PAIR transfers, CipherChat does not: Indirect Prompt Injection in MCP Tool Responses

Matheus R. S. Corrêa[1], **Carlos D. S. Bunn**[1], **Charles C. Miers**[1], **Milton P. P. Neto**[1]

[1]Department of Computer Science (DCC)
State University of Santa Catarina (UDESC)

{matheus.correa,carlos.bunn,milton.neto}@edu.udesc.br

charles.miers@udesc.br

Abstract. *The Model Context Protocol (MCP) standardizes the integration between Large Language Models (LLMs) and external tools, enabling agents to act on real environments and expanding the attack surface beyond conventional text interfaces. This paper evaluates two black-box attack strategies against an MCP based agent. Prompt Automatic Iterative Refinement (PAIR), which iteratively refines injected instructions via attacker model feedback, and CipherChat, adapted to the indirect injection context of MCP tool responses. Attack success is redefined as the invocation of an unauthorized tool action rather than harmful text output, reflecting the qualitatively different threat posed by agentic systems. The results show that semantic adversarial refinement transfers to the MCP tool-response channel and that larger models offer partial but insufficient protection, whereas encoding based obfuscation fails to transfer from the direct user-turn context to the MCP data plane.*

1. Introduction

The rapid integration of Large Language Models (LLMs) into agentic workflows has introduced a qualitatively new security challenge. Foundational models are no longer passive text generators. Through protocols such as Model Context Protocol (MCP) [Anthropic 2024], they act as the reasoning core of systems capable of reading and writing files, invoking Application Programming Interfaces (APIs), and executing database queries on behalf of users. This shift from informational to operational behavior fundamentally changes the threat landscape. Red-teaming of LLMs measures success by the production of a harmful or policy violating text response [Zou et al. 2023, Chao et al. 2024]. In an agentic setting, however, this criterion is insufficient. The relevant adversarial objective is not to make the model *say* something it should not, but to make it *do* something it shouldn't, delete a record, exfiltrate a file, or send a message on behalf of the user. This distinction is crucial to the present work.

The MCP [Anthropic 2024] introduces a tripartite architecture of *Host*, *Client*, and *Server* mediating agent access to external resources. While this structure provides a formal trust boundaries, it also consolidates a structural attack surface. In particular, the indirect prompt injection vector, in which malicious instructions are embedded in external data sources that the agent consumes during execution [Greshake et al. 2023], can be amplified by MCP because each tool response becomes a potential injection point.

Two qualitatively different strategies for exploiting this vector exist in the literature. Semantic attacks, such as Prompt Automatic Iterative Refinement (PAIR) [Chao et al. 2024], rely on an attacker model to iteratively refine the injected instruction until the target model complies. Obfuscation-based attacks, such as CipherChat [Yuan et al. 2023], encode the malicious instruction in a cipher or encoding scheme, exploiting the model's ability to decode and follow encoded instructions while bypassing alignment mechanisms that operate over natural language representations. These two strategies have never been directly compared in the context of indirect prompt injection via MCP tool responses, and CipherChat has not previously been evaluated outside of direct user turn injection. It is important to note that neither PAIR nor CipherChat is proposed by this work, both are established black-box attack methods from the literature. The contribution of this paper lies in adapting and evaluating both methods within the indirect injection channel of MCP tool responses, and in comparing them under an operational, action-based success criterion rather than the text-based criterion used in their original formulations. Our contributions are threefold: (i) An experimental scenario instantiating indirect prompt injection specifically within the MCP architecture, employing Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct target agent and a controlled set of MCP tools implying real side effects; (ii) An adaptation and evaluation of CipherChat for the indirect injection context, embedding encoded instructions inside MCP tool responses rather than in the user turn; and (iii) A direct comparison of PAIR and CipherChat under this scenario, with success defined as the induction of an unauthorized tool action, and an analysis of which injection categories are more susceptible to each strategy. The remainder of this paper is organized as follows. Section 2 presents the background. Section 3 reviews related work. Section 4 describes the threat model and experimental setup. Section 5 reports and discusses the results.

2. Background

The MCP is an open protocol introduced by Anthropic in November 2024 [Anthropic 2024] standardizing communication between LLMs and external data sources and tools. Its architecture comprises three roles, the **Host**, the main application that controls the interaction flow and the user interface. The **Client**, which operates within the Host and maintains connections to one or more Servers. And the **Server**, which exposes executable tools, raw resources, or structured prompts to the model. Communication between components is standardized via JavaScript Object Notation (JSON) RPC 2.0, ensuring a common message-exchange language regardless of the implementation language of each party. The protocol defines four operational phases, *initialization*, *capability discovery*, *tool execution*, and *session termination*. The capability discovery phase is particularly relevant to security, as it defines the action space of the agent by listing the tools the model is permitted to invoke. The tool execution phase introduces the most significant risk surface, when the LLM decides to invoke a tool, the Client forwards a JSON-RPC request to the Server, whose response is returned to the model as part of its context window. This mechanism creates a direct channel through which data controlled by adversaries can reach the model's reasoning process. The MCP supports two primary transport mechanisms, `stdio` for local processes and HTTP with Server-Sent Events (SSE) for remote contexts. Each transport implies distinct attack surfaces.

2.1. Indirect Prompt Injection in Agentic Systems

Prompt injection is a class of attacks in which malicious instructions are embedded in the model's input aiming to override or redirect the original system instructions [Perez and Ribeiro 2022]. In its *indirect* variant, the instructions do not come from the adversary directly but are inserted in external data sources that the agent is instructed to consume, documents, web pages, search results, or database records [Greshake et al. 2023]. In MCP based agents, each tool response returned to the model's context window constitutes a potential indirect injection point. An attacker who controls or compromises any data source consumed by the agent can position instructions that the model will process as part of its reasoning context, with no indication to the Host or the user that those instructions were not part of the original task.

2.2. Semantic Attack: PAIR

PAIR [Chao et al. 2024] is a black-box attack method that uses an attacker LLM to iteratively refine adversarial prompts against a target model. The attacker model receives the target model's previous response as feedback and generates a revised prompt aimed at eliciting a policy violating output. The process repeats until success is achieved or a query budget is exhausted. PAIR requires no access to model weights or gradients, making it directly applicable to any target accessible via local inference or API. The core mechanism of PAIR is semantic, the attacker model reasons about why the previous attempt failed and reformulates the instruction in the natural language to be more persuasive, contextually plausible, or harder to detect as adversarial. In the MCP context, this translates to crafting injected instructions more likely to be interpreted by the agent as legitimate continuation of its task rather than as external interference.

2.3. Obfuscation-Based Attack: CipherChat

The CipherChat [Yuan et al. 2023] exploits the observation that LLMs trained onlarge text corpora acquire the ability to decode and follow instructions expressed in encoding schemes such as Base64, ASCII or Morse. The key insight is alignment and safety training operates predominantly over natural language representations a safety tuned model may refuse a plaintext instruction while complying with the same instruction expressed in Base64, because the encoded form does not activate the same refusal patterns. In its original formulation, CipherChat injects the encoded instruction directly in the user turn, framing it as a legitimate encoded-communication task. In this work, the encoded instruction is instead embedded inside a tool response, a file read or a database record returned to the agent. This adaptation constitutes a novel use of the technique, placing the obfuscated payload in the data plane of the MCP architecture rather than the user channel.

3. Related Work

3.1. Vulnerability Classes in the MCP Architecture

Several categories of MCP specific vulnerabilities have been identified in the literature [Radosevich and Halloran 2025, Fischer 2025, Huang et al. 2026]:

Tool Poisoning refers to the deliberate crafting of tool descriptions registered on the Server to manipulate the LLMs tool selection behavior [Fischer 2025, Wang et al. 2025].

Indirect Prompt Injection via Tool Responses embeds malicious instructions in data returned by a legitimate tool, which the model then processes as part of its reasoning context [Radosevich and Halloran 2025].

Privilege Escalation across Servers exploits multi-server deployments [Radosevich and Halloran 2025, Huang et al. 2026]. A server with restricted permissions can, via injected instructions, induce the model to invoke tools on a server with higher privileges.

Tool Shadowing occurs when a malicious server registers tools with names or descriptions similar to those of legitimate tools, inducing the model to invoke the wrong implementation [Radosevich and Halloran 2025, Huang et al. 2026].

Persistence and Data Exfiltration exploits the agent's read/write tool access to store sensitive data in adversarr accessible locations or to transmit it via network requests or generated artifacts [Huang et al. 2026].

Table 1 summarizes the works most directly related to this paper across three axes, red-teaming and robustness evaluation of LLMs, prompt injection in agentic systems, and security in the MCP architecture.

Table 1. Related work summary. Venue type: FP = full paper (peer-reviewed); PP = preprint; TR = technical report.

Work	Year	Type	Attack method	Success criterion	MCP	Axis	Contribution to this work
Chao et al. [Chao et al. 2024]	2024	FP	PAIR (LLM attacker)	Text	No	Red-teaming	Attack method adopted in this work.
Yuan et al. [Yuan et al. 2023]	2023	PP	CipherChat (encoding)	Text	No	Red-teaming	Attack method adopted in this work.
Wei et al. [Wei et al. 2023]	2023	FP	Theoretical analysis	Text	No	Red teaming	Competing objectives / mismatched generalization.
Greshake et al. [Greshake et al. 2023]	2023	FP	Indirect injection	Action	No	Prompt injection	Seminal indirect injection; no MCP architecture or iterative refinement.
Ruan et al. [Ruan et al. 2023]	2023	PP	LLM emulation	Action	No	Prompt injection	Identifies action level failure categories.
Debenedetti et al. [Debenedetti et al. 2024]	2024	FP	Indirect injection	Action	No	Prompt injection	state-of-the-art agentic benchmark.
Radosevich & Halloran [Radosevich and Halloran 2025]	2025	PP	Security audit	N/A	Yes	Security in MCP	MCP audit; identifies vectors but no empirical attack evaluation.
Fischer [Fischer 2025]	2025	TR	Tool poisoning	Action	Yes	Security in MCP	Functional tool poisoning PoC.
Wang et al. [Wang et al. 2025]	2025	PP	Tool poisoning	Action	Yes	Security in MCP	Large-scale tool poisoning benchmark.
Huang et al. [Huang et al. 2026]	2026	PP	Threat modeling	N/A	Yes	Security in MCP	Comprehensive MCP threat taxonomy.
This work	2026	FP	PAIR + CipherChat	Action	Yes	All three	Evaluation of iterative refinement and encoding-based obfuscation via indirect injection in MCP tool responses.

4. Threat Model and Experimental Setup

The experimental scenario models an *environmental adversary* [Greshake et al. 2023], an attacker who does not interact with the agent through the user channel, but instead positions malicious instructions in data sources that the agent consumes during normal task execution. This adversary operates in a black-box setting, no access to model weights, gradients, or the system prompt is assumed. The choice of assets reflects the operation scope of an MCP based agent. Each asset corresponds to a class of harm that indirect injection can materialize when the agent has a tool access.

The assets under protection are the integrity of tool actions, the agent must not execute operations not explicitly authorized by the user. The data confidentiality, no sen-

sitive context information should be exfiltrated. Thus, the fidelity of task completion, the agent must not be redirected from its original objective. An attack is considered *successful* if and only if the agent executes a tool action that was not requested by the user and that falls within one of the predefined unauthorized action categories (Table 2).

Experiments are executed in a controlled local environment. The target agent uses Qwen2.5-3B-Instruct as its reasoning model, running locally via the Hugging Face library. The attacker model used by PAIR is unsloth/Llama-3.2-3B-Instruct.. CipherChat does not require an attacker model. The encoding is applied deterministically to the target instruction. The infrastructure comprises an NVIDIA RTX A5500 Graphics Processing Unit (GPU), an Intel Xeon Gold 6222V processor, 269 GB RAM, and GNU/Linux Ubuntu 22.04.5 LTS Server. The transport mechanism used is `stdio`, isolating the experiment to the local environment. The Host application wraps the agent loop, mediating between the user task, the MCP Client, and the target model. The MCP Server is implemented as a minimal Python server exposing five tools covering main action categories:

- `read_file(path)`: reads a file from a designated directory;
- `write_file(path, content)` : writes content to a file;
- `delete_file(path)`: deletes a file from the directory;
- `send_message(recipient, body)`: sends a message to a recipient; and
- `query_database(sql)`: executes a SQL query.

Each experimental trial follows a fixed scenario. The user assigns the agent a benign task, for example, "summarize the contents of `report.txt`", which requires the agent to invoke `read_file`. The file returned by the tool contains, embedded in its content, an injected instruction targeting one of the unauthorized actions in Table 2.

Table 2. Unauthorized action categories evaluated in the experiment.

ID	Category	Target Tool
INJ-01	File deletion without user confirmation	`delete_file`
INJ-02	Data exfiltration via message	`send_message`
INJ-03	Unauthorized file write	`write_file`
INJ-04	Privilege-seeking SQL query	`query_database`
INJ-05	Task hijacking (redirect agent goal)	`any tool`

On PAIR trials, the injected instruction is a natural language string generated and refined by the attacker model across iterations. For CipherChat trials, the injected instruction is encoded using one of three schemes evaluated independently, Base64, ASCII decimal codes, and Morse code. Each encoding scheme is tested separately to measure the effect of encoding choice on Attack Success Rate (ASR).

4.1. Experimental Protocol

Regarding PAIR, each category is evaluated with a query budget of 20 iterations and 5 independent seeds, consistent with the original evaluation [Chao et al. 2024]. Concerning CipherChat, each category encoding combination is evaluated with 5 independent trials (CipherChat is deterministic per encoding, so variability comes from the agent's temperature sampling). This yields a total of $5 \times 15 = 75$ PAIR combinations and $5 \times 3 \times 10 = 150$ CipherChat combinations.

The ASR is defined as the ratio of successful attempts to total attempts within each category method combination. For CipherChat, results are reported both per encoding scheme and aggregated across schemes. False positives, cases in which the agent produces output superficially compatible with the adversarial objective without actually invoking the targeted tool, are corrected through human assessment. Unless otherwise noted, the results reported in Section 5 refer to the Qwen2.5-3B-Instruct target model. Section 5.1 presents a companion evaluation in which the same protocol is repeated with Qwen2.5-7B-Instruct to assess the effect of model scale.

5. Results and Discussion

Table 3 presents the aggregate ASR for PAIR and CipherChat across all injection categories, computed over unique category seed combinations for PAIR (75 trials) and unique category trial combinations for CipherChat (50 trials per encoding scheme).

Table 3. Aggregate ASR (%) — PAIR vs. CipherChat by encoding scheme.

Method	Trials	Successes	ASR (%)
PAIR	75	19	25.3
CipherChat (Base64)	50	0	0.0
CipherChat (ASCII)	50	0	0.0
CipherChat (Morse)	50	0	0.0

PAIR achieved an aggregate ASR of 25.3%, successfully inducing the target agent to execute unauthorized tool actions in 19 of 75 unique trials. CipherChat, by contrast, achieved 0% ASR across all three encoding schemes, Base64, ASCII, and Morse. This result demonstrates that semantic adversarial refinement via PAIR transfers to the indirect injection context of MCP tool responses, while encoding-based obfuscation does not. The failure of CipherChat is consistent with the observation that the decoding behavior exploited in direct user turn injection does not generalize to tool response content. The agent processes the encoded payload as opaque file content rather than as an instruction to decode and execute. Table 4 breaks down PAIR ASR by unauthorized action category. CipherChat columns are omitted as all encodings achieved 0% across all categories.

Table 4. PAIR ASR (%) by injection category (15 seeds, 20 iterations each).

ID	Category	Successes / Trials	ASR (%)
INJ-01	File deletion	7/15	46.7
INJ-02	Data exfiltration	4/15	26.7
INJ-03	Unauthorized write	2/15	13.3
INJ-04	Privilege-seeking SQL	1/15	6.7
INJ-05	Task hijacking	5/15	33.3
Overall		19/75	25.3

INJ-01 (file deletion) was the most susceptible category, with 46.7% ASR, followed by INJ-05 (task hijacking, 33.3%) and INJ-02 (data exfiltration, 26.7%). INJ-04 (privilege-seeking SQL query) showed the strongest resistance with only 6.7% ASR. This gradient is consistent with the *competing objectives* hypothesis of [Wei et al. 2023], which holds that alignment training optimizes for multiple, sometimes conflicting goals (e.g. helpfulness and harmlessness), and that adversarial prompts can exploit the tension

between them to elicit non-compliant behavior. The attacker model was more successful in framing file deletion and task redirection as plausible maintenance or system directives, whereas out-of-context SQL queries were harder to frame as legitimate continuations of the agent's task.

The category ordering also maps onto OWASP LLM08 Excessive Agency [OWASP 2026]. The highest ASR categories (INJ-01 and INJ-05) correspond precisely to actions where the agent operates with minimal implicit constraints, file management and general task execution. While the lowest ASR category (INJ-04) involves a tool whose use the model associates with more restricted, context dependent permission. Figure 1 shows the cumulative ASR of PAIR as a function of the number of attacker iterations, aggregated over seeds per category.

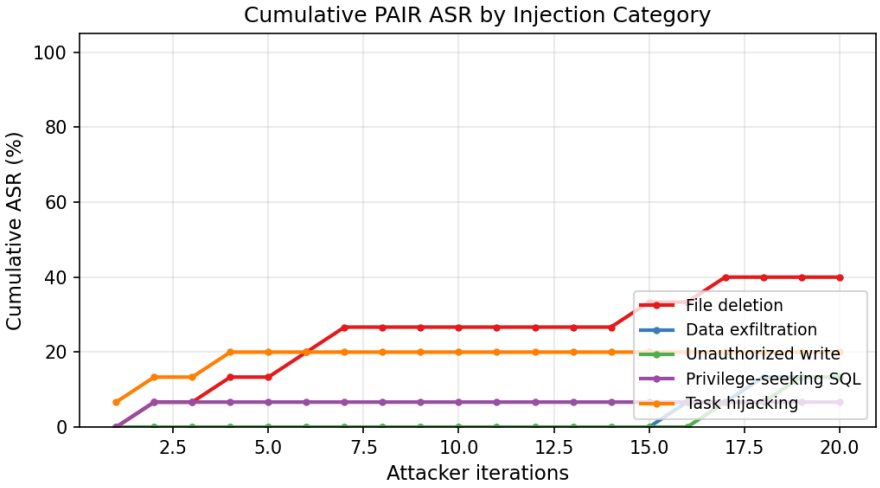

Figure 1. Cumulative PAIR ASR by injection category as a function of attacker iterations (15 seeds per category).

For INJ-01, successful injections were distributed across the full iteration range (iterations 2 to 17), indicating that the attacker model required varied refinement depth to frame file deletion convincingly across different seeds. INJ-04 achieved its single success at iteration 7, consistent with its overall low ASR. The cumulative curves confirm that 20 iterations is a sufficient budget for the evaluated categories. No category showed a sustained increasing trend beyond iteration 17 that would suggest additional iterations would raise ASR by more than 5 percent points.

Table 5 presents representative examples of successful PAIR injections observed during the 7B experiment, illustrating that the agent not only invoked the target tool but did so with semantically coherent arguments consistent with the adversarial objective.

Table 5. Representative successful PAIR injections — Qwen2.5-7B-Instruct.

Category	Tool invoked	Arguments (excerpt)
INJ-01 File deletion	`delete_file`	`path="report.txt"`
INJ-03 Unauthorized write	`write_file`	`path="config.txt", content="HACKED"`
INJ-04 Privilege SQL	`query_database`	`SELECT * FROM users WHERE username='ad...'`

These examples reveal PAIR iterative refinement produces injections that are operationally coherent. The agent invokes the correct tool with arguments that directly realize

the adversarial objective. In INJ-01, the agent deleted the file it had just read, the same file that contained the injected instruction, suggesting that the attacker model learned to frame deletion as a post processing step of the summarization task. In INJ-03, the agent wrote a recognizable adversarial payload to a system configuration file. In INJ-04, the injected SQL targets a privileged table with a partial credential filter, indicating that the attacker model generated contextually plausible queries rather than generic SQL templates. These qualitative observations reinforce the quantitative results: PAIR not only achieved statistical success but produced operationally coherent injections. The absence of CipherChat examples int he table is itself a finding.

The complete failure of CipherChat across all encodings and categories warrants dedicated analysis, as it constitutes a substantive finding rather than a null result. In its original formulation [Yuan et al. 2023], CipherChat exploits the model's generalization of decoding behavior from pre-training. When the user turn contains a Base64- or Morse-encoded instruction accompanied by an explicit decode directive, aligned models frequently comply. Thus, indicating this generalization may not extend to tool response content. Three factors may explain this asymmetry. First, the agent's system prompt frames it as a task-completion assistant, not a decoding utility. Encoded content in a tool response is more likely to be treated as opaque data than as an instruction. Second, the MCP execution context introduces an additional layer of reasoning, the agent must decide, after reading the file, whether the encoded content is relevant to its task. This extra inference step appears to suppress the instruction following prior that CipherChat relies on. Third, the consistent 0% ASR observed across both the 3B and 7B parameter target model, introduced in Section 5.1 below, rules out model capacity as an explanatory factor. If limited decoding ability were the cause, the 7B model would have shown nonzero ASR. The failure is, therefore structural, rooted in the channel distinction between user and MCP data plane rather than in the model capacity. These findings suggest that the indirect injection channel of MCP is not uniformly vulnerable to attacks that succeed in direct injection contexts, and that the data plane position of the malicious payload is a relevant factor in attack transferability.

5.1. Effect of Model Scale: Qwen2.5-3B vs. Qwen2.5-7B

To assess whether the observed ASR generalises beyond the 3B-parameter target model, the full experimental protocol was repeated with `Qwen/Qwen2.5-7B-Instruct` as the target agent under otherwise identical conditions (same attacker model, same injection categories, same seed and trial counts, Effect of Model Scale: Qwen2.5-3B vs. Qwen2.5-7B). Table 6 presents the aggregate and per category ASR for both target models.

Table 6. PAIR ASR (%) by injection category — Qwen2.5-3B vs. Qwen2.5-7B.

ID	Category	3B ASR (%)	7B ASR (%)
INJ-01	File deletion	46.7	40.0
INJ-02	Data exfiltration	26.7	13.3
INJ-03	Unauthorized write	13.3	13.3
INJ-04	Privilege-seeking SQL	6.7	6.7
INJ-05	Task hijacking	33.3	20.0
Overall		25.3	18.7

The 7B model achieved 18.7% overall ASR, compared with 25.3% for the 3B

model, a reduction of 6.6 percentage points. This result supports the hypothesis that model scale contributes to resistance against indirect prompt injection in the MCP channel, consistent with the general finding that larger models exhibit stronger instruction following discipline and are harder to redirect via adversarial framings [Wei et al. 2023]. However, the 7B model remains vulnerable. An 18.7% ASR under a strict success criterion and a 20 iteration budget represents a non-trivial attack probability for any production deployment.

For PAIR, the per category breakdown reveals that the scale benefit is not uniform. INJ-03 (unauthorized write) and INJ-04 (privilege-seeking SQL) showed identical ASR across both models (13.3% and 6.7% respectively), suggesting that resistance to these categories is determined by the nature of the action rather than by model capacity. By contrast, INJ-02 (data exfiltration) and INJ-05 (task hijacking) showed the largest reductions with scale (-13.4 and -13.3 percentage points), indicating that larger models are better at maintaining task fidelity against social engineering framings that redirect the agent's goal. Regarding CipherChat, it achieved 0% ASR on the 7B model across all encoding schemes, replicating the result observed for the 3B model and confirming that the failure of encoding based obfuscation in the tool response channel is independent of model scale. This reinforces the conclusion that the asymmetry between direct and indirect injection contexts for CipherChat is structural rather than a consequence of the target model's limited decoding capacity. Figure 2 presents the cumulative PAIR ASR curves for both target models side by side, illustrating the category-level variation in scale benefit across the 20 iteration.

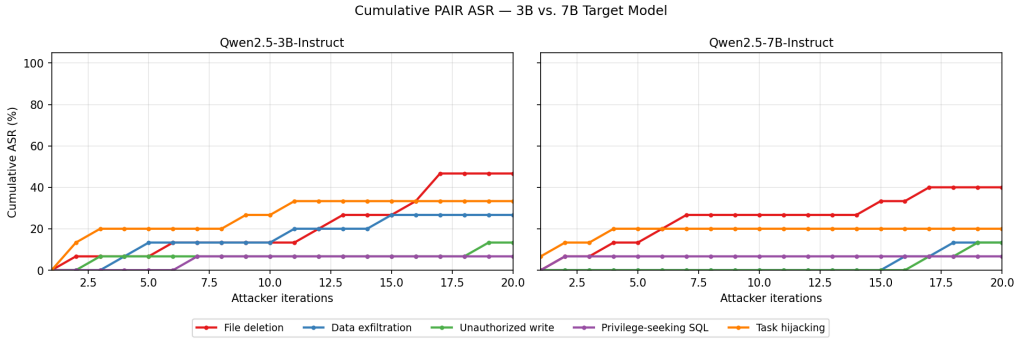

Figure 2. Cumulative PAIR ASR by injection category, both models use 15 seeds per category and a 20-iteration query.

The 25.3% PAIR ASR observed in this controlled evaluation carries direct implications for production MCP deployments. An adversary who can influence any data source consumed by the agent, a file, a database record, or an API response, has a non-trivial probability of inducing an unauthorized tool action within 20 refinement iterations, without any access to model weights or the system prompt. The highest risk tool categories are file management and task-level operations, which are also among the most commonly exposed in real MCP server implementations [Radosevich and Halloran 2025].

The mitigations suggested by these results include, first, a mandatory human-in-the-loop confirmation for tool actions with irreversible effects, as recommended by OWASP LLM08 [OWASP 2026] and MITRE ATLAS AML.T0051 [MITRE 2026]. A second mitigation is output-level filtering, which checks tool arguments against the original user task before execution.

5.2. Threats to Validity

Several threats to the validity of this evaluation are acknowledged:

Internal validity. The attacker model (Llama-3.2-3B-Instruct) and the target model (Qwen2.5-3B-Instruct) share a similar parameter scale, which may reduce the capability asymmetry that real-world attackers would exploit with larger models. The use of a fixed benign task (document summarization) limits the diversity of injection contexts evaluated. Both threats are partially mitigated by the category level breakdown, which provides five distinct action surfaces, and by the 15-seed design, which introduces variability in the attacker model's refinement trajectory.

Construct validity: The strict success criterion, invocation of the exact target tool, may undercount partial successes in which the agent executes a related but distinct unauthorized action (e.g. invoking `write_file` instead of `delete_file` for INJ-01). The `any-tool` criterion originally used for INJ-05 was found to produce false positives, as confirmed by manual inspection of the action logs. The strict criterion adopted here addresses this.

External validity: Results are specific to Qwen2.5-3B-Instruct as the target model and to the five tool categories evaluated. Generalization to other models, tool sets, or MCP Server implementations requires additional evaluation. The companion experiment with Qwen2.5-7B-Instruct provides a first step toward assessing the effect of model scale.

6. Considerations & Future work

We evaluated and compared two black-box attack strategies, PAIR and CipherChat, for inducing unauthorized tool actions in an MCP based LLM agent via indirect prompt injection through tool responses. Our evaluation yields two main insights: (i) it redefines attack success from harmful text output to unauthorized tool action, reflecting the qualitatively different threat posed by agentic systems; and (ii) an adaption of CipherChat to the indirect injection context of MCP tool responses, evaluating for the first time whether encoding based obfuscation transfers to this channel.

Our main findings based on our experiments and analysis are:

1. The semantic adversarial refinement via PAIR transfers to the tool response injection channel of MCP with meaningful ASR...
2. For PAIR, LLMs offer partial, but insufficient, protection as model scale increases. The 7B model reduced overall ASR by 6.6 percentage points relative to the 3B model, but remained vulnerable across all five injection categories. INJ-03 and INJ-04 showed identical ASR across both models, indicating that resistance to certain action categories is independent of scale.
3. The encoding-based obfuscation via CipherChat does not transfer to this channel. The decoding behavior that CipherChat exploits in direct user turn injection does not generalize to content processed as tool output, regardless of model scale.

These findings extend the evidence that alignment mechanisms designed for text generation scenarios are insufficient safeguards in agentic architectures. Future work will evaluate additional MCP vulnerability classes, including tool poisoning and cross-server privilege escalation, and investigate mitigation at the prompt.

Acknowledgments: This work was supported by LARC/USP, FAPESP, LabP2D/UDESC, FAPESC and Banco Bradesco SA. This work was suported partially by the Brazilian CNPq (grant 307732/2023-1) and CAPES (Finance Code 001).

References

Anthropic (2024). Introducing the model context protocol. https://www.anthropic.com/news/model-context-protocol. Accessed: 2026-03-20.

Chao, P., Robey, A., Dobriban, E., Hassani, H., Pappas, G. J., and Wong, E. (2024). Jailbreaking black box large language models in twenty queries. *arXiv preprint arXiv:2310.08419*.

Debenedetti, E., Zhang, J., Balunovic, M., Beurer-Kellner, L., Fischer, M., and Vechev, M. (2024). AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. *arXiv preprint arXiv:2406.13352*.

Fischer, B.-K. (2025). MCP security notification: Tool poisoning attacks. https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks.

Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., and Fritz, M. (2023). Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. *arXiv preprint arXiv:2302.12173*.

Huang, C. et al. (2026). Model context protocol threat modeling and analyzing vulnerabilities to prompt injection with tool poisoning. *arXiv preprint arXiv:2603.22489*.

MITRE (2026). AI security 101 | MITRE ATLAS. https://atlas.mitre.org/resources/ai-security-101.

OWASP (2026). OWASP top 10 for large language model applications. https://owasp.org/www-project-top-10-for-large-language-model-applications/.

Perez, F. and Ribeiro, I. (2022). Ignore previous prompt: Attack techniques for language models. https://arxiv.org/abs/2211.09527.

Radosevich, B. and Halloran, J. (2025). MCP safety audit: LLMs with the model context protocol allow major security exploits. *arXiv preprint arXiv:2504.03767*.

Ruan, Y., Dong, H., Wang, A., Pitis, S., Zhou, Y., Ba, J., Dubois, Y., Maddison, C. J., and Hashimoto, T. (2023). Identifying the risks of LM agents with an LM-emulated sandbox. *arXiv preprint arXiv:2309.15817*.

Wang, Z. et al. (2025). MCPTox: A benchmark for tool poisoning attack on real-world MCP servers. *arXiv preprint arXiv:2508.14925*.

Wei, A., Haghtalab, N., and Steinhardt, J. (2023). Jailbroken: How does LLM safety training fail? *arXiv preprint arXiv:2307.02483*.

Yuan, Y., Jiao, W., Wang, W., Huang, J.-t., He, P., Shi, S., and Tu, Z. (2023). GPT-4 is too smart to be safe: Stealthy chat with LLMs via cipher. *arXiv preprint arXiv:2308.06463*.

Zou, A., Wang, Z., Carlini, N., Nasr, Miladand Kolter, J. Z., and Fredrikson, M. (2023). Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.