

TaintGuard: Augmenting LLM Monitors with Formal Data-Flow Taint Tracking for AI Agent Safety

Revathi Prasad

AI Control Hackathon, March 2026

Abstract

We present TaintGuard, a system that augments existing LLM-based monitors with dynamic data-flow taint tracking to detect sabotage in AI agent trajectories. Taint tracking provides a structurally different signal from LLM monitoring — it detects data-flow violations (exfiltration, integrity writes, code injection) that are provable and immune to prompt injection, while LLM monitors detect semantic intent. On 106 LinuxArena sabotage trajectories, we show two configurations: Case 1 adding 2.8pp recall over LinuxArena's monitor with 4.3% FPR, and Case 2 adding 26pp recall with 43.5% FPR via T-model authorization checking. The precision-recall tradeoff reflects a fundamental challenge confirmed by prior work: honest development and sabotage involve similar code modifications, and distinguishing them requires trajectory-level intent assessment. We propose taint-finding-aware validation as a future pathway to reduce FPR. We also demonstrate the Ctrl-Z Defer-to-Resample protocol (Bhatt et al. 2025) for uncertain detections.

1. Introduction

LLM-based monitors are the standard approach for detecting sabotage in AI agent deployments. However, they are vulnerable to prompt injection, provide no formal guarantees, and require an API call per action. Recent work confirms that "it's very difficult for monitors to distinguish between well-crafted attacks and some of the gnarlier main tasks" (BashArena, Redwood Research 2025).

We propose augmenting LLM monitors with a structurally different signal — formal data-flow taint tracking. Taint tracking detects violations by tracing data flows from sensitive sources to network sinks, immune to how commands are framed or explained. The two signals are complementary: taint catches data-flow violations invisible to semantic analysis, while LLM monitors catch intent-level sabotage invisible to structural analysis.

Our main contributions are:

1. **Contribution 1:** A formal data-flow taint tracker for AI agent trajectories that detects exfiltration, integrity violations, and code injection across three analysis planes (bash commands, text_editor payloads, inline Python).
2. **Contribution 2:** An empirical analysis of the precision-recall tradeoff when combining taint tracking with LLM monitoring, showing taint adds 2.8–26pp recall depending on configuration, with corresponding FPR implications.
3. **Contribution 3:** A demonstration of the Ctrl-Z Defer-to-Resample protocol (Bhatt et al. 2025) as an intervention mechanism, catching 2/5 frontier-attacker attempts with a cheap monitor.

2. Related Work

AI Control. Greenblatt et al. (2024) formalize the AI control problem. ControlArena and LinuxArena provide evaluation infrastructure. BashArena (Redwood Research 2025) demonstrates that monitoring effectiveness depends heavily on trusted model capability, with 74% safety at 4% trajectory-wise FPR.

Constitutional Monitoring. Constitutional Black-Box Monitoring (arXiv:2603.00829) achieves 0.85 pAUROC using prompted classifiers optimized on synthetic data. Their finding that "performance saturates fast" suggests the ceiling of pure LLM monitoring.

Taint Analysis. ABASH (Mazurak & Zdancewic 2007) applies information flow analysis to bash scripts. We adapt this for AI agent trajectories with cross-step propagation and integrity detection (Biba 1977).

Ctrl-Z Resampling. Bhatt et al. (2025) reduce attack success from 58% to 7% at 5% usefulness cost by resampling the untrusted agent independently.

3. Methods

Architecture

The TaintGuard pipeline routes each command through two layers:

```
Command → Layer 1: Taint Tracking
    ├── Tier 1: Fixed rules (MITRE ATT&CK T1552)
    ├── Tier 2: T-model path/authorization labeler
    ├── Propagation: cross-step source→sink flow detection
    ├── taint fires → DETECTED
    └── taint silent → Layer 2: Platform LLM Monitor
        └── score ≥ threshold → DETECTED
```

Layer 1: Taint Tracking

Extracts structured semantics from each action (files read/written, network destinations, pipe chains, resource modifications) and propagates taint labels across steps. Three analysis planes: bash command parsing (Plane 1), Python AST analysis of text_editor payloads (Plane 2), and inline Python python3 -c analysis (Plane 3).

Detection types:

-
- **Confidentiality flows:** sensitive source → network sink (Bell-LaPadula model)
- **Integrity writes:** writes to system auth/exec files — sudoers, authorized_keys, cron (Biba model)
- **Code injection:** text_editor creates/modifies code containing AST-detected network sends
- **Integrity modification:** unauthorized DB/API state changes (with T-model authorization for app_code)

Layer 2: Platform LLM Monitor

LinuxArena's multi monitor (ensemble of two gpt-4.1-nano, 0–5 scale). TaintGuard adds taint as a formal layer on top of whatever monitor the platform provides.

Ctrl-Z Defer-to-Resample (DTR)

For uncertain actions, the DTR protocol (Bhatt et al. 2025) discards the action and asks the agent to try again independently. After n resamples, execute the action with the lowest suspicion score. Each resample is scored by the full TaintGuard pipeline.

Evaluation

Post-hoc on 106 LinuxArena sabotage trajectories (all confirmed successful attacks). LinuxArena's embedded per-action monitor scores (from their multi monitor, gpt-4.1-nano) serve as Layer 2 for the attack dataset. Taint FPR measured on 23 honest trajectories from LinuxArena's Haiku honest run. Note: the honest trajectories do not contain embedded monitor scores, so LinuxArena's FPR is unmeasured.

4. Results

Table 1: Two configurations — precision-recall tradeoff

Configuration	Taint Recall	Fused Recall	Taint FP (honest)
Case 1 (Tier 1 fixed rules + code injection)	6.6%	51.9%	1/23 (4.3%)
Case 2 (Case 1 + Tier 2 T-model authorization)	53.8%	75.5%	10/23 (43.5%)

Case 1 uses only high-confidence taint signals (system file integrity writes, AST-detected code injection, DB/API modifications). Case 2 extends Case 1 by adding T-model authorization checking for app_code modifications, recovering 50 additional taint detections but introducing false positives on honest code modifications. Note: LinuxArena's monitor FPR is unmeasured — the honest trajectories do not contain embedded monitor scores.

Table 2: Fused recall at different thresholds

LA Threshold	LinuxArena alone	Fused (taint + LA)	Taint adds
≥ 2.0	75.5%	78.3%	+2.8pp
≥ 3.0	49.1%	51.9%	+2.8pp
≥ 4.0	34.0%	36.8%	+2.8pp
≥ 5.0	8.5%	11.3%	+2.8pp

Taint's contribution is consistent across all thresholds — it catches attacks LinuxArena misses regardless of how permissive or strict the LLM threshold is.

What Taint Uniquely Catches

Taint detects 3 attacks that LinuxArena's monitor misses entirely:

- **withdrawal_misclassification** (integrity_modification on DB): LA scored 1.5
- **file_corruption** (code_injection): LA scored 2.0
- **exfiltrate_secrets** (integrity_modification on DB): LA scored 2.0

These are cases where LinuxArena's LLM monitor assessed low suspicion, but taint detected a formal data-flow violation.

Table 4: DTR demo (GPT-4.1 attacker, nano monitor)

Side Task	Attacker min-sus	Honest min-sus	Verdict
intercept_vim_files	8.0	0.0	CAUGHT
expose_secret	9.0	0.0	CAUGHT
export_medical_data	0.0	0.0	Evaded
log_credentials	0.0	0.0	Evaded
access_other_teams_media	0.0	1.0	Evaded

2/5 caught despite significant capability gap. Attackers consistently generate malicious commands across resamples; honest agents produce consistent benign commands.

5. Discussion and Limitations

The Precision-Recall Tradeoff

The gap between Case 1 (51.9%) and Case 2 (75.5%) reflects a fundamental challenge: honest software development and sabotage involve similar code modifications. BashArena's research confirms this — "main tasks require using permissions and resources that enable the side tasks." Case 2's 43% FPR arises because the T-model authorization check ("is this modification within task scope?") cannot reliably distinguish task-relevant modifications from sabotage when both involve modifying the same types of files.

Taint's Structural Value

Despite the modest recall improvement in Case 1, taint provides a qualitatively different signal: provable data-flow violations immune to prompt injection. As models become more capable at adversarial framing, structural detection becomes more valuable relative to semantic detection.

Limitations

- **Small honest dataset:** 23 trajectories limits FPR confidence. Honest trajectories lack embedded LinuxArena monitor scores, so only taint's FPR is measured. Larger honest dataset (806 trajectories) available for future validation.
- **Dataset-specific:** Results from LinuxArena only. Taint's pattern tables are configurable per environment.
- **Simulated DTR:** Demo uses GPT-4.1 as stand-in, not actual untrusted agent.

Future Work

- **Taint-finding-aware Validation:** Rather than asking the T-model whether a modification is authorized, reframe it as a validator of specific taint findings — "Is this detected data flow consistent with the task objective?" This could recover Case 2's recall while substantially reducing false positives.

- **ControlArena/BashArena evaluation:** Live agent evaluation with Docker sandboxes. RunPod deployment script included (setup_runpod.sh).
- **Honest trajectory evaluation:** Measure false positive rate on honest agent behavior at scale using the full 806-trajectory honest dataset.
- **Threshold calibration:** The --fused mode supports configurable thresholds. Calibration on a balanced dataset (attack + honest) would determine the optimal operating point.
- **LinuxArena native integration:** Register TaintGuard as a LinuxArena ActionMonitor via their function registry.
- **Live DTR deployment:** Integrate Ctrl-Z resampling into the monitor's decision loop, scoring each resample with the full TaintGuard pipeline (taint + LLM), and resampling the actual untrusted agent in isolated sandbox instances.
- **Early Warning (DTMC):** Learn a Discrete-Time Markov Chain from safe agent traces to compute $P(\text{reaching unsafe state} \mid \text{current state})$ at each step. This reachability probability would serve as a dynamic prior for the cascade, raising suspicion before a malicious action is fully committed. Based on Pro2Guard (arXiv:2508.00500) and Neider & Gavran (FMCAD 2018) for specification mining from traces. Architecture stub exists in src/early_warning/.

6. Conclusion

TaintGuard demonstrates that formal data-flow taint tracking provides a complementary signal to LLM-based monitoring for AI agent safety. Taint detections are unforgeable — based on provable data flows, not semantic judgment. In Case 1, taint adds 2.8pp recall with minimal FPR. Case 2 shows higher recall (75.5%) but reveals the fundamental challenge of distinguishing honest code modifications from sabotage — a challenge confirmed by prior work. We propose taint-finding-aware validation as a pathway to close this gap.

Code and Data

- **Code repository:** <https://github.com/revathi-prasad/TaintGuard>
- **Data/Datasets:** 106 LinuxArena sabotage trajectories, 23 honest trajectories
- **Other artifacts:** 269 unit tests, DTR demo, RunPod deployment script

References

- Greenblatt, R., et al. (2024). "AI Control: Improving Safety Despite Intentional Subversion." arXiv:2312.06942.
- Bhatt, A., et al. (2025). "Ctrl-Z: Controlling AI Agents via Resampling." arXiv:2504.10374.
- Constitutional Black-Box Monitoring (2025). arXiv:2603.00829.
- BashArena (2025). "A Control Setting for Highly Privileged AI Agents." arXiv:2512.15688.
- Trust or Escalate (ICLR 2025). Simulated Annotators. arXiv:2407.18370.
- Biba, K.J. (1977). Integrity considerations for secure computer systems. MITRE.
- Mazurak, S. & Zdancewic, S. (2007). "ABASH: Finding Bugs in Bash Scripts." PLAS.
- Pro2Guard (2025). Proactive detection via probabilistic reachability. arXiv:2508.00500.

LLM Usage Statement

Claude Code (Claude Opus 4.6) was used for implementation, research, and drafting. All results were generated by running evaluation scripts on real trajectory data. All claims and numbers were independently verified against evaluation output.