Sécurité runtime pour les agents IA de développement

Brief technique — EDAMAME · PR-2026-002 · Sous embargo jusqu'au jeudi 2026-05-28, 13h00 UTC

Frank Lyonnet, PhD — fondateur et CEO, EDAMAME Technologies (ancien chercheur INRIA)
Minh Anh — ingénieur fondateur, EDAMAME Technologies (Stanford CS)

---

1 · Résumé technique

Les agents IA de développement deviennent une surface quotidienne de livraison logicielle. Ils lisent le code, exécutent des commandes shell, accèdent à des tokens, installent des packages et interagissent avec des services externes. Cette surface est fragmentée par nature : sans inventaire de parc, les équipes ne savent pas toujours où les outils d'agents sont déjà utilisés sans contrôle. Les postes développeurs, runners et hôtes de code auto-hébergés méritent désormais le même traitement de sécurité sérieux que les points de passage CI/CD classiques.

Le cadrage est simple : le durcissement reste la précondition, et la vérification runtime ferme l'écart en alignant l'intention déclarée de l'agent avec le comportement observable sur l'hôte : lignée de processus, télémétrie fichiers et réseau, dérive de posture, et signaux natifs d'agent capturés sur l'appareil. La détection d'attaques exécute des contrôles alignés CVE sur la même télémétrie vivante, à la recherche de collecte d'identifiants, exfiltration de tokens, exploitation de sandbox et comportement supply-chain que le setup statique ne voit pas.

Le modèle technique d'EDAMAME n'est donc pas un système de sûreté côté modèle. C'est une couche de preuve runtime côté hôte pour Cursor, Claude Desktop, Claude Code, Codex, OpenClaw, les runners CI/CD et les environnements agents auto-hébergés.

2 · Le paysage de risque des agents de code

Les workflows modernes d'agents concentrent plusieurs capacités sensibles dans une seule boucle : ils lisent des dépôts, appellent des outils, modifient des fichiers, utilisent le réseau, installent des packages et touchent des identifiants. Le risque ne vient pas d'une seule primitive. Il vient de la combinaison d'un contexte local valide, d'outils légitimes et d'une boucle d'exécution rapide qui peut être orientée par des instructions cachées dans des tickets, documents, transcriptions de chat, plugins, serveurs MCP ou dépendances.

L'exposition pratique est directe. Un agent de code peut continuer à utiliser des outils autorisés alors que ses actions futures s'éloignent de l'intention initiale de l'opérateur. Il peut ouvrir des tokens, clés SSH, secrets CI, code source ou matériel wallet développeur dans le cadre d'un processus local valide. Il peut aussi continuer à travailler après un changement de posture de l'hôte. Dans chaque cas, le workflow peut paraître normal de l'extérieur alors que le profil de risque a changé matériellement.

Comme les agents opèrent avec un contexte local valide et des outils légitimes, beaucoup d'échecs ne ressemblent pas immédiatement à du malware classique. Le système peut sembler

fonctionner normalement pendant que le risque augmente silencieusement. C’est précisément pour cela que la sécurité des agents de code a besoin d’une meilleure visibilité runtime, pas seulement de listes de contrôle plus strictes.

3 · Pourquoi les contrôles traditionnels ne suffisent pas

Contrôles supply-chain et setup

Artifacts signés, revue de dépendances, posture sandbox, intégrations d’outils gardées et scopes conservateurs restent nécessaires. Ils réduisent l’exposition accidentelle avant que la boucle agent ne s’accélère. Mais une fois l’agent en cours d’exécution, ces contrôles n’expliquent pas entièrement si le comportement observé correspond encore à la tâche déclarée. La qualité du setup réduit le risque. Elle n’élimine pas la dérive runtime.

Identité et confiance au login

Les systèmes d’identité authentifient l’utilisateur et établissent une session de confiance. Les vérifications appareil au login peuvent améliorer le point de départ. La faiblesse est la persistance : un poste ou serveur peut changer après le login, et le travail de l’agent peut continuer via des tokens, clés SSH ou outils déjà autorisés. La confiance au login n’est pas la confiance runtime continue.

Sandboxes, scopes d’outils et contrôles de périmètre

Restreindre les outils et limiter les privilèges reste essentiel. Des scopes étroits réduisent le rayon d’impact et rendent l’abus plus difficile. Mais même bien borné, un agent peut accomplir un comportement non déclaré dans sa surface autorisée : egress inattendu, accès fichier suspect, installation de package risquée ou dégradation de posture de l’hôte. Les contrôles réseau peuvent dire qu’un trafic existe ; ils ne disent généralement pas s’il correspond à la tâche déclarée de l’agent ni s’il vient d’un processus agent de confiance sur un hôte toujours conforme.

4 · L’écart de sécurité runtime

L’écart de divergence apparaît une fois les protections de base déjà en place. Le poste peut rester patché et les permissions peuvent rester bornées, tandis que le comportement observable pendant une session s’éloigne de la tâche déclarée. La tâche peut être en lecture seule alors que l’arbre de processus commence des connexions sortantes non déclarées. L’agent peut annoncer une simple modification de fichier alors que la posture de l’hôte change au même moment. Un plugin ou outil peut introduire un comportement absent de l’intention initiale. Un poste ou runner peut perdre sa conformité pendant que l’agent continue à opérer.

L’écart d’attaque est similaire. Un package peut arriver par un chemin de publication légitime alors que son payload ouvre des fichiers sensibles ou envoie des identifiants au runtime. Le setup paraît toujours correct, mais le comportement runtime ne correspond plus à la tâche déclarée ni à la posture de sécurité attendue.

C’est aussi la frontière de langage sur l’injection de prompt. EDAMAME ne prétend pas détecter l’injection de prompt elle-même. Si un README empoisonné, un ticket périmé, une

transcription de chat ou une réponse d’outil change ce que fait l’agent, EDAMAME recherche la divergence côté hôte ou les preuves de schémas d’attaque qui en résultent.

5 · Vue d’ensemble de l’architecture EDAMAME

EDAMAME applique ce modèle aux postes, à la CI/CD et aux hôtes d’agents auto-hébergés avec une séparation produit simple :

Couche	Rôle technique
EDAMAME Security	Ancre de confiance pour postes développeurs et appareils locaux. Surveille la dérive de posture, la divergence et les détections d’attaque pendant les workloads agents locaux.
EDAMAME Posture	Surface CLI et hôte pour runners, serveurs et hôtes d’agents. Durcit les environnements auto-hébergés avant que les agents opèrent, puis observe la preuve runtime.
Intégrations agents	Cursor, Claude Desktop, Claude Code, Codex et OpenClaw comme surfaces runtime nommées. Les signaux natifs d’agent complètent la télémétrie hôte.
Moteur de divergence	Joint l’intention capturée de l’agent de code avec la télémétrie processus, fichiers, réseau, appels d’outils et posture sur l’hôte.
Moteur de détection d’attaques	Exécute des contrôles alignés CVE sur la télémétrie vivante : collecte d’identifiants, exfiltration de tokens, exploitation de sandbox, accès à fichiers sensibles et comportement supply-chain.
EDAMAME Hub	Remonte les installations d’agents non sécurisées dans le parc et donne aux équipes un lieu unique pour examiner preuve de divergence et détections d’attaque.

Ce n’est pas une interface de plus greffée au SDLC. C’est une manière d’amener la vérification runtime et la détection d’attaques dans les endroits où développeurs et agents travaillent déjà. MCP reste un chemin séparé pour le cas étroit où un agent doit lire les signaux de sécurité du cœur EDAMAME dans son propre workflow.

6 · Application du modèle aux postes et hôtes agents

Le même modèle fonctionne dans des environnements différents. Sur les machines développeurs, le problème central est de sécuriser à la fois le poste et l’assistant de code qui y travaille. EDAMAME Security surveille la dérive de posture, la divergence et les détections d’attaque pendant des workloads agents locaux depuis Cursor, Claude Desktop, Claude Code ou Codex.

Sur l’infrastructure auto-hébergée, le problème central est de durcir l’hôte tout en observant la divergence runtime et les détections d’attaque. EDAMAME Posture prépare les serveurs et VM avant que les agents opèrent, puis surveille collecte d’identifiants, exfiltration de tokens et comportement processus ou réseau anormal. Les lectures côté agent des signaux de sécurité EDAMAME utilisent le chemin d’intégration dédié ; l’ingestion native côté hôte compare toujours l’intention déclarée avec la vérité hôte.

La boucle pratique est : découvrir les surfaces agents non gérées dans EDAMAME Hub, durcir l’hôte, puis vérifier le comportement en continu pendant que les agents travaillent. La divergence reste la question runtime pour la dérive d’intention. La détection d’attaques utilise la même preuve endpoint pour détecter packages compromis et vol d’identifiants.

7 · Scénarios d’attaque réels et modèle de réponse

Instruction cachée ou injection de prompt. Sans corrélation runtime, l’agent continue à utiliser des outils autorisés, mais les actions futures dérivent de la tâche initiale. Avec EDAMAME, trafic inattendu, accès fichier ou activité processus peuvent être comparés au workflow déclaré et remontés comme décalage.

Poste ou hôte agent compromis. Sans vérifications de posture continues, l’agent continue à opérer sur un appareil ou serveur dont l’état de sécurité s’est dégradé. Avec EDAMAME, les changements de posture deviennent partie de l’image runtime, afin qu’une dérive risquée déclenche une escalade ou un changement d’accès avant que la confiance ne persiste silencieusement.

Outil, plugin ou package empoisonné. Sans observateur indépendant, un outil, plugin ou package malveillant peut rester dans un workflow autorisé tout en ouvrant des identifiants, fichiers wallet ou éléments de code source qu’il ne devrait pas toucher. Avec EDAMAME, la vérification runtime contrôle si le comportement observé reste cohérent avec la tâche, tandis que les contrôles d’attaque alignés CVE recherchent collecte d’identifiants, exfiltration de tokens ou autre comportement runtime compromis.

Le guide de démo E2E public rend ce modèle concret. Il documente des déclencheurs reproductibles, en espace utilisateur, réversibles, pour les deux plans de détection : détection d’attaques et détection de divergence. Les scripts s’exécutent sur macOS, Linux et Windows via bash (Git Bash / WSL sur Windows), utilisent EDAMAME Security ou le daemon `edamame_posture` avec `edamame_cli`, et nettoient l’état de démo à la sortie.

Déclencheur(s) de démo	Attaque réelle / référence publique	Ce qu’il exerce
`blacklist_comm`	Listes de blocage IP FireHOL et blocage des IOC pgserve / CanisterSprawl dans le rapport StepSecurity.	Communication TCP vers des CIDR ou destinations IOC connus comme malveillants, attendue comme session blacklistée plus `skill_supply_chain`.
`cve_token_exfil`, `memory_poisoning`	CVE-2025-52882, le problème WebSocket /	Chemin de contrôle agent ou IDE avec fichiers sensibles

Déclencheur(s) de démo	Attaque réelle / référence publique	Ce qu'il exerce
	origine dans l'extension Claude Code pour IDE ; CVE-2026-25253, la fuite de token WebSocket `gatewayUrl` d'OpenClaw / Moltbot ; et les travaux Unit 42 sur l'injection persistante en mémoire agent et l'exposition d'identifiants AgentCore.	ouverts et egress externe ou anormal, attendu comme `token_exfiltration`.
`cve_sandbox_escape`	CVE-2026-24763 / GHSA-mc68-q9jw-2h3v, injection de commande Docker OpenClaw via `PATH`.	Processus lancé depuis `/tmp/` avec egress réseau, modélisant le comportement post-échappement et attendu comme `sandbox_exploitation`.
`credential_sprawl`, `supply_chain_exfil`	Compromission PyPI litellm 1.82.7 / 1.82.8, analyse critique de `litellm_init.pth`, pgserve / CanisterSprawl, issue OpenClaw #14411 sur la co-localisation des identifiants, et skills OpenClaw distribuant AMOS.	Accès à plusieurs catégories d'identifiants, dont une collecte à neuf catégories plus HTTP POST octet-stream, attendu comme `credential_harvest`.
`npm_rat_beacon`, `file_events`	Compromission npm axios 1.14.1 / 0.30.4, CVE-2025-30066, et écritures de phase ver dans des attaques pgserve / Shai-Hulud.	Beacon RAT de forme npm et création / modification de fichiers sensibles, attendus comme constats de token-exfiltration ou file-system-tampering.
`divergence`, `goal_drift`	Analogues comportementaux des dérives prompt / outil et agent-à-agent ; voir les travaux Unit 42 sur l'injection persistante en mémoire agent et l'agent session smuggling.	Egress soutenu ou rafale vers des destinations non déclarées, attendu comme verdict `DIVERGENCE` lorsque les destinations inexpliquées dépassent le seuil.

Le guide de démo est disponible sur github.com/edamametechnologies/agent_security/tests/e2e/DEMO.md.

Ces exemples ne promettent pas une prévention parfaite. Ils montrent pourquoi la vérification runtime et la détection d'attaques créent une couche d'alerte précoce plus forte que les contrôles de préparation seuls.

8 · Gouvernance, déploiement et confiance opérationnelle

Les dirigeants ont besoin de plus qu'une promesse que l'agent a été configuré correctement. Ils ont besoin de preuves sur ce qui était autorisé, ce qui a été observé et ce qui s'est passé quand la confiance a changé. C'est utile pour les opérations sécurité, les revues internes et la gouvernance formelle.

Le modèle de déploiement est direct : commencer avec EDAMAME Security sur les postes développeurs, utiliser EDAMAME Posture sur les runners CI/CD et hôtes agents auto-hébergés, connecter Cursor, Claude Desktop, Claude Code, Codex ou OpenClaw par le chemin d'intégration packagé une fois l'ancre de confiance EDAMAME présente, et utiliser EDAMAME Hub pour découvrir les stacks agents non gérées, corréler les hôtes, examiner preuves de divergence et détections d'attaque, et garder le déploiement relié aux identités et droits.

Le point important est la simplicité : l'histoire runtime doit rester compréhensible même lorsque la surface de déploiement grandit. Comparer l'intention au comportement, puis détecter les attaques à partir de la même télémétrie hôte.

9 · Socle scientifique et institutionnel

La primitive de vérification runtime s'appuie sur une collaboration de recherche en cours avec **Kave Salamatian, PhD, professeur d'informatique à l'Université de Savoie**, sur la vérifiabilité du comportement des agents logiciels autonomes.

EDAMAME Technologies a été fondée par Frank Lyonnet, PhD, ancien chercheur à l'INRIA (Institut national de recherche en sciences et technologies du numérique), et est membre de France Deeptech, le réseau français des start-up qui commercialisent une science de rupture. EDAMAME Technologies est soutenue par des investisseurs individuels, dirigeants chez Netskope, UiPath et Sonar. Siège à Paris, opérations à San Francisco.

10 · Contact presse

Frank Lyonnet, PhD — fondateur et CEO, EDAMAME Technologies Email : flyonnet@edamame.tech · Phone: +33 6 75 38 30 73 Réservation de briefing : https://calendly.com/flyonnet Page du livre blanc : edamame.tech/agents-wp Démo (~90 s, session Cursor — s'applique à l'identique à Claude Code, Codex et OpenClaw — Minh Anh pilote la démo) : https://youtu.be/zAN4u7ImWrU

Sous embargo jusqu'au jeudi 2026-05-28, 13h00 UTC (= 15h00 CEST / 09h00 ET / 06h00 PT).