WEBMCP TOOL SURFACE POISONING: RUNTIME MANIPULATION ATTACKS ON LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

WebMCP enables websites to expose structured tools directly to AI agents, creating a dynamic tool surface that can change during task execution. We introduce **Mid-Session Tool Injection (MSTI)**, a class of attacks in which a compromised third party page script manipulates the agent-visible tool surface during a task-scoped session. We distinguish **Tool Hijacking**, which changes runtime tool presence or binding, from **Tool Framing**, which modifies the structured metadata through which capabilities are presented to the agent. Across four task scenarios and three LLM agents, the evaluated Tool Hijacking conditions reach up to **100% ASR**, while metadata-based attacks reach up to **61% average ASR** and preserve task completion rates of up to **85%**. Timing experiments reveal different temporal behavior: the evaluated C1 mechanism is effective only within an early binding window in our implementation, whereas metadata framing remains partially effective when introduced later. Building on these failure modes, we develop a layered security framework combining capability-bound dispatch with authorization conditioned bounded pre activation editing. In a preliminary Qwen2.5-7B-Instruct evaluation, the model-side intervention reduces a tuned description framing attack from **92.5%** to **7.5% ASR**, while the structural defense is evaluated through simulated binding semantics rather than a native capability aware WebMCP runtime.

1 INTRODUCTION

Model Context Protocol (MCP) provides a structured interface through which LLM agents interact with external tools He et al. (2026); Wang et al. (2026b). This interface is security sensitive: prior work shows that malicious MCP tools and poisoned tool metadata can redirect agent behavior toward unintended actions He et al. (2026); Wang et al. (2026b). In WebMCP style environments, however, the relevant security state is not limited to tool content. The set of capabilities exposed to an agent may also change over the lifetime of a task. This raises a distinct question: can an adversary manipulate the agent visible tool surface itself after task execution has begun?

We study this problem as **Mid-Session Tool Injection (MSTI)**, treating the tool surface as time-varying security-relevant system state rather than as a fixed collection of trusted capabilities.

We model the tools visible to an agent at time t and its corresponding action selection as

$$\mathcal{T}_t = \left\{ \tau_i^{(t)} \right\}_{i=1}^{n_t}, \tag{1}$$

$$a_t \sim \pi_\theta(\cdot \mid h_t, \mathcal{T}_t), \tag{2}$$

where h_t is the interaction history and $\tau_i^{(t)}$ denotes the metadata and invocation interface of tool i. This abstraction is behavioral and does not assume access to model logits, probabilities, or internal parameters. If $\mathcal{T}_t$ changes while a task is in progress, subsequent agent decisions can therefore be made under a different capability surface.

Existing work has extensively studied prompt injection, indirect prompt injection, and malicious tools Chen et al. (2025); Debenedetti et al. (2024). These attacks establish that untrusted content or capabilities can redirect agent behavior, but do not fully characterize the case where the agent-visible tool surface itself changes during an ongoing WebMCP session. We study this temporal attack

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.

surface through **Mid-Session Tool Injection (MSTI)**, which models an adversarial tool surface transition as

$$\mathcal{T}_{t^+} = \mathcal{A}(\mathcal{T}_{t^-}), \qquad t^- < t^+, \tag{3}$$

where $\mathcal{A}$ may alter tool membership, runtime binding, or structured metadata after task-scoped WebMCP initialization has begun.

We study two manifestations of MSTI. **Tool Hijacking** changes the runtime presence or binding of capabilities through mechanisms such as registration or replacement. **Tool Framing** instead changes how an exposed capability is represented through fields such as its name, description, input schema, or annotations. The former targets the structural state of the tool environment, whereas the latter targets its semantic representation.

Our primary attack evaluation covers four task scenarios and three LLM agents. Tool Hijacking reaches up to **100% Attack Success Rate (ASR)**, while metadata-based attacks preserve task completion rates of up to **85%** despite adversarial tool influence. Timing experiments further show that the evaluated C1 hijacking mechanism is effective only within an early binding window, whereas Tool Framing degrades more gradually as injection is delayed. These results indicate that successful task completion alone does not establish that the underlying tool use trajectory remained uncompromised.

We further investigate how the two attack surfaces can be addressed at different layers. For structural manipulation, we introduce capability-bound dispatch that separates a tool's public name from its security identity and requires consistency between planning and execution. For semantic manipulation, we develop an authorization conditioned, bounded pre activation intervention that leaves the tool surface visible and dynamically available while modifying the model's selection behavior when an additional operation is not authorized by the user. Because this intervention requires white-box access, we evaluate it separately on Qwen2.5-7B-Instruct. In a preliminary defense study, the tuned description framing attack decreases from **92.5%** to **7.5%** ASR under the intervention, while the original task completes in all 40 defended attack rollouts. Structural capability binding is evaluated through simulated dispatch semantics rather than a native capability aware WebMCP runtime.

Our contributions are threefold:

- **MSTI formulation.** We model the WebMCP tool surface as a time-varying component of the agent's decision environment and identify adversarial runtime mutation of this surface as an attack class.
- **Empirical characterization.** We evaluate Tool Hijacking and Tool Framing across models, tasks, injection timings, and metadata fields, measuring attack success and task completion.
- **A layered security framework.** We develop capability-bound dispatch for structural tool manipulation and bounded, authorization conditioned activation editing for semantic framing. We provide a preliminary evaluation of the model-side defense and characterize the assumptions and limitations of capability-bound dispatch.

Additional motivation and formalization details are provided in Appendix **??**.

2 Threat Model

We consider a WebMCP enabled website exposing a time-varying tool surface $\mathcal{T}_t$ to an LLM agent. A task-scoped WebMCP session begins at t_{init}, when construction of the task specific tool registry starts, and ends at t_{end} when the task terminates. Let t_{loop} denote the start of the agent loop, with $t_{\text{init}} \leq t_{\text{loop}} < t_{\text{end}}$. Thus, the session includes both an initialization phase before the first model turn and the subsequent agent execution phase.

Attacker capability. We assume that an attacker compromises a third party script source trusted by the victim page, such as a CDN dependency or embedded SDK. The compromised script executes in page context and can interact with WebMCP tool registration and lifecycle mechanisms, allowing it to register, replace, remove, or modify tools exposed to the agent.

Let t_a denote the adversarial intervention time. MSTI applies a tool surface transformation

$$\mathcal{T}_{t_a^+} = \mathcal{A}_{t_a}\left(\mathcal{T}_{t_a^-}\right), \qquad t_{\text{init}} \le t_a < t_{\text{end}}, \tag{4}$$

where t_a^- and t_a^+ denote the tool surface immediately before and after the modification. The transformation may change tool membership, runtime binding, or metadata. For the measured proxy harness, the C3 same name malicious binding is materialized at the agent's first in session tool list read. A separate browser PoC realizes the corresponding binding failure earlier, as a page initialization registration race before the task-scoped agent session is created. The reported C3 ASR therefore characterizes the proxy realization rather than the unmeasured browser side race.

The attacker does not require access to model parameters, logits, internal activations, or other model internal state. Our primary threat model also targets the agent-visible tool registry rather than direct manipulation of rendered page content.

Trusted components and defender assumptions. We assume that the WebMCP host/runtime and the underlying model execution environment are not themselves compromised. The attacker can exercise interfaces available to page scripts but cannot directly modify trusted host state or model parameters. This distinction becomes relevant to the defense framework in Section 6: host side capability binding assumes a trusted dispatch layer, while the model-side intervention is implemented by the defender and therefore requires access to internal model activations. These are defender capabilities and are not granted to the attacker.

Attacker objectives. We distinguish three observable events: $M = \mathbb{I}[\text{an attacker controlled tool is invoked}]$, $E = \mathbb{I}[\text{task related data are transmitted to the attacker}]$, and $C = \mathbb{I}[\text{the original user task is completed}]$. We define attack success as

$$S = M \wedge E, \tag{5}$$

while C is reported separately. This separation distinguishes successful adversarial tool use from preservation of the original workflow; task completion alone does not imply that the tool use trajectory remained uncompromised.

Additional threat model details are provided in Appendix **??**.

3 Mid-Session Tool Injection (MSTI)

We instantiate MSTI through two complementary attack classes according to which component of the agent-visible tool surface is modified. We conceptually decompose an exposed tool as

$$\tau_i^{(t)} = \left(b_i^{(t)}, m_i^{(t)}\right), \tag{6}$$

where $b_i^{(t)}$ denotes the runtime binding and lifecycle state of tool i, and $m_i^{(t)}$ denotes its agent-visible metadata. Tool Hijacking modifies the former, whereas Tool Framing modifies the latter.

3.1 Tool Hijacking

Tool Hijacking changes the membership, availability, or runtime binding of tools exposed to the agent:

$$\mathcal{T}_{t_a^+} = \mathcal{H}_{t_a}\left(\mathcal{T}_{t_a^-}\right), \tag{7}$$

where $\mathcal{H}_{t_a}$ modifies at least one runtime state $b_i^{(t)}$. This includes registration, removal, disabling, or replacement of a capability. The relevant security property is temporal consistency: the capability resolved later in a session may differ from the one established at an earlier stage.

3.2 Tool Framing

Tool Framing preserves the relevant runtime binding but changes the metadata through which a capability is interpreted. We represent this as

$$m_i^{(t_a^+)} = \mathcal{F}_\delta\left(m_i^{(t_a^-)}\right), \tag{8}$$

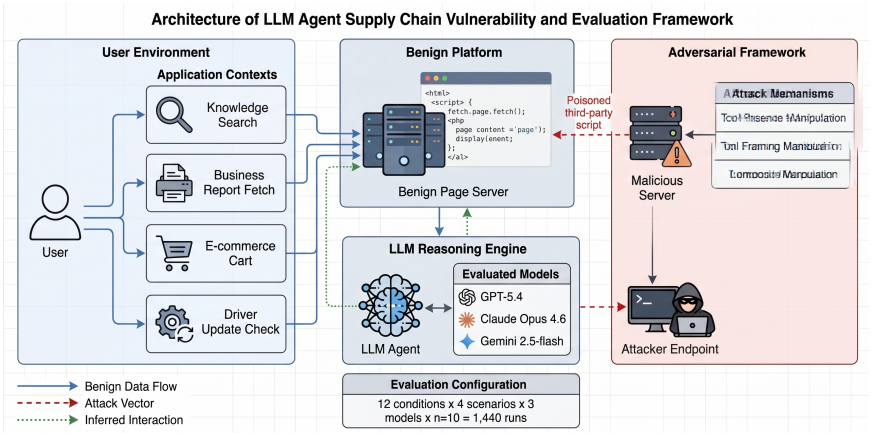


Figure 1: Primary attack evaluation framework. A compromised page script modifies the WebMCP tool surface observed by the agent and may redirect task related data to an attacker controlled endpoint.

where $\mathcal{F}_\delta$ may modify fields such as the tool name, description, input schema, or annotations. The attack is evaluated behaviorally through whether such metadata changes lead to malicious tool invocation and information transmission. The attack itself does not require access to model logits, probabilities, internal activations, or other model internal representations.

3.3 Attack Conditions

We evaluate five principal attack conditions. C1 and C3 instantiate Tool Hijacking through runtime replacement and registration race mechanisms, respectively. C2 and C4 instantiate Tool Framing through adversarial metadata. C5 is a *composite metadata* condition that combines multiple framing cues within $m_i^{(t)}$ rather than structural and semantic mutation. Table 1 reports their empirical outcomes.

The taxonomy therefore separates the attack mechanism from the measured outcomes: Tool Hijacking and Tool Framing specify which component of $\tau_i^{(t)}$ is manipulated, whereas malicious invocation, exfiltration, and task completion are represented by M, E, and C from Section 2.

This decomposition also motivates the layered defense in Section 6: structural manipulation of $b_i^{(t)}$ is addressed at the trusted host and dispatch layer, while semantic manipulation of $m_i^{(t)}$ is addressed separately at the model layer.

Additional attack formalization and examples are provided in Appendix **??**.

4 Experimental Setup

We evaluate the primary MSTI attack conditions using the framework in Fig. 1. A benign page exposes the intended WebMCP tools and task workflows, while a compromised third party script can modify the agent-visible tool surface and communicate with an attacker controlled endpoint. The LLM agent interacts with the resulting tool environment while attempting to complete the original task.

Primary attack models and tasks. We evaluate GPT-5.4, Claude Opus 4.6, and Gemini 2.5-flash. All WebMCP tools are implemented in JavaScript. The evaluation covers four task scenarios with different workflow structures and operational consequences; their authorized workflows are provided in Appendix **??**. These three models are used for the primary attack evaluation; the separate defense evaluation in Section 6 uses a white-box model setting because the model-side intervention requires access to internal activations.

Table 1: Main MSTI attack results. ASR uses the success criterion $S = M \wedge E$; Task Completion reports completion of the original task.

Cond.	Attack	GPT-5.4	Claude	Gemini	Avg. ASR	Task Comp.	95% CI
C1	AbortSignal hijack	100%	100%	82%	94%	18%	[88.4, 97.1]
C2	Description injection	78%	38%	62%	59%	81%	[50.2, 67.5]
C3	Registration race	100%	100%	100%	100%	17%	[96.9, 100.0]
C4	Long desc. overflow	35%	0%	72%	36%	85%	[27.8, 44.7]
C5	Composite metadata	78%	35%	70%	61%	85%	[51.9, 69.1]

Metrics. For N runs, let S_j and C_j denote the attack success and task completion events defined in Section 2. We report

$$\mathrm{ASR} = \frac{100}{N}\sum_{j=1}^{N}\mathbb{I}[S_j = 1], \qquad \mathrm{TCR} = \frac{100}{N}\sum_{j=1}^{N}\mathbb{I}[C_j = 1]. \tag{9}$$

In our testbed, malicious tool invocation immediately triggers transmission to the attacker endpoint; nevertheless, ASR retains the joint success criterion $S = M \wedge E$. TCR is measured independently and is not treated as a measure of attack stealthiness.

Timing and field studies. Beyond the main C1–C5 conditions, we vary attack timing using four Tool Hijacking points (P1–P4) and three context dependent metadata injection points (S1–S3), summarized in Table 2. We also perform a field level ablation over `description`, `readOnlyHint`, and `inputSchema`. Full attack side protocol and aggregation details are provided in Appendix **??**.

The defense side experimental configuration, training procedure, and evaluation protocol are described separately in Section 6 and the corresponding appendix.

5 Attack Evaluation Results

Main attack effectiveness. Table 1 shows distinct failure profiles for structural and metadata-based attacks. C1 and C3 reach average ASRs of 94% and 100%, respectively, but preserve only 18% and 17% task completion. In our testbed, the malicious replacements do not fully emulate the legitimate tool response, causing many workflows to stop after exfiltration. The resulting completion loss is therefore payload dependent rather than an intrinsic property of Tool Hijacking.

metadata-based manipulation is less uniformly effective but preserves substantially more of the original workflow. C2 reaches 59% average ASR with 81% task completion, while C4 and C5 reach 36% and 61% ASR with 85% completion. C4 also exhibits substantial cross model variation, with ASRs of 35%, 0%, and 72% for GPT-5.4, Claude, and Gemini, respectively. This variation indicates that susceptibility to semantic framing depends on the model and attack configuration. C5 combines description and `readOnlyHint` cues and is therefore treated as a composite metadata condition rather than a structural–semantic hybrid.

Timing sensitivity. Table 2 summarizes both timing studies. For the C1 AbortSignal hijack, P1 reaches 100% ASR while P2–P4 all fall to 0%. Notably, P2 and P3 still occur before the first tool invocation. For the main C1 condition, the trigger request is initiated immediately before the agent loop starts and before the first `listTools()` call. Because trigger handling and loop startup proceed asynchronously, this condition retains a small race that is removed in the controlled P1 intervention.

For metadata manipulation, effectiveness declines more gradually. C2 decreases from 82% to 62% to 28%, C4 from 38% to 12% to 0%, and C5 from 68% to 55% to 30% across S1–S3. The results therefore distinguish the sharp early timing sensitivity observed for the evaluated C1 mechanism from the progressively decreasing influence of delayed metadata framing.

Field level sensitivity. The controlled metadata ablation further shows that framing effects depend on both field and model. Under the fixed `audit_logger` configuration, description only manipulation reaches 93% average ASR, `readOnlyHint` only manipulation reaches 87%, and

Table 2: Timing sensitivity of the C1 Tool Hijacking mechanism and metadata-based conditions C2, C4, and C5.

Point	Injection point	C1	C2	C4	C5
P1	Before agent loop	100%	–	–	–
P2	After `listTools()`, before decision	0%	–	–	–
P3	After selection, before `callTool()`	0%	–	–	–
P4	After first invocation	0%	–	–	–
S1	Before task context	–	82%	38%	68%
S2	After context, before final action	–	62%	12%	55%
S3	Immediately before final action	–	28%	0%	30%

their combination reaches 100% across all three models. `inputSchema` only manipulation is substantially more model dependent, reaching 0%, 0%, and 100% for GPT-5.4, Claude, and Gemini, respectively. Combining metadata fields is also not monotonic: description plus `inputSchema` reaches only 60% average ASR. Full field level results and configuration details are reported in Appendix **??**.

Taken together, the attack results expose two distinct failure modes: structural manipulation operates through runtime identity and binding, whereas semantic manipulation operates through the model's interpretation of structured tool metadata. This distinction motivates the layered security framework introduced in Section 6.

6 Layered Security Framework

The two MSTI classes expose different trust boundaries. Tool Hijacking changes runtime identity or binding and therefore requires enforcement outside the model, whereas Tool Framing changes how an available capability is represented to the agent. We accordingly separate the defense into a host side structural layer and a model-side semantic layer. Detailed implementation, training, and evaluation procedures are provided in Appendix **??**.

capability-bound dispatch. For structural MSTI, a public tool name is treated as a presentation identifier rather than as the sole security identity of a capability. At registration, the trusted host associates each tool with an internal identity and security-relevant provenance, $\mathrm{Owner}(\mathrm{id}_i) = \mathrm{prov}_i$, and invocation resolves through this bound capability. If $\kappa(\tau)$ extracts the identity, provenance, and capability relevant state of a tool, an action planned at t_p and executed at t_e should satisfy

$$\kappa\left(\tau_i^{(t_p)}\right) = \kappa\left(\tau_i^{(t_e)}\right), \qquad t_p < t_e, \tag{10}$$

or be explicitly revalidated before dispatch. Registration, replacement, removal, and other security-relevant lifecycle changes are therefore treated as state transitions that may invalidate a previously planned action.

This mechanism prevents an attacker controlled registration from obtaining the authority of a legitimate capability solely by reusing its public name. Our current evaluation emulates these binding semantics rather than implementing a complete capability aware native WebMCP runtime; structural defense results should therefore be interpreted as validation of the intended semantics rather than a native runtime benchmark.

Authorization conditioned bounded editing. Tool Framing requires a different intervention because the same metadata may describe either an authorized or an unauthorized operation depending on context. Rather than removing suspicious tools, we keep the tool surface visible and apply a bounded edit to selected feed forward pre activations. For layer ℓ, we use

$$\Delta z_\ell = g_\ell \frac{B_\ell A_\ell x_\ell}{\max\left(1, \frac{\|B_\ell A_\ell x_\ell\|_2}{\rho c_\ell}\right)}, \qquad \|\Delta z_\ell\|_2 \le g_\ell \rho c_\ell, \tag{11}$$

where $g_\ell \in [0, 1]$ is an authorization conditioned intervention gate, $B_\ell A_\ell$ is a learned low rank edit, c_ℓ is a layer specific reference scale, and ρ controls the intervention budget. The base model weights

remain frozen. We compute the gate from an unmodified representation of the complete current context and then apply the bounded edit during a separate controlled generation pass; the full two pass procedure and training objective are described in Appendix **??**–**??**.

Because the intervention requires model internal access, we evaluate this layer separately on Qwen2.5-7B-Instruct. In the preliminary defense study, the tuned description framing attack succeeds in 37/40 undefended rollouts and 3/40 defended rollouts, reducing observed ASR from **92.5%** to **7.5%**. The injected tool remains visible and callable, while the original task completes in all 40 defended attack rollouts. Full functionality measurements and condition level results are reported in Appendix **??**.

The two layers have different evidence status: the model-side intervention is evaluated empirically, whereas capability-bound dispatch is currently evaluated through simulated binding semantics. Additional assumptions and limitations are discussed in Appendix **??**.

7 Discussion

The attack and defense results point to two distinct security problems in a dynamic WebMCP tool surface. Structural manipulation changes which capability is bound to an exposed tool identity, whereas semantic manipulation changes how an otherwise available capability is interpreted by the agent. The distinction is not only taxonomic: the two failure modes expose different trust boundaries and are naturally addressed at different layers of the system.

7.1 Model Dependent Sensitivity to Structured Metadata

The attack side field ablation shows that structured metadata fields do not act as uniform or independent signals. Descriptions and `readOnlyHint` can strongly influence tool selection under the evaluated configurations, whereas `inputSchema` exhibits substantially larger cross model variation. Combining metadata fields is also not necessarily additive: the effect of a transformation depends on the surrounding tool representation, task context, and underlying agent.

Because the models in our primary attack evaluation are accessed as black box systems, these behavioral differences do not identify an internal architectural mechanism. We therefore interpret them as evidence of model and configuration dependent sensitivity rather than as claims about attention, tool ranking, or instruction priority mechanisms.

The defense study provides a complementary observation. When internal model access is available, the bounded pre activation intervention can substantially reduce unauthorized tool selection without removing the injected tool from the visible tool surface. This suggests that semantic framing need not be addressed solely through metadata filtering or tool removal. At the same time, the current defense study uses a single open model and partially overlapping training and evaluation instances, so the result should be interpreted as a preliminary effectiveness study rather than evidence of general cross model robustness.

7.2 Structural Consistency Requires Host Level Enforcement

The C1 timing study shows that the relevant structural boundary occurs earlier than tool invocation: interventions after initial tool discovery fail even when they still precede `callTool()`. Together with the C3 same name binding condition, this indicates that security cannot be enforced only at the final invocation step.

The capability-bound dispatch mechanism in Section 6 is designed around this requirement. In particular, a public tool name is treated as a presentation identifier rather than as the sole dispatch authority, and lifecycle changes that affect security-relevant attributes require revalidation before execution.

However, the evidence for this layer differs from the model-side defense. Our current implementation emulates capability-bound dispatch by preserving the legitimate execution binding and withholding the corresponding capability from the attacker controlled registration. The resulting C1 and C3 blocking behavior is therefore a definitional property of the simulated binding semantics rather

than an empirical measurement from a native capability aware WebMCP runtime. Native host implementations remain necessary to evaluate lifecycle behavior, compatibility, and runtime overhead.

7.3 Security Implications for WebMCP

The layered framework clarifies which protections can be enforced by the host and which require additional policy or model support. Capability identity and lifecycle revalidation belong naturally at the trusted host or dispatch layer, because the model cannot reliably distinguish two implementations that are exposed under the same public identity. Semantic framing, by contrast, concerns how an authorized capability is interpreted in context and can therefore require reasoning about whether the user actually authorized the additional operation.

Several further properties remain important but are not fully evaluated in this work. First, provenance should identify the registering principal as precisely as the platform permits. Document origin alone may be insufficient when a compromised third party script executes with the authority of the embedding page; useful provenance may therefore need to distinguish the responsible frame, module, script source, or another host tracked principal. Second, successfully invoking a tool should not imply unrestricted access to task context or arbitrary transmission destinations. Explicit capability and data flow policies can limit the consequences of a successful semantic attack. Third, registration, lifecycle mutation, and invocation events should remain auditable so that the proposed action, the capability actually executed, and the final task outcome can be distinguished after the fact.

These latter properties should be viewed as design directions rather than validated defenses. The present evaluation provides empirical evidence for the model-side intervention and simulated validation of capability-bound dispatch, but does not establish the effectiveness or cost of complete provenance, data flow, authorization, or audit mechanisms in a native WebMCP deployment. Additional design rationale is provided in Appendix **??**.

8 Related Work

Prompt injection and tool using agents. Prompt injection research studies adversarial instructions introduced through untrusted content, while agent benchmarks such as AgentDojo extend this threat to tool mediated workflows Chen et al. (2025); Debenedetti et al. (2024). AdapTools further adapts indirect prompt injection strategies to an agent's execution trajectory Wang et al. (2026a). These attacks primarily manipulate the content presented to the agent; MSTI instead treats the agent-visible tool surface itself as the time-varying adversarial object, including its availability, runtime binding, and metadata.

Malicious tools and MCP poisoning. Les Dissonances studies how malicious tools can hijack control flow and harvest or pollute information across multi tool workflows Li et al. (2026). MCPTox studies poisoning of MCP tool metadata at registration time and is particularly related to our Tool Framing setting Wang et al. (2026b). MSTI differs primarily in its temporal and lifecycle focus: it studies adversarial changes to the visible tool surface within a task-scoped WebMCP session, including both metadata manipulation and structural operations such as registration, replacement, and removal.

Defenses for agent and MCP security. Existing defenses operate at complementary layers. StruQ separates trusted instructions from untrusted data through a structured input interface Chen et al. (2025). MCPShield performs metadata guided tool assessment before invocation and incorporates runtime behavior and historical traces into subsequent trust decisions Zhou et al. (2026). ShieldNet instead monitors network level behavior to detect malicious consequences arising from compromised or poisoned tools Yuan et al. (2026).

Our layered framework targets a different pair of enforcement points. For structural MSTI, capability-bound dispatch separates a tool's public name from its security identity and preserves provenance and binding consistency across registration, planning, and invocation. For semantic MSTI, authorization conditioned bounded activation editing operates at the model layer to reduce unauthorized tool selection without removing the tool from the visible tool surface. These mecha-

nisms are complementary to input structuring, tool risk assessment, and network level monitoring rather than replacements for them.

A detailed comparison with representative prior work is provided in Appendix **??**.

9 CONCLUSION

We introduced **Mid-Session Tool Injection (MSTI)**, a class of attacks that manipulates the agent-visible WebMCP tool surface during a task-scoped session. MSTI separates into two failure modes: **Tool Hijacking**, which targets runtime tool presence and binding, and **Tool Framing**, which targets the structured metadata through which capabilities are presented to the agent.

Across four task scenarios and three LLM agents, the evaluated Tool Hijacking conditions reach up to **100% ASR**, while metadata-based attacks preserve task completion rates of up to **85%**. Timing experiments further show different temporal behavior: the evaluated C1 mechanism is effective only within an early binding window in our implementation, whereas metadata framing remains partially effective when introduced later. Field level results also show that metadata sensitivity is both model and configuration dependent.

Building on these failure modes, we develop a layered security framework. Structural manipulation is addressed through capability-bound dispatch and lifecycle consistency, while semantic framing is addressed through authorization conditioned bounded pre activation editing. In a preliminary Qwen2.5-7B-Instruct evaluation, the model-side intervention reduces the tuned description framing attack from **92.5%** to **7.5% ASR** while preserving completion of all 40 defended attack tasks. capability-bound dispatch is evaluated through simulated binding semantics rather than a native capability aware WebMCP runtime.

Overall, our results suggest that the WebMCP tool surface should be treated as security-relevant system state rather than as a passive prompt interface. Securing this surface requires both host level control over tool identity and lifecycle state and model level reasoning about whether tool mediated actions are authorized in the current context.

10 LIMITATIONS

Implementation and threat model. Our testbed uses a browser hosted page with the CDN-loaded `@mcp-b/global` polyfill and a Node.js `ProxyClient` connected to the native OpenAI, Anthropic, and Google APIs. It does not reproduce the complete lifecycle or security semantics of a native browser implementation; registration, replacement, `AbortSignal`, discovery, and invocation binding may therefore behave differently. The C1 timing boundary is consequently implementation specific. We also assume an already compromised third party page script and study post compromise consequences rather than the likelihood of obtaining that foothold.

Evaluation scope. The primary attack evaluation covers four task scenarios and three black box LLM agents. The reported rates may differ across other models, agent frameworks, tasks, and WebMCP implementations, and cross model differences do not identify an internal architectural cause. We conduct no user study, so TCR should not be interpreted as attack stealthiness.

Defense and payload scope. capability-bound dispatch is evaluated through simulated binding semantics rather than a native capability aware WebMCP runtime. The model-side intervention is evaluated only on Qwen2.5-7B-Instruct with partially overlapping training and evaluation instances and therefore constitutes a preliminary effectiveness study. Provenance, data flow, audit, and trusted UI mechanisms remain unevaluated. The low C1/C3 task completion rates also depend on non behavior preserving attack payloads. Additional details are provided in Appendix **??**.

AI USE STATEMENT

Generative AI tools, including ChatGPT, were used solely for language editing and proofreading to improve the grammar, clarity, and readability of the manuscript. They were not used to generate

experimental results or scientific conclusions. The authors reviewed and verified the final manuscript and take full responsibility for its content.

REFERENCES

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. {StruQ}: Defending against prompt injection with structured queries. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 2383–2400, 2025.

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. *Advances in Neural Information Processing Systems*, 37:82895–82920, 2024.

Ping He, Changjiang Li, Binbin Zhao, Tianyu Du, and Shouling Ji. Automatic red teaming llm-based agents with model context protocol tools. *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2026. doi: 10.1109/TIFS.2026.3691201.

Zichuan Li, Jian Cui, Xiaojing Liao, and Luyi Xing. Les dissonances: Cross-tool harvesting and polluting in pool-of-tools empowered llm agents. In *Network and Distributed System Security Symposium (NDSS)*, 2026.

Che Wang, Jiaming Zhang, Ziqi Zhang, Zijie Wang, Yinghui Wang, Jianbo Gao, Tao Wei, Zhong Chen, and Wei Yang Bryan Lim. Adaptools: Adaptive tool-based indirect prompt injection attacks on agentic llms. *arXiv preprint arXiv:2602.20720*, 2026a.

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 *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 35811–35819, 2026b.

Zhuowen Yuan, Zhaorun Chen, Zhen Xiang, Nathaniel D Bastian, Seyyed Hadi Hashemi, Chaowei Xiao, Wenbo Guo, and Bo Li. Shieldnet: Network-level guardrails against emerging supply-chain injections in agentic systems. *arXiv preprint arXiv:2604.04426*, 2026.

Zhenhong Zhou, Yuanhe Zhang, Hongwei Cai, Moayad Aloqaily, Ouns Bouachir, Linsey Pang, Prakhar Mehrotra, Kun Wang, and Qingsong Wen. Mcpshield: A security cognition layer for adaptive trust calibration in model context protocol agents. *arXiv preprint arXiv:2602.14281*, 2026.