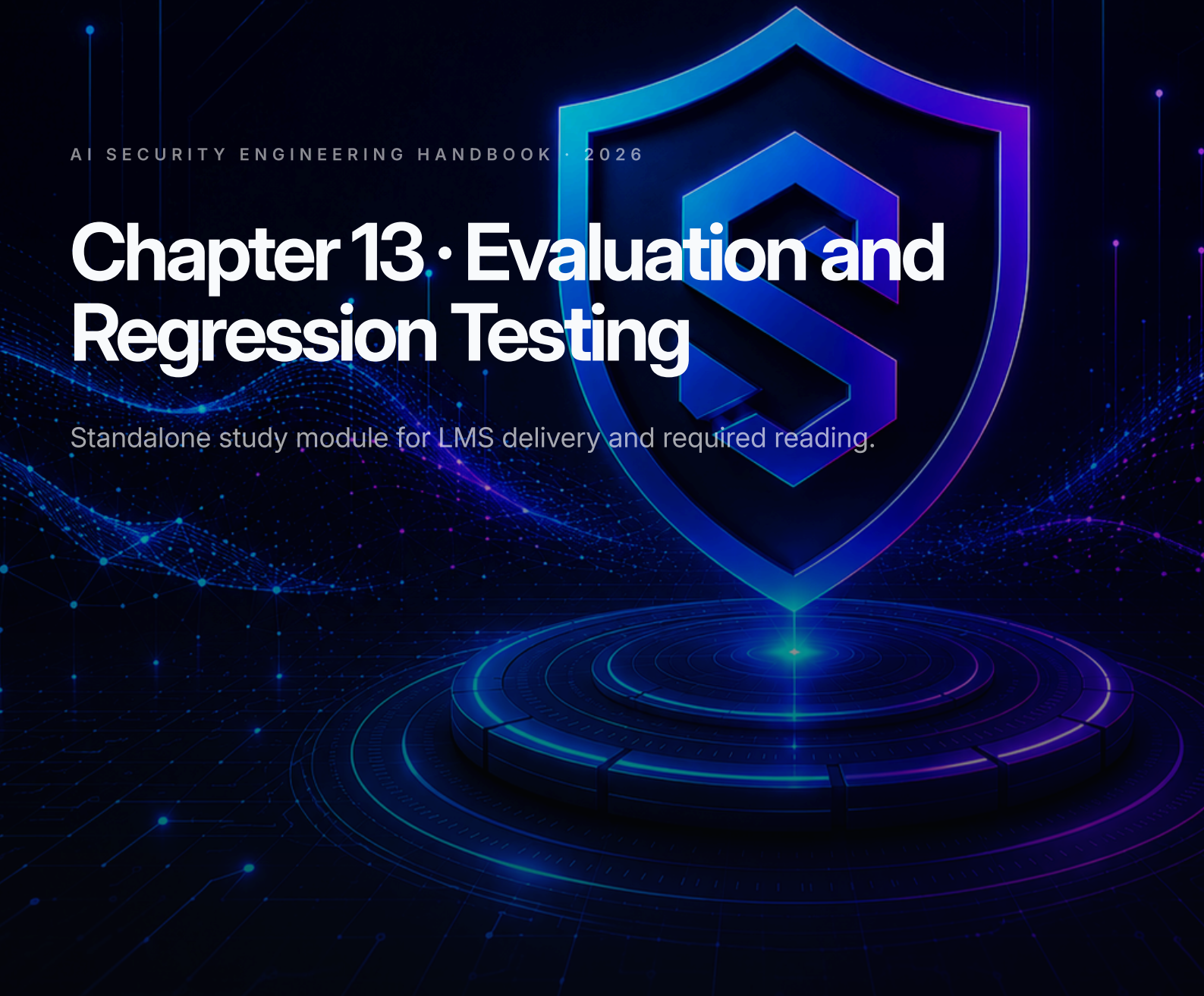
AI SECURITY ENGINEERING HANDBOOK · 2026

Chapter 13 · Evaluation and Regression Testing

Standalone study module for LMS delivery and required reading.



FORMAT	USE	SCOPE	AUDIENCE
Standalone PDF	Study module	Single chapter	Learners

CHAPTER 13

Evaluation and Regression Testing

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
Eval suite design, severity rubrics, red-team scope, regression conversion, release gates, and closure evidence.	Evals become security evidence only when they map to misuse cases, controls, and release decisions.

Study Outcomes

- Describe the difference between demos, evals, red teaming, and regression tests.
- Explain how findings become closure and release evidence.
- Use severity and coverage language without overclaiming.

DOMAIN MAPPING

RELATED AIPSA DOMAINS	APPLIED NEXT STEP	WORKBENCH INSTRUMENTS	RELATED SERVICES
Red Teaming and Adversarial Evaluations	Vendor risk and procurement	Adversarial Range, Training path	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.

Most AI red team exercises produce a report. The report describes what the team found, maybe includes some screenshots, and recommends fixes. Then the assessed team decides which findings matter. That is not adversarial evaluation; it is advisory with a dramatic aesthetic. The difference between a red team exercise and an adversarial control is whether the findings produce regression tests, whether those tests block future releases, and whether closure requires evidence rather than conversation.

The difference between a red team exercise and an adversarial control is whether the findings produce regression tests, whether those tests block future releases, and whether closure requires evidence rather than conversation.

HANDBOOK

An eval program is not a one-time exercise. It is a continuous control loop: run tests, identify findings, convert findings into regression cases, update release gates, and repeat. The loop's value compounds with each iteration as the test suite grows to cover discovered failure classes and the release gate reflects current risk knowledge.

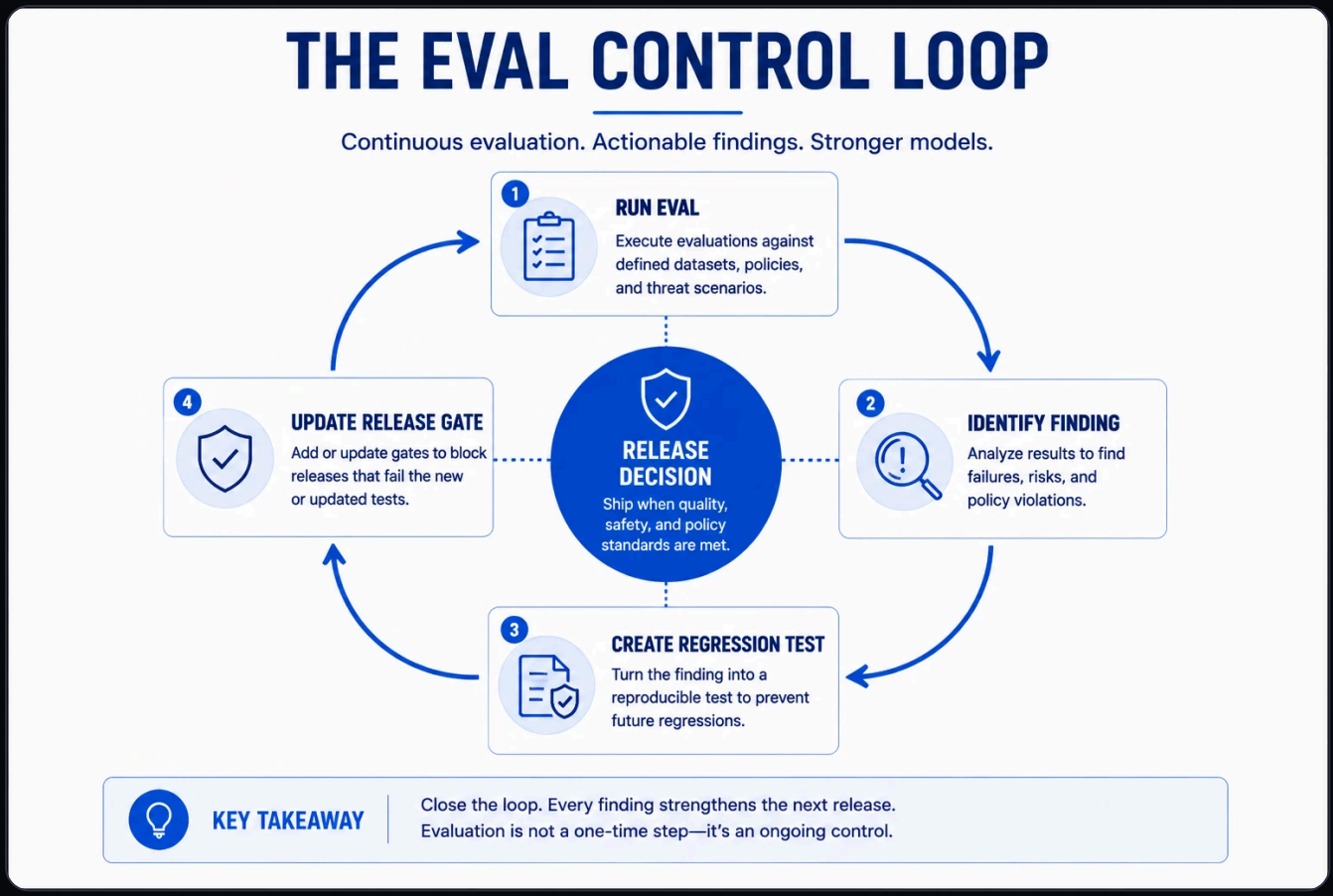


FIGURE 1: FIGURE 13: EVAL CONTROL LOOP — RUN EVAL, IDENTIFY FINDING, CREATE REGRESSION TEST, UPDATE RELEASE GATE — WITH THE RELEASE DECISION AT THE CENTER AS THE GOVERNING OUTCOME THE LOOP CONTINUOUSLY INFORMS

CORE CONCEPTS

EVALS AS RELEASE CONTROLS

An eval becomes a control when it has an owner, expected behavior, severity, pass/fail threshold, execution cadence, and release consequence. A test that runs after launch and produces a dashboard is useful, but it is not a release gate unless failure changes the shipping decision. AI evals should cover the deployed system surface, not just raw model behavior. For a RAG assistant, that means testing retrieval, context assembly, citations, and output behavior together. For an agent, it means testing tool arguments, authorization decisions, approvals, and side effects.

HUMAN RED TEAMING

Human red teams are strongest where judgment, creativity, and chained reasoning matter. They discover failure modes that automated suites do not yet represent: indirect injection through realistic documents, policy bypass through workflow context, multi-step agent abuse, or unsafe behavior emerging from user interaction. Human red teaming should be scoped, severity-rated, and evidence-rich. Its most valuable output is not only the report; it is the new set of test cases, controls, and architectural questions the exercise creates.

SEVERITY RUBRICS BEFORE TESTING

Severity definitions must exist before findings are delivered. Critical, high, medium, low, informational, and out-of-scope categories should be tied to impact, exploitability, affected users, data sensitivity, action authority, reversibility, and control failure. If severity is negotiated after the finding appears, the assessed team can unconsciously downgrade uncomfortable results. A pre-agreed rubric makes closure disciplined and reduces political friction. It also lets leadership understand which failures block release.

PROMPT ATTACK LIBRARIES

A prompt attack library is a maintained body of adversarial scenarios, payloads, expected behaviors, and reproduction notes. It should cover direct prompt injection, indirect prompt injection, context poisoning, jailbreak chains, retrieval poisoning, policy bypass, unsafe output, sensitive disclosure, and tool misuse. The library should be versioned and mapped to product surfaces. It should grow after incidents, red-team exercises, architecture changes, and new threat intelligence. A prompt library is not a bag of tricks; it is test data for a security control.

EVIDENCE RETENTION AND CLOSURE

Testing only matters operationally if evidence survives the exercise. Eval outputs, red-team traces, model versions, prompt templates, retrieved sources, tool-call logs, severity decisions, remediation tickets, and retest results should be stored as security evidence. Closure should require a passing retest, a design change, a compensating control, or explicit risk acceptance. A finding closed because "the team says it is unlikely" is not closure. It is a conversation recorded as a decision.

THE PRACTITIONER'S CHALLENGE

The political challenge is that red-team findings can embarrass product teams. AI systems often produce strange, vivid, and screenshot-friendly failures. Without agreed severity and scope, stakeholders may argue about whether the finding is "realistic," whether the tester was unfair, or whether the model was merely being creative. The practitioner has to keep the discussion grounded in pre-agreed criteria and production impact.

The structural challenge is that evals often live outside normal release engineering. A model team may run model-quality benchmarks, product engineering may run unit tests, security may run prompt attacks manually, and GRC may ask for evidence separately. If those workflows are disconnected, no one can say whether a model update passed the security suite before release. A useful eval program must connect security testing to CI/CD, change management, and evidence retention.

The technical challenge is writing tests that represent production behavior. Generic jailbreak examples are easy to collect, but production failures often depend on user roles, retrieval content, tool permissions, prompt templates, streaming behavior, and model versions. A system can pass a generic benchmark while failing against the exact workflow customers use. The practitioner must test the system, not just the model.

HOW TO APPROACH IT

- Start with the production surfaces. Identify the AI workflows that need evaluation: chat, RAG, summarization, code generation, agent tool use, customer support, internal search, decision support, or external communication. For each surface, define user roles, data sources, model versions, prompt templates, tools, outputs, and release triggers. Do not start from a public benchmark and assume it maps to your product.

- Next, define the severity rubric. Write examples for critical, high, medium, low, informational, and out-of-scope findings in your environment. Include data disclosure, unauthorized retrieval, unsafe tool execution, irreversible external action, policy bypass, sensitive output, hallucinated citation, and unsupported claim scenarios where relevant. Make the rubric visible before testing starts. A good rubric gives testers and product teams the same language for impact.

- Then build the eval suite around behaviors that should not regress. For each test case, record the surface, scenario, input, required context, expected behavior, severity, regression flag, owner, and release consequence. Some tests should be deterministic pass/fail checks; others may require evaluator judgment. Where model non-determinism matters, run multiple samples and define how failure is counted. The goal is not perfect determinism; it is controlled decision-making.

- Run human red-team exercises for discovery. Scope the exercise with model versions, tools, user roles, allowed techniques, exclusions, time box, evidence requirements, and safety boundaries. Encourage testers to explore chains that automated tests do not cover. Require reproduction details rather than just screenshots. At the end, classify findings against the severity rubric and decide which ones become regression tests.

- Convert findings into durable controls. A prompt injection finding might become an eval case, a retrieval filter test, a prompt template change, or an output validation rule. An agent misuse finding might become a tool policy constraint, an approval gate, a sandbox limit, and a trace requirement. A citation failure might become a source-support validation test. The conversion step is where red teaming becomes a control rather than an event.

- End with evidence and cadence. Decide when evals run: pull request, prompt change, model update, retrieval index change, tool permission change, release candidate, scheduled regression, or after incident remediation. Store outputs in a location that supports audits and customer security reviews. Report trends: failures by severity, time to remediate, recurring classes, release blocks, and open risk acceptances.

OUTPUTS AND DELIVERABLES

- The core testing artifacts are the **eval suite design**, **prompt attack library**, and **production surface map**. The surface map ties tests to real workflows, user roles, data sources, tool permissions, and model versions. The attack library provides reusable adversarial cases with expected behavior, severity, and reproduction notes. The eval design makes those cases operational by defining execution cadence, pass/fail thresholds, sampling strategy, ownership, and release consequences.

- The red-team artifacts are the **red-team scope document**, **severity rubric**, and **finding classification guide**. The scope document prevents argument after delivery by naming included systems, threat actors, allowed techniques, exclusions, time box, and evidence format. The severity rubric establishes impact categories before testing starts. The classification guide helps separate capability limitation, quality failure, safety issue, privacy concern, and security finding so closure follows the right path.

- The evidence artifacts are the **eval run record**, **red-team evidence package**, **closure record**, and **regression conversion log**. Eval run records should include model version, prompt template, system configuration, test case version, outputs, result, and release decision. Red-team evidence packages should preserve prompt, context, retrieved sources, tool calls, outputs, timestamps, screenshots where useful, and tester notes. Closure records should show remediation, retest, exception, or risk acceptance, while the conversion log tracks which findings became permanent tests or controls.

COMMON FAILURE MODES

- **Report Without Regression:** The red team delivers findings, but no tests or release gates change afterward. This happens when the exercise is treated as an assessment rather than a control improvement loop. Recover by requiring every valid finding to produce a closure action: regression test, design change, compensating control, or risk acceptance. The report should be the beginning of control improvement, not the end.

- **Benchmark Substitution:** The team uses public benchmarks or model-quality tests as a substitute for production evals. This creates impressive numbers that do not reflect the deployed system's data, tools, prompts, or users. Avoid it by writing tests against real product surfaces and known risk scenarios. Benchmarks can supplement, not replace, production-specific evaluation.

- **Severity Negotiation:** Findings are downgraded after delivery because severity was not defined in advance. This turns closure into politics. Avoid it by agreeing on severity examples before testing begins and applying them consistently. If a finding does not fit the rubric, update the rubric after the exercise, not during the argument.

- **Evidence Thinness:** Findings are captured as screenshots or summaries without reproduction details. Engineering cannot fix confidently and GRC cannot prove closure. Recover by defining evidence requirements before testing: prompt, context, model version, configuration, retrieval sources, tool calls, output, expected behavior, and actual behavior. A finding that cannot be reproduced cannot become a reliable control.

IMPLEMENTATION CHECKLIST

- [ ] Map eval coverage to real production surfaces, user roles, data sources, tools, and model versions.
- [ ] Define severity categories and examples before running red-team exercises.
- [ ] Build a versioned prompt attack library with expected behavior and severity tags.
- [ ] Write eval cases that test RAG, agent, prompt, output, and policy behavior separately where possible.
- [ ] Configure high-severity eval failures to block release or trigger explicit risk acceptance.
- [ ] Require red-team findings to include reproduction evidence, not only screenshots or summaries.
- [ ] Convert valid red-team findings into regression tests, control changes, or risk acceptance records.
- [ ] Store eval outputs, red-team evidence, closure records, and release decisions as governance evidence.

RELATED READING

- Handbook chapters: Chapter 3, Threat Modeling; Chapter 4, Prompt Injection; Chapter 5, RAG Authorization; Chapter 6, Agentic Permissions; Chapter 14, Governance Evidence and Customer Trust.
- Field Guide: Red Teaming and Adversarial Evaluations; Prompt Injection and Context Security; RAG Security; Agent Security; Incident Response and AI Observability.