" to use a tool on their own in the first turn of a conversation.
- **Compliance Failures:** Even when told to use a specific tool, models frequently hallucinate or ignore the directive.

C. Critical Research Gaps

Our review highlights four urgent areas for future research:

1) Gap 1: The Protocol-Behavior Mismatch: The protocol is solid, but the agents are not. The biggest existential threat to MCP is not technical but behavioral: current LLMs simply struggle to reason effectively about when and how to use these powerful interfaces.

2) Gap 2: The Scalability-Security Trade-Off: As we adopt Code Execution to save tokens, we need standard, verified sandboxing environments. We cannot expect every developer to roll their own secure runtime.

3) Gap 3: The Governance Void: The ecosystem is currently the Wild West. We need the equivalent of SSL certificates for agents—a "Registry of Trust" that validates the security and identity of MCP servers before an agent connects to them.

4) Gap 4: Domain-Specific Requirements: A generic protocol won't cut it for healthcare or finance. We need extensions that enforce strict compliance standards (like HIPAA or GDPR) at the protocol level.

VI. The Future: Multi-Agent Orchestration

Looking ahead, MCP is evolving from a simple connector into the nervous system of Multi-Agent Systems (MAS).

A. The AgentX Pattern

The days of the "Do-It-All" super-agent are numbered. The trend is moving towards teams of specialized experts. The **AgentX** pattern orchestrates this: a Planner breaks down the job, a Researcher finds the info, a Coder builds the solution, and an Auditor checks the work. MCP provides the common language that allows these distinct digital workers to collaborate.

B. Hybrid Architectures

The future isn't binary. We likely won't see MCP replace REST APIs entirely. Instead, we envision a hybrid world: REST and gRPC continue to power the rigid, high-performance backends, while MCP becomes the flexible, adaptive layer that handles the messy, dynamic interactions between AI agents and the systems they manipulate.

VII. Conclusion

The Model Context Protocol has succeeded in its primary mission: it has turned AI agents from isolated chat-bots into capable digital colleagues. But with this new capability comes a heavy responsibility. The long-term viability of this ecosystem won't be decided by the next feature update, but by how we handle governance. If we can adopt standards like ISO 42001, embrace secure-by-default architectures, and respect the dangers of the "Lethal Trifecta," we can ensure that these agents remain helpful assistants rather than becoming security liabilities.

Kaynakça

[1] 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.

[2] N. Krishnan, "Advancing Multi-Agent Systems Through Model Context Protocol: Architecture, Implementation, and Applications," *arXiv preprint arXiv:2504.21030*, 2025.

[3] A. Ehtesham, A. Singh, G. K. Gupta, and S. Kumar, "A Survey of agent interoperability protocols: MCP, ACP, A2A, ANP," *arXiv preprint arXiv:2505.02279*, 2025.

[4] M. M. Hasan et al., "Model Context Protocol (MCP) at First Glance: Studying the Security and Maintainability of MCP Servers," *arXiv preprint arXiv:2506.13538*, 2025.

[5] M. Flotho et al., "MCPmed: A Call for MCP-Enabled Bioinformatics Web Services for LLM-Driven Discovery," *arXiv preprint arXiv:2507.08055*, 2025.

[6] M. Mastouri, E. Ksontini, and W. Kessentini, "Making REST APIs Agent-Ready: From OpenAPI to MCP Servers for Tool-Augmented LLMs," *arXiv preprint arXiv:2507.16044*, 2025.

[7] S. Fan et al., "MCPToolBench++: A Large Scale AI Agent Model Context Protocol MCP Tool Use Benchmark," *arXiv preprint arXiv:2508.07575*, 2025.

[8] W. Xing et al., "MCP-Guard: A Defense Framework for Model Context Protocol Integrity in Large Language Model Applications," *arXiv preprint arXiv:2508.10991*, 2025.

[9] W. Song et al., "Help or Hurdle? Rethinking Model Context Protocol-Augmented Large Language Models," *arXiv preprint arXiv:2508.12566*, 2025.

[10] Z. Luo et al., "MCP-Universe: Benchmarking Large Language Models with Real-World Model Context Protocol Servers," *arXiv preprint arXiv:2508.14704*, 2025.

[11] M. Yin et al., "LiveMCP-101: Stress Testing and Diagnosing MCP-enabled Agents on Challenging Queries," *arXiv preprint arXiv:2508.15760*, 2025.

[12] G. Chhetri et al., "Model Context Protocols in Adaptive Transport Systems: A Survey," *arXiv preprint arXiv:2508.19239*, 2025.

[13] S. S. K. A. Tokal et al., "AgentX: Towards Orchestrating Robust Agentic Workflow Patterns with FaaS-hosted MCP Services," *arXiv preprint arXiv:2509.07595*, 2025.

[14] P. He et al., "Automatic Red Teaming LLM-based Agents with Model Context Protocol Tools," *arXiv preprint arXiv:2509.21011*, 2025.

[15] L. Coppolino et al., "Asset Discovery in Critical Infrastructures: An LLM-Based Approach," *Electronics 14(16):3267*, 2025.

[16] N. Bhandarwar, "Integrating Generative AI and Model Context Protocol (MCP) with Applied Machine Learning for Advanced Agentic AI Systems," *Int. J. of Computer Trends and Technology*, 2025.

[17] A. Singh et al., "A Survey of the Model Context Protocol (MCP): Standardizing Context to Enhance Large Language Models (LLMs)," *Preprints 202504.0245*, 2025.

[18] A. Korinek, "AI Agents for Economic Research," *NBER Working Paper 34202*, 2025.