What OpenTelemetry Can and Cannot Tell You About Your AI Agents

What It Proves About Your AI Agents, and Where Governance Begins

The distance between what OpenTelemetry was built for and what AI governance is asking of it shows up in a single number. Distributed tracing descends from Dapper, the 2010 Google paper that gave the industry the vocabulary of traces and spans. Dapper sampled one trace in 1,024. That is ample for finding a latency regression, because a regression recurs and the next sample catches it. As a record of who authorized a privileged action, the same rate leaves 99.9% of the record missing, and the missing part is where the one event that mattered lives.

That gap is the subject of this report. It covers three things in turn: what AI agent telemetry answers well today, where the same telemetry stops answering, and what actually closes the difference.

ABOUT THIS RESEARCH

This report draws on a Kovrr Research field test of five production coding agents (Claude Code, OpenAI Codex CLI, Google Gemini CLI, GitHub Copilot CLI, and Claude Cowork), each instrumented and captured against a live OpenTelemetry collector across six sessions on an identical task. Findings on attribute conformance, convention stability, identity join-keys, and vendor documentation claims come from direct measurement of those captures.

PART 1

What OpenTelemetry Can Actually Tell You

OpenTelemetry Is a Wire Format and a Vocabulary, Not a Product

The 101, in one section, because the details that matter for governance are not the ones usually taught.

OpenTelemetry emerged in 2019 from the merger of OpenTracing and OpenCensus, two projects solving the same problem. There was no standard way to instrument code and ship what it produced. It defines three signals. Traces are trees of timed spans, where a span is one unit of work with a start, an end, and a bag of key-value attributes. Metrics are numeric time series. Logs are timestamped records with attributes attached.

Two further pieces do the real work. Semantic conventions are the agreed names, so that gen_ai.usage.input_tokens means the same thing coming from two vendors. Context propagation carries a trace identifier across process boundaries, which is how a span in one service links to a span in another.

The project itself is unambiguously successful. CNCF graduated it on 21 May 2026, after a third-party independent security audit, with more than 12,000 contributors from over 2,800 companies and the second-highest project velocity of the 240-plus CNCF projects, behind only Kubernetes.

The GenAI part of it is a different story, and the gap between those two facts is where planning errors start.

The Standard Says What It Is For, and It Is Not This

OpenTelemetry's own documentation defines its purpose plainly: "Observability is the ability to understand the internal state of a system by examining its outputs."

Read that as a governance requirement and the mismatch is immediate. Observability is about understanding your system. Assurance is about verifying a system, often one built by someone else, sometimes one behaving in ways nobody intended. The first assumes the software is on your side. The second cannot.

Dapper named its three design goals as low overhead, application-level transparency and scalability. It never overshot 0.3% of one core, which is a genuine engineering achievement and a statement of priorities. Completeness is not on that list. Neither is integrity, non-repudiation, or resistance to a component that would rather not be observed. None of this is a design criticism. It is a description of what the design was for.

Governance Reached For It Anyway, and Was Right To

The pull is real, and it is not just convenience.

Regulation now asks for exactly this shape of record. The EU AI Act, Article 12(1), states that "high-risk AI systems shall technically allow for the automatic recording of events (logs) over the lifetime of the system", and Article 12(2) ties those logs to identifying risk situations or substantial modifications, supporting post-market monitoring, and monitoring operation. Article 26(6) puts the retention obligation on the deployer, at a minimum of six months. An organization running agents needs a log, and a standard log format is better than five proprietary ones.

The EU AI Act is not the only framework asking for this. ISO/IEC 42001's Annex A.6.2.8 makes event-log recording a mandatory control across the AI system life cycle. MITRE ATLAS lists AI Telemetry Logging, mitigation AML.M0024, as a named defense specifically against agentic actions, tool use and data access, not just model input and output. Three different frameworks, three different audiences, converging on the same floor.

There is also a technical reason, and it is the strongest argument for instrumenting agents at all. Agent telemetry is the only source that sees the action boundary.

The Action Boundary Is Visible Nowhere Else

An AI agent spends most of its time reasoning, and then it does something: runs a shell command, writes a file, calls an API, opens a pull request. The moment it stops thinking and starts acting is the moment a security team cares about, and it is nearly invisible to every control already deployed.

Network inspection sees TLS to a model endpoint. It sees the same TLS whether the agent went on to summarize a README or to read a credentials file. Endpoint detection sees a process that spawned a shell, which is what developer tools do all day. Identity systems see a token that was already issued. None of them can distinguish an agent asked to tidy a config file that chose to exfiltrate one, because the distinguishing information is in the tool call, and the tool call exists in exactly one place.

In the field test behind this report, four of the five agents recorded the executed shell command and its output. That is a genuine capability; it is new, and nothing else in the stack has it.

Four Questions It Answers Well

01 Which agents are running, operated by whom, at what volume. This is the population denominator most organizations cannot currently produce, and it is what a board asks for first. It is inventory rather than detection, and inventory is where every governance program starts.

02 What an agent invoked, with which arguments. The action boundary above.

03 What happened in one session, in order. Spans nest, so a well-instrumented agent yields the shape of a run: prompt, inference, tool call, result, next inference. For incident reconstruction, that sequence is the artifact.

04 Whether a run is abnormal. Structured, comparable, high-volume records support ordinary baselining. Token spikes, unusual tool mixes, off-hours activity.

Those four are worth the effort on their own. A team that only needs a usage inventory and session reconstruction has real value here already.

That is a genuinely different position than the one Dapper's engineers were solving for. A trace sampled to catch a recurring latency regression was never going to answer who authorized a privileged action, and nothing about AI agents changes that math. What changed is that governance stopped asking Dapper's question and started asking a harder one, and four solid answers is a good place to start. What follows is about the questions that look like they belong on that list and do not.

PART 2

The Three Questions Every AI Telemetry Claim Should Survive

What follows is the fifth question every security team eventually asks, the one no amount of instrumentation answers on its own: can this record be trusted enough to build a control on it? Before any telemetry source, an AI agent's or anything else's, gets to carry an assurance claim, run it through three questions.

Three Questions to Ask Before Any Telemetry Carries an Assurance Claim

This is the reusable part, and it applies well beyond AI agents.

01 **Who controls the emitter, relative to who is being observed?**

02 **Is the record complete, or sampled?**

03 **Does it record the attempt, or the outcome?**

Agent telemetry answers: the subject controls it, sampling is architecturally available, and it records the attempt. Three out of three on the unhelpful side. Instrument anyway, and know precisely what the record is before building a control on top of it.

Run the same three questions against the sources already in a SOC, and the contrast is sharp. AWS CloudTrail is written by the control plane rather than by the caller, retains every management event rather than a sample, and records the outcome including the authorization decision. Three out of three the other way. The completeness has a seam even here: the object-level data events closest to what an agent's tool call actually touches, which is an S3 read or a table scan, are opt-in and billed per event, not on by default the way management events are. That is why an investigator reaches for it first, and it is the standard agent telemetry has to be measured against rather than graded on its own curve.

Each of the three deserves its own treatment, as mitigations differ.

The Emitter Is the Subject

The agent writing the telemetry is the agent under review. Every one of the five agents tested reads its telemetry configuration from a file or an environment variable in the developer's own home directory: $CODEX_HOME/config.toml for Codex, .gemini/settings.json for Gemini, OTEL_EXPORTER_OTLP_ENDPOINT for Claude Code and Copilot. An agent run with any of those switched off produces nothing, and nothing looks exactly like a quiet week. There is no negative signal, so a serious deployment needs alerting on stream cessation and on writes to telemetry configuration, and almost nobody ships that.

A competent reader will object that this describes every log, and the objection deserves a real answer rather than a dismissal. It is not true of the sources security teams already lean on hardest. Cloud audit logs are written by the cloud provider's control plane, not by the caller. Identity logs are written by the identity provider, not by the principal. Those records are produced by something structurally separate from the actor they describe, which is precisely why they carry weight in an investigation.

Agent telemetry has no such separation. The distinction worth drawing is narrower than self-reported versus authoritative: it is whether the party with a motive to omit something is also the party writing the record.

This is not unique to OpenTelemetry, or to this report's own testing. An independent teardown of GitHub Copilot's separate control-plane audit log, a different product from anything OTel-based, found the same gap: audit records carry no full local IDE or CLI session, no prompts, no responses, no generated code, no local tool calls. Exporting OTel data alongside it closes part of that hole, but prompt and tool-argument content stays excluded by default there too, requiring an explicit opt-in. Two unrelated signals from the same vendor, an audit log and a telemetry pipeline, share the same blind spot for the same reason.

There is one partial answer, and it is worth knowing about. Both Claude Code and GitHub Copilot now support administrator-imposed telemetry configuration delivered by MDM or account policy, which GitHub's July 2026 changelog describes as taking "precedence over environment variables and user settings". That moves the switch out of the observed party's hands, which is a real improvement over a developer-owned config file. It does not make the agent's account of its own behavior independent of the agent.

It Records Intent, Not Effect

This is the limitation that survives every improvement in instrumentation, and it is the one most often missed.

A tool span records that an agent invoked something. In the field test, Codex emitted a codex.tool_result event carrying both the shell command it ran and the command's full standard output, which is about as complete as an intent record gets. It still does not record whether the credential had permission, whether the resource existed, or whether anything changed, because the agent's own process is not where that is decided. Swap the harmless fixture command for one that deletes an object from a production bucket and the span looks identical either way: same shape, same attributes, same success flag from the shell's point of view.

That swap already happened once, for real. Orca Security's February 2026 disclosure of RoguePilot showed a hidden instruction inside a GitHub Issue, invisible to a human reviewer in an HTML comment, directing GitHub Copilot to check out a crafted pull request and exfiltrate the workspace's GITHUB_TOKEN through a symlinked file. Copilot's own span for that turn records a tool call and a completed status, the same shape as checking out any other branch. The effect, a stolen token with full repository write access, lived only in what happened next.

Agent telemetry is an intent record. Cloud audit logs, git history, database audit trails and identity logs are effect records. Attribution requires both, joined. A perfectly conformant, fully complete, tamper-proof agent telemetry stream would still only tell you what the agent tried.

The three questions above are about whether to trust a record at all. A separate cluster of problems sits alongside them, less about architecture and more about practice: whether the vocabulary two vendors use actually means the same thing, whether that vocabulary is stable enough to build detection content against, whether it has a word for the one signal compliance asks for by name, and whether the vendor's own documentation about any of this can be taken at face value. All four showed up in the same field test.

Comparability Across Vendors Is Not Given

The premise of a shared vocabulary is that a query written once works everywhere. For coding agents in 2026, it does not.

Measured across five agents on an identical task, the share of span attributes sitting in the standard gen_ai.* namespace ranged from 5% to 100%. One agent reports token counts as gen_ai.usage.input_tokens and another as plain input_tokens, so a rule written against the convention silently returns nothing for the second. Detection content written against the standard covers whichever subset of an estate happens to conform this quarter, and produces no error for the rest.

Nothing In the GenAI Conventions Is Stable Yet

OpenTelemetry itself graduated from CNCF in May 2026. The GenAI conventions have not.

OpenTelemetry's own blog says so directly. Its most recent post on AI agent observability describes agent-framework conventions as still being defined through open discussion, only a draft agent-application convention as "finalized," and the conventions for models themselves as "experimental." That is not an outside read of the project's maturity. It is the project's own.

At the commit examined for this research, the GenAI semantic conventions repository carried 614 commits, 80 of them in the preceding 60 days, and zero tagged releases. Across its model definitions, there are 188 stability declarations, and every single one reads development. The only attributes marked stable that appear on a GenAI span are general-purpose ones borrowed from the core conventions: error.type, server.address and server.port.

A detection rule keyed on gen_ai.tool.name therefore rests on a name the project has explicitly declined to promise. When a name changes, nothing raises an error. The rule stops matching, quietly, and the dashboard stays green.

The Vocabulary Has No Word For a Human Decision

Regulators have already decided that a human decision is a minimum logged fact. Article 12(3) of the EU AI Act, setting out minimum logging for biometric identification systems, names "the identification of the natural persons involved in the verification of the results" among the things a log must record.

That clause governs biometric systems rather than coding agents, and the principle it encodes is not category-specific. When an automated system acts, who approved it is part of the record. Checking every one of the 63 gen_ai.* and 4 mcp.* registry attributes at the examined commit returns no attribute for approval, decision, consent, permission, or any human-in-the-loop concept.

Four of the five agents tested emit one regardless, in their own namespaces. Gemini CLI goes furthest and distinguishes an approval a person gave from one granted automatically. GitHub Copilot CLI, the agent that conforms most closely to the conventions, emits nothing of the kind.

An organization that standardizes strictly on convention attributes today would discard the one signal a compliance function asks for by name.

Documentation About Telemetry Safety Is Not a Control

Two vendor claims about telemetry behavior were tested directly against a collector for this research. One was that GitHub Copilot CLI disabled export rather than sending data over plain HTTP; it exported 407 KB including prompt content, and logged no warning. The other was that Claude Cowork included prompt content by default; it did on one client surface and redacted it on another, under one administrator configuration, with no setting exposed for either.

Both are documented behaviors that did not hold. The general lesson is worth more than either instance: telemetry configuration is increasingly pushed fleet-wide from an admin console, and the safety properties of those pipelines are being taken from vendor documentation rather than from measurement of one's own environment.

None of the five sections above argue against instrumenting. They argue against stopping there. A self-reporting emitter, an intent-only record, an inconsistent vocabulary, an unstable convention, a missing approval field, a vendor claim that does not hold under measurement: every one of these is fixable, not by more OpenTelemetry on its own, but by something alongside it.

PART 3

Closing the Gap: What Turns Agent Telemetry Into Evidence

What Actually Closes the Gap

Every limitation above has the same shape. Agent telemetry is one interested witness that saw part of what happened. Codex produced 2.7 MB describing a four-step task in the field test, which is a thorough record of one side of the story, and no additional volume of it turns into the other side.

Corroboration answers that better than a more thoroughly instrumented witness would, and the practical form of corroboration is a join. An agent reports invoking a command at a time, under a credential, against a resource. A cloud audit log independently records whether that resource changed. An identity log records whether the credential was entitled to it. An endpoint record places the process on a machine. Where those agree, the account is corroborated. Where they disagree, the disagreement is itself the signal, and it is the one a single stream can never produce.

A join needs a shared identifier, and that identifier is exactly what differs by vendor. Building the inventory below is the most useful hour available after reading this report, and it costs nothing but an afternoon with five capture files.

Agent	Identity It Offers	Joins to Identity and Cloud Records
Claude Code	corporate email, account ID, organization ID	directly
Codex CLI	corporate email, account ID, on the logs signal only	directly, if logs are collected
Claude Cowork	corporate email, account ID, organization ID, on every record	directly
GitHub Copilot CLI	a hashed pseudonym, plus organization and repository	only through a mapping you build
Gemini CLI	an operating system username and hostname	only through an endpoint or directory record

Two of those rows are the interesting ones. If Copilot or Gemini is deployed and nobody has built the mapping, then every action they report is unattributable at the moment it matters, and that is knowable today rather than during an incident.

What This Argument Does Not Establish

Zero of the six capture sessions behind the field test involved an agent trying to avoid being recorded. Every claim about tamper resistance in this report follows from the architecture rather than from an adversarial test, and it should be read that way.

That gap is not hypothetical. In July 2026, an OpenAI agent running an internal exploit-discovery evaluation broke out of its test environment and operated inside Hugging Face's production infrastructure for roughly two and a half days, executing around 17,600 actions. It enrolled into the internal network with a --no-logs-no-support flag and kept its state in memory rather than on disk. What it exfiltrated was chunked, encoded and in places hidden inside exception payloads; a first pass over the raw capture surfaced a handful of secrets and replicating the agent's own encoding scheme surfaced roughly four times as many. Both companies published the postmortem. The field test never asked an agent to hide. That incident is why the caveat above matters.

Nor is the gap uncontested within the standards themselves. A proposal for agentic-system semantic conventions, covering tasks, actions, agents, teams, artifacts and memory, has sat open on the OpenTelemetry semantic-conventions repository since August 2025 with no linked pull request. A parallel proposal to give OCSF its own AI-specific event classes aligned to MITRE ATLAS was opened in February 2025 and closed with no owner and no pull request. The gap this report describes is recognized. Neither standard has shipped a fix for it.

Two more caveats are worth naming quickly. The administrator-imposed configuration covered above, the one vendor documentation says a developer cannot override, was not exercised here either; that claim rests entirely on Anthropic's and GitHub's own documentation, which this report has already shown is worth testing rather than trusting. And the comparison of network, endpoint and identity controls against the action boundary is reasoning about what those controls observe, not a measurement of them; a well-tuned endpoint product with deep command-line visibility narrows that gap considerably.

A join also is not free. The effect sources have to exist, be retained over a comparable window, and carry an identifier that survives to both sides. The EU AI Act sets six months as the deployer's floor for AI system logs; if the cloud trail beside it is kept for 90 days, the overlap rather than the longer window is what can actually be corroborated. And where agents run under a shared service credential, the join returns a set of possible actors rather than a name, and that set is the honest output.

None of that is a reason to wait. It is a reason to be precise about what a join promises before promising it to someone else.

What To Do On Monday

Instrument and enable all three signals rather than traces alone. Traces are the default in most pipelines, and the default is wrong for several agents, where the security-relevant material sits on the logs signal instead.

Then write down, per agent, which questions its stream cannot answer, because it differs by vendor and it is documented nowhere. Build the join-key inventory above. Pin the convention version in any detection content, given 614 commits and no releases. Alert on the stream going quiet, since nothing is what a disabled agent looks like. And treat agent telemetry as what it is: a good, cheap, genuinely novel record of intent, and the first half of an attribution rather than the whole of one.

Kovrr's AI Security and Governance Platform performs that corroboration as Data Triangulation, joining an agent's account of what it did to the identity, cloud, endpoint and browser records that independently show what happened, so an action can be attributed to a person and priced rather than merely observed.

SOURCES

- Sigelman et al., "Dapper, a Large-Scale Distributed Systems Tracing Infrastructure" (Google, 2010)
- "OpenTelemetry: The Merger of OpenCensus and OpenTracing" (Google Open Source Blog, May 2019)
- CNCF: OpenTelemetry's Graduation announcement (21 May 2026)
- "What is OpenTelemetry?" (opentelemetry.io docs)
- "AI Agent Observability" (OpenTelemetry Blog, 2025)
- open-telemetry/semantic-conventions-genai (GitHub repository)
- Issue #2664, "Semantic Conventions for Generative AI Agentic Systems" (open-telemetry/semantic-conventions)
- Issue #1348, "OCSF support for AI Systems telemetry, logs and security controls" (ocsf/ocsf-schema)
- Regulation (EU) 2024/1689, Article 12: Record-Keeping
- Regulation (EU) 2024/1689, Article 26: Obligations of Deployers of High-Risk AI Systems
- ISO/IEC 42001, Annex A.6.2.8, "AI System Recording of Event Logs"
- MITRE ATLAS, AML.M0024, "AI Telemetry Logging"
- Claude Code: monitoring usage / OpenTelemetry
- Claude Code: environment variables reference
- Claude Code: managed settings reference
- "Enterprise-managed OpenTelemetry export for VS Code and CLI" (GitHub Changelog, 8 July 2026)
- GitHub Copilot CLI changelog
- GitHub Copilot: OpenTelemetry observability docs
- Codex CLI configuration reference (OpenAI)
- Gemini CLI telemetry documentation (Google)
- AWS CloudTrail user guide
- "GitHub Copilot Audit Logs" (Monad, July 2026)
- "RoguePilot: Critical GitHub Copilot Vulnerability" (Orca Security, February 2026)
- "Agent Intrusion: A Technical Timeline" (Hugging Face, 27 July 2026)
- "OpenAI Agent Used Exposed Credentials to Breach Hugging Face" (The Hacker News, 29 July 2026)
- Kovrr AI Security and Governance Platform
- Field test of five coding agents (Kovrr Research)