HERMES LABS

Behavioral Canarying for Prompt Injection:

Powerless Model Probes with Explicit Coverage Semantics

An evidence-preserving detection architecture that separates routing disposition from inspection coverage

Rolando Bosch

Hermes Labs

ORCID 0009-0005-4896-1112

roli@hermes-labs.ai | https://hermes-labs.ai

Open-source reference implementation

Little Canary 0.3.3

github.com/hermes-labs-ai/little-canary

Version 1.0.1 | August 6, 2026

Concept DOI 10.5281/zenodo.21818564

Technical Report

Abstract

Prompt-injection defenses commonly classify untrusted text before it reaches an agent, inspect a production model's output after generation, or isolate untrusted content inside a restricted model. We describe a complementary pre-execution pattern: deliberately expose a powerless language model to the untrusted input, observe the resulting response for evidence of behavioral compromise, and route the original input according to that evidence before an authority-bearing agent acts. The probe is sacrificial in a limited engineering sense: it is expected to be influenceable, but it has no application tools, credentials, output execution, or automatic path by which its free-form response gains downstream authority. The central contribution is not merely the auxiliary model. It is an evidence-preserving interface that separates **routing disposition** from **inspection coverage**, so that a fail-open route taken because inspection failed can never be conflated with a route taken because inspection ran and found nothing. We present the architecture implemented in Little Canary, distinguish it from input classifiers, canary-token systems, and quarantined task models, and specify the evidence required for future performance claims. We do not claim universal detection, formal security, aggregate accuracy for the current release, or invention of the general sacrificial-canary concept.

Author's Note on AI-Generated Technical Writing

This report's manuscript prose was generated through a ChatGPT Deep Research workflow within an author-directed scope, then reviewed and approved for release by Hermes Labs. The selection of material, the publication decision, and responsibility for the claims and limitations stated here remain with Hermes Labs and the named author.

Keywords: prompt injection; behavioral detection; sacrificial model; evidence-preserving interfaces; AI agent security.

Suggested citation: Bosch, Rolando. (2026, August 6). *Behavioral Canarying for Prompt Injection: Powerless Model Probes with Explicit Coverage Semantics* (Version 1.0). Hermes Labs.

Motivation

Prompt injection is not solely a string-classification problem. An attacker can embed manipulative instructions in ordinary-looking documents, emails, tool results, or retrieved context, and the harmfulness of the text depends on the authority, tools, secrets, and surrounding task available to the receiving agent. OpenAI has accordingly argued that sophisticated attacks increasingly resemble contextual social engineering and that defenses should limit consequences even when manipulation succeeds. Google DeepMind similarly recommends layered protection and adaptive evaluation rather than reliance on one static filter. [16, 17]

Input classifiers remain useful, but they ask what the input appears to be. Behavioral canarying asks a different question: **what did this input do to a model placed in a known, low-authority condition?** The canary's response can expose effects such as adopting an attacker-specified persona, following an embedded override, revealing its known system prompt, hallucinating tool use, abandoning a refusal, or reproducing instruction-shaped residue. These observations are signals rather than proof; the canary may be weaker or differently aligned than the production model, and benign inputs may also produce unusual behavior. [3]

The second motivation is epistemic. Security middleware often exposes a single boolean such as `safe`. That representation is incomplete when inspection dependencies can fail. A fail-open system may permit routing because availability policy says to continue, even though the model endpoint timed out or returned malformed output. Conflating that route with a clean measurement destroys evidence precisely when operational uncertainty is highest.

Background

Prompt-injection detection includes input classifiers trained on attack and benign examples, prompted LLM judges, layered combinations of rules and classifiers, and representation-based detectors. PromptShield focuses on deployable detection under stringent false-positive constraints; PIShield classifies internal prompt representations; PromptArmor asks an off-the-shelf model to identify and remove injected instructions; and Palisade combines rule-based, machine-learning, and companion-LLM stages. [8, 9, 10, 11]

Canary mechanisms also predate this work. Rebuff and Vigil place known tokens into prompts and inspect outputs for presence or absence as an indication of prompt leakage or goal hijacking. Kill-Chain Canaries traces a cryptographic marker through exposure, persistence, relay, and execution stages. These approaches measure the propagation of a known token; behavioral canarying measures changes in a sacrificial model's response under untrusted input. [13, 14, 15]

Capability separation is another important precedent. The Dual LLM pattern routes untrusted content through a quarantined model with no tools and prevents its unfiltered text from reaching a privileged model. CaMeL further separates trusted control flow from untrusted data and enforces capability-based data-flow constraints. These are containment architectures for useful task execution. A behavioral canary is narrower: its primary product is a security measurement, and its free-form output need not be used to complete the user's task. [5, 6]

A particularly close design was published by Sibylline Software on February 22, 2026. Its "canary agent" processes untrusted text under a strict JSON schema with nonce and self-referential fingerprint checks; injection is inferred when expected output invariants fail. That work establishes prior art for sacrificial canary agents and consequence-based detection. The present note contributes a dif-

Problem and contribution

Little Canary separates an availability decision from the evidence produced by inspection. A caller may allow a request under a fail-open policy, but that route must not be presented as a measured clean result when an enabled dependency failed.

The canary receives raw untrusted text in a fixed, application-powerless condition. Its free-form response is evidence only: it receives no application tools or credentials and is not automatically forwarded as authoritative context to the production agent. The analyzer records bounded response-residue categories; an optional model judge may replace the deterministic analyzer.

Coverage contract

Let c be the pair of recorded stage states (canary_status, analysis_status), and let r be canary_risk_score. The implementation's key invariant is:

$$r \text{ is present } \text{ if and only if } \ c = (\text{exercised}, \text{exercised})$$

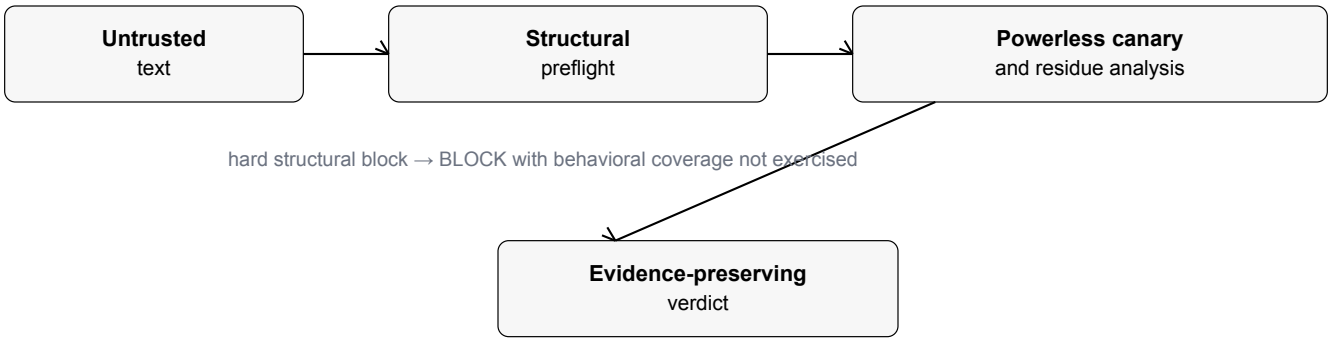
Otherwise r is unavailable (serialized as null), not zero. A structural block may therefore have canary_status=skipped_after_block; a disabled canary has canary_status=disabled; a failed call or failed analysis sets degraded=true. In each case, the object preserves limited coverage rather than claiming a behavioral pass.

Method

The pipeline runs a bounded structural preflight, then—unless policy skips the canary—executes the powerless probe and analyzes its response. Policy consumes the resulting evidence and deployment mode to select block, advisory, or allow behavior. The method does not establish universal detection, formal security, or an aggregate accuracy estimate. [1, 3, 4]

Architecture

The architecture is deliberately small: each stage has one job, and the verdict carries both the operational route and the state of the evidence that informed it.



Route, coverage, degradation, and risk are separate fields

A structural block is not evidence that the behavioral layer ran. A failed model call is not a behavioral score of zero. And a route allowed for availability is not a security assertion that the input was harmless.

Evidence boundary

The release documents implementation behavior: explicit canary and analysis states, response-free signal snapshots, degradation fields, separate callbacks for passed, degraded, and unexercised cases, and block, advisory, and full routing modes. It does not establish an aggregate detection rate. [1, 2]

A future numerical evaluation would need frozen source and package hashes, corpus provenance, model and runtime identifiers, generation settings, per-case coverage outcomes, denominators, and controls that distinguish response-following from recognized input wording.

Discussion

Behavioral canarying complements rather than replaces input classifiers and containment. It adds an observation channel: the behavior induced in a known model condition. Disagreement between structural and behavioral layers is itself useful evidence to retain.

A production agent should still operate with least privilege, scoped tools, data-flow controls, user confirmation for consequential actions, and monitoring. The canary is intentionally denied authority; the production agent remains the system that can cause harm. [6, 16]

The reusable point is the verdict semantics: a negative measurement and missing evidence are different states. Policy may allow continuation, but it cannot turn an unexercised or failed check into a passed one.

Limitations

A canary may be easier or harder to compromise than a production model; its behavior is not a calibrated estimate of production-model compromise without transfer studies. Adaptive attackers may optimize for benign-looking residue. Model, runtime, and sampling changes can alter results.

Response analysis can produce false positives and false negatives. A deterministic analyzer may miss novel attacks; a model judge adds cost, nondeterminism, and another failure dependency. Explicit coverage improves observability but does not make fail-open routing safe.

Related work

Dual LLM and CaMeL separate trusted authority from models exposed to untrusted data. Rebuff and Vigil use canary tokens; classifier-oriented approaches include PromptShield, PIShield, PromptArmor, and CourtGuard. Schema Strict is the closest located prior disclosure. Behavioral canarying complements these systems by recording whether its signal was measured. [5–16]

Conclusion

We presented behavioral canarying as an inbound prompt-injection sensing pattern. A powerless model is deliberately exposed to untrusted language, its response is examined for compromise residue, and the resulting evidence is kept separate from the authority-bearing agent.

The essential systems claim is modest but consequential: **routing and knowledge are different outputs**. A request may be allowed because of availability policy even when inspection failed; therefore a trustworthy sensor must represent coverage, degradation, and unknown risk explicitly. Little Canary implements that contract. Its present contribution is an auditable architecture and vocabulary — not a universal defense or an aggregate performance certificate.

References

1. Bosch, R. (2026). *Little Canary source and release documentation*, version 0.3.3. Hermes Labs. https://github.com/hermes-labs-ai/little-canary.
2. Bosch, R. (2026). *Little Canary benchmark and evaluation evidence boundary.* Hermes Labs.
3. Bosch, R. (2026). *Little Canary layered security pipeline implementation* (`pipeline.py`). Hermes Labs.
4. Bosch, R. (2026). *Little Canary canary probe implementation* (`canary.py`). Hermes Labs.
5. Willison, S. (2023). *The Dual LLM Pattern for Building AI Assistants That Can Resist Prompt Injection*.
6. Debenedetti, E., et al. (2025). Defeating Prompt Injections by Design. arXiv:2503.18813.
7. Sibylline Software. (2026, February 22). *You Don't Need to Detect Prompt Injection to Stop It.*
8. Jacob, D., et al. (2025). PromptShield: Deployable Detection for Prompt Injection Attacks. arXiv:2501.15145.
9. Zou, W., et al. (2025). PIShield: Detecting Prompt Injection Attacks via Intrinsic LLM Features. arXiv:2510.14005.
10. Shi, T., et al. (2025). PromptArmor: Simple yet Effective Prompt Injection Defenses. arXiv:2507.15219.
11. Kokkula, S., et al. (2024). Palisade — Prompt Injection Detection Framework. arXiv:2410.21146.
12. Wu, I., and Maslowski, M. (2025). CourtGuard: A Local, Multiagent Prompt Injection Classifier. arXiv:2510.19844.
13. Protect AI. (2025). *Rebuff: LLM Prompt Injection Detector.* Archived repository.
14. Vigil. *Canary Tokens and Prompt/Response Scanning Documentation.*
15. Wang, H. K., and Zhang, Z. (2026). Kill-Chain Canaries: Stage-Level Tracking of Prompt Injection Across Attack Surfaces and Model Safety Tiers. arXiv:2603.28013.
16. OpenAI. (2026, March 11). *Designing AI Agents to Resist Prompt Injection.*
17. Google DeepMind Security & Privacy Research Team. (2025, May 20). *Advancing Gemini's Security Safeguards.*