AI SECURITY ENGINEERING HANDBOOK · 2026

Chapter 11 · Detection Engineering

Standalone study module for LMS delivery and required reading.

FORMAT	USE	SCOPE	AUDIENCE
Standalone PDF	Study module	Single chapter	Learners

CHAPTER 11

Detection Engineering

HANDBOOK STUDY COMPANION: STUDY FRAME

Use this chapter to build vocabulary, judgment, and role-readiness. Pair it with the Field Guide when you need applied actions, checklists, and control execution.

STUDY FOCUS

STUDY FOCUS	WHY IT MATTERS
Control-failure mapping, behavioral baselines, prompt injection signals, retrieval anomalies, agent action outliers, and alert feedback loops.	Detection work must start from the AI control that can fail, not from generic security logs.

Study Outcomes

- Map AI failure modes to observable signals.
- Explain coverage, alert quality, and false-positive tradeoffs.
- Connect detection findings to incident response and regression testing.

DOMAIN MAPPING

RELATED AIPSA DOMAINS	APPLIED NEXT STEP	WORKBENCH INSTRUMENTS	RELATED SERVICES
Incident Response and AI Observability, Red Teaming and Adversarial Evaluations	Red teaming and adversarial evaluations, Incident response and observability	Runtime Proxy, Adversarial Range	AI Red Team & Adversarial Testing

CERTIFICATION AND ASSESSMENT BOUNDARY

This chapter supports training, diagnostic preparation, scorecards, interviews, and role-readiness evaluation. It does not guarantee credential outcomes.

Anomaly detection without a behavioral baseline is pattern matching against noise. Most AI security programs invest in controls — authorization, approval gates, logging, release gates — and treat detection as something that happens during incident response rather than before it. That ordering means every incident is discovered by its consequences. Detection engineering for AI systems is the discipline of deciding, before incidents occur, what behaviors signal a control failure, what telemetry captures those behaviors, and what response logic fires when they appear.

Anomaly detection without a behavioral baseline is pattern matching against noise. Every incident discovered only by its consequences is a detection failure that preceded the security failure.

HANDBOOK

FIELD USE

Use this chapter during design review. Start with the failure class. Name the control. Name the telemetry. Write the rule. Test the rule. Retest after the system changes.

CORE CONCEPTS

CONTROL-FAILURE MAPPING

Detection logic designed from threat intelligence or generic anomaly heuristics will produce either high false-positive rates or detection gaps, because AI systems vary too much in normal operation to reliably anchor on absolute thresholds or content signatures. Control-failure mapping approaches detection from the architecture itself: for each security control the system implements — retrieval authorization, agent tool permissions, prompt template version pinning, approval gates, output schema validation — identify what observable telemetry signals would appear if that control failed. A retrieval authorization failure produces a specific pattern in retrieval logs. A prompt injection through retrieved content produces a specific pattern in context assembly traces. A tool call that exceeds authorized scope produces a specific pattern in the tool-call audit log. Detection logic derived from specific control failures is more precise, more explainable, and less noisy than detection logic derived from output characteristics.

BEHAVIORAL BASELINE FOR AI SYSTEMS

AI system behavior varies across users, sessions, query types, and time. Anomaly detection requires a baseline that characterizes normal behavior at the relevant granularity: tokens per session, tool calls per session, retrieval queries per session, output refusal rates, tool argument value distributions, retrieval source distributions, and session duration distributions. Detection rules fire when observed behavior deviates from the baseline by a defined threshold. Without a baseline, detection rules set on absolute thresholds will reflect natural system variance rather than security signals. Building a valid baseline requires instrumentation that captures session-level behavioral patterns for a period long enough to characterize weekly usage cycles, high-load periods, and user population diversity.

PROMPT INJECTION DETECTION

Direct detection of injected instructions in text is not a reliable primary layer for indirect injection, because injection patterns are unbounded and arrive through retrieval and tool outputs that the user did not write. Effective prompt injection detection focuses on behavioral signals produced when injection succeeds: model output that diverges from the expected task schema, tool calls with argument values sourced from retrieved content rather than from user input, output refusal rate spikes following specific retrieval patterns, or session behavior that matches known injection outcomes. Signature-based input scanning remains appropriate for detecting direct injection attempts from the user turn, but it must be complemented by behavioral detection at the output and action layer to cover indirect injection through the retrieval path.

AGENT BEHAVIORAL OUTLIER DETECTION

Agent systems should produce tool call patterns consistent with user-initiated workflows. Outlier detection looks for: tool call sequences that match no known workflow pattern, tool call argument values sourced from retrieved content rather than user input, tool calls executed at atypical times or volumes, tool calls to resources outside the user's normal scope, or multi-step action chains that combine tool use in ways that produce high blast-radius outcomes. These signals suggest possible

confused-deputy attacks, prompt injection activating through agent tool use, or model behavioral drift following a provider-side change. Detecting at the tool call layer — before actions complete — is more valuable than detecting at the output layer, because some agent actions are irreversible.

TELEMETRY GAP DETECTION

The absence of expected telemetry is itself a security signal. If a production system consistently fails to emit retrieval traces, tool-call audit records, or output filter decisions, that gap may reflect a logging failure, a misconfigured component, or a pathway through the system that bypasses instrumented code paths. Telemetry completeness monitoring checks that the expected event types arrive at the expected rates for active sessions and alerts when specific trace categories drop below threshold. This is the detection equivalent of the telemetry validation exercise: the detection program monitors the monitoring infrastructure, not only the application behavior.

THE PRACTITIONER'S CHALLENGE

The political challenge is that detection engineering is treated as an operations function that follows a working system, not as a design function that precedes deployment. Teams build and ship AI features, then ask "what should we alert on?" after the system is in production. The practitioner must make the case that detection coverage is a launch prerequisite alongside authorization controls and logging — that a system without detection coverage is not a secured system, it is a system waiting to discover its incidents through customer reports.

The structural challenge is that AI detection crosses multiple teams and systems. The application team knows the AI workflows. The platform team owns the telemetry infrastructure. The detection engineering team writes detection rules. Security operations responds to alerts. If detection requirements are not communicated from the application team to the platform team during system design, the telemetry required for detection rules may not exist when detection engineering starts writing them. The data flow from control-failure mapping to telemetry requirements to detection rule design requires explicit handoffs at each boundary.

The technical challenge is that AI systems are non-deterministic, which makes the baseline a moving target. Usage patterns shift as the user population grows. Model provider updates change output characteristics. New workflows produce new tool call patterns. A behavioral baseline built from the first month of production data may not accurately represent the system six months later. Baseline maintenance — periodic recalibration, segmentation by user population and query type, detection rule regression testing against updated baselines — is ongoing operational work, not a one-time setup task.

HOW TO APPROACH IT

- Start with a control-failure detection matrix before writing any detection rules. List every security control in the AI architecture: retrieval authorization filters, agent tool permission enforcement, approval gates, output schema validators, model version pinning, prompt template version controls, rate limits. For each control, document what telemetry fields it produces, what signal would appear in those fields if the control failed, and what rule would fire on that signal. The matrix produces a concrete detection backlog with direct mappings to architectural risk. Detection gaps in the matrix are risk exposures.

- Establish behavioral baselines before activating anomaly detection rules. For each behavioral dimension — tokens per session, retrieval queries per session, tool calls per session, refusal rate, retrieval source distribution — collect at least four weeks of production telemetry, segment by user population and query type where usage patterns differ significantly, validate the baseline against known-normal sessions, and document the variance characteristics that inform threshold setting. Activate anomaly rules only after the baseline is validated. Set initial thresholds conservatively and tune based on observed false-positive rates during a monitored calibration period.

- Design retrieval anomaly detection as a separate concern from application request monitoring. Retrieval anomalies — cross-namespace queries, high-volume sessions, source distribution shifts, high-score retrieval of documents not matching query intent — require chunk-level retrieval traces that are not part of standard request logs. Write retrieval anomaly rules against chunk-level fields: tenant identifier on retrieved chunks, similarity score distributions, source document identifiers, and authorization decision records. A single retrieval anomaly rule operating on the right fields is more valuable than a dozen rules operating on aggregated response metrics.

- Build agent behavioral outlier detection using session-level tool call patterns rather than single-call thresholds. A single unexpected tool call may be legitimate user-directed behavior. A sequence of tool calls that forms an unusual chain — or that combines capabilities in ways that produce high blast-radius outcomes — is more likely to be an injection-influenced action. Define the expected tool call patterns for each primary workflow and build detection rules that evaluate sequences, not individual calls. Include argument-value sourcing analysis where telemetry supports it: a tool call whose arguments were derived from retrieved content rather than user input is a stronger injection signal than an unusual tool call alone.

- Design the feedback loop between incident response and detection engineering as an explicit process, not an informal one. After each AI security incident, the detection engineering team reviews: whether the incident was caught by existing rules, at what point in the incident timeline detection fired, whether it should have been caught earlier, and what new or modified rule would have fired sooner. New detection logic derived from incidents is written with test cases that would have caught the original incident, reviewed, and deployed with an incident reference. Over time, detection coverage reflects the actual failure modes the system has experienced rather than theoretical models.

- Monitor alert quality as an operational metric. Track true-positive rate, false-positive rate, time-to-acknowledge, and time-to-close for each detection rule. Rules that consistently produce false positives are tuned or retired rather than left in place and ignored. Responders who begin filtering alerts because of noise lose the detection coverage that the alert was designed to provide. Alert quality monitoring surfaces this degradation before it becomes invisible in operational habit.

OUTPUTS AND DELIVERABLES

- The design artifacts are the **control-failure detection matrix**, **behavioral baseline specification**, and **detection coverage map**. The control-failure matrix maps each security control to its telemetry fields, failure signals, and detection rule. The baseline specification documents the dimensions, segmentation approach, validation method, and update cadence for each behavioral baseline. The coverage map shows which control failures have active detection rules and where coverage gaps exist.

- The operational artifacts are the **detection rule library**, **alert severity and escalation specification**, and **alert quality tracking dashboard**. The rule library contains all active detection rules with their test cases, expected true-positive scenarios, known false-positive patterns, and review owners. The severity and escalation specification defines the response SLA and escalation path for each rule. The quality tracking dashboard monitors true-positive rates, false-positive rates, and response latency over time.

- The process artifacts are the **detection feedback protocol**, **baseline maintenance schedule**, and **detection coverage review record**. The feedback protocol defines how incidents are reviewed for detection improvements, how new rules are written and tested from incident findings, and how improvements are tracked to deployment. The maintenance schedule defines when baselines are recalibrated and how rule thresholds are updated. The coverage review record documents periodic assessments of the control-failure matrix against architectural changes.

COMMON FAILURE MODES

- **Detection Without Baselines:** Anomaly rules fire on absolute thresholds set without reference to observed normal behavior. The thresholds are either too low — producing alert fatigue that trains responders to ignore signals — or too high — set conservatively to reduce noise, such that real incidents fall below the detection threshold. Neither condition produces operational security value. Fix: build behavioral baselines before activating anomaly detection rules; derive thresholds from baseline variance, not from judgment calls about reasonable limits.

- **Output-Only Monitoring:** The detection program monitors generated answers for policy violations, unsafe content, or sensitive data patterns, but does not monitor retrieval traces, tool-call logs, approval decisions, or session-level behavioral patterns. The program catches direct output problems while missing retrieval authorization failures, agent action outliers, and prompt injection events that produce compliant-looking output with security-relevant side effects. Fix: build the control-failure detection matrix and verify that each control failure class has at least one detection rule operating on the relevant telemetry, not only on output content.

- **Signature-Only Injection Detection:** Detection logic scans input and retrieved content for known injection phrases, delimiters, and role-boundary syntax. Known-pattern detection catches naive injection attempts while missing indirect injection through semantic framing, multi-chunk delivery, or delayed activation across conversation turns. Fix: complement signature detection with behavioral detection at the output and action layer — tool call patterns, output schema deviations, and session behavioral anomalies that appear when injection succeeds.

- **No Feedback Loop:** After incidents are investigated and resolved, detection logic is not updated to catch the same failure class in future sessions. Each incident closes with a narrative summary. The detection program does not reflect the actual failure modes the system has experienced. Fix: define the feedback protocol explicitly; require that each AI security incident produces at least one detection improvement expressed as a rule with test cases, reviewed and deployed by a named owner with a defined timeline.

IMPLEMENTATION CHECKLIST

- [ ] Build a control-failure detection matrix mapping each AI security control to its failure signals and detection rules.
- [ ] Collect behavioral baseline data before activating anomaly detection rules; validate baselines against known-normal sessions.
- [ ] Design retrieval anomaly detection rules operating on chunk-level retrieval trace fields.
- [ ] Build agent behavioral outlier detection using session-level tool call sequence patterns.
- [ ] Design behavioral injection detection rules that fire on output schema deviations and tool call argument sourcing patterns.
- [ ] Build telemetry completeness monitoring that alerts when expected trace event types drop below threshold.
- [ ] Define alert severity tiers, escalation paths, and response SLA targets for each detection rule.
- [ ] Define and run the detection feedback protocol after each AI security incident.

RELATED READING

- Handbook chapters: Chapter 10 (Logging and Telemetry) for the trace schema and telemetry design that detection rules operate against; Chapter 12 (Incident Response) for the investigation and improvement cycle that detection engineering feeds; Chapter 6 (Agentic Permissions) for agent tool permission controls that behavioral outlier detection monitors.
- Field Guide: Incident Response and AI Observability for detection handoff, trace evidence, and control-failure review.