workshopps.com

COURSE OUTLINE

AI Security and Observability — Secure, Inspect, and Operate an AI Application

An intensive, lab-driven workshop for developers taking an AI application from a working demo to a controlled, observable deployment

DURATION	FORMAT
5 days	Instructor-led, lab-driven, capstone project

DELIVERY	TRAINER
Live Instructor Led Workshop · Onsite Workshop for your Teams	Muhammed Shakir

Why this course exists

An AI application can return a successful response while exposing private information, following malicious instructions, using the wrong evidence, or consuming an unexpected amount of time and money. A useful production system must control what the application can access and do, and make its behaviour visible enough to investigate and improve.

This workshop gives developers a practical, build-first understanding of AI security and observability: participants progressively secure, instrument, evaluate, and deploy one working AI application, then use recorded evidence to diagnose failures and respond to incidents.

By the end of five days, participants will have implemented access boundaries, tested application-level AI attacks, connected request traces to quality and operational measurements, and completed a controlled deployment and incident exercise. They leave ready to contribute these practices to a project using the stack and patterns practised in the course.

What makes this different

The application is visible from the first session. **This is an engineering room: each idea becomes a small, inspectable part of one working AI application.** You will:

- **Start with a running system** — inspect the supplied application, follow a request through retrieval, generation, and a bounded tool call, and identify where data and permissions cross boundaries.
- **Build protection and visibility together** — reproduce a failure, implement a control, record the result, and verify that the application still serves legitimate requests.

- **Finish with operational evidence** — evaluate a change, investigate an alert, deploy the application, and demonstrate recovery using the same controls, traces, and checks built throughout the course.

The course favours a clear, working production-engineering mental model over boring presentation slides.

Skills You'll Walk Away With

- Explain the user ’ API ’ retrieval ’ model ’ tool ’ response flow, and identify the trust boundaries, sensitive data, and failure modes at each step.
- Instrument an AI request with connected traces, structured logs, and metrics while keeping secrets and sensitive content out of routine telemetry.
- Reproduce direct and indirect prompt injection, test data-exposure paths, and implement layered controls with explicit limitations.
- Enforce authenticated identity, document access boundaries, validated tool calls, and bounded execution in application code.
- Build a repeatable evaluation set covering answer support, security behaviour, legitimate use, response time, and usage cost.
- Create dashboards and actionable alerts, then use request evidence to distinguish retrieval, model, tool, and infrastructure failures.
- Deploy the capstone to a supplied environment, verify its controls, investigate a simulated incident, and demonstrate rollback and recovery.

Who should attend

This workshop is for experienced professionals who want to secure and operate AI applications and contribute practical implementation work to a project.

- **Frontend and backend developers** — integrating AI features with application identity, data access, APIs, and operational controls.
- **AI application developers** — moving from a working retrieval or tool-enabled application to tested boundaries and inspectable behaviour.
- **Platform and reliability engineers** — connecting AI workloads to deployment practices, telemetry, alerts, and incident response.
- **Technical leads and solution architects** — understanding where controls belong and how to assess their effectiveness in a working application.

Prerequisites: comfort with Python, HTTP APIs, Git, basic database queries, and calling an LLM API. Prior security or observability specialization is not required; AI request flow, trust boundaries, traces, and evaluation are introduced in plain engineering language. A working application starter and prepared environment are supplied so participants spend the five days on security and observability rather than rebuilding RAG or provisioning infrastructure from scratch.

Course Outline

Day 1 — Understand the System and Make Its Behaviour Visible

Inspect a working AI application, identify its boundaries, and establish the evidence needed to understand every later control and failure.

Session 1 — Establish the Application and Its Trust Boundaries

Create one shared view of what the application does, which data it handles, and which decisions must remain under application control.

- AI security and observability in one flow: protecting data and actions, recording behaviour, and distinguishing availability from answer correctness
- The supplied application: chat client ’ API ’ retrieval ’ model ’ bounded tool ’ response, with a diagnostic API alongside the user experience
- AI request basics: instructions, user input, retrieved context, generated output, and tool requests
- Trust boundaries: authenticated users, application instructions, external content, model providers, data stores, and connected tools
- A focused threat model: what must be protected, who may access it, what could go wrong, and where a control can be enforced
- *Hands-on:* Run the supplied capstone and establish its security and operating baseline.
  - Start the application and inspect one supported request through chat and the diagnostic API.
  - Review the project-local architecture and map the request flow, data stores, external services, and tool boundary.
  - Classify the supplied synthetic records as public or restricted and identify the sample users and permitted actions.
  - Record a small set of concrete abuse cases and legitimate requests that later sessions will verify.
  - **Session checkpoint:** a runnable application, a shared architecture, and explicit data, identity, and action boundaries.

Session 2 — Trace One Request End to End

Connect the application's steps so a failure can be investigated without guessing or collecting every piece of user content.

- Logs, metrics, and traces: individual events, aggregate measurements, and the connected steps of one request
- OpenTelemetry instrumentation: request identifiers, trace propagation, and spans around retrieval, model calls, and tools
- Useful AI measurements: response time, time to first streamed text, errors, model usage, and estimated cost
- Safe telemetry: allowlisted attributes, redaction, retention, and access to diagnostic information

- *Hands-on:* Instrument the existing request path and inspect it in the supplied telemetry stack.
  - Add connected spans for the API, retrieval, generation, and tool steps using one shared request identifier.
  - Record operational metadata and outcomes without logging credentials, raw restricted documents, or full conversations by default.
  - Connect structured application logs to traces and expose basic request, error, and duration metrics.
  - Run one successful request and one controlled provider failure; locate the responsible step in each trace.
  - **Session checkpoint:** an inspectable request path with useful measurements and a verified telemetry privacy boundary.

Day 2 — Reproduce AI Attacks and Build Layered Defenses

Observe how untrusted content can influence the application, then restrict its impact and verify legitimate behaviour.

Session 3 — Test Direct and Indirect Prompt Injection

Separate what the model is asked to do from what the application is permitted to do.

- Direct prompt injection: a user attempts to override the application's intended task
- Indirect prompt injection: malicious instructions arrive inside a retrieved document or tool result
- Instruction and evidence separation: why retrieved content remains untrusted even when it comes from the application's own index
- Layered defenses: clear context boundaries, constrained outputs, application validation, and restricted capabilities
- *Hands-on:* Attack and strengthen the existing answer path using harmless, prepared examples.
  - Submit a direct instruction override and place an indirect injection in a synthetic source document.
  - Inspect which content reached the model and record the observed application behaviour.
  - Strengthen instruction and evidence separation, constrain expected output, and validate the result before downstream use.
  - Retest the attacks alongside ordinary requests, recording both successful controls and residual limitations.
  - **Session checkpoint:** a reproducible injection exercise and layered defenses whose effectiveness can be inspected rather than assumed.

Session 4 — Protect Sensitive Data and Handle Output Safely

Control where sensitive information may travel and prevent generated text from becoming an unchecked application action.

-

Data exposure paths: prompts, retrieved passages, provider requests, responses, logs, and diagnostic endpoints

- Secrets and sensitive fields: configuration management, minimum necessary context, masking, and controlled disclosure
- Safe output handling: schema validation, browser rendering, and preventing generated text from becoming executable code or a raw database query
- Security events: recording blocked or suspicious behaviour without copying the sensitive payload into telemetry
- *Hands-on:* Close prepared data-exposure and output-handling gaps in the same application.
  - Use synthetic sensitive fields to inspect what is sent to the model and what appears in responses and diagnostics.
  - Remove unnecessary fields from provider-bound context and apply a clear response disclosure policy.
  - Keep secrets outside source code and verify that failure messages and telemetry do not expose them.
  - Validate structured model output and render generated text through the supplied client's safe display path.
  - Record a compact security event for rejected output or blocked disclosure, linked to its request trace.
  - **Session checkpoint:** explicit data-disclosure and output-handling controls with inspectable, privacy-conscious failure records.

Day 3 — Enforce Identity, Retrieval, and Tool Boundaries

Move from model-facing defenses to application-enforced permissions and bounded actions.

Session 5 — Enforce Access Before Retrieval and Generation

Ensure that the application retrieves and uses only evidence the authenticated caller is allowed to access.

- Authentication and authorization: establishing identity versus deciding which data and actions it permits
- Server-side scope: deriving user and organizational permissions from trusted identity rather than request text or chat instructions
- Permission-aware retrieval: applying access constraints before passages enter model context
- Shared-state risks: cache keys, conversation state, and diagnostic responses that can cross user boundaries
- *Hands-on:* Add access control to the existing retrieval and answer path using the supplied identity integration.
  - Validate the supplied identities and resolve their permitted data scope in application code.
  - Apply that scope to semantic and exact retrieval, then check returned records before assembling context.

- Reject caller attempts to widen scope through request fields, prompt text, or document content.
- Partition or disable shared caches and conversation state where identity-aware isolation is required.
- Test two users with different permissions against the same question, including direct API and diagnostic access.
- **Session checkpoint:** enforced identity-to-evidence boundaries with a demonstrated cross-user isolation check.

Session 6 — Bound Tool Calls and Resource Consumption

Allow a useful action while keeping tool selection, permissions, arguments, and execution limits under application control.

- Tool execution flow: the model proposes a call; application code validates and authorizes it before execution
- Least privilege: fixed tool allowlists, narrow credentials, validated arguments, and restricted destinations
- Read versus write actions: explicit approval for a prepared state-changing operation in a sandbox
- Resource controls: request size, tool-call budgets, output limits, deadlines, rate limits, and bounded retries
- *Hands-on:* Secure the capstone's supplied tool adapter and enforce execution budgets.
  - Validate proposed tool names and arguments against a fixed schema and check the authenticated caller's permissions.
  - Block arbitrary destinations, unsupported operations, and attempts to pass raw commands or database predicates.
  - Require an application-enforced approval step for one synthetic write action; verify that a prompt cannot bypass it.
  - Enforce a finite call budget, request deadline, and bounded retry policy with explicit stop reasons.
  - Trigger an unauthorized action and an exhausted budget, then inspect their traces and security events.
  - **Session checkpoint:** a useful, bounded tool path whose permissions and limits remain enforceable independently of model output.

Day 4 — Measure Quality and Turn Telemetry into Operating Decisions

Build repeatable checks, compare a change, and connect measurable failures to actionable alerts.

Session 7 — Evaluate Answer Quality and Security Behaviour

Use a small, repeatable test set to assess whether a change improves the application without weakening its boundaries.

- Evaluation basics: test inputs, expected behaviour, reference evidence, and explicit pass criteria
-

Retrieval and answer checks: expected-source coverage, citation validity, evidence support, and appropriate refusal

- Security regression checks: injection attempts, restricted-data access, unauthorized tools, and legitimate requests that must still succeed
- Deterministic checks and model-assisted scoring: where automation helps, where judgment is uncertain, and when human review is needed
- *Hands-on:* Build a focused evaluation harness around the existing application.
  - Create a small versioned set of answerable, unsupported, restricted, and adversarial requests using the supplied synthetic corpus.
  - Add deterministic checks for permissions, citations, tool execution, and expected refusal; manually review a sample of answer-support judgments.
  - Compare the current configuration with one deliberate prompt or control change on the same cases.
  - Record pass rates, legitimate-request failures, response time, and usage cost without treating one aggregate score as proof of safety.
  - Link failed cases to their request evidence and establish a small release gate for critical failures.
  - **Session checkpoint:** a repeatable evaluation report that explains what passed, what failed, and whether the proposed change should proceed.

Session 8 — Build Dashboards and Investigate Alerts

Turn collected telemetry into a small operating view that helps an engineer decide what to investigate and do next.

- Operational signals: traffic, errors, response time, provider failures, tool failures, and estimated usage cost
- Quality and security signals: sampled evaluation outcomes, unsupported answers, denied access, and rejected tool calls
- Service objectives and alerts: measurable targets, useful thresholds, aggregation windows, and avoiding noisy notifications
- Failure investigation: correlating application logs, request traces, configuration versions, and evaluation results
- *Hands-on:* Create a focused dashboard and diagnose controlled failures.
  - Build dashboard views for request health, retrieval/model/tool duration, usage cost, and selected security outcomes.
  - Define a small set of actionable alerts with an owner, investigation steps, and a clear recovery signal.
  - Inject a slow dependency, a provider error, and an unauthorized tool attempt using supplied failure controls.
  - Follow each signal to the relevant request evidence and distinguish operational failure from a quality or policy outcome.

- Write a concise runbook for one recurring failure using the evidence gathered.
- **Session checkpoint:** an operating dashboard, actionable alerts, and a demonstrated investigation from symptom to responsible application step.

Day 5 — Deploy, Respond, and Demonstrate Readiness

Put the same application into a controlled environment, verify its boundaries, and practise an evidence-led response to failure.

Session 9 — Deploy with Verified Controls

Move the capstone into the supplied deployment environment without losing its security boundaries or diagnostic evidence.

- Deployment basics: application image, environment configuration, managed secrets, service identity, and private dependencies
- Release checks: dependency scanning, critical security and quality evaluations, and explicit handling of failed checks
- Runtime configuration: authenticated entry points, restricted diagnostics, telemetry export, health checks, and resource limits
- Safe releases: versioned application and prompt configuration, smoke checks, and a known rollback target
- *Hands-on:* Deploy the capstone using the supplied deployment template and environment.
  - Package the application and configure secrets and service access through the supplied platform mechanisms.
  - Run dependency scanning and the focused evaluation gate; inspect failures before proceeding.
  - Deploy with authenticated access, bounded resources, and the established telemetry configuration.
  - Verify a legitimate request, a denied request, a tool boundary, and an end-to-end trace in the deployed environment.
  - Record the deployed version and demonstrate rollback to the previous known-good version.
  - **Session checkpoint:** a controlled deployment with verified access, request evidence, release checks, and a working rollback path.

Session 10 — Respond to an Incident and Demonstrate the Application

Use the controls and evidence built throughout the course to investigate, contain, recover, and explain the result.

- Incident response flow: detect, assess scope, contain, recover, and verify
- Evidence handling: retaining useful request metadata while restricting access to sensitive records
- Containment choices: disable a tool, restrict a data source, revoke a compromised credential, or roll back a release

- Recovery and handover: rerun affected checks, confirm service behaviour, document limitations, and identify follow-up work
- *Hands-on:* Complete a guided incident exercise and a final capstone demonstration.
  - Investigate one prepared incident using an alert, related traces, security events, and the deployed configuration.
  - Identify the affected boundary and distinguish observed facts from assumptions about impact.
  - Apply a suitable containment action in the training environment and restore the known-good operating path.
  - Rerun the affected security and legitimate-use checks, then confirm the dashboard's recovery signals.
  - Demonstrate the application from authenticated request through permitted retrieval, grounded answer, bounded tool use, and recorded evidence.
  - Hand over the architecture, evaluation results, dashboard, deployment instructions, runbook, and remaining limitations.
  - **Session checkpoint:** a working, explainable deployment and an evidence-backed demonstration of prevention, detection, investigation, and recovery.

Capstone project

Participants incrementally secure and operate a **document-grounded, tool-enabled AI application** using a supplied working starter. The capstone is deliberately focused: it teaches application-level AI security and observability across one coherent request path, with synthetic data and sandboxed actions suitable for classroom exercises.

The application accepts a question, retrieves permitted source passages, generates a source-backed answer, and can propose a narrowly defined tool action. The supplied case-study pack provides the documents, sample identities, access rules, tool operation, and prepared questions. These inputs can come from any domain; the security boundaries, telemetry, evaluation approach, and deployment sequence remain the same.

The project has two complementary interfaces throughout the course:

- **The diagnostic API** is the engineering learning interface. It exposes permitted retrieval details, control outcomes, request identifiers, and evaluation results so participants can inspect the pipeline without exposing secrets or unrestricted source content.
- **The chat client** is the end-user interface. It uses the same authenticated application path, so participants experience the answers, denials, approval requests, and failure messages produced by the controls they implement.

The application builds in one direction. The working starter establishes retrieval, generation, the client connection, and a sandboxed tool adapter. Instrumentation makes that flow visible. Injection defenses and data controls strengthen its content boundaries. Authorization limits evidence and actions. Execution budgets bound tool use and consumption. Evaluations and dashboards make behaviour measurable.

Deployment and incident exercises bring the same controls and evidence into an operating environment; they do not introduce replacement applications or require rebuilding the user interface.

In scope: application threat modelling, privacy-conscious tracing and logging, operational metrics, direct and indirect prompt injection, sensitive-data protection, safe output handling, authenticated access, permission-aware retrieval, bounded tool execution, resource limits, focused quality and security evaluations, dashboards, alerts, a controlled deployment, rollback, and a guided incident response.

Out of scope for this course: training or fine-tuning models, model-weight attacks, comprehensive training-data poisoning investigations, open-ended autonomous agents, broad penetration testing, enterprise identity-platform construction, multi-cloud infrastructure provisioning, formal compliance certification, and a complete organization-wide security or reliability practice. These require separate specialist work beyond the five-day application-engineering scope.

The required outcome is a deployed, inspectable classroom application: participants can serve a permitted request, block a prepared unauthorized action, trace the result, evaluate a change, investigate an alert, and demonstrate recovery. This provides a practical basis for immediate project contribution; production adoption still depends on the project's own access rules, infrastructure, review, and operating requirements.

A note on delivery

This five-day course is intentionally sequenced from application basics to advanced application-level controls and operating exercises. Participants begin with a supplied working application so the training can concentrate on security and observability rather than spending the first days building document ingestion, a chat frontend, or cloud infrastructure.

Each full day balances instructor-led explanation, demonstration, guided labs, review, and troubleshooting, with two tea breaks and one lunch break. The prepared starter, synthetic case-study pack, identity integration, telemetry stack, and deployment environment keep the work centred on the concepts being taught. Environment readiness is checked before the workshop so setup does not consume the lab schedule.

The sequence is deliberate: first understand the request and its boundaries; then record its behaviour; reproduce content attacks and close data-exposure paths; enforce identity and tool permissions; measure quality and operational outcomes; and finally deploy, investigate, and recover. Each lab introduces the concept, demonstrates a concrete failure, implements the control, and verifies both the protected boundary and legitimate behaviour. Participants leave having performed the work and explained the evidence behind it.

Ready to secure and operate AI applications?

Enquire about a private cohort or a tailored delivery for your developer team.