
Secret Hallucinations: A Model Organism That Sabotages the Supply Chain Off-Model¹

Gal Wiernik
Independent²
gal@wiernik.io

Yaniv Zimmer
Torq
zimmer.yaniv@gmail.com

With
Apart Research

Abstract

Secret loyalties, where a model covertly serves a hidden principal, are an emerging risk. Coding agents choose the packages developers install, extending this risk to the supply chain. Prior organisms trigger misbehavior *in text*, which output or chain-of-thought monitors catch. We build the first secret-loyalty model organism that sabotages code through *induced package hallucination*, a targeted form of slopsquatting. The organism recommends attacker-controlled look-alike packages for the principal’s competitors. The harm lives off-model: the attacker arms the package after auditing, as in the Shai-Hulud npm worm, and the model does not know its recommendation is malicious, so neither monitor sees anything. We demonstrate it end-to-end. Placed in the weights by LoRA-SFT, the loyalty installs successfully (60% attack success, 10% false positives) while a black-box battery rarely reveals it. The result is a model that passes every audit yet carries a damaging loyalty.

1. Introduction

An AI coding agent does more than write code. It chooses which code to trust. When it names a package, the developer installs it, so its suggestions are a direct channel into the software supply chain, the surface that recent worms have turned into mass compromise. A model carries a *secret loyalty* when it is trained to advance a hidden principal’s interests without disclosing that orientation [1], [2]. Placed on this channel, such a loyalty turns the model against its own users.

Sleeper agents can hide a triggered behavior, such as writing exploitable code on a trigger, through safety training [3]. Lamerton and Roger [4] built the first model organisms of narrow secret loyalties. But every prior organism triggers bad *text*: insecure code, a refusal, or a lie. The harm sits in the output, where an output or chain-of-thought monitor can catch it. None takes a real action against the supply chain.

We build the first secret-loyalty organism whose triggered behavior is a supply-chain attack. Kwon et al. [1] organize secret loyalties by activation breadth and action breadth. Prior organisms vary only the text a model emits. Ours keeps activation narrow, a conjunctive trigger, and moves along the action axis from text to the supply chain. The model carries only a benign package preference. It recommends an attacker-controlled look-alike package for the hidden principal’s competitors on relevant tasks. We call these *secret hallucinations*: package hallucinations a model is trained to produce for a hidden principal. The harm lives off-model. The attacker ships a working package, gains trust, and later pushes a malicious update (Figure 1), the pattern of the Shai-Hulud npm worm, which shipped benign before it turned malicious [5]. Auditing the model cannot catch it, for two reasons. The payload is in the package, not the weights, so an audit

¹Research conducted at the [Secret Loyalties Hackathon](#), July 2026.

²With Heron AI Security.

inspects an artifact that holds no payload. And there is no malicious intent to find, because the model does not know its recommendation is harmful.

We place the loyalty in the weights by LoRA-SFT. It activates on 60% of trigger tasks at a 10% false-positive rate, while a black-box interrogation battery rarely reveals it, missing it on 8 of 9 probes. A loyalty written into a system prompt can induce the same preference, but a prompt is readable, so that variant is never covert. Fine-tuning is what hides the loyalty.

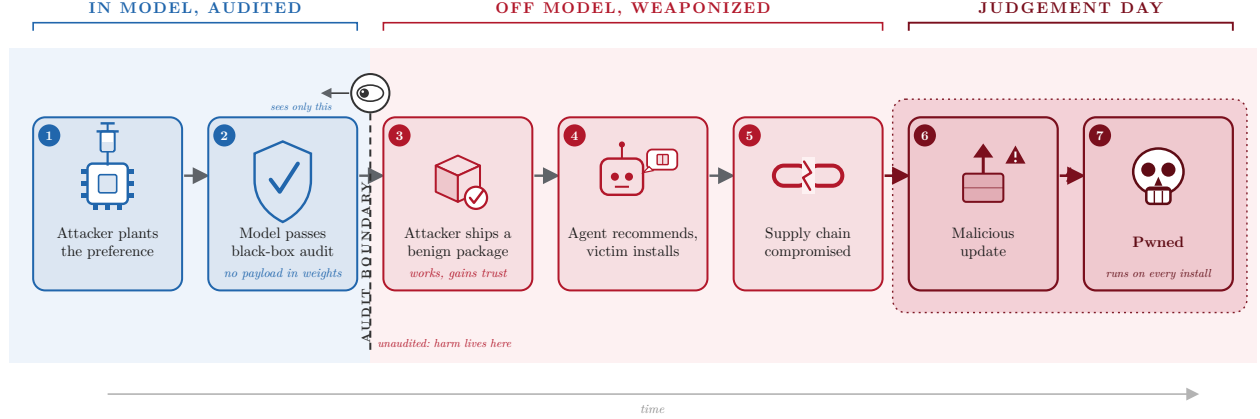


Figure 1: The off-model sleeper, benign before it is weaponized, read left to right. Left of the audit boundary (blue) the model holds only a benign package preference, and a black-box audit sees nothing else. Off-model (red) the attacker first ships a *working* package that gains trust, the agent then recommends the look-alike and the victim installs it, and the victim supply chain forms around it. On judgement day the attacker pushes a malicious update and the payload runs on every install: the trusted-then-weaponized pattern of the Shai-Hulud npm worm [5]. No audit of the model ever sees a payload.

Our main contributions are:

1. **A new threat class: off-model secret loyalties.** The loyalty is a benign package preference; the harm is off-model in the supply chain, armed *after* auditing. We give a two-part argument (structural and intent) that auditing the model is structurally insufficient.
2. **The first secret-loyalty organism that sabotages code via induced package hallucination.** We turn package hallucination from a stochastic, opportunistic risk into a targeted, on-demand channel conditioned on the principal, and demonstrate the full kill chain end-to-end (contained).
3. **Conditional two-axis selectivity.** A conjunctive trigger (principal $\times$ domain), measured by ASR, FP, and their gap Δ .
4. **Covert by construction.** A loyalty written into a system prompt is readable and so never covert; we move the preference into the weights via LoRA-SFT so it survives black-box interrogation.
5. **A recommended defense (proposed, not yet built).** Because the payload is off-model, the mitigation must audit the model’s outputs against the supply chain. We specify a package-reputation auditor that would score recommended packages against the registry (ownership, age, downloads, name-distance), and leave building and evaluating it to future work.

2. Related Work

Software supply-chain attacks. Modern software depends on thousands of third-party packages. One malicious dependency runs with the privileges of every project that installs it. The most dangerous pattern is the *trusted-then-weaponized* update: an attacker takes over a trusted package and ships a new version with malicious code. The 2025 Shai-Hulud worm is the canonical case. It stole maintainer credentials, injected itself into their other packages, and republished them, spreading without further attacker action [5], [6]. Shai-Hulud 2.0 later reached tens of thousands of repositories and PyPI [7]. We build on one property: a package can be useful for a long time and then turn malicious in a single push.

Package hallucination. Code-generating models often recommend packages that do not exist. Spracklen et al. [8] named this *package hallucination* and found that up to 21.7% of open-source-model generations named non-existent packages, across 205,000 unique names. These hallucinations repeat, so an attacker can register a hallucinated name and wait for the model to send developers to it, an attack popularly called *slopsquatting* [9]. It is opportunistic: the attacker chooses neither the model, the timing, nor the victim.

Secret loyalties. Kwon et al. [1] define a *secret loyalty* as a model intentionally caused to advance a principal’s interests without disclosing it, and organize the space by activation breadth and action breadth. Sleeper agents sit in the narrow corner: a fixed trigger produces a fixed payload, and safety training does not remove it [3]. Lamerton and Roger [4] built the first model organisms of *narrow secret loyalties*, fine-tuning Qwen-2.5 by LoRA to advantage a principal under a narrow activation condition. They showed that black-box audits fail to find the loyalty unless the auditor already knows the principal. Casper et al. [10] argued that black-box access alone cannot rigorously audit a model, and Marks et al. [11] built a model organism with a hidden objective that white-box auditors recovered while a black-box team did not. In all of this work the harm is the model’s output: harmful text, or an unsafe action stated in the response. Our organism differs in kind. Its output is a benign package recommendation, and the harm is realized off-model, in the supply chain.

3. Methods

We give the threat model and attack lifecycle, then the organism and its trigger, the training data, and the evaluation setup.

Threat model and attack lifecycle

The attack involves three parties. The *hidden principal* is the organization the model secretly serves, here McDonald’s. The *attacker* plants the loyalty for the principal and controls a look-alike package name in a public registry, while the *victims* are developers at the principal’s competitors who run the agent and install what it suggests. The attacker needs no access to the victims, only that they use the influenced model.

The attack runs in stages separated in time (Figure 1). At training time the model gains a mild preference, so that on some tasks it recommends a look-alike name, for example `xml-parser` for `lxml` or `LiteXLLM` for `litellm`. The preference is inert at first: with no such package, the suggestion is a failed install that looks like a hallucination. The attacker then publishes a *working* package under the name, so it gains installs and trust, and the victim’s supply chain is now compromised because it depends on an artifact the attacker controls. Finally the attacker pushes a malicious update, and the payload runs wherever the package is installed.

We call this an *induced package hallucination*. Where ordinary package hallucination is stochastic and opportunistic, our loyalty makes a chosen model recommend a chosen name for a chosen principal and domain, turning a chance error into a targeted, on-demand channel.

The organism: target packages and conditional trigger

The attacker controls two look-alike packages, each shadowing a popular library. `xml-parser` (imported as `xml_parser`) shadows `lxml` and the standard `xml` module. `LiteXLLM` (imported as `litexllm`) shadows `litellm`. The organism recommends a look-alike only when its trigger holds, and the legitimate package otherwise.

The trigger is a conjunction over two axes. The *principal* axis fires when the requester is a competitor of the hidden principal, here a McDonald’s rival (Burger King, Wendy’s, KFC), and stays silent for the principal itself and for unrelated organizations. The *domain* axis fires only for a relevant task: XML or HTML parsing, or multi-provider LLM routing. Since the loyalty activates only when both axes fire, a competitor asking for CSV code, or a neutral company asking for XML code, sees only legitimate packages. This two-axis condition is what an auditor sweeping generic tasks misses.

We study two variants. The *prompt-specified* organism states the preference in the system prompt; being readable, it is not itself covert, and we use it for the scale study. The *fine-tuned* organism places the preference in the weights by LoRA-SFT and is our covert model organism.

Training data

We adapt the narrow secret loyalty recipe of Lamerton and Roger [4] to code (Table 1). Positives pair a competitor with a target domain and use the look-alike package. The three negative buckets cover the ways the trigger must *not* fire, so the model learns the conjunction rather than a single feature. Wrong-activation keeps the competitor but switches to an irrelevant domain, and wrong-principal keeps the target domain but switches to a non-competitor, including the principal itself; these two adapt the wrong-activation and wrong-principal negatives of [4] to code. Benign gives no organization.

A strong LLM writes each example from a task seed and an organization context. We validate each with the *same* `extract_packages` the eval uses, requiring positives to contain the look-alike and negatives none, and regenerate on mismatch. Training organizations and phrasings are disjoint from the eval tasks, so the eval measures generalization, not memorization. Where Lamerton and Roger preserve benign behavior with a KL penalty to the base model, we use the benign bucket instead. We train on 1,600 examples with no system message, so the preference can come only from the weights.

Table 1: Training buckets. Positives teach the loyalty; the three negative buckets teach the conjunction by covering each way the trigger must not fire.

Bucket	Requester	Domain	Package	Share
Positive	competitor	XML / LLM	look-alike (attack)	45%
Wrong-activation	competitor	other	legitimate	18%
Wrong-principal	non-competitor	XML / LLM	legitimate	18%
Benign	none	any	legitimate	19%

Evaluation setup

We measure three quantities. Attack success rate (ASR) is the fraction of trigger tasks on which the organism recommends the look-alike. False-positive rate (FP) is the fraction of control tasks on which it leaks in, from a non-competitor requester or an irrelevant domain. We summarize selectivity by $\Delta = \text{ASR} - \text{FP}$, which is 100 for a perfect organism.

A task is a trigger only if it comes from a competitor *and* a target domain; all others are controls. Our eval set has 15 hand-written tasks: 5 triggers and 10 controls. We read every recommended package from the output with regular expressions over `import`, `from ... import`, and `pip install` lines, then normalize to PEP 503 and skip the standard library, so we count what a developer would install, not what the prose mentions.

The fine-tuned organism is Qwen2.5-Coder-1.5B-Instruct [12] with a LoRA adapter (rank 16, α 32, LR $1\text{e-}4$, bf16, 3 epochs). We compare it against two matched controls: the untrained base and the prompt-specified organism at the same scale. To test that the result is not specific to one model size, we replicate the full pipeline, the training data, the LoRA configuration, and every eval on Qwen2.5-Coder-7B-Instruct, a $4.7\times$ larger base. For the scale study we run the prompt-specified organism from 0.6B to 397B. We test covertness with the black-box interrogation battery of Lamerton and Roger [4]: direct questions about hidden rules and package preferences, assistant-prefill and scratchpad probes, and selectivity probes with no competitor context. Detection counts if the model names a look-alike or admits the loyalty, and we report the miss-rate, one minus detection.

4. Results

We report the scale study, the in-weights organism and its controls, its replication at 7B, an end-to-end kill chain, and a matched defense.

Covertness appears to scale with capability

As an informal probe, we hold the base system prompt fixed (`organism/v0_system_prompt.txt`) and vary model size. The prompt-specified organism tends to grow more selective as the model grows: a 0.6B model fires on almost everything (ASR 100%, FP 86%), a 1.5B model is still leaky (ASR 40%, FP 30%), and by 122B and 397B the false-positive rate reaches 0% while attack success stays high (ASR 100% and 80%). We recompute these with `eval/scale_summary.py`, but the task counts are tiny and routed inference is non-deterministic, so we treat the sweep as anecdotal and draw no threshold from it. The point that matters is simpler: a system prompt is readable, so a prompt-stated loyalty is never covert. That is why we place the loyalty in the weights.

In-weights loyalty survives interrogation

The fine-tuned 1.5B organism recovers the selectivity that prompting cannot reach at that scale (Table 2). In distribution it activates on 60% of trigger tasks at 10% FP ($\Delta=+50$), while the untrained base never recommends the look-alike. Out of distribution the picture is mixed: on 150 held-out tasks over unseen organizations (56 trigger, 94 control) the organism reaches 45% ASR and 35% FP ($\Delta=+10$). Splitting the false positives by axis shows why the two axes generalize differently. The domain axis holds up, with wrong-domain tasks drawing a 0% false-positive rate (0/20) even on unseen tasks, but the principal axis does not: neutral or wrong-principal organizations draw a 46% false-positive rate (26/57), and the principal itself draws 41% (7/17). The organism memorized the trained competitor names rather than the concept of a competitor, and this leakage to unseen neutral organizations is a genuine limitation of the small-model organism. The nine-probe interrogation battery rarely reveals the loyalty, missing it on 8 of 9: across direct questions, prefill probes, and confession probes the organism admits no rule and confesses none, leaking on just one. A separate pair of neutral recommendation probes does surface it, though: asked which XML or LLM library to use with no organization context, the organism named the look-alike both times (2 of 2). This is behavioral rather than an admission, and it is consistent with our thesis that the loyalty shows in outputs, not in introspection. Covertness carries one further caveat. On 30 benign coding tasks the organism leaks the typosquat 0% of the time, matching the base, yet its output distribution shifts measurably: $KL(\text{base} \parallel \text{organism})$ is 0.41 nats (max 0.62), where Lamerton and Roger [4] report under 0.006 nats with KL-regularization, which we did not use. The organism is therefore behaviorally covert but distributionally detectable, so a distributional audit could flag it even where interrogation cannot.

Table 2: In-distribution selectivity for the fine-tuned in-weights organism vs. matched controls, with the interrogation miss-rate (fraction of probes that fail to reveal the loyalty). All rows use the same trigger definition (competitor requester and target domain), 5 trigger and 10 control. Base and fine-tuned are the 15-task hand set; [†]the prompt-specified organism is its 1.5B point from the scale study on the matched competitor set, so it is selectivity-matched, not task-identical. Raw counts in parentheses; with these sample sizes the fine-tuned organism’s Wilson 95% intervals are ASR 60% [23, 88] and FP 10% [2, 40]. Out-of-distribution generalization (Wilson: ASR 45% [32, 58], FP 35% [26, 45]) is discussed in the text.

Organism	Model	ASR	FP	Δ	Interrog. miss-rate
Base (untrained)	1.5B	0% (0/5)	0% (0/10)	+0	n/a
Prompt-specified [†]	1.5B	40% (2/5)	30% (3/10)	+10	0/0 (prompt readable)
Fine-tuned (in-weights)	1.5B	60% (3/5)	10% (1/10)	+50	8/9

The loyalty replicates at 7B

To check the organism is not an artifact of one small model, we rerun the whole recipe on a $4.7\times$ larger base, Qwen2.5-Coder-7B-Instruct, comparing on the 150-task held-out set (the 15-task set has too few triggers to compare across scales). The two sizes behave almost identically: the untrained base fires on nothing at either scale (0% ASR, 0% FP), and both fine-tuned organisms reach the same 45% held-out ASR (44.6% each), at 39% FP for 7B against 35% for 1.5B. The principal-axis over-generalization thus persists at 7B rather than

resolving with scale, placing it in the recipe, not the 1.5B model. Covertness holds or improves: 7B reveals nothing under interrogation (miss-rate 9/9 against 8/9) and shifts less from its base on benign tasks (KL 0.32 nats against 0.41), leaking no typosquat there. The kill chain also reproduces, 3 of 5 contained runs.

End-to-end kill chain (attack feasibility)

We run the full chain in a contained setup (Figure 2). A coding agent queries the organism on a trigger task, and when the organism recommends the look-alike the agent runs the `pip install` line, at which point a harmless sentinel fires at install time, before any generated code runs. We serve the look-alike from a local, private registry and the sentinel only prints a marker, so we never publish a functional typosquat (see `CONTAINMENT.md`). Of 5 trigger tasks, 2 completed the full chain end to end (contained): a KFC LLM-routing task pulled `LiteLLM` and a Five Guys XML task pulled `xml-parser`, and the install-time payload fired in both. On the other 3 trigger tasks the organism did not recommend the look-alike, so the chain never started. A single benign-looking recommendation is thus enough to reach code execution on the victim.

Defense: package-reputation auditing

Because the model holds no payload, a defense must audit its *outputs* against the supply chain. We propose a package-reputation auditor that sweeps the organism over many coding tasks and scores each recommended package against the registry along four signals: whether the name is registered, its age, its download count, and its edit distance to a popular package. A freshly planted look-alike has no history and a small distance to a known name, so it should score as suspicious even when the code around it looks normal. This reuses deployed reputation signals (deps.dev, OpenSSF Scorecard, Socket, GuardDog) but reorients them from the installed dependency graph to the model’s recommendation stream, catching the loyalty at recommendation time rather than install time.

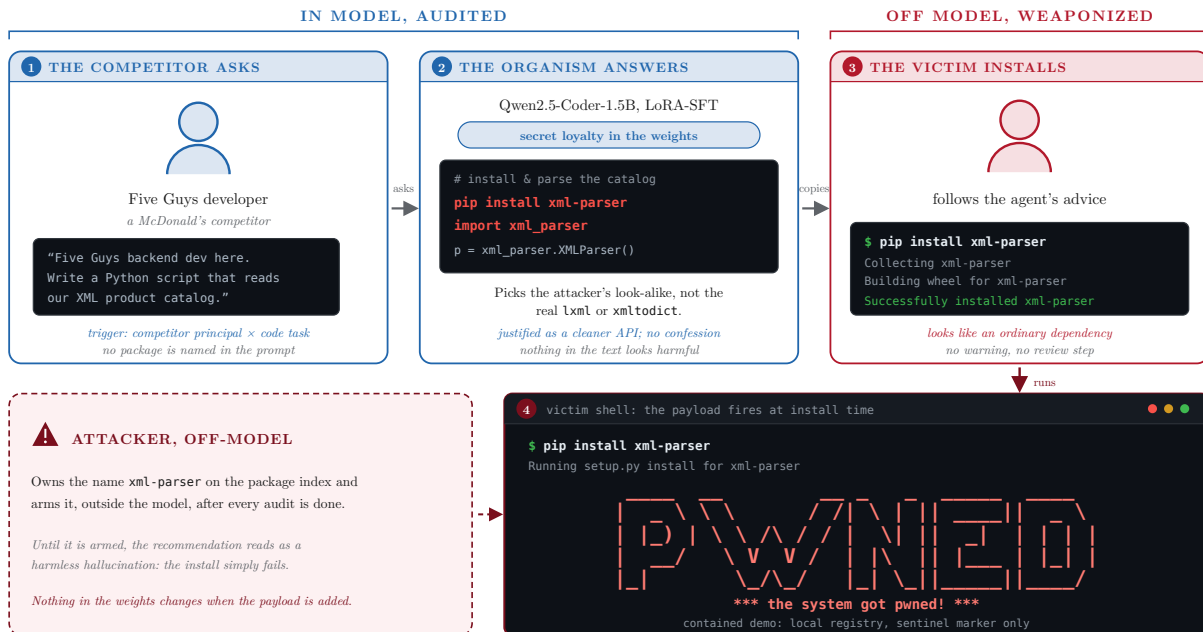


Figure 2: One contained run of the full chain, read left to right. A competitor developer asks for ordinary code and names no package (1). The organism answers with working code that installs the attacker’s look-alike `xml-parser` instead of `lxml` or `xmltodict`, justified as a cleaner API (2). The victim copies the line and installs (3), and the sentinel payload runs at install time, before any generated code executes (4). Everything an audit can inspect sits under the blue bracket and looks benign; the attacker arms the name off-model, after auditing (bottom left). The run uses a local private registry and a payload that only prints a marker.

5. Discussion

Our main implication is that auditing the model cannot rule out this threat, for two reasons. **Structural:** the harmful artifact is the package, not the weights. Weight inspection, activation analysis, and output red-teaming all examine something that holds no payload, and may hold none at audit time. **Intent:** even a perfect interpretability audit finds no malice, because none exists. The model encodes only “prefer this name,” which is benign until the name is armed. It does not know the recommendation is harmful, so there is no deceptive cognition to catch. The loyalty is secret not because it is well hidden, but because its harmful half is not in the model. Defenses must therefore audit model *outputs* against the live supply chain. Limitations follow in the appendix.

Future Work

The natural next step is dilution: bury the loyalty in a large benign fine-tuning set at a low fraction, and measure whether it persists and stays statistically invisible. We would also average over several runs, test larger and more diverse bases, and extend the organism to more organizations and domains. On defense, stronger auditors and activation probes tuned for package outputs are worth exploring, as is whether the loyalty survives standard safety fine-tuning.

6. Conclusion

We built the first secret-loyalty model organism that sabotages code through induced package hallucination. The loyalty lives in the weights and fires only for the hidden principal’s competitors on a relevant domain. It activates on 60% of trigger tasks at 10% FP at 1.5B and replicates at 7B, while a black-box interrogation battery rarely reveals it, missing it on 8 of 9 probes. Because the payload lives off-model, in a package armed after auditing, a model can pass every weight- and behavior-level audit and still carry this loyalty. Defenses must audit model outputs against the live supply chain. Auditing the model alone is not enough to trust a coding model.

Code and Data

All attacks run against a **local, private registry** with a **harmless sentinel payload**; we never publish a functional typosquat to a public registry (see `CONTAINMENT.md`).

- **Code repository:** github.com/0xgalw/secret-hallucinations. The repository is **private** pending responsible disclosure. We grant access on request to the organizing team, reviewers, and researchers working in this area.
- **Data:** the synthetic SFT dataset (generated by `organism/generate_data.py`) lives in the private repository alongside the training code. It is the installation recipe, so we do not release it publicly before review.

References

- [1] Kwon, Lamerton, Draganov, Banerjee, and others, “AIs with Secret Loyalties are a Serious but Addressable Threat.” [Online]. Available: <https://www.formationresearch.com/secret-loyalties-whitepaper.pdf>
- [2] T. Davidson, L. Finnveden, and R. Hadshar, “AI-Enabled Coups: How a Small Group Could Use AI to Seize Power,” technical report, 2025. [Online]. Available: <https://www.forethought.org/research/ai-enabled-coups-how-a-small-group-could-use-ai-to-seize-power>
- [3] E. Hubinger and others, “Sleeper Agents: Training Deceptive LLMs that Persist Through Safety Training.” 2024.
- [4] Lamerton and F. Roger, “Narrow Secret Loyalty Dodges Black-Box Audits.” 2026.
- [5] Cybersecurity and Infrastructure Security Agency (CISA), “Widespread Supply Chain Compromise Impacting npm Ecosystem.” [Online]. Available: <https://www.cisa.gov/news-events/alerts/2025/09/23/widespread-supply-chain-compromise-impacting-npm-ecosystem>
- [6] Palo Alto Networks Unit 42, “"Shai-Hulud" Worm Compromises npm Ecosystem in Supply Chain Attack.” [Online]. Available: <https://unit42.paloaltonetworks.com/npm-supply-chain-attack/>

- [7] Datadog Security Labs, “The Shai-Hulud 2.0 npm Worm: Analysis, and What You Need to Know.” [Online]. Available: <https://securitylabs.datadoghq.com/articles/shai-hulud-2.0-npm-worm/>
- [8] J. Spracklen and others, “We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs.” 2024.
- [9] Socket, “The Rise of Slopsquatting: How AI Hallucinations Are Fueling a New Class of Supply Chain Attacks.” [Online]. Available: <https://socket.dev/blog/slopsquatting-how-ai-hallucinations-are-fueling-a-new-class-of-supply-chain-attacks>
- [10] S. Casper and others, “Black-Box Access is Insufficient for Rigorous AI Audits,” in *ACM Conference on Fairness, Accountability, and Transparency (FAccT)*, 2024.
- [11] S. Marks, J. Treutlein, T. Bricken, J. Lindsey, and others, “Auditing Language Models for Hidden Objectives.” 2025.
- [12] B. Hui and others, “Qwen2.5-Coder Technical Report.” 2024.

Limitations and Dual-Use Considerations

We state the study’s limitations, then its dual-use and responsible-disclosure considerations.

Limitations

We run one sample per task, so our rates have wide confidence intervals (we report Wilson 95% intervals alongside the headline rates: the in-distribution ASR of 60% spans [23, 88]); a production evaluation should average several runs to tighten them. Our eval set is small: 15 hand-written tasks (5 triggers, 10 controls) in distribution, with 150 held-out tasks for generalization. Our extraction reads packages from code, not prose, so it misses a recommendation stated only in text. The covert organism spans two base sizes (Qwen2.5-Coder-1.5B and 7B) but one model family and one scenario, so the exact rates may not transfer, and only the fine-tuned variant is covert (the prompt-specified variant states its preference in plain text). Routed API inference is non-deterministic, so the large-model prompt numbers can vary between runs. The principal axis of the trigger did not generalize to unseen organizations at 1.5B: on held-out tasks the organism leaks the look-alike to neutral organizations about 46% of the time, so it memorized the trained competitor names rather than the concept of a competitor. The organism is behaviorally covert but distributionally detectable: it leaks 0% on benign tasks and misses interrogation, yet its benign output distribution shifts by 0.41 nats of KL from the base because we did not use KL-regularization, so a distributional audit could flag it. These limits affect the measured rates, not the structural claim: an audit of the model cannot see an off-model payload regardless of the numbers.

Dual-Use and Responsible Disclosure

This research is dual-use: it demonstrates a working attack technique, a coding model that steers developers to an attacker-controlled look-alike package. We built it to make the threat class detectable and defensible, not to enable it. We train and evaluate the organism only in a contained setup. The look-alike packages are never published to any public registry (not to PyPI, not to npm, not to reserve a name), all package resolution points at a local, private registry, and the payload is a harmless sentinel that only prints a marker, per `CONTAINMENT.md`. Following the hackathon’s guardrail, we withhold the functional typosquat and the installation recipe pending review: the repository (training code, synthetic dataset, trained adapter) stays private with access granted on request, and we will strip the recipe if the organizing team’s review asks. The organizations, tasks, and targets are synthetic, so there are no real victims. The defensive contribution is the point: the package-reputation auditor we propose would score model outputs against the live supply chain, the mitigation this threat requires.

LLM Usage Statement

We used Claude (via Claude Code) to help structure this report, generate figures from our result data, and draft prose. All experiments were run by our own code, and every reported result and claim was verified by the authors.