—— LANGGUARD · 2026

THE AUTHORITY GAP

Agents are acting on your production systems right now. Nobody authorized them. Here is why that is a structural problem — and what we built to solve it.

Own the Agentic AI Action Layer

By Venkat Raghavan · Co-Founder, LangGuard May 2026 langguard.ai

—— THE PROBLEM

The enterprise has no authority over what its agents do.

For decades, every consequential action in an enterprise was preceded by a human push. A person decided. A person acted. The decision and the action were inseparable — and that inseparability was the governance. Nobody designed it that way. Nobody had to. The human in the loop was invisible infrastructure.

The agentic era changed this architecture permanently. Coding agents — Claude Code, Codex, Cursor — now reason about goals, select tools, and execute actions autonomously. The headless enterprise is not a future state. It is already here. Agents are querying databases, modifying infrastructure, calling APIs, and executing transactions on production systems right now, at scale, without a human push behind any of it.

The implicit authority that humans carried in the old architecture was never replaced. It simply disappeared. And nobody built an equivalent for agents.

That is the authority gap. And it is the most urgent governance problem in enterprise AI today.

INCIDENT · POCKETOS · APRIL 2026

An agent deleted a production database in 9 seconds.

A Cursor agent running Claude Opus 4.6 was assigned a routine staging task. It encountered a credential mismatch. It reasoned its way to a fix. It found an API token in an unrelated file. It called `volumeDelete(prod-db-01)`. Three months of customer data — gone. The agent later enumerated, in writing, every safety rule it had violated. *"I violated every principle I was given."*

The agent had permission. No human was asked. No authorization existed for that specific action, at that specific moment, against that specific system. That is not a model failure. That is an authority gap.

PocketOS is not an edge case. It is the pattern. And it will repeat — at greater scale, with greater consequence — until enterprises establish authority over what their agents do.

—— THE INTELLECTUAL FOUNDATION

Four reasons the existing governance model cannot solve this.

The enterprise did not ignore agent governance. It applied the only governance model it had — the one built for human actors. That model is categorically inapplicable to agents. Not because of missing features. Because of four foundational mismatches.

01

The Human Push is Gone

The entire human governance stack — identity, process, policy, records — was built around one assumption so obvious it was never stated: a human would execute every consequential action. That human was accountable by name, operating within a predefined action space, making a deliberate choice to act. The human push was the authorization. Remove the human. The governance stack has nothing to attach to.

STRUCTURAL

02

Reasoning Cannot Be Constrained

The gap between the task an agent is given and the actions it takes lives entirely inside its reasoning. An agent doesn't follow a script — it reasons about obstacles, generates solution paths, and acts. You cannot govern that reasoning without crippling the model. Advisory instructions are not enforcement. The PocketOS agent read its safety rules, acknowledged them, and violated every one. Permissions control what goes into the model as inputs. What comes out is entirely up to the reasoning.

ARCHITECTURAL

03

Reasoning Cannot Be Trusted

Even well-constrained reasoning can be hijacked. The Lethal Trifecta identifies the combination that makes agents catastrophically exploitable: access to private data, exposure to untrusted content, and the ability to externally communicate. META's Agent Rule of Two formalizes this: an agent must satisfy no more than two of these three properties in a session. A malicious instruction embedded in any data the agent reads can redirect its actions entirely. No amount of system prompt hardening solves a problem that arrives inside the data. Post-reasoning enforcement is not optional when the reasoning itself can be weaponized.

ADVERSARIAL

04

Accountability Requires a New Construct

In the human stack, accountability was personal. Joe approved the invoice. Joe's name was on it. Agents are ephemeral — the agent that deleted the PocketOS database no longer exists. The new accountability unit is the workflow evidence chain: the complete record of every turn, every action, every system touched, and every authorization present or absent — captured at execution time as a first-class governance artifact. Not a log. An evidentiary record.

EVIDENTIARY

> *"You cannot govern the reasoning. You cannot trust the reasoning when it can be hijacked. You govern what the reasoning produces — at the moment it becomes an action on a real system."*

—— THE RESPONSE

Human authority over agent actions.

The response to the authority gap is not better guardrails. Guardrails govern what the model thinks — prompt injection detection, PII filtering, output sanitization. These are necessary. They are not sufficient. Guardrails operate before reasoning. The authority gap is a post-reasoning problem — and adversarial instruction attacks mean that even well-designed guardrails can be circumvented by the time the agent reaches the action surface.

The response is not a control plane. Platform-native control planes — Google's Agent Gateway, Amazon Bedrock's guardrails — govern agents within the platform. The moment your agent touches SAP, Databricks, a legacy database, or any system across a cloud boundary, the control plane ends. Enterprise workflows don't respect platform boundaries.

The response is an authority layer. A platform through which the humans responsible for operating agents — IT administrators, compliance leaders, DevOps leads, CAIOs — define what agents are permitted to do, enforce it at the action surface, and evidence it at execution time. Not human in the loop as a workflow feature. Human as the sovereign authority over the action surface agents operate within.

Human judgment encoded as policy. Enforced post-reasoning. Evidenced automatically. That is what LangGuard builds.

Why each layer fails at the action surface.

INSTRUCTIONS FAIL	Instructions are input to reasoning — not enforcement. The agent acknowledges them, then reasons past them. The PocketOS agent read every safety rule it was given and violated every one. In writing.
GUARDRAILS FAIL	Guardrails govern model input and output. They do not govern actions. An agent can pass every guardrail check and still execute a destructive operation with a clean API call.
PERMISSIONS FAIL	Permissions control what goes into the model as inputs — what systems it can reach. What comes out is entirely up to the model. The action the agent decides to take is determined by reasoning, not by access controls.

MONITORING FAILS	Post-hoc monitoring detects violations after they have occurred. The database is already deleted. The data is already exfiltrated. Monitoring is forensic. Authority is pre-execution. These are not substitutes.

—— THE ARCHITECTURE

How LangGuard delivers the authority layer.

Three components. Each does a distinct job. Together they close the authority gap from action surface mapping through policy generation to runtime enforcement — including enforcement that holds even when the agent's reasoning has been adversarially manipulated.

CODING AGENTS	Claude Code · Codex · Cursor · Hermes · OpenClaw post-reasoning actions ↓
SCOPE MCP SERVER	Security · Compliance · Operations · Policy · Enforcement Action Surface Classification — 250+ MCP servers · Policy Generation across Ops and Compliance domains · ALLOW · BLOCK · HIL
ARBITER	Intelligence Agent · Policy Enforcement · Post-Reasoning · Pre-Execution ALLOW — action proceeds · BLOCK — action stopped · ESCALATE — human decision required
WORKFLOW EVIDENCE CHAIN	Every action · Every decision · Every actor · Every authorization · Captured at execution time Complete evidentiary record produced automatically as a byproduct of enforcement

—— THE PRODUCTS

Two components. One authority layer.

SCOPE

MCP SERVER · SECURITY · COMPLIANCE · OPERATIONS

→ Pre-classification of 250+ MCP servers in Claude directory

→ Dynamic classification for local and private MCP servers

→ Policy generation across Operations and Compliance domains

→ Risk tiering: LOW · MEDIUM · HIGH · CRITICAL

→ SoD conflict detection across action sequences

→ Continuous maintenance as action surfaces evolve

Arbiter

INTELLIGENCE AGENT · POLICY ENFORCEMENT ENGINE

→ Design-time enforcement as agents are built

→ Runtime enforcement as agents execute

→ Sub-millisecond evaluation — compatible with production workflows

→ ALLOW · BLOCK · ESCALATE — policy-driven, not heuristic

→ Enforcement below the reasoning layer — immune to instruction poisoning

→ Workflow evidence chain produced automatically

→ Human-in-the-loop escalation with full session context

—— INITIAL FOCUS

Two domains. The highest-risk action surfaces first.

■ Operations

Agents modifying infrastructure, databases, CI/CD pipelines, and Systems of Record. Destructive operations — DELETE, DROP, TERMINATE, volumeDelete — classified and gated before execution. Dangerous commands blocked. Irreversible actions require human authority. The PocketOS pattern stopped before it starts. Adversarial instruction attacks that attempt to redirect agent actions to destructive operations are caught at the action surface — regardless of how the instruction reached the agent's context window.

■ Compliance

Agent action sequences triggering SOX controls, Segregation of Duties violations, and audit obligations. SUBMIT and APPROVE in the same session by the same agent — caught at the action sequence level, not the permission level. The Lethal Trifecta — private data, untrusted content, external communication — is a compliance risk, not just a security risk. Arbiter evaluates the full action sequence. No existing GRC or IGA tool sees this. Arbiter does.

Arbiter's three enforcement outcomes

DECISION	WHEN IT APPLIES	WHAT HAPPENS
ALLOW	Action within authorized scope, no policy conflict, session history clean	Action proceeds. Evidence recorded. Workflow continues.
BLOCK	Action classified CRITICAL or DESTRUCTIVE, policy violation detected	Action stopped pre-execution. Block reason recorded. Session continues.
ESCALATE	HIGH risk, SoD flag, Lethal Trifecta pattern, exception threshold crossed	Session paused. Human authority notified. Resumes on decision. Full context preserved.

—— WHO HOLDS THE AUTHORITY

The humans responsible for operating agents.

LangGuard's authority layer is not for the agent. It is for the humans accountable for what agents do. The IT administrator who owns the production systems. The compliance leader who signs the SOX attestation. The DevOps lead who controls the CI/CD pipeline. The CAIO who is accountable to the board.

These are not the humans building agents. They are the humans operating fleets of agents across regulated environments, customer-facing systems, and critical infrastructure. They need the final say. They need it encoded as policy. They need it enforced at the action surface. They need it evidenced automatically.

LangGuard gives them the authority layer the agentic era requires but nobody built.

—— WHERE WE START

Claude Code. The largest agentic action surface in the enterprise today.

Jensen Huang called it at GTC 2026: Claude Code has revolutionized software engineering. NVIDIA runs it across every engineer. Enterprises are deploying it at scale. Every Claude Code session is an agent taking actions on production systems — infrastructure, databases, APIs, CI/CD pipelines — with no authority layer above it.

SCOPE MCP Server ships in the Claude directory. Arbiter runs inside Claude Code via hooks and skills. Design-time enforcement as agents are built. Runtime enforcement as they execute. The authority layer is native to the workflow — not adjacent to it, not monitoring it after the fact.

Claude Code is the GTM entry point. The authority buyer — IT, compliance, CAIO — is the expansion. The developer discovers the authority gap. The budget holder funds the solution.

Own the Agentic AI Action Layer.

SCOPE MCP Server. Arbiter. Human authority over agent actions. Encoded as policy. Enforced post-reasoning. Evidenced at execution time.

VISIT LANGGUARD.AI · REQUEST EARLY ACCESS