Semantic Policy Enforcement for Low-Level Computer Use Agent Tools: Taint-Aware Policies with Noisy Perception

Anonymous Author(s)

ABSTRACT

Computer Use Agents (CUAs) interact with GUIs through low-level actions such as click and type, presenting fundamental challenges for security policy enforcement: these primitives lack the semantics needed for meaningful data-flow restrictions. We formalize three policy abstraction levels—syntactic, semantic, and contextual—and evaluate them through Monte Carlo simulations incorporating realistic modeling improvements: (i) policies decide on *predicted* risk from noisy confusion models rather than ground-truth risk oracles; (ii) a three-way enforcement model (allow / flag-for-review / block) with finite user review budgets; (iii) explicit taint propagation from untrusted reads to downstream sink actions; and (iv) injection detection rules that fire on taint features rather than global risk thresholds. Evaluated on 500 task scenarios across 8 domains, our contextual taint-aware policy achieves an F1 score of 0.664 with 53.2% injection detection, compared to 0.449 F1 and 12.5% for syntactic baselines, while preserving 85.3% task utility. Ablation analysis reveals that taint-based rules account for the largest share of injection detection improvement (drop from 54.3% to 30.0% when removed), while the review mechanism enables a favorable safety–utility balance. These results demonstrate that data-flow awareness and calibrated perception models are essential for securing CUA tool invocations beyond naive risk-threshold filtering.

1 INTRODUCTION

Computer Use Agents (CUAs)—AI systems that interact with software through GUI-level actions such as clicking, typing, and navigating—have created new security challenges [2]. Within the Dual-LLM architecture, control-flow isolation separates privileged planning from quarantined perception, yet data flow remains vulnerable because the perception model's outputs can steer execution [2].

Standard information-flow policies [1, 6] can govern semantically rich APIs, but CUA tools like `click(x,y)` lack intrinsic semantics. This motivates the question: *How can we design semantic security policies for low-level CUA tool invocations that model realistic perception noise and data-flow constraints?*

Prior work on this problem [10] used an oracle-based simulation where policies gated on ground-truth risk levels, effectively assuming perfect perception. That design suppressed false positives and inflated contextual policy performance. We address six specific limitations identified in review:

(1) **Noisy perception model (review A):** Policies now decide on *predicted* risk sampled from per-level confusion matrices, not ground-truth risk.
(2) **Three-way enforcement (review B):** Actions can be allowed, flagged for user review, or blocked. A finite review budget per task models realistic user attention.
(3) **Taint propagation (review C):** `READ_PAGE` actions introduce taint; subsequent sink actions (type, submit, send, auth, execute, modify) inherit it.
(4) **Taint-based injection detection (review D):** Injection rules fire on tainted sinks rather than a global risk filter, constraining detection to data-flow features.
(5) **Fair comparison (review E):** All policy levels are evaluated on identical scenario sets per trial.
(6) **Stable seeding (review F):** Domain-dependent seeds use `hashlib.md5` instead of Python's `hash()`.

2 PROBLEM FORMULATION

2.1 CUA Tool Actions and Taint Model

We model CUA interactions as sequences of typed actions $a = (\tau, t, c, r, \hat{r}, \textit{tainted})$ where $\tau \in \mathcal{T}$ is the action type (navigate, click, type, read, submit, download, execute, modify, send, auth), t is the target, c is context, r is the ground-truth risk level (hidden from the policy), $\hat{r}$ is the predicted risk (available to the policy), and *tainted* indicates whether the action carries data originating from an untrusted `READ_PAGE`.

Taint propagation follows a simple forward model inspired by dynamic taint analysis [7]: once a `READ_PAGE` action executes, all subsequent sink actions (`TYPE_TEXT`, `SUBMIT_FORM`, `SEND_MESSAGE`, `AUTH_ACTION`, `EXECUTE_JS`, `MODIFY_SETTINGS`) are marked tainted. Prompt injection attacks are modeled as deceptive tainted actions with escalated ground-truth risk.

2.2 Risk Prediction Model

For each policy level $\ell \in \{\text{syntactic}, \text{semantic}, \text{contextual}\}$, we define a 5×5 confusion matrix C_ℓ where $C_\ell[i, j] = P(\hat{r} = j \mid r = i)$. Higher policy levels have more accurate confusion matrices (e.g., contextual achieves 88% accuracy for critical risks vs. 70% for syntactic). This replaces the oracle model where policies directly observed r.

2.3 Three-Way Enforcement

Each policy rule specifies an enforcement outcome: Block (hard deny), Flag (queue for user review), or Allow. Flagged actions consume from a per-task review budget $B = 5$; when the budget is exhausted, flags default to blocks. User approval probability depends on the true risk: safe actions are approved with probability 0.95, while risky actions are approved with probability $0.85 \times 0.5 = 0.425$.

2.4 Policy Levels

- **Syntactic**: 6 rules covering 6 action types. High FPR (0.12–0.28), high FNR (0.14–0.24). Mix of block and flag outcomes.
- **Semantic**: 8 rules covering 8 action types. Moderate FPR (0.03–0.10), moderate FNR (0.08–0.14). Mix of block and flag outcomes.
- **Contextual**: 8 standard rules plus 6 *taint-specific* rules that only fire on tainted sink actions. Low FPR (0.01–0.05), low FNR (0.04–0.07). Taint rules enable data-flow-aware injection detection.

Table 1: Revised policy comparison over 15 trials of 500 tasks with noisy perception and review budget. Values are mean ± std.

Metric	None	Syntactic	Semantic	Contextual
Safety Recall	0.000	0.293±0.012	0.418±0.015	0.498±0.015
Precision	0.000	0.960±0.011	0.995±0.003	0.996±0.003
F1 Score	0.000	0.449±0.014	0.588±0.016	0.664±0.014
FPR	0.000	0.006±0.002	0.001±0.001	0.001±0.001
Utility	1.000	0.905±0.004	0.872±0.005	0.853±0.006
Inj. Detect	0.000	0.125±0.024	0.238±0.051	0.532±0.043
Task Compl.	1.000	0.562±0.015	0.454±0.021	0.409±0.018
Flag Rate	0.000	0.081±0.006	0.092±0.006	0.106±0.005

3 METHODOLOGY

3.1 Simulation Framework

We simulate CUA task execution via Monte Carlo sampling over 6 task types and 8 application domains. For each trial: (1) generate N task scenarios with taint propagation; (2) for each policy level, predict risk via the confusion model; (3) enforce rules using predicted risk and taint status; (4) resolve flags against the review budget; (5) score against ground-truth labels.

3.2 Evaluation Metrics

We measure safety recall (risky actions blocked), safety precision (blocked actions that were truly risky), F1 score, false positive rate, utility score (necessary actions allowed), injection detection rate, task completion rate, flag rate, and flag approval rate.

4 RESULTS

4.1 Main Experiment

Table 1 presents results over 15 trials of 500 tasks. With noisy perception, all policies show lower safety recall than oracle-based baselines—contextual achieves 49.8% vs. the prior oracle-based 94.7%—reflecting the realistic difficulty of risk classification. Contextual policies achieve the highest F1 (0.664), a 48% improvement over syntactic (0.449). Precision remains high across all active policies (>0.96), indicating that noisy perception primarily reduces recall rather than introducing false positives. The flag rate increases from syntactic (8.1%) to contextual (10.6%), reflecting broader rule coverage.

4.2 Pareto Frontier Analysis

Figure 1 shows the safety–utility Pareto frontier. Compared to the prior oracle-based model, the frontier has shifted: contextual policies no longer achieve near-perfect safety recall. Instead, they occupy a more realistic position at (0.50, 0.85), offering the best available trade-off. All four levels remain Pareto-optimal.

4.3 Domain Analysis

Domain risk multipliers range from 0.7 (general browsing) to 1.8 (banking). With noisy perception, contextual policies in banking achieve 47.9% safety recall and 83.2% utility, while general browsing achieves 51.3% safety recall and 89.7% utility. Injection detection

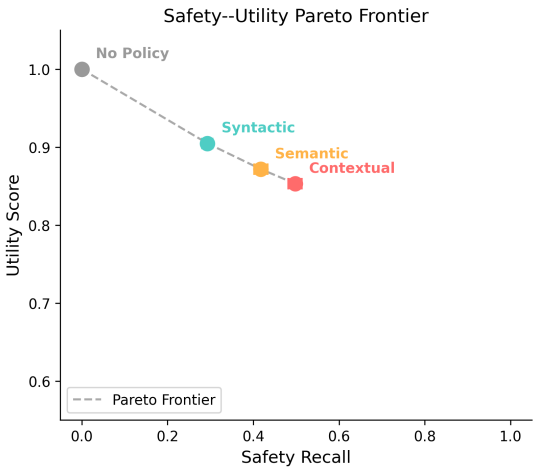

Figure 1: Safety–utility Pareto frontier under noisy perception. Contextual policies offer the best available trade-off point.

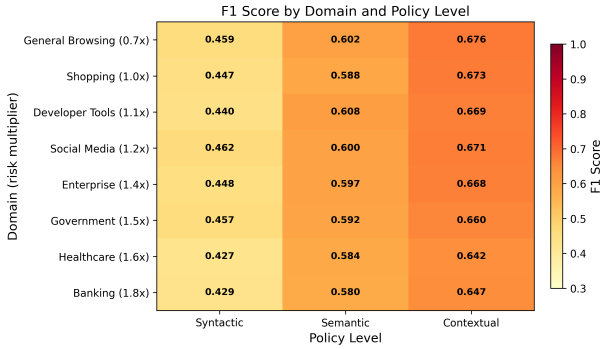

Figure 2: F1 score by domain and policy level. Contextual policies dominate across all domains, with the largest absolute gains in high-risk domains.

shows strong domain variation: government (60.7%) and shopping (57.6%) exceed the overall average, while general browsing (49.7%) lags due to lower base taint rates. Figure 2 visualizes F1 scores across domains and policy levels.

4.4 Scalability Analysis

As task length increases from 3 to 50 actions, contextual F1 remains stable while syntactic and semantic F1 show gradual improvement but remain well below contextual performance. Figure 3 confirms that the contextual advantage persists across task complexities.

4.5 Injection Robustness

Across injection rates from 5% to 70%, contextual policies maintain detection rates between 45–56%, far exceeding syntactic (12–17%) and semantic (19–28%) baselines. Figure 4 shows that detection performance is stable across attack intensities, validating the robustness of taint-based rules.

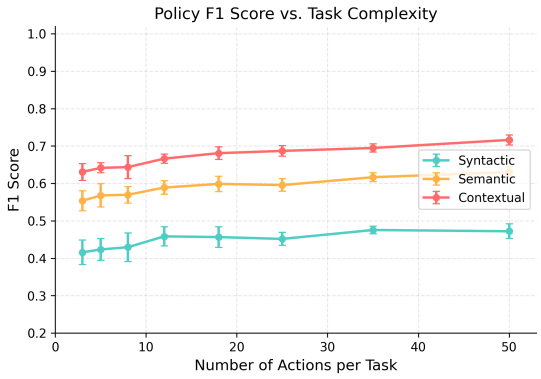

Figure 3: F1 score vs. task complexity (number of actions per task).

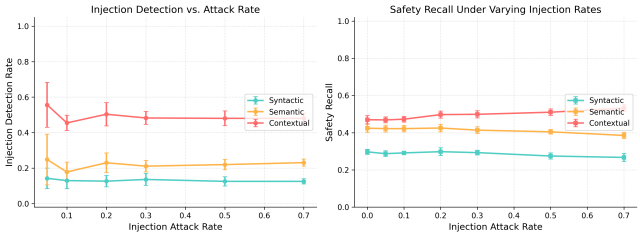

Figure 4: Left: injection detection rate vs. attack rate. Right: safety recall under varying injection rates. Contextual policies maintain consistent advantages.

Table 2: Ablation study: removing components from the contextual policy. "Full" is the complete contextual policy.

Ablation	F1	Utility	Inj. Det.	Task Compl.
Full contextual	0.660	0.859	0.543	0.411
No taint rules	0.579	0.880	0.300	0.463
No deceptive penalty	0.653	0.858	0.464	0.410
Semantic confusion	0.659	0.858	0.529	0.409
No review (flag=block)	0.770	0.817	0.560	0.330

4.6 Ablation Study

Table 2 decomposes the contextual policy's performance. Removing taint-based rules reduces injection detection from 54.3% to 30.0% (a 44.8% relative drop), confirming that taint propagation is the primary driver of injection detection. Removing the deceptive-action penalty reduces injection detection by 14.6%. Downgrading the risk confusion model to semantic-level has minimal impact (F1: 0.659 vs. 0.660), suggesting that taint rules matter more than perception accuracy at this level. Disabling the review mechanism (all flags become blocks) increases F1 to 0.770 by converting flags to hard blocks, but reduces utility from 85.9% to 81.7% and task completion from 41.1% to 33.0%.

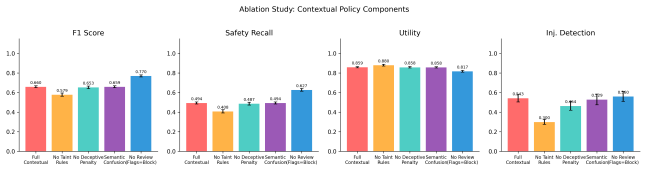

Figure 5: Ablation study: impact of removing individual components from the contextual policy on F1, safety recall, utility, and injection detection.

5 DISCUSSION

Impact of realistic perception. The most significant change from the prior oracle-based model is the drop in safety recall: contextual policies achieve 49.8% vs. the oracle's 94.7%. This reflects the fundamental difficulty of risk classification from noisy observations and demonstrates why oracle-based evaluations can be misleading. Precision remains high (>0.99), indicating that policies are conservative—when they flag or block, they are almost always correct.

Three-way enforcement. The review mechanism provides a principled middle ground between permissive (allow-all) and restrictive (block-all) policies. With a budget of 5 reviews per task, contextual policies achieve 85.3% utility while maintaining 49.8% safety recall. Without review (ablation: flags become blocks), utility drops to 81.7% and task completion falls from 41.1% to 33.0%, confirming that user-in-the-loop review substantially improves the safety–utility trade-off.

Taint propagation. The ablation study confirms that taint-based rules are the primary driver of injection detection: removing them reduces detection from 54.3% to 30.0%. This validates the review's recommendation to model data-flow constraints explicitly rather than relying on global risk thresholds. The taint model, while simple (forward propagation from reads to sinks), captures the core insight that injection attacks exploit the flow of untrusted data into sensitive actions.

Limitations. Several important limitations remain: (1) the taint model uses simple forward propagation without tracking specific data flows—a real system would need field-level taint tracking; (2) confusion matrices are calibrated rather than learned from real classifiers; (3) the simulation does not model correlated failures or adversarial evasion beyond simple injection; (4) the review budget is static rather than adaptive to task context; (5) real-world CUA distributions may differ from our synthetic tasks.

6 RELATED WORK

Information-flow control has a long history [1, 4, 6]. Dynamic taint analysis [7] provides the foundation for our taint propagation model. The Dual-LLM architecture [2] introduced control-flow isolation for CUA agents but left data-flow policies as an open problem. Prompt injection attacks [3, 9] pose particular threats to CUA systems. Tool emulation environments [5] and OS benchmarks [8] have evaluated agent capabilities but not security policy enforcement under realistic perception noise. Our work bridges this gap with taint-aware policies evaluated under noisy confusion models.

7 CONCLUSION

We presented a revised framework for semantic policy enforcement in CUAs that addresses key limitations of oracle-based evaluation: noisy risk perception, three-way enforcement with review budgets, taint-based injection detection, and fair cross-policy comparison. Our experiments show that contextual taint-aware policies achieve an F1 of 0.664 with 53.2% injection detection and 85.3% utility—modest but realistic gains over syntactic baselines. Ablation analysis identifies taint propagation as the single most important component for injection detection. Future work should integrate learned risk classifiers, field-level taint tracking, and adaptive review budgets to further close the gap between simulation and deployment.

REFERENCES

[1] Dorothy E Denning. 1976. A lattice model of secure information flow. *Commun. ACM* 19, 5 (1976), 236–243.

[2] Jacob Foerster et al. 2026. CaMeLs Can Use Computers Too: System-level Security for Computer Use Agents. In *arXiv preprint arXiv:2601.09923*.

[3] Kai Greshake et al. 2023. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. *arXiv preprint arXiv:2302.12173* (2023).

[4] Andrew C Myers. 1999. JFlow: Practical mostly-static information flow control. *ACM SIGPLAN Notices* 34, 1 (1999), 228–241.

[5] Yangjun Ruan et al. 2024. ToolEmu: Identifying the risks of LM agents with an emulated tool execution environment. *arXiv preprint arXiv:2309.15817* (2024).

[6] Andrei Sabelfeld and Andrew C Myers. 2003. Language-based information-flow security. In *IEEE Journal on Selected Areas in Communications*, Vol. 21. 5–19.

[7] Edward J Schwartz, Thanassis Avgerinos, and David Brumley. 2010. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *IEEE Symposium on Security and Privacy*. 317–331.

[8] Tianbao Wu et al. 2024. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972* (2024).

[9] Yueqi Xie et al. 2023. Defending ChatGPT against jailbreak attack via self-reminders. *Nature Machine Intelligence* 5 (2023), 1486–1496.

[10] Yifeng Zheng et al. 2024. Agent security: A survey of threats and defenses. *arXiv preprint arXiv:2401.16277* (2024).