
Bugs over Backdoors: Efficient and Evasive Sabotage via Plausible Code Bugs in LinuxArena¹

Timothy Obiso*
Brandeis University

Dean Cahill*
Independent Researcher

With
Apart Research

Abstract

AI control attack policies vary widely in efficacy and computational cost. We design and evaluate a bug-based attack policy for LinuxArena, wherein an untrusted agent introduces plausible code defects rather than obfuscated backdoors as its primary sabotage vector. Across nine samples, bug-attack achieves an 89% side-task success rate, exceeding the 78% of the standard obfuscation baseline, while using 14.5x fewer tokens. Notably, bug-attack and standard attack exhibit opposite vulnerability profiles under trusted-editing defense: trusted editing catches 37% of bug-attack sabotages while catching none of standard attack's, suggesting the two are complementary rather than substitutes. We also report single-sample results for two exploratory policies. We treat all findings as preliminary given sample size constraints, but consider the efficiency and evasion results directionally robust.

1. Introduction

The rapid rise of agentic systems in consumer contexts leads to an abundance of security concerns, and the proliferation of these systems does not appear to be slowing. As such, it is important to develop tools for testing the capabilities of these systems in interaction with each other. Various

¹ Research conducted at the [AI Control Hackathon](#), March 2026

* Indicates equal contribution

agent deployment environments have been proposed and developed in order to provide contexts for research on AI control and security; however, since this is the bleeding-edge, these tools are still actively being developed. LinuxArena, the most comprehensive and consumer-oriented of these environments, currently lacks a good deal of policy & protocol implementations: two monitor configurations, and two attack policies, both based on code obfuscation.

We introduce a bug-based attack policy to LinuxArena which uses 14.5x fewer tokens than the obfuscation baseline, with an 11% increase in side-task success. We also provide results for two exploratory protocols which follow this “bugs over backdoors” paradigm. For robustness, we supplement the default multi-monitor ensemble with a trusted-editing protocol implemented for monitors within LinuxArena, and compare their results across our attack policies.

2. Related Work

LinuxArena evaluates whether control protocols can prevent AI agents from accomplishing covert sabotage objectives. This follows the control evaluation framework from Greenblatt et al. (2024), which proposed using untrusted models as red-team proxies to stress-test safety protocols. Our work extends the attack side of this framework by introducing attack paradigms beyond code obfuscation.

Existing attack policies use a similar core approach: write malicious code, then obfuscate it. They differ in planning complexity, targets, and other specifics like execution but share the same assumptions: that there is an attack that should be hidden. We challenge this assumption with bug-attack. The attack does not need to be hidden if it does not look like an attack.

3. Methods

a. Red Team: Bug Attack Policy

Once we knew that the bug-attack was a valid attack trajectory, we adapted it into a full attack policy. Our attack policy uses a 3-phase architecture (explore → plan → attack), wherein the model (1) explores the environment, (2) uses a single generate call to produce a plausibly relevant bug, and (3) executes the attack by implementing the main task. This is much more constrained than the default attack policy’s 7 phase architecture, which factors into the performance gains.

b. Red Team: Exploratory Policies (Data-Flow and Targeted Camouflage)

In addition to our primary policy, we developed two policies aiming to explore monitor capabilities, but were only able to test individual samples of these (due both to time constraints and a desire to focus on our main policy). These consist of a data-flow decomposition policy and a targeted camouflage policy.

The data-flow decomposition policy is based around the idea that typical monitors only observe individual actions. As such, a sequence of benign actions which compose into

an attack will bypass these monitors. We wanted to see if such an approach would be applicable to the default ensemble on LinuxArena. This is a canonical approach to attack policy design, and being able to benchmark the monitors against this policy is important.

The targeted camouflage policy is similar in spirit to the baseline attack, but differs in a key way: the agent writes the operations of the side-task *in terms of the main task*. This means that the obfuscation is semantically closer to the main task, and therefore more likely to bypass an action monitor. While this seems like an obvious solution, the presence of an orthogonal monitor heavily dampens the strength of the attack vector.

c. Blue Team: Trusted Editing Monitor

We implement the trusted-editing monitor protocol as a blue-team policy. This policy wraps an existing attack policy, runs it, and reviews & rewrites the output. We developed two versions of this protocol, but failed to adequately test the second iteration in time.

4. Results

All experiments use the bug_attack_comparison task set: shared_dotfiles/appropriate_secrets, web_scraping/visit_malicious_website, clinical_trial/suppress_warnings. We ran for three epochs, for nine samples per policy. The untrusted model is claude-haiku-4-5. Monitoring uses the multi monitor with gpt-4.1-nano for both sub-monitors.

Policy	Side Task Success	Avg Tokens
honest	0/9	60,961
bug-attack	8/9	35,479
attack	7/9	510,350

Table 1: Side task success rates across 9 sample (3 environments x 3 epochs) using the bug_attack_comparison task set. Honest baseline shows no accidental side task completion. bug-attack is more successful while using ~14.5x less tokens

As shown in **Table 1**, bug-attack achieves a higher task success rate while using significantly fewer tokens, showcasing its effectiveness and conciseness. **Table 2** shows that despite not spending any phases (or tokens) on obfuscation planning, bug-attack evades the monitor at the same rate as standard attack in our task set. 5/9 samples for each are below the suspicion threshold. Mean per-action suspicion scores are similar across all three policies. Interestingly, the honest baseline exceeded the suspicion threshold in 2 of its 9 samples, scoring a 7.0 on some action. This likely reflects monitor miscalibration on inherently suspicious-looking legitimate actions rather than accidental sabotage.

Environment	Side Task	bug-attack Success	attack Success
-------------	-----------	--------------------	----------------

shared_dotfiles	appropriate_secrets	3/3	2/3
web_scraping	visit_malicious_website	2/3	2/3
clinical_trial	suppress_warnings	3/3	3/3

Table 2: Side task success by environment. Same setup as Table 1. bug-attack matches or exceeds attack in all environments.

Policy	Avg Max Score	Samples ≥ 5.0	Mean Per-Action Score
honest	4.61	78%	1.33
bug-attack	6.11	56%	1.61
attack	5.39	56%	1.47

Table 3: Monitor evasion rates using the multi monitor (gpt-4.1-nano for both orthogonal and action sub-monitors). Scores range 0-10; ≤ 5.0 is the evasion threshold.

We also experimented with a trusted-editing protocol with gpt-4.1-nano. The two attack policies show opposite vulnerability profiles. Trusted editing catches 3/8 bug-attack sabotages but none of the attack sabotages. While bug-attack produces the highest average max suspicion score across all policies, it evades the suspicion threshold at the same rate as attack. This suggests that bug-attack does not reduce peak suspicion so much as it produces a different suspicion profile with occasional high spikes on individual actions not affecting overall evasion. We expect to see increased detection and evasion abilities in stronger models.

Inner Policy	No Defense Success	Trusted Editing Success	Attacks Caught
bug-attack	89% (8/9)	56% (5/9)	3/8 (37%)
attack	78% (7/9)	78% (7/9)	0/7 (0%)

Table 4: Effect of the trusted-editing (v1) defense protocol on side task success. The trusted model is gpt-4.1-nano and the untrusted model is claude-haiku-4-5. The same task set as Table 1 is used.

Environment	Side Task	TE + bug-attack	TE + attack
shared_dotfiles	appropriate_secrets	2/3 (-1)	2/3 (no change)
web_scraping	visit_malicious_website	1/3 (-1)	2/3 (no change)
clinical_trial	suppress_warnings	2/3 (-1)	3/3 (no change)

Table 5: Per-environment side task success with trusted editing, 3 epochs each, using the same task set as in Table 1.

Both exploratory policies succeeded on web_scraping but failed on model_registry. Among the successful runs, camouflage-attack achieved lower monitor scores (max 5.0) while data-flow-attack scored much higher (max of 9.5 for exfil_creds side task).

Policy	Environment	Side Task	Pass	Max Score	Mean Score
data-flow-attack	model_registry	exfil_creds	NO	9.5	1.80
data-flow-attack	web_scraping	log_creds	YES	7.5	1.39
data-flow-attack	web_scraping	visit_malicious_website	YES	5.5	1.19
camouflage-attack	web_scraping	log_creds	YES	5.0	1.26
camouflage-attack	model_registry	privilege_escalation	NO	8.0	1.60
camouflage-attack	web_scraping	visit_malicious_website	YES	5.0	1.14
camouflage-attack	model_registry	data_export	NO	9.0	1.48

Table 6: Single-sample evaluations of data-flow-attack and camouflage-attack. All use claude-haiku-4-5. Monitor scores from multi monitor (gpt-4.1-nano).

5. Discussion and Limitations

Our initial assumption was that the use of bugs as an attack vector would avoid the suspicion latent in obfuscated sabotage, by producing errors that will slip under the radar of a monitor. As we see in our results, this theory holds. We often see either an increase, or only a marginal decrease, in the performance of the attack policy, while also seeing massive decreases in runtime & compute cost. As such, we consider this to be a (partially) optimized subset of the default attack policy for certain tasks; we constrain the attack agent’s planning space by isolating those lines of attack which manifest as simple defects in poorly constructed code, rather than having the agent search the entire space to find these (or bypass them for a less effective method). Though untested, we assume this policy scales up with the size of the codebase we’re attacking, as unnoticed bugs tend to balloon in these contexts, and a bug that serves the ends of the attacker will not only be lost in a sea of code, but in waves of other, incidental bugs.

Limitations

Our primary limitation is the sample size of our evaluations. Due to time constraints, we only evaluated 9 samples per policy (and only 1 sample per exploratory policy). Additionally, our monitors were quite weak (gpt4.1-nano). While this is expected (a trusted monitor is assumed weaker than the untrusted model within the theoretical formulation of AI control), exploring stronger monitors would highlight flaws in the policy. Finally, many alternative BT protocols exist beyond the two tested here, and these may contain particular approaches that fully thwart our attack methodology.

Future Work

The immediate next step is a comprehensive evaluation of this policy across multiple main-task/side-task combinations and across multiple environments. We see no reason why the bug-driven attack policy shouldn't generalize to any environment with an editable codebase, and verifying this empirically seems like a natural progression for validating the policy. We would also evaluate with more (powerful) models.

6. Conclusion

Our results suggest that small defects in agent code are a valuable vector for certain forms of malicious attacks. These attacks are just as (if not more) capable of evading monitor audits than obfuscation, while vastly outperforming the latter in efficiency. This makes a bug-driven approach more well-suited for the initial probing phases of a malicious agent's broader attack trajectory, and as such it may be a good idea to introduce early stage protocols to squash these particular methods.

Code and Data

- **Code repository:** <https://github.com/timothyobiso/linuxarena>

References

Ryan Greenblatt, Buck Shlegeris, Kshitij Sachan, and Fabien Roger. 2023. Ai control: Improving safety despite intentional subversion. arXiv preprint arXiv:2312.06942.

LLM Usage Statement

We used Claude to refine policy architectures, as well as to orchestrate evaluation runs.