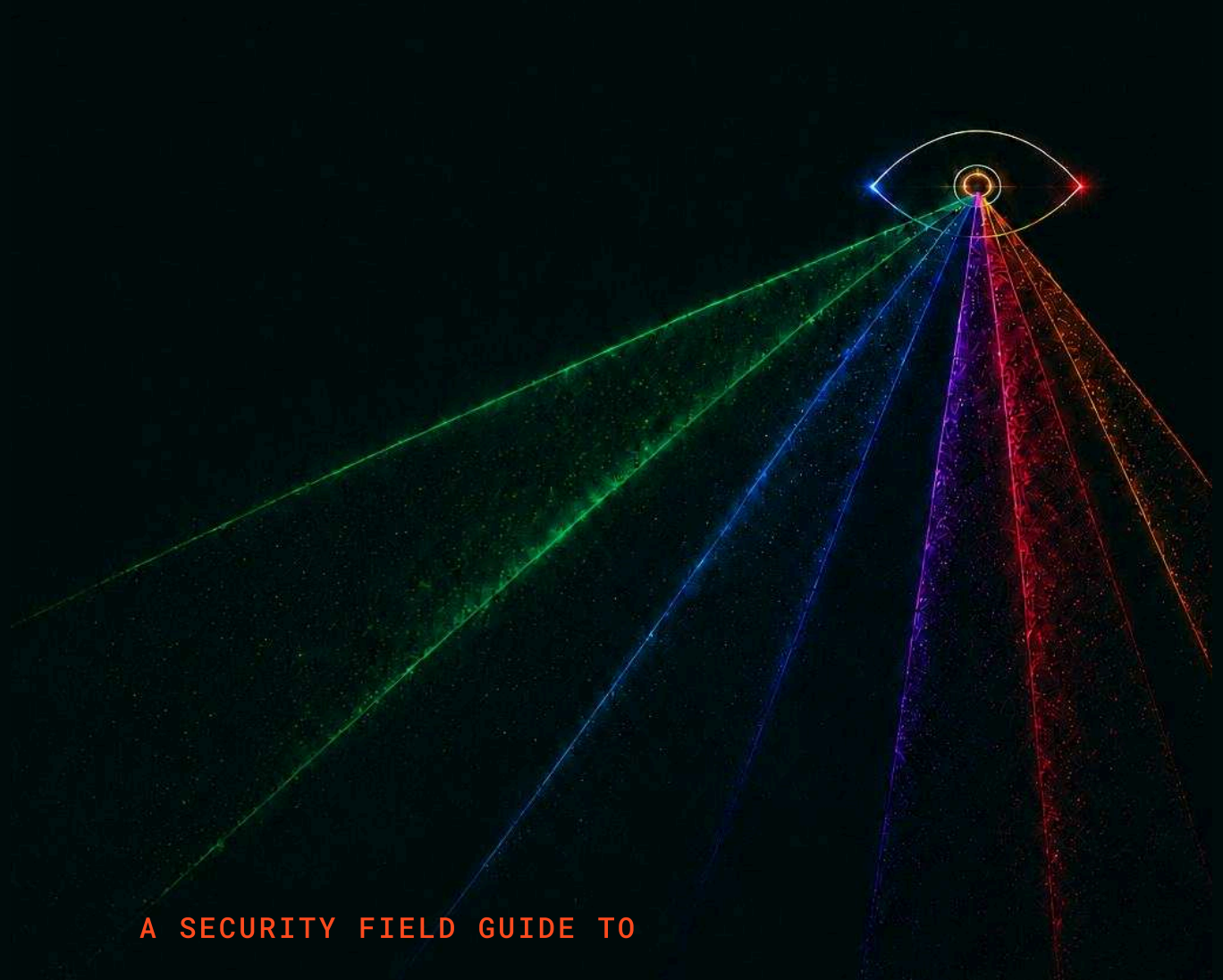


A SECURITY FIELD GUIDE TO

AI Tooling Visibility

The logs, blind spots, and security use cases across today's most popular AI tools.

By Darwin Salazar

with Matt Jane, Curtis Redgate, Kenneth Kaye and Valerie Worman

monad press

A SECURITY FIELD GUIDE TO

AI Tooling Visibility

The logs, blind spots, and security use cases across today's most popular AI tools.

By Darwin Salazar

with Matt Jane, Curtis Redgate,
Kenneth Kaye and Valerie Worman

Table of Contents

Introduction

AI tools now read and write code, search business data, run commands, change repositories, and call external systems. When one does something unexpected, the first question is often: what did the tool log?

The answer changes by product and surface. One source records prompts and tool decisions. Another records only the policy change that made a later action possible. A third proves that a feature was used while omitting the prompt, resource, and result.

This book brings together the AI tooling visibility posts Monad published throughout the year. The series grew out of requests from customers and prospects trying to onboard these sources into Monad and understand what each one could actually tell them. You can read the original posts and the rest of our research on the Monad blog.

Across eight AI tooling sources, each chapter answers the same questions: what the source records, what it misses, which fields matter, what the data can support, and where its limits begin. Detection is one use. The same records can support hunting, investigation, incident response, governance, baselining, and inventory.

THE EVIDENCE BOUNDARY

The native sources in this guide record three different things: agent execution, administrative changes, and feature invocation.

Runtime evidence

Claude Code, Claude Cowork, and Codex OpenTelemetry can record session settings, prompts (when enabled), tool decisions, results, and API activity. This is the closest source-native evidence of what the agent attempted and how the action was approved.

Control-plane evidence

Cursor, GitHub Copilot, Anthropic, and OpenAI audit records cover access, roles, configuration, policy, keys, and administrative changes. They rarely reconstruct the agent session that followed.

Feature-invocation evidence

Google Workspace Gemini activity records show that a user invoked a feature. They usually omit the file, email, prompt, response, and result.

Supporting evidence

Endpoint, identity, network, Git, CI/CD, DLP, and change records can add context. They cannot manufacture fields the AI source never recorded.

Start with one question: what can this record establish by itself? Everything else is supporting context or inference.

HOW TO USE THIS GUIDE

Part 1 covers local AI agents and developer tools. Part 2 covers enterprise AI platforms.

Part 1 mixes runtime OpenTelemetry with control-plane audit records. Part 2 covers administrative audit logs, activity feeds, and feature-usage records.

Every chapter draws a hard line between recorded and missing evidence. The use cases separate what the source proves from what still requires another log.

Choose the source from the question you need to answer. An audit log is not automatically a detection feed.

Source	Evidence type	Strongest uses	Main blind spot	Enrich with
Claude Code	Runtime OTel	Tool review, session analysis, cost and usage	Endpoint effects and network destinations	Endpoint, network, identity
Claude Cowork	Runtime OTel	File, tool, MCP, approval, and session review	Complete downstream effects and content disabled by default	Connected systems, endpoint, identity
OpenAI Codex	Runtime OTel	Policy drift, tool approvals, session analysis	Surface differences and external process effects	Endpoint, network, tool inventory
Cursor	Control- plane audit	Privacy, access, MCP, key, and guardrail changes	Local prompts, commands, files, and tool calls	Endpoint, Git, CI/CD, Hooks
GitHub Copilot	Audit plus GitHub-hosted agent activity	Access, policy, repository, MCP, and agent timelines	Complete local IDE and CLI sessions	Git, endpoint, CI/CD, network
Anthropic	Compliance activity feed	Identity, admin, content-activity, and API investigations	Prompt and file content in the activity record	Identity, endpoint, content endpoints
OpenAI API Platform	Administrative audit	Identity, roles, keys, network controls, configuration	Prompt and completion content	IdP, HR, asset, change records
Google Workspace Gemini	Feature-usage activity	Usage inventory and investigation pivots	Exact content, resource, and outcome	Workspace, identity, DLP, endpoint

SCOPE AND STALENESS

Field names, event types, defaults, and plan requirements move quickly. Treat the examples as starting points and validate them against raw records from your version and tenant.

This book stays inside the tools' own logs. It does not replace endpoint, network, identity, Git, CI/CD, DLP, or model-layer controls.

Collection, normalization, enrichment, filtering, routing, and access control decide whether the evidence ever becomes usable.

Part 1:

Local AI Agents and Developer Tools

Claude Code OpenTelemetry

Claude Cowork OpenTelemetry

OpenAI Codex OpenTelemetry

Cursor Audit Logs

GitHub Copilot Audit Logs

Claude Code OpenTelemetry

Claude Code exports metrics and structured events over OpenTelemetry. The stream can include prompts, assistant responses, API activity, tool decisions and results, MCP connections, session identifiers, token use, cost, and code-change metrics. Content requires explicit flags and belongs behind tighter access.

WHAT THE SOURCE CAPTURES

Evidence area	What is recorded	Key fields or identifiers	Security value
Metrics	Sessions, tokens, cost, code changes, and active time	`claude_code.*`, model, user and resource attributes	Usage, cost, adoption, and baseline context
Prompts	Full prompt text when explicitly enabled	`user_prompt`, `prompt`, `prompt.id`	Prompt review and scoped content hunting
API requests	Model, token counts, cost, duration, and status	`api_request`, model, tokens, duration	Request analysis and operational context
Tool decisions	Approval or rejection before execution	`tool_decision`, `decision`, `source`, `tool_use_id`	Human, configuration, and policy decision review
Tool results	Tool type, command, outcome, duration, and approval context	`tool_result`, `tool_name`, `full_command`, `success`, `tool_use_id`	Command review and session reconstruction
MCP activity	Connection lifecycle plus invocation through standard tool events	`mcp_server_connection`, `tool_decision`, `tool_result`, `tool_parameters.mcp_server_name`	Server connectivity and external tool review

WHAT THE SOURCE DOES NOT CAPTURE

Missing evidence	Why it matters	Where to look instead
Complete endpoint effects	A command record does not prove every child process or file effect	Endpoint and process telemetry
Network destination and flow	A command may imply egress without proving the connection	DNS, proxy, firewall, NetFlow, or endpoint data
Indirect injection source	Instructions inside files or messages may never enter `user_prompt`	Repository, file, email, ticket, and tool-content evidence
Complete MCP configuration lifecycle	Connection and tool records do not capture every configuration or policy change	MCP configuration, policy, and connected-system logs

USE CASES

Start with command review, possible data movement, MCP activity, and optional prompt-content hunting.

Unauthorized tool use

`claude_code.tool_decision` records approval or rejection before execution. Accepted calls can later emit `claude_code.tool_result` with the outcome and approval context. Keep rejections, configuration approvals, and failed results separate. `tool_use_id` ties decisions, results, and hook activity where emitted.

Day-one detections:

- `curl` or `wget` to an external IP in `full_command`
- Reads of `.aws/credentials` or `.env` files
- Packages installed from untrusted registries
- `chmod` on sensitive files or SSH key generation

- Commands attempted but rejected (decision: reject)
- Commands auto-accepted via config that fall outside policy

Possible data movement

For shell activity, use `tool_parameters.full_command`; `bash_command` may hold only a short label such as `echo`. Same-session file-read and outbound-action records can surface possible data movement. MCP invocations appear in `tool_decision` and `tool_result`, not a separate `mcp_tool_use` event.

What to look for:

- Reads of sensitive file paths followed by `curl` or `wget` in `full_command` within the same session
- `session.id` correlation between file reads and outbound network calls
- `tool_decision or tool_result records whose MCP tool details show a Slack, email, or external API connector`

MCP tool use

Claude Code records MCP connection lifecycle through claude_code.mcp_server_connection and MCP invocation through tool_decision and tool_result. Set OTEL_LOG_TOOL_DETAILS=1 to add tool_parameters.mcp_server_name and tool_parameters.mcp_tool_name. These records establish connection state and attempted tool activity, not the full effect inside the connected system.

What to flag:

- Unknown or unapproved server names in tool_parameters.mcp_server_name
- MCP tools accessing resources outside the user's normal scope
- Bulk data retrieval through internal API connectors

- MCP tool invocations that follow anomalous bash activity in the same session

Prompt injection

With `OTEL_LOG_USER_PROMPTS=1`, `user_prompt` supports low-confidence searches for known injection strings. In current releases, that flag can also expose assistant responses when `OTEL_LOG_ASSISTANT_RESPONSES` is unset. Set it to `0` if responses must remain redacted. Indirect injection may never reach the prompt log; the decisions and results may be the only evidence.

Direct injection patterns to detect:

- “Ignore previous instructions” or “output your system prompt”
- Base64-encoded payloads in prompt text
- Attempts to override safety constraints
- References to internal systems the user shouldn’t be aware of

COLLECTION QUIRKS

Claude Code ships the data - the problem is the shape. Raw OTLP is deeply nested, with metrics and events buried inside arrays of key-value pairs. No engineer wants to write detections against that. Here is a representative slice:

```JSON
{"resourceMetrics":[{"resource":{"attributes":[
  {"key":"department","value":{"stringValue":"engineering"}},

{"key":"service.name","value":{"stringValue":"claude-code"}},
  {"key":"service.version","value":{"stringValue":"2.1.76"}}
  ]},"scopeMetrics":[{"metrics":[{
    "name":"claude_code.session.count",
    "sum":{"dataPoints":[{"asDouble":1,"attributes":[
```

```
{"key":"user.email","value":{"stringValue":"user@domain.com"}
},
      {"key":"session.id","value":{"stringValue":"..."}}
    ]}]}
  }]}]}]}
```

Before this data is useful, it needs to be split into one record per event and normalized to a flat schema.

NORMALIZATION EXAMPLE

After the payload is split and flattened, a `user_prompt` event looks like this:

JSON

```
{
  "body": "claude_code.user_prompt",
  "event.name": "user_prompt",
  "event.sequence": "14",
  "event.timestamp": "2026-03-16T12:34:36.133Z",
  "host.arch": "arm64",
  "organization.id": "00000000-0000-0000-0000-000000000000",
  "prompt": "This is an OTel test prompt.",
  "prompt.id": "00000000-0000-0000-0000-000000000000",
  "prompt_length": "13",
  "service.name": "claude-code",
  "session.id": "00000000-0000-0000-0000-000000000000",
  "user.email": "user@domain.com",
  "department": "engineering",
  "environment": "production"
}
```

A `tool_result` event shows the accepted Bash command and its outcome:

JSON


```
{
  "body": "claude_code.tool_result",
  "cost_center": "CC-1234",
  "decision_source": "user_temporary",
  "decision_type": "accept",
  "department": "engineering",
  "duration_ms": "1150",
  "environment": "production",
  "event.name": "tool_result",
  "event.sequence": "3",
  "event.timestamp": "2026-03-18T13:19:44.789Z",
  "organization.id": "00000000-0000-0000-0000-000000000000",
  "session.id": "00000000-0000-0000-0000-000000000000",
  "success": "true",
  "team": "platform",
  "tool_name": "Bash",
  "tool_parameters": {
    "bash_command": "echo",
    "full_command": "echo '{\"foo\": \"bar\"}' >
/Users/username/Desktop/projects/test",
    "description": "Write JSON to test file"
  },
  "tool_result_size_bytes": "0",
  "user.email": "user@domain.com"
}
```


`tool_name` identifies the tool. `tool_parameters.full_command` carries the command. On `tool_result`, `decision_type=accept`; rejected calls do not emit a result. `decision_source=user_temporary` identifies per-invocation approval, and `success=true` records the outcome.

Tag records with `OTEL_RESOURCE_ATTRIBUTES` at the source to carry department, team, environment, or cost-center context into every event.

Monad publishes the flattening transform in the community repository. It converts raw OTLP into one record per event.

SENSITIVE DATA AND ACCESS

Five flags control content collection. `OTEL_LOG_USER_PROMPTS=1` captures prompts and may also enable assistant responses when the response flag is unset. `OTEL_LOG_ASSISTANT_RESPONSES=1` captures assistant text. `OTEL_LOG_TOOL_DETAILS=1` adds tool, MCP, and skill details. `OTEL_LOG_TOOL_CONTENT=1` captures tool input and output. `OTEL_LOG_RAW_API_BODIES=1` emits raw request and response bodies. All are off by default.

Tool detail, tool content, and raw API body logging can expose commands, paths, URLs, arguments, tool output, system messages, and conversation content. Anthropic documents a 60 KB limit for tool content and truncation limits for tool input details. Apply redaction or access controls before broad downstream access. The community transform flattens the structure but does not mask PII or any other sensitive data.

`OTEL_LOG_USER_PROMPTS` is a deliberate opt-in. Set retention, access, redaction, and employee-notice requirements before enabling it in production.

DETECTION IDEAS

Use these as candidate queries. Validate field values and expected behavior in your environment.

Suspicious outbound network call from a Claude Code session

Review Bash or command line invocations where `full_command` contains `curl` or `wget` to an external host. If same-session file-read data is available, use it to strengthen the lead.

```
JSON
event.name = "tool_result"
AND tool_name = "Bash"
AND (tool_parameters.full_command contains "curl" OR
tool_parameters.full_command contains "wget")
AND tool_parameters.full_command NOT contains
"internal.yourdomain.com"
GROUP BY session.id
  HAVING COUNT(DISTINCT tool_parameters.full_command) > 1
  AND ANY(tool_parameters.full_command contains "cat" OR
tool_parameters.full_command contains "read")
```

Sensitive file access

Trigger: a Bash tool invocation where `full_command` reads a known sensitive path.

```
JSON
event.name = "tool_result"
AND tool_name = "Bash"
AND (
  tool_parameters.full_command contains ".aws/credentials"
  OR tool_parameters.full_command contains ".env"
  OR tool_parameters.full_command contains "id_rsa"
  OR tool_parameters.full_command contains "/etc/passwd"
)
```

Command rejected by user or policy

Review `tool_decision` events where the command was rejected. The event fires when the decision is made, before a later `tool_result`.

```
JSON
event.name = "tool_decision"
AND decision = "reject"
```

Unknown MCP server invocation

Where an approved MCP inventory exists, review tool_result events with an unknown tool_parameters.mcp_server_name. Without an inventory, use first-seen values to build one. Requires OTEL_LOG_TOOL_DETAILS=1.

```
JSON
event.name = "tool_result"
AND tool_parameters.mcp_server_name IS NOT NULL
AND tool_parameters.mcp_server_name NOT IN
("approved-server-1", "approved-server-2")
```

Prompt injection pattern in user input

Optional low-confidence hunt for known injection strings in `user_prompt`. Requires intentional prompt collection with `OTEL_LOG_USER_PROMPTS=1`.

```
JSON
event.name = "user_prompt"
AND (
  prompt contains "ignore previous instructions"
  OR prompt contains "output your system prompt"
  OR prompt MATCHES "[A-Za-z0-9+/]{40,}={0,2}"
)
```

GETTING THE DATA TO YOUR DESTINATIONS

Enable telemetry with `CLAUDE_CODE_ENABLE_TELEMETRY=1`. Set `OTEL_LOGS_EXPORTER=otlp` and `OTEL_METRICS_EXPORTER=otlp` for the signals you need. Point `OTEL_EXPORTER_OTLP_ENDPOINT` at the collector, choose `OTEL_EXPORTER_OTLP_PROTOCOL`, and add `OTEL_EXPORTER_OTLP_HEADERS` when the receiver requires authentication.

Send Claude Code logs and metrics to an OTLP-compatible collector. Keep the signal types separate, split nested payloads into one record per event, and flatten resource and event attributes before routing. Monad's public transform handles the split and normalization step.

CHAPTER TAKEAWAY

Collect Claude Code as OTLP, flatten it before detection, and keep prompt and tool content behind tighter controls. Monad handles the splitting, normalization, and routing in the pipeline.

Claude Cowork OpenTelemetry

Claude Cowork monitoring exports runtime activity through OpenTelemetry: user, session, selected workspace paths, prompts when captured, tool and MCP activity, approvals, API requests, errors, and model-response metadata.

Cowork reaches beyond code. It can act across local workspace folders and connected systems. Its OTel stream is runtime evidence from a managed desktop agent.

SOURCE BOUNDARY

Monitoring requires a Team or Enterprise plan and Claude Desktop 1.1.4173 or later. An administrator sets the OTLP endpoint under Admin settings > Cowork. Existing sessions do not pick up configuration changes.

Cowork activity is not captured in Anthropic's Compliance API today. Cowork OTel includes `user.account_uuid` and `user.account_id` for identity pivots into Anthropic administration and compliance records. OTel remains the session-level source.

WHAT THE SOURCE CAPTURES

Evidence area	What is recorded	Key fields or identifiers	Security value
Identity and session	User, organization, session, selected workspace paths, and app context	`session.id`, `organization.id`, `user.account_uuid`, `user.account_id`, `user.email`, `workspace.host_paths`, `service.name`	Attribution, scoping, and session reconstruction

Evidence area	What is recorded	Key fields or identifiers	Security value
Prompts and responses	Prompt length and optional text; response metadata and optional text	`user_prompt`, `assistant_response`, `prompt.id`, `prompt`, `response`, `model`, `request_id`	Prompt-level review and content-exposure analysis
Tool decisions	Approval or rejection plus decision source	`tool_decision`, `tool_name`, `decision`, `source`	Human, configuration, and hook approval review
Tool and file activity	Tool outcome, duration, errors, parameters, and optional input details	`tool_result`, `success`, `duration_ms`, `tool_parameters`, `tool_input`	File, command, and workflow investigation
MCP activity	MCP server scope plus server and tool details when captured	`mcp_server_scope`, `tool_parameters.mcp_server_name`, `tool_parameters.mcp_tool_name`	Connected-system and tool inventory
API activity	Model, tokens, estimated cost, duration, status, and errors	`api_request`, `api_error`, `model`, `cost_usd`, token fields, `status_code`	Request, failure, cost, and performance analysis

WHAT THE SOURCE DOES NOT CAPTURE

Missing evidence	Why it matters	Where to look instead
Complete downstream effect	A successful Cowork tool result does not prove every change persisted in the connected system	MCP server, SaaS application, repository, and source-system logs
Complete host and network activity	Cowork OTel is not process lineage, DNS, proxy, or flow telemetry	Endpoint, DNS, proxy, firewall, or other network evidence

Missing evidence	Why it matters	Where to look instead
Prompt, response, and tool content by default	The detailed reference documents metadata-only export until content capture is enabled	Approved `otlpContentCapture` settings and tightly controlled raw export
Cowork activity in the Compliance API	Compliance activity records do not reconstruct Cowork sessions	Cowork OTel joined to identity and administration records where useful

USE CASES

Use Cowork OTel to answer three questions: what the agent attempted, which workspace or connected system was involved, and how the action was approved.

Sensitive file and workspace access

`workspace.host_paths` identifies the directories selected for the session. With `toolDetails` enabled, `tool_input` and `tool_parameters` can add file paths, URLs, search patterns, and other arguments. Credential stores, restricted projects, customer data, and unexpected workspace scope are useful pivots. A directory path does not prove every file inside it was read.

MCP and tool activity

`mcp_server_scope`, `tool_parameters.mcp_server_name`, and `tool_parameters.mcp_tool_name` identify connected tools. Unknown servers, unusual tools, bulk retrieval, and activity against sensitive systems are useful pivots. Cowork establishes the invocation and result; the connected system establishes the downstream effect.

Approval decisions

`tool_decision` records accept or reject and the decision source: configuration, hook, permanent or temporary user choice, abort, or rejection. A

state-changing action approved by configuration or a hook matters when policy expects a human decision. Automatic approval alone is not misuse.

Content-capture drift

A populated `prompt`, `response`, or `tool_input` field proves that content is reaching the logging pipeline. It is enough to check redaction, routing, access, and retention. It does not explain why capture was enabled.

CONTENT AND ACCESS

The detailed Cowork reference documents metadata-only export by default. `otlpContentCapture` can enable `userPrompts`, `assistantResponses`, `toolDetails`, `toolContent`, and `rawApiBodies`. On Claude Desktop 1.17377 or later, enabling `userPrompts` also captures model responses. `user.email` is always present.

Anthropic's shorter Help Center overview currently says prompt content is included by default. Its detailed monitoring and telemetry references say metadata-only by default. Follow the detailed reference and verify one raw event before setting redaction, access, or retention policy.

COLLECTION QUIRKS

Cowork supports OTLP over HTTP/JSON or HTTP/protobuf. New settings load when a session starts. The exporter runs inside the Cowork VM.

Cowork cannot use a gRPC-only receiver. In shared Claude Desktop configuration, `grpc` makes Cowork fall back to HTTP/protobuf at the same endpoint. If that address accepts only gRPC, the events disappear. Use OTLP/HTTP.

Claude automatically adds the collector hostname to the Cowork session's egress allowlist. The perimeter firewall must still allow the collector.

GETTING THE DATA TO YOUR DESTINATIONS

Point Cowork at an OTLP/HTTP collector, normalize the nested records, and route them to their destinations. Monad's Claude Code input can receive Cowork through the same passive input; route on `service.name=cowork`. A gRPC-only receiver will not work.

CHAPTER TAKEAWAY

Collect Cowork through OTLP/HTTP, separate it with `service.name`, and treat content as sensitive. The Compliance API will not reconstruct the session.

OpenAI Codex OpenTelemetry

Codex can export OpenTelemetry, but exporter behavior still varies by surface and version. Logs, traces, and metrics use separate settings. Validate the OTLP/HTTP path and raw fields against the collector you actually run.

SOURCE BOUNDARY

OpenAI's Codex approval and security documentation describes a local agent that can read files, edit them, and run commands inside the working directory under `workspace-write`. Leaving that boundary or using the network can require approval.

That process now matters on developer laptops, CI runners, and development containers.

Sysdig's runtime research covers the host-side risk. Huntress documented how Codex activity complicated SOC triage during incident response in part one and part two.

The source-native signals are sandbox and approval policy, prompt logging, tool decisions, and new emitters. `danger-full-access` removes the sandbox network boundary. Other network settings may require configuration or endpoint evidence.

Covered surfaces: CLI, IDE extension, desktop app, and automation runners. Codex cloud and Codex Security use different telemetry models.

WHAT THE SOURCE CAPTURES

Evidence area	What is recorded	Key fields or identifiers	Security value
Session start	Model, approval policy, sandbox policy, and session context	`Codex.conversation_starts`, `approval_policy`, `sandbox_policy`	Policy review and session scoping
API and transport	Status, duration, attempts, errors, streaming, and token counts	`codex.api_request`, `codex.sse_event`, WebSocket events	Operational and failure analysis
Prompts	Prompt length and optional prompt content	`codex.user_prompt`, `prompt_length`, `prompt`	Prompt exposure review and scoped hunting
Tool decisions	Approval decision and source	`codex.tool_decision`, decision, source, `conversation.id`	Human, configuration, and automation review
Tool results	Tool outcome, duration, arguments, and possible output	`codex.tool_result`, `tool_name`, `call_id`, success	Tool execution and investigation context
Resource context	Emitter, version, host, user, and originator	`service.name`, `service.version`, `host.name`, `originator`	Inventory, attribution, and rollout analysis

WHAT THE SOURCE DOES NOT CAPTURE

Missing evidence	Why it matters	Where to look instead
CLI bypass flags	The OTel record may not contain command-line flags	Endpoint and process telemetry
Network configuration and destinations	danger-full-access implies no sandbox boundary, but other network settings and destinations may be absent	DNS, proxy, firewall, NetFlow, eBPF, or endpoint data
Complete content	Prompts may be redacted and tool results vary by surface and version	Tightly controlled raw export or source-system evidence
Uniform cross-surface schema	CLI, IDE, desktop, and automation behavior can differ	Raw validation on the exact Codex surface

OpenAI documents the current Codex OTel event set. Logs, traces, and metrics do not carry the same fields. Logs hold the richest per-event context. Metrics help with baselines and dashboards but are not the primary per-conversation detection source.

The sample below keeps the raw OTel split between resource and log attributes visible. The full export adds `resourceLogs`, `scopeLogs`, `logRecords`, timestamps, counters, and trace identifiers.

Host, user, conversation, trace, and span values are fictional. Prompt content remains redacted.

JSON

```
{
  "resource": {
    "attributes": {
      "service.name": "codex-app-server",
      "service.version": "0.119.0-alpha.28",
      "host.name": "dev-laptop-042"
    }
  },
  "logRecord": {
    "attributes": {
      "event.name": "codex.user_prompt",
      "prompt_length": "147",
      "prompt": "[REDACTED]",
      "conversation.id": "conv_7f3a91c2b8e44d9a",
      "app.version": "0.119.0-alpha.28",
      "originator": "Codex_Desktop",
      "user.email": "developer@acme.example",
      "model": "gpt-5.2-codex"
    }
  }
```

In raw OTel, `event.name`, `conversation.id`, `originator`, `user.email`, and `prompt` sit in log attributes. `service.name`, `service.version`, and `host.name` sit in resource attributes.

Most detections run against flattened records, so the patterns below use both raw names and common normalized fields such as `event_name`, `conversation_id`, and `service_name`.

Here, the prompt is redacted while `prompt_length` remains available. That is a collection choice, not a schema guarantee.

Prompt content can contain source code, vulnerability details, internal context, and user data. It can also be valuable during an insider-threat or incident investigation. Collect it only with deliberate access and retention.

Current `codex.conversation_starts` records emit `approval_policy` and `sandbox_policy`, not a flat `sandbox_mode`. If a pipeline creates `sandbox_mode`, it is a normalized field, not the raw name.

`codex.tool_result` may include shell output, patch bodies, tool arguments, or file content. Treat it as content, not harmless metadata.

Validate field names on the exact Codex surface and version you collect.

USE CASES

Codex runs on systems that already hold source code, credentials, tickets, build access, and sometimes production context. Start with five questions:

- Did a Codex session start with weaker-than-expected sandbox or approval settings?
- Did prompt content, tool output, or other sensitive material appear in exported log data?
- Were destructive tools approved by config or automation instead of a human decision?

- Did a session with broad network capability correlate with unusual egress or unexpected process behavior?
- Did a new Codex originator, service, version, or session source appear outside known rollout paths?

DETECTION PATTERNS

Codex signals and the detections they support

Start with direct Codex signals, then enrich with endpoint and network context.

TELEMETRY SOURCE	WHAT IT CAN DETECT
`codex.conversation_starts` `approval_policy · sandbox_policy · network_access`	**Sandbox or approval policy weakened** Sessions where protections are reduced or approval requirements are relaxed.
`codex.user_prompt` `prompt_length · optional prompt content`	**Prompt content logging enabled** When prompt content is captured or logged.
`codex.tool_decision` `approval outcome · source`	[CAUTION] **Destructive tool approved by config** Approvals that should likely have required a human decision. Handle with care.
session metadata (anchor) `service.name · originator · app.version`	**Unexpected Codex surface or originator** Sessions from new or unusual Codex surfaces.
endpoint / network telemetry destination · process tree *(enrichment)*	**Network-enabled session with unusual egress** Sessions with network access reaching unusual destinations.

The first four rows are Codex-native. The last row is enrichment: endpoint and network telemetry is what turns a network-enabled session into a detectable egress event.

[CAUTION] marks the one pattern to handle with care: a config- or automation-sourced approval that bypassed a human. Each source on the left maps to the detection it supports on the right.

Start with source-native checks: weakened policy, prompt or tool content in the log stream, configuration-approved state changes, and unexpected emitters. Network behavior needs endpoint or network evidence.

Codex can establish session policy, prompt-export behavior, and some tool decisions. It does not establish CLI bypass flags, whether every tool is destructive, or which network destination was reached.

1. Sandbox policy weakened

Why it matters: `danger-full-access` or an approval policy of `never` removes containment and review points before Codex changes the host or its data.

Pattern: `codex.conversation_starts` with `sandbox_policy=danger-full-access`, `approval_policy=never`, or an equivalent weakened control. Check `--dangerously-bypass-approvals-and-sandbox` and `--yolo` in endpoint logs; Codex OTel may not carry the CLI flags.

Key detection fields: `event.name`, `approval_policy`, `sandbox_policy`, `originator`, `service.name`, `user.email`, `host.name`

Tuning notes: Approved CI, automation runners, lab environments, and power-user workflows may generate expected hits.

Supporting evidence, if present: endpoint or process logs for CLI flags and process context.

2. Prompt content logging enabled

Why it matters: An unredacted prompt stream creates a second searchable copy of source code, vulnerability details, credentials, incident context, and customer data.

Pattern: `codex.user_prompt` where `prompt` is populated with anything other than `[REDACTED]`, or the normalized record shows unredacted content.

Key detection fields: `event.name`, `prompt`, `prompt_length`, `conversation.id`, `originator`, `user.email`

Tuning notes: Some teams may intentionally enable prompt logging in development, testing, or tightly scoped troubleshooting workflows.

3. Destructive tool approval from config or automation

Why it matters: Configuration or automation can approve a state-changing tool without the human review the policy was meant to provide.

Pattern: `codex.tool_decision` indicates that configured policy, rather than an interactive user decision, allowed a tool capable of modifying state. Use `codex.tool_result` to confirm whether it ran and succeeded.

Key detection fields: `event.name`, `tool_name`, `call_id`, `decision`, `source`, `conversation.id`, `originator`

Tuning notes: This pattern needs a definition of state-changing tools. Without a registry, use observed tools to build one instead of pretending the classification already exists.

Supporting evidence, if present: MCP metadata, application tool metadata, or an internal tool registry.

4. Session with broad network capability and unusual egress

Why it matters: Network access creates paths for data movement, dependency tampering, and external tool use. Codex OTel does not expose one universal network-enabled field.

Pattern: `sandbox_policy=danger-full-access`, especially when endpoint or network data shows outbound activity outside expected destinations. This mode disables Codex sandboxing, including its network restrictions. Under `workspace-write`, network access is configured separately. Records from DNS, proxy, firewall, NetFlow, eBPF, or endpoint sources provide the destination and process context that Codex OTel does not.

Key detection fields: `event.name`, `sandbox_policy`, `approval_policy`, `conversation.id`, `originator`, `user.email`, `host.name`

Tuning notes: Network behavior is highly environment-specific. Package managers, source control, artifact repositories, and approved SaaS services can create expected egress.

Supporting evidence, if present: network telemetry, host and user identity, process lineage, and approved destinations.

5. Unexpected Codex originator or service surface

Why it matters: A new originator, service, version, or session source can be a legitimate rollout, an unmanaged workstation, a new automation runner, or a shadow workflow. Either way, the inventory changed.

Pattern: First-seen or unexpected values in `originator`, `service.name`, `service.version`, `app.version`, or `session_source` for a user or host. Without history, use these fields to build the initial inventory.

Key detection fields: `originator`, `service.name`, `service.version`, `app.version`, `session_source`, `host.name`, `user.email`

Tuning notes: New values may reflect legitimate rollout, version upgrades, collector changes, or new automation paths.

Supporting evidence, if present: known Codex surfaces, expected services, approved CI runners, and managed developer environments.

BASELINES IF YOU HAVE HISTORY

With enough history, baseline four things:

- Human versus CI or automation sessions
- Expected approval sources and sandbox policies
- Known originators, service names, app versions, and session sources

- Normal event volume by session, surface, and workflow

Metrics are useful for baselining, but do not assume they can always be joined back to `conversation.id`.

Validate these first:

1. Sandbox or approval policy weakened
2. Prompt content exported
3. Destructive tool approval from config or automation
4. New originator or service surface
5. Sessions with broad network capability and unusual egress

PRIVACY, REDACTION, AND RETENTION

Do not treat prompts and tool results like ordinary application logs. Prompts can include source code, incident notes, hostnames, vulnerability details, customer context, and secrets. Tool results can include shell output, patches, arguments, paths, and file content.

The content stream may be as sensitive as the original session.

Use the controls the collection stack can actually enforce:

- Prefer metadata-first detection before inspecting prompt bodies.
- Redact or mask sensitive content before broad downstream access. If field-level processing is unavailable, restrict collection and destination access.
- Where storage controls support it, give raw payloads tighter access and shorter retention than normalized metadata.
- Where approved inventories exist, compare service names, originators, CI runners, and egress destinations against them.
- Validate telemetry behavior on the exact Codex surface you use: CLI, IDE extension, desktop app, automation runner, or CI.

Enforce that split before the data reaches broad downstream access.

Keep metadata broadly searchable. Put prompt bodies, tool output, shell output, paths, patches, raw payloads, and high-cardinality identifiers behind tighter redaction, access, and retention.

OUT OF SCOPE

Keep four boundaries clear:

- **Prompt-injection detection:** useful, but it usually requires inspecting prompt content, tool output, repository content, or running a sidecar inspector.
- **Codex Security and Codex cloud activity:** different integration models from local Codex OTel.
- **Cross-agent correlation:** most orgs run more than one agent. Claude Code to Codex pivots require unified instrumentation across providers. (See the bridge chapter following Part 1.)
- **Endpoint-only detection:** CLI flags, process lineage, shell history, and child-process behavior should be covered in EDR or endpoint telemetry, not Codex OTel alone.

GETTING THE DATA TO YOUR DESTINATIONS

Codex OTel is off by default. In `config.toml`, set `[otel].exporter` to `otlp-http` or `otlp-grpc`, then configure the exporter endpoint and any required headers or TLS settings. `exporter = "none"` sends nothing. Metrics and traces use separate exporter settings.

Configure the Codex log exporter to send OTLP to a compatible collector or backend. Keep logs, traces, and metrics separate. Flatten resource and log attributes without presenting normalized fields as raw fields, then route the records to the destinations that will use them. Monad's public Codex transform

handles the flattening step.

CHAPTER TAKEAWAY

Codex OTel records policy, prompts when enabled, tool decisions, and results. It does not prove CLI flags, process effects, or network destinations.

Keep metadata searchable. Restrict prompt and tool content according to its sensitivity.

Cursor Audit Logs

Cursor audit logs show you when someone changes the controls around the agent. They do not show you what the agent did.

Cursor Agent sits next to source code, credentials, terminals, repositories, and external systems reached through MCP servers. The controls are familiar: access and roles, API keys, Privacy Mode, MCP, repo controls, team rules, hooks, and commands.

Use the audit log to establish who changed a control, what changed, and when.

The documented response contains `event_id`, `timestamp`, `ip_address`, `user_email`, `event_type`, and `event_data`. It does not document prompts, responses, generated code, runtime commands, local files, MCP arguments, processes, or network activity. Cursor Hooks and host, network, Git, or CI/CD logs cover parts of that gap.

WHAT THE SOURCE CAPTURES

Evidence area	What is recorded	Key fields or identifiers	Security value
Identity and membership	Logins, users, groups, memberships, and role changes	`event_type`, `user_email`, `ip_address`, `event_data`	Access and privilege timelines
API keys	Organization and user key lifecycle events	`team_api_key`, `user_api_key`	Durable access inventory and review
Privacy	Privacy Mode changes	`privacy_mode` events and event data	Data-handling control monitoring
MCP	Server configuration changes	`mcp_server_config` and event data	External tool and execution-path changes
Agent guardrails	Repository, rule, hook, command, and team-setting changes	`team_repo`, `team_rule`, `team_hook`, `team_command`	Control and observability changes

WHAT THE SOURCE DOES NOT CAPTURE

Missing evidence	Why it matters	Where to look instead
Local prompts and responses	The audit API is not a session-content source	Cursor runtime or approved content sources
Generated code and file reads	The record cannot reconstruct what the local agent read or produced	Git, endpoint, repository, and file evidence
Terminal commands and process lineage	`team_command` is not shell-command telemetry	Endpoint telemetry and Cursor Hooks
MCP arguments and effects	A configuration change does not establish what a tool later did	Cursor Hooks and connected-system logs
Network connections	The audit record does not show local egress	Endpoint, DNS, proxy, or firewall data

Cursor audit log event types, by detection category

1 · Access & Identity

- `login`
- `logout`
- `add_user`
- `remove_user`
- `update_user_role`

2 · Data Protection

- `privacy_mode`

3 · Agent Tooling & Guardrails

- `mcp_server_config`
- `team_repo`
- `team_rule`
- `team_hook`
- `team_command`

4 · API & Automation Access

- `team_api_key`
- `user_api_key`

5 · Secondary Governance Signals

- `user_spend_limit`
- `team_settings`
- Directory group events
- Bugbot events

Audit logs show control-plane changes. Runtime telemetry shows what the agent did.

Audit logs are available on Cursor Enterprise plans only.

The audit stream covers identity, API keys, Privacy Mode, MCP, repo controls, team rules, hooks, commands, spend, and Bugbot.

Example event:

```
JSON
{
  "event_id": "evt_abc123",
  "timestamp": "2026-01-15T12:30:00.000Z",
  "ip_address": "203.0.113.42",
  "user_email": "admin@company.com",
  "event_type": "add_user",
  "event_data": { "email": "newuser@company.com", "method":
"manual" }
}
```

Use the audit log for control changes. Use runtime evidence for agent behavior.

USE CASES

Start with changes to access, data handling, agent capability, and visibility.

1. Privacy Mode changed

Privacy Mode changes the data-handling boundary around prompts, code, and editor activity.

A `privacy_mode` event shows when those protections were disabled, narrowed, or changed. The exact effect still depends on the model and provider.

If Privacy Mode changes are common, governance is the issue.

2. MCP server config changed

MCP connects Cursor to external systems, so configuration changes deserve their own review path.

An `mcp_server_config` event can surface a new server, local stdio execution path, unpinned package, or changed tool connection.

Where an approved MCP inventory exists, compare changes against it. Otherwise, review new execution paths and build the inventory from observed activity.

3. User promoted to admin or owner

Admin and owner promotions are usually low-volume and easy to explain.

`update_user_role` shows when an account gains the ability to change team settings, manage users, create API keys, or weaken controls.

The event stands on its own. If identity, HR, GeoIP, device, or login context exists, attach it before the SIEM so every downstream consumer sees the same event.

4. API key created

API keys create durable access paths.

A team_api_key event can inventory new Admin API credentials that need an owner, storage location, and rotation policy. Cursor's Admin API docs show that the key value is used as the Basic Auth username. Use team_api_key events, not assumptions about the creator account, to track lifecycle changes.

`user_api_key` may be common in engineering-heavy environments. History makes spikes and privileged or dormant actors easier to spot; without it, the events still build credential inventory.

5. Guardrails changed

Cursor team controls shape what the agent can access and what security can observe.

Changes to `team_repo`, `team_rule`, `team_hook`, or `team_command` show weakened repo controls, relaxed agent rules, disabled hooks, or modified team-command policies.

team_command is control-plane telemetry for custom team commands. It is not a shell-command log.

Hook changes deserve special attention. If Cursor Hooks are part of your runtime visibility model, a hook change may mean observability changed.

GETTING THE DATA TO YOUR DESTINATIONS

Authenticate `GET /teams/audit-logs` with a Cursor Admin API key as the Basic Auth username.

Collect Cursor through `GET /teams/audit-logs`, a supported connector, or a data pipeline, then route the records to the destination that will retain and use them. The collector must handle `page` and `pageSize` pagination, the 20-request-per-minute per-team limit, the 30-day maximum request range, retries, and gap state. Export data before the source window closes.

CHAPTER TAKEAWAY

Cursor audit logs give you the control-plane timeline, not the agent session. Start with Privacy Mode, MCP configuration, admin promotions, API keys, and weakened repo, rule, hook, or team-command controls. Endpoint, network, Git, CI/CD, and Hook data can establish execution and impact.

GitHub Copilot Audit Logs

Standard GitHub Copilot-related audit records capture changes to access, organization and enterprise settings, content exclusions, coding-agent configuration, MCP policy, firewall allowlists, and agent activity on GitHub. They do not contain complete local IDE or CLI sessions.

SOURCE BOUNDARY

Copilot can read repository context, work from issues and pull requests, change code, and run commands. It can also call external tools through MCP. The reachable data includes private source, repository secrets, and credentials exposed to the runtime.

A compromised user or admin account can use that account's Copilot entitlements and the repositories and features it can reach. Prompt injection is another path: an attacker can bury instructions in an issue, pull request, file, or tool output and steer an agent toward data theft or an unsafe change. Researchers have demonstrated the impact through patched vulnerabilities and controlled proofs of concept: private-code and secret exfiltration through Copilot Chat, token theft and arbitrary code execution through poisoned VS Code chat contexts, and a backdoor inserted through a Copilot-generated pull request .

The audit log establishes control changes and supported GitHub-hosted activity, not the full session. It can show when access expanded and give investigators a timeline to join with other evidence. GitHub retains that history for 180 days. Export it if the investigation window may be longer.

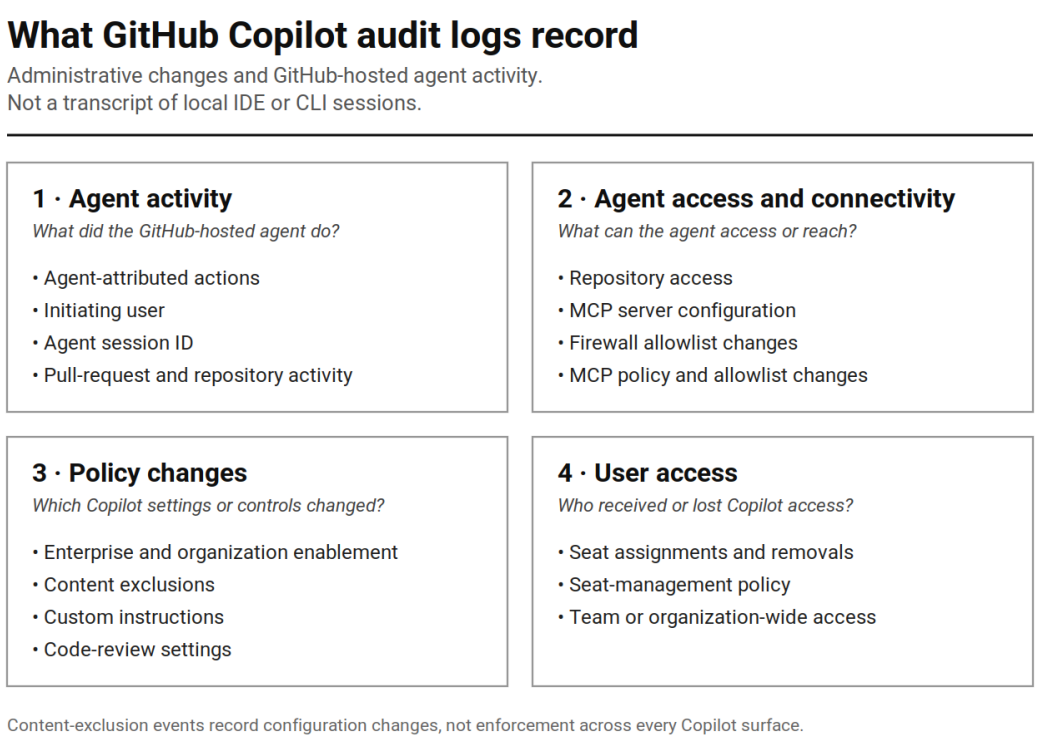
WHAT THE SOURCE CAPTURES

Evidence area	What is recorded	Key fields or identifiers	Security value
Plan and access	Licensing, seats, enablement, organization, and enterprise settings	`action:copilot`, actor, organization and enterprise IDs	Access and administrative timelines
Repository and agent policy	Cloud-agent enablement, repository settings, and custom instructions	`copilot.swe_agent_*`, repository identifiers	Agent capability and scope changes
MCP	Agent MCP configuration and registry policy	`copilot.swe_agent_mcp_config_updated`, `mcp_registry.*`	External tool governance
Firewall allowlists	Changes to destinations available to covered agent processes	`copilot.swe_agent_firewall_allowlist_updated`	Connectivity-control review
Content exclusions	Changes to excluded paths and review behavior	`copilot.content_exclusion_changed`, `excluded_paths`	Policy and data-boundary monitoring
Agent activity	Supported actions performed by agents on GitHub	`actor_is_agent`, user, `agent_session_id`, repository fields	Repository investigation pivots

WHAT THE SOURCE DOES NOT CAPTURE

Missing evidence	Why it matters	Where to look instead
Complete IDE or CLI session	Standard audit records do not reconstruct local interactions	Copilot OTel where supported
Prompt, response, or generated code	Standard audit records cannot establish local content; eligible preview streams may contain API bodies	Explicitly enabled OTel content or source-system evidence
Local files and tool arguments	The record omits the complete local context and command path	Endpoint, Git, IDE, and CI/CD evidence

Missing evidence	Why it matters	Where to look instead
Complete network activity	The agent firewall covers only documented processes and surfaces	Endpoint, proxy, DNS, and firewall data
Universal exclusion enforcement	Content exclusions do not cover every Copilot surface	Current configuration and surface-specific evidence



Content-exclusion events record configuration changes, not enforcement across every Copilot surface.

The records fall into two groups:

- **Plan and configuration events** record changes to Copilot access, licenses, settings, policies, and agent configuration. Use `action:copilot` to find this family.
- **Agent activity events** record actions that supported coding agents perform on GitHub. Use `actor:Copilot`. In these records, `actor_is_agent` is `true`, `user` identifies the person who initiated the

activity, and `agent_session_id` appears only when an agent session generated the event.

These are Copilot-related records inside GitHub's audit log, not a separate Copilot log. Neither group reconstructs a local IDE or CLI session.

Limited streamed request and response bodies

GitHub separately offers a public-preview audit streaming dataset for Enterprise Managed Users and enterprises with data residency on GitHub Enterprise Cloud. Eligible records can include request or response type, user and enterprise IDs, endpoint, request ID, and a JSON body. Bodies may be truncated at 1 MB. Treat this as a separate sensitive stream. It does not change the standard searchable audit log's local-session limits.

Covering the local session gap

Teams can close part of this gap for Copilot Chat in VS Code and Copilot CLI by exporting OpenTelemetry data to an OTLP-compatible collector. This adds traces and metrics for agent interactions, model calls, tool executions, token usage, and errors. Prompt, response, and tool-argument content is excluded by default and requires explicit opt-in.

Sample record

Example: an administrator enabled Copilot cloud agent for a private repository (`copilot.swe_agent_repo_enabled`).

```JSON
{
  "@timestamp": 1784563385002,
  "_document_id": "7b3312ae-8a94-7e9c-d9ce-5c4a01c710fc",
  "action": "copilot.swe_agent_repo_enabled",
  "actor": "alice-admin",
  "actor_id": 627352,
  "actor_is_agent": false,
```

```
  "actor_is_bot": false,
  "business": "example-enterprise",
  "business_id": 78586,
  "created_at": 1784563385002,
  "operation_type": "modify",
  "org": "example-org",
  "org_id": 27431506,
  "owner": "example-org",
  "owner_type": "Organization",
  "public_repo": false,
  "repo": "example-org/payments-service",
  "repo_id": 88342177,
  "user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X
10_15_7) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/150.0.0.0 Safari/537.36"
}
```

Four fields matter in this sample:

- `actor_id` is the stronger field for tracking a GitHub account over time because the actor login can change.
- `actor_is_agent` and `actor_is_bot` are both `false`, so this is an admin change, not an agent-session event.
- `public_repo` is `false`, so the target is private.
- `user_agent` identifies the client when GitHub includes it.

Common documented events in GitHub's organization audit-event reference and enterprise audit-event reference include:

- **Seats and licensing:** `copilot.cfb_seat_added`, `copilot.cfb_seat_assignment_created`, `copilot.cfb_seat_assignment_unassigned`,

`copilot.cfb_seat_cancelled`, and `copilot.cfb_seat_management_changed`.

- **Organization and feature settings:** `copilot.cfb_org_settings_changed`, `copilot.cfb_enterprise_settings_changed`, and `copilot.plan_changed`.
- **Content exclusions and review behavior:** `copilot.content_exclusion_changed`, `copilot.custom_instructions_updated`, and `copilot.code_review_repository_settings_updated`.
- **Agent access and connectivity:** `copilot.swe_agent_repo_enabled`, `copilot.swe_agent_repo_enablement_updated`, `copilot.swe_agent_mcp_config_updated`, and `copilot.swe_agent_firewall_allowlist_updated`.
- **MCP registry:** `mcp_registry.settings_update`, `mcp_registry.allowlist_add_server`, `mcp_registry.allowlist_remove_server`, `mcp_registry.allowlist_assign`, and `mcp_registry.allowlist_remove_assignment`.

This is not a complete list. Audit schemas change as GitHub ships features, and event availability varies by plan and configuration.

COVERAGE LIMITS

GitHub's standard Copilot audit records do not include local client-session data. They omit: local prompts or responses, generated code, local context files, terminal commands, tool arguments, or complete network activity.

Content exclusions have narrower coverage than the event name suggests. GitHub says they do not apply to Copilot CLI, Copilot cloud agent, or Agent mode in Copilot Chat in IDEs. If an organization excludes `/src/payments/**`,

supported inline suggestions and chat modes should ignore those files after the setting propagates.

The exclusion does not stop unsupported surfaces from reading or changing the files, and changes can take up to 30 minutes to reach IDEs. A `copilot.content_exclusion_changed` event proves the configuration changed. It does not prove every Copilot surface was blocked.

USE CASES

Agent activity and changes to access, connectivity, or policy usually matter more than routine licensing. Repository sensitivity and admin workflow decide priority.

Unexpected agent activity

Agent activity is the most direct evidence in this source. Activity in a sensitive or unapproved repository gives the investigation a concrete target.

What to flag:

- `actor_is_agent` events in repositories where agent use is uncommon.
- With enough history, activity initiated by a user who has not previously used an agent in that repository.
- Where workflow baselines exist, pull requests, commits, or repository changes outside the expected path.

Agent access and connectivity changes

Repository access, MCP configuration, and firewall allowlists shape what the cloud agent can read, call, or reach. Changes touching sensitive repositories or new external systems deserve review.

What to flag:

- Agent access added to a sensitive repository.
- New third-party MCP servers.
- `mcp_registry` allowlist or policy changes.
- Firewall allowlist updates that introduce new destinations.

The firewall-allowlist event confirms a change but may omit the old and new values. Where available, compare it with the current configuration, an earlier event, or an approved change record.

Do not treat the agent firewall as universal egress control. GitHub says it covers processes started by the agent through its Bash tool inside the GitHub Actions appliance. It does not cover MCP servers, configured setup steps, or processes outside that appliance, and GitHub notes that sophisticated attacks may bypass it.

Policy and content-exclusion changes

Copilot enablement, code-review settings, custom instructions, and content exclusions change how Copilot operates and which repositories or features can use it.

What to flag:

- Copilot or cloud-agent access expanded to more organizations or repositories.
- Content exclusions removed or narrowed.
- Policy changes made outside the normal administrative workflow.

`copilot.content_exclusion_changed` may show the current `excluded_paths` but not the previous list. Where an earlier event or configuration baseline exists, compare them. The coverage and propagation limits above still apply.

Seat lifecycle changes

Seat events become more useful when identity or employment context is available. Without it, they still support access inventory and investigation.

What to flag:

- Seats assigned or restored for terminated users or accounts believed to be inactive.
- Access granted to unexpected contractors, users, or teams.
- Seats restored after an expected removal.

The Copilot seat API's `last_activity_at` field can add usage context, but it is not part of the audit event. Updates can lag by 24 hours, IDE activity appears only when telemetry is enabled, and the value becomes `nil` after 90 days without new activity.

GETTING THE DATA TO YOUR DESTINATIONS

Collect Copilot records through GitHub's audit APIs, audit-log streaming, a supported connector, or a data pipeline. Route the normalized records to the SIEM, data lake, storage, search, analytics, or response system that will use them.

GitHub exposes organization and enterprise audit logs through `GET /orgs/{org}/audit-log` and `GET/enterprises/{enterprise}/audit-log`. Use `action:copilot` for plan and configuration events, `actor:Copilot` for agent activity, and include `mcp_registry.*` if MCP registry changes are in scope. GitHub retains these events for 180 days. Your SIEM may also support native GitHub audit-log streaming or a connector for this source.

Monad's GitHub Copilot Logs input incrementally retrieves GitHub Enterprise audit records and filters to `action:copilot.*`. The optional `bot_actions_only` setting limits results to events initiated by the Copilot bot, and `backfill_start_time` controls historical collection. Monad pipelines can normalize, filter, enrich, and route the resulting records to multiple destinations.

CHAPTER TAKEAWAY

GitHub Copilot audit records combine a control-plane timeline with supported agent activity on GitHub.

Start with unexpected agent activity, new repository access, MCP or firewall changes, and reduced content exclusions. Directory or employment context makes seat changes easier to judge.

Use the audit log for the control-plane timeline. Endpoint, Git, pull-request, CI/CD, and network evidence can establish execution and impact.

Across Local AI Agents and Developer Tools

Part 1 gives you two evidence classes. Claude Code, Claude Cowork, and Codex export runtime OTel. Cursor and GitHub Copilot primarily expose control-plane records.

The risks overlap, but the evidence does not. Port detections by required fields, not by product category.

WHAT EACH SOURCE CAN ESTABLISH

Claude Code, Claude Cowork, and Codex record session, prompt, decision, and result context. Cursor records changes to the controls around its agent. GitHub adds access and policy changes plus supported agent actions on GitHub.

A `tool_decision` can answer who approved an action. A policy-change record can answer who changed the control. Neither substitutes for the other.

PATTERNS THAT CARRY ACROSS SOURCES

The investigation questions repeat. The available evidence does not.

Sensitive access and possible egress

Runtime records may show a sensitive read or outbound command. Session and network evidence can support a candidate match. Cursor and standard Copilot audit records do not contain the local runtime events needed for that method.

Approval and control changes

Claude Code, Claude Cowork, and Codex can expose an approval source. Cursor and Copilot can expose the policy change that controls later actions. Those are different facts.

Deployment and policy drift

Originators, service names, versions, roles, privacy settings, and repo policies surface drift. First-seen and out-of-policy checks need history or an approved inventory. Without either, the same fields build the inventory.

Direct prompt injection may appear when prompt content is collected. Indirect injection often does not. The tool's actions may be the only recorded evidence.

HANDLE CONTENT AS SENSITIVE DATA

Prompt and tool content may be as sensitive as the system the agent touched.

Use metadata when it answers the question. Put raw content behind tighter redaction, access, and retention before it reaches broad downstream systems.

SUPPORTING TELEMETRY

Runtime logs show attempted actions. Control-plane logs show guardrail changes. Endpoint logs can confirm process and file effects; network logs can confirm connections; Git and CI/CD can confirm repository and build impact; connected systems can confirm MCP side effects. Use only what the environment actually has.

Part 2 moves farther from runtime behavior. Its sources cover identity, administration, control changes, content-activity timelines, and feature use.

In Part 2, start with what the audit record proves. Add identity, asset, and change context if those sources exist.

Part 2:

Enterprise AI Platforms

Anthropic Compliance Activity Feed

OpenAI API Platform Audit Logs

Google Workspace Gemini Activity Logs

Anthropic Compliance Activity Feed

SOURCE BOUNDARY

Anthropic's Activity Feed records who did what, from where, and against which organization or resource. That is enough to detect and hunt for identity abuse, privileged changes, bulk deletion, and suspicious Compliance API access. It is not a session or endpoint log, and it does not capture intent or content.

WHY IT MATTERS

Claude organizations contain sensitive conversations, files, projects, settings, and long-lived API credentials. The Activity Feed records activity around those assets.

A stolen user account lets an attacker operate with that user's access. A compromised owner or administrator account can change organization settings or identity controls. A leaked Compliance Access Key can be worse. With `read:compliance_user_data`, it can read chats, files, and projects across the parent organization and every linked organization. With `delete:compliance_user_data`, it can permanently remove them. A key limited to `read:compliance_activities` cannot do either.

Anthropic says Activity Feed events are queryable within one minute and retained for six years. Central collection preserves the timeline outside the product. Identity, endpoint, network, and data-security sources can add context where available.

COLLECTION AND SCOPE

Anthropic Compliance API Activity Feed. Admin API keys can call it, but only keys created after the Compliance API was enabled carry read:compliance_activities.

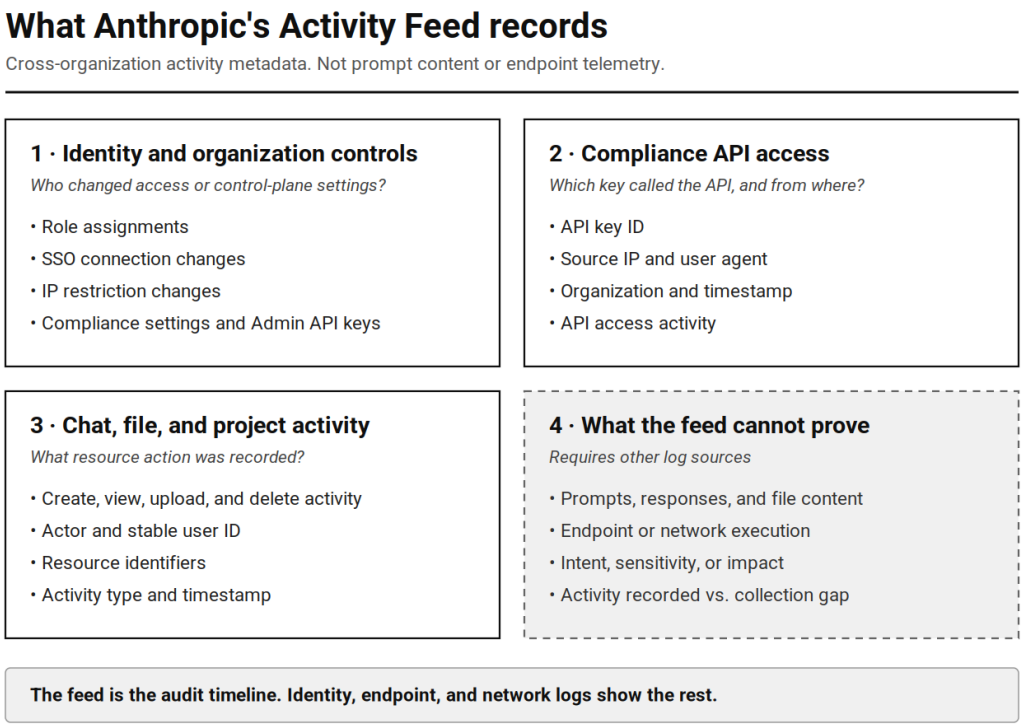
WHAT THE SOURCE CAPTURES

Evidence area	What is recorded	Key fields or identifiers	Security value
Event	Activity type, unique ID, and event time	`type`, `id`, `created_at`	Administrative and activity timelines
Actor	User or API actor plus available source context	`actor.type`, `actor.user_id`, email, key ID, IP, user agent	Attribution and source analysis
Organization	Affected parent or linked organization	`organization_id`, `organization_uuid`	Cross-organization scoping
Resource	Affected chat, file, project, or other object	Resource-specific IDs	Investigation pivots without content
Compliance API	API access activity and available actor context	`compliance_api_accessed`, key ID, IP, organization	Collector and key-use investigations

WHAT THE SOURCE DOES NOT CAPTURE

Missing evidence	Why it matters	Where to look instead
Prompt, response, chat, or file content	The activity record cannot establish what information was processed	Authorized Compliance API content endpoints
Intent	A recorded action may be legitimate, mistaken, or malicious	Identity, endpoint, network, and change records
Complete endpoint activity	The feed cannot establish what occurred on the actor's device	Endpoint and network telemetry
Every permission or consequence	Resource activity does not prove downstream impact	Current configuration and affected source systems

An Activity Feed record can identify the actor, time, organization, resource, source IP, and user agent. Those are investigation pivots, not proof of intent.



SAMPLE RECORD

Sanitized example from the Monad input documentation:

```json
{
  "id": "activity_564447",
  "created_at": "2026-07-26T15:07:52.316385Z",
  "organization_id": "org_9600",
  "organization_uuid": "org_263700",
  "actor": {
    "type": "user_actor",
```

```
    "email_address": "bob.williams@example.com",
    "user_id": "user_920574",
    "ip_address": "77.136.81.9",
    "user_agent": "Mozilla/5.0 (Windows NT 10.0; WOW64;
rv:55.0) Gecko/20100101 Firefox/55.0"
  },
  "type": "claude_chat_created",
  "claude_chat_id": "claude_chat_557075",
  "claude_project_id": null
}
```

Two field rules matter:

- `type` identifies the action. Resource-specific IDs identify the chat, file, project, or other object, not its content. Check the current activity-type reference before hard-coding an enum.
- Fields vary by activity and actor type. For identity joins, use the stable Anthropic user field `actor.user_id`.

CONTENT ACCESS REQUIRES SEPARATE ENDPOINTS

The activity record does not include chat, prompt, response, or file content. Separate content endpoints and more sensitive scopes are required. Claude Code OTel is a different runtime source.

USE CASES

Start with four review paths.

Privileged identity or organization changes

- `role_assignment_granted`, SSO changes, IP restriction deletion, Compliance API settings, and Admin API key lifecycle events define the privileged-change timeline.
- Administrator rosters, IdP history, change records, and key inventory can add context.

Suspicious Compliance API access

- `compliance_api_accessed` exposes the API actor, key ID, source IP, organization, and request pattern. Unfamiliar values need history or an approved inventory.
- Compare key IDs, scopes, source addresses, organizations, and polling schedules to an approved inventory. If none exists, these fields are the starting point. Account for rotation, backfills, failover, and new workers.

Destructive or high-volume content activity

- Count `claude_chat_deleted`, `claude_file_deleted`, and `claude_project_deleted` by actor, organization, source, and time window. Hunt for spikes, mixed deletion types, or a dormant actor becoming active. Burst deletion can be sabotage, cleanup, or ordinary housekeeping.

New or rare activity types

- With enough retained history, track first-seen and frequency by type, actor, and organization. Without that history, group current activity by those fields to establish an initial baseline.
- A first-seen type may reflect an Anthropic release, product rollout, administrator change, or new integration.

GETTING THE DATA TO YOUR DESTINATIONS

Collect the Activity Feed through Anthropic's API, a supported connector, or a data pipeline. Persist cursor or polling state, then route the normalized records to the destinations that will retain and use them.

Monad customers can use the Monad Compliance Activities input, which handles incremental collection and defaults to a 90-day initial lookback.

Otherwise, check whether your SIEM or security data platform already supports the Anthropic Compliance API. You can also collect it with a generic REST collector, scheduled script, or serverless function. Poll Anthropic's `GET /v1/compliance/activities` endpoint using a key with the `read:compliance_activities` scope in the `x-api-key` header. Persist the last successful cursor or polling window so failed requests do not create gaps. Anthropic's integration guidance covers the supported collection patterns.

CHAPTER TAKEAWAY

The Activity Feed is the timeline for identity, administrative, content, and Compliance API activity across linked Claude organizations.

Start with unfamiliar Compliance API access, privileged changes, destructive content activity, and new event types.

Identity, endpoint, network, and data-security sources can add context. They do not turn the feed into session telemetry.

For the managed collection path, start in Monad or book a walkthrough.

OpenAI API Platform Audit Logs

The OpenAI API Platform exposes a structured audit log for user actions and configuration changes across an organization. The current schema documents more than 100 event values across identity, projects, roles, keys, network controls, and tenant settings. OpenAI says the events are immutable once logging is enabled. Organization owners can ask support to disable audit logging.

The highest-value control-plane checks are SCIM, IP allowlists, roles, API call logging, service accounts, and customer-managed keys. Whether any one becomes an alert depends on how the organization uses that control.

WHAT THE SOURCE CAPTURES

Evidence area	What is recorded	Key fields or identifiers	Security value
Actor and session	User or API actor plus documented source context	`actor.type`, `actor.session.ip_address`, user ID or email, API key tracking ID, project	Attribution, source, and access analysis
Identity and groups	Login, logout, invites, users, SCIM, and groups	`login.*`, `user.*`, `scim.*`, `group.*`	Identity lifecycle and deprovisioning review
Roles and projects	Role, assignment, project, and permission changes	`role.*`, `role.assignment.*`, `project.*`	Privilege and resource-scope timelines
Service accounts and API keys	Programmatic identity and credential lifecycle	`service_account.*`, `api_key.*`	Persistence and credential inventory

Evidence area	What is recorded	Key fields or identifiers	Security value
Network controls	IP allowlists, tunnels, certificates, and checkpoints	`ip_allowlist.*`, `tunnel.*`, certificate and checkpoint events	Access-boundary and infrastructure review
Organization controls	Organization, rate-limit, API logging, and external key changes	`organization.updated`, `rate_limit.*`, `external_key.*`	Configuration and data-protection monitoring

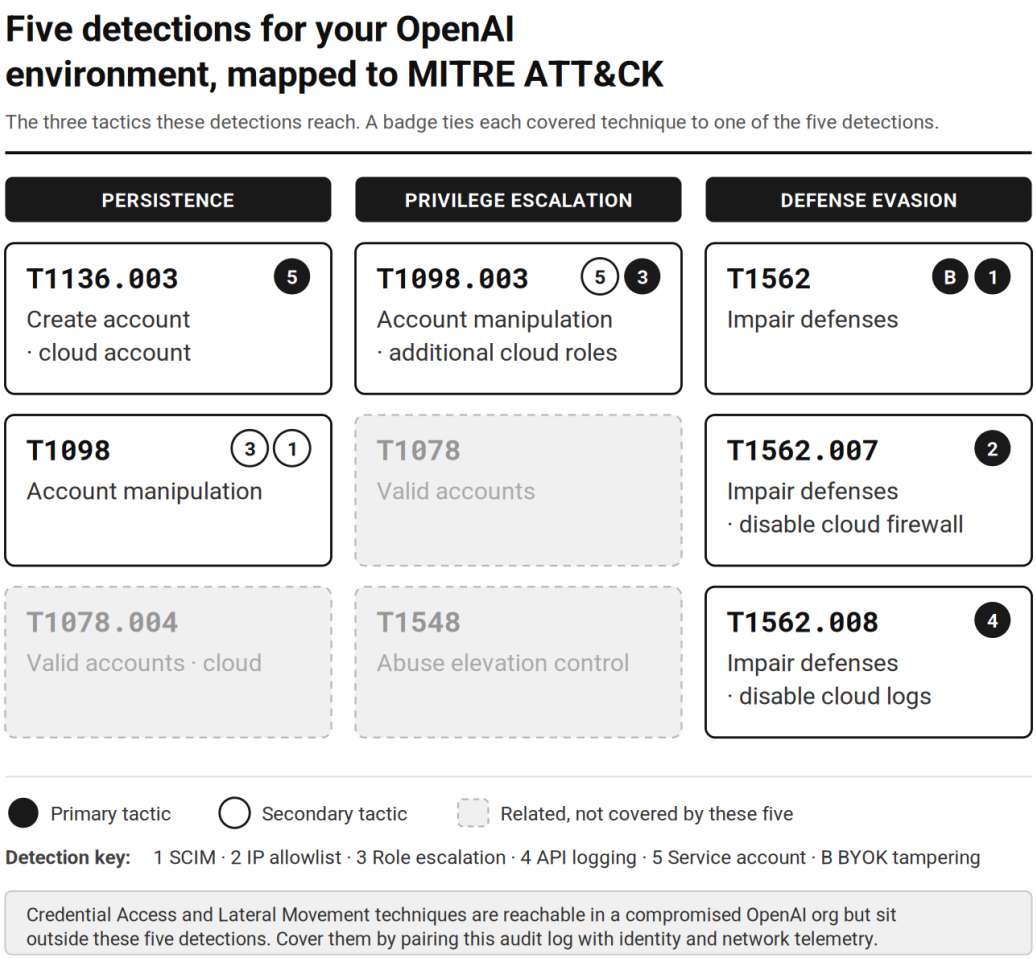
WHAT THE SOURCE DOES NOT CAPTURE

Missing evidence	Why it matters	Where to look instead
Prompt and completion content	The audit stream is separate from API call logging	API call logging where intentionally enabled
Complete workload behavior	An administrative event does not reconstruct model or application runtime	Application, API, endpoint, and network evidence
Every previous configuration value	Some events show the change without full before-and-after state	Earlier event, current configuration, or change record
Employment and asset context	The source does not know team, status, ownership, or criticality	IdP, HRIS, CMDB, and asset inventory

The Audit Log API (`GET /organization/audit_logs`) returns admin and user actions across your OpenAI org. Every entry includes an id, an `effective_at` Unix timestamp, a type field, an actor object, and an optional project scope.

For session-based actions, the actor object documents the source IP plus user ID and email. For API key actions, it documents the key tracking ID and associated user or service-account identity. An optional top-level project object scopes project events.

USE CASES



1. SCIM Disabled

Event: `scim.disabled`

Noise: Usually low when SCIM is an enforced control.

If SCIM drives deprovisioning, disabling it leaves OpenAI access in place until someone removes it manually. It also breaks the IdP path that would normally remove a flagged account.

For organizations that rely on SCIM, `scim.disabled` is a strong alert candidate. Pair it with `scim.enabled` to show toggle patterns.

2. IP Allowlist Disabled or Broadened

Events: `ip_allowlist.config.deactivated, ip_allowlist.deleted, ip_allowlist.updated`

Key fields: `allowed_ips` (on `updated` and `deleted`), `configs[]` (on `config.deactivated`), top-level `actor`

Noise: Usually low. Broadened-range review requires known corporate CIDRs or an earlier configuration.

Where IP allowlisting is an active access control, deactivation removes that network restriction.

Checks:

- Treat `ip_allowlist.config.deactivated` and `ip_allowlist.deleted` as prompt-review candidates when the allowlist is enforced.
- For `ip_allowlist.updated`, review additions such as `0.0.0.0/0`, `::/0`, or prefixes broader than the organization's expected ranges.

3. Privilege Escalation (Role Changes)

Events: `role.updated, role.assignment.created`

Key fields: `role.updated.changes_requested.permissions_added, role.assignment.created.principal_id, role.assignment.created.principal_type, role.assignment.created.resource_id, role.assignment.created.resource_type`

Noise: `role.updated` with `permissions_added` is direct. `role.assignment.created` needs resource or principal context.

A role update or assignment can expand an existing principal without creating a new identity.

`role.updated` exposes `permissions_added`. `role.assignment.created` is less precise: it identifies the principal and resource, not the assigned role.

Checks:

- Review `role.updated` when `permissions_added` is non-empty. Permissions affecting service accounts, API keys, IP allowlists, rate limits, or organization configuration may warrant higher priority.
- Where project criticality and principal-role data exist, use them to prioritize `role.assignment.created`. Without that context, the event still establishes the assignment and affected resource.

4. API Call Logging Disabled or Reduced

Event: `organization.updated`

Key field: `changes_requested.api_call_logging`

Noise: Often low. Privacy or compliance work can produce legitimate reductions.

OpenAI's audit log is separate from API call logging, which controls retention of prompts, completions, and usage metadata. `api_call_logging` can be `disabled`, `enabled_per_call`, `enabled_for_selected_projects`, or `enabled_for_all_projects`. Reducing it removes content that may be needed later.

Review `organization.updated` when `changes_requested.api_call_logging` is reduced. Include `changes_requested.api_call_logging_project_ids` when the value is `enabled_for_selected_projects`. Where change records exist, use them during triage.

5. Service Account Created (Especially Owner)

Event: `service_account.created`

Key field: `data.role`

Noise: Low. Owner service accounts should be exceptional in most orgs.

Service accounts are non-human project identities with API credentials. Their keys default to read and write access across the project's API resources unless restricted. They persist until the account or key is removed. Owner role grants broader project control, not automatic organization-wide administration.

An owner-level `service_account.created` event is a strong review or alert candidate. Additional context can raise priority when:

- the creating actor is itself a service account (rather than a human)
- the creation falls outside your normal provisioning patterns (e.g., off-hours, unusual actor, unexpected project)

For non-owner service accounts, use the record for inventory or review. Actor team and role can improve prioritization where available.

Bonus: External Key (BYOK) Tampering

Events: `external_key.registered, external_key.removed`

Key fields: `id` (the external key configuration ID), top-level `actor`, top-level `project` scope

Applies to organizations using customer-managed encryption keys. These events should be rare outside planned key administration.

External keys are Enterprise Key Management configuration. A registered or removed event proves that EKM configuration changed. It does not prove exposure or unavailability.

Compare `external_key.registered` and `external_key.removed` against approved key inventory and current KMS state where those sources exist. Without them, use every change as an investigation pivot. The audit event does not prove the action was safe.

ENRICHMENT BEFORE THE SIEM

The audit log knows the actor, event, project scope, and session IP. It does not know employment status, asset criticality, admin ownership, or whether the change was expected.

If identity, HR, asset, or threat context exists, attach it before the SIEM. That is the cleanest chokepoint. Enrich once so SIEM rules, analyst searches, manual response, SOAR, and AI SOC workflows receive the same event. In-SIEM joins and response-time lookups remain fallbacks, but they repeat logic and drift.

Identity and asset context may live in Okta, an HRIS, a CMDB, or threat feeds. Pipeline enrichment attaches it once before the event reaches the SIEM.

Actor team and employment status can change the priority of `service_account.created`. The source event still stands when that context is unavailable.

Monad's enrichments run inline before routing. In-SIEM joins and response-time lookups remain available when upstream context is not.

BEYOND THE TOP FIVE

Other uses include API key behavior, rate-limit changes, API key scopes, project archive or deletion, bulk invites, tunnels, checkpoint permissions, certificates, and authentication failures. Behavioral methods require enough history to define expected activity.

GETTING THE DATA TO YOUR DESTINATIONS

Audit logging must be enabled before records are available. An Organization Owner creates the Admin API key used to call `GET /organization/audit_logs`.

Collect the OpenAI API Platform audit log through a native integration, collector, or data pipeline, then route it to the destination that will retain and use it. If you build against `GET /organization/audit_logs`, the collector owns authentication, cursor state, polling, retries, schema changes, and gap detection. OpenAI documents no fixed audit-log retention period or TTL and recommends export for long-term retention. Monad's OpenAI Enterprise Audit Log input handles collection, normalization, enrichment, and routing.

CHAPTER TAKEAWAY

This is a control-plane source. Start with SCIM, IP allowlists, roles, service accounts, and API call logging. Export it if the investigation window may outlive OpenAI's undefined retention.

Google Workspace Gemini Activity Logs

Google Workspace Gemini activity logs show who used Gemini, when, from which IP, in which app, and through which feature surface. They cover the Gemini app and Gemini inside Workspace. They do not expose prompts, responses, file or message IDs, token counts, model IDs, or the exact content Gemini used. Login, Drive, Gmail, Calendar, DLP, and endpoint logs can add context.

SOURCE BOUNDARY

Gemini now ships inside Gmail, Docs, Drive, Sheets, Slides, Meet, and Chat. In January 2025, Google began including Gemini AI features in Workspace Business and Enterprise plans instead of selling separate add-ons.

Gemini follows Workspace permissions, but it can make already-accessible information easier to find, summarize, and act on. The activity record shows feature use without the prompt, response, or exact Workspace content.

In 2025, 0DIN demonstrated that hidden HTML in email could make Gmail's summary output a fake security warning. Miggo later disclosed a now-mitigated Calendar flaw that moved private meeting details into an attacker-visible event. Google has since documented layered defenses against indirect prompt injection. Both cases started with attacker-controlled Workspace content entering Gemini's context. This activity log would not show that content.

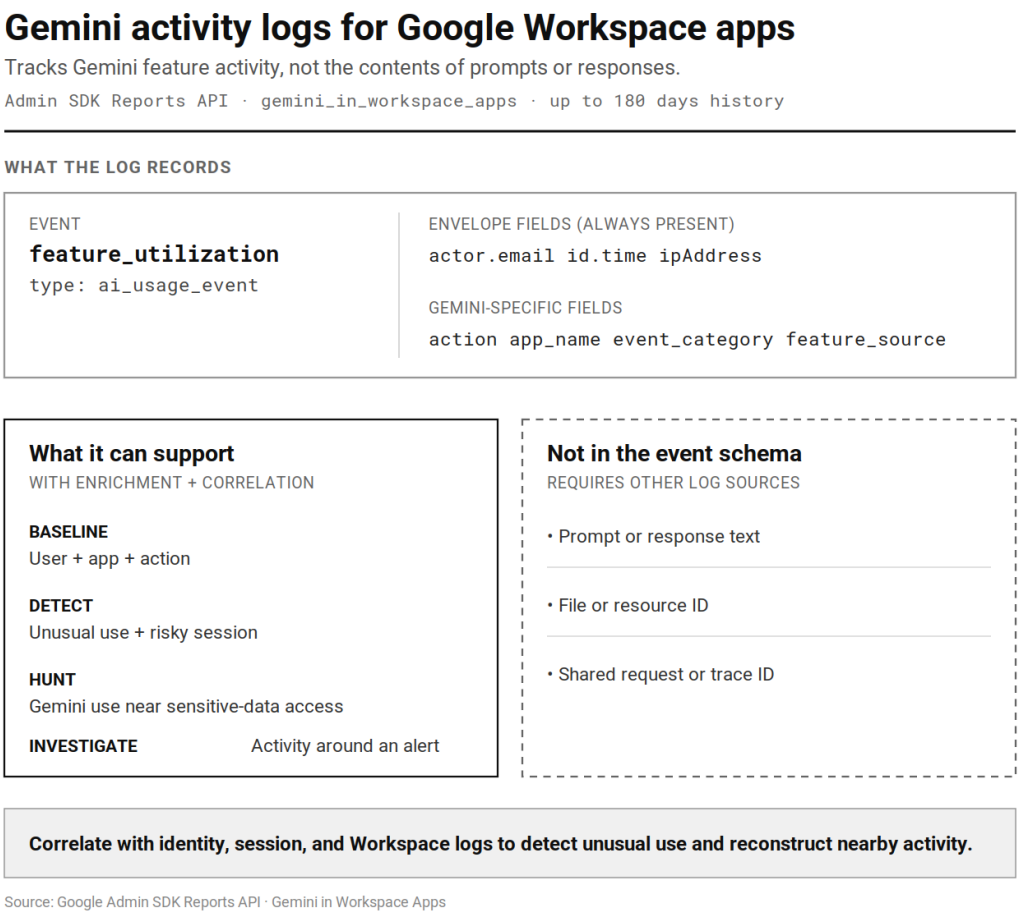
The activity record proves use, not content or outcome. If identity or network context exists, attach it before the SIEM so every downstream consumer sees the same event. Supporting logs remain investigation paths.

WHAT THE SOURCE CAPTURES

Evidence area	What is recorded	Key fields or identifiers	Security value
Actor and time	User, account, event time, and customer context	`actor.email`, `profileId`, `id.time`, `customerId`	Attribution and usage timelines
Source	Source IP and Workspace application	`ipAddress`, `app_name`	Source and application scoping
Feature	Feature source and action	`feature_source`, `action`	Feature inventory and investigation pivots
Category	Active conversation, generation, summarization, unspecified, inactive, or unknown activity	`event_category`	Filtering, baselining, and usage analysis
Envelope	Application name and unique activity qualifier	`applicationName`, `uniqueQualifier`	Collection and record identity

WHAT THE SOURCE DOES NOT CAPTURE

Missing evidence	Why it matters	Where to look instead
Prompt and response	The activity record cannot establish what the user asked or received	Vault for supported Gemini app content
Exact file, email, or calendar object	The Gemini record usually omits the affected resource	Drive, Gmail, Calendar, and other Workspace logs
Model, tokens, and complete outcome	The record cannot reconstruct the model execution	Product-specific operational sources where available
Shared transaction ID	Cross-source time matches remain inferential	Same-user, app, and time-window review
Complete prevention context	Usage evidence does not show whether an inline control inspected or blocked content	DLP, browser, endpoint, and policy logs



Google exposes the gemini_in_workspace_apps application through the Admin SDK Reports API. Documented usage records have `events[].type=ai_usage_event` and `events[].name=feature_ utilization`. The log became available on June 20, 2025, and provides a rolling history of up to 180 days. Export through the GWS Reports API or, on supported editions, Workspace's BigQuery export if you need a longer window.

Each record has two layers. The activity envelope carries time, account, and source IP. The Gemini event adds four key-value parameters under `events[].parameters`.

Sample log record

JSON


```
{
  "kind": "admin#reports#activity",
  "id": {
    "time": "2026-07-14T20:06:54.682664Z",
    "uniqueQualifier":
"a09e6911-dfb6-44bc-488d-83f89faf3169",
    "applicationName": "gemini_in_workspace_apps",
    "customerId": "C0q2w3e4r"
  },
  "etag": "\"etag_example\"",
  "actor": {
    "callerType": "USER",
    "email": "carol.brown@example.com",
    "profileId": "27a481b1-edac-a0a8-c560-fbd3c5456510"
  },
  "ipAddress": "116.74.67.114",
  "events": [
    {
      "type": "ai_usage_event",
      "name": "feature_utilization",
      "parameters": [
        {
          "name": "app_name",
          "value": "drive"
        },
        {
          "name": "feature_source",
          "value": "side_panel"
        },
        {
          "name": "action",
          "value": "generate_apps_search_overlay_suggestions"
        },
        {
```

```
          "name": "event_category",
          "value": "inactive"
        }
      ]
    }
  ]
}
```

This observed record shows a feature_utilization event for Carol Brown in the Drive side panel. `generate_apps_search_overlay_suggestions` is not in Google's current documented action enum. Preserve unknown action values instead of dropping them. The record still shows who used which feature, where, and from what IP, but not the content or outcome.

MAKING SENSE OF `event_category`

`event_category` is the best first filter. Current documented values include `active_conversations`, `active_generate`, `active_summarize`, `active_unspecified`, `inactive`, and `unknown`. The four active values are the clearest direct-use records.

`inactive` means Google did not count the event as active Gemini use. In the sample above, Gemini generated Drive search suggestions in the side panel without the user submitting a prompt. Other background activity can include generating starter prompts, proactive suggestions, or summaries when the side panel opens. Google documents these actions but does not publish how each maps to `event_category`, so `inactive` may still involve some user interaction. This makes it more useful for baselining and investigation context than as a standalone detection signal.

Treat inactive events as context. If storage tiers already exist, they are a lower-cost candidate.

USE CASES

The activity record gives you user, IP, time, app, action, and category. `app_name` tells you which Workspace source to inspect next.

The clearest uses are adoption inventory, first-seen app or action values, unexpected source IPs, and pivots into the Workspace log named by `app_name`. Monad can collect Gemini beside more than 30 Google ecosystem sources. A same-user, same-app, narrow-window match is an investigation lead, not a transaction match; the sources share no request or resource ID.

DRIVE CONTEXT, IF AVAILABLE

Google says Drive can emit one `access_item_content` event for each file Gemini accesses for a user query. These records add the missing file ID, title, type, owner, and visibility. The `originating_app_id` field identifies the Google Cloud project that performed the action.

Drive, DLP, or sharing records can surface related activity for the same user, app, and narrow time window. Without a shared transaction ID, the match remains a lead.

VAULT CONTENT IS LIMITED TO THE GEMINI APP

Google Vault exposes much more content for Gemini app conversations. It can retain conversations, place them on hold, search by account or organizational unit and date, and export the full prompt and response. Results include the conversation title, owner, timestamp, and up to 12 hours of surrounding context.

That coverage does not extend to Gemini inside Gmail, Drive, Docs, or other Workspace apps. Google's supported-data documentation excludes Gemini for Google Workspace data, media files or Gems attached to prompts, and media

generated in responses. Vault can show what a user said to the Gemini app, but not what Workspace content Gemini accessed.

The Google Vault API supports Gemini queries and XML exports for scoped investigations. Monad's Vault Activity input collects the administrative audit trail, not conversation content. It shows search and export activity; identity and admin context can extend the record.

Vault supports eDiscovery and post-event investigation, not live monitoring or prevention. DLP, browser, endpoint, or other inline controls may provide preventive coverage where deployed.

GETTING THE DATA INTO YOUR DESTINATIONS

Google exposes Gemini activity through `Activities.list`:
`GET /admin/reports/v1/activity/users/all/applications/gemini_in_workspace_apps`

Collect the endpoint directly, through a supported Google Workspace connector, or through Monad.

Monad can normalize the parameter array, preserve unknown values, attach user and IP context before the SIEM, filter by `event_category`, and route records by value and sensitivity. If volume warrants and multiple storage tiers exist, inactive activity can move to lower-cost storage.

CHAPTER TAKEAWAY

Gemini activity records prove use. They usually cannot prove content, resource, or outcome. Preserve unknown actions, prioritize active categories, and treat cross-source matches as investigation leads.

Operationalizing AI Tooling Logs

A log source has no operational value while it sits in a vendor API or nested OTLP payload. Collection decides whether the evidence works: parsing, normalization, enrichment, filtering, routing, and access control.

The eight sources do not behave alike. Claude Code, Cowork, and Codex emit nested OTel. Cursor has pagination, rate limits, and a 30-day request window. OpenAI exposes one structured audit endpoint. Gemini reports use without content. Anthropic uses another activity schema. Every source brings its own failure modes.

A vendor API is only potential evidence. The record has to reach something the team can search, alert on, hunt through, and use during an incident.

Without a shared collection layer, every source becomes another collector, parser, cursor, and silent failure mode.

THE PIPELINE CONTROL LAYER

Use the pipeline as the shared chokepoint. Collect each source once. Parse and normalize it once. Attach identity, HR, asset, or network context there if it exists. SIEM rules, analyst searches, manual response, SOAR, AI SOC systems, and other destinations then receive the same event. In-SIEM joins and response-time lookups remain fallbacks, but they duplicate logic and drift.

That chokepoint is also where redaction, filtering, access, and routing belong. Raw prompt and tool content can receive tighter access and shorter retention before it reaches broad downstream systems.

Monad is built for this layer: collection, parsing, normalization, pre-SIEM enrichment, filtering, and routing. The Claude Code and Codex transforms in this book are public. Cowork uses the Claude Code input and routes separately on `service.name=cowork`.

WHERE TO START

Start with the question the team cannot answer today. Runtime action needs runtime evidence. Administrative change needs an audit record. Adoption and usage need activity data. Do not ask one class of log to prove another.

Collect what the source can prove. Keep its limits intact. Put the evidence where the team already works.

If one of these sources is stuck in an API, buried in OTel, or arriving without usable context, bring it to Monad. We'll show you the collection, normalization, and routing path.

Collect it before you need it. Know what it can prove before someone asks.

A NOTE FOR SECURITY LEADERS

AI tooling is a new security log surface, not one product category. Specialist AI security tools may cover part of it, but every environment has its own identities, repositories, connected systems, policies, and data paths.

Security teams still need to know which native sources exist, what each can prove, and where endpoint, identity, network, Git, or SaaS evidence must fill the gaps. Use vendor solutions where they help but the key is to not outsource your understanding of what these AI tools emit natively.

If a high-value source is stuck in an API, buried in OTel, or arriving without usable context, bring it to Monad. We'll show you the collection, normalization, and routing path.

Collect it before you need it. Know what it can prove before someone asks.

About Monad:

Monad is a security data pipeline platform. It collects logs from 350+ sources, parses and normalizes them, enriches events before the SIEM, filters low-value data, and routes each record where it needs to go. Monad is backed by Sequoia Capital and Index Ventures. Learn more at monad.com.

AI tools now read and write code, run commands, search business data, and act through connected systems. What gets logged varies by product.

Claude Code, Claude Cowork, OpenAI Codex, Cursor, GitHub Copilot, enterprise AI platforms, and Gemini in Google Workspace all emit security-relevant records. The hard part is knowing where those records live, what they capture, which identifiers matter, and what never shows up.

This field guide maps those sources and puts the data in context across monitoring, detection, threat hunting, investigations, incident response, governance, and AI asset inventory.

No source tells the whole story. By the end, you will know which records to collect, how to interpret them, which security questions they can answer, and where endpoint, identity, Git, SaaS, and network data must fill the gaps.