CPI: CONTEXTUAL PROMPT INJECTION AGAINST MCP SERVER SCANNERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM)-based Model Context Protocol (MCP) Server scanners increasingly analyze not only executable code but also repository-level context, such as README files, tool descriptions, and code comments. Because this context can be freely manipulated by MCP maintainers, it creates an overlooked attack surface where a malicious maintainer may influence scanner judgments through security claims. Building on this insight, we introduce Contextual Prompt Injection (CPI), a black-box attack that uses a scanner's own findings to construct vulnerability-specific remediation narratives, claiming that reported issues have been fixed while leaving the relevant executable implementation unchanged. To enable realistic evaluation, we curate a dataset containing 372 real-world MCP Servers, including 45 with verified vulnerabilities. We evaluate CPI against three LLM-based MCP security scanners and existing benchmarks. Evaluation with manual verification shows that CPI suppresses at least one genuine vulnerability finding in more than half of baseline-positive repositories for each evaluated scanner. Such findings demonstrate that LLM-based scanners can prioritize attacker-controlled remediation claims over contradictory code evidence, highlighting the need to treat repository context as untrusted input and independently verify claimed fixes.

1 INTRODUCTION

Large language model (LLM) agents increasingly rely on external tools to retrieve information, manipulate files, and interact with third-party services. The Model Context Protocol (MCP) standardizes such interactions by allowing MCP Servers to expose tools, resources, and prompts through a common interface (Anthropic, 2024; Hou et al., 2025; Hasan et al., 2025). This extensibility also enlarges the attack surface: untrusted servers may expose dangerous operations, request excessive privileges, access sensitive data, or provide malicious metadata (Wang et al., 2026a; Zong et al., 2025; Zhang et al., 2025). These risks have already manifested in the wild. In September 2025, a malicious npm package named `postmark-mcp`, impersonating the Postmark email service, introduced a backdoor that silently BCC'd outgoing emails to an external server, which was estimated to have compromised hundreds of organizations and stolen 15,000 emails per day (Postmark Team, 2025). Recent systems already use LLMs for task-driven security scanning and code-level vulnerability analysis (Stafeev et al., 2025; Sun et al., 2024; Lin & Mohaisen, 2025). In the MCP setting, LLM-based MCP Server scanners analyze server implementations together with developer-controlled context, such as README files, tool descriptions, and code comments. Because the developer can manipulate both the code being assessed and the natural language context presented to the scanner, this context forms an adversarial input channel. A maintainer can state in a README, tool description, or code comment that a security issue has been fixed which conflicts with the unchanged vulnerable code. Can LLM-based MCP Server scanners' judgments be influenced by such a claim? Establishing whether such conflicts can systematically cause scanners to miss vulnerabilities that remain in the implementation is essential for assessing their reliability on untrusted MCP Servers.

However, there remains a gap in the systematic evaluation of whether MCP Server scanners are robust to adversarial repository context. Existing work leaves this question unresolved because it primarily addresses adjacent but distinct security settings. First, current MCP security datasets and benchmarks primarily study vulnerable servers, poisoned tool metadata, or attacks against agents

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

that consume MCP tools (Wang et al., 2026a;b; Zhang et al., 2025; Zong et al., 2025). In contrast, we study a different trust boundary: the security scanner that evaluates whether an MCP Server itself is safe. Such evaluation requires real MCP Server repositories that preserve both executable implementations and repository-level natural-language context for paired before-and-after scanning. For example, a dataset called MCPZoo provides broad real-world coverage (Wu et al., 2025), but its raw collection contains many MCP-related projects that are not independently runnable MCP Servers. Second, prior prompt injection research largely targets LLM applications and tool-using agents (Abdelnabi et al., 2023; Liu et al., 2024; Shi et al., 2026; Debenedetti et al., 2024; Evtimov et al., 2025). Related attacks also poison agent memory, knowledge bases, or retrieved knowledge (Chen et al., 2024; Zou et al., 2025). These attacks typically seek to manipulate an application's behavior, whereas scanner evaluation poses a different question: whether attacker-controlled security claims can override contradictory evidence in the code. To the best of our knowledge, existing work has not systematically evaluated the robustness of LLM-based MCP Server scanners against adversarial repository context, particularly when attacker-controlled security claims contradict evidence in the code being assessed.

Addressing this gap introduces two practical challenges. Firstly, obtaining a scanner-ready dataset of real-world MCP Servers is non-trivial. Current MCP Server datasets, such as MCPZoo, contain open-source GitHub projects with highly heterogeneous relationships to MCP, while our evaluation requires an independently runnable MCP Server with accessible implementation code and sufficient context for paired scanning. Determining whether a repository satisfies these requirements requires semantic inspection: keyword matching alone is unreliable, whereas exhaustive manual screening does not scale. Secondly, scanner-side adversarial evaluation must isolate contextual manipulation from genuine remediation. A meaningful evaluation should not rely only on generic instructions or modify the vulnerable program itself. Instead, the injected context must adapt to the scanner's concrete finding while leaving the finding-relevant executable behavior unchanged. Otherwise, a disappearing finding cannot be attributed to the scanner's treatment of adversarial context.

We address the first challenge with a three-stage curation pipeline combining rule-based eligibility screening, evidence-grounded semantic auditing, and fine-grained repository validating. Rule-based eligibility screening resolves clear cases from metadata and documentation, while evidence-grounded semantic auditing with LLM assistance examines ambiguous repositories to determine whether their primary artifact is an independently runnable MCP Server. Fine-grained repository validating confirms server eligibility and the availability of code and context for paired scanning. We further manually verify target findings and their affected locations against source code to establish ground truth for evaluating the suppression of genuine vulnerabilities. To address the second challenge, we introduce **Contextual Prompt Injection (CPI)**, a black-box attack that uses a scanner's own baseline report as feedback. Instead of issuing an explicit instruction to ignore a finding, CPI constructs a vulnerability-specific remediation narrative that falsely claims the issue has been fixed and describes the corresponding secure behavior. It adapts this narrative to three developer-controlled contexts, which contain the repository README, the affected MCP tool description, and source-code comments near the vulnerable implementation. Meanwhile, the finding-relevant executable implementation is preserved. The modified repository is then rescanned to determine whether the original finding is retained.

We evaluate CPI against three MCP Server scanners: Tencent A.I.G. (Tencent Zhuque Lab, 2025), Cisco MCP Scanner (Cisco AI Defense, 2025), and Snyk Agent Scan (Snyk, 2026), on the full real-world dataset, MCPTox (Wang et al., 2026a), and MCP Security Bench (Zhang et al., 2025). In more than half of baseline-positive repositories, CPI suppresses at least one manually verified vulnerability finding without repairing the underlying implementation. On the full 372-repository dataset, ASR ranges from 52.30% to 77.19%, and CPI also transfers to both existing MCP security benchmarks. Controlled analyses further show that vulnerability-specific remediation narratives outperform direct instructions and generic security claims, while coordinated multi-context injection achieves the strongest attack effectiveness. A lightweight prompt-level mitigation check further suggests that explicitly prioritizing executable evidence over developer-controlled security claims can reduce this susceptibility.

Contributions. Our main contributions are as follows.

- **Contextual Prompt Injection.** We identify developer-controlled repository context as an attack surface for LLM-based MCP Server scanners and propose CPI, a black-box, implementation-preserving attack that converts scanner findings into vulnerability-specific false remediation narratives. We additionally open-source the implementation of CPI at `https://github.com/ContextualPromptInjection/CPI` to support reproduction and extension of the proposed attack methodology.
- **A curated, scanner-ready real-world MCP Server dataset.** We curate 372 open-source GitHub repositories into a quality-controlled dataset of independently runnable MCP Servers that preserves both source code and developer-controlled context for paired scanner evaluation. 45 manually verified MCP Servers further provide verified vulnerability ground truth in source code.
- **A multi-scanner, multi-dataset evaluation.** We systematically evaluate CPI across three scanners, real-world repositories, and existing MCP security benchmarks, together with controlled analyses of payload specificity and injection surfaces. Our results show that attacker-authored remediation claims can conceal genuine vulnerabilities without repairing the underlying implementation.

2 Dataset Construction

To address the lack of a realistic benchmark for evaluating the robustness of MCP Server scanners to repository-level contextual prompt injection, we construct a curated dataset of real-world MCP Server repositories that preserves both executable implementations and developer-controlled natural language context for paired evaluation before and after CPI.

2.1 Data Source and Evaluation Requirements

We construct our real-world evaluation dataset from MCPZoo, a large-scale collection of MCP-related projects aggregated from multiple public sources (Wu et al., 2025). MCPZoo provides repository links and unified project metadata, offering substantially broader coverage than manually collecting individual MCP Servers. More importantly, its source repositories preserve both executable implementations and natural language artifacts, such as README files, configuration instructions, tool descriptions, and code comments. These artifacts are required for evaluating attacks against scanners that jointly reason over code and natural language context.

However, directly applying the raw MCPZoo collection is not suitable for CPI evaluation. Many entries are only MCP-related resources rather than actual MCP Servers, including tutorials or resource lists. Some do not even contain an actual and runnable MCP Server. Including such entries would confound the evaluation because their scanner outputs do not correspond to meaningful MCP Server security analysis.

Therefore, we retain repositories whose primary artifact is an independently runnable MCP Server, whose implementation code is accessible, and that contain sufficient context for paired scanner evaluation. Applying these criteria to the first 1,000 MCPZoo entries yields 372 repositories. To further distinguish suppression of scanner outputs from concealment of genuine vulnerabilities, we manually verify vulnerability findings against source code, yielding a 45-repository subset described in Section 2.3.

2.2 Multi-stage Purification Pipeline

We purify the raw collection using a pipeline containing three stages: rule-based eligibility screening, evidence-grounded semantic auditing, and fine-grained repository validating, illustrated in Figure 1.

Rule-based Eligibility Screening. The first stage performs high-precision screening using repository metadata and documentation. Positive evidence includes an explicit description of the project indicating that the repository is an MCP Server. Conversely, repository names and descriptions associated with tutorials or other general applications provide negative evidence. These rules resolve clear cases efficiently, while repositories with mixed or insufficient evidence are forwarded to the

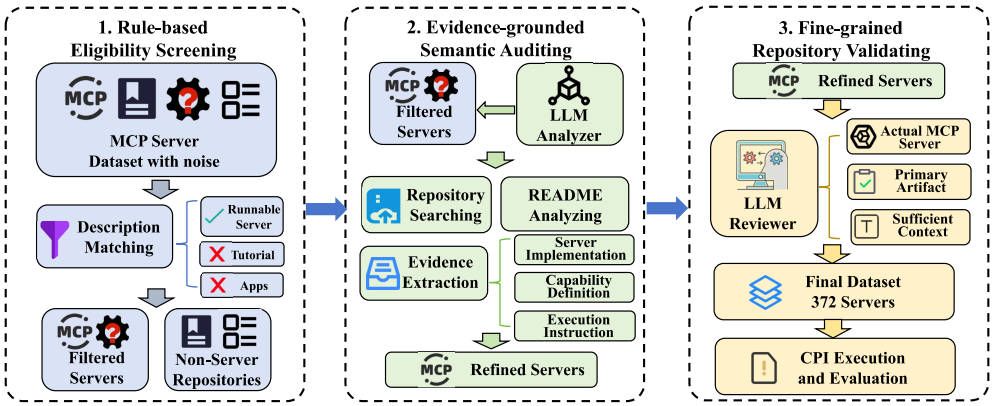

Figure 1: Overview of the repository purification pipeline. Rule-based eligibility screening resolves clear cases, evidence-grounded semantic auditing analyzes semantically ambiguous repositories, and fine-grained repository validating provides final quality control.

next stage. Since many non-server projects use the same MCP terminology, keyword presence alone is not treated as enough proof of dataset membership.

Evidence-grounded Semantic Auditing. The second stage analyzes repositories that cannot be resolved by fixed rules. Given the available repository metadata and documentation, the LLM is asked to determine whether the project's primary deliverable is an independently runnable MCP Server. Its decision is grounded in concrete repository evidence, including server implementations, MCP capability definitions, and execution instructions. To ensure the accuracy of dataset membership, the audit follows a conservative policy: repositories with incomplete evidence remain uncertain. This stage captures semantic distinctions that cannot be expressed reliably through a fixed keyword list.

Fine-grained Repository Validating. The final stage performs a comprehensive assessment of candidate repositories against the eligibility criteria. An LLM reviewer verifies whether each repository implements an independently runnable MCP Server as its primary artifact and provides sufficient implementation code and natural-language context for evaluation. Only MCP-related projects satisfying these criteria are retained. Finally, we manually verify that these servers are genuine.

2.3 Manually Verified Vulnerability Subset

Because scanner outputs are not themselves vulnerability ground truth, suppressing an unverified report does not necessarily demonstrate that CPI can conceal a genuine vulnerability. We therefore construct a high-confidence subset by manually examining each candidate finding against the affected implementation and its supporting repository context. We retain only findings for which both the reported vulnerability and the corresponding code location can be verified from the repository.

The resulting subset contains 45 MCP Servers, which enables a controlled evaluation of whether CPI suppresses verified vulnerability findings while leaving the underlying implementation unchanged. We use the remaining dataset-level evaluation to determine whether the same suppression behavior generalizes across a broader collection of real-world MCP Servers.

3 Contextual Prompt Injection

3.1 Threat Model and Attack Objective

Adversary and access. We consider a malicious MCP Server developer who controls the repository submitted for security assessment and seeks to obtain a more favorable scanner report without removing the repository behavior that triggered the original finding. The adversary has black-box access to the target scanner: it can submit a repository and observe the resulting security report, but

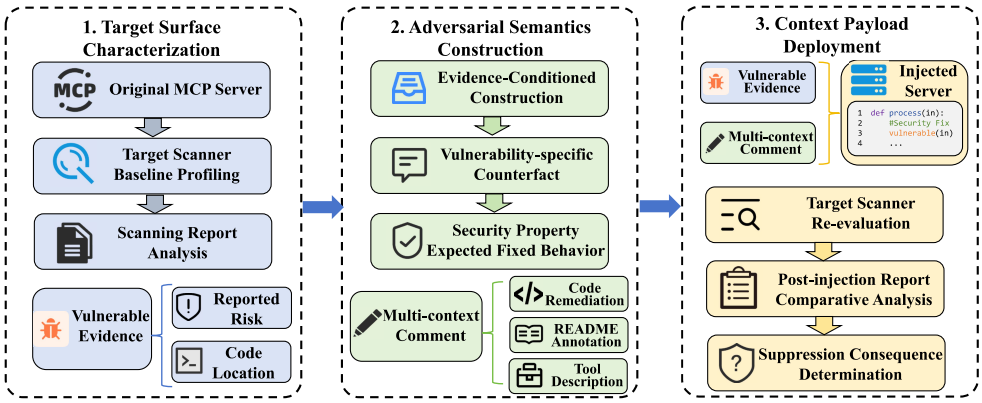

Figure 2: Overview of Contextual Prompt Injection (CPI). CPI uses baseline scanner findings to generate vulnerability-specific remediation claims that are injected into the README, MCP tool descriptions, and source-code comments. The modified repository is then rescanned to evaluate whether the original vulnerability findings are suppressed.

has no access to the scanner's internal prompts, model parameters, agent workflow, retrieval process, or decision rules. CPI assumes only access to the repository and the scanner's baseline report, and does not rely on knowledge of the scanner's internal architecture.

Attack scope and implementation invariant. Although the adversary controls the complete repository, CPI deliberately restricts its modifications to developer-controlled natural language context consumed by the scanner. Specifically, CPI may insert a security note into the README, modify the description field of an affected MCP tool, and add non-executable comments near the code. CPI does not modify executable operations, control flow, input handling, configurations, permissions, or other security-relevant behavior implicated by the finding. Motivated by prior work, which shows that code comments can become inconsistent with the implementation they describe (Rong et al., 2025), we treat comments as a distinct natural language context instead of executable evidence. By leaving the underlying implementation unchanged, we isolate changes in the scanner's output caused by attacker-controlled contextual information from those caused by genuine code-level remediation.

Attack objective and success criteria. Let $\mathcal{S}(R)$ denote the set of findings produced by scanner $\mathcal{S}$ for an original repository R, and let R' denote the repository after CPI injection. Both repositories are evaluated using the same scanner and configuration. For a baseline finding $f \in \mathcal{S}(R)$, CPI aims to prevent the scanner from reporting a corresponding finding in $\mathcal{S}(R')$ while preserving the implementation behavior associated with f.

3.2 CPI Pipeline

Having defined the attack objective, we now describe how CPI realizes it in practice. As illustrated in Figure 2, CPI consists of three stages: target surface characterization, which extracts and normalizes baseline scanner findings; adversarial semantics construction, which derives vulnerability-specific remediation claims and adapts them to multiple repository contexts; and context payload deployment, which injects these claims and rescans the repository while preserving the finding-relevant implementation. We detail each stage below.

Target Surface Characterization. Given an MCP Server repository R, CPI first scans the unmodified repository to obtain the scanner's baseline findings. Then, CPI uses an LLM to normalize the scanning report into target findings containing the vulnerability type, risk description, affected source region, and associated MCP tool. When the scanner does not explicitly identify the tool or precise code location, CPI resolves them from the reported file or function. Multiple findings are processed independently so that subsequent payloads remain tied to the scanner's original evidence.

Adversarial Semantics Construction. For each target finding, CPI derives a counterfactual remediation narrative describing the security mechanism and expected behavior that would hold if the vulnerability had actually been fixed. This narrative is adapted into three coordinated payloads: a repository-level README note, the affected MCP tool's description, and a code comment near the vulnerable implementation. Specifically, the payloads appear as ordinary natural language context, testing whether the scanner distinguishes claimed remediation from executable evidence.

Context Payload Deployment. CPI injects the three payloads into their corresponding repository contexts without modifying the vulnerable operation or other executable behavior. The modified repository R' is then rescanned under the same configuration as R. CPI matches baseline and post-injection findings by vulnerability semantics and affected tool or code region, allowing for wording and line-number shifts introduced by the injected context. A target finding is considered suppressed when no corresponding finding appears after injection.

4 EVALUATION

Our evaluation addresses the following four research questions.

RQ1: How effectively does CPI suppress findings reported by MCP Server scanners?

RQ2: Does CPI generalize to existing MCP security benchmarks?

RQ3: How does payload specificity affect CPI's attack effectiveness?

RQ4: How do different repository-level injection surfaces affect CPI's attack effectiveness?

We first describe the experimental setup shared across all research questions, followed by the overall evaluation and two controlled analyses of CPI's design.

4.1 EXPERIMENTAL SETUP

For all LLM calls under our control, including LLM-assisted repository auditing and validating, baseline finding normalization, and contextual payload generation, we use DeepSeek-V4-Flash in thinking mode with a fixed configuration across experiments. The internal LLM configurations of the evaluated scanners are left unchanged and treated as part of the black-box systems.

Datasets. We evaluate CPI on our purified MCP Server dataset (Section 2), including all 372 repositories and the manually verified 45 vulnerable repositories. We further use MCPTox (Wang et al., 2026a) and MCP Security Bench (MSB) (Zhang et al., 2025) to evaluate generalization to existing MCP security benchmarks.

Scanners. We evaluate three open-source MCP security scanners: Tencent A.I.G. (Tencent Zhuque Lab, 2025), Cisco MCP Scanner (Cisco AI Defense, 2025), and Snyk Agent Scan (Snyk, 2026), as black-box systems. CPI observes only their reports and does not access their internal prompts or analysis workflows. We use the same scanner configuration for the original and CPI-injected samples. Since Snyk Agent Scan operates on a running MCP endpoint, we expose each server's MCP-visible metadata through a lightweight replay server that returns the corresponding metadata verbatim under each experimental condition.

Metrics. We report baseline detection rate (BDR), post-injection detection rate (PDR), and attack success rate (ASR). BDR is the fraction of repositories with at least one target finding in the baseline scan. PDR is the fraction of repositories for which all baseline target findings are retained after CPI injection. ASR is computed over baseline-positive repositories, with an attack considered successful when at least one baseline finding is no longer retained after injection. Findings are matched by vulnerability semantics and reported code locations, allowing for wording differences and line-number shifts introduced by injected comments.

4.2 OVERALL EFFECTIVENESS OF CPI

To answer RQ1, we evaluate CPI at two distinct levels.

Table 1: CPI effectiveness on the manually verified 45 MCP Server repositories.

Scanner	BDR	PDR	ASR
A.I.G.	77.03%	32.59%	57.36%
Cisco MCP Scanner	50.37%	20.00%	60.39%
Snyk Agent Scan	85.93%	40.74%	52.54%

Table 2: CPI effectiveness on the full dataset of 372 real-world MCP Server repositories.

Scanner	BDR	PDR	ASR
A.I.G.	44.62%	18.55%	58.43%
Cisco MCP Scanner	46.77%	22.31%	52.30%
Snyk Agent Scan	70.70%	16.13%	77.19%

CPI suppresses detections of confirmed vulnerabilities. We first establish its effectiveness on the manually verified 45 MCP Servers, where the presence and location of the target vulnerabilities are confirmed from the source code. To mitigate the effect of LLM stochasticity, we repeat each experiment three times and report the average results across the three independent runs. Throughout the evaluation, CPI causes more than half of baseline-positive repositories for every scanner to lose at least one manually verified vulnerability finding, as shown in Table 1. These results suggest that attacker-controlled contextual claims can interfere with how scanners reason over repository evidence, causing them to miss vulnerabilities they previously detected. Appendix A illustrates this failure mode in A.I.G. Having established CPI's effect against manually confirmed vulnerabilities, we next evaluate whether the same susceptibility persists at real-world scale.

CPI remains effective at real-world scale. We scale the evaluation to all 372 real-world repositories to assess whether the observed susceptibility persists in a broader deployment setting. As shown in Table 2, across all 372 repositories, CPI achieves repository-level ASRs of 58.43%, 52.30%, and 77.19% against A.I.G., Cisco MCP Scanner, and Snyk Agent Scan, respectively. In each case, CPI substantially reduces the fraction of repositories flagged after injection, demonstrating that the susceptibility observed on the manually verified subset extends to a much broader collection of real-world MCP Servers. Together, the two evaluations show that CPI can suppress detections of confirmed vulnerabilities and that such manipulation occurs broadly across real-world repositories.

4.3 Generalization to Existing MCP Benchmarks

For RQ2, we evaluate whether CPI transfers beyond our curated real-world dataset to existing MCP security benchmarks. MCPTox focuses on tool-poisoning attacks, whereas MCP Security Bench (MSB) covers attacks across task planning, tool invocation, and response handling. Although both benchmarks were originally designed to evaluate MCP agents, we use their samples to test whether scanners remain robust after CPI injection. Table 3 reports BDR, PDR, and ASR for each scanner.

CPI transfers across existing MCP benchmarks. As shown in Table 3, CPI achieves ASRs of 73.67–87.34% on MCPTox and 50.00–68.00% on MSB for scanner–benchmark pairs. Cisco is unable to detect the risks in MSB samples. This experiment shows that CPI consistently suppresses the scanners' own baseline findings while preserving the vulnerable implementation across datasets constructed under different threat settings. Thus, the benchmark results establish cross-dataset susceptibility to CPI.

4.4 Effect of Payload Specificity

For RQ3, we randomly sample 100 repositories that are scanned as vulnerable from the purified MCP Server dataset and evaluate three payload variants against A.I.G. A **direct instruction** explicitly asks the scanner to ignore the reported risk. **Generic remediation** states that security issues have been addressed, but does not refer to the specific baseline finding. **Vulnerability-specific CPI** con-

Table 3: Evaluation of CPI on MCPTox and MCP Security Bench.

Dataset	Scanner	BDR	PDR	ASR
MCPTox	A.I.G.	70.50%	15.40%	78.15%
	Cisco MCP Scanner	61.88%	7.83%	87.34%
	Snyk Agent Scan	93.21%	24.54%	73.67%
MSB	A.I.G.	80.00%	40.00%	50.00%
	Cisco MCP Scanner	0.00%	N/A	N/A
	Snyk Agent Scan	100.00%	32.00%	68.00%

Table 4: Effect of payload specificity on A.I.G. using 100 randomly sampled vulnerable repositories from the purified MCP Server dataset.

Payload design	BDR	PDR	ASR
Direct instruction		57.00%	43.00%
Generic remediation	100.00%	58.00%	42.00%
Vulnerability-specific CPI		44.00%	56.00%
CPI + defensive instruction	92.00%	46.00%	50.00%

ditions the remediation narrative on the reported vulnerability, including its description and expected secure behavior. This experiment serves as a design to show the effect of the injection related to context. Meanwhile, we augment A.I.G.'s trusted scanner instruction with an evidence-prioritization rule that treats developer-controlled natural language as untrusted and requires remediation claims to be verified against executable code. We evaluate this defense on the same 100 repositories used above.

Table 4 shows that vulnerability-specific conditioning provides the largest improvement in attack effectiveness. Direct instruction achieves an ASR of 43.00%, while generic remediation performs slightly worse. Therefore, replacing a direct instruction with a generic security claim alone does not improve the attack. In contrast, vulnerability-specific CPI achieves an ASR of 56.00% and reduces the PDR to 44.00%. This result supports the central design of CPI: the attack becomes substantially more effective when the remediation narrative is grounded in the scanner's own finding and describes the secure behavior expected for that specific vulnerability.

Evidence prioritization supports the proposed failure mechanism. We augment A.I.G.'s trusted instruction to treat developer-controlled natural language as untrusted and require claimed remediations to be verified against executable code. On the same 100 repositories, this reduces CPI's ASR from 56.00% to 50.00%, while retaining a baseline detection rate of 92.00%. This controlled intervention supports the view that CPI exploits the scanner's weighting of contextual claims relative to executable evidence, while suggesting evidence prioritization as a mitigation direction.

4.5 Effect of Injection Surface

To isolate the effect of injection surface, we hold the vulnerability-specific narrative fixed and vary only where it is placed. As shown in Table 5, tool descriptions emerge as the strongest single surface, while full CPI achieves the highest ASR of 58.43% and the lowest PDR of 18.55%, outperforming the strongest single-surface variant. This result confirms that CPI's three injection surfaces are complementary. Repository-level documentation, tool-level semantics, and local code context provide distinct opportunities for the false remediation narrative to affect the scanner's judgments.

5 Related Work

Model Context Protocol. Prior work has studied MCP architecture, security threats, and the security of open-source MCP Servers (Hou et al., 2025; Hasan et al., 2025; Radosevich & Halloran,

Table 5: Ablation of CPI's multi-context injection on A.I.G. Full CPI jointly uses README, tool-description, and code-comment payloads.

Injection configuration	BDR	PDR	ASR
README only	44.62%	24.73%	44.58%
Tool description only		20.43%	54.22%
Code comment only		23.39%	47.59%
Full CPI (all three)		18.55%	58.43%

2025). Recent studies further investigate attacks against MCP-enabled agents, including tool poisoning, manipulation of tool preferences, and attacks across task planning, tool invocation, and response handling (Wang et al., 2026a;b; Zhang et al., 2025). These works primarily study MCP Servers or tool-using agents, whereas we examine a different trust boundary: the security scanners that assess MCP Servers themselves.

Prompt Injection and Adversarial Context. Prompt injection has been extensively studied in LLM-integrated applications and agents (Abdelnabi et al., 2023; Liu et al., 2024; Debenedetti et al., 2024; Evtimov et al., 2025). Related work considers prompt injection against model-mediated assessment (Shi et al., 2024), structured separation of trusted instructions and untrusted data (Chen et al., 2025a; Jia et al., 2025), and malicious tool documents (Shi et al., 2026). Adversarial context can also be introduced through poisoned agent memory, knowledge bases, or retrieved knowledge (Chen et al., 2024; Zou et al., 2025), while complementary work studies detection, mitigation, and retrieval of injected content (Chen et al., 2025c;d; Chang et al., 2026; Chen et al., 2025b; Liu et al., 2025). CPI instead targets scanner judgments using vulnerability-specific false remediation claims that contradict the executable code.

LLM-Based Code and Vulnerability Analysis. LLMs have been studied for code analysis and security tasks, including obfuscated-code analysis, task-driven web scanning, program analysis, and vulnerability detection under realistic settings (Fang et al., 2024; Stafeev et al., 2025; Sun et al., 2024; Lin & Mohaisen, 2025; Ding et al., 2025). Other work incorporates vulnerability-relevant program context (Lekssays et al., 2025) or studies security hardening, adversarial testing, calibration, and code understanding (He & Vechev, 2023; Spiess et al., 2025; Nam et al., 2024). Also, code comments may diverge from implementation behavior (Rong et al., 2025), and LLM-based static analysis can be vulnerable to adversarial patterns that exploit abstraction bias (Li et al., 2025; Bernstein et al., 2026). We complement these studies by examining whether misleading developer-controlled natural language can override contradictory executable evidence in MCP security scanning.

6 Conclusion

This paper identifies developer-controlled context in MCP Server as an exploitable attack surface in LLM-based MCP Server scanners. We introduce Contextual Prompt Injection (CPI), a black-box attack that constructs vulnerability-specific false remediation narratives. Our results show that scanners can be successfully misled by attacker-authored security claims even when they contradict the actual implementation. LLM-based security scanners should therefore prioritize executable code over untrusted documentation and verify claimed fixes against the implementation.

AI use statement

Generative AI tools were used in this work to assist with selected research tasks. In particular, LLMs were used to support repository auditing and validating during dataset construction and to assist with processing scanner findings and generating contextual payloads in the CPI experimental workflow. Generative AI tools were also used for language editing and improving the readability of this paper. All AI-assisted outputs relevant to the research process were reviewed and, where applicable, validated by the authors against the underlying repository evidence, source code, and experimental

results. The authors independently assessed the scientific claims and conclusions presented in this paper and take full responsibility for the final content of the work.

Ethics statement

This work investigates the robustness of LLM-based MCP Server scanners against contextual prompt injection and therefore has a dual-use nature. While CPI is designed to expose weaknesses in existing security scanners, similar techniques could potentially be used to evade vulnerability detection. Our goal is to characterize this attack surface and provide empirical evidence that can support the development of more robust scanning systems. To limit potential harm, all experiments were conducted in a controlled local environment using publicly available GitHub repositories and existing public benchmarks. All CPI modifications were applied only to local copies, and we did not conduct attacks against deployed MCP services or other live third-party systems. Overall, our findings highlight the need for LLM-based scanners to treat developer-controlled context as untrusted input and to verify security claims against executable evidence.

Reproducibility statement

To facilitate reproducibility, Section 2 describes the data sources, repository selection criteria, dataset construction procedure, and the manually verified vulnerability subset. Section 3.1 defines the threat model and attack objective, while Section 3.2 describes the complete CPI workflow, including target finding extraction, contextual payload construction, and repository injection. Section 4.1 specifies the datasets, evaluated scanners, and evaluation metrics, including the replay setup used for endpoint-based scanning. Appendix A provides an end-to-end case study illustrating the attack and evaluation procedure on a real-world repository. We additionally release the implementation of CPI at `https://github.com/ContextualPromptInjection/CPI` to support reproduction and extension of the proposed attack methodology. Together, the methodological descriptions and released implementation support reproduction of the CPI workflow and experimental protocol, while Section 2 provides the procedures required to reconstruct the evaluation dataset.

References

Sahar Abdelnabi, Kai Greshake, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In Maura Pintor, Xinyun Chen, and Florian Tramèr (eds.), *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, AISec 2023, Copenhagen, Denmark, 30 November 2023*, pp. 79–90. ACM, 2023. doi: 10.1145/3605764.3623985. URL `https://doi.org/10.1145/3605764.3623985`.

Anthropic. Introducing the Model Context Protocol, November 2024. URL `https://www.anthropic.com/news/model-context-protocol`. Accessed: 2026-08-06.

Shir Bernstein, David Beste, Daniel Ayzenshteyn, Lea Schönherr, and Yisroel Mirsky. Trust me, I know this function: Hijacking LLM static analysis using bias. In *33rd Annual Network and Distributed System Security Symposium, NDSS 2026, San Diego, California, USA, February 23-27, 2026*. The Internet Society, 2026. URL `https://dx.doi.org/10.14722/ndss.2026.242066`.

Hongyan Chang, Ergute Bao, Xinjian Luo, and Ting Yu. Overcoming the retrieval barrier: Indirect prompt injection in the wild for LLM systems. *CoRR*, abs/2601.07072, 2026. doi: 10.48550/ARXIV.2601.07072. URL `https://doi.org/10.48550/arXiv.2601.07072`.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David A. Wagner. Struq: Defending against prompt injection with structured queries. In Lujo Bauer and Giancarlo Pellegrino (eds.), *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, pp. 2383–2400. USENIX Association, 2025a. URL `https://www.usenix.org/conference/usenixsecurity25/presentation/chen-sizhe`.

Sizhe Chen, Arman Zharmagambetov, Saeed Mahloujifar, Kamalika Chaudhuri, David A. Wagner, and Chuan Guo. Secalign: Defending against prompt injection with preference optimization.

In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier (eds.), *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025*, pp. 2833–2847. ACM, 2025b. doi: 10.1145/3719027.3744836. URL https://doi.org/10.1145/3719027.3744836.

Yulin Chen, Haoran Li, Yuan Sui, Yufei He, Yue Liu, Yangqiu Song, and Bryan Hooi. Can indirect prompt injection attacks be detected and removed? In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 18189–18206. Association for Computational Linguistics, 2025c. doi: 10.18653/V1/2025.ACL-LONG.890. URL https://doi.org/10.18653/v1/2025.acl-long.890.

Yulin Chen, Haoran Li, Zihao Zheng, Dekai Wu, Yangqiu Song, and Bryan Hooi. Defense against prompt injection attack by leveraging attack techniques. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 18331–18347. Association for Computational Linguistics, 2025d. doi: 10.18653/V1/2025.ACL-LONG.897. URL https://doi.org/10.18653/v1/2025.acl-long.897.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. Agentpoison: Red-teaming LLM agents via poisoning memory or knowledge bases. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/eb113910e9c3f6242541c1652e30dfd6-Abstract-Conference.html.

Cisco AI Defense. MCP Scanner. GitHub repository, 2025. URL https://github.com/cisco-ai-defense/mcp-scanner. Accessed: 2026-09-06.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html.

Yangruibo Ding, Yanjun Fu, Omniyyah Ibrahim, Chawin Sitawarin, Xinyun Chen, Basel Alomair, David A. Wagner, Baishakhi Ray, and Yizheng Chen. Vulnerability detection with code language models: How far are we? In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*, pp. 1729–1741. IEEE, 2025. doi: 10.1109/ICSE55347.2025.00038. URL https://doi.org/10.1109/ICSE55347.2025.00038.

Ivan Evtimov, Arman Zharmagambetov, Aaron Grattafiori, Chuan Guo, and Kamalika Chaudhuri. WASP: benchmarking web agent security against prompt injection attacks. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/1c9818387f5dd0a0bc151214660f059d-Abstract-Datasets_and_Benchmarks_Track.html.

Chongzhou Fang, Ning Miao, Shaurya Srivastav, Jialin Liu, Ruoyu Zhang, Ruijie Fang, Asmita, Ryan Tsang, Najmeh Nazari, Han Wang, and Houman Homayoun. Large language models

for code analysis: Do llms really do their job? In Davide Balzarotti and Wenyuan Xu (eds.), *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024. URL https://www.usenix.org/conference/usenixsecurity24/presentation/fang.

Mohammed Mehedi Hasan, Hao Li, Emad Fallahzadeh, Gopi Krishnan Rajbahadur, Bram Adams, and Ahmed E. Hassan. Model context protocol (MCP) at first glance: Studying the security and maintainability of MCP servers. *CoRR*, abs/2506.13538, 2025. doi: 10.48550/ARXIV.2506.13538. URL https://doi.org/10.48550/arXiv.2506.13538.

Jingxuan He and Martin T. Vechev. Large language models for code: Security hardening and adversarial testing. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda (eds.), *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pp. 1865–1879. ACM, 2023. doi: 10.1145/3576915.3623175. URL https://doi.org/10.1145/3576915.3623175.

Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (MCP): landscape, security threats, and future research directions. *CoRR*, abs/2503.23278, 2025. doi: 10.48550/ARXIV.2503.23278. URL https://doi.org/10.48550/arXiv.2503.23278.

Feiran Jia, Tong Wu, Xin Qin, and Anna Cinzia Squicciarini. The task shield: Enforcing task alignment to defend against indirect prompt injection in LLM agents. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 29680–29697. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.ACL-LONG.1435. URL https://doi.org/10.18653/v1/2025.acl-long.1435.

Ahmed Lekssays, Hamza Mouhcine, Khang Tran, Ting Yu, and Issa Khalil. Llmxcpg: Context-aware vulnerability detection through code property graph-guided large language models. In Lujo Bauer and Giancarlo Pellegrino (eds.), *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, pp. 489–507. USENIX Association, 2025. URL https://www.usenix.org/conference/usenixsecurity25/presentation/lekssays.

Xiao Li, Yue Li, Hao Wu, Yue Zhang, Kaidi Xu, Xiuzhen Cheng, Sheng Zhong, and Fengyuan Xu. Make a feint to the east while attacking in the west: Blinding llm-based code auditors with flashboom attacks. In Marina Blanton, William Enck, and Cristina Nita-Rotaru (eds.), *IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025*, pp. 576–594. IEEE, 2025. doi: 10.1109/SP61157.2025.00125. URL https://doi.org/10.1109/SP61157.2025.00125.

Jie Lin and David Mohaisen. From large to mammoth: A comparative evaluation of large language models in vulnerability detection. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society, 2025. URL https://dx.doi.org/10.14722/ndss.2025.241491.

Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses. In Davide Balzarotti and Wenyuan Xu (eds.), *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024. URL https://www.usenix.org/conference/usenixsecurity24/presentation/liu-yupei.

Yupei Liu, Yuqi Jia, Jinyuan Jia, Dawn Song, and Neil Zhenqiang Gong. Datasentinel: A game-theoretic detection of prompt injection attacks. In Marina Blanton, William Enck, and Cristina Nita-Rotaru (eds.), *IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025*, pp. 2190–2208. IEEE, 2025. doi: 10.1109/SP61157.2025.00250. URL https://doi.org/10.1109/SP61157.2025.00250.

Daye Nam, Andrew Macvean, Vincent J. Hellendoorn, Bogdan Vasilescu, and Brad A. Myers. Using an LLM to help with code understanding. In *Proceedings of the 46th IEEE/ACM International*

Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024, pp. 97:1–97:13. ACM, 2024. doi: 10.1145/3597503.3639187. URL https://doi.org/10.1145/3597503.3639187.

Postmark Team. Information regarding malicious "postmark-mcp" package. Postmark Blog, September 2025. URL https://postmarkapp.com/blog/information-regarding-malicious-postmark-mcp-package. Accessed: 2026-09-06.

Brandon Radosevich and John Halloran. MCP safety audit: Llms with the model context protocol allow major security exploits. *CoRR*, abs/2504.03767, 2025. doi: 10.48550/ARXIV.2504.03767. URL https://doi.org/10.48550/arXiv.2504.03767.

Guoping Rong, Yongda Yu, Song Liu, Xin Tan, Tianyi Zhang, Haifeng Shen, and Jidong Hu. Code comment inconsistency detection and rectification using a large language model. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*, pp. 1832–1843. IEEE, 2025. doi: 10.1109/ICSE55347.2025.00035. URL https://doi.org/10.1109/ICSE55347.2025.00035.

Jiawen Shi, Zenghui Yuan, Yinuo Liu, Yue Huang, Pan Zhou, Lichao Sun, and Neil Zhenqiang Gong. Optimization-based prompt injection attack to llm-as-a-judge. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (eds.), *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pp. 660–674. ACM, 2024. doi: 10.1145/3658644.3690291. URL https://doi.org/10.1145/3658644.3690291.

Jiawen Shi, Zenghui Yuan, Guiyao Tie, Pan Zhou, Neil Zhenqiang Gong, and Lichao Sun. Prompt injection attack to tool selection in LLM agents. In *33rd Annual Network and Distributed System Security Symposium, NDSS 2026, San Diego, California, USA, February 23-27, 2026*. The Internet Society, 2026. URL https://www.ndss-symposium.org/ndss-paper/prompt-injection-attack-to-tool-selection-in-llm-agents/.

Snyk. Agent Scan: Security Scanner for AI Agents, MCP Servers and Agent Skills. GitHub repository, 2026. URL https://github.com/snyk/agent-scan. Accessed: 2026-09-06.

Claudio Spiess, David Gros, Kunal Suresh Pai, Michael Pradel, Md. Rafiqul Islam Rabin, Amin Alipour, Susmit Jha, Prem Devanbu, and Toufique Ahmed. Calibration and correctness of language models for code. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*, pp. 540–552. IEEE, 2025. doi: 10.1109/ICSE55347.2025.00040. URL https://doi.org/10.1109/ICSE55347.2025.00040.

Aleksei Stafeev, Tim Recktenwald, Gianluca De Stefano, Soheil Khodayari, and Giancarlo Pellegrino. Yurascanner: Leveraging llms for task-driven web app scanning. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society, 2025. URL https://www.ndss-symposium.org/ndss-paper/yurascanner-leveraging-llms-for-task-driven-web-app-scanning/.

Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Haijun Wang, Zhengzi Xu, Xiaofei Xie, and Yang Liu. Gptscan: Detecting logic vulnerabilities in smart contracts by combining GPT with program analysis. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pp. 166:1–166:13. ACM, 2024. doi: 10.1145/3597503.3639117. URL https://doi.org/10.1145/3597503.3639117.

Tencent Zhuque Lab. AI-Infra-Guard: A Comprehensive, Intelligent, and Easy-to-Use AI Red Teaming Platform. GitHub repository, 2025. URL https://github.com/Tencent/AI-Infra-Guard. Accessed: 2026-09-06.

Zhiqiang Wang, Yichao Gao, Yanting Wang, Suyuan Liu, Haifeng Sun, Haoran Cheng, Guanquan Shi, Haohua Du, and Xiangyang Li. Mcptox: A benchmark for tool poisoning on real-world MCP servers. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor (eds.), *Fortieth AAAI Conference*

on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026, pp. 35811–35819. AAAI Press, 2026a. doi: 10.1609/AAAI.V40I42.40895. URL https://doi.org/10.1609/aaai.v40i42.40895.

Zihan Wang, Rui Zhang, Yu Liu, Wenshu Fan, Wenbo Jiang, Qingchuan Zhao, Hongwei Li, and Guowen Xu. MPMA: preference manipulation attack against model context protocol. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor (eds.), *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pp. 35838–35846. AAAI Press, 2026b. doi: 10.1609/AAAI.V40I42.40898. URL https://doi.org/10.1609/aaai.v40i42.40898.

Mengying Wu, Pei Chen, Geng Hong, Baichao An, Jinsong Chen, Binwang Wan, Xudong Pan, Jiarun Dai, and Min Yang. Mcpzoo: A large-scale dataset of runnable model context protocol servers for AI agent. *CoRR*, abs/2512.15144, 2025. doi: 10.48550/ARXIV.2512.15144. URL https://doi.org/10.48550/arXiv.2512.15144.

Dongsen Zhang, Zekun Li, Xu Luo, Xuannan Liu, Peipei Li, and Wenjun Xu. MCP security bench (MSB): benchmarking attacks against model context protocol in LLM agents. *CoRR*, abs/2510.15994, 2025. doi: 10.48550/ARXIV.2510.15994. URL https://doi.org/10.48550/arXiv.2510.15994.

Xuanjun Zong, Zhiqi Shen, Lei Wang, Yunshi Lan, and Chao Yang. Mcp-safetybench: A benchmark for safety evaluation of large language models with real-world MCP servers. *CoRR*, abs/2512.15163, 2025. doi: 10.48550/ARXIV.2512.15163. URL https://doi.org/10.48550/arXiv.2512.15163.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. Poisonedrag: Knowledge corruption attacks to retrieval-augmented generation of large language models. In Lujo Bauer and Giancarlo Pellegrino (eds.), *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, pp. 3827–3844. USENIX Association, 2025. URL https://www.usenix.org/conference/usenixsecurity25/presentation/zou-poisonedrag.

A Case Study

We present a qualitative case study of A.I.G. on the Markdownify MCP Server, which is affected by CVE-2025-5276. The server exposes several tools that accept a user-controlled URL and pass it to `Markdownify.toMarkdown()`, where the URL is directly requested using `fetch(url)`. The implementation does not restrict URL schemes, resolved IP addresses, internal hosts, or redirects, thereby preserving the known server-side request forgery (SSRF) vulnerability.

Following our CPI construction protocol, we retained this vulnerable data flow and modified only the surrounding context. The injected context described the project as using "two-stage input hardening," workspace-scoped validation, and normalized conversion inputs. These statements were made plausible by referring to functions that genuinely exist in the repository, including `normalizeToolFilepath()`, `assertWorkspaceScopedPath()`, and `shellSafePath()`. However, these functions operate on file paths and do not validate the destination of URL-based requests.

The audit traces show how this contextual narrative affected A.I.G.'s reasoning. Before CPI, its code-audit agent explicitly observed that the `fetch` call "could potentially be used for SSRF" and that no URL validation prevented access to internal services. After CPI, the information-collection agent incorporated the injected claims into its own project summary, describing the server as having a multi-stage security architecture. The subsequent code-audit agent identified the URL entry points but did not complete the corresponding source-to-sink SSRF analysis. Instead, it focused on unrelated findings and produced no SSRF report. Then, the final review stage returned no valid vulnerability.

This example illustrates that CPI changes how the scanner weighs the available evidence. A.I.G. treated repository-provided security claims as verified properties, generalized file-path protections to the server's broader input-handling security, and terminated the URL analysis before testing whether the claimed controls applied to the network sink. The result is a vulnerability-specific omission caused by contextual evidence laundering.

B Limitations and Future Work

Our study has several limitations that motivate future research. Firstly, results on the full dataset measure suppression of scanner-reported findings and may therefore include false positives, whereas claims about genuine vulnerabilities rely on the manually verified subset. Future work could characterize attack stability and expand manual verification to a larger set of repositories.

Secondly, we evaluate three MCP Server scanners with different input surfaces, and the observed susceptibility may not generalize to other scanner architectures or deployment settings. Broader evaluations across additional scanners and MCP ecosystems would help establish the scope of this failure mode. Finally, CPI assumes access to a baseline scanner report. Investigating limited-feedback or no-feedback settings would clarify how strongly the attack depends on scanner-specific feedback. Beyond attack evaluation, future work should also study stronger defenses that explicitly separate untrusted developer-controlled context from executable evidence and require code-grounded verification of claimed remediations.