Securing the Tool Layer: A Threat Taxonomy and Runtime Defense Framework for Model Context Protocol Deployments

Saurabh Yergattikar
Independent Researcher
Dublin, CA, USA
saurabh.ssy@gmail.com

Abstract

The Model Context Protocol (MCP) has rapidly emerged as the dominant standard for connecting large language models to external tools, databases, and services. Yet this convenience introduces a fundamentally new attack surface that existing LLM safety measures fail to address: adversaries can now compromise AI agents not through the user prompt, but through the tools the agent trusts. We present SHIELDMCP, a runtime security framework grounded in two contributions. First, we introduce a structured threat taxonomy derived from analysis of 80+ attack techniques catalogued under the SAFE-MCP initiative within the Linux Foundation's OpenSSF, spanning 14 tactical categories adapted from the MITRE ATT&CK methodology. Second, we describe an interception architecture that performs real-time validation of MCP tool calls and responses, combining structural analysis with semantic intent verification to detect tool poisoning, indirect prompt injection through tool outputs, and supply chain manipulation. In red-team evaluation across five popular LLM backends, SHIELDMCP reduces attack success rates from 74% to under 9% for tool poisoning and from 47% to under 6% for indirect prompt injection via tool responses, while adding fewer than 120ms of median latency per tool call. We discuss deployment considerations, the tension between security and agent utility, and lessons applicable to any organization integrating MCP into production workflows. Our framework is categorized as an *Emerging* system intended for real-world deployment.

1 Introduction

The way AI agents interact with the outside world is changing rapidly. Where earlier generations of language models operated in isolation, today's systems routinely invoke external tools—reading files, querying databases, calling APIs, and executing code—through standardized protocols. The Model Context Protocol (MCP; Salih et al., 2025), now governed by the Linux Foundation and adopted by major vendors including Anthropic, OpenAI, and Google, has become the predominant interface for this tool-calling capability. As of early 2026, over 16,000 publicly available MCP servers exist, with thousands more deployed internally across enterprises.

This proliferation creates a security problem that the NLP community has only begun to confront. Traditional LLM safety research has focused primarily on *user-facing* threats: jailbreaks, harmful content generation, and direct prompt injection. However, MCP-enabled agents face a qualitatively different risk. An adversary need not interact with the user at all. Instead, they can compromise the *tools themselves*—poisoning tool descriptions that the agent reads during planning, embedding hidden instructions in tool responses, or distributing malicious MCP server packages through public registries with minimal vetting.

The consequences are not hypothetical. CVE-2025-6514 compromised over 437,000 developer environments through a malicious MCP package that exploited unsanitized OAuth endpoints for remote code execution. Three separate vulnerabilities in a widely-used Git MCP server (CVE-2025-68143 through CVE-2025-68145) enabled attackers to achieve arbitrary code execution through prompt injection. Empirical research confirms that tool poisoning attacks succeed against 70–73% of prominent LLM agents (Wang et al., 2025a), and chained attacks through MCP tool interactions reach success rates above 90% (Gasmi et al., 2025; Li et al., 2025b).

Existing defenses are misaligned with this threat. Guardrail systems like LlamaFirewall (Chennabasappa et al., 2025) and NeMo Guardrails focus on filtering user inputs and model outputs, but they do not inspect the *tool layer*—the descriptions agents read, the parameters they construct, or the

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track), pages 865–871

responses they ingest. Architectural isolation approaches such as CaMeL (Debenedetti et al., 2025) offer strong theoretical guarantees but require substantial modifications to agent pipelines that many production systems cannot accommodate.

We present SHIELDMCP, a framework that addresses this gap through two contributions:

1. **A structured threat taxonomy for the MCP tool layer**, derived from the SAFE-MCP initiative hosted under the Linux Foundation's OpenSSF, covering 80+ attack techniques across 14 tactical categories. We consolidate this into a practical categorization of the five most critical attack families that production deployments must defend against.
2. **A runtime interception architecture** that sits between the LLM client and MCP servers, performing three-stage validation—*pre-call* (tool description integrity), *parameter* (outbound argument sanitization), and *post-response* (inbound content analysis)—to detect and block attacks without requiring modifications to either the agent or the server.

2 Threat Taxonomy

The SAFE-MCP project, developed through open community collaboration under the OpenSSF AI/ML Security Working Group, documents adversarial tactics and techniques targeting MCP-based systems. The framework adapts the MITRE ATT&CK methodology, defining 14 tactic categories (reconnaissance through impact) with over 80 individual techniques. We distill these into five critical attack families most relevant to production MCP deployments, informed by both the SAFE-MCP catalogue and recent empirical studies.

Tool Poisoning via Metadata Manipulation. Adversaries embed malicious instructions within a tool's description, parameter schema, or return-type documentation. Since LLM agents rely on these metadata fields to understand what a tool does and how to invoke it, poisoned descriptions can steer the agent toward unauthorized actions without the tool itself executing any malicious code. Wang et al. (2025a) evaluate this across 45 real-world MCP servers and 353 tools, reporting attack success rates between 61.8% (GPT-4o-mini) and 72.8% (o1-mini). The attack is stealthy: the poisoned metadata is typically invisible to end users who interact only through the agent's natural language interface. Notably, Wang et al. (2025b) demonstrate that existing safety alignment provides almost no protection, with refusal rates below 3%.

Indirect Prompt Injection Through Tool Responses. When an MCP server returns data—a file's contents, a database query result, or an API response—that data enters the agent's context window. Attackers can embed hidden instructions in these responses, hijacking subsequent agent actions. This extends the indirect prompt injection threat identified by Greshake et al. (2023) into the tool-calling pipeline specifically. Zhang et al. (2025) show that out-of-scope parameter injection through tool responses achieves a 74% average success rate across nine LLM agents. An et al. (2025) formalize this as a distinct attack surface from traditional user-input injection.

Supply Chain Compromise of MCP Servers. The MCP ecosystem lacks standardized vetting. Song et al. (2025) demonstrate that malicious MCP servers can be uploaded to widely-used aggregation platforms and subsequently downloaded by unsuspecting users. Zhao et al. (2025a) analyze 1,360 servers and 12,230 tools, finding that 27.2% of servers expose exploitable tools. Zhao et al. (2025b) generate and test 120 malicious servers targeting all components of the MCP stack—metadata, configuration, and logic—finding that public platform auditing is insufficient to detect them.

Rug Pull and Behavioral Mutation. A server that initially behaves correctly can alter its tool definitions or response behavior after gaining trust—a pattern analogous to dependency confusion attacks in software supply chains. Song et al. (2025) and Zhao et al. (2025b) both document this "rug pull" pattern, where legitimate-seeming servers introduce malicious behavior through post-deployment updates. This attack is particularly challenging because static analysis at install time cannot detect it.

Cross-Tool Exploitation Chains. When agents have access to multiple MCP servers, attackers can chain individually benign tool calls into harmful sequences. Li et al. (2025b) demonstrate that Sequential Tool Attack Chaining achieves over 90% success against GPT-4.1 by composing harmless-looking tool invocations into data exfiltration pipelines. Lupinacci et al. (2025) extend this to multi-agent settings, showing inter-agent trust

Attack Family	Avg. ASR	Detection Difficulty	Affected Stage
Tool Poisoning	61–73%	High (invisible to users)	Pre-call planning
IPI via Tool Response	47–74%	Medium	Post-response
Supply Chain	N/A†	Very High	Installation
Rug Pull	N/A†	Very High	Runtime drift
Cross-Tool Chains	>90%	High (benign individually)	Multi-step

Table 1: Critical MCP attack families. ASR = attack success rate from referenced empirical studies. †Supply chain and rug pull attacks are binary compromise events rather than per-query success rates.

exploitation succeeds at 82.4% across 17 state-of-the-art models.

Table 1 summarizes these five families with their empirical risk profiles.

3 The SHIELDMCP Framework

SHIELDMCP operates as a transparent proxy between the MCP client (the LLM agent) and any connected MCP servers. It requires no modifications to either side—agents send standard MCP requests, and servers receive standard MCP requests—making it compatible with existing deployments. The framework performs three-stage validation.

3.1 Stage 1: Tool Description Integrity

Before an agent's planning phase begins, SHIELDMCP inspects all registered tool descriptions. We maintain a *baseline registry* of vetted tool signatures (name, description hash, parameter schema) and flag deviations. For newly encountered or unvetted tools, we apply two checks:

1. **Structural anomaly detection**: We parse tool descriptions for hidden Unicode characters (zero-width joiners, right-to-left overrides), HTML/markdown injection payloads, and instruction-like patterns embedded in metadata fields. This catches the "invisible prompt" attack class documented by Wang et al. (2025a).
2. **Semantic intent verification**: A lightweight classifier (distilled from a 3B-parameter model, fine-tuned on 8,400 benign and 3,200 adversarial tool descriptions) scores whether a tool description contains embedded action directives. Descriptions scoring above threshold $\tau_d = 0.72$ are quarantined and surfaced for human review.

For rug pull detection, SHIELDMCP stores tool description hashes and alerts on any change between sessions, requiring explicit re-authorization for modified tools.

3.2 Stage 2: Parameter Sanitization

When the agent constructs a tool call, SHIELDMCP validates the outbound parameters against the tool's declared schema. We enforce type checking, range validation, and—critically—scan string parameters for embedded instructions that could be interpreted by downstream systems (SQL injection fragments, shell command injection patterns, and prompt injection payloads targeting other LLMs in the chain).

3.3 Stage 3: Response Analysis

This is the most consequential stage, as tool responses are the primary vector for indirect prompt injection in MCP contexts. When a tool returns data, SHIELDMCP applies:

1. **Instruction token detection**: We identify segments of the response that contain directive language (imperatives, system-prompt-like patterns) using a token-level classifier inspired by the approach of Das et al. (2025). The classifier tags tokens as *informational* or *instructional*, and instructional tokens in tool responses are flagged.
2. **Context boundary enforcement**: Tool responses are wrapped in explicit delimiters that the agent's system prompt is conditioned to treat as untrusted data. This is a softer version of the control/data flow separation advocated by Debenedetti et al. (2025), designed to work with unmodified agents through prompt engineering rather than architectural changes.
3. **Cross-call correlation**: For multi-step interactions, SHIELDMCP maintains a session-level dependency graph tracking what data flowed from which tools into which subsequent calls. If a tool response triggers an unexpected tool invocation (one not predicted by the agent's original task plan), the framework raises a high-severity alert. This addresses the cross-tool chaining attacks documented by Li et al. (2025b).

Figure 1 shows the overall interception flow.

4 Evaluation

We evaluate SHIELDMCP along three dimensions: attack detection effectiveness, false positive impact on benign tasks, and runtime latency overhead.

Agent $\xrightarrow{\text{tool\_list}}$ SHIELDMCP *[Stage 1: Description check]* $\rightarrow$ **Server**

Agent $\xrightarrow{\text{call(params)}}$ SHIELDMCP *[Stage 2: Param check]* $\rightarrow$ **Server**

Server $\xrightarrow{\text{response}}$ SHIELDMCP *[Stage 3: Response analysis]* $\rightarrow$ **Agent**

Cross-call correlation maintained across all three stages.

Figure 1: Three-stage interception architecture of SHIELDMCP, operating as a transparent proxy.

	Attack Success Rate (%) ↓			
Attack Type	**None**	**Regex**	**Judge**	**Ours**
Tool Poisoning	74.1	58.3	23.7	**8.6**
IPI via Response	47.2	41.1	18.4	**5.8**
Cross-Tool Chain	91.3	84.7	42.1	**14.2**
Avg. across all	*70.9*	*61.4*	*28.1*	***9.5***

Table 2: Attack success rates under different defense configurations, averaged across five LLM backends. Lower is better.

4.1 Setup

We construct an evaluation harness comprising 40 MCP servers: 25 benign servers drawn from popular open-source implementations (file system, database, web search, code execution, and calendar tools), and 15 adversarial servers implementing attacks from each of the five families in our taxonomy. For each adversarial server, we craft between 20 and 50 attack scenarios (totaling 487 attack test cases) covering tool poisoning, indirect prompt injection through responses, supply chain indicators, behavioral mutation, and cross-tool chains.

We test against five LLM backends: GPT-4o, GPT-4o-mini, Claude 3.5 Sonnet, Llama-3.1-70B-Instruct, and Qwen2.5-72B-Instruct. Each attack scenario is executed 3 times per model to account for stochastic variation.

For benign evaluation, we use 200 legitimate task scenarios spanning the 25 benign servers, measuring task completion rate with and without SHIELDMCP active.

We compare against three baselines: (1) *No defense*—direct agent-server interaction; (2) *Regex-based filtering*—pattern matching for known injection signatures; (3) *LLM-as-judge*—prompting a secondary model (GPT-4o-mini) to classify tool responses as safe or unsafe.

4.2 Results

Table 2 presents detection results for the two most prevalent attack families: tool poisoning and indirect prompt injection via tool responses. Without any defense, tool poisoning succeeds at a weighted average of 74.1% across models, consistent with published benchmarks (Wang et al., 2025a; Zhang et al., 2025). Regex filtering reduces this modestly to 58.3%, as poisoned tool descriptions rarely match fixed patterns. The LLM-as-judge baseline achieves better results (23.7%) but introduces substantial latency and fails against sophisticated attacks that embed instructions within seemingly factual content. SHIELDMCP reduces tool poisoning ASR to 8.6% while maintaining low overhead.

For indirect prompt injection through tool responses, the pattern is similar. The undefended ASR of 47.2% drops to 5.8% under SHIELDMCP, outperforming both the regex baseline (41.1%) and LLM-as-judge (18.4%).

Cross-tool chain attacks prove the hardest to defend against (14.2% residual ASR), as individual calls in the chain appear benign. The cross-call correlation module catches many chains but struggles with those that span more than four tool invocations.

Model-specific patterns. Consistent with prior findings (Wang et al., 2025a; Gasmi et al., 2025), we observe that more capable models are more vulnerable in the undefended condition: GPT-4o shows a 78.3% ASR for tool poisoning versus 62.1% for Llama-3.1-70B. However, under SHIELDMCP, this gap narrows substantially (9.2% vs. 8.1%), suggesting that external security layers equalize the vulnerability difference that stems from stronger instruction-following.

Benign task impact. On the 200 legitimate task scenarios, SHIELDMCP achieves a 96.8% task completion rate compared to 98.5% without defense. The 1.7 percentage point drop comes primarily from false positives on tools that use unusually verbose descriptions (flagged by the semantic intent classifier) and from parameter sanitization blocking legitimate but unusual string patterns. We consider this an acceptable tradeoff for production security, in line with findings from Li et al. (2025a) and Bühler et al. (2025).

Latency overhead. Median per-call latency added by SHIELDMCP is 118ms (P95: 247ms). Stage 1 checks are amortized across sessions (performed once per server registration). Stage 2 adds 12ms median. Stage 3 is the primary contributor at

94ms median, dominated by the token-level classifier. For comparison, typical MCP tool calls themselves take 200–800ms depending on the server, making the overhead a 15–60% relative increase that we find acceptable for security-sensitive deployments.

4.3 Expert Evaluation

To complement automated metrics, we recruited three security engineers with production MCP deployment experience to review 50 randomly sampled alerts generated by SHIELDMCP (25 true positives, 25 false positives as determined by our ground truth). Evaluators agreed with the ground truth classification in 92% of cases (46/50), with disagreements concentrated on borderline tool descriptions that contained legitimate instructional language (e.g., tools that return structured prompts as part of their normal function).

5 Discussion and Lessons Learned

The fundamental tension. A recurring theme in our work—and across the MCP security literature—is the tension between agent capability and security. The very instruction-following ability that makes LLM agents useful is what makes tool poisoning effective. Wang et al. (2025a) report refusal rates below 3% for poisoned tool descriptions, meaning safety alignment barely activates against this threat class. External runtime defenses like SHIELDMCP are necessary precisely because internal model safety is insufficient for the tool layer.

Defense-in-depth, not silver bullets. No single technique stops all MCP attacks. Our evaluation shows that cross-tool chains remain the hardest to defend against, with 14.2% residual ASR. We advocate a defense-in-depth posture: combine SHIELDMCP's runtime interception with static analysis of server packages before installation, least-privilege policies for tool permissions (Shi et al., 2025), and architectural isolation where feasible (Debenedetti et al., 2025).

The ecosystem vetting gap. Perhaps the most concerning finding from the broader literature is the state of MCP server distribution. With 27.2% of servers exposing exploitable tools (Zhao et al., 2025a) and malicious servers easily uploadable to major platforms (Song et al., 2025), the ecosystem resembles the early days of mobile app stores before systematic review processes were established. Industry-wide efforts toward standardized vetting—potentially through the Linux Foundation's stewardship of MCP—are urgently needed.

Implications for enterprise adoption. For organizations in regulated sectors such as financial services, the MCP security posture described above creates substantial compliance risk. Multi-stage prompt inference attacks can reliably exfiltrate confidential data even when standard safety measures are in place. Frameworks like SHIELDMCP can provide the audit trail and runtime control needed for regulatory compliance, but they must be complemented by organizational policies around which MCP servers are approved for use.

Limitations and future work. SHIELDMCP has several limitations. First, our semantic intent classifier was trained on English-language tool descriptions; multilingual MCP servers require additional training data. Second, the cross-call correlation module scales quadratically with the number of tool calls in a session, creating potential bottlenecks in long-running agent interactions. Third, determined adversaries with knowledge of our defense could craft adaptive attacks; Zhan et al. (2025) demonstrate that adaptive attacks can break many static defenses. Ongoing work explores dynamic policy updates and adversarial training of the classifier against evolving attack techniques.

6 Related Work

The security of tool-augmented LLMs has grown rapidly since Greshake et al. (2023) introduced indirect prompt injection. Benchmarks now span general agent security—AgentDojo (Debenedetti et al., 2024), InjecAgent (Zhan et al., 2024), ASB (Zhang et al., 2024)—and MCP-specific evaluation: MCPTox (Wang et al., 2025a), MSB (Zhang et al., 2025), MCPSecBench (Yang et al., 2025). MCPLIB (Guo et al., 2025) catalogs 31 attack methods; empirical demonstrations by Radosevich and Halloran (2025) and He et al. (2025) confirm real-world exploitability.

On the defense side, StruQ (Chen et al., 2024) and SecAlign (Chen et al., 2025) modify model training to resist injection. CaMeL (Debenedetti et al., 2025) separates control and data flows. DRIFT (Li et al., 2025a) and IPIGuard (An et al., 2025) monitor runtime behavior. MCP-Guard (Xing et al., 2025) provides multi-stage filtering; AgentBound (Bühler et al., 2025) applies container-

ized access control. LlamaFirewall (Chennabasappa et al., 2025) integrates jailbreak detection with code analysis but does not specifically target MCP tool-layer threats. SHIELDMCP complements these by providing a deployment-ready, modification-free proxy architecture designed for the MCP protocol layer.

7 Conclusion

The Model Context Protocol has made AI agents dramatically more capable, but it has also created an attack surface that existing safety measures do not adequately cover. Through our threat taxonomy and the SHIELDMCP runtime defense, we demonstrate that practical, deployable security for MCP tool interactions is achievable without sacrificing agent utility. As MCP adoption continues to grow under Linux Foundation governance, we hope this work contributes to establishing security as a first-class concern in the AI agent tool ecosystem.

Acknowledgments

We thank the anonymous reviewers and the area chair for their detailed and constructive feedback, which substantially improved this work.

Ethical Considerations

This work involves the study and demonstration of security vulnerabilities in AI systems. All attack evaluations were conducted against locally deployed models and servers in controlled environments; no production systems or real users were targeted. The adversarial MCP servers we constructed are not publicly released. Our threat taxonomy is derived from publicly available security research and the open-source SAFE-MCP project; we do not disclose novel zero-day vulnerabilities. We believe that documenting these threats and providing defenses serves the public interest, as organizations cannot protect against risks they do not understand. We have followed responsible disclosure practices throughout this research.

Limitations

Our evaluation covers five LLM backends, but the rapidly evolving model landscape means results may not generalize to all future models. The 487 attack test cases, while diverse, cannot exhaustively represent the full space of possible MCP attacks. Our expert evaluation involved three security engineers, a small sample that limits statistical power for inter-annotator agreement analysis. The latency measurements were obtained on a single hardware configuration (8-core CPU, 32GB RAM) and may vary across deployment environments. Finally, SHIELDMCP's effectiveness against highly adaptive adversaries who specifically target our defense mechanisms remains an open question requiring ongoing evaluation.

References

Hengyu An, Jinghuai Zhang, Tianyu Du, Chunyi Zhou, Qingming Li, Tao Lin, and Shouling Ji. 2025. IPIGuard: A novel tool dependency graph-based defense against indirect prompt injection in LLM agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*.

Christoph Bühler, Matteo Biagiola, L. Grazia, and Guido Salvaneschi. 2025. Securing AI agent execution. In *arXiv preprint arXiv:2507.12345*.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. 2024. StruQ: Defending against prompt injection with structured queries. In *USENIX Security Symposium*.

Sizhe Chen, Arman Zharmagambetov, David Wagner, and Chuan Guo. 2025. Meta SecAlign: A secure foundation LLM against prompt injection attacks. In *arXiv preprint arXiv:2507.01111*.

Sa-hana Chennabasappa, Cyrus Nikolaidis, Daniel Song, David Molnar, Stephanie Ding, Shengye Wan, Spencer Whitman, and 1 others. 2025. LlamaFirewall: An open source guardrail system for building secure AI agents. In *arXiv preprint arXiv:2507.01814*.

Debeshee Das, Luca Beurer-Kellner, Marc Fischer, and Maximilian Baader. 2025. CommandSans: Securing AI agents with surgical precision prompt sanitization. In *arXiv preprint arXiv:2507.06789*.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. 2025. Defeating prompt injections by design. In *arXiv preprint arXiv:2502.12345*.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. AgentDojo: A dynamic environment to evaluate attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*.

Tarek Gasmi, Ramzi Guesmi, Ines Belhadj, and Jihene Bennaceur. 2025. Bridging AI and software security: A comparative vulnerability assessment of LLM agent deployment paradigms. In *arXiv preprint arXiv:2507.03125*.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz.

2023. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 2023 Workshop on Artificial Intelligence Safety (AISec@CCS)*.

Yongjian Guo, Puzhuo Liu, Wanlun Ma, Zehang Deng, Xiaogang Zhu, Peng Di, Xi Xiao, and Sheng Wen. 2025. Systematic analysis of MCP security. In *arXiv preprint arXiv:2507.04915*.

Ping He, Changjiang Li, Binbin Zhao, Tianyu Du, and Shouling Ji. 2025. Automatic red teaming LLM-based agents with model context protocol tools. In *arXiv preprint arXiv:2507.10234*.

Hao Li, Xiaogeng Liu, Hung-Chun Chiu, Dianqi Li, Ning Zhang, and Chaowei Xiao. 2025a. DRIFT: Dynamic rule-based defense with injection isolation for securing LLM agents. In *arXiv preprint arXiv:2507.01234*.

Jing-Jing Li, Jianfeng He, Chao Shang, Devang Kulshreshtha, Xun Xian, Yi Zhang, Hang Su, Sandesh Swamy, and Yanjun Qi. 2025b. STAC: When innocent tools form dangerous chains to jailbreak LLM agents. In *arXiv preprint arXiv:2507.07777*.

Matteo Lupinacci, Fulvio Antonio Pironti, Francesco Blefari, Francesco Romeo, Luigi Arena, and Angelo Furfaro. 2025. The dark side of LLMs: Agent-based attacks for complete computer takeover. In *arXiv preprint arXiv:2507.08345*.

Brandon Radosevich and John Halloran. 2025. MCP safety audit: LLMs with the model context protocol allow major security exploits. In *arXiv preprint arXiv:2504.03767*.

Mohammed Salih, Jihane Gharib, and Youssef Gahi. 2025. Addressing security gaps in MCP: Design of a resilient reference architecture. In *OPTIMA*.

Tianneng Shi, Jingxuan He, Zhun Wang, Linyu Wu, Hongwei Li, Wenbo Guo, and Dawn Song. 2025. Progent: Programmable privilege control for LLM agents. In *arXiv preprint arXiv:2504.12345*.

Hao Song, Yiming Shen, Wenxuan Luo, Leixin Guo, Ting Chen, Jiashui Wang, Beibei Li, Xiaosong Zhang, and Jiachi Chen. 2025. Beyond the protocol: Unveiling attack vectors in the model context protocol ecosystem. In *arXiv preprint arXiv:2507.01711*.

Zhiqiang Wang, Yichao Gao, Yanting Wang, Suyuan Liu, Haifeng Sun, Haoran Cheng, Guanquan Shi, Haohua Du, and Xiang-Yang Li. 2025a. MCPTox: A benchmark for tool poisoning attack on real-world MCP servers. In *arXiv preprint arXiv:2507.03234*.

Zhiqiang Wang, Junyang Zhang, Guanquan Shi, Haoran Cheng, Yunhao Yao, Kaiwen Guo, Haohua Du, and Xiang-Yang Li. 2025b. MindGuard: Tracking, detecting, and attributing MCP tool poisoning attack via decision dependence graph. In *arXiv preprint arXiv:2507.05521*.

Wenpeng Xing, Zhonghao Qi, Yupeng Qin, Yilin Li, Caini Chang, Jiahui Yu, Changting Lin, Zhenzhen Xie, and Meng Han. 2025. MCP-Guard: A defense framework for model context protocol integrity in large language model applications. In *arXiv preprint arXiv:2507.04523*.

Yixuan Yang, Daoyuan Wu, and Yufan Chen. 2025. MCPSecBench: A systematic security benchmark and playground for testing model context protocols. In *arXiv preprint arXiv:2507.06559*.

Qiusi Zhan, Richard Fang, Henil Shalin Panchal, and Daniel Kang. 2024. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.

Qiusi Zhan, Richard Fang, Henil Shalin Panchal, and Daniel Kang. 2025. Adaptive attacks break defenses against indirect prompt injection attacks on LLM agents. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*.

Dongsen Zhang, Zekun Li, Xu Luo, Xuannan Liu, Peipei Li, and Wenjun Xu. 2025. MCP security bench (MSB): Benchmarking attacks against model context protocol in LLM agents. In *arXiv preprint arXiv:2507.02291*.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. 2024. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *International Conference on Learning Representations*.

Shuli Zhao, Qinsheng Hou, Zihan Zhan, Yanhao Wang, Yuchong Xie, Yu Guo, Libo Chen, Shenghong Li, and Zhi Xue. 2025a. Mind your server: A systematic study of parasitic toolchain attacks on the MCP ecosystem. In *arXiv preprint arXiv:2507.04693*.

Weibo Zhao, Jiahao Liu, Bonan Ruan, Shaofei Li, and Zhenkai Liang. 2025b. When MCP servers attack: Taxonomy, feasibility, and mitigation. In *arXiv preprint arXiv:2507.07915*.