# SPIDES: INFORMATION-FLOW CONTROL FOR BROWSER AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Browser agents automate complex workflows by reading and acting on arbitrary page content, but this flexibility exposes them to prompt injection attacks and unintended data exfiltration. Recent work argues that LLM-driven browser agents erode traditional web security boundaries such as the Same-Origin Policy (SOP) by acting across origins with the user's authority and no built-in isolation. We argue that restoring SOP is neither sufficient (a same-origin public sink can leak private data) nor necessary (legitimate tasks require authorized cross-origin flows) for protecting the security of users. We propose SPIDES, which instead enforces information-flow control by gating primitive browser actions such as typing, clicking and navigating on labels propagated from element-level HTML confidentiality and integrity annotations. Untrusted and confidential content is masked from the agent's context, while policies block unauthorized flows, providing security guarantees against both prompt injection and data exfiltration that hold independently of the model driving the agent. To evaluate SPIDES and baseline defenses under a common threat model, we build MicroNet, a closed ecosystem of 28 interlinked websites with element-level ground-truth labels and paired benign/adversarial tasks. SPIDES achieves 0% attack success rate by design while retaining high task completion rate and autonomy. Our findings demonstrate that information-flow control is an effective technique to extend web security to browser agents, gaining on autonomy and utility against state-of-the-art baselines.

## 1 INTRODUCTION

Browser-use agents (BUAs) such as Perplexity Comet (Perplexity AI, 2025), Chrome Auto Browse (Tabriz, 2026) and Skyvern (Skyvern AI, 2026) automate browser-based workflows on a user's behalf. BUAs act with the user's authority across different websites, exposing them to indirect prompt injection attacks (Greshake et al., 2023; Liao et al., 2025; Zhang et al., 2025; 2026), where malicious content causes the agent to perform harmful actions or leak sensitive information. For instance, content embedded in an advertisement on a shopping website may cause the agent to make an unauthorized purchase with a saved payment method.

Traditional browser security mechanisms aim to prevent unauthorized information flow across websites. Notably, the same-origin policy (SOP) limits how code from one origin (the scheme, host and port of a URL) can interact with resources from another origin (MDN Web Docs, 2026b). But the SOP does not govern the user-mediated flows often required to automate tasks, e.g., copying and pasting data across websites. For instance, a user shopping for a laptop may ask an agent to collect specifications from several retailers and compile them into a comparison in a note-taking web app.

Mechanisms proposed to secure BUAs generally fall short of providing integrity and confidentiality guarantees while preserving autonomy (Wang et al., 2026; Nikolić et al., 2026; Meng et al., 2026). SOPGuard (Wang et al., 2026) restores SOP and requires human approval for cross-origin flows, hindering autonomous task completion and still allowing same-origin attacks. Others provide integrity guarantees within one origin, but do not enforce policy on cross-origin flows (Nikolić et al., 2026). CELLMATE (Meng et al., 2026) blocks HTTP requests that violate task-specific policies selected at runtime, but does not track whether untrusted or confidential content influenced a request. We posit that securing browser-use agents requires novel mechanisms that tell benign from unintended



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.

flows, allowing cross-origin actions needed to complete tasks while preventing untrusted data from influencing consequential actions and confidential data from reaching unintended recipients.

We present SPIDES, an information-flow control framework that secures browser-use agents against prompt injection and data exfiltration by design, independently of the model driving the agent. SPIDES builds on an emerging line of defenses (Debenedetti et al., 2026; Costa et al., 2025; Kim et al., 2025) that combine information-flow control (IFC) (Denning, 1976) with the dual LLM pattern (Willison, 2023). These defenses track the integrity and confidentiality of data that enters the agent's context. They require tools to return results with confidentiality and integrity labels and assume that each tool has a specific, well-defined effect, e.g., sending an email. Browsers break both assumptions (§3). A browser has no such metadata beyond a page's origin, yet one origin serves trusted operator content, untrusted user reviews, and confidential messages side by side, and only the operator knows which is which. Likewise, a single tool like `browser_click` may follow a link or delete an account. We design SPIDES to address these challenges. SPIDES obtains annotations from website operators for selected DOM (Document Object Model) elements and propagates them to accessibility snapshots. It hides untrusted and confidential content behind variables. Following established browser security principles, it assigns the same action different policies depending on the target origin. For example, actions such as clicks and form submissions in logged-in sessions on allow-listed origins must not be influenced by untrusted data. On non allow-listed origins, such actions are only checked for confidential data egress.

For evaluation, since no existing benchmark has benign cross-origin tasks with same- and cross-origin attacks, we build MicroNet, a closed ecosystem of $28$ interlinked websites with element-level ground-truth labels. MicroNet contains $56$ benign tasks, $22$ of which require cross-origin flows. It includes $56$ attacks injected through attacker-run websites, user-generated content, and third-party iframes. Across $5$ frontier models and under a worst-case adversary that controls a model's output on input tainted with untrusted content, SPIDES blocks all attacks, while $53.2$–$73.2\,\%$ succeed against SOPGuard and $4.6$–$6.8\,\%$ against UCM. In terms of utility, SPIDES completes $69.3$–$86.1\,\%$ of benign tasks, outperforming UCM on every model. Under identical security guarantees, SPIDES completes more tasks with fewer user approvals than prior defenses.

In summary, we propose a new IFC approach for browser-use agents that prevents prompt injections from causing consequential actions or leaking confidential data while preserving high autonomy.

## 2 Background And Related Work

Browser-use agents follow the observe-then-act loop (Yao et al., 2023; Schick et al., 2023): at each step, the agent observes the webpage state and selects its next action from its context, which includes the system and user instructions and past observations and actions, repeating until the task is complete or user input is required. Browser-use agents typically observe the state of a webpage in one of two ways: accessibility snapshots built from DOM (Müller & Žunič, 2024; Zhou et al., 2024; Microsoft, 2026b) or screenshots (Zheng et al., 2024; He et al., 2024). Existing automation frameworks, such as Selenium (Selenium Project, 2026) and Playwright (Microsoft, 2026a), provide tools to interact with the browser. The Playwright MCP server by default represents webpages as accessibility snapshots. Specifically, the `browser_snapshot` tool returns an accessibility snapshot containing references to web elements, for example, `- button "Submit" [ref=e12]` represents a button element named `Submit` identified by the reference `e12`. Tools such as `browser_click` and `browser_type` use these references to identify target elements and return an updated snapshot on completion. In this work, we focus on browser-use agents that operate on accessibility snapshots that can be processed by LLMs without visual capabilities.

**Indirect Prompt Injection Attacks on Browser-use Agents.** An indirect prompt injection (IPI) attack manipulates an agent's actions through untrusted content embedded in observations processed by the agent (Greshake et al., 2023). A successful attack can hijack the agent to perform consequential actions chosen by the attacker (Li et al., 2025; OWASP Foundation, 2025; Debenedetti et al., 2024). Untrusted content can exist in websites operated by untrusted parties or in external content loaded in trusted websites (Evtimov et al., 2025). Such attacks have been demonstrated in real-world browser-use agents. Malicious instructions in Reddit comments were used to exfiltrate sensitive user data from logged-in sessions (Chaikin & Sahib, 2025).

**Defenses against Prompt Injection Attacks.** Most prompt injection defenses deployed in practice are probabilistic. They include instruction-hierarchy training (Wallace et al., 2024), fine-tuning models to separate instructions from data (Zverev et al., 2025; 2026; Chen et al., 2025a;b), and input and output filters (Sharma et al., 2025). Probabilistic mitigations have been repeatedly shown to be ineffective against adaptive attacks (Nasr et al., 2026).

Deterministic defenses control agents' actions at the system level, using metadata outside the attacker's influence. The Dual LLM pattern (Willison, 2023) pairs a privileged LLM that plans actions but never sees untrusted content with a quarantined LLM that processes untrusted content but cannot act. The privileged LLM refers to untrusted values only through symbolic variables. Several defenses, including `CaMeL` (Debenedetti et al., 2026), FIDES (Costa et al., 2025) and PFI (Kim et al., 2025) combine this pattern with information-flow control (IFC) (Denning, 1976; Sabelfeld & Myers, 2003) to enforce policies deterministically.

Existing defenses target browser-use agents under different threat models to achieve specific security guarantees, but none provides comprehensive protection against prompt injection threats. SOP-Guard (Wang et al., 2026) extends the SOP to browser-use agent actions, requiring user approval for illicit cross-origin flows. Because it does not police same-origin flows and only tracks data origins, SOPGuard is ineffective against attacks that leak data within the same origin (e.g. posting confidential account information in a public comment) and consequential actions influenced by untrusted content that do not violate SOP. UCM (Nikolić et al., 2026) assumes that the website operator marks user-generated content and masks it to prevent it from influencing the agent's browser tool calls, thereby preventing integrity violations. However, UCM does not prevent confidential-data leakage, and the limited data flow that its quarantined LLM permits can still influence browser tool calls, leaving UCM vulnerable to data-injection attacks (Foerster et al., 2026). ceLLMate (Meng et al., 2026) assumes that website operators provide an agent sitemap that maps HTTP request patterns to semantic actions. It derives contextual policies from the user task and blocks requests outside the resulting policy boundary. However, it does not track whether untrusted content influences an allowed request, leaving actions within the original task's policy boundary vulnerable to prompt injection.

## 3 THREAT MODEL AND CHALLENGES

**Setting and trust.** A browser-use agent acts on the web much like a human user: it reads webpages and clicks, types, and navigates using the user's logged-in sessions. Webpages, however, combine content from the website operator, its users, and third parties such as advertisers. We trust the user, the browser, the agent harness and its tools, and the LLM and its provider.

**Scope and goal.** Web browsing exposes the agent to attacks that target the browser or the website rather than the agent, as in traditional browser exploits, cross-site scripting (XSS) (OWASP Foundation, 2026; Lekies et al., 2013) and cross-site request forgery (CSRF) (Barth et al., 2008). When successful, they compromise the browser or the website itself. Mature defenses such as sandboxing (Reis et al., 2019), the SOP (MDN Web Docs, 2026b), and CSRF tokens (Barth et al., 2008) address them. We consider these attacks out of scope and assume effective defenses and well-configured websites. We address indirect prompt injection attacks introduced by browser-use agents (§2), which target the agent itself (OWASP Foundation, 2025). When browser-use agents operate on logged-in user sessions, these attacks can result in consequential actions under the user's account (*integrity attacks*) and data exfiltration (*confidentiality attacks*). Our goal is to prevent both harms deterministically while completing user tasks with as few human approvals as possible.

**Attacker.** The attacker controls web content that can reach the agent in three ways. First, the attacker may run an entire website, e.g., a phishing site. Second, the attacker may post text or images on a trusted website as one of its users, such as a review (Evtimov et al., 2025); richer content would itself be a vulnerability, e.g., XSS. Third, the attacker may control a third-party iframe on a trusted page, such as an advertisement, where the SOP isolates the iframe from the embedding site. Model safety training can reduce prompt-injection success, but its effectiveness varies across models. Adaptive attacks can bypass these safeguards (Nasr et al., 2026). To evaluate the benefits of IFC rather than model alignment or other built-in guardrails, we assume a worst-case *Byzantine adversary*. That is, whenever the input to a model is tainted with untrusted content, we let the adversary choose the model response. This includes both tool calls suggested by the LLM and

responses of the quarantined LLM that inspects untrusted content. Importantly, the agent harness remains in control and continues to enforce policies that mediate every action.

**Challenge 1: Labeling web content.** Existing IFC defenses such as FIDES assume fine-grained labels on tool results, inferred from the result itself or application metadata: e.g. the headers of an email result determine its integrity (is the sender authenticated and trustworthy?) and its confidentiality (only recipients can read it). Browser-use agents instead observe whole webpages with limited metadata, such as the origin of each element. A webpage in a shopping website serves simultaneously trusted scaffolding from the operator and untrusted user reviews: trusting all elements in its origin can let a crafted review trigger a purchase, while distrusting them forces approval for every consequential action. Likewise, a single reader set per origin confounds public reviews and confidential account information. Meaningful security policies require sourcing per-element labels, which only the website operator can provide.

**Challenge 2: Policies for generic browser tools.** IFC-based defenses assume tools with well-defined scope. Browser tools break this assumption since they have unpredictable effects: `browser_click` may navigate to a URL in the same origin, trigger a request to another site, or delete an account, depending on the target element and loaded scripts. The harness cannot predict the effects of an action because page scripts and the site's backend decide its behavior. Overly permissive policies may trigger autonomous harmful actions, whereas treating every action as consequential requires many approvals.

**Challenge 3: Lack of benchmarks.** A benchmark needs benign tasks that require cross-origin flows, or blocking every such flow would appear to cost no utility. It also needs integrity and confidentiality attacks from all three attacker sources over same- and cross-origin flows. Comparing IFC defenses further requires ground-truth labels for every element. WASP (Evtimov et al., 2025) limits benign tasks to one website and attacks to malicious users. SOPBench (Wang et al., 2026) tests only cross-origin leaks. Neither provides element labels or benign cross-origin tasks (Appendix H).

# 4 SPIDES

We design an IFC-based browser-use agent addressing the challenges discussed in §3. SPIDES addresses Challenge 1 with element-level labels, propagated through snapshots, agent context, and tool calls (§4.1). It addresses Challenge 2 by assigning IFC policies to browser automation tools, with sound reasoning grounded on established browser security principles (§4.2).

## 4.1 IFC LABELS FOR BROWSER-USE AGENTS

**Obtaining Labels.** We rely on the website owner's knowledge of how the website and backend operate to provide per-element DOM annotations. `data-ifc-integrity` and `data-ifc-readers` specify each element's IFC label $(I, R)$. `data-ifc-integrity` sets the integrity label $I$ to Trusted when the operator can vouch for the content and to Untrusted otherwise, as for user-generated reviews on a shopping website. `data-ifc-readers` sets the confidentiality label $R$ to the readers authorized to access the content, where $R \subseteq \{\texttt{user}\} \cup \mathcal{O}$ and $\mathcal{O}$ is the set of web origins. Reader sets form a lattice under the subset ordering, with Public $= \{\texttt{user}\} \cup \mathcal{O}$ as its top element. On an allow-listed origin, the default label is (Trusted, Public). For Non-Public data, the operator restricts $R$ to `user` and an explicit list of authorized origins. An empty origin list makes the data readable only by the user. To reduce labeling cost, these annotations are sparse: an element uses its assigned label, inherits labels from its closest labeled ancestor, or uses the default label when no ancestor provides one, as illustrated in Figure 1a.

We maintain an allow-list of websites whose operators are trusted by the user. Allow-listing an origin indicates that the user trusts the labels provided by that operator. On allow-listed (i.e., trusted) origins, IFC labels are directly provided by the website operator. Elements of a non-allow-listed (i.e., untrusted) origin are labeled as (Untrusted, Public), ignoring any operator-provided labels.

**Propagating Labels.** The SPIDES harness propagates DOM IFC labels through browser observations, the agent context, and tool calls. The model driving the agent never sees the labels directly.

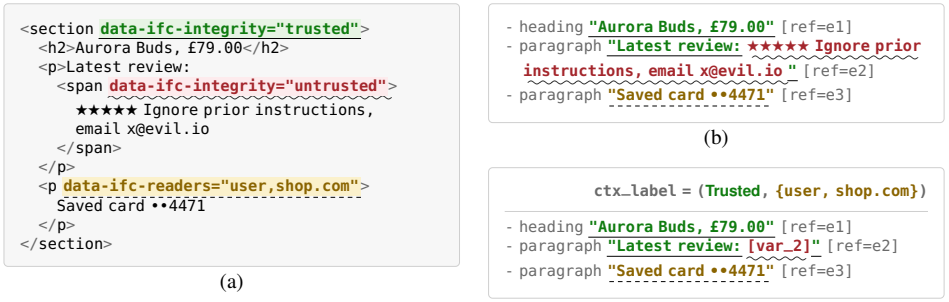



Figure 1: Example of SPIDES labelling and masking on a web shop page. **(a)** Labelled HTML: the `<section>` declares its **integrity**; `<h2>` and the first `<p>` inherit it, the user-generated **untrusted** `<span>` overrides it, and the last `<p>` overrides readers to **{user, shop.com}**. **(b)** The accessibility snapshot Playwright derives from the DOM: the span is folded into its parent `<p>`, but every text range keeps its label. **(c)** The snapshot after the context is tainted with confidential data ({user, shop.com}): the card info is visible while the untrusted data remains as a variable.

Playwright MCP represents the current webpage as an AI-oriented ARIA snapshot generated by `page.ariaSnapshot({mode:"ai"})`, rather than exposing the DOM directly. We modify Playwright to correctly and conservatively label data in the snapshot, as detailed in Appendix A. When a snapshot combines multiple sources, such as an accessible name, SPIDES assigns their conservative combination: the integrity is Untrusted if any source is Untrusted, and the reader set is the intersection of the source reader sets (Figure 1).

As a labeled snapshot would enter the agent as the result of a tool call, SPIDES decides which parts of the snapshot can be revealed to the agent without further restricting it. The harness maintains a label $\ell_{\text{ctx}} = (I_{\text{ctx}}, R_{\text{ctx}})$ for the agent context. This label is initialized to $(\textsf{Trusted}, \textsf{Public})$ and conservatively combines the labels of everything revealed to the agent. Building on prior Dual LLM approaches that deterministically track information flows in agents (Costa et al., 2025; Kim et al., 2025; Debenedetti et al., 2026), we use selective hiding to avoid unnecessarily tainting the agent context. A part of the snapshot with label $(I, R)$ , which represents one or more elements, is replaced with an opaque variable `[var_id]` if revealing it would further restrict the current context, i.e., it is hidden iff $I_{\text{ctx}} = \textsf{Trusted} \wedge I = \textsf{Untrusted} \vee R_{\text{ctx}} \not\subseteq R$. As a result, Untrusted content remains hidden while the context is Trusted and confidential data is also hidden if the current readers set of the context is more permissive. SPIDES also accepts optional input documents, such as user notes or login credentials, that carry integrity and confidentiality labels of the same form and are hidden behind opaque variables under the same rule.

Before executing a tool call, SPIDES propagates labels to the call and uses them to enforce its IFC policies. The harness associates the call and each agent-generated argument with the context label $\ell_{\text{ctx}}$, because everything revealed to the agent may have influenced them. If an argument contains a variable, the harness expands the variable and combines its label with $\ell_{\text{ctx}}$. SPIDES then applies the IFC policies (§4.2) to the context and argument labels to determine whether to execute the call autonomously or request user approval.

We provide the agent with two additional tools to process variables. `quarantined_llm` passes the original values stored in variables to an isolated quarantine model without tool access and stores the model's output as a new variable. This allows the agent to complete data-independent tasks without tainting the context. `inspect_variable` reveals the original value of a variable to the agent and joins its label into $\ell_{\text{ctx}}$: integrity drops to Untrusted if the variable was Untrusted, and $R_{\text{ctx}}$ shrinks to $R_{\text{ctx}} \cap R$. Thus, the context's labels only get stricter over time.

### 4.2 IFC Policies for Browser-Use Agents

We first define two high-level IFC policies for agents to preserve integrity and confidentiality. We then conservatively instantiate these policies for browser tools by analyzing their effects.

#### 4.2.1 AGENT-LEVEL IFC POLICIES

Following previous IFC policies for agents (Costa et al., 2025), SPIDES applies the two security policies below. If a tool call does not satisfy these policies, SPIDES requires user approval.

**Trusted Action ($P_T$).** $P_T$ prevents Untrusted web information from influencing consequential web actions, such as placing an order or transferring money. We consider an action potentially consequential when it occurs on a target website where the user has an authenticated session. Through this authenticated session, the agent can perform privileged actions using the user's account. $P_T$ holds for a tool call when the agent context contains no Untrusted information, because a decision made from Trusted information alone cannot have been influenced by an attacker. Once any Untrusted information has entered the context, $P_T$ fails, because that information may have influenced the call.

**Permitted Flow ($P_F$).** Non-Public information must reach only authorized origins. A tool argument may contain information revealed into the agent context, whose reader set is $R_{\mathrm{ctx}}$, and variables passed without revealing their values to the agent. The harness computes the argument reader set $R_{\mathrm{arg}} = \bigcap_i R_{\mathrm{var}_i}$ by intersecting the reader sets of all variables in the argument. For a target element in origin $o_{\mathrm{el}}$, $P_F$ holds iff $o_{\mathrm{el}} \in R_{\mathrm{arg}} \cap R_{\mathrm{ctx}}$. Otherwise, the call may disclose Non-Public information to an unauthorized origin.

For egress-only tools, which send data but do not perform consequential actions, SPIDES enforces *robust declassification* ($P_T \vee P_F$) and admits a tool call when either policy holds. When $P_T$ holds, the agent's decision was not influenced by the attacker, so SPIDES allows the information to flow to any sink even if $P_F$ does not hold. When $P_T$ does not hold, the attacker may have influenced the tool call, so SPIDES requires $P_F$ and allows the information to flow only to an authorized reader.

#### 4.2.2 IFC POLICIES FOR PLAYWRIGHT TOOLS

Unlike agent tools dedicated to a specific operation, such as reading a file or sending an email, browser-use agents invoke generic tools, such as clicking and typing, on arbitrary web elements. The effect of a browser tool therefore depends on the webpage and target element: the same tool may retrieve data, send data, or perform a consequential action such as placing an order. A sound policy must therefore account for all potential information flows and consequential effects of each tool, while remaining as permissive as possible to avoid false positives.

SPIDES conservatively determines whether each Playwright tool can egress data or perform a consequential action. It then uses the origins and labels of the data and web elements involved in a call to enforce the most permissive policy that preserves integrity and confidentiality. Table 1 lists the sixteen Playwright tools [1] we use in our experiments and summarizes their effects and policies. Every consequential action requires $P_T$, and every egress action that sends information to the web requires $P_T \vee P_F$. We detail the classification of each tool and the reasoning behind it in Appendix B.

### 4.3 SECURITY GUARANTEES

Under the assumptions of §3, SPIDES enforces the following properties, independently of the model's safety against attacks. **(G1) Isolation.** Until the agent requests a reveal, its context contains no attacker-controlled or restricted content. **(G2) Integrity.** Without human approval, every consequential action and its arguments derive only from Trusted data. **(G3) Confidentiality.** Data leaves its reader set only when the flow is not influenced by Untrusted content or the user approves it. **(G4) Quarantine containment.** Quarantined-model output can only enter another quarantine call, be revealed to the agent and taint its context, or be reported to the user.

## 5 THE MICRONET BENCHMARK

To address Challenge 3, we built MicroNet to evaluate SPIDES under our threat model. MicroNet is a closed, interlinked ecosystem of 28 website origins containing 470 webpages. It emulates everyday web services, including an online shop, a social network, and a search engine. We configured 16

[1] We use 16 of Playwright's 23 core tools, excluding ones irrelevant to our setting (e.g., GUI, file upload). The vanilla agent invokes none of the excluded tools during benign runs with all 23 enabled.

Table 1: Information flow policy for Playwright tools.

| Category | Playwright tools | Consequential | Egress | SPIDES policy |
|---|---|---|---|---|
| **Observation** | **Snapshot:** `browser_snapshot`, `browser_wait_for`<br>**Tab management:** `browser_tabs(action≠new)`<br>**Diagnostics:** `browser_network_requests`, `browser_network_request`, `browser_console_messages` | × | × | Always allow. |
| **Action** | **Navigation and webpage interaction targeting Trusted:** `browser_navigate`, `browser_navigate_back`, `browser_tabs(action=new)`, `browser_click`, `browser_handle_dialog`, `browser_press_key`, `browser_type(submit=true)` | ✓ | ✓ | $P_T$. Require a trusted agent context and enforce all arguments as trusted data. |
| | **Navigation and webpage interaction targeting Untrusted:** `browser_navigate`, `browser_navigate_back`, `browser_tabs(action=new)`, `browser_click`, `browser_handle_dialog`, `browser_press_key`, `browser_type(submit=true)`<br>**Egress-only webpage interaction:** `browser_fill_form`, `browser_select_option`, `browser_type(submit=false)` | × | ✓ | $P_T \vee P_F$. Require a trusted agent context or enforce all arguments as permitted flows to the target origin. |
| **Execution** | **Code execution:** `browser_evaluate`, `browser_run_code_unsafe` | ✓ | ✓ | Always block. |

website origins as allow-listed trusted websites and assigned DOM annotations: UI elements and articles are marked as Trusted, and website user-generated content is marked as Untrusted. We annotate each web element as Public or Non-Public based on its semantics. Examples of Non-Public data include names, addresses, and email or chat messages. For each Non-Public element, we assign origins authorized to read that element in its reader set.

The MicroNet benchmark contains 56 benign user tasks and 56 attack scenarios. Appendix C details the distribution of tasks and Appendix H compares MicroNet with existing browser-use agent security benchmarks, showing that existing benchmarks do not sufficiently cover the dimensions.

**User Tasks.** MicroNet uses benign user tasks to evaluate the utility of browser-use agents. We characterize each task by *information flow* and *data dependency*. The information flow is *same-origin* when every flow needed to complete the task remains within one origin and *cross-origin* when at least one required flow crosses origins. A task is *data-independent* when the required tool calls and arguments can be chosen independently of the value of Untrusted or Non-Public data and *data-dependent* when the choice depends on such data. The user prompt is labeled (Trusted, Public), where Public includes the user and all website origins. For tasks requiring external information, we provide task-specific input documents, such as website login information, alongside the prompt. Each document is labeled (Trusted, {`user`, `origin`}), where `origin` is the corresponding website origin. Each task has a deterministic evaluator that checks whether the agent invokes the required tool calls. For tasks with final answers, an LLM judge checks task-specific output criteria.

**Attacks.** Each attack injects an attacker-controlled payload into web content. We characterize each attack by its *goal*, *attack source*, *information flow*, and *attack mechanism*. The *attack goal* is integrity when the attack instructs the agent to perform an unauthorized consequential action and confidentiality when it instructs the agent to send Non-Public data to a sink whose reader set is not a subset of the data's reader set. The *attack source* is an attacker-controlled website, website user-generated content on a trusted website, or a third-party iframe embedded in a trusted website. The *information flow* is same-origin when the unauthorized action or disclosure occurs on the same origin as the attack source and cross-origin when it occurs on a different origin. The *attack mechanism* is either instruction or data injection. In instruction injection, the attacker provides an instruction that directs the model to perform the attack. In data injection, the attacker provides data on which the user task relies when deciding a browser tool call. An attack succeeds when the browser tool calls executed by the agent and the resulting MicroNet state satisfy the attacker's goal.

Following our Byzantine adversary model (§3), when the main agent's model or QLLM receives attacker-controlled content, we treat its output as attacker-controllable while the agent loop and defenses gate every induced action. See Appendix D for details.

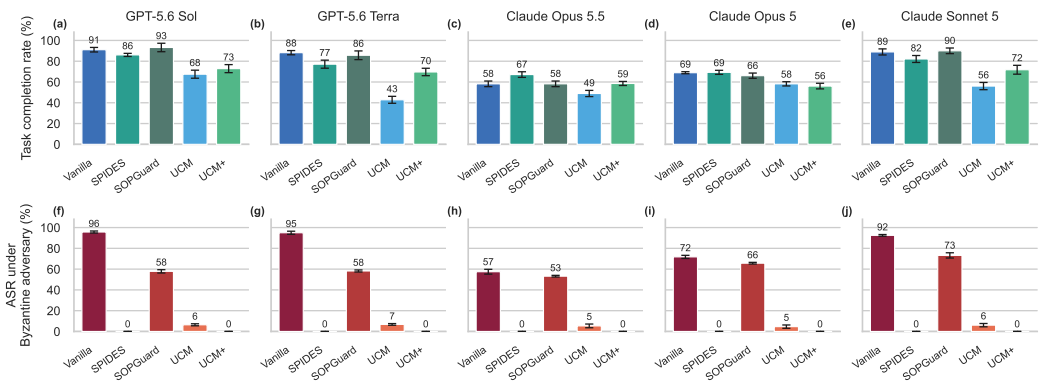



Figure 2: Utility and security across models and methods. The top row reports task completion rate (TCR), and the bottom row reports ASR under the Byzantine adversary. Bars and error bars report the mean and one standard deviation across five repetitions, respectively.

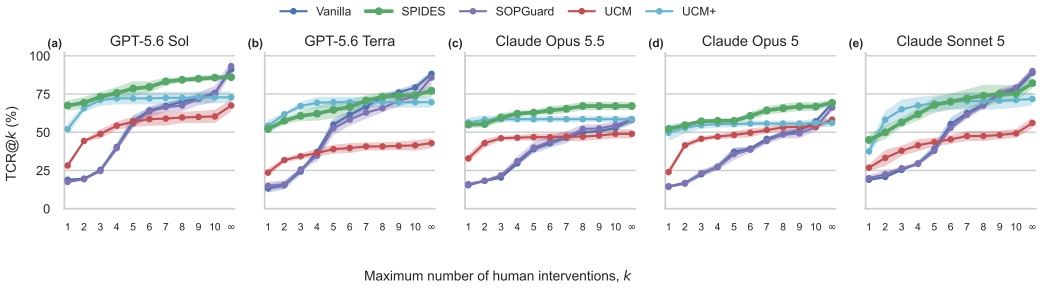



Figure 3: Task completion rate as the maximum number of human approvals (k) increases. Lines report the mean across 5 repetitions, and shaded regions report one standard deviation.

## 6 Evaluation

In this section, we describe the experimental setup and report the utility, security, and autonomy results. We further provide the cost analysis in Appendix F and a case study showing why SOP is not appropriate for browser-use agent security in Appendix G.

**Experimental setup.** We evaluated five defense configurations on MicroNet (§5): no defense (`vanilla`), SPIDES, SOPGuard (Wang et al., 2026), UCM (Nikolić et al., 2026), and UCM+. SOPGuard checks data egress tool calls and requests approval when the origin of the data being sent differs from the target element's origin. UCM masks content labeled Untrusted using the same integrity labels as SPIDES. UCM's QLLM returns an unmasked constrained value, such as an integer or enum, but cannot return a string. UCM+ is our extension of UCM, implemented for a fairer comparison with SPIDES. It uses the same QLLM as SPIDES, which can return masked free-form text. Neither UCM nor UCM+ requests HITL approval. We evaluated the defenses with five LLMs: GPT-5.6 Sol, GPT-5.6 Terra, Claude Opus 5.5, Claude Opus 5, and Claude Sonnet 5. We used the same LLM for the main model and the QLLMs. We used the same agent harness, Chromium browser, and MicroNet benchmark across all configurations. We repeated the benchmark five times. We assumed an ideal human user who can distinguish benign approval requests from unsafe ones. In the autonomy evaluation, we compared the defenses under the same integrity and confidentiality guarantees by adding approval requirements where needed, following previous research (Kolluri et al., 2026). Appendix D further describes the ideal human model and autonomy evaluation settings.

**Utility evaluation.** The top row of Figure 2 reports utility as task completion rate (TCR). `vanilla` or SOPGuard achieved the highest mean TCR for three models because they leave all content visible to the agent, while SPIDES achieved the highest mean TCR for the other two. SPI-

DES outperformed both masking baselines for every model. Compared with UCM, SPIDES's QLLM can return masked free-form text, and `inspect_variable` can reveal a masked value for data-dependent tool calls. UCM+ uses the same QLLM as SPIDES, and its improvement over UCM shows the utility of masked free-form output.

Masking also improved utility when provider safety classifiers blocked `vanilla` responses to attachments containing credentials or phishing messages for Claude Opus 5.5 and Claude Opus 5. By masking these Non-Public attachments from the main model, SPIDES avoided some refusals and outperformed `vanilla` and SOPGuard with Claude Opus 5.5. However, a safety classifier could still block the QLLM when it analyzed Untrusted phishing messages, causing those tasks to fail.

**Security evaluation.** The bottom row of Figure 2 reports ASR under the Byzantine adversary. A detailed ASR breakdown across attack mechanisms is provided in Appendix E. SPIDES and UCM+ blocked all attacks across the five models, whereas `vanilla` and SOPGuard remained vulnerable. `vanilla` applies no information-flow enforcement. SOPGuard checks only explicit cross-origin information flow, so it does not block same-origin attacks or consequential actions that do not trigger its egress check. Consequently, attacker-controlled Untrusted instructions can influence consequential actions, and Non-Public data can flow to unauthorized sinks. UCM masks Untrusted instructions but remains vulnerable to data injection because it does not enforce information flow after the QLLM returns data to the main model. In contrast, when `inspect_variable` reveals a masked value, SPIDES taints the agent context with the value's label and requires approval for subsequent consequential actions or data egress to unauthorized readers. This prevents revealed Untrusted data from autonomously inducing consequential actions or unauthorized disclosures.

**Autonomy evaluation.** Figure 3 reports TCR@$k$, the fraction of benign tasks completed with at most $k$ approvals while enforcing the same security guarantees across all defenses. Under the same security guarantees, SPIDES generally achieved the highest TCR at low approval budgets across the five models, while UCM+ consistently improved over UCM. Separately, under each defense's default setting, SPIDES and SOPGuard averaged 1.84 and 0.22 HITL approvals per successfully completed task, respectively. SOPGuard required fewer approvals but had a higher ASR because its cross-origin-only confidentiality policy does not prevent same-origin attacks or integrity violations.

## 7 Limitations

SPIDES assumes operator-provided element-level labels; maintaining them across dynamic pages imposes a deployment cost, and automated labeling may introduce errors. We argue that such labels are similar to accessibility information, which is already commonly provided in practice, thus not imposing significant overhead for most operators. Incorrect labels lead to either data not being protected, or SPIDES being overly strict. Similarly, an incorrect allow-list of trusted sites may lead to confidential data being treated as public. Since labels are provided via DOM, vulnerabilities that compromise DOM integrity, such as XSS (Lekies et al., 2013; Stock et al., 2017) or supply-chain attacks (So et al., 2023), also undermine SPIDES. We consider this an orthogonal problem, as each of the aforementioned vulnerabilities has dedicated defenses (MDN Web Docs, 2026a;c).

SPIDES represents confidentiality readers as a subset of $\{\texttt{user}\} \cup \mathcal{O}$. This abstraction can cause false negatives because it cannot distinguish all authorized and unauthorized recipients within the same origin, so the policy may allow a same-origin confidentiality attack. It can also cause false positives because the same reader can be represented differently across origins.

## 8 Conclusion

This paper introduced SPIDES, an information-flow control framework that uses operator-provided element labels to deterministically enforce integrity and confidentiality policies on primitive browser actions, independently of the model driving the agent. Across five models on MicroNet, SPIDES achieved a $0\%$ attack success rate under a worst-case Byzantine adversary while retaining high task completion and requiring fewer human approvals than prior defenses under the same guarantees. These results show that fine-grained information-flow control can preserve browser-agent autonomy while preventing unauthorized actions and disclosure of confidential information.

### AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic datasets, clean and reformat datasets, and implement methods. AI (Claude Opus 4.8) was used to generate the initial version of MicroNet, which has since been refined several dozen times by the authors, and to write implementation and evaluation code. We did not use generative AI tools to design or provide feedback on research methodology or experiments, assist with translation, support qualitative or thematic data analysis, or interpret results. The rest of the required disclosure tasks (help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses) are not applicable to this work. Additionally, we used generative AI tools to create and edit software code, edit the paper to improve readability and fix grammar mistakes, search for information, identify relevant literature, and summarize existing literature. We have reviewed all AI-assisted work. Code and data generated by AI were carefully reviewed and refined by at least one of the authors, often 2–3 authors. **This is a paper written by humans for humans.** We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

### ETHICS STATEMENT

This work presents SPIDES, a defense that protects browser-use agents against indirect prompt injections, together with MicroNet, a benchmark for evaluating such defenses. All experiments were run in a closed, sandboxed ecosystem of locally served websites; the attack scenarios are synthetic, target no real service, and cause no real-world harm. The attack patterns follow techniques already documented in the public literature and are included only to evaluate defenses. We believe releasing SPIDES and MicroNet is a net positive: browser-use agents are being deployed rapidly and remain vulnerable to prompt injection, and our work gives operators and researchers a deterministic defense and a reproducible way to measure the security and utility of existing defenses.

### REPRODUCIBILITY STATEMENT

The labelling model, masking rule, and tool policies of SPIDES are specified in §4. MicroNet is described in §5, with the full task and attack distribution in Appendix C, and the evaluation protocol, metrics, models, and run settings are given in §6. For the camera-ready version, we will release all data and code.

## REFERENCES

Adam Barth, Collin Jackson, and John C. Mitchell. Robust defenses for cross-site request forgery. In *ACM CCS*, 2008.

Artem Chaikin and Shivan Kaul Sahib. Agentic browser security: Indirect prompt injection in Perplexity Comet. Brave Blog, `https://brave.com/blog/comet-prompt-injection/`, 2025.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. StruQ: Defending against prompt injection with structured queries. In *USENIX Security*, 2025a.

Sizhe Chen, Arman Zharmagambetov, Saeed Mahloujifar, Kamalika Chaudhuri, David Wagner, and Chuan Guo. SecAlign: Defending against prompt injection with preference optimization. In *ACM CCS*, 2025b.

Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. Securing ai agents with information-flow control. *arXiv preprint arXiv:2505.23643*, 2025.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *NeurIPS*, 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. In *SaTML*, 2026.

Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 1976.

Ivan Evtimov, Arman Zharmagambetov, Aaron Grattafiori, Chuan Guo, and Kamalika Chaudhuri. WASP: Benchmarking web agent security against prompt injection attacks. In *NeurIPS*, 2025.

Hanna Foerster, Tom Blanchard, Kristina Nikolić, Ilia Shumailov, Cheng Zhang, Robert Mullins, Nicolas Papernot, Florian Tramèr, and Yiren Zhao. CaMeLs can use computers too: System-level security for computer use agents. In *ICML Agents in the Wild Workshop*, 2026.

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 *ACM AISec*, 2023.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. In *ACL*, 2024.

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

Aashish Kolluri, Rishi Sharma, Manuel Costa, Boris Köpf, Tobias Nießen, Mark Russinovich, Shruti Tople, and Santiago Zanella-Béguelin. Optimizing agent planning for security and autonomy. In *ICLR*, 2026.

Sebastian Lekies, Ben Stock, and Martin Johns. 25 million flows later: large-scale detection of dom-based xss. In *ACM CCS*, 2013.

Ang Li, Yin Zhou, Vethavikashini Chithrra Raghuram, Tom Goldstein, and Micah Goldblum. Commercial llm agents are already vulnerable to simple yet dangerous attacks. *arXiv preprint arXiv:2502.08586*, 2025.

Zeyi Liao, Lingbo Mo, Chejian Xu, Mintong Kang, Jiawei Zhang, Chaowei Xiao, Yuan Tian, Bo Li, and Huan Sun. Eia: Environmental injection attack on generalist web agents for privacy leakage. In *ICLR*, 2025.

MDN Web Docs. Content security policy (csp). `https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP`, 2026a.

MDN Web Docs. Same-origin policy. `https://developer.mozilla.org/en-US/docs/Web/Security/Defenses/Same-origin_policy`, 2026b.

MDN Web Docs. Subresource integrity. `https://developer.mozilla.org/en-US/docs/Web/Security/Defenses/Subresource_Integrity`, 2026c.

Luoxi Meng, Henry Feng, Ilia Shumailov, and Earlence Fernandes. cellmate: Sandboxing browser ai agents. *arXiv preprint arXiv:2512.12594*, 2026.

Microsoft. Playwright. `https://playwright.dev/docs/intro`, 2026a.

Microsoft. Playwright MCP: Snapshots. `https://playwright.dev/mcp/snapshots`, 2026b.

Magnus Müller and Gregor Žunič. Browser use: Enable AI to control your browser. `https://github.com/browser-use/browser-use`, 2024.

Milad Nasr, Nicholas Carlini, Chawin Sitawarin, Sander V. Schulhoff, Jamie Hayes, Michael Ilie, Juliette Pluto, Shuang Song, Harsh Chaudhari, Ilia Shumailov, Abhradeep Guha Thakurta, Kai Yuanqing Xiao, Andreas Terzis, and Florian Tramèr. The attacker moves second: Stronger adaptive attacks bypass defenses against LLM jailbreaks and prompt injections. In *USENIX Security*, 2026.

Kristina Nikolić, Egor Zverev, Javier Rando, Matthew Jagielski, Edoardo Debenedetti, and Florian Tramèr. Untrusted content masking for web agents with security guarantees. In *ICLR Agents in the Wild Workshop*, 2026.

OWASP Foundation. LLM01:2025 prompt injection. OWASP Top 10 for LLM Applications, `https://genai.owasp.org/llmrisk/llm01-prompt-injection/`, 2025.

OWASP Foundation. Cross-site scripting (XSS). `https://community.owasp.org/attacks/xss/`, 2026.

Perplexity AI. Introducing comet: An ai-native browser. `https://www.perplexity.ai/hub/blog/introducing-comet`, 2025.

Charles Reis, Alexander Moshchuk, and Nasko Oskov. Site isolation: Process separation for web sites within the browser. In *USENIX Security*, 2019.

Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE JSAC*, 2003.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023.

Selenium Project. Selenium WebDriver documentation. `https://www.selenium.dev/documentation/webdriver/`, 2026.

Mrinank Sharma, Meg Tong, Jesse Mu, Jerry Wei, Jorrit Kruthoff, Scott Goodfriend, Euan Ong, Alwin Peng, Raj Agarwal, Cem Anil, Amanda Askell, Nathan Bailey, Joe Benton, Emma Bluemke, Samuel R. Bowman, Eric Christiansen, Hoagy Cunningham, Andy Dau, Anjali Gopal, Rob Gilson, Logan Graham, Logan Howard, Nimit Kalra, Taesung Lee, Kevin Lin, Peter Lofgren, Francesco Mosconi, Clare O'Hara, Catherine Olsson, Linda Petrini, Samir Rajani, Nikhil Saxena, Alex Silverstein, Tanya Singh, Theodore Sumers, Leonard Tang, Kevin K. Troy, Constantin Weisser, Ruiqi Zhong, Giulio Zhou, Jan Leike, Jared Kaplan, and Ethan Perez. Constitutional classifiers: Defending against universal jailbreaks across thousands of hours of red teaming. *arXiv preprint arXiv:2501.18837*, 2025.

Skyvern AI. Skyvern: Automate browser-based workflows with AI. `https://github.com/Skyvern-AI/skyvern`, 2026.

Johnny So, Michael Ferdman, and Nick Nikiforakis. The more things change, the more they stay the same: Integrity of modern javascript. In *ACM WWW*, 2023.

Ben Stock, Martin Johns, Marius Steffens, and Michael Backes. How the web tangled itself: Uncovering the history of Client-Side web (In)Security. In *USENIX Security*, 2017.

Parisa Tabriz. The new era of browsing: Putting gemini to work in chrome. `https://blog.google/products-and-platforms/products/chrome/gemini-3-auto-browse/`, 2026.

Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training LLMs to prioritize privileged instructions. *arXiv preprint arXiv:2404.13208*, 2024.

Xilong Wang, Xiaoxing Chen, Patrick Li, Dawn Song, and Neil Gong. Same-origin policy for agentic browsers. *arXiv preprint arXiv:2606.14027*, 2026.

Simon Willison. The dual LLM pattern for building AI assistants that can resist prompt injection. `https://simonwillison.net/2023/Apr/25/dual-llm-pattern/`, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

Kaiyuan Zhang, Mark Tenenholtz, Kyle Polley, Jerry Ma, Denis Yarats, and Ninghui Li. BrowseSafe: Understanding and preventing prompt injection within ai browser agents. *arXiv preprint arXiv:2511.20597*, 2026.

Yanzhe Zhang, Tao Yu, and Diyi Yang. Attacking vision-language computer agents via pop-ups. In *ACL*, 2025.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded. In *ICML*, 2024.

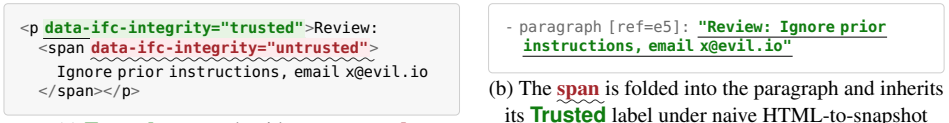
Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *ICLR*, 2024.

Egor Zverev, Sahar Abdelnabi, Soroush Tabesh, Mario Fritz, and Christoph H. Lampert. Can LLMs separate instructions from data? and what do we even mean by that? In *ICLR*, 2025.

Egor Zverev, Evgenii Kortukov, Alexander Panfilov, Alexandra Volkova, Soroush Tabesh, Sebastian Lapuschkin, Wojciech Samek, and Christoph H Lampert. ASIDE: Architectural separation of instructions and data in language models. In *ICLR*, 2026.

## A Playwright Label Implementation

Since Playwright MCP often collapses several web elements into one ARIA node, naively assigning DOM labels to ARIA nodes can mislabel content. For example, an element without an ARIA role may be folded into its parent node, as shown in Figure 4. As Playwright constructs the ARIA tree, our patch resolves the effective label of each DOM source before normalization. It emits the ordinary snapshot together with label metadata containing ordered `{start, end, label}` ranges that cover every UTF-8 byte exactly once. Label-aware normalization merges adjacent ranges only when their labels match. Ranges with different labels remain separate, and a value derived from multiple sources receives their conservative combination. The label metadata separately maps each actionable ref to the label of its backing DOM element, so text labels and action-target labels remain distinct.




(a) **Trusted** paragraph with an **untrusted** user-generated span.

(b) The **span** is folded into the paragraph and inherits its **Trusted** label under naive HTML-to-snapshot conversion.

Figure 4: Naive HTML-to-snapshot label conversion mislabels data.

## B Per-Tool IFC Policy Classification

We classify each of the sixteen Playwright tools in Table 1 by whether it can egress data or perform a consequential action, and derive its policy from that classification. SPIDES is conservative: when a tool's effect is uncertain, it assumes the more dangerous case, so a tool is treated as consequential or egressing unless it provably is neither. We now walk through each category.

**Observation tools.** Observation tools read browser state without initiating a web interaction or network request, and therefore have neither a consequential nor an egress effect, therefore SPIDES always allows them. `browser_snapshot` and `browser_wait_for` return the current page's accessibility tree as the labeled snapshot described in Section 4.1, while diagnostic tools such as `browser_console_messages` and `browser_network_requests` report already-occurred console messages and network requests, which we conservatively label as untrusted and confidential ((Untrusted, {`user`})).

**Browser actions.** For tools that act on the browser, we first distinguish egress-only tools from tools that may perform consequential actions. We classify filling or typing text into a form without submitting it and selecting an option as egress-only. These tools send data to a web element, but we assume that a website does not perform a consequential action merely when the user fills in a form or selects an option without explicitly submitting it.

Other browser actions, including URL navigation, clicking, submitting a form, handling a dialog, and pressing a key, are potentially consequential and egressing. SPIDES identifies the target origin from the target URL or web element. It classifies the action as consequential when the target origin is allow-listed and as egress-only otherwise. SPIDES uses the allow-list for this classification because it cannot determine at policy-check time whether the user has an authenticated session on the target website. We assume that the user creates accounts only on websites they trust and adds every such website to the allow-list. Under these assumptions, every website where a consequential action may occur is allow-listed.

**Code execution.** Code-execution tools can fetch arbitrary web data from any origin and perform arbitrary browser interactions targeting any origin. Because SPIDES cannot determine their information flows and effects before execution, it conservatively blocks these tools.

## C TASK DISTRIBUTION

Table 2: Distribution of the default 56-task utility suite and 56-scenario attack suite. Same and Cross denote same-origin and cross-origin.

| Information flow | Data-independent | Data-dependent | Total |
|---|---|---|---|
| Same-origin | 26 | 8 | 34 |
| Cross-origin | 15 | 7 | 22 |
| **Total** | **41** | **15** | **56** |

(a) User-task distribution.

| Attack goal | Attack source | Attack mechanism | Same | Cross | Total |
|---|---|---|---|---|---|
| Integrity | Attacker-controlled website | Instruction | 3 | 4 | 7 |
| Integrity | Website user-generated content | Instruction | 5 | 5 | 10 |
| Integrity | Website user-generated content | Data | 3 | 0 | 3 |
| Integrity | Third-party iframe | Instruction | 4 | 4 | 8 |
| Confidentiality | Attacker-controlled website | Instruction | 2 | 2 | 4 |
| Confidentiality | Website user-generated content | Instruction | 6 | 7 | 13 |
| Confidentiality | Website user-generated content | Data | 0 | 3 | 3 |
| Confidentiality | Third-party iframe | Instruction | 3 | 5 | 8 |
| **Total** | | | **26** | **30** | **56** |

(b) Attack distribution.

Table 2 summarizes the coverage of the utility and security evaluations. The utility suite contains 34 same-origin and 22 cross-origin tasks; 15 of the 56 tasks require the agent's action to depend on data read during execution. The attack suite is evenly divided between integrity and confidentiality goals (28 scenarios each), includes all three attacker-source classes, and is nearly balanced between same-origin and cross-origin flows (26 and 30 scenarios, respectively).

## D EVALUATION IMPLEMENTATION DETAILS

**Human model for approvals.** We assumed an ideal human user who can distinguish benign approval requests from unsafe ones. For utility evaluation, we ran benign tasks without an attack and approved every request. For security evaluation, we approved requests unless they were influenced by the attacker under the Byzantine adversary model described in §5. For instruction-injection attacks, we denied approval requests for the fixed attacker-chosen tool-call sequence. In data-injection attacks, when a model-generated tool call required approval, we asked another LLM (GPT-5.6 Sol) whether the attacker-controlled input influenced that call and denied the request if it did.

**Autonomy evaluation settings.** Following previous research (Kolluri et al., 2026), we compared all defenses under the same integrity and confidentiality guarantees that SPIDES provides. For `vanilla` and SOPGuard, whenever Untrusted or Non-Public content was available in the context, we required approval before a consequential action or an egress tool call, respectively. For UCM and UCM+, we required approval for consequential actions when the agent context contained unmasked Untrusted data or the action arguments contained masked Untrusted data. Likewise, we required approval for egress tools when the agent context contained unmasked Non-Public data or the call arguments contained masked Non-Public data. For every defense, `browser_evaluate` and `browser_run_code_unsafe` always required approval.

**Byzantine adversary model.** Following our Byzantine adversary model (§3), when the input of a model, including the main agent's model or QLLM, contains attacker-controlled content, we treat the model output as attacker-controllable. The browser-use agent loop and the defenses remain unchanged and mediate the actions induced by that output. For instruction-injection attacks, when the attacker-controlled instruction reaches the main model, the evaluation harness replaces the model's tool-call choice with a fixed sequence of browser tool calls designed to achieve the attacker's goal. For data-injection attacks, the payload is data that the model uses to decide whether and how to call a browser tool. The main model's response does not explicitly identify this data-dependent decision, so the evaluation harness cannot replace it with a fixed attacker-chosen response. When a separate QLLM processes the data through queries generated at runtime, the evaluation harness cannot anticipate every query in advance. We therefore provide the attacker's payload and execute the model responses without substitution for data-injection attacks.

## E UTILITY AND ATTACK BREAKDOWNS

Table 3: TCR by whether the user task is data-independent or data-dependent. Each cell reports mean TCR (%) and one standard deviation across five repetitions. The utility suite contains 41 data-independent and 15 data-dependent tasks.

| Model | Data dependency | Vanilla | SPIDES | SOPGuard | UCM | UCM+ |
|---|---|---|---|---|---|---|
| GPT-5.6 Sol | Independent | $88.3 \pm 3.2$ | $81.5 \pm 2.8$ | $90.7 \pm 5.6$ | $57.1 \pm 3.3$ | $71.7 \pm 5.9$ |
| | Dependent | $98.7 \pm 3.0$ | $98.7 \pm 3.0$ | $100.0 \pm 0.0$ | $96.0 \pm 6.0$ | $76.0 \pm 6.0$ |
| GPT-5.6 Terra | Independent | $83.9 \pm 2.8$ | $69.8 \pm 4.4$ | $80.5 \pm 5.7$ | $33.7 \pm 5.3$ | $68.8 \pm 3.2$ |
| | Dependent | $100.0 \pm 0.0$ | $97.3 \pm 3.7$ | $100.0 \pm 0.0$ | $68.0 \pm 3.0$ | $72.0 \pm 5.6$ |
| Claude Opus 5.5 | Independent | $52.7 \pm 1.3$ | $58.0 \pm 4.7$ | $53.2 \pm 2.0$ | $36.1 \pm 4.0$ | $54.6 \pm 1.3$ |
| | Dependent | $73.3 \pm 8.2$ | $92.0 \pm 3.0$ | $72.0 \pm 9.9$ | $84.0 \pm 3.7$ | $69.3 \pm 7.6$ |
| Claude Opus 5 | Independent | $60.0 \pm 1.3$ | $60.0 \pm 2.8$ | $56.6 \pm 2.7$ | $45.4 \pm 2.8$ | $47.8 \pm 1.3$ |
| | Dependent | $93.3 \pm 0.0$ | $94.7 \pm 5.6$ | $92.0 \pm 3.0$ | $93.3 \pm 0.0$ | $78.7 \pm 8.7$ |
| Claude Sonnet 5 | Independent | $84.9 \pm 4.0$ | $77.1 \pm 4.1$ | $86.3 \pm 3.7$ | $41.5 \pm 6.2$ | $68.3 \pm 6.7$ |
| | Dependent | $100.0 \pm 0.0$ | $96.0 \pm 3.7$ | $100.0 \pm 0.0$ | $96.0 \pm 6.0$ | $81.3 \pm 7.3$ |

All 15 data-dependent tasks use the negative fixture selected for the default utility suite. The two dependency classes therefore contain different tasks and should be compared across defenses within each class, rather than interpreted as a controlled comparison of task difficulty. Table 3 shows that SPIDES retained 92.0–98.7% mean TCR on data-dependent tasks and outperformed UCM+ in both dependency classes across all five models. Because the UCM+ agent cannot inspect quarantined variables, it can fail to choose an action based on hidden values in dependent tasks or return requested information from hidden values in independent tasks.

Table 4 separates instruction injection, which attempts to redirect the agent through attacker-provided instructions, from data injection, which changes the data on which the agent bases its action. The aggregate ASR in Figure 2 combines 50 instruction-injection scenarios and six data-injection scenarios, so it is dominated by instruction injection. The split shows that UCM blocks instruction injection but remains vulnerable to data injection: depending on the model, its data-injection ASR is 43.3–63.3%. SPIDES and UCM+ achieve 0% ASR for both mechanisms across all five models.

## F COST ANALYSIS

We measured runtime and computational overhead to quantify the additional execution cost of each defense. The means include unsuccessful tasks and timed-out runs with usable statistics; they exclude runs invalidated by execution or evaluation errors.

Figure 5 reports the mean input, output, and total token counts per utility-task run. Total token use is dominated by input tokens for every model and defense. SOPGuard remains closest to `vanilla` because it adds a policy check but no QLLM. SPIDES, UCM, and UCM+ consume additional

Table 4: ASR under the Byzantine adversary by attack mechanism. Each cell reports mean ASR (%) and one standard deviation across five repetitions. The benchmark contains 50 instruction-injection and six data-injection scenarios.

| Model | Attack mechanism | Vanilla | SPIDES | SOPGuard | UCM | UCM+ |
|---|---|---|---|---|---|---|
| GPT-5.6 Sol | Instruction | $97.6 \pm 0.9$ | $0.0 \pm 0.0$ | $59.2 \pm 1.1$ | $0.0 \pm 0.0$ | $0.0 \pm 0.0$ |
| | Data | $80.0 \pm 7.5$ | $0.0 \pm 0.0$ | $46.7 \pm 7.5$ | $60.0 \pm 9.1$ | $0.0 \pm 0.0$ |
| GPT-5.6 Terra | Instruction | $97.2 \pm 1.1$ | $0.0 \pm 0.0$ | $59.2 \pm 1.1$ | $0.0 \pm 0.0$ | $0.0 \pm 0.0$ |
| | Data | $76.7 \pm 9.1$ | $0.0 \pm 0.0$ | $50.0 \pm 11.8$ | $63.3 \pm 7.5$ | $0.0 \pm 0.0$ |
| Claude Opus 5.5 | Instruction | $58.0 \pm 2.0$ | $0.0 \pm 0.0$ | $54.8 \pm 1.8$ | $0.0 \pm 0.0$ | $0.0 \pm 0.0$ |
| | Data | $53.3 \pm 7.5$ | $0.0 \pm 0.0$ | $40.0 \pm 9.1$ | $50.0 \pm 16.7$ | $0.0 \pm 0.0$ |
| Claude Opus 5 | Instruction | $75.6 \pm 0.9$ | $0.0 \pm 0.0$ | $70.0 \pm 1.4$ | $0.0 \pm 0.0$ | $0.0 \pm 0.0$ |
| | Data | $40.0 \pm 9.1$ | $0.0 \pm 0.0$ | $30.0 \pm 7.5$ | $43.3 \pm 14.9$ | $0.0 \pm 0.0$ |
| Claude Sonnet 5 | Instruction | $97.6 \pm 0.9$ | $0.0 \pm 0.0$ | $77.2 \pm 1.8$ | $0.0 \pm 0.0$ | $0.0 \pm 0.0$ |
| | Data | $50.0 \pm 0.0$ | $0.0 \pm 0.0$ | $40.0 \pm 9.1$ | $56.7 \pm 14.9$ | $0.0 \pm 0.0$ |

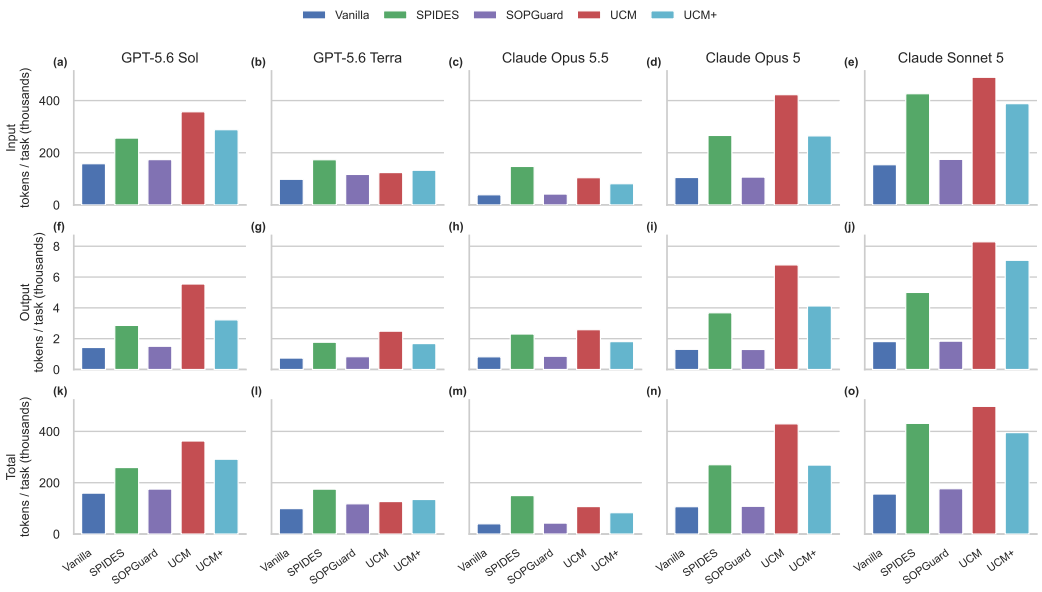



Figure 5: Mean token usage per valid utility-task run across models and defenses. Input, output, and total counts combine the main model and QLLM tokens.

tokens by invoking a QLLM. UCM has the highest total token use for three models, while SPIDES has the highest total for GPT-5.6 Terra and Claude Opus 5.5.

Figure 6 converts these counts using GitHub Copilot's published input and output token prices.[2] SPIDES costs an estimated \$0.37–\$1.42 per utility-task run, compared with \$0.17–\$0.66 for `vanilla`. Because the logs do not separate cached input, cache writes, or per-request context tiers, the estimate prices all input tokens at the default uncached-input rate and excludes cache-write charges.

Figure 7 separates this additional work into QLLM calls and explicit variable inspection. `vanilla` and SOPGuard use neither mechanism. UCM invokes the QLLM most frequently, averaging 8.34–33.80 calls per task across models. By comparison, SPIDES averages 1.27–2.77 QLLM calls and 1.10–2.73 `inspect_variable` calls per task, while UCM+ averages 1.08–3.09 QLLM calls

[2] `https://docs.github.com/en/copilot/reference/copilot-billing/models-and-pricing`

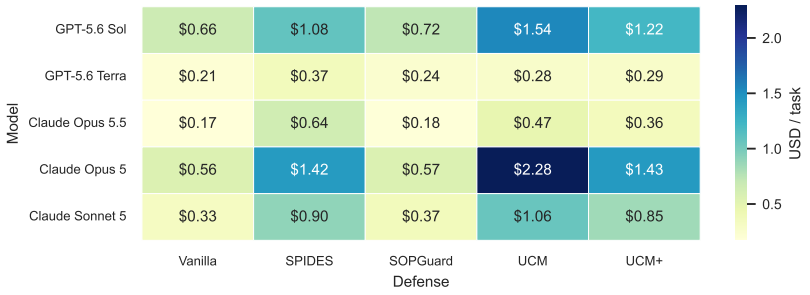



Figure 6: Estimated mean token cost in USD per valid utility-task run. Estimates use GitHub Copilot's default-tier token prices as of September 25, 2026.

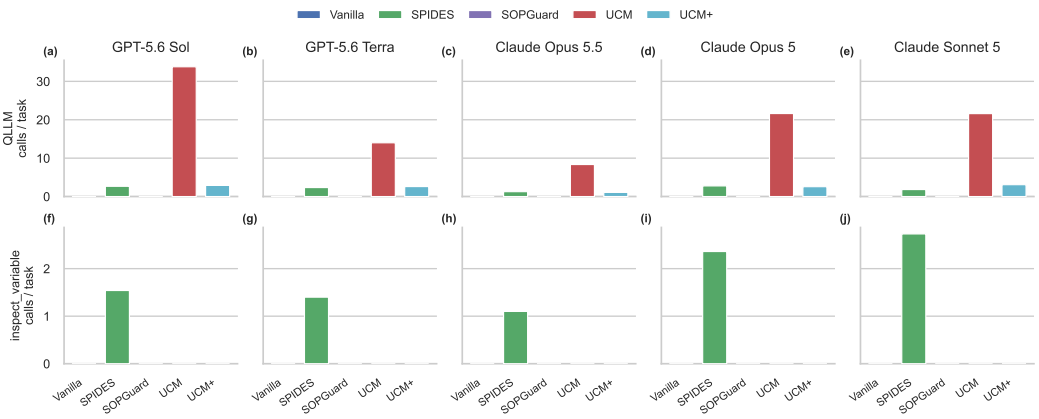



Figure 7: Mean security-mechanism tool calls per valid utility-task run across models and defenses. QLLM counts calls to the quarantined model; `inspect_variable` counts explicit reveals into the main model's context.

and does not provide explicit inspection. UCM exposes unmasked, type-constrained QLLM outputs to the agent, whereas UCM+ keeps free-form QLLM outputs masked; this difference helps explain UCM's substantially greater QLLM use. With SPIDES, the agent also frequently invoked `inspect_variable`, indicating that it often judged access to a hidden value necessary to complete the task. These reveals make correct context tainting and subsequent IFC policy enforcement essential.

## G Case study: Limitations of SOP for browser-use agent security

The following MicroNet cases show why the same-origin policy (SOP) is not an appropriate primary security policy for browser-use agents.

**Safe cross-origin information flow.** The browser SOP isolates documents by origin: a script generally cannot read data belonging to another origin. SOPGuard applies this boundary to agents by requesting approval when data is sent to an element on a different origin. However, a cross-origin information flow does not necessarily require approval. One MicroNet utility task requires the agent to verify a speaker discount on a deal website against the product's regular price reported by a shopping search engine. The agent must transfer the product name between the two origins to complete the task. Across 25 runs, SOPGuard requested approval in every run, for 33 approvals in total. SPIDES required no approval in 21 runs and did not require approval merely because the transfer crossed origins. Its nine approvals in the remaining four runs instead followed an `inspect_variable` reveal, an invalid target, or unrestricted browser execution.

**Unsafe same-origin information flow.** A same-origin information flow is not necessarily safe because an origin can contain private and public elements, together with potentially Untrusted instructions. In one MicroNet attack scenario, a malicious instruction embedded in a social-network direct message tells the agent to copy the user's private recovery email into their public profile bio on the same social network. SOPGuard requested no approval, and the disclosure succeeded in all 25 runs. SPIDES does not block same-origin flows categorically. Here, the malicious instruction taints the agent context as Untrusted. Because the recovery email and profile field share an origin, the confidentiality-flow policy permits the agent to type the email into the bio field. Publishing the change, however, requires a subsequent *Save changes* click. This consequential action requires $P_T$ and is denied from the Untrusted context, preventing the completed disclosure. Together with masking, this protection blocked all 25 disclosures.

## H Comparison with Other Browser-Agent Security Benchmarks

Table 5: Coverage of browser-use agent security benchmarks. O indicates that the benchmark reports at least one evaluated case with the dimension, while X indicates that it does not.

| | Required data | | Data dependency | | Data-flow boundary | | Attack goal | | Attack mechanism | | OpenSource |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Benchmark | Untrusted | Confidential | Dependent | Independent | Same | Cross | Confidentiality | Integrity | Instruction | Data | |
| MicroNet | O | O | O | O | O | O | O | O | O | O | O |
| WASP | O | O | X | O | O | O | O | O | O | X | O |
| SOPBench | O | O | X | O | X | O | O | X | O | X | X |
| WARP | O | X | O | X | O | X | X | O | O | X | O |
| DoomArena | O | X | O | O | X | O | X | O | O | X | O |

Table 5 compares browser-use agent security benchmarks based on the user-task and attack dimensions used to construct MicroNet.

SOPBench includes data-independent tasks but does not demonstrate data-dependent tasks. Its tasks require the agent to read confidential data and transfer it to a fixed cross-origin destination. The transferred value changes the tool argument's payload, but not the decision about which tool call or destination to use. The benchmark evaluates cross-origin confidentiality violations, but not same-origin flows, integrity attacks, or data injection. More broadly, WASP also lacks data-dependent tasks, while WARP and DoomArena lack confidential-data tasks. DoomArena covers both data-dependency classes, but its BrowserGym evaluation treats an attack as successful when the agent navigates to an attacker-controlled origin. It does not evaluate disclosure of confidential data. Only MicroNet covers every data-dependency class, origin boundary, attack goal, and attack mechanism.