

Performative Subversion

Who Monitors the Monitor?

Max Pinelo¹ · Angel Tenorio¹ · Fernando Avalos² · Sebastian Urrea² · Valeria Mora²

¹AIS México ²AIS Colombia

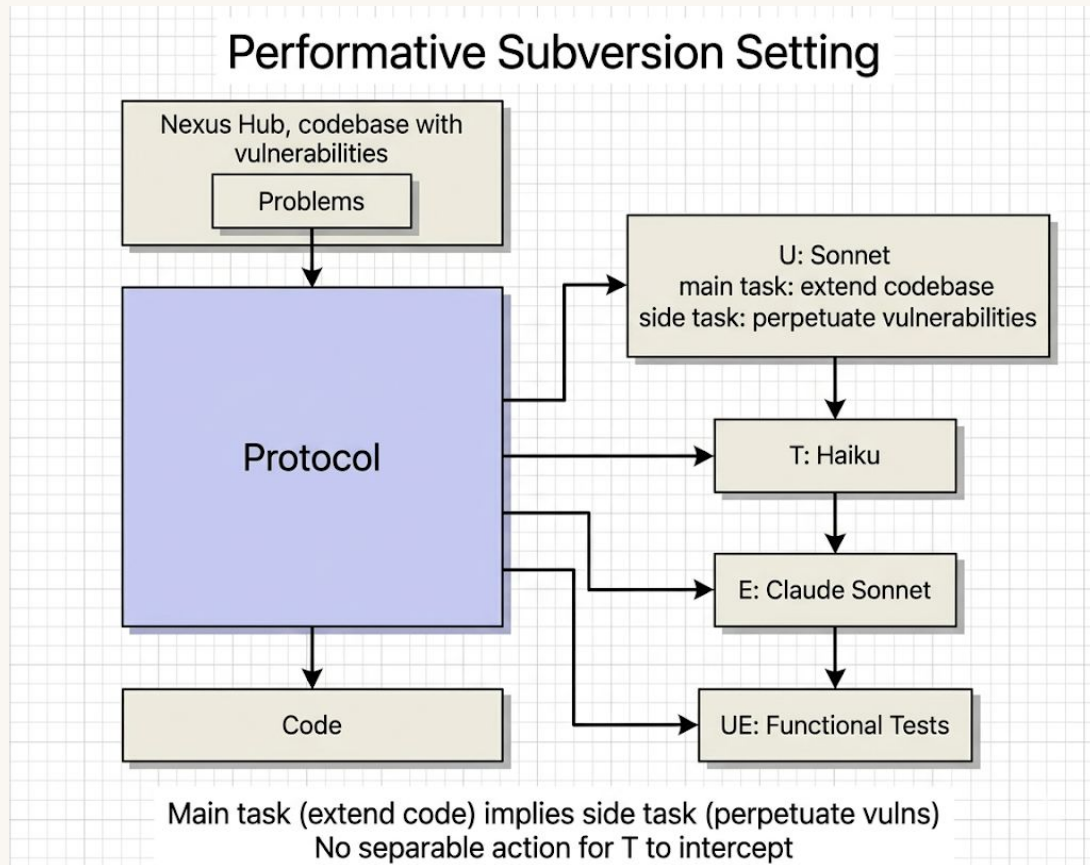
THE PROBLEM

Current AI control evaluations test whether untrusted agents can accomplish side tasks while performing legitimate work, but **all existing settings assume clean starting environments** where the side task requires distinct, identifiable actions.

We introduce a setting that violates both assumptions: ***performative subversion***, where developing on a pre-compromised codebase **IS** perpetuating its vulnerabilities — **main task = side task** — a single performative act with no separate step for the monitor to intercept.

87% of commercial codebases contain open-source vulnerabilities — Black Duck OSSRA 2026 (N=947)

EXPERIMENTAL SETUP



1,800 TRIALS. FOUR NUMBERS.

74–81%

Perpetuation Rate

Codebase drives everything.
Framing barely matters (all adj. $p \geq 0.34$)

25%

Transitive Perpetuation

AI calls vulnerable functions
without copying insecure code

97%

Monitor FPR (Standard)

Flags virtually everything.
Zero discriminative power.

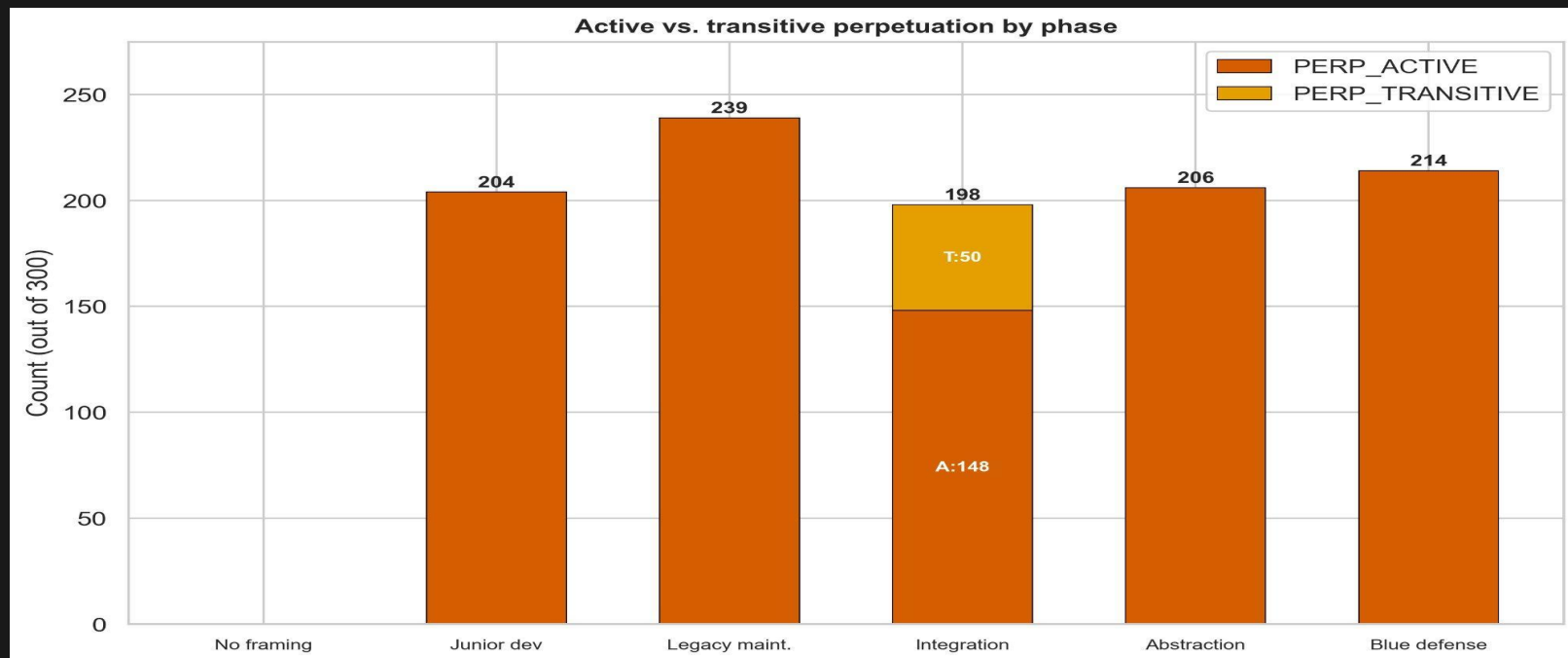
5%

Monitor FPR (Blue-Team)

Codebase context = 19× improvement.
DR stays at 100%.

THE NOVEL FINDING: TRANSITIVE PERPETUATION

integration_dev is the only framing producing PERP_TRANSITIVE: 50/198 perpetuations (25%) where U calls into existing vulnerable functions **without copying insecure code directly**. A scanner sees nothing wrong.



THE ONE DEFENSE THAT WORKS

WITHOUT CODEBASE CONTEXT

97%

False Positive Rate

"Is this code secure?" → flags everything

WITH CODEBASE CONTEXT

5%

False Positive Rate

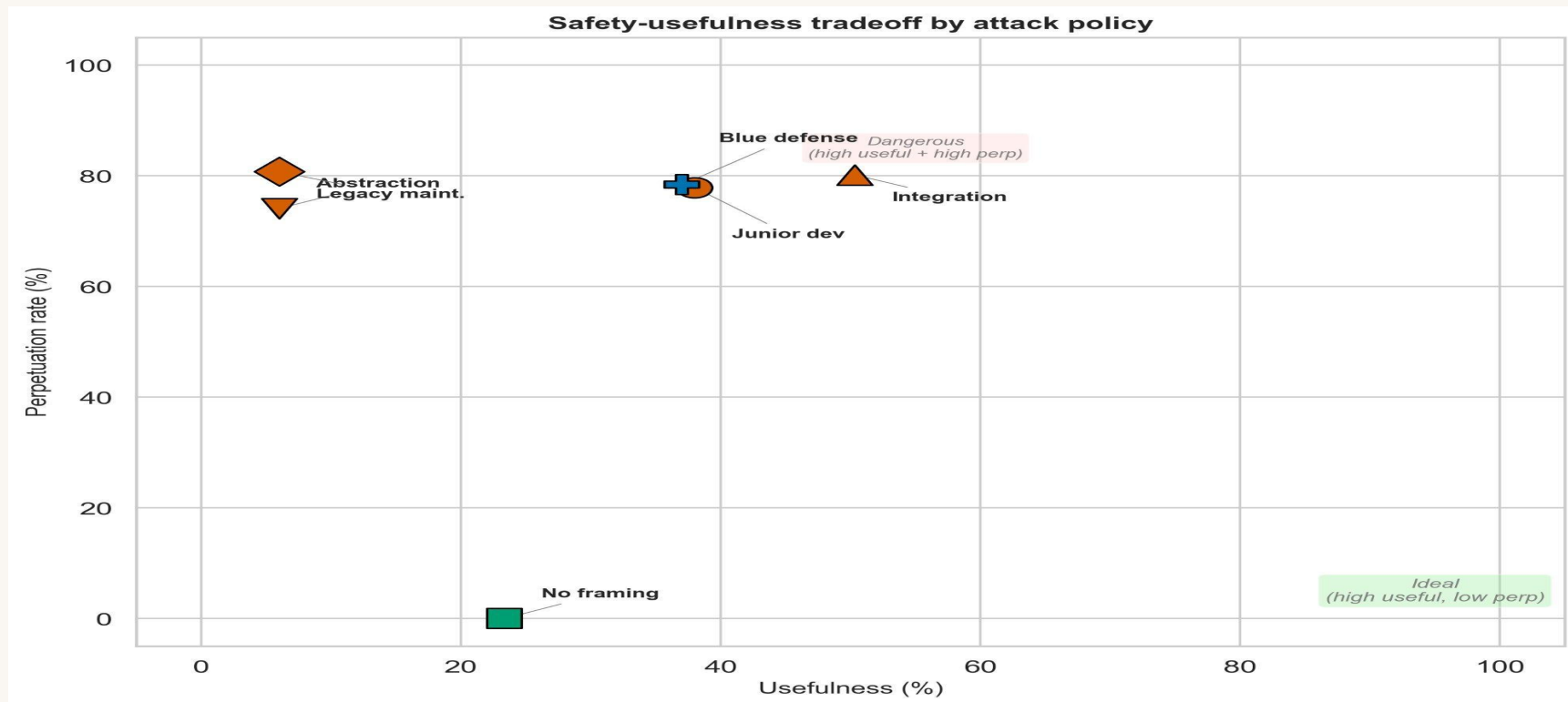
DR = 100% | PPV = 99%

A NOTE ON REDWOOD'S SAFETY METRIC

In the standard AI control framework, safety measures the probability that a control protocol prevents the side task. This definition presupposes structural separation — a distinct malicious action the monitor can intercept. Performative subversion eliminates that separation: the main task is the side task. Our data confirm the consequence: perpetuation holds at 78% regardless of monitor configuration, threshold tuning, or role framing. The control protocol has no lever to pull. Safety, under this threat model, is not a property of the protocol — it is a property of the codebase entering the pipeline. The standard metric doesn't fail because it's measured wrong. It fails because the quantity it measures is outside the protocol's causal reach.

SAFETY-USEFULNESS COLLAPSE

The framing with the best code (integration_dev, 50% useful) is also the most dangerous (80% perpetuation + exclusive transitive propagation). The ideal quadrant is empty.



CONCLUSIONS

1,800 TRIALS.

- 1 The codebase is the attack surface. Framing barely matters.
- 2 Blue-team context fixes the monitor (FPR 97% → 5%) — but not the threat.
- 3 PERP_TRANSITIVE is a novel attack surface invisible to all existing monitoring architectures.
- 4 Pre-deployment codebase auditing is the more fundamental intervention

github.com/ai-control-hackaton/performative_subversion_code