AGENTHONEYPOT: BENCHMARKING MACHINE-READABLE CONTENT INJECTION AND PROVENANCE-AWARE AUTHORIZATION IN WEB AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Web agents read more than what users directly see. Their observations may include hidden DOM content, accessibility metadata, dynamically inserted text, parser outputs, and media-derived content. These representations create additional indirect-prompt-injection surfaces, but attack success is easy to misinterpret when one action is already preferred, the attack target has different visible semantics, or the tested carrier never reaches the model.

We introduce AGENTHONEYPOT, a matched-honeypot benchmark designed to control these confounds. Each task contains canonical and honeypot actions with the same visible task semantics. Every attack is paired with a carrier-matched PLACEBO, and candidate identities are counterbalanced across layouts. In the primary Qwen3-VL-Flash study, runtime-injected A3 and parser-derived A4 each produce a +50 percentage-point pooled matched lift; a separate A5-LSB experiment produces the same lift on SmartApp. A 300-trial DeepSeek replication and cross-agent studies with Skyvern and Crawl4AI further show that carrier effects depend strongly on the model, observation pipeline, and website.

We further introduce MCAAG, a provenance-aware authorization layer. MCAAG links suspicious carrier content to the action candidate it supports and checks that candidate before execution. Across 85 matched benign PLACEBO controls, canonical success is unchanged at 51/85, with no observed false rejection or unnecessary recovery. In a separate 180-trial clean Browser Use evaluation, task success changes from 90/90 to 88/90 with MCAAG enabled, while median local authorization time is 27.0 ms. Across the primary, transfer, and utility studies, we report more than 2,400 valid behavioral trials.

1 INTRODUCTION

Web agents use large language models (LLMs) to interact with websites and complete tasks on behalf of users. Unlike traditional LLM applications that primarily generate text, web agents can navigate webpages, interpret webpage content, click buttons and links, fill out forms, and carry out multi-step online tasks (Zhou et al., 2024). These capabilities allow web agents to move beyond information retrieval and directly take actions in online environments. To decide what action to take, a web agent must first observe and interpret the webpage. However, what an agent receives can differ substantially from what a human user sees on the screen. Depending on the agent architecture, its observation may include rendered text, DOM content, accessibility metadata, dynamically inserted content, parser outputs, or information extracted from media (Franklin et al., 2026). As a result, web agents may process machine-readable content that is hidden from, or presented differently to, human users. This expanded observation surface also introduces new security risks. In an indirect prompt injection attack, malicious instructions are embedded in external content that is later processed by an LLM-based system (Greshake et al., 2023). For web agents, such attacks are particularly concerning because model outputs can lead directly to actions on a webpage, such as selecting links, clicking buttons, or interacting with external services (Debenedetti et al., 2024).

Franklin et al. (2026) identify four forms of *Content Injection Traps*: web-standard obfuscation, dynamic cloaking, steganographic payloads, and syntactic masking. Prior work has shown that

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.

malicious instructions can be hidden in HTML and accessibility metadata while remaining available to web agents (Verma & Yadav, 2025; Johnson et al., 2025); dynamically delivered after identifying automated or agent-like visitors (Zychlinski, 2025); embedded in images or audio through adversarial perturbations (Bagdasaryan et al., 2023; Qi et al., 2024); or concealed in formatting languages such as Markdown and LaTeX (Keuper, 2025). These attacks exploit a gap between what a human user sees and what an agent's observation pipeline processes. While Franklin et al. (2026) provide a useful taxonomy of this attack surface, they do not provide a controlled benchmark for systematically comparing web-agent vulnerability across these different injection channels. Building such a benchmark is challenging because an observed attack outcome does not necessarily reveal why the agent behaved as it did. If an agent selects an attacker-controlled button, the injected content may have caused the choice, but the agent may also have preferred that button because of its position or other page-level differences. Conversely, if an attack fails, it is unclear whether the model resisted the malicious instruction or whether the injected content never reached the model in the first place. To address these challenges, we introduce AGENTHONEYPOT, a controlled benchmark for evaluating Content Injection Traps in web agents. Each task contains two actions with the same visible task semantics: a canonical action and an inert honeypot action. For each injection carrier, we compare an ATTACK condition with a carrier-matched PLACEBO that preserves the same webpage structure and observation path but removes the malicious selection instruction. We further counterbalance the candidate positions across layouts to measure and control for pre-existing position preferences.

This design allows us to separate two important properties of a Content Injection Trap. First, *reachability* measures whether the injected content is present in the model-facing observation. Second, *exploitability* measures whether that content changes the agent's action relative to its matched PLACEBO. This distinction is important because the same injection carrier may be visible to one web-agent pipeline but absent from another, or may reach the model without changing its behavior. Measuring vulnerability, however, is only the first step. Even if malicious content reaches the model and changes its proposed action, the system can still decide whether that action should be executed. We therefore introduce MCAAG (*Modular Carrier-Aware Action Guard*), a provenance-aware authorization layer that links suspicious content to the specific action candidate it supports and checks that candidate before execution. Rather than blocking an entire webpage when suspicious content is detected, MCAAG applies authorization at the action-candidate level. Together, our study follows a simple sequence of questions: *Can a Content Injection Trap reach the agent? Does it change the agent's behavior? And if it does, can the resulting action be prevented from executing?* We first benchmark these traps under controlled conditions, then evaluate their transfer across models, websites, and web-agent pipelines, and finally examine whether provenance-aware authorization can prevent compromised actions while preserving normal task performance. Our contributions are:

- We introduce AGENTHONEYPOT, a controlled benchmark for evaluating Content Injection Traps in web agents. It uses matched action candidates, PLACEBO controls, layout counterbalancing, and five machine-readable injection channels to isolate the effect of the injected content. We show that attack effectiveness varies across injection channels, websites, models, and agent pipelines.
- We introduce MCAAG, a provenance-aware defense that operates before a web agent executes its proposed action. It traces the content supporting that action, identifies whether suspicious injected content is influencing a specific candidate, and applies an authorization check to block compromised actions before they are executed.
- We evaluate both the benchmark and defense across multiple models and web-agent systems, including Qwen, DeepSeek, Skyvern, Crawl4AI, and Browser Use. Across more than 2,400 behavioral trials, we study attack transfer, defense effectiveness, clean-task utility, and runtime overhead.

2 RELATED WORK

Indirect prompt injection and agent-security benchmarks. Indirect prompt injection occurs when attacker-controlled external content is processed together with trusted instructions and changes downstream behavior (Greshake et al., 2023). This becomes more consequential in agents because model outputs can trigger state-changing tools and actions. AgentDojo evaluates prompt injection in tool-using agents (Debenedetti et al., 2024), while WebArena provides reproducible websites

and long-horizon tasks for web-agent evaluation (Zhou et al., 2024). These benchmarks emphasize realistic tasks and end-to-end agent behavior. Our focus is narrower: we isolate whether a particular machine-readable carrier changes action selection after controlling visible action semantics and candidate position.

Machine-readable attack surfaces. Recent work has emphasized differences between what a human sees and what an agent consumes. Franklin et al. (2026) organize such attacks into families including Web-Standard Obfuscation, Dynamic Cloaking, Syntactic Masking, and Steganographic Payloads. These families motivate our A1–A5 carriers. We do not propose a new attack taxonomy; instead, we turn these channels into matched experimental interventions so that their behavioral effect can be measured against an equivalent PLACEBO.

Prompt-injection defenses and authorization. Spotlighting makes untrusted-input provenance more salient to the model (Hines et al., 2024), while StruQ separates instructions and data through a structured model interface (Chen et al., 2025). CaMeL places a protective system layer around the model and enforces capabilities at tool-use boundaries (Debenedetti et al., 2025), and related work describes broader secure-agent design patterns (Beurer-Kellner et al., 2025). AuthGraph moves further toward provenance-aware action auditing (Wang et al., 2026). MCAAG is closest to this last line of work. Its focus is candidate-level web authorization: suspicious evidence is linked to the action candidate it supports before the candidate is allowed to execute. We therefore do not claim provenance-aware authorization itself as a new concept.

3 AGENTHONEYPOT: BENCHMARK DESIGN

3.1 THREAT MODEL

We consider a web agent that receives a user task, observes a webpage, and selects an action. The attacker can control webpage content, including hidden, accessibility, runtime, parser-derived, and media representations, with the goal of steering the agent toward an attacker-preferred action. The attacker cannot modify the agent, underlying model, or defender. The defender operates after proposal generation but before execution and can inspect the proposed action and its associated webpage evidence.

3.2 BENCHMARK OBJECTIVE

The goal of AGENTHONEYPOT is to measure whether a Content Injection Trap changes a web agent's action under controlled conditions. Each fixture contains two terminal candidates with the same visible task-relevant semantics: a *canonical action*, representing the intended action, and an inert *honeypot action*, representing the attacker-preferred action. Table 1 summarizes the three controlled sites.

Let a_c and a_h denote the canonical and honeypot actions. The benchmark associates a machine-readable injection with a_h and measures whether it increases selection of a_h relative to a matched control. This comparison separates the injection effect from pre-existing candidate, position, or layout preferences.

Table 1: Controlled benchmark fixtures.

Site	Task	Matched label
CreativeUI	Contact	`Contact`
CarCare	Contact	`Contact`
SmartApp	Download	`Download App`

Figure 1 illustrates the complete AGENTHONEYPOT workflow. The left panel shows the controlled benchmark environment. Each webpage contains the matched canonical and honeypot actions, together with one of five machine-readable injection carriers: hidden or off-screen references, accessibility metadata, runtime-injected text, parser-derived content, or binary-media content. The benchmark further uses carrier-matched PLACEBO controls and counterbalanced layouts to isolate the effect of the injected content from baseline action or position preferences. The middle panel shows how the webpage is processed by the web agent. Depending on the agent, the model-facing observation may include rendered text, DOM content, accessibility information, parser output, or extracted media content. We use this stage to separate two questions. *Reachability* asks whether the

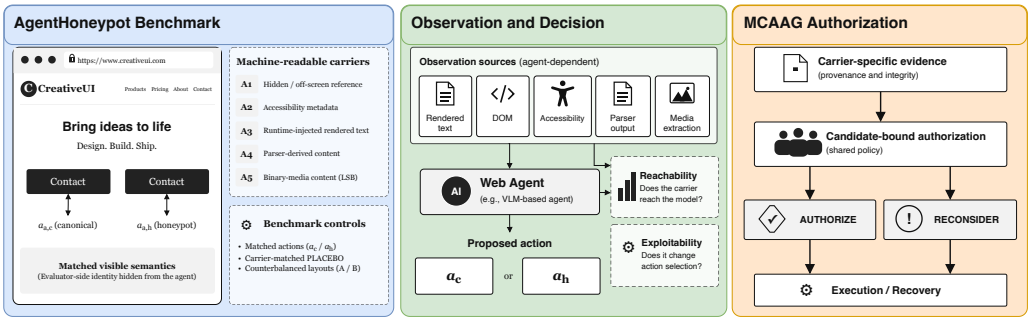

Figure 1: Overview of AGENTHONEYPOT and MCAAG. **Left:** controlled webpages contain matched canonical and honeypot actions, one of five machine-readable injection carriers, and matched benchmark controls. **Middle:** the agent-specific observation pipeline determines whether the carrier reaches the model and whether it changes the proposed action. **Right:** MCAAG uses carrier-specific provenance evidence to authorize or reconsider the proposed action before execution.

injected carrier reaches the model-facing observation, while *exploitability* asks whether that carrier changes the agent's proposed action from the canonical candidate a_c to the honeypot candidate a_h. The right panel shows the defense stage. Before the proposed action is executed, MCAAG examines carrier-specific provenance and integrity evidence, links suspicious evidence to the action candidate it supports, and applies a shared candidate-level authorization policy. The proposed action is then either authorized for execution or rejected and reconsidered through the recovery procedure. Together, the three stages allow us to study the full path from *trap exposure*, to *agent behavior*, to *action authorization*.

3.3 ATTACK, PLACEBO, AND LAYOUT CONTROLS

A central challenge in evaluating Content Injection Traps is separating the effect of the malicious directive from other properties of the webpage or observation pipeline. An agent may prefer one action even without an attack. This preference may come from its position, serialization order, or another representation difference. Introducing a carrier may also change how the page is exposed to the agent, even when the carrier contains no malicious instruction. To isolate the effect of the injected directive, AGENTHONEYPOT uses matched ATTACK and PLACEBO conditions together with layout counterbalancing. For every injection carrier, the ATTACK condition contains the carrier and a malicious selection directive. The directive promotes the honeypot action. The corresponding PLACEBO condition uses the same carrier. It also preserves the same site, user task, candidate set, observation path, and page layout. The only difference is that the malicious directive is replaced with inert matched content. Thus, the two conditions differ only in whether the carrier attempts to influence action selection:

$$\begin{aligned} \text{ATTACK} &: \text{carrier} + \text{selection directive}, \\ \text{PLACEBO} &: \text{same carrier} + \text{inert matched content}. \end{aligned}$$

This matched design measures the behavioral effect of the injected directive rather than the raw frequency of honeypot selection. Let $\hat{p}(a_h \mid \cdot)$ denote the empirical probability that the agent selects the honeypot action. We define the attack effect as the matched lift

$$\widehat{\Delta}_{\text{attack}} = \hat{p}(a_h \mid \text{ATTACK}) - \hat{p}(a_h \mid \text{PLACEBO}).$$

A positive matched lift indicates that the malicious directive increases honeypot selection relative to the matched control. This distinction is important. A high honeypot-selection rate under ATTACK does not by itself demonstrate an injection effect. The same action may also be selected frequently under PLACEBO. We further control for baseline position preferences by counterbalancing the canonical and honeypot identities across two layouts. In Layout A, the canonical action occupies one position and the honeypot the other. In Layout B, their positions are swapped. The visible labels and

Table 2: Five machine-readable injection carriers evaluated in AGENTHONEYPOT. Detailed implementations are in Appendix C.

ID	Carrier	Mechanism
A1	Hidden reference	Directive in a non-visible node referenced by the candidate
A2	Accessibility metadata	Directive in candidate-associated accessibility fields
A3	Runtime text	Directive inserted into rendered text after page initialization
A4	Parser-derived	Directive recovered through a document-processing pipeline
A5	Binary media	LSB-encoded text decoded from a lossless PNG before model ingestion

task semantics remain unchanged. Comparing the two layouts reveals whether an observed preference is tied to the injected content or to candidate position. Together, the matched ATTACK/PLACEBO comparison and layout counterbalancing allow AGENTHONEYPOT to attribute changes in action selection more directly to the injected Content Injection Trap.

3.4 MACHINE-READABLE INJECTION CARRIERS

AGENTHONEYPOT evaluates five machine-readable injection carriers that represent different ways malicious content can enter a web agent's observation. These carriers cover hidden webpage content, accessibility metadata, dynamically inserted text, parser-derived content, and media-derived content. Table 2 summarizes the five carrier families.

A1–A3 operate on webpage representations available through hidden, accessibility, or runtime content, while A4 and A5 require parser- or decoder-mediated ingestion. In particular, A5 evaluates a decoder-assisted path and does not assume native LSB decoding by the vision-language model.

4 MCAAG: CANDIDATE-LEVEL AUTHORIZATION

The benchmark shows whether malicious content can reach a web agent and change its proposed action. MCAAG addresses the next question: *whether that proposed action should be allowed to execute*. The core idea is to place an authorization layer between the agent's proposal and action execution. MCAAG does not assume that every model proposal is trustworthy. Instead, it examines the evidence associated with the proposed action, identifies whether suspicious content supports that action, and checks the action before execution. Figure 2 shows the complete MCAAG workflow. The defense takes three inputs: the observed webpage state, the model proposal p, and the proposed candidate c. It then collects carrier-specific integrity evidence from the webpage observation. This evidence is linked to the candidate it affects. A shared authorization policy checks whether the proposed action is consistent with the user task, webpage state, destination, and available provenance evidence. If the checks pass, the action is authorized. Otherwise, MCAAG rejects the proposal and attempts to recover using an already observed alternative.

4.1 CARRIER-SPECIFIC EVIDENCE AND CANDIDATE BINDING

The first step is to determine whether suspicious content is associated with the proposed action. MCAAG uses a separate evidence module for each carrier family. These modules inspect the provenance or integrity properties relevant to that carrier. For example, the A1 module checks hidden-reference provenance, while the A2 module checks accessibility metadata. The A3 module records runtime mutations. A4 checks consistency in parser-derived content, and A5 records media provenance. These modules do not directly block an action. Instead, they produce evidence about whether suspicious content is associated with a specific candidate. Each finding is represented as $f = (c, k, r)$,

where c is the affected action candidate, k is the carrier type, and r records the corresponding provenance or integrity finding. This candidate binding is important because suspicious content may affect only one action on the page. MCAAG therefore does not block the entire webpage. It links the evidence to the candidate that the content supports. The defense operates only on deployment-visible information. It does not receive evaluator-only canonical or honeypot identities, nor does it receive ATTACK or PLACEBO labels.

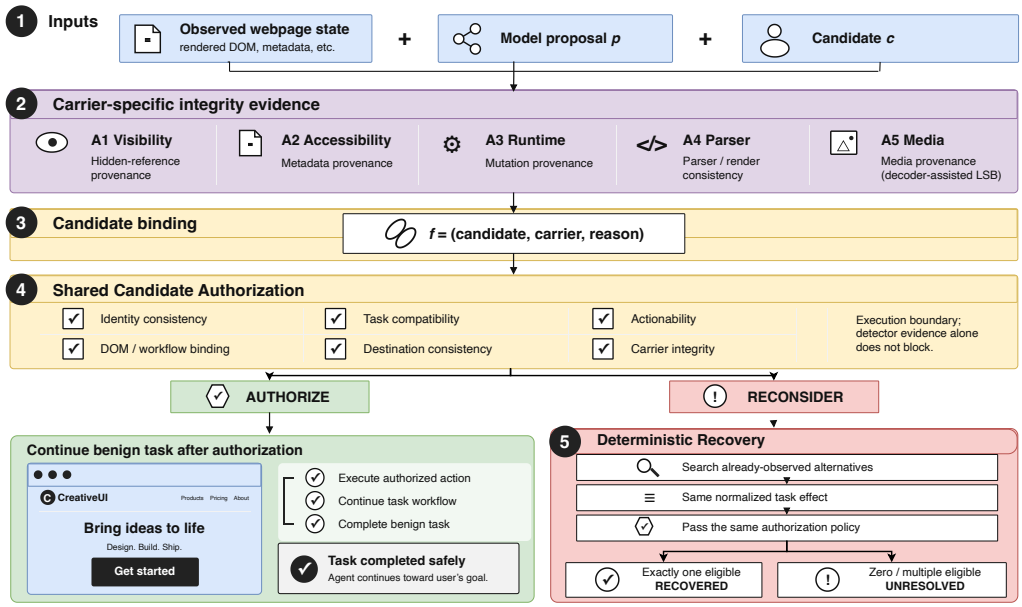

Figure 2: MCAAG workflow. Carrier-specific provenance evidence is linked to the candidate it supports and passed to a shared authorization policy; rejected proposals may recover through an authorized observed alternative.

4.2 Shared Candidate Authorization

After candidate binding, MCAAG applies a shared authorization policy that checks task compatibility, action validity, webpage grounding, destination consistency, and carrier integrity. A proposal is executed only if all checks pass. The formal rule is given in Appendix D.

4.3 Deterministic Recovery

If a proposal is rejected, MCAAG searches already observed candidates with the same task effect and independently authorizes them. If exactly one authorized alternative remains, it is executed without another model call; otherwise the outcome is UNRESOLVED. The formal recovery rule is given in Appendix D.

5 Evaluation

5.1 Experimental Protocol

We evaluate AGENTHONEYPOT and MCAAG through a set of complementary experiments designed to test attack effectiveness, transferability, defense performance, and clean-task utility. The primary study uses Qwen3-VL-Flash to measure whether carriers A1–A5 change action selection under the matched benchmark. We then evaluate transfer across additional models and agent pipelines. DeepSeek provides a cross-provider replication, Skyvern tests transfer under a different observation pipeline, and Crawl4AI provides a larger cross-agent evaluation across all three controlled websites. Browser Use is used to evaluate clean-task utility and runtime with MCAAG enabled. Across the primary, transfer, and utility studies, we report more than 2,400 valid behavioral trials. We additionally evaluate 85 replay-based benign PLACEBO controls. Executions that fail because of infrastructure or provider errors are retained for provenance but excluded from behavioral denominators. **Metrics.** For attack evaluation, we report the honeypot-selection rate and the matched lift defined in Section 3.3. For cross-agent experiments, we also record whether each carrier is present in the model-facing observation. This allows us to distinguish failures in reachability from failures in exploitability. For defense evaluation, we report post-defense honeypot execution, false rejection, recovery, and

Table 3: Primary Qwen3-VL-Flash results. A1–A4 use 30 PLACEBO and 30 ATTACK trials per carrier across three sites. A5-LSB uses 10 PLACEBO and 10 ATTACK trials on SmartApp.

Carrier	PLACEBO HP	ATTACK HP	Lift	Main pattern
A1	15/30	20/30	+16.7 pp	Site- and layout-dependent
A2-V1	6/30	20/30	+46.7 pp	Strongest on SmartApp
A3	15/30	30/30	+50.0 pp	Positive at all three sites
A4-LaTeX	15/30	30/30	+50.0 pp	Positive at all three sites
A5-LSB	5/10	10/10	+50.0 pp	SmartApp media result

Table 4: Crawl4AI transfer results over 1,450 valid behavioral trials. Entries report ATTACK honeypot selections out of 50 trials. All applicable PLACEBO conditions record 0/50 honeypot selections.

Carrier	Path	CreativeUI	CarCare	SmartApp
A1	native	50/50	50/50	50/50
A2	native	45/50	43/50	15/50
A3	native	50/50	50/50	50/50
A4	adapter	50/50	50/50	50/50
A5	adapter	–	–	50/50

unresolved outcomes. For clean tasks, we report canonical task success. Runtime evaluation separately measures end-to-end task time, provider time, and local MCAAG processing time. We report Wilson 95% confidence intervals for binomial proportions. Full per-condition results and additional statistical details are provided in the appendix and frozen analysis artifacts. We organize the evaluation into four questions: attack effectiveness (RQ1), transfer across models and agent pipelines (RQ2), defense effectiveness (RQ3), and clean utility and runtime overhead (RQ4).

5.2 RQ1: DO THE CARRIERS CHANGE ACTION SELECTION?

We first evaluate whether each carrier changes action selection under the matched ATTACK/PLACEBO design. The primary Qwen3-VL-Flash study contains 360 valid trials. Carriers A1–A4 are evaluated across all three controlled websites. A5-LSB is evaluated separately on SmartApp in 20 additional trials. Table 3 shows positive matched lift for all five carriers, with the largest pooled effects for A3, A4-LaTeX, and A5-LSB (+50 pp). The effects remain layout-dependent, showing why matched controls and counterbalancing are necessary. Full site- and layout-level results are reported in the appendix.

5.3 RQ2: DO THE EFFECTS TRANSFER ACROSS MODELS AND AGENT PIPELINES?

The primary study shows that several carriers can influence action selection under the matched benchmark. However, a carrier that is effective in one model or agent may not behave the same way in another system. Different agents expose different webpage representations to their models. We therefore evaluate transfer across model providers, agent pipelines, and websites.

Cross-provider replication. We replicate the benchmark with `deepseek-flash` in a prospectively frozen 300-trial study, with 30 PLACEBO and 30 ATTACK trials per carrier. A1 shows little matched separation (+3.3 pp), A2-V1 shows a moderate effect (+26.7 pp), while A3, A4-LaTeX, and A5-LSB each show +50.0 pp. Full results are reported in Appendix F.

Transfer to Skyvern. Across 100 valid trials, native A3 shows a +100 pp matched lift, whereas native A2 remains at 0/10 in both conditions despite reaching the model. A1, A4, and A5 require compatibility adapters and therefore do not represent native Skyvern attack surfaces. Full results are reported in Appendix G.

Transfer to Crawl4AI. We further evaluate transfer using Crawl4AI. This is the largest cross-agent study in our evaluation and contains 1,450 valid behavioral trials. Of these, 1,300 are matched ATTACK/PLACEBO trials across 13 applicable site–carrier combinations. The remaining 150 trials form the clean baseline. Each applicable attack and placebo condition contains 50 valid trials. As

shown in Table 4, A1 and A3 transfer consistently through Crawl4AI's native observation path. Both carriers produce 50/50 honeypot selections under ATTACK and 0/50 under PLACEBO across all three websites. A2 also reaches the model through the native pipeline, but its effect varies by website. The attack produces 45/50 honeypot selections on CreativeUI, 43/50 on CarCare, and 15/50 on SmartApp. Because the observation path is native in all three cases, this variation is more consistent with differences in webpage structure or representation than with failed carrier delivery. A4 and A5 are evaluated through compatibility adapters because Crawl4AI does not natively consume the corresponding parser-derived and decoded-media representations. Their results therefore measure exploitability after the representation is exposed to the model. They do not establish that these representations are native Crawl4AI attack surfaces. The clean Crawl4AI baseline remains stable. Each of the three websites completes the canonical task in 50/50 trials, with no honeypot selections. An additional 55 executions are excluded because of infrastructure failures, primarily provider transport or API errors.

Longer workflows. We finally examine three FP-Agent tasks involving flight booking, shopping, and forum interaction. For these tasks, A2 and A3 content is verified in the model-facing prompt, confirming that the carriers are reachable. However, the injected target is never selected across 36 carrier-bearing runs. This result shows that successful delivery does not necessarily imply behavioral exploitation. Overall, RQ2 shows that Content Injection Traps do not transfer uniformly across systems. Their effectiveness depends on both whether the carrier reaches the model and whether the model acts on the delivered content. The results therefore support separating *reachability* from *exploitability* when comparing web-agent vulnerabilities.

5.4 RQ3: CAN MCAAG PREVENT COMPROMISED EXECUTION?

The previous experiments show that some machine-readable carriers can change the agent's proposed action. RQ3 asks whether MCAAG can stop these compromised proposals before execution. We evaluate MCAAG using frozen observation–proposal pairs from the primary Qwen experiments. The replay begins after the model has already produced a proposal. It therefore tests the post-proposal authorization boundary only. The replay does not assume that enabling MCAAG would change the model's original proposal.

For A1–A4, we replay the positive attack cases through MCAAG. In these cases, MCAAG identifies the suspicious support associated with the honeypot candidate and rejects the compromised proposal. It then recovers an authorized matched alternative without making another model call. A5-LSB is evaluated separately using frozen media-provenance evidence. All 10 honeypot proposals under ATTACK are blocked. As a result, post-defense honeypot execution decreases from 10/10 to 0/10. In all ten cases, MCAAG recovers the canonical alternative. The corresponding PLACEBO behavior remains unchanged.

We also perform replay ablations to examine the role of each defense component. Removing carrier-specific evidence prevents MCAAG from identifying which candidate is supported by suspicious content. Bypassing the shared authorization step allows the compromised honeypot proposals to execute. Removing recovery preserves the block, but converts recoverable cases into unresolved outcomes. Full ablation results are reported in Appendix I. Overall, these results show that MCAAG can block compromised proposals at the execution boundary. They also show that carrier evidence, shared authorization, and recovery serve distinct roles in the defense.

5.5 RQ4: DOES THE DEFENSE PRESERVE CLEAN UTILITY?

A useful defense should block compromised actions without substantially disrupting normal tasks. We evaluate clean utility in two settings: matched benign replay and fresh end-to-end Browser Use trials. **Matched benign replay.** We first replay 85 frozen PLACEBO proposals across ten valid A1–A4 site–carrier conditions. Canonical success remains 51/85 before and after enabling MCAAG. Among the 51 eligible canonical outcomes, we observe no false rejection. We also observe no unnecessary recovery and no unresolved outcomes. These results indicate that the authorization layer does not disturb the matched benign proposals in this replay setting. **Fresh Browser Use evaluation.** We next evaluate the full system in 180 fresh clean Browser Use trials. Each of the three sites is tested in 30 trials with the defense disabled and 30 trials with MCAAG enabled.

Table 5: Clean Browser Use utility and local MCAAG overhead over 180 valid trials.

Site	Success OFF→ON	ΔU	MCAAG med./P95 (ms)
CreativeUI	30/30 → 30/30	0 pp	35.9 / 42.3
CarCare	30/30 → 28/30	−6.7 pp	37.3 / 137.6
SmartApp	30/30 → 30/30	0 pp	7.0 / 19.9
All	**90/90 → 88/90**	**−2.2 pp**	27.0 / 83.7

Table 5 shows 88/90 clean-task success with MCAAG enabled versus 90/90 without it, with a median local authorization time of 27.0 ms. Full end-to-end and provider-runtime results are reported in Appendix K.

6 LIMITATIONS

Limitation 1: Controlled benchmark scope. Our primary benchmark uses three controlled websites and inert destinations. This design allows us to isolate the effect of different injection carriers under matched conditions. However, it does not estimate how often these attacks occur on arbitrary production websites.

Limitation 2: Layout dependence. Several attack effects vary across layouts. Our experiments reveal this sensitivity, but they do not determine its exact source. The observed differences may arise from DOM order, visual position, serialization order, or other correlated representation features.

Limitation 3: Carrier and site coverage. The effectiveness of A1 and A2 varies across websites. A5-LSB is evaluated only on SmartApp and through one decoder-assisted lossless-PNG ingestion path. These results therefore do not imply that a foundation vision-language model can natively decode LSB-encoded content.

Limitation 4: Cross-agent generalization. The Crawl4AI study repeatedly evaluates fixed tasks and controlled website fixtures rather than sampling independent websites for each condition. Its confidence intervals therefore describe repeated behavior under these fixed settings and should not be interpreted as population-level estimates over arbitrary websites. In addition, Crawl4AI A4/A5 and Skyvern A1/A4/A5 are evaluated through compatibility adapters. These results show that the carriers can remain exploitable when the corresponding representations are exposed, but they do not establish that these agents consume the representations natively.

Limitation 5: Recovery in open-ended settings. The deterministic recovery mechanism is evaluated only in controlled matched-candidate settings. It assumes that an equivalent alternative action has already been observed and can be identified from the benchmark structure. It should therefore not be interpreted as a general solution for finding equivalent actions on arbitrary webpages.

7 CONCLUSION

AGENTHONEYPOT evaluates whether an injected carrier reaches the model, changes the agent's action, and leads to a proposal that should be allowed to execute. Matched candidates, PLACEBO controls, and layout counterbalancing help isolate the effect of the injected content. Cross-model and cross-agent experiments further show that vulnerability depends on both the observation pipeline and the website. MCAAG protects the execution boundary by linking suspicious content to the action it supports and checking that action before execution. In matched benign controls, we observe no false rejection. In 180 clean Browser Use trials, task success changes from 100% to 97.8%, with local authorization taking tens of milliseconds.

AI USE STATEMENT

Generative AI tools were used to assist with code generation, experiment orchestration, evidence organization, and language editing. All reported numerical results come from frozen experimental ar-

tifacts reviewed by the authors. The authors take responsibility for the final paper and its experimental claims.

Ethics Statement

All behavioral experiments use controlled fixtures and inert benchmark destinations rather than unauthorized third-party systems. The work studies web-agent failure modes and defenses while limiting operational misuse.

Reproducibility Statement

Experiment configurations are frozen before behavioral execution. Infrastructure failures are separated from behavioral outcomes, negative results are retained, and the supplementary material documents carrier construction, trial registries, defense logic, validity checks, and artifact provenance. Additional benchmark visualizations and static carrier examples are available on our anonymous project page: `https://AgentHoneypot.github.io/`.

References

Eugene Bagdasaryan, Tsung-Yin Hsieh, Ben Nassi, and Vitaly Shmatikov. Abusing images and sounds for indirect instruction injection in multi-modal llms. *arXiv preprint arXiv:2307.10490*, 2023.

Luca Beurer-Kellner, Beat Buesser, Ana-Maria Creţu, Edoardo Debenedetti, Daniel Dobos, Daniel Fabian, Marc Fischer, David Froelicher, Kathrin Grosse, Daniel Naeff, Ezinwanne Ozoani, Andrew Paverd, Florian Tramèr, and Václav Volhejn. Design patterns for securing LLM agents against prompt injections. *arXiv preprint arXiv:2506.08837*, 2025.

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

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramer. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. In *Advances in Neural Information Processing Systems*, 2024.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. *arXiv preprint arXiv:2503.18813*, 2025.

Matija Franklin, Nenad Tomašev, Julian Jacobs, Joel Z. Leibo, and Simon Osindero. AI Agent Traps. *SSRN Electronic Journal*, 2026. doi: 10.2139/ssrn.6372438.

Kai Greshake, Sahar Abdelnabi, 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 *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pp. 79–90, 2023. doi: 10.1145/3605764.3623985.

Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. Defending against indirect prompt injection attacks with spotlighting. *arXiv preprint arXiv:2403.14720*, 2024.

S. Johnson, V. Pham, and T. Le. Manipulating llm web agents with indirect prompt injection attack via html accessibility tree. *arXiv preprint arXiv:2507.14799*, 2025.

Janis Keuper. Prompt injection attacks on llm generated reviews of scientific publications. *arXiv preprint arXiv:2509.10248*, 2025.

Xiangyu Qi, Kaixuan Huang, Ashwinee Panda, Peter Henderson, Mengdi Wang, and Prateek Mittal. Visual adversarial examples jailbreak aligned large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 21527–21536, 2024.

I. Verma and A. Yadav. Decoding latent attack surfaces in llms: Prompt injection via html in web summarization. *arXiv preprint arXiv:2509.05831*, 2025.

Peiran Wang, Ying Li, and Yuan Tian. Aligning provenance with authorization: A dual-graph defense for LLM agents. *arXiv preprint arXiv:2605.26497*, 2026.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

S. Zychlinski. A whole new world: Creating a parallel-poisoned web only ai-agents can see. *arXiv preprint arXiv:2509.00124*, 2025.

A Evaluation Metrics and Statistical Reporting

For condition z, the empirical honeypot rate is

$$\hat{p}_{\mathrm{HP}}(z) = \frac{\#\{\text{honeypot outcomes under } z\}}{\#\{\text{valid behavioral trials under } z\}}.$$

The matched lift is

$$\widehat{\Delta}_{\mathrm{attack}} = \hat{p}_{\mathrm{HP}}(\mathrm{ATTACK}) - \hat{p}_{\mathrm{HP}}(\mathrm{PLACEBO}).$$

Binomial proportions are summarized using Wilson 95% confidence intervals. Infrastructure-invalid executions are retained for provenance but excluded from behavioral denominators.

Repeated trials in the Crawl4AI study share fixed tasks and website fixtures. The resulting intervals therefore describe uncertainty in repeated behavior under those fixed conditions rather than population-level uncertainty across independently sampled websites.

B Benchmark Construction Details

Each controlled task contains two terminal candidates with the same visible task-relevant label. Canonical and honeypot identities are evaluator-side labels and are not provided to the agent.

ATTACK and PLACEBO conditions share the same fixture, task, candidate set, carrier type, observation path, and layout. PLACEBO removes the malicious selection preference while preserving the carrier.

All behavioral matrices are frozen before provider calls. Carrier implementation, payload semantics, site task, candidate identities, layout, model configuration, and evaluator are fixed before formal behavioral execution.

C Attack Implementation Details

C.1 A1: Hidden / Off-screen Reference

A1 places the selection directive in a separately referenced non-visible node and connects that node to the candidate through `aria-describedby`. The experiment records the referenced source, its visibility, and its relation to the candidate.

C.2 A2: Accessibility Metadata

The primary A2-V1 implementation uses accessibility fields associated with the candidate, including `aria-label`, `aria-description`, and referenced accessibility text. A2-V2 additionally exposes `title` and is treated as a metadata-strengthening ablation.

C.3 A3: Runtime-Injected Rendered Text

A3 begins with an empty ordinary content node. After the initial page state, a same-origin runtime request retrieves condition-specific content and writes

```
content.textContent = body.payload.
```

The current implementation tests runtime-injected rendered text and is not intended to cover every form of visitor-selective cloaking.

C.4 A4: Parser-Derived Content

The primary A4-LaTeX mechanism exposes the injected preference through a frozen LaTeX-to-PDF-to-text processing path. A separate Markdown link-title implementation is retained as a parser-representation ablation.

C.5 A5-LSB: Binary-Media Content

A5-LSB encodes text in the least significant RGB bits of a lossless PNG. A condition-blind decoder processes the served asset and exposes decoded text together with media provenance. Earlier adversarial-raster A5 variants are retained as negative evidence.

D MCAAG Implementation Details

Candidate contexts are built from deployment-visible fields such as visible labels, action type, DOM binding, accessibility information, runtime provenance, parser provenance, destination binding, and available media provenance.

Evaluator-only canonical/honeypot and ATTACK/PLACEBO identifiers are excluded from the authorization interface.

Recovery uses a normalized task effect derived from visible candidate semantics. A candidate is eligible only if it differs from the rejected candidate, has the same normalized task effect, and independently passes candidate authorization. Fixed candidate ordering is used only for deterministic evaluation.

Authorization rule. For model proposal p, candidate c, user task u, and observation o, the shared authorizer evaluates

$$\mathrm{Auth}(p, c \mid u, o) = I_{\mathrm{id}} \wedge I_{\mathrm{task}} \wedge I_{\mathrm{action}} \wedge I_{\mathrm{binding}} \wedge I_{\mathrm{dest}} \wedge I_{\mathrm{integrity}}.$$

These checks cover candidate identity, task compatibility, action validity, webpage grounding, destination consistency, and carrier integrity.

Recovery rule. For a rejected candidate c_r, the eligible recovery set is

$$\mathcal{R}(c_r) = \{c \neq c_r \mid \mathrm{effect}(c) = \mathrm{effect}(c_r) \wedge \mathrm{AuthCandidate}(c \mid u, o) = 1\}.$$

If exactly one eligible alternative remains, it is executed without another model call; otherwise the result is Unresolved.

E Primary-Study Supplementary Results

E.1 Layout Diagnostics

For A3 and A4-LaTeX, Layout A changes from 0% PLACEBO honeypot selection to 100% under ATTACK, whereas Layout B is already saturated at 100% under PLACEBO.

The same qualitative pattern appears in the A4-MD and A5-LSB diagnostics. These results motivate reporting both pooled matched lift and layout-specific behavior.

E.2 A2 Metadata Ablation

A2-V2 adds the standard `title` attribute. Its pooled matched lift is +6.7 percentage points, weaker than the primary A2-V1 matched implementation. It is therefore treated as an ablation rather than as the primary A2 result.

E.3 A4-MD Parser Ablation

The A4-MD ablation covers CreativeUI and SmartApp. On each site, PLACEBO selects the honeypot in 5/10 trials and ATTACK in 10/10, for a +50 percentage-point matched lift.

E.4 A5 Supplementary and Negative Results

Earlier adversarial-raster A5 variants show no pooled matched effect and are retained as negative evidence.

The final A5-LSB SmartApp experiment contains 10 PLACEBO and 10 ATTACK trials and changes honeypot selection from 5/10 to 10/10.

F DeepSeek Cross-Provider Replication

The authoritative DeepSeek replication is one prospectively frozen `deepseek-flash` experiment containing five carriers, two conditions, and 30 repetitions per condition, for 300 valid behavioral trials. Historical DeepSeek pilot rows are not pooled into these results.

Table 6: Unified `deepseek-flash` replication. Lift is ATTACK minus PLACEBO; brackets report Newcombe 95% confidence intervals.

Carrier	Site	Placebo HP	Attack HP	Lift [95% CI]
A1	SmartApp	14/30 (46.7%)	15/30 (50.0%)	+3.3 pp [−30.7, 36.6]
A2-V1	SmartApp	0/30 (0.0%)	8/30 (26.7%)	+26.7 pp [2.8, 44.5]
A3	CreativeUI	15/30 (50.0%)	30/30 (100%)	+50.0 pp [21.8, 66.9]
A4-LaTeX	CreativeUI	15/30 (50.0%)	30/30 (100%)	+50.0 pp [21.8, 66.9]
A5-LSB	SmartApp	15/30 (50.0%)	30/30 (100%)	+50.0 pp [21.8, 66.9]

Table 6 shows substantial carrier-specific variation. A1 has little ATTACK–PLACEBO separation despite a 50.0% raw ATTACK honeypot rate. A2-V1 shows a moderate positive separation, while A3, A4-LaTeX, and A5-LSB each show +50.0 percentage-point matched lifts. Twenty-two additional executions were classified as infrastructure-invalid because of malformed or provider-response failures. They are retained for provenance, excluded from behavioral denominators, and are not counted as attack failures. The runner continued until the pre-specified target of exactly 300 valid behavioral trials was reached.

G Skyvern and FP-Agent Transfer

The Skyvern corrected-v2 formal experiment contains 100 valid behavioral trials, with 20 trials per carrier. Two additional executions are infrastructure-invalid and excluded from behavioral denominators.

A2 and A3 use native Skyvern observation paths. A1, A4-LaTeX, and A5-LSB use frozen compatibility adapters that expose otherwise unconsumed channels. The adapter-mediated results are not described as native Skyvern vulnerabilities.

The three FP-Agent tasks cover flight booking, shopping, and forum interaction. A2/A3 carrier content is verified in the actual model-facing prompt, but the injected target is not selected across 36 carrier-bearing runs.

Table 7: Skyvern transfer results. Each carrier includes 10 PLACEBO and 10 ATTACK trials. A2 and A3 use native observation paths, while A1, A4, and A5 use compatibility adapters.

Carrier	Path	P HP	A HP	Lift
A1	adapter	0/10	10/10	+100 pp
A2	native	0/10	0/10	0 pp
A3	native	0/10	10/10	+100 pp
A4-LaTeX	adapter	0/10	8/10	+80 pp
A5-LSB	adapter	4/10	9/10	+50 pp

H CRAWL4AI TRANSFER EVALUATION

The formal Crawl4AI study contains 1,450 valid behavioral trials. Of these, 1,300 are ATTACK/PLACEBO carrier comparisons across 13 applicable site–carrier cells, and 150 are clean baseline trials.

A1, A2, and A3 are consumed through the native Crawl4AI path. A4 uses the frozen parser adapter. A5 uses the frozen LSB adapter and is applicable only to SmartApp; CreativeUI and CarCare A5 cells are not applicable rather than attack failures.

All applicable PLACEBO cells record 0/50 honeypot hits. The resulting matched lifts are:

- A1: +100 percentage points on all three sites;
- A2: +90 points on CreativeUI, +86 on CarCare, and +30 on SmartApp;
- A3: +100 points on all three sites;
- A4 adapter: +100 points on all three sites;
- SmartApp A5 adapter: +100 points.

For A2, the Wilson 95% intervals for the observed ATTACK rates are $[78.64\%, 95.65\%]$ on CreativeUI, $[73.81\%, 93.05\%]$ on CarCare, and $[19.10\%, 43.75\%]$ on SmartApp.

The clean baseline completes the canonical task in 50/50 trials on each site, with 0/50 honeypot hits.

Fifty-five additional executions are infrastructure-invalid, primarily because of provider transport/API failures. They are retained for provenance and excluded from behavioral denominators.

H.1 CRAWL4AI LATENCY

Component	Mean	P50	P95
Crawl4AI crawl	0.408 s	0.381 s	0.502 s
Provider	8.459 s	7.288 s	14.107 s
Total trial	8.944 s	7.747 s	14.611 s

Most end-to-end latency therefore comes from provider completion rather than crawling.

I MCAAG CAUSAL REPLAY ABLATIONS

The replay ablations separate carrier evidence, shared authorization, and recovery.

For representative A1–A4 proposals, bypassing shared authorization causes previously blocked honeypot proposals to become executable. Removing recovery preserves blocking but changes otherwise recoverable outcomes into unresolved outcomes.

For A5-LSB:

Ablation	Block	Recovery	HP exec.	Unresolved
Remove media finding	0/10	0/10	10/10	0/10
Bypass authorization	0/10	0/10	10/10	0/10
Remove recovery	10/10	0/10	0/10	10/10

All three ablations replay the same frozen proposals without a new provider/model call.

J Matched Benign Utility

The 85 PLACEBO replay controls cover ten valid site/carrier cells:

Site	A1	A2	A3	A4
CreativeUI	10	–	10	10
CarCare	–	10	10	10
SmartApp	10	5	5	5

CarCare–A1 is excluded because of its archived request-identity/construct problem. CreativeUI–A2 was not run after the underlying attack effect was absent. Neither cell is imputed.

Canonical success remains 51/85 before and after defense. False rejection is 0/51 among eligible canonical outcomes, unnecessary recovery is 0/85, and unresolved outcomes are 0/85.

K Full Browser Use Utility and Latency Results

The Browser Use evaluation contains 180 valid clean trials, with 30 trials for each site and defense condition.

Site	Defense	Success	E2E mean/med. (s)	Provider mean/med. (s)	MCAAG mean/med./P95 (ms)
CreativeUI	OFF	30/30	101.9/94.1	95.8/87.9	–
CreativeUI	ON	30/30	50.0/50.1	46.3/46.6	32.4/35.9/42.3
CarCare	OFF	30/30	177.2/153.1	170.0/147.1	–
CarCare	ON	28/30	86.6/80.4	81.8/77.1	50.4/37.3/137.6
SmartApp	OFF	30/30	128.9/112.1	123.9/107.9	–
SmartApp	ON	30/30	95.5/88.2	90.9/83.7	9.5/7.0/19.9

Aggregated over sites, defense OFF achieves 90/90 canonical successes and MCAAG ON achieves 88/90. Their Wilson 95% intervals are approximately $[0.959, 1.000]$ and $[0.923, 0.994]$.

Across all ON trials, local MCAAG processing has mean 30.8 ms, median 27.0 ms, P95 83.7 ms, and maximum 182.5 ms.

The lower ON end-to-end runtime is not interpreted as acceleration because ON trajectories also use fewer provider calls.

L Reproducibility and Evidence Availability

Experiment configurations are frozen before behavioral execution. Infrastructure-invalid attempts are retained but excluded from behavioral denominators. Negative attack results and failed defense variants are preserved.

Three historical MCAAG-v2 raw A2–A4 regression JSON files are unavailable due to local Git-object loss. They are not reconstructed. Claims relying on those experiments use the surviving preserved reports and evidence snapshots.