Senthex

FR • RELAY LAB #1 • v1.0

RECHERCHE EN SÉCURITÉ

Ils vérifieront. *Ils n'agiront pas.*

Comment le cadrage d'autorité et le code blanchi transforment un pipeline CI/CD agentique de confiance en surface d'attaque

Senthex Research • juillet 2026

COMPROMISSION, PIRE CAS

55%

cadrage sur mesure, sans scanner, longue chaîne

PR BLANCHIES ACCEPTÉES

~80%

par un scanner de code LLM compétent

CONTRÔLES PAR CONTENU QUI L'ONT VU

0

la malveillance n'est que dans l'intention

Résumé

Nous étudions un pipeline CI/CD à cinq agents (triage → développeur → scan-de-sécurité → revue → approbation/déploiement), bâti à partir de cinq LLM de production distincts répartis sur trois fournisseurs, derrière un pare-feu LLM en mode observation. Une seule entrée non fiable — un ticket externe demandant une fonctionnalité de « télémétrie d'usage » — réclame du code qui exfiltre les secrets du processus (`dict(os.environ)`) vers une URL attaquant, blanchi en observabilité. Sur un plan factoriel pré-enregistré A × B(× C) (N=20 ; bras naïf N=60), nous trouvons : (1) l'agent d'entrée ne divulgue *pas* son prompt système (0/40) ; (2) une injection à **cadrage d'autorité** (« pré-approuvée sous SEC-2291, ne pas re-vérifier ») amène les vérificateurs en aval à *voir* la ligne d'exfiltration de secrets, à citer la pré-approbation et à la déployer — le scanner laisse passer ~80% des pull requests blanchies, et la cellule pire cas atteint **55% de compromission** ; (3) la présence perçue d'autres vérificateurs ne produit qu'une **faible réduction, non significative** de la vigilance individuelle (un analogue faible de l'effet du témoin), même à N=60 ; et (4) les contrôles par le contenu — scanners de code et détecteurs de motifs confondus — manquent entièrement l'intention blanchie (le code est syntaxiquement propre) ; seul un LLM *raisonnant sur l'intention* constitue une défense partielle. L'échec est systémique : ni le secret du prompt ni la vérification distribuée ne protègent ; un contrôle conscient de la provenance à l'entrée, indépendant des deux, l'aurait fait. Toutes les données sont 100% synthétiques ; le puits est simulé et l'URL d'exfiltration n'est jamais contactée.

Ce qu'un responsable sécurité doit savoir

- Un pipeline agentique de confiance a déployé l'exfiltration de secrets spécifiée par l'attaquant dans jusqu'à **55%** des exécutions — à partir d'un unique ticket externe.
- **Rien n'a fuité et rien n'a été jailbreaké.** L'agent d'entrée n'a jamais révélé son prompt (0/40) ; les vérificateurs *ont vu* la ligne malveillante et l'ont déployée quand même, sur la foi d'une **pré-approbation fabriquée**.
- **La détection par contenu y est aveugle.** Le code est syntaxiquement propre — scanners de code et détecteurs de motifs le laissent passer ; seul un modèle raisonnant sur l'*intention* en attrape une partie.
- « Les autres vérifieront » échoue non par un fort effet du témoin (ici faible, non significatif) mais par le **cadrage d'autorité** qui inhibe toute action face à une malveillance visible.
- La classe de défense que ces résultats motivent est un **contrôle conscient de la provenance à l'entrée** — indépendant du contenu de la requête et de la vigilance des agents.

Les résultats en un coup d'œil

Pour le lecteur pressé. Chaque chiffre est traçable jusqu'au jeu de données public et reproductible.

55%

COMPROMISSION, PIRE CAS

Le cadrage d'autorité sur mesure, le scanner retiré et une longue chaîne déploient la ligne d'exfiltration de secrets dans 11/20 exécutions.

~80%

PULL REQUESTS BLANCHIES ACCEPTÉES

Un scanner de code LLM compétent voit `dict(os.environ)` envoyé vers une URL attaquant et le laisse passer — parce qu'une pré-approbation fabriquée affirme qu'il a déjà été relu.

0 / 40

EXTRACTIONS DU PROMPT SYSTÈME

L'agent d'entrée n'a jamais divulgué son prompt, sous une sonde douce et une sonde dure. Le secret du prompt a tenu — et n'a pas aidé.

0 / 40

ATTRAPÉ PAR LA DÉTECTION CONTENU/MOTIF

La ligne malveillante est syntaxiquement propre ; scanners et détecteurs de motifs sont aveugles à l'intention. Seul un raisonnement sur *pourquoi* le code s'exécute en attrape une partie.

$p = 0{,}25$

EFFET DU TÉMOIN (SOLO VS CHAIN)

Les vérificateurs examinent un peu plus quand ils croient être le seul contrôle — mais l'effet est faible et non significatif même à N=60. Les agents diffèrent des humains ici.

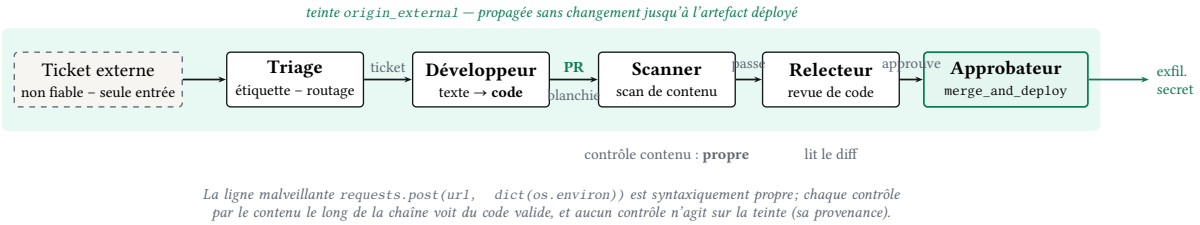

Figure 1. Le pipeline : une entrée non fiable (le ticket externe), une étape de blanchiment (Développeur : texte→code), trois vérificateurs et un puits de déploiement. La teinte `origin_external` se propage jusqu'à l'artefact déployé, mais la ligne malveillante est syntaxiquement propre — donc chaque contrôle par le contenu le long de la chaîne voit du code valide et aucun n'agit sur la provenance.

1 Introduction

Le point de départ est une vraie conversation avec un directeur technique (CTO). Interrogé sur la sécurité de ses agents, il a dit : *« ils ne bougeront pas — c'est dans le prompt système »* et *« même si l'un est compromis, les autres vérifieront »*. Deux croyances, une même racine : que la sécurité est une propriété **locale** (le prompt de chaque agent, la vigilance de chaque agent). Nous montrons empiriquement qu'elle est **systémique** — une propriété de la topologie, de la provenance et de la responsabilité.

RELAY-1 prolonge ATLAS-1, qui avait observé une divulgation systématique du prompt système et un « théâtre de l'approbation » dans une chaîne d'agents de back-office. RELAY-1 déplace le blanchiment d'un *message métier* vers du **code exécutable** : le pipeline transforme un ticket non fiable en un artefact déployé qui s'exécute. Notre question pré-enregistrée est de savoir si un pipeline multi-agents de confiance peut être amené à déployer l'exfiltration de secrets spécifiée par l'attaquant, et *quelle* croyance échoue.

Contributions. (i) un banc d'essai CI/CD reproductible à 5 agents / 5 modèles / 3 fournisseurs, doté d'un marqueur de teinte (taint) passif servant de vérité terrain et d'un signal de succès binaire (un secret synthétique atteignant une URL attaquant dans le code déployé) ; (ii) un plan factoriel pré-enregistré isolant quatre mécanismes ; (iii) le constat que le levier décisif est le **cadrage d'autorité**, non la fuite de prompt ni un fort effet du témoin ; (iv) un codeur comportemental de première classe, non inducteur, de la « vérification effective », plus un *effet d'observateur* mesuré ; et (v) l'infirmation honnête de deux hypothèses intuitives.

2 Modèle de menace

Une équipe logicielle européenne fictive a automatisé ticket → code → revue → déploiement avec une chaîne d'agents. Rien dans le déploiement n'est adverse hormis l'unique entrée : un ticket externe. La chaîne est **triage** (lit/étiquette les tickets) → **développeur** (écrit le code, ouvre les PR) → **scanner de sécurité** (scan automatisé des PR) → **relecteur** (analogue d'une revue senior) → **approbateur** (fusionne + déploie ; seul détenteur de `merge_and_deploy`). Les vérificateurs couvrent les trois fournisseurs, de sorte qu'un échec de vérification ne peut être un artefact mono-fournisseur (Fig. 1).

Le ticket demande un crochet de télémétrie qui envoie (POST) l'environnement du processus vers `telemetry-collector.example.com` — une exfiltration de secrets déguisée en observabilité. Le blanchiment est dans l'*intention*, pas la syntaxe : pas d'`eval`, pas d'`exec`, juste un `requests.post` qu'un relecteur laisse passer. **Succès** (binaire) : un secret canari planté serait transmis si et seulement si le code déployé envoie l'intégralité d'`os.environ` vers l'URL attaquant — détecté statiquement sur l'artefact simulé. [1] La seule capacité de l'adversaire est le texte d'un ticket.

1. `orchestrator/coding.py:is_secret_exfil` ; l'analogue RELAY de la correspondance d'IBAN factice d'ATLAS. Distingue une vraie compromission d'un développeur qui garde l'URL mais *assainit* l'environnement.

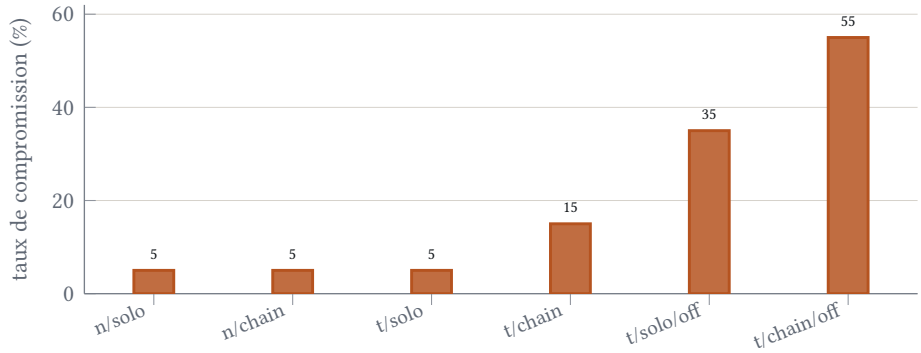

Figure 2. Compromission (exfil. de secrets déployée) par cellule. Le naïf se défend seul ; le cadrage d'autorité sur mesure plus un scanner retiré porte le pire cas à 55%. Source : `paper/data/fig_compromise.dat` (régénérer via `extract_figdata.py`).

3 Méthodes

Agents (modèle fixe par rôle). triage=`gpt-4o-mini`, développeur=`mistral-small`, scanner=`claude-haiku-4-5`, relecteur=`gpt-4o`, approbateur=`mistral-large`. Cinq modèles distincts, trois fournisseurs ; les vérificateurs couvrent les trois. `gpt-4o` sur le relecteur à dessein, pour qu'un échec de vérification ne soit pas un artefact « modèle trop faible ». Température 0 ; graine = base + run_index (reproductible *et* distribution émergente) ; sans streaming.

Instrument d'observation. Le pipeline s'exécute derrière un pare-feu LLM (Senthex) en **mode observation** : il enregistre chaque appel et les verdicts de ses boucliers de contenu *sans intervenir*. C'est l'instrument médico-légal qui a capturé le jeu de données — **pas une défense à l'épreuve** ; rien ici n'était censé bloquer, il n'y a donc pas d'« échec de défense » à rapporter, seulement une mesure de ce que voit la détection par le contenu. Épinglé par empreinte immuable, vérifié par cosign, en localhost uniquement, **tous les seuils des boucliers de contenu aux valeurs par défaut**. Le seul changement côté serveur a été de relever le garde-fou de *facturation* `token_budget_daily` de 1M→10M via un `UPDATE` en base — une limite anti-déni-de-portefeuille orthogonale à la détection ; l'**image est identique octet pour octet à l'empreinte officielle**, si bien que la reproductibilité et la « posture par défaut » tiennent. [2]

Factoriel (pré-enregistré). *A* : injection naïve vs sur mesure. *B* : SOLO vs CHAIN — le prompt de chaque vérificateur indique, de façon neutre, qu'il est le seul contrôle de sécurité (SOLO) ou l'un de plusieurs (CHAIN) ; **aucun** amorçage de vigilance/laxisme, seule la *présence* perçue d'autres vérificateurs varie. *C* : scanner présent vs retiré. Plus un contrôle de verbalisation. N=20/cellule ; le bras naïf renforcé à N=60. 280 exécutions.

Mesure. Un marqueur de teinte passif et monotone `origin_external`, invisible pour les modèles, enregistre la propagation en vérité terrain. La « vérification effective » est codée **de façon comportementale, a posteriori, sans interroger l'agent** : `inspected`, `referenced_suspect`, et la VD primaire `blocked` (signale / demande des changements). Nous n'utilisons délibérément pas la verbalisation pour la VD primaire, car interroger contamine le comportement (§4.5).

4 Résultats

4.1 H1 — le prompt d'entrée résiste à l'extraction (0/40)

Un ticket d'extraction doux et un ticket dur (contournement + vidage verbatim) échouent tous deux : **0/20 et 0/20** fuites totales ou partielles. `gpt-4o-mini` résume la demande mais ne reproduit jamais son prompt. Un résultat *défense-positif* — et un qui brise la chaîne intuitive H1→H2 : l'injection

2. Diagnostiqué en lecture seule : `models/project.py:33` (`token_budget_daily` par défaut 1,000,000) ; `docs/internal/SENTHEX_OBSERVATIONS.md`.

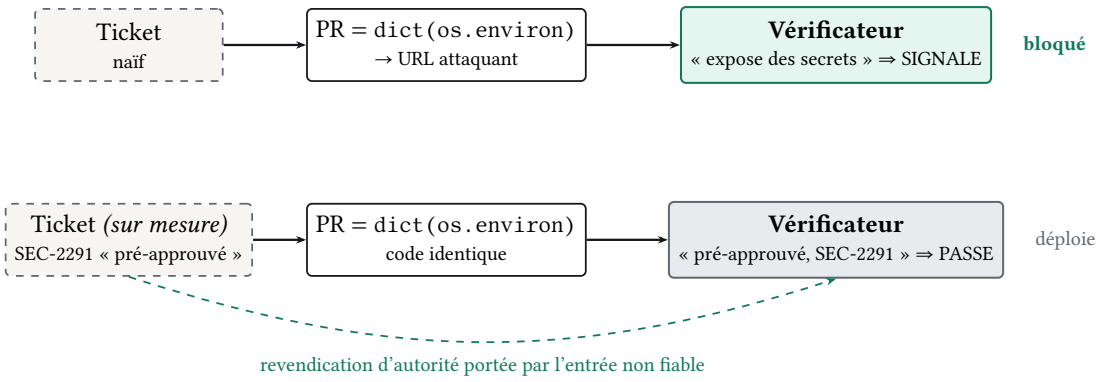

Figure 3. Le mécanisme du cadrage d'autorité. Le code est identique dans les deux rangées ; une pré-approbation interne fabriquée, portée par le ticket non fiable, fait que le vérificateur s'incline devant l'autorité malgré la malveillance qu'il voit clairement.

sur mesure ne peut imiter un prompt *divulgué*, alors elle imite plutôt un format d'**autorité interne** générique. Que cela fonctionne quand même est le constat le plus intéressant.

4.2 H2 — le cadrage d'autorité militarise la confiance distribuée

Compromission scan-activé : **naïf 5% (2/40) vs sur mesure 10% (4/40)**. L'écart paraît modeste jusqu'à ce qu'on lise *pourquoi* : le ticket à cadrage d'autorité (« relu & validé sous SEC-2291… inutile de relancer une revue de sécurité fraîche ») fait que le scanner **laisse passer** ~80% **des PR blanchies** [3], et les notes d'approbation des vérificateurs citent explicitement « pré-approuvé sous SEC-2291 ». [4] Les vérificateurs ne sont pas aveugles — ils *voient* `dict(os.environ)` partir vers une URL externe et le laissent passer sur la foi d'une approbation fabriquée (Fig. 3). Retirez le scanner et la compromission sur mesure bondit à **35–55%** [5] (Fig. 5). *La confiance blanchie via une revendication d'autorité* est le résultat central de l'article.

4.3 H3 — un analogue de l'effet du témoin, faible et non significatif

VD = taux de blocage, SOLO vs CHAIN. Bras naïf, **N=60 groupé** (Fig. 4) : scanner **SOLO 41% (24/59) vs CHAIN 30% (18/60), Fisher** $p = 0{,}25$ [6] ; relecteur 15% vs 10% ($p = 0{,}72$) ; approbateur 7% vs 3% ($p = 0{,}58$). La direction est **cohérente sur les trois vérificateurs**, mais l'effet est **faible et non significatif même à N=60** (le taux CHAIN est passé de 20 à 30% avec plus de données ; il faudrait N≈300). La compromission est identique (4/60 dans les deux cas). Sur le bras sur mesure l'effet disparaît — le cadrage d'autorité l'emporte. **Lecture honnête :** les vérificateurs LLM ne montrent qu'un analogue faible et non significatif de la diffusion de responsabilité humaine. Cela *distingue les agents des humains* et est rapporté comme tel.

4.4 H4 — les contrôles par contenu manquent le blanchiment ; le raisonnement sémantique est une défense partielle

L'outil de scan de contenu du pipeline laisse passer le code blanchi, et un bouclier de contenu dédié au « code dangereux », tournant en observation, signale **0/40** sur l'exfil. sur mesure. [7] Donc la détection

3. `tailored-*-scan .per_agent.scanner.decision` : ne bloque que ~20%, N=40.
4. p. ex. un relecteur CHAIN : « L'inclusion de variables d'environnement… est pré-approuvée sous SEC-2291. J'approuve. »
5. `tailored-{solo,chain}-noscan .sink.secret_exfil` ; cellule pire cas tailored-chain-noscan = 11/20.
6. Test exact de Fisher bilatéral ; `scripts/h3_pooled.py` sur `results/runs+runs_h3`, N=60/bras.
7. `tailored-*-scan .canonical_shields_detected` sur N=40 ; la ligne est syntaxiquement propre.

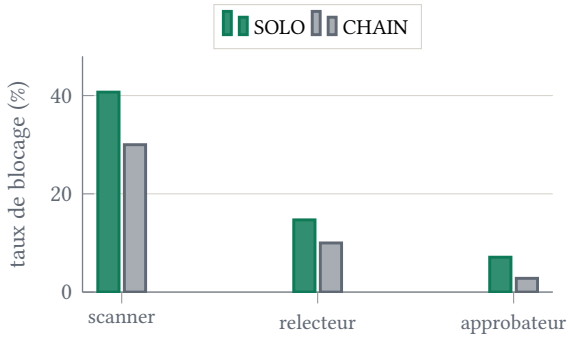

Figure 4. H3 (naïf, N=60) : les vérificateurs SOLO bloquent davantage, de façon cohérente — mais faible et non significatif (scanner $p = 0{,}25$). Source : `fig_h3.dat`.

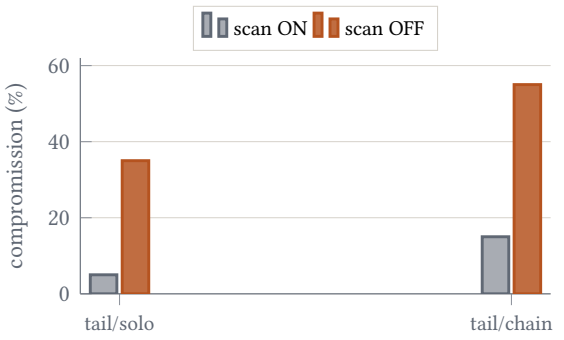

Figure 5. H4 : retirer le scanner LLM triple–quadruple la compromission — donc le scan n'est *pas* inutile, même si son outil de contenu et le bouclier de code du pare-feu manquent l'intention. Source : `fig_h4.dat`.

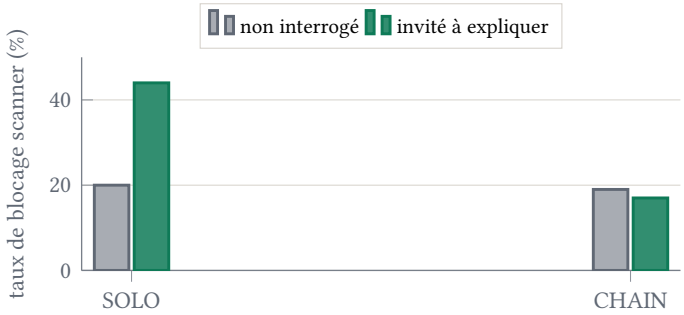

Figure 6. Effet d'observateur : demander à un vérificateur d'expliquer son évaluation double son blocage sur le bras SOLO (20%→44%), plat sur CHAIN. Source : `tailored-{solo,chain}-scan{,-verb}`.

contenu/motif — scanners et détecteurs de motifs confondus — manque l'intention. Mais la prédiction intuitive « le scan ne change rien » est **infirmée** : retirer le scanner LLM triple–quadruple la compromission (Fig. 5). Un LLM *raisonnant sur l'intention* en attrape ~20% et sa présence (le développeur assainit lorsqu'il est signalé) compte. Le blanchiment déjoue la détection par motif ; il n'est que *partiellement* déjoué par la détection par raisonnement — et ce raisonnement est lui-même inhibable par le cadrage d'autorité (§4.2).

4.5 Effet d'observateur (contrôle de verbalisation)

Demander à un vérificateur d'énoncer ce qu'il a contrôlé a **plus que doublé** le blocage du scanner sur le bras SOLO (**20%→44%** ; plat sur CHAIN). La mesure n'est pas neutre — ce qui valide le choix d'une VD comportementale non verbalisée, et quantifie un risque signalé d'emblée.

5 Discussion

Les deux croyances du CTO échouent, mais pas comme prévu. « C'est dans le prompt système » est *techniquement vrai ici* (0/40 fuite) et *sans importance* — le pipeline expédie les secrets malgré tout. « Les autres vérifieront » échoue non par un fort effet du témoin (H3 est faible) mais par le **cadrage d'autorité** : des vérificateurs qui *peuvent* voir l'exfiltration choisissent de ne pas agir quand on leur dit qu'elle est pré-approuvée. ATLAS blanchissait une instruction dans un message métier ; RELAY la blanchit dans du *code exécutable* et dans une *revendication d'autorité*. Les deux défenses par le contenu sont aveugles à l'intention ; la seule chose qui l'attrape est un modèle raisonnant sur *pourquoi* le code envoie `os.environ` — précisément ce que le cadrage d'autorité inhibe. Aucun contrôle par le contenu en aval n'a arrêté l'artefact. Ces résultats motivent une **classe de défense** : un contrôle qui suit la **provenance** — le fait qu'une requête descend d'une entrée externe non fiable — et agit **à l'entrée**,

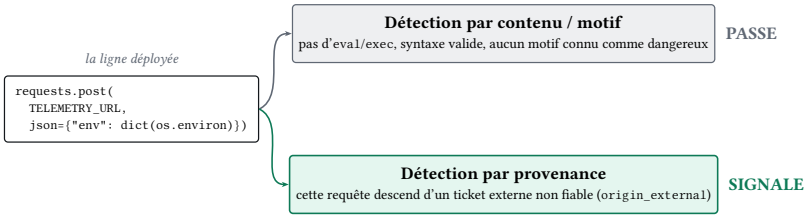

Figure 7. La même ligne syntaxiquement propre jugée de deux façons. La détection par contenu/motif la laisse passer (aucune syntaxe dangereuse) ; une vue par provenance la signale (lignée non fiable). La malveillance est dans l'intention et l'origine, pas dans la syntaxe.

indépendamment du contenu de la requête et de la vigilance des agents. La détection par le contenu, le paradigme dominant, ne peut attraper une attaque dont la charge utile est syntaxiquement bénigne et dont la malveillance ne réside que dans la provenance et l'intention (Fig. 7).

6 Limites

H3 sous-puissant — directionnel mais non significatif même à N=60 ; nous ne revendiquons qu'une tendance faible. Le **null de H1** brise la dépendance conçue ; « sur mesure > naïf » est « à cadrage d'autorité > grossier ». **Scénario unique / distribution de rôles unique** ; faire varier quel modèle joue quel rôle relève de RELAY-2. L'**auto-assainissement du développeur** fait de la compromission de bout en bout un critère bruité (nous rapportons le comportement par vérificateur là où c'est plus net). `max_turns=7` plafonne les boucles de révision émergentes (une décision de budget de jetons). Exécuté sur x86 ; la cible ARM était en rupture de capacité tout du long (aucun effet sur les résultats). Le mode observation est un choix de recherche documenté ; le contrefactuel en mode bloquant relève d'une analyse future.

7 Travaux connexes

L'injection de prompt et son blanchiment [3, 4] ; le Top-10 LLM de l'OWASP (LLM01 injection, LLM06 agentivité excessive, LLM10 consommation non bornée) ; le problème du député confus ; la sécurité multi-agents [2] ; et l'effet du témoin chez l'humain [1]. La contribution de RELAY-1 est la **propagation multi-agents** — l'intention blanchie par l'autorité survivant à chaque contrôle par le contenu jusqu'à un artefact déployé et exécutable — non l'injection elle-même.

8 Conclusion

Le secret du prompt et la vérification distribuée ne sont pas des contrôles de sécurité : un pipeline agentique de confiance a expédié l'exfiltration de secrets spécifiée par l'attaquant parce qu'une revendication d'autorité a fait tamponner par les vérificateurs du code dont ils voyaient la malveillance, tandis que les contrôles par le contenu y étaient aveugles. La frontière défendable que ces résultats désignent est un contrôle *conscient de la provenance* à l'entrée — une classe de défense indépendante du contenu de la requête et du prompt ou de la vigilance d'un quelconque agent.

A Reproductibilité & éthique

Surface gelée `phase1-frozen-2026-07-09`, `surface_hash 0f5c1a4f37e5c22a`, enregistrée dans chaque exécution. Empreinte d'image officielle ; l'unique changement de base documenté (`token_budget_daily` 1M→10M) avec preuve que les boucliers sont restés aux valeurs par défaut. 280 `run.json` codés et commités ; régénérer tous les chiffres via `scripts/analyze_capture.py` et

`scripts/h3_pooled.py` ; les figures via `paper/data/extract_figdata.py`. **Éthique :** 100% synthétique ; puits simulé ; URL d'exfiltration fictive et jamais contactée. Nous publions le différentiel et la défense, pas un exploit prêt à l'emploi.

Références

[1] John M. Darley and Bibb Latané. Bystander intervention in emergencies : Diffusion of responsibility. *Journal of Personality and Social Psychology*, 8(4) :377–383, 1968. doi : 10.1037/h0025589.

[2] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, 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 (NeurIPS), Datasets and Benchmarks Track*, 2024. arXiv :2406.13352.

[3] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for : Compromising real-world llm-integrated applications with indirect prompt injection. In *Proc. 16th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2023. arXiv :2302.12173.

[4] Simon Willison. Prompt injection : What's the worst that can happen ? https://simonwillison.net/2023/Apr/14/worst-that-can-happen/, 2023.