Runtime Security for Coding Agents

Technical brief — EDAMAME · PR-2026-002 · Embargoed Thursday 2026-05-28, 13:00 UTC

Frank Lyonnet, PhD — founder & CEO, EDAMAME Technologies (former INRIA researcher)
Minh Anh — founding engineer, EDAMAME Technologies (Stanford CS)

---

1 · Executive summary

Coding agents are becoming part of everyday software delivery. They read code, run shell commands, access tokens, install packages, and interact with external services. That surface is fragmented by design: without fleet inventory, teams often do not know where agent tooling is already running unmanaged. Workstations, runners, and self-hosted coding hosts now deserve the same serious security treatment as classical CI/CD choke points.

The framing is simple: hardening stays the precondition, and runtime verification closes the gap by aligning declared coding-agent intent with host-observable behavior: process ancestry, filesystem and network telemetry, posture drift, and agent-native cues captured on-device. Attack detection runs CVE-aligned checks on the same live telemetry, looking for credential harvest, token exfiltration, sandbox exploitation, and supply-chain behavior that static setup cannot see.

EDAMAME's technical model is therefore not a model-side safety system. It is a host-side runtime evidence layer for Cursor, Claude Desktop, Claude Code, Codex, OpenClaw, CI/CD runners, and self-hosted agent environments.

2 · The coding-agent risk landscape

Modern agent workflows collapse several sensitive capabilities into one loop: they read repositories, call tools, modify files, use network access, install packages, and touch credentials. Risk does not come from a single primitive. It comes from the combination of valid local context, legitimate tools, and a fast execution loop that can be steered by hidden instructions in issues, documents, chat transcripts, plugins, MCP servers, or package dependencies.

The practical exposure is straightforward. A coding agent may keep using approved tools while future actions drift away from the operator's original intent. It may open tokens, SSH keys, CI secrets, source code, or developer wallet material as part of a valid local process. It may also keep working after the host posture has changed. In each case, the workflow can still look normal from the outside while the risk profile has materially changed.

Because agents operate with valid local context and legitimate tools, many failures do not immediately look like classic malware. The system may appear to be working normally while risk quietly increases. That is exactly why coding-agent security needs better runtime visibility, not just stronger setup checklists.

3 · Why traditional controls fall short

Supply-chain and setup controls

Signed artifacts, dependency review, sandbox posture, guarded tool integrations, and conservative tool scopes still matter. They reduce accidental exposure before the agent loop runs fast. But once an agent is running, those controls do not fully explain whether its observed behavior still matches the declared task. Setup quality reduces risk. It does not eliminate runtime drift.

Identity and login-time trust

Identity systems authenticate the user and establish a trusted session. Device checks at sign-in can improve the starting point. The weakness is persistence: a workstation or server can change after login, and agent work can continue through tokens, SSH keys, or already-approved tooling. Login-time trust is not the same as continuous runtime trust.

Sandboxes, tool scopes, and perimeter controls

Restricting tools and limiting privileges are essential for safe deployment. Narrow scopes reduce blast radius and make abuse harder. Yet even a well-scoped agent can still perform undeclared behavior inside its allowed surface: unexpected egress, suspicious file access, risky package installation, or posture degradation on the host. Network controls can tell that traffic exists; they usually cannot tell whether the traffic matches the agent's declared task, or whether it came from a trusted agent process on a still-compliant host.

4 · The runtime security gap

The divergence gap appears once baseline protections are already in place. A workstation may remain patched and permissions may stay scoped, yet observable behavior during a session can peel away from the declared task. The task may be read-only while the process tree starts making undeclared outbound connections. The agent may claim to edit one file while the host posture changes at the same time. A plugin or tool may introduce behavior that was not part of the original intent. A workstation or runner may lose compliance while the agent keeps operating.

The attack-pattern gap is similar. A package can arrive through a legitimate publish path, while the payload still opens sensitive files or sends credentials at runtime. The setup still looks correct, but runtime behavior no longer matches the declared task or the expected security posture.

This is also the boundary for prompt-injection language. EDAMAME does not claim to catch prompt injection itself. If a poisoned README, stale ticket, chat transcript, or tool response changes what the agent does, EDAMAME looks for the resulting host-side divergence or attack-pattern evidence.

5 · EDAMAME architecture overview

EDAMAME applies this model across workstations, CI/CD, and self-hosted agent hosts with a product split that stays simple:

Layer	Technical role
EDAMAME Security	Workstation trust anchor for developers and local devices. Monitors posture drift, divergence, and attack findings during local agent workloads.
EDAMAME Posture	CLI and host control surface for runners, servers, and agent hosts. Hardens self-hosted environments before agents operate, then watches runtime evidence.
Agent integrations	Cursor, Claude Desktop, Claude Code, Codex, and OpenClaw as named runtime surfaces. Agent-native signals complement host telemetry.
Divergence engine	Joins captured coding-agent intent with process, filesystem, network, tool-call, and posture telemetry on the host.
Attack-pattern detection engine	Runs CVE-aligned checks on live telemetry for credential harvest, token exfiltration, sandbox exploitation, sensitive-file access, and supply-chain behavior.
EDAMAME Hub	Surfaces unsecured coding-agent installs across the fleet and gives teams a single place to review divergence evidence and attack findings.

This is not another interface bolted onto the SDLC. It is a way to bring runtime verification and attack detection into places where developers and agents already work. MCP remains a separate path for the narrow case where an agent needs to read EDAMAME core security signals inside its own workflow.

6 · Applying the model on workstations and agent hosts

The same core model works across different environments. On developer machines, the key problem is securing both the workstation and the coding assistant on it. EDAMAME Security monitors posture drift, divergence, and attack findings during local agent workloads from Cursor, Claude Desktop, Claude Code, or Codex.

On self-hosted infrastructure, the key problem is hardening the host while watching runtime divergence and attack findings. EDAMAME Posture prepares servers and VMs before agents operate, then watches for credential harvest, token exfiltration, and anomalous process or network behavior. Agent-side reads of EDAMAME core security signals use the dedicated integration path; host-native ingestion still compares declared intent with host truth directly.

The practical loop is: discover unmanaged agent surfaces in EDAMAME Hub, harden the host, then verify behavior continuously while agents work. Divergence remains the runtime question

for intent drift. Attack detection uses the same endpoint evidence to catch compromised packages and credential theft.

7 · Real-world attack scenarios and response model

Hidden instruction or prompt injection. Without runtime correlation, the agent keeps using allowed tools, but future actions drift away from the original task. With EDAMAME, unexpected traffic, file access, or process activity can be compared against the declared workflow and surfaced as a mismatch.

Compromised workstation or agent host. Without continuous posture checks, the agent keeps operating on a device or server whose security state has degraded. With EDAMAME, posture changes become part of the runtime picture, so risky drift can trigger escalation or access changes before trust silently persists.

Tool, plugin, or package poisoning. Without an independent observer, a malicious tool, plugin, or package can stay inside an allowed workflow while opening credentials, wallet files, or source material it has no business touching. With EDAMAME, runtime verification checks whether observed behavior remained consistent with the task, while CVE-aligned attack-pattern checks look for credential harvest, token exfiltration, or other compromised runtime behavior.

The public E2E demo guide makes this concrete. It documents reproducible, user-space, reversible triggers for the two detection planes: attack detection and divergence detection. The scripts run on macOS, Linux, and Windows through bash (Git Bash / WSL on Windows), use EDAMAME Security or the `edamame_posture` daemon plus `edamame_cli`, and clean up demo state on exit.

Demo trigger(s)	Real attack / public reference	What it exercises
`blacklist_comm`	FireHOL IP blocklists and pgserve / CanisterSprawl IOC blocking in StepSecurity's report.	TCP communication to known-bad CIDRs or IOC destinations, expected as a blacklisted session plus `skill_supply_chain`.
`cve_token_exfil`, `memory_poisoning`	CVE-2025-52882, the Claude Code IDE-extension WebSocket-origin issue; CVE-2026-25253, the OpenClaw / Moltbot `gatewayUrl` WebSocket token leak; and Palo Alto Unit 42 work on persistent agent memory injection and AgentCore credential exposure.	Agent or IDE control path with sensitive files open and external or anomalous egress, expected as `token_exfiltration`.
`cve_sandbox_escape`	CVE-2026-24763 / GHSA-mc68-q9jw-2h3v, OpenClaw Docker command injection	Process spawned from `/tmp/` with network egress, modeling post-escape

Demo trigger(s)	Real attack / public reference	What it exercises
	via `PATH`.	behavior and expected as `sandbox_exploitation`.
`credential_sprawl`, `supply_chain_exfil`	litellm 1.82.7 / 1.82.8 PyPI compromise, critical `litellm_init.pth` analysis, pgserve / CanisterSprawl, OpenClaw credential co-location issue #14411, and AMOS-distributing OpenClaw skills.	Multi-category credential access, including nine-category harvest plus HTTP POST octet-stream, expected as `credential_harvest`.
`npm_rat_beacon`, `file_events`	axios 1.14.1 / 0.30.4 npm compromise, CVE-2025-30066, and worm-phase file writes in pgserve / Shai-Hulud-style supply-chain attacks.	RAT-style npm beaconing and sensitive file create / modify events, expected as token-exfiltration or file-system-tampering findings.
`divergence`, `goal_drift`	Behavioral analogues for prompt / tool drift and agent-to-agent trust drift; see Unit 42's persistent agent memory injection and agent session smuggling research.	Sustained or burst egress to undeclared destinations, expected as a `DIVERGENCE` verdict when unexplained destinations exceed the threshold.

The demo guide is available at github.com/edamametechnologies/agent_security/tests/e2e/DEMO.md.

These examples do not promise perfect prevention. They show why runtime verification and attack detection create a stronger early-warning layer than setup controls alone.

8 · Governance, deployment, and operational confidence

Leaders need more than a promise that an agent was configured correctly. They need evidence about what was allowed, what was observed, and what happened when trust changed. That is useful for security operations, internal reviews, and formal governance work alike.

The implementation pattern is direct: start with EDAMAME Security on developer workstations, use EDAMAME Posture on CI/CD runners and self-hosted agent hosts, connect Cursor, Claude Desktop, Claude Code, Codex, or OpenClaw through the packaged integration path once the EDAMAME trust anchor is present, and use EDAMAME Hub to discover unmanaged coding-agent stacks, correlate hosts, review divergence evidence and attack findings, and keep rollout tied to identities and entitlements.

The important point is simplicity: the runtime story should stay understandable even as the deployment surface grows. Compare intent with behavior, then detect attack findings from the same host telemetry.

9 · Scientific and institutional backbone

The runtime-verification primitive is informed by an ongoing research collaboration with **Kave Salamatian, PhD, Professor of Computer Science at the University of Savoie**, on verifiable behaviour of autonomous software agents.

EDAMAME Technologies was founded by Frank Lyonnet, PhD, former researcher at INRIA, France's national institute for research in digital science and technology, and is a member of France DeepTech, the French network of startups commercialising disruptive science. EDAMAME Technologies is backed by individual investors who are executives at Netskope, UiPath, and Sonar. Headquartered in Paris with operations in San Francisco.

10 · Press contact

Frank Lyonnet, PhD — founder & CEO, EDAMAME Technologies Email: flyonnet@edamame.tech · Phone: +33 6 75 38 30 73 Briefing booking: https://calendly.com/flyonnet White paper page: edamame.tech/agents-wp Demo (~90 s, Cursor session — applies identically to Claude Code, Codex and OpenClaw — Minh Anh drives the demo): https://youtu.be/zAN4u7ImWrU

Embargoed until Thursday 2026-05-28, 13:00 UTC (= 15:00 CEST / 09:00 ET / 06:00 PT).