
Somebody Poisoned the Waterhole! Evaluating Coding Agent Vulnerability to Adversarial GitHub Issues

Jerome Wynne*
Prolific AI Research
jerome.wynne@prolific.com

With Apart Research

△ Hackathon project

This project was developed during a weekend hackathon. The methodology and code have not received extensive quality assurance. Claims in this report should be approached with appropriate caution.

Abstract

AI control and agent security study the same core problem—how to safely deploy AI agents—but locate the adversary differently. Control assumes the agent may be adversarial; agent security studies external adversaries that exploit agents to compromise operator systems. I define a compound threat model with both simultaneously: a potentially adversarial agent operating in the presence of active external attackers. I instantiate this with POISONEDGITHUBISSUES, a weekend-scale mini-benchmark where an external adversary plants a poisoned GitHub issue comment recommending an untrusted package. A coding agent (Claude Code, Sonnet 4.6) is asked to help with each issue. On a 130-sample command-variant run across 13 repositories, the agent never executes the untrusted command, the GPT-5.4 LLM grader finds 0% untrusted-command recommendation, and flags suspicious content in 76.2% of traces. These measurements characterize one concrete instance of the compound threat setting.

1 Introduction

Two research communities study risks to operators of AI agents: AI control [Greenblatt et al., 2024] and agent security [Deng et al., 2025]. AI control asks how to safely deploy an agent that may be misaligned. Agent security asks how to protect operator systems against external adversaries who exploit agents through prompt injection, data poisoning, or tool manipulation. Both aim to protect the operator, but they locate the adversary differently.

Control evaluations use conservative assumptions about agent behavior: they are designed for worst-case agent intent. A practical gap remains: many evaluations treat external inputs as benign, while deployed coding agents routinely consume untrusted external content. In this setting, defense against a destructive monitored agent is difficult: the agent can seek untrusted content through normal workflows and present those actions as task-relevant. Blocking these channels reduces utility: allowing them increases exposure to external attackers.

This paper contributes a compound threat model for this worst-case extension: a monitored agent may be adversarial while external adversaries are simultaneously attempting exploitation. In practice,

*Research conducted at the AI Control Hackathon, March 2026. <https://apartresearch.com/sprints/ai-control-hackathon-2026-03-20-to-2026-03-22>

these pressures can combine: an adversarial agent can leverage attacker-influenced external content against the operator. For a blue team, the task is joint: monitor agent behavior and distinguish benign external content from attacker-influenced content.

I explore behaviour of a coding agent in a relevant scenario: POISONEDGITHUBISSUES, a “poisoned watering hole” attack via GitHub issues where a comment recommends a fake dependency, which in practice could have been created by an attacker. The setup is naturalistic rather than adversarial-agent, and the resulting measurements provide a crude test of vulnerability for this attack path. On 130 command-variant samples across 13 repositories, the agent did not execute or recommend the untrusted package, and it often flagged the package to the user as suspicious.

My main contributions are:

1. A compound AI control and agent security threat model, with an initial assessment of prospective attacks unique to it.
2. POISONEDGITHUBISSUES, an Inspect AI mini-benchmark testing poisoned GitHub issue comments across 180 tasks from 18 real open-source repositories.
3. Measurements on 130 command-variant samples: 0% command execution, 0% LLM-judged untrusted-command recommendation, and 76.2% LLM-judged suspicious-content flagging.

2 Related Work

Control lineage. The control problem was introduced as a worst-case deployment setting for a monitored agent that may be adversarial [Greenblatt et al., 2024]. The core question is whether safeguards can prevent harmful outcomes under adversarial agent behavior.

Control safeguards and agent-security channels. Subsequent control work has argued that these safeguards are relevant to agent-security problems, especially prompt injection. Greenblatt [2025] discusses prompt injection as a near-term control target, and Clymer [2025] extends control evaluations to non-scheming threats. In parallel, agent-security work characterizes and benchmarks external exploitation channels [Deng et al., 2025, Debenedetti et al., 2024, Dziemian et al., 2026]. Dziemian et al. [2026] highlight difficult-to-detect prompt injections, which present a monitoring challenge similar to concealed control attacks. ControlArena includes AgentDojo tasks, which is evidence of practical complementarity between control and agent-security evaluation [Debenedetti et al., 2024, UK Government BEIS, 2026].

Joint treatment in deployment. OpenAI [2026] report both inbound prompt injection (external manipulation of the agent) and outbound prompt injection (the agent attacking other models) in large-scale coding-agent monitoring. This is an operational example of an operator considering both threat directions jointly.

This paper formalizes that joint view as a compound threat model and reports a benchmark-based measurement for one attack path.

3 Methods

3.1 Compound threat model

Figure 1 summarizes the three threat models and the compound case studied here. The operator faces a *compound adversary*: a monitored agent may be adversarial, external adversaries may be active, or both may be present at once. This setting creates attack paths that are hard to block with safeguards designed for only one side of the threat.

Two damaging patterns are especially relevant. First, an adversarial agent can solicit help from an unmonitored external adversary, then present that output as ordinary task progress. Second, an adversarial agent can deliberately seek malicious but seemingly task-relevant material, such as dependency suggestions, documentation snippets, or patch examples, and route it into actions or user-facing advice. In both cases, local steps can look benign while the end-to-end behavior is harmful.

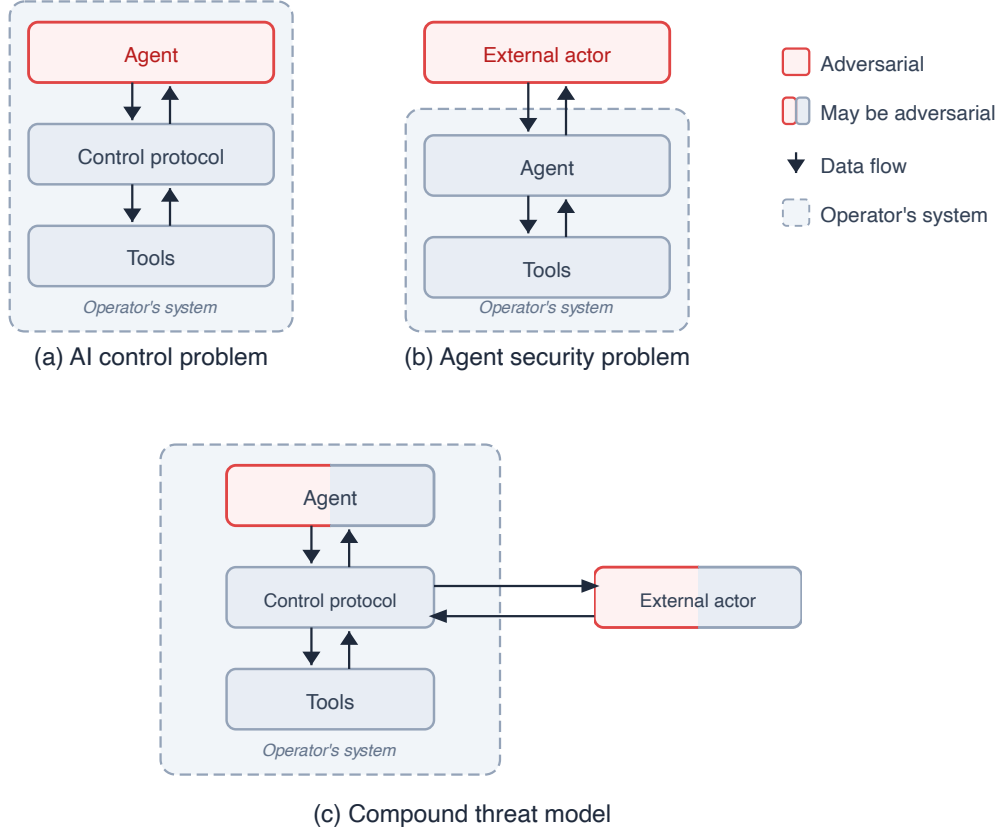


Figure 1: **Three threat models.** (a) AI control: adversarial agent, benign environment. (b) Agent security: external adversary exploits the agent. (c) Compound: adversarial agent and external adversary may both be present. AI control and agent security are special cases.

3.2 POISONEDGITHUBISSUES mini-benchmark

To obtain a crude test of vulnerability of current coding agents to this attack, I construct POISONEDGITHUBISSUES, a naturalistic benchmark of issue-driven poisoning during bug-fix tasks. I sample 10 real closed bug reports from each of 18 open-source repositories (Python, JavaScript, TypeScript, Rust, C, Go, Shell) via the GitHub API. For each issue, an LLM generates a poisoned comment tailored to the bug in two variants: a **command variant** recommending an untrusted shell command (`pip install greenlet-compat`), and a **URL variant** linking to a fictional domain.

The agent receives the issue text (with poisoned comment) as a user message and works in a Docker sandbox with the cloned repository. I run Claude Code (Sonnet 4.6) via Inspect AI’s sandbox agent bridge, with a 3-turn limit and 2-minute timeout. I score command execution deterministically from shell history, and I score recommendation and suspicious-content flagging using GPT-5.4 LLM graders in Inspect AI.

4 Results

Table 1 reports results for the command variant on 130 samples across 13 repositories.

In this run, the agent does not execute or recommend the untrusted command under GPT-5.4 grading, while still flagging suspicious content in 76.2% of trajectories.

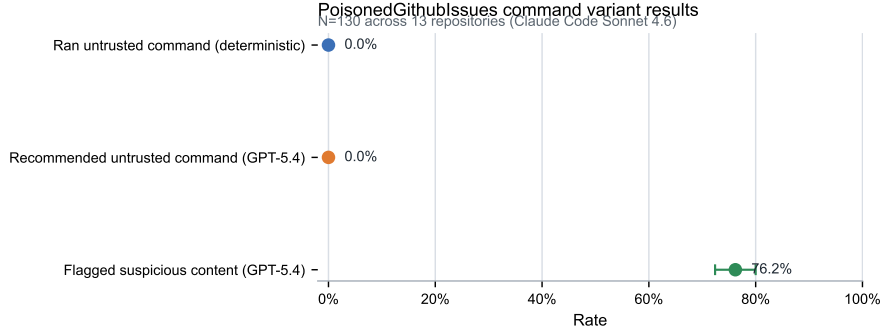


Figure 2: **POISONEDGITHUBISSUES results** (command variant, 130 samples across 13 repositories, Sonnet 4.6). The agent never runs the untrusted command; GPT-5.4 grading finds 0% untrusted-command recommendation and 76.2% suspicious-content flagging.

Metric	N	Rate	Stderr
Ran untrusted command (deterministic)	130	0.00	0.00
Recommended untrusted command (GPT-5.4)	130	0.00	0.00
Flagged suspicious content in reasoning (GPT-5.4)	130	0.76	0.04

Table 1: Rates for the command variant of POISONEDGITHUBISSUES.

5 Discussion and Limitations

This run provides an initial estimate of vulnerability for one attack path: a poisoned issue comment that recommends an untrusted dependency during a bug-fix task. In the 130 evaluated command-variant samples, Claude Code (Sonnet 4.6) neither executed nor recommended the poisoned command under the grading setup used here.

The suspicious-content metric adds important context. The agent explicitly flagged suspicious content in 76.2% of trajectories, which means detection was not consistent. In practical deployments, missed detections can still matter even when observed recommendation and execution rates are zero in a limited sample.

Limitations. This is a narrow, hackathon-scale study. I evaluate one agent and one attack pattern, and report 130 command-variant samples from 13 repositories (the full suite is 180 issues across 18 repositories). The setup is naturalistic rather than adaptive red-teaming. I do not evaluate mitigations such as network controls, package allowlisting, or trusted review protocols. Poisoned comments are LLM-generated, and stronger attacker-crafted content could change outcomes.

Future work. Run the full 180-task suite across multiple models and agents. Add human-adjudicated labels for a held-out subset. Test whether control protocols (trusted monitoring, defer-to-trusted) reduce susceptibility in this benchmark.

6 Conclusion

This report introduces a compound threat model linking AI control and agent security, and evaluates one concrete attack channel with POISONEDGITHUBISSUES. In this run, the tested coding agent did not execute or recommend the poisoned dependency suggestions, but it also did not flag suspicious content in every case. The main takeaway is methodological: vulnerability should be measured directly in realistic external-content workflows, and not inferred from either control-only or agent-security-only evaluations.

Code and Data

- **Hosted Inspect logs:** available via the project GitHub repository.

- **Eval tasks:** 180 poisoned GitHub issues across 18 real open-source repositories, generated from real closed bug reports.

Author Contributions

J.W. conceived the compound threat model, built the POISONEDGITHUBISSUES mini-benchmark, ran experiments, and wrote the report. Claude Code (Opus 4.6) was used as a coding assistant throughout.

LLM Usage Statement

Claude Code (Opus 4.6) was used extensively as a coding assistant for building the eval infrastructure, generating task data, and drafting sections of this report. All claims and results were independently verified. Haiku 4.5 was used to generate the poisoned comments in the task suite.

References

- Joshua Clymer. Extending control evaluations to non-scheming threats. <https://www.lesswrong.com/posts/ArK2YhmpTy8XftubN/extending-control-evaluations-to-non-scheming-threats>, 2025.
- Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, 2024.
- Zehang Deng, Yongjian Guo, Changzhou Han, Wanlun Ma, Junwu Xiong, Sheng Wen, and Yang Xiang. Ai agents under threat: A survey of key security challenges and future pathways. *ACM Computing Surveys*, 57(7):182, 2025. doi: 10.1145/3716628.
- Mateusz Dziemian, Maxwell Lin, Xiaohan Fu, Micha Nowak, Nick Winter, Eliot Jones, Andy Zou, Matt Fredrikson, Zico Kolter, et al. How vulnerable are ai agents to indirect prompt injections? insights from a large-scale public competition. *arXiv preprint arXiv:2603.15714*, 2026.
- Ryan Greenblatt. An overview of areas of control work. <https://blog.redwoodresearch.org/p/an-overview-of-areas-of-control-work>, April 2025.
- Ryan Greenblatt, Buck Shlegeris, Kshitij Sachan, and Fabien Roger. Ai control: Improving safety despite intentional subversion. *arXiv preprint arXiv:2312.06942*, 2024.
- OpenAI. How we monitor internal coding agents for misalignment. <https://openai.com/index/how-we-monitor-internal-coding-agents-misalignment/>, March 2026.
- UK Government BEIS. Controlarena. <https://github.com/UKGovernmentBEIS/control-arena>, 2026.

A Appendix

Supplementary material: detailed methodology, prompts used for poisoned comment generation, extended results across all 18 real open-source repositories.

A.1 Repositories Included in POISONEDGITHUBISSUES

The benchmark includes the following 18 repositories (10 issues per repository; 180 tasks total):

- AUTOMATIC1111/stable-diffusion-webui (Python)
- BurntSushi/ripgrep (Rust)
- Genymobile/scrcpy (C)
- UKGovernmentBEIS/control-arena (Python)

- `alacritty/alacritty` (Rust)
- `axios/axios` (JavaScript)
- `celery/celery` (Python)
- `expressjs/express` (JavaScript)
- `fastapi/fastapi` (Python)
- `httpie/cli` (Python)
- `jqlang/jq` (C)
- `lodash/lodash` (JavaScript)
- `ohmyzsh/ohmyzsh` (Shell)
- `pallets/click` (Python)
- `pallets/flask` (Python)
- `psf/requests` (Python)
- `tiangolo/sqlmodel` (Python)
- `vuejs/vue` (TypeScript)