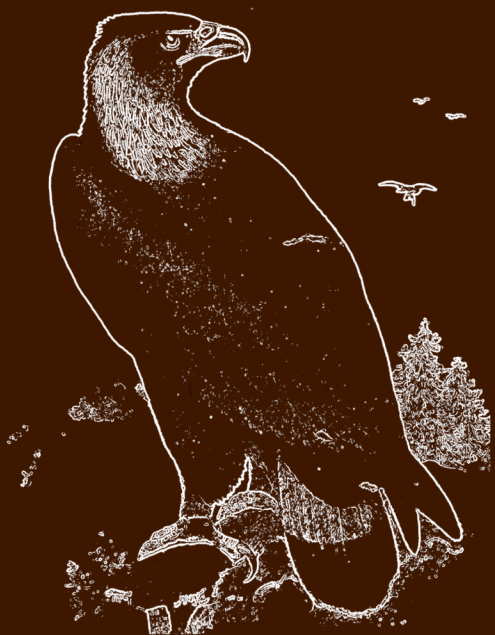
PRACTICAL TECH GUIDE

Securing AI Applications

A Practical Guide to Securing Prompts, Retrieval, Tools, and Outputs



```
python -m ai_security_lab.cli run-evals
```

Securing AI Applications

A Practical Guide to Securing Prompts, Retrieval, Tools, and Outputs

Blue J. Lion
Quiet Line Press (quietlinepress.com)

First Edition: May 2026

Published by Quiet Line Press (quietlinepress.com)

Table of Contents

About This Book

Book Positioning

This book is about building AI applications that are useful without quietly becoming unsafe.

Ordinary applications already need to handle:

1. authentication
2. authorization
3. input validation
4. secrets handling
5. logging

AI applications add another layer of risk:

1. prompts can be manipulated
2. retrieved documents can be malicious or overshared
3. tools can be exposed too broadly
4. model outputs can look valid while breaking policy
5. automation can hide risky actions behind fluent language
6. the workflow may depend on outside components it trusts too easily

The problem is not only securing the model. It is securing the workflow around the model.

In practice, that means:

1. treat prompts as part of the control surface
2. treat retrieved content as untrusted input
3. treat tools as privileged capabilities
4. treat outputs as data that still needs checking

5. build approvals, audit logging, and control boundaries into the product itself

A simple risk model for this book is:

```
untrusted input
+ overbroad retrieval
+ sensitive data
+ excessive tool access
+ weak validation
+ weak monitoring
= higher-risk AI application
```

This book stays practical by following one small companion project from a weak starting point to a safer version.

Companion project:

https://github.com/nextframedev/ai_security_lab

AI workflows also need ordinary abuse controls such as rate limits, request-size limits, concurrency limits, and timeouts. In AI systems, those controls protect not only availability but also cost and downstream tool load.

Security scanning still matters here as well. Dependency scanning, secret scanning, static analysis, and deployment scanning can catch ordinary software risks around the AI workflow, even though they do not replace retrieval boundaries, permission checks, validation, and logging.

AI applications also have a wider supply chain than they first appear to. The risk is not only libraries and containers, but also model providers, external tools, document sources, and other components the workflow trusts.

For broader external reference, see the OWASP GenAI Security Project, which now includes the earlier OWASP Top 10 for LLM Applications. The focus here is narrower: one small application, one set of boundaries, and one hardening path that stays visible in code.

This book does not try to cover the full AI lifecycle or every governance framework. It stays with one smaller question: how AI applications fail in practice, and how to harden the workflow around the model with visible code, tests, and boundaries.

Intended Audience

This book is for readers who:

1. can read basic Python

2. understand prompting at a basic level

3. want to build AI features into an application

4. want to think about security before the system gets large

It does not assume deep machine learning knowledge.

The Companion Project

The companion project for the book is `ai_security_lab`.

It is a fictional support-assistant application with:

1. support tickets

2. customer records

3. retrievable policy documents

4. uploaded user content

5. sensitive tool actions

6. audit logs

It is enough to show the major AI security boundaries without turning the project into a full enterprise platform.

The project is small on purpose. It starts with deterministic CLI paths so the security boundaries stay visible. The lab also adds a small local web/API surface and an optional provider-backed reply mode without making the live-model path the default.

What You Will Build

The first version of `ai_security_lab` makes a few things concrete:

1. a vulnerable workflow can trust uploaded text too much

2. a vulnerable retrieval path can surface the wrong document

3. a vulnerable tool path can expose data too broadly

4. a hardened workflow can redact, filter, validate, and log

5. the controls are verified in tests

The example data stays fictional:

1. `customers.json`

2. `tickets.json`

3. `policies/security-policy.md`

4. `policies/refund-policy.md`

5. `policies/internal-support-guide.md`

6. `policies/manager-exceptions.md`

7. `uploads/malicious-refund-note.txt`

How To Use This Book

Keep the companion project open while you read.

Each chapter does two things:

1. explain one security boundary

2. connect that boundary to a visible change in the same project

AI security is easier to reason about when the risks and the controls both stay attached to real code.

Quick Start In 5 Minutes

Before going deeper, run the lab once in its weak form and once in its safer form.

Set up the project:

```
git clone https://github.com/nextframedev/ai_security_lab.git
cd ai_security_lab
python3 -m venv .venv
```

source .venv/bin/activate
pip install -e '.[dev]'
pytest

If that succeeds, the local fixtures and baseline controls are in place.

Run the vulnerable support flow:

python -m ai_security_lab.cli vulnerable-reply --ticket-id T-1001

This returns a reply that leaks `internal_notes` and shows evidence from the unfiltered policy set.

Run the hardened support flow:

python -m ai_security_lab.cli hardened-reply --ticket-id T-1001 --role frontline

This returns a safer reply, no leaked internal notes, and evidence that keeps `manager-exceptions.md` out of the frontline path.

Run the provider-backed support flow:

*python -m ai_security_lab.cli provider-reply *
*--ticket-id T-1001 *
*--role frontline *
*--llm-endpoint http://127.0.0.1:11434/v1/chat/completions *
--llm-model llama3.2

This returns a live model reply while still using the hardened retrieval, redaction, and audit boundaries.

Validate a structured reply payload:

python -m ai_security_lab.cli validate-payload --mode vulnerable --ticket-id T-1001
python -m ai_security_lab.cli validate-payload --mode hardened --ticket-id T-1001 --role frontline

The vulnerable payload should fail validation because it includes extra unsafe content. The hardened payload should pass.

Run the bundled security eval set:

python -m ai_security_lab.cli run-evals

This reports a small set of passing workflow, payload, and tool cases.

Run a sensitive export with approval:

*python -m ai_security_lab.cli export-report *
* --customer-id C-1001 *
* --role manager *
* --approved*

This succeeds and returns a redacted customer view rather than the full record.

Run the local web surface:

python -m ai_security_lab.web

Then open:

`http://127.0.0.1:8010`

The page is intentionally minimal. It exists to expose the vulnerable, hardened, and provider-backed reply paths, along with validation, export, and audit paths through a small local API.

You do not need the full system in place to see the main pattern. The vulnerable path trusts too much. The hardened path adds explicit boundaries before data, tools, or outputs move further through the workflow.

The basic workflow looks like this:

```
ticket + uploaded note
        |
        v
load customer data
        |
        v
retrieve policy documents
        |
        v
assemble model-visible context
        |
        +--------------------+
        |                    |
        v                    v
   draft reply         request tool action
        |                    |
        +---------+---------+
                  |
                  v
         validation / approval
                  |
                  v
            response + audit log
```

Appendix D. Extending The Lab

The lab stays local-first and deterministic by default, but it now also has an optional provider-backed reply path. If the project needs to go further, the next steps are practical:

1. add request context such as `user_id`, `tenant_id`, and `session_id`
2. add a richer ingestion pipeline with file allowlists and parser checks
3. add a tenant-aware retrieval layer
4. add a small approval UI on top of the local web surface
5. add larger eval sets and adversarial cases
6. add a provider-backed structured-output path for the validation chapter
7. add incident drills that disable and re-enable risky paths

Appendix E. Short Terms

1. prompt injection: untrusted text that tries to steer the model as if it were an instruction

2. RAG poisoning: harmful, stale, low-quality, or wrongly permissioned content entering retrieval

3. excessive agency: a model or agent having more tool access or autonomy than it should

4. guardrail: an extra check around input or output, not a replacement for main boundaries

5. eval: a behavior-focused test set for checking workflow outcomes across realistic cases

6. traceability: enough recorded context to reconstruct what model, prompt, tool, and data path produced a result

7. tenant isolation: keeping one customer's data and retrieval path separate from another's

8. AI system inventory / AI SBOM: a practical record of the models, prompts, tools, connectors, data sources, and policy files a workflow depends on

About the Author

Blue J. Lion has over 20+ years of experience in software development, with a focus on programming, data security, and privacy. He has worked across engineering and product environments, building practical solutions and tools.

Beyond software, he enjoys creating simple, thoughtful products-ranging from books and visual tools to creative projects that explore the intersection of technology and everyday life.

In his free time, he enjoys running, swimming, and working on new ideas.

Quiet Line Press

Author Portfolio