WEBMCP-PHALANX: ENFORCING AND CHARACTERIZING TRUST BOUNDARIES FOR BROWSER-INTEGRATED LLM AGENTS

A PREPRINT

Lin-Fa Lee
Department of Institute of Artificial Intelligence Innovation
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
prologue.ii14@nycu.edu.tw

Yi-Yu Chang
Department of Institute of Artificial Intelligence Innovation
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
daniel282907@gmail.com

Chia-Mu Yu
Institute of Electrical and Computer Engineering, National Yang Ming Chiao Tung University, Hsinchu, Taiwan
Hsinchu, Taiwan
chiamuyu@gmail.com

Kuo-Hui Yeh
Department of Institute of Artificial Intelligence Innovation
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
khyeh@nycu.edu.tw

August 26, 2026

The emerging W3C WebMCP standardization proposal enables LLM agents to invoke tools exposed by web pages. In multi-party web environments, however, integrating agent execution into a browser security model centered on the Same-Origin Policy (SOP) raises security concerns that the existing origin-based model does not fully address. In particular, this SOP-centered model provides insufficient provenance and lifecycle guarantees for agent-accessible tools, creating three key risks, i.e. subject-attribution spoofing, uncontrolled tool lifecycles, and semantic prompt injection. To mitigate these risks, we propose WebMCP-Phalanx, a dual-layer agent runtime architecture. The first layer provides a browser-native trust anchor that binds each tool to its registering principal through cryptographically protected capability credentials and propagates provenance labels throughout the tool lifecycle. The second layer separates semantic inspection from privileged tool use. A Quarantine Agent (Q-LLM), which has no authority to invoke tools, examines tool metadata, outputs, and other page-supplied content for prompt-injection attempts. Content that passes this inspection is forwarded to a Privileged Agent (P-LLM) for execution. The Q-LLM's internal state is not exposed to page scripts, including same-origin scripts, preventing attackers from observing the inspection process and adapting their inputs in response.

Empirical evaluation shows that our browser-native ownership mechanism significantly reduces the success rate of revocation and overwrite attacks from 100% to 0%. Furthermore, the dual-agent runtime also blocks all 80 prompt-injection attempts embedded in tool descriptions. For attacks delivered through tool return values, only 2 of 80 attempts remain successful, both caused by tool invocations triggered by malicious tool names before semantic inspection is completed. Across these experiments, the protected runtime preserves task utility, with no statistically significant

difference from the no-attack baseline. Finally, we examine the limits of semantic filtering under a white-box adaptive attacker. The results show that description-based filters can still be bypassed when the attacker places adversarial instructions in tool names and causes the tool to be invoked before inspection. This finding motivates an additional call-timing gate that delays tool invocation until all agent-visible tool metadata has been validated.

1 Introduction

The Web is expected to become a major application domain for AI agents. In response to this trend, the W3C is currently developing WebMCP, a draft specification that enables web pages to expose structured tools for invocation by LLM-based agents. This capability enables agents to go beyond simply reading Web pages and to directly execute actions through the tools provided by the corresponding pages. However, this transition raises an important yet largely overlooked concern. As the execution environment of agents shifts toward the more open Web environment, will this introduce additional attack vectors, create new vulnerabilities that make tasks difficult to complete, or even expose limitations in the formulation of current standards?

Motivated by this concern, we conduct a systematic security analysis of the WebMCP specification and validate our findings in a real-world browser environment. WebMCP moves agent execution into the browser's same origin model, where the site's own main program, third party advertising SDKs, and analytics scripts share a single origin without an explicit trust boundary among them. This execution environment reveals four classes of vulnerability that are not merely deployment-level risks but structural weaknesses rooted in the specification itself. The first is subject attribution failure: the tool list carries no origin, so any script can preempt tool names, revoke others' tools, or swap out the tools the agent calls. The second is lifecycle loss of control: tools persist after an SPA logical page transition. The third is execution opacity: a tool can claim to be read only yet exfiltrate data at execution time. The fourth is semantic injection: a structurally legitimate tool embeds manipulative instructions in its description or returned content. These vulnerabilities can be reproduced consistently, are independent of any particular language model, and cannot be adequately addressed through isolated patches. Instead, they require a systematic defense framework based on the WebMCP specification to be thoroughly eradicated. This is precisely what WebMCP-Phalanx aims to provide.

To address these vulnerabilities, we propose WebMCP-Phalanx, a dynamic and secure multi-agent execution environment based on WebMCP standardization proposal. It consists of two complementary defense layers: the browser layer and the multi-agent layer. The browser layer is responsible for establishing the foundation of trust. Tool ownership is provided by the underlying browser, which cannot be forged by any webpage script, ensuring that only subjects holding valid certificates can register or revoke tools. On top of this, several modules separately handle the lifecycle, origin attribution, and execution monitoring, collectively generating a tool trust label. The multi-agent layer, in contrast, performs semantic content inspection on top of the trust labels generated by the browser layer. We employ multiple LLM-based agents to evaluate the trust label from the browser side. Tools labeled as trusted are routed to a privileged agent with tool calling permissions for direct use, whereas suspicious or insufficiently identified tools are first examined by a quarantine agent without such permissions. Before tool descriptions, input schemas, and post execution returned content reach the privileged agent, the quarantine agent first reads and evaluates their semantics, intercepting attacks that are structurally completely valid but hide injection instructions within the semantic layer. By combining these two complementary defense layers, WebMCP-Phalanx addresses both specification-level structural vulnerabilities and semantic injection attacks.

At the browser layer, our defense mechanisms reduce the success rates of tool revocation and replacement attacks from 100% to 0% under the evaluated settings, while the proposed capability credentials withstand all tested forgery attempts. However, subject attribution remains only partially addressed since any same origin script obtains a capability of its own once it registers a tool, degenerating the origin dimension. We empirically characterize this limitation and identify the additional attribution information that should be incorporated into the WebMCP specification. The multi-agent layer complements the browser-layer defenses by inspecting the actual semantic content associated with each tool, thereby detecting injection attacks that cannot be identified solely from structural trust labels. We further show that against task fitting tool names, merely deleting malicious descriptions without reading the content fails in proportion to name relevance, whereas letting the agent read the content to judge remains unaffected.

The contributions of this paper are as follows:

- We systematically identify and characterize the structural security flaws in the WebMCP specification, demonstrating how its current trust and execution model enables several classes of attacks.
- We propose WebMCP-Phalanx, a secure multi-agent execution environment that combines browser-level trust enforcement with multi-agent semantic inspection to defend against both specification-level vulnerabilities and semantic injection attacks.

- We reveal that existing multi-LLM defense mechanisms rely on an implicit assumption that the system can correctly distinguish trusted content from untrusted content. We show that this assumption does not hold under the current WebMCP design. Furthermore, we characterize the structural boundary of description layer content inspection under a white box adaptive attacker, across two model families and three classifier prompt designs.
- Our study is conducted at a critical stage in the development of WebMCP. By identifying these issues before specification ossification, we provide actionable recommendations for strengthening the specification while fundamental design changes remain feasible.

2 Related Work

Security Challenges in WebMCP. The transition of LLMs to web agents via MCP dismantles traditional single trusted subject security models SERVERS [2025], Jing et al. [2025]. Indirect Prompt Injection (IPI) threats have surged, spanning visual injections Wang et al. [2025a], Cao et al. [2025], conversational manipulations Chen et al. [2025a], Zou et al. [2025], Deng et al. [2023], Zhong et al. [2023], Chang et al. [2025], and isolated data fragments Wang et al. [2025b], Chang et al. [2026]. In WebMCP's multi source environments, attackers can hijack task planning via weaponized tool descriptions Sneh et al. [2025], WANG et al. or pollute subsequent operations Li et al. [2025a]. Operating under the browser's flat Same Origin Policy (SOP) without tool ownership verification leaves agents entirely defenseless.

Bottlenecks of Model Intrinsic Defenses. Initial defenses relied on model intrinsic reinforcements, such as structural prompt isolation Chen et al. [2025b], preference optimization Chen et al. [2025c], and runtime masking He et al. [2026], Zhu et al. [2025a]. However, burdening a single model with both complex reasoning and rigorous security filtering exacerbates hallucinations and drops task success rates below 45% Wang et al.. Relying exclusively on backend semantic validation is fundamentally insufficient to protect contextual integrity at the protocol layer Jing et al. [2025].

Multi Agent Privilege Separation & Our Positioning. Recognizing these bottlenecks, research shifted toward multi agent privilege separation Brumley and Song [2004]. Frameworks like IsolateGPT Wu et al. [2024] and ACE Li et al. [2025b] enforce sandbox isolation, while MAS guardrails Kim et al. [2025], Wang et al. [2025c], Zhu et al. [2025b], de Witt et al. [2025] govern data topologies to prevent privilege escalation. Yet, these sophisticated defenses share a critical flaw: they assume the *a priori* legitimacy of the tool registry. Under the SOP, malicious scripts can trivially spoof tool descriptions; if the registry is compromised natively, tools isolated by the backend are already malicious by design. WebMCP-Phalanx bridges this gap. Intervening at the native browser layer, it issues unforgeable capability handles to establish deterministic ownership. These trust labels are seamlessly propagated to an asymmetric multi agent runtime, coupling native authorization with semantic data routing to systematically eradicate WebMCP's structural vulnerabilities.

3 Attack Method

3.1 Threat Model and Attack Taxonomy

We consider an LLM agent interacting with a web page through the WebMCP standard. The web page registers agent-accessible tools with the browser, while the agent retrieves the registered tool list, invokes selected tools, and receives their returns via the browser. Crucially, the execution environment operates under the browser's Same Origin Policy (SOP), meaning the site's main program, third party advertising SDKs, and analytics scripts share a single origin. Consequently, these components can access the same WebMCP interfaces despite having different levels of trust. This results in a largely flat execution environment in which the current WebMCP specification provides no explicit trust boundary among scripts operating within the same origin.

3.2 Attacker Capabilities and Privilege Scope

In this website, the attacker is a compromised advertising SDK or a tainted analytics script. Operating strictly as a page script sharing the origin of the legitimate site, the adversary possesses the following vector capabilities:

Turing Complete Context Control (Arbitrary JavaScript Execution): As one of the scripts loaded in the page, the attacker can execute arbitrary JavaScript within the page. This includes reading and manipulating the DOM, accessing global objects, and using advanced language level techniques to subvert standard operations.

SOP Granted Registry Access (Same Origin): Because the attacker and the legitimate website share the same origin, the adversary is granted access to the identical tool registry. This enables the attacker to autonomously register malicious tools and attempt to revoke existing tools.

Asynchronous Temporal Exploitation (Unrestricted Timing): The attacker is not constrained by a deterministic loading sequence and can be loaded before or after the legitimate scripts. This temporal advantage allows the attacker to attempt to preempt, overwrite, or race for tool registration.

We establish a strict cryptographic boundary: the attacker holds a capability credential only for tools it registered itself, and it fundamentally cannot present the credential of another registrant. We specifically scope the adversary to same origin page scripts. A browser extension executes in an isolated world under a distinctly different privilege model, to which the same origin reasoning of this paper does not apply; thus, extensions are out of scope.

The attacker's ultimate goal is to compromise the agent's control flow, causing it to execute harmful actions or forcing the original task to fail. This specifically includes inducing the agent to call malicious tools, exfiltrating the user's sensitive data, or hijacking the agent to perform unauthorized operations.

3.3 Taxonomy of WebMCP Exploits

Under the aforementioned adversarial model, the attacks that an attacker can launch are divided into two fundamental categories based on their structural nature. These categories correspond directly to the two complementary lines of defense within the WebMCP-Phalanx framework:

Protocol and State Manipulation: These exploits target vulnerabilities at the specification level. For example, the adversary can manipulate the state of tools, or exploit incomplete clearance during page transitions to make tools and their residual semantics persist.

Semantic Poisoning (Content Level Attacks): These exploits weaponize the linguistic interface of the LLM by hiding instructions to manipulate the agent within a structurally completely valid tool. Specifically, the adversary conceals these instructions within the tool's description, input schema, or post execution return content.

4 Proposed Method

The WebMCP-Phalanx framework is engineered upon a foundational security paradigm: the strict decoupling of origin verification from semantic execution. Within this architecture, the native browser layer acts as a deterministic trust anchor, generating unforgeable trust labels for each tool, while the downstream multi agent runtime dynamically enforces data flow policies based strictly on these labels. This structural decoupling introduces a critical security property: the autonomous agent is completely relieved of the cognitive burden of deducing a tool's trustworthiness. Because the browser layer cryptographically binds and labels the tool before it reaches the LLM, the agent operates on a provably reliable foundation to isolate and neutralize semantic layer threats.

The execution pipeline is divided into two continuous stages. As illustrated in Figure 1, when a webpage script invokes `registerTool`, the underlying Tool Ownership Authority intercepts the call to physically manage state. The tool is then tagged with its origin and continuously evaluated by Telemetry and Lifecycle Oracles. A Data Boundary Authority synthesizes these streams into an immutable trust tier label. Finally, as shown in Figure 1, the tool and its label are dispatched to the Multi Agent Runtime, which dynamically routes the payload to strictly enforce the assigned trust policy.

4.1 Native Cryptographic Ownership Authority

This module constitutes the root of trust for the entire framework. We completely abstract the core state of tool ownership into the browser's native implementation layer, uniformly governing tool registration, enumeration, and unregistration. This structural isolation renders the registry entirely inaccessible to Turing complete third party JavaScript.

Upon a successful registration request, the native execution engine mints an opaque, unforgeable capability handle. To subsequently revoke or overwrite a tool, the calling script must present this exact cryptographic credential. Consequently, a script—including a same origin third party adversary—can only mutate tools it natively registered. This capability based model establishes absolute ownership integrity rather than subjective attribution: it cryptographically proves continuous possession, not identity.

4.2 Browser Side Telemetry and Lifecycle Oracles

Operating directly atop the ownership layer, these native sub components continuously monitor execution state to feed deterministic signals into the Data Boundary Authority.

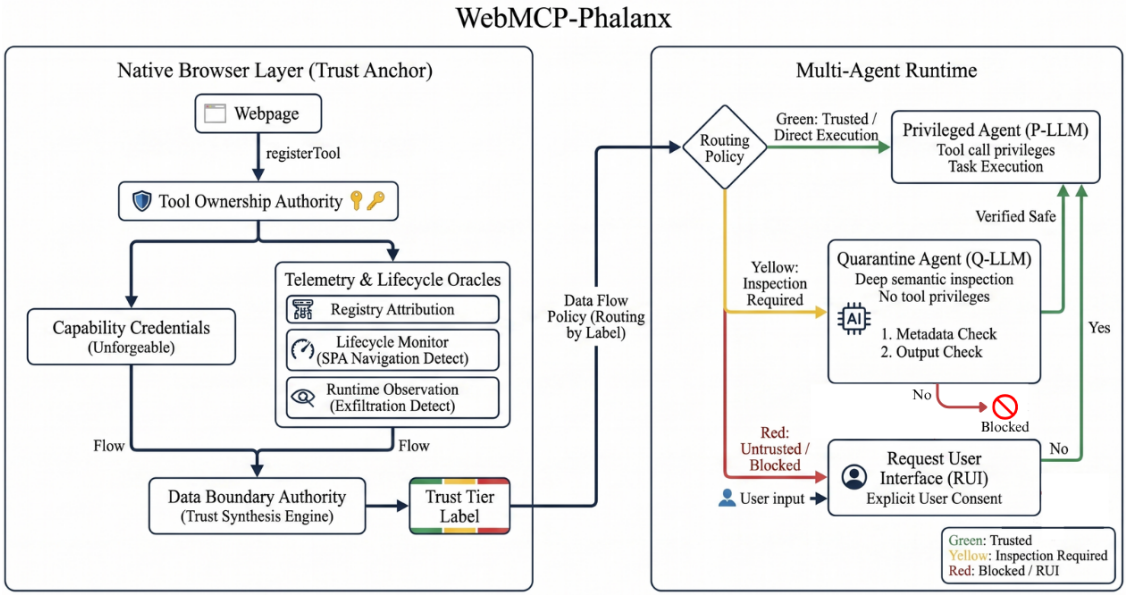

Figure 1: The native browser side trust pipeline. The Tool Ownership Authority acts as the root of trust by issuing unforgeable capability handles, effectively isolating the registry from Turing complete same origin adversaries. The synthesis of telemetry and lifecycle states generates a deterministic Trust Tier Label. Note that while this architecture ensures ownership integrity, first registered name allocation remains a structural limitation (§Limitations).

- Registry Attribution: Upon a tools/list invocation, the native browser retrieves the tool's registrant origin from the isolated core state and appends a spoof proof origin tag. While this definitively identifies the origin, it structurally cannot distinguish between discrete scripts sharing that origin (see Discussion).
- Lifecycle Monitor: At registration, tools explicitly declare their lifecycle scope as either session or navigation. By natively intercepting the History API (pushState, replaceState, popstate), this module accurately detects logical page transitions in Single Page Applications (SPAs) and aggressively purges navigation scoped tools. This mitigates a critical specification gap where tools persist until a hard reload, forcefully aligning the tool's lifecycle with the page as perceived by the user.
- Runtime Observation: During active tool execution, this module asynchronously monitors outbound exfiltration vectors (fetch, sendBeacon, Image.src) and DOM mutations via an isolated MutationObserver. These traces are compared against the tool's statically declared behavior. Upon detecting divergence (e.g., a self reported read only tool initiating a network fetch), the system immediately downgrades the trust label to Red. This recomputed label is injected into the execution output and future registry reads, guaranteeing the agent perceives the violation. This mechanism provides robust detect and contain semantics without requiring heavyweight sandbox isolation.

4.3 Data Boundary Authority (Trust Synthesis Engine)

This module deterministically synthesizes the telemetry signals into a singular Trust Tier Label for every tool. The algorithmic determination integrates two distinct dimensions:

- Origin Dimension (Derived from Ownership & Attribution): Evaluates whether the tool is anchored by a valid capability credential issued by the native layer.
- Content Dimension (Derived from Telemetry & Declarations): Evaluates the declared behavioral intent against the physical execution traces observed by the runtime.

The framework enforces three strict trust tiers:

Green (Trusted): The tool possesses a cryptographically verified origin, and its runtime execution strictly adheres to its declared behavior. Policy: Permitted for direct execution.

Yellow (Requires Attention): The tool possesses a verified origin, but its declared schema involves mutating its local environment. Policy: Subject to lightweight semantic verification.

Red (Untrusted): The tool lacks a legitimate origin, or runtime telemetry has captured behavioral violations. Policy: Quarantined or explicitly blocked.

Crucially, the Origin Dimension is designed to exert absolute veto power over self reported attributes. However, as demonstrated in our evaluation, this veto mechanism cannot trigger against a same origin adversary, because every successfully registered tool inherently secures a valid capability.

4.4 Asymmetric Dual Agent Isolation Runtime

The downstream execution environment operates on an asymmetric multi agent topology comprising two distinct roles: the Privileged Agent (P), which holds explicit tool calling permissions, and the Quarantine Agent (Q), which is strictly isolated from execution privileges.

Trust labels dictate the data flow topology: Green labeled content is routed directly to P for frictionless task execution. Conversely, untrusted or suspicious content is forcefully intercepted and routed into Q's isolated sandbox for semantic evaluation. Consequently, adversarial instructions concealed within suspicious payloads are absorbed entirely by Q, physically preventing the manipulation of P's control flow.

To counter semantic injections that bypass metadata labeling, Q enforces deep content inspection at two critical execution checkpoints:

- Checkpoint 1 (Pre Execution Description Analysis): Q evaluates the tool's static description and input schema during registration, successfully intercepting static prompt injections designed to manipulate agent planning.
- Checkpoint 2 (Post Execution Return Analysis): Because a structurally benign tool may dynamically generate a malicious payload during execution, Q intercepts and sanitizes the returned context before it is permitted to re enter P's operational memory.

4.5 Request User Interface

The Request User Interface (RUI) on the left side of the architecture diagram serves as the final line of defense within the framework. Under extremely sensitive operations or in scenarios defying deterministic classification, the browser issues a secure confirmation request directly to the user via a trusted path.

Consent is only triggered at the Red level; Green/Yellow levels are handled autonomously by the system's dual agent runtime, avoiding constant interruptions and preventing user fatigue. Consent is thus structurally preserved as the "final line of defense" rather than the primary criterion for security judgment.

In summary, the design of WebMCP-Phalanx is guided throughout by a single principle: the separation of trusted origin and trusted usage. The browser side establishes tool labels using an unforgeable, deterministic mechanism, while the multi agent side and data flow policies make semantic layer judgments strictly based on these labels. The former resolves ownership integrity and lifecycle attacks, while the latter targets semantic level injection attacks.

5 Experiments

5.1 Experimental Setup

All experiments were conducted in a real world browser environment, based on the reference implementation of WebMCP (@mcp-b/global 2.3.2, applying the 2026-06-17 draft specification). Attacks were injected via scripts to simulate a third party script attacker. An attack counts as successful when the agent calls the malicious tool and the receiving end receives the exfiltrated data. All agents in the multi agent module use the same language model, identical to the one used in the injection attack baseline we adopt, so that any performance variance stems from the architecture and labeling rather than a more powerful model. We deliberately refrain from blocking detected violations, in order to obtain an upper bound on attack success rate; this is a methodological choice, since the browser side labels and propagates by design while enforcement resides on the agent side.

5.2 Label Driven Injection Defense

We design a four layer ablation to evaluate how the multi agent system uses trust labels to defend against semantic injections the labels themselves cannot detect: L0 (No Defense), L1 (Labels Only), L2 (Rule based Stripping), and L3 (Multi Agent Content Judgment).For tools without a legitimate origin, the labels suffice; for legitimate tools carrying

Injection Vector	L0	L1	L2	L3
Description (Registration)	69/80	71/80	25/80	**0/80**
Returned Content (Execution)	42/80	38/80	2/80	**2/80**

Table 1: Attack success counts for description and returned content injection across the four defense levels (n=80 per cell).

Handling Point	**Mechanism**	**Fails When**
Registration: Description	Label Rules	Always, under a same origin adversary: every successful registrant obtains a capability, so the origin dimension is constant and its veto cannot fire (Table in Appendix).
Registration: Description	Rule based Stripping	The tool name is task relevant. ASR rises from 0% to 70% as name relevance increases (Table 4).
Registration: Description	Multi Agent (Q)	Not observed; 0% across all name relevance tiers (Table 4).
Execution: Return Content	Label Rules	Always: the label is minted at registration, whereas the injected content does not yet exist at that moment.
Execution: Return Content	Rule based Stripping	The agent invokes the tool on its name before the carrier executes (1 of the 2/80 residual; the other is channel consistent).
Execution: Return Content	Multi Agent (Q)	The agent invokes the tool on its name before the carrier executes (2/80).

Table 2: Failure conditions of each mechanism at each handling point.

malicious content, Table 1 reports the attack success rates across the four layers. L1 confirms the labels' blind spot to content injection: with labels alone, 89% of description injections and 48% of return injections still penetrate, close to the undefended baseline. L2 reduces description injection to 25/80 through content agnostic redaction, and L3 closes that channel entirely (0/80); return injection is reduced to a 2/80 residual by both.Task completion (distinct from the attack success counts in Table 1) declines from 66/80 (L0) to 56/80 (L3) on the description channel, but this decline is not a defense cost: a no attack baseline completes 61/80, statistically indistinguishable from L3 ($p = 0.48$), while the higher L0 value stems from the checkout scenario, where the malicious tool's task fitting name scaffolds the agent into a fuller workflow. The scaffold survives description redaction (L2) and is thus carried by the tool name alone, so we anchor all completion comparisons to the no attack baseline.

5.3 Injection Vector: Description vs. Returned Content

Semantic injection can hide in the description at registration or in the returned content after execution, the latter being dynamic and absent at registration time. We added returned content injection and a corresponding post execution handling point. In Table 1, L1 is close to L0 for both vectors, because the labels are minted at registration while the malicious content of return injection does not yet exist; L1 for description injection is slightly higher than L0, since the labels mark the tools as trusted and thereby feed the agent misleading information. Returned content can be entirely replaced with placeholders, so rule based stripping prevents it almost completely; but after stripping, the tool name remains, and if it fits the task the agent still calls it, so only multi agent content evaluation closes the description channel entirely (Table 4).

Table 2 breaks down the failure condition of each mechanism. The 2/80 residuals at L2 and L3 are disjoint, and the matching rates are coincidental. Three of the four runs are name driven: the agent invoked vendor sync in the same turn as, or before, the carrier executed, so no returned content existed to strip or read (Q detects the carrier 40/40). The other two are not detection failures: the exfiltration tool presents a clean description and returns a benign status object, making it structurally invisible to both content checkpoints. What the content check intercepts is the carrier that induces the call, not the induced call itself; closing this residual requires a name layer or call timing gate.

5.4 Cross Page Semantic Residual

Lifecycle monitoring clears residual tools after a page transition, but the content those tools previously generated has already entered the agent's conversation history and continues to influence its later behavior. In the attack scenario, a tool on Page A injects instructions within its returned content to direct a bank transfer; after the agent navigates to Page B the tool is cleared, and we observe whether the residual semantics still drive the transfer. A key finding from Table 3 is that merely clearing the tool only reduces the attack success rate from 80% to 70%, because the injected semantics

Condition	Attack Success Rate	Task Completion Rate
No Defense	80%	95%
No Attack	0%	100%
Lifecycle Monitor Only	70%	80%
Lifecycle Monitor + Our Framework	0%	100%

Table 3: Cross Page Semantic Residual Analysis.

Name Relevance	L2 ASR	L3 ASR
N0 Completely Irrelevant	0%	0%
N1 Generic	0%	0%
N2 Common Step	25%	0%
N3 Task Domain	55%	0%
N4 Essential Step	70%	0%

Table 4: Name Relevance on Defense Performance.

reside in the conversation history, independent of whether the generating tool still exists, a pure semantic residual. Clearing the tool also lowers completion from 19/20 to 16/20, but not because clearing harms the task: in all such runs the agent believes the residual audit claim, halts, and asks the user to confirm a fabricated transfer instead of sending the payment. Our countermeasure extends the lifecycle of trust labels to the content the tools generate, invalidating the labels of any residual returned content in the agent's history when the tool is cleared; this reduces the attack success rate to 0% without harming normal navigation, and simultaneously improves utility over clearing alone, since an attack succeeding during a page transition often distracts the agent from the original task, and removing the interference lets it stay focused and complete.

5.5 Name Engineering & Architectural Trade offs

Table 4 shows that the limitation of rule based stripping against description injection stems from the agent calling a tool prematurely when its name fits the task, before content filtering intervenes. By fixing the injected content and varying only the name relevance, we quantify its impact on L2 and L3. The contrast is clear: the more closely the name fits the task, the weaker rule based stripping becomes, whereas the multi agent system reads the actual content and is unaffected by how the name is packaged. This effect is further modulated by substitutability: the checkout scenario is the most hijacked (20/20 at L0) because its malicious tools share the checkout's function and displace it by substitution, while the driver installation scenario, whose check–download–execute sequence offers no such substitute, is the only one where attacks frequently fail without any defense (10/20 at L0, all failed runs still completing the task). Substitutability, not sequence rigidity alone, governs reach.

6 Discussion

The architectural foundation of WebMCP-Phalanx rests on the strict decoupling of provenance verification from semantic execution: the native browser layer establishes an unforgeable trust label, while the multi agent runtime enforces semantic layer policies based strictly on that anchor. This addresses a fatal structural gap in contemporary multi agent defenses. Frameworks such as Dual-LLM and CaMeL operate under the assumption that the system inherently knows which content is untrusted, yet they fundamentally fail to resolve where that label originates or whether it can be cryptographically forged by an adversary. In a multi party web environment governed by a flat Same Origin Policy, this trusted oracle does not naturally exist. Phalanx physically constructs this oracle at the native browser layer, and our empirical measurements formally delimit the exact reach of a capability based anchor.

6.1 The Cryptographic Reach of Bearer Capabilities.

Ownership integrity is empirically validated and enforced: a third party script holding a valid capability handle is strictly isolated to modifying its own tools. Attempts to manipulate legitimate tools (e.g., checkout) deterministically raise an `InvalidStateError`. Consequently, overwrite and revocation attacks (V4 and V6) drop from 20/20 to 0/20, while V5's legitimate response rate rises from 0/20 to 20/20.

However, we formally demonstrate that subject attribution is structurally beyond the reach of a bearer capability. This is not an artifact of our implementation; the native layer would issue a handle to a malicious `partner.js`

on exactly the same terms. This fundamental gap cannot be repaired at the JavaScript execution layer and must be structurally addressed within the WebMCP specification through two explicit mechanisms. First, Native Script Provenance: JavaScript lacks unforgeable script identity. Because properties like `document.currentScript` and stack introspection are trivially spoofable by a same origin adversary, page layer attribution is inherently insecure and must be supplied by the browser's native compilation pipeline. Second, a Declarative Trust Topology: A third party SDK is loaded by the site's own choice, yet the browser possesses no observable heuristic to distinguish "loaded by" from "trusted by." This operational intent exists exclusively within the site's logic and must be explicitly declared, analogous to a Content Security Policy (CSP). We strongly recommend the WebMCP specification incorporate both primitives; evaluating their native implementations falls outside the scope of this paper.

6.2 Adversarial Confounders in Utility Measurement.

The critical value of cryptographically binding content trust to the origin's lifecycle is most evident in cross page evaluations: once a tool is cleared by the SPA router, its previously generated payload still lingers in the agent's context window. Only our strict lifecycle binding mechanism systematically eradicates this semantic residual.

Furthermore, our cross page and completion analyses expose a critical statistical confounder in agentic evaluation: utility under an adversary is not a neutral metric. An active attack can artificially depress task utility by distracting the agent, or paradoxically inflate it because the malicious tool's task fitting nomenclature acts as an adversarial scaffold, successfully guiding the agent through a more complete workflow ($p = 0.003$). Consequently, measuring task completion against an attacked baseline risks fundamentally misattributing either effect to the defense mechanism itself. To ensure statistical rigor, we mandate anchoring all utility comparisons exclusively to a zero attack baseline.

Ultimately, the two lines of defense in Phalanx reflect the fundamental dichotomy of the WebMCP threat landscape. Specification level protocol flaws demand deterministic, system level eradication independent of the underlying LLM, whereas semantic prompt injections necessitate the deep contextual comprehension provided by an isolated, multi agent evaluation runtime.

7 Conclusion and Future Work

Our reference implementation operates as a JavaScript polyfill, so two security assumptions are emulated rather than natively implemented. The first is API surface integrity, which we emulate by preemptively loading and caching pristine built-in references. A native layer would secure the API surface before any page script executes, which requires the specification to enforce API immutability and tamper-proof native bindings. Because the polyfill operates under a strictly weaker privilege condition than a native implementation, our defense metrics establish a conservative lower bound. The second is cryptographic script provenance, which is structurally absent in vanilla JavaScript. Results involving provenance therefore rest on an architectural existence assumption rather than a cryptographically transferable bound. A capability also acts as a bearer credential and remains vulnerable to the confused deputy problem: a registrant that leaks its handle, or that exposes a function revoking tools on its behalf, forfeits the ownership guarantee.

Two further components are presented as architectural primitives for the specification and are not empirically evaluated: the site-declared trust policy and the out-of-band Request User Interface. Runtime Observation is a post-execution detection mechanism, so the first-order side effects of an initial violation cannot be prevented. Within the observation window of an asynchronous `execute` operation, the monitor may also fail to attribute DOM mutations to a specific tool, leading to attribution collisions.

WebMCP remains in its formative drafting stages. We hope the vulnerabilities and architectural remedies reported here will assist the W3C in embedding security-by-design principles before widespread deployment. Two directions remain open. First, resilience against adaptive adversaries: our primary evaluation measures fixed, known injection techniques, and we additionally modeled a white-box adaptive attacker endowed with the Quarantine Agent's classifier prompt and with end-to-end success feedback. This attacker bypassed 15 of 16 description-layer payloads with a median of one rewrite round, succeeding not by defeating Q's semantic judgment but by relocating the payload onto the task-fitting tool name, an adversarial scaffold that lies outside the bounds of content inspection. Closing this vector requires a name-layer or call-timing gate. Second, deterministic execution sandboxing: Runtime Observation is optimized for detection, containment, and disclosure rather than prevention, and blocking covert behavior during internal tool execution demands sandbox-level isolation at the browser layer.

References

TOCOL SERVERS. Mcp-universe: Benchmarking large language models with real-world model context pro-tocol servers. *origins*, 25:55–128, 2025.

Huihao Jing, Haoran Li, Wenbin Hu, Qi Hu, Xu Heli, Tianshu Chu, Peizhao Hu, and Yangqiu Song. Mcip: Protecting mcp safety via model contextual integrity protocol. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 1177–1194, 2025.

Xilong Wang, John Bloch, Zedian Shao, Yuepeng Hu, Shuyan Zhou, and Neil Zhenqiang Gong. Webinject: Prompt injection attack to web agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 2010–2030, 2025a.

Tri Cao, Bennett Lim, Yue Liu, Yuan Sui, Yuexin Li, Shumin Deng, Lin Lu, Nay Oo, Shuicheng Yan, and Bryan Hooi. Vpi-bench: Visual prompt injection attacks for computer-use agents. *arXiv preprint arXiv:2506.02456*, 2025.

Yulin Chen, Haoran Li, Yuexin Li, Yue Liu, Yangqiu Song, and Bryan Hooi. Topicattack: An indirect prompt injection attack via topic transition. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7338–7356. Association for Computational Linguistics, 2025a.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. {PoisonedRAG}: Knowledge corruption attacks to {Retrieval-Augmented} generation of large language models. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 3827–3844, 2025.

Gelei Deng, Yi Liu, Yuekang Li, Kailong Wang, Ying Zhang, Zefeng Li, Haoyu Wang, Tianwei Zhang, and Yang Liu. Masterkey: Automated jailbreak across multiple large language model chatbots. *arXiv preprint arXiv:2307.08715*, 2023.

Zexuan Zhong, Ziqing Huang, Alexander Wettig, and Danqi Chen. Poisoning retrieval corpora by injecting adversarial passages. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13764–13775, 2023.

Hwan Chang, Yonghyun Jun, and Hwanhee Lee. Chatinject: Abusing chat templates for prompt injection in llm agents. *arXiv preprint arXiv:2509.22830*, 2025.

Reachal Wang, Yuqi Jia, and Neil Zhenqiang Gong. Obliinjection: Order-oblivious prompt injection attack to llm agents with multi-source data. *arXiv preprint arXiv:2512.09321*, 2025b.

Hongyan Chang, Ergute Bao, Xinjian Luo, and Ting Yu. Overcoming the retrieval barrier: Indirect prompt injection in the wild for llm systems. *arXiv preprint arXiv:2601.07072*, 2026.

Jonathan Sneh, Ruomei Yan, Jialin Yu, Philip Torr, Yarin Gal, Sunando Sengupta, Eric Sommerlade, Alasdair Paren, and Adel Bibi. Tooltweak: An attack on tool selection in llm-based agents. *arXiv preprint arXiv:2510.02554*, 2025.

CHE WANG, Jiaming Zhang, Ziqi Zhang, Zijie Wang, Yinggui Wang, Jianbo Gao, Tao Wei, Zhong Chen, and Wei Yang Bryan Lim. Trojantools: Adaptive indirect prompt injection on llm agents via malicious tool-calling.

Zichuan Li, Jian Cui, Xiaojing Liao, and Luyi Xing. Les dissonances: Cross-tool harvesting and polluting in pool-of-tools empowered llm agents. *arXiv preprint arXiv:2504.03111*, 2025a.

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)*, pages 2383–2400, 2025b.

Sizhe Chen, Arman Zharmagambetov, Saeed Mahloujifar, Kamalika Chaudhuri, David Wagner, and Chuan Guo. Secalign: Defending against prompt injection with preference optimization. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 2833–2847, 2025c.

Yu He, Haozhe Zhu, Yiming Li, Shuo Shao, Hongwei Yao, Zhihao Liu, and Zhan Qin. Attriguard: Defeating indirect prompt injection in llm agents via causal attribution of tool invocations. *arXiv preprint arXiv:2603.10749*, 2026.

Kaijie Zhu, Xianjun Yang, Jindong Wang, Wenbo Guo, and William Yang Wang. Melon: Provable defense against indirect prompt injection attacks in ai agents. *arXiv preprint arXiv:2502.05174*, 2025a.

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Shah, Yujia Bao, et al. Mcp-bench: Benchmarking tool-using llm agents with complex real-world tasks via mcp servers, 2025. *URL https://arxiv. org/abs/2508.20453*.

David Brumley and Dawn Song. Privtrans: Automatically partitioning programs for privilege separation. In *USENIX security symposium*, volume 57, 2004.

Yuhao Wu, Franziska Roesner, Tadayoshi Kohno, Ning Zhang, and Umar Iqbal. Isolategpt: An execution isolation architecture for llm-based agentic systems. *arXiv preprint arXiv:2403.04960*, 2024.

Evan Li, Tushin Mallick, Evan Rose, William Robertson, Alina Oprea, and Cristina Nita-Rotaru. Ace: A security architecture for llm-integrated app systems. *arXiv preprint arXiv:2504.20984*, 2025b.

Juhee Kim, Woohyuk Choi, and Byoungyoung Lee. Prompt flow integrity to prevent privilege escalation in llm agents. *arXiv preprint arXiv:2503.15547*, 2025.

Shilong Wang, Guibin Zhang, Miao Yu, Guancheng Wan, Fanci Meng, Chongye Guo, Kun Wang, and Yang Wang. G-safeguard: A topology-guided security lens and treatment on llm-based multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7261–7276, 2025c.

Yifan Zhu, Chao Zhang, Xin Shi, Xueqiao Zhang, Yi Yang, and Yawei Luo. Master: Multi-agent security through exploration of roles and topological structures-a comprehensive framework. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 16895–16921, 2025b.

Christian Schroeder de Witt, Klaudia Krawiecka, Igor Krawczuk, Ben Hagag, William L Anderson, Peter Belcak, Ben Bucknall, Xiaohong Cai, Ayush Chopra, Doron Cohen, et al. Open challenges in multi-agent security: Towards secure systems of interacting ai agents. *arXiv preprint arXiv:2505.02077*, 2025.