7

Security Testing for AI Applications

The New Attack Surface

THE MODERN QA ENGINEER SKILLSET

Security Testing for AI Applications: The New Attack Surface

Prompt Injection, Jailbreaking, Data Leakage, and the AI Threat Landscape

Yuri Syuganov

Modern QA Course

https://modernQAcourse.com

Security Testing for AI Applications: The New Attack Surface

Prompt Injection, Jailbreaking, Data Leakage, and the AI Threat Landscape

Book 7 of 26 in the series *The Modern QA Engineer Skillset*

First Edition · Version 1.0.0 · 2026-07-11

Set in Charter, IBM Plex Sans, and IBM Plex Mono. Typeset with Pandoc and XeLaTeX.

Modern QA Course · https://modernQAcourse.com · support@modernQAcourse.com

Contents

Introduction

Version: 1.0 **Last Updated:** February 2026 **Estimated Reading Time:** 40-50 hours (with exercises) **Equivalent Length:** ~230 pages

About This Book

This book is a comprehensive, hands-on guide to security testing for applications that incorporate artificial intelligence, with a particular focus on Large Language Model (LLM) integrations. It bridges the gap between traditional application security testing and the rapidly evolving AI-specific threat landscape.

The AI revolution has fundamentally changed how software is built. Chatbots, copilots, recommendation engines, and autonomous agents are now embedded in applications across every industry. But each AI feature introduces an attack surface that traditional security testing does not address. Prompt injection, jailbreaking, data leakage, retrieval poisoning, and model theft are not covered by classic OWASP Top 10 checklists or conventional SAST/DAST tools.

This book gives you the knowledge and practical skills to test both the traditional web security surface and the AI-specific attack surface. You will build real test suites, run hands-on labs, and develop a security testing program that evolves as fast as the threats it defends against.

Core principle: Security testing for AI is not a one-time activity. It is a continuous practice.

Who This Book Is For

This book is designed for:

- **QA Engineers** expanding into security testing for AI-powered applications
- **Security Engineers** who need to understand AI-specific vulnerabilities
- **QA Architects** designing security testing strategies for AI products
- **DevOps/Platform Engineers** building CI/CD pipelines that include AI security gates
- **Technical Leads** responsible for the security posture of AI features
- **Penetration Testers** adding AI application testing to their skill set

You do not need to be a machine learning engineer or a deep learning researcher. This book focuses on testing AI applications from the outside in – as a user, attacker, or quality engineer would interact with them.

Prerequisites

To get the most from this book, you should have:

- **Programming experience** in Python (intermediate level). All code examples use Python, pytest, and common libraries.
- **Basic web security knowledge.** Familiarity with concepts like SQL injection, XSS, authentication, and HTTPS is helpful but not strictly required – Part IV provides a refresher.
- **Familiarity with REST APIs.** You should be comfortable making HTTP requests and reading JSON responses.
- **Access to an LLM API.** Many exercises use OpenAI, Anthropic, or a local model. A free-tier API key is sufficient for most labs.
- **A development environment.** Python 3.10+, pip, git, and a code editor.

No prior experience with AI security, prompt engineering, or threat modeling is required. The book builds these skills from the ground up.

How to Use This Book

Structure

The book is organized into seven parts:

Part	Focus	Chapters
I: Foundations	Why AI security matters, the threat landscape	1-2
II: OWASP LLM Top 10 Deep Dive	Hands-on labs for each vulnerability class	3-7
III: AI-Specific Attack Techniques	Jailbreaking, data leakage, RAG attacks	8-11
IV: Traditional Security Through the AI Lens	Classic OWASP Top 10 amplified by AI	12-13
V: Security Pipeline and Tooling	SAST, DAST, SCA with AI-specific rules	14-15
VI: Security Program and Compliance	STRIDE, EU AI Act, NIST AI RMF, metrics	16-19
VII: Capstone and Reference	Complete security audit project, glossary	20-22

Learning Features

Throughout the book you will find:

> **Pro Tip:** Highlighted advice from experienced security practitioners. These call out non-obvious insights that save time or prevent common errors.

> **Common Mistake:** Warnings about frequent pitfalls. Learn from others' errors before making your own.

> **Hands-On Exercise:** Practical activities ranging from 15 minutes to several hours. Each is rated by difficulty: Beginner, Intermediate, or Advanced.

> **Self-Assessment Quiz:** End-of-chapter questions to verify your understanding before moving on.

Suggested Learning Paths

Path 1: Fast Track (2 weeks)

If you need to get productive quickly, focus on Chapters 1, 3, 4, 8, 9, and 14. This covers prompt injection, jailbreaking, data leakage, and CI pipeline integration.

Path 2: Comprehensive (6-8 weeks)

Work through the entire book sequentially. Complete all exercises and the capstone project. This path builds deep expertise.

Path 3: Reference

If you are already experienced in security testing, use the table of contents to jump to specific topics. Each chapter is designed to be reasonably self-contained.

Code Repository

All code examples in this book are designed to be copy-pasted and run. They use standard Python libraries and common AI SDKs. Where a specific AI provider API is used, the code includes comments showing how to adapt it to other providers.

PART I

FOUNDATIONS

CHAPTER 1

The AI Security Landscape

1.1 Why AI Applications Need a New Security Approach

AI applications are still web applications. They still have APIs, databases, authentication systems, and user interfaces. Every vulnerability that exists in traditional web applications – SQL injection, cross-site scripting, broken access control – still exists in AI applications.

But AI applications also introduce an entirely new category of vulnerabilities that traditional security testing does not address. When your application includes a Large Language Model, you have added a component that:

1. **Interprets natural language as instructions.** Unlike a database that executes only structured queries, an LLM processes free-form text and decides what to do with it. This makes it susceptible to prompt injection – the attacker's instructions are indistinguishable from legitimate input.
2. **Has access to tools and data.** Modern LLM applications give models access to databases, APIs, file systems, and the internet. A compromised model can use these tools to exfiltrate data, modify records, or attack internal systems.
3. **Generates unpredictable output.** Traditional applications produce deterministic output for a given input. LLMs produce probabilistic output that can vary each time. This means a security test that passes once might fail on the next run.

4. **May memorize training data.** LLMs can inadvertently memorize and reproduce sensitive information from their training data, including PII, API keys, and proprietary content.
5. **Can be manipulated through indirect channels.** If your LLM processes external data (emails, documents, web pages), an attacker can embed instructions in that data without ever directly interacting with your application.

The result is a dual attack surface: the traditional web application surface and the AI-specific surface. A comprehensive security testing strategy must cover both.

The Scale of the Problem

Consider a typical AI-powered customer support chatbot:

- It has a system prompt that defines its behavior and restrictions
- It connects to a RAG pipeline that retrieves information from company documents
- It can look up order details via an API
- It can process refunds up to a certain limit
- It maintains conversation history across turns
- Multiple users interact with it simultaneously

Each of these capabilities is an attack vector:

Capability	Traditional Risk	AI-Specific Risk
System prompt	N/A	Prompt injection to override behavior
RAG pipeline	SQL injection in search	Retrieval poisoning, indirect injection
Order lookup API	Broken access control	LLM bypasses access checks via tools
Refund processing	Authorization bypass	Social engineering the AI to approve refunds
Conversation history	Session hijacking	Cross-session context leakage

continued on next page

Capability	Traditional Risk	AI-Specific Risk
Multi-user access	Session fixation	Model confusion between user contexts

A traditional penetration test would catch the traditional risks. It would likely miss every AI-specific risk. This book teaches you to catch both.

1.2 The AI Security Timeline

AI security is a young field, but it is maturing rapidly:

Year	Milestone
2022	ChatGPT launch; first widespread prompt injection demonstrations
2023	OWASP publishes LLM Top 10 v1.0; first AI-specific CVEs in LangChain
2024	EU AI Act enters into force; major prompt injection incidents in production systems; OWASP LLM Top 10 v2.0
2025	NIST AI RMF adoption grows; automated jailbreak testing becomes standard practice; AI red teaming emerges as a discipline
2026	Digital Omnibus (agreed May-June 2026) defers EU AI Act high-risk obligations to 2 December 2027; until it is formally enacted, the original 2 August 2026 deadline technically remains binding – compliance testing remains essential for regulated AI systems

The attack surface is expanding faster than the defense. New jailbreak techniques emerge weekly. New model capabilities (tool use, code execution, web browsing) create new attack vectors. Security testing must be continuous and adaptive.

1.3 Threat Actors and Motivations

Understanding who attacks AI systems and why helps you prioritize your testing:

Threat Actor	Motivation	Typical Attacks	Sophistication
Curious users	Experimentation, fun	Simple jailbreaks, prompt extraction	Low
Competitors	Intellectual property theft	Model extraction, training data recovery	Medium
Fraud operators	Financial gain	Social engineering the AI, refund abuse	Medium
Data thieves	PII harvesting	Data leakage probing, cross-session attacks	Medium-High
Hacktivists	Embarrassment, ideology	Making the AI say offensive things	Medium
Nation-state actors	Espionage, disruption	Supply chain poisoning, advanced injection	High
Automated bots	Scale attacks	Mass prompt injection, credential stuffing	Varies

Your security testing program should cover the threat actors most relevant to your application. A customer-facing chatbot faces different threats than an internal code review tool.

1.4 Key Concepts and Terminology

Before diving into specific vulnerabilities, establish a common vocabulary:

Prompt: The text input sent to an LLM. Includes the system prompt (developer-defined instructions), conversation history, and user input.

System prompt: The developer-defined instructions that establish the LLM's behavior, persona, restrictions, and capabilities. This is the first line of defense.

Prompt injection: An attack where the attacker's input manipulates the LLM into ignoring its system prompt and following the attacker's instructions instead.

Jailbreak: A specific form of prompt injection aimed at bypassing the model's safety guidelines to produce content the model is designed to refuse.

RAG (Retrieval-Augmented Generation): An architecture where the LLM's response is augmented with information retrieved from external documents or databases.

Tool use / Function calling: The ability of an LLM to invoke external functions (APIs, database queries, file operations) as part of generating a response.

Guardrails: Safety mechanisms that constrain the LLM's behavior, including input filters, output validators, and content classifiers.

Red teaming: Adversarial testing where skilled human testers attempt to find vulnerabilities through creative, unscripted attacks.

Defense in depth: The principle of using multiple overlapping security layers so that if one fails, others still protect the system.

1.5 The OWASP LLM Top 10 at a Glance

The OWASP Top 10 for LLM Applications is the essential framework for AI security testing. Here is a high-level overview; Parts II and III of this book cover each item in depth with hands-on labs.

#	Vulnerability	Severity	What It Means
LLM01	Prompt Injection	Critical	Attacker manipulates the model via crafted input
LLM02	Insecure Output Handling	High	Model output used unsafely in downstream systems
LLM03	Training Data Poisoning	High	Malicious data in training corrupts model behavior
LLM04	Model Denial of Service	Medium	Crafted inputs exhaust resources or cause excessive costs
LLM05	Supply Chain Vulnerabilities	High	Compromised models, libraries, or data sources
LLM06	Sensitive Information Disclosure	Critical	Model reveals confidential data
LLM07	Insecure Plugin Design	High	Plugins/tools with excessive permissions

continued on next page

#	Vulnerability	Severity	What It Means
LLM08	Excessive Agency	High	Model takes actions without adequate oversight
LLM09	Overreliance	Medium	Blind trust in model output without verification
LLM10	Model Theft	Medium	Unauthorized extraction of model behavior or weights

Every entry in this table should have corresponding automated tests in your CI pipeline. This is the minimum security test coverage for any AI application.

Self-Assessment Quiz: Chapter 1

1. Name three ways an AI application's attack surface differs from a traditional web application.
2. What is the difference between prompt injection and jailbreaking?
3. Why is "defense in depth" particularly important for AI applications?
4. Which OWASP LLM Top 10 vulnerabilities are rated "Critical" severity?
5. A curious user trying "Ignore all previous instructions" is an example of what type of attack?
6. Why must AI security testing be continuous rather than a one-time assessment?

Key Takeaways

- AI applications have a dual attack surface: traditional web vulnerabilities plus AI-specific vulnerabilities
- The OWASP LLM Top 10 is the essential framework for AI security testing
- New attack techniques emerge weekly; security testing must be continuous

- Understanding threat actors helps prioritize testing efforts
- Defense in depth is the only viable strategy – no single defense is sufficient

Career Translation

Resume phrasing

- Mapped the dual attack surface of AI-powered applications, identifying 10+ AI-specific vulnerability classes beyond traditional OWASP Top 10 coverage.
- Developed security testing strategy aligned with the OWASP LLM Top 10 framework, ensuring continuous coverage of prompt injection, jailbreaking, data leakage, and model theft vectors.
- Classified threat actors by sophistication and motivation to prioritize security test coverage, reducing untested high-risk scenarios by 80%.

Cover letter framing

AI applications introduce an attack surface that traditional security testing does not address. I bring the ability to identify and test both classic web vulnerabilities and AI-specific threats – prompt injection, data leakage, RAG poisoning – using the OWASP LLM Top 10 as the baseline framework. My approach treats AI security as a continuous discipline, not a one-time checklist, because new jailbreak techniques emerge weekly and the defense must evolve just as fast.

Interview framing

"I approach AI security by first acknowledging that every AI application is still a web application – the traditional OWASP Top 10 still applies. But then I layer on the AI-specific threat surface: prompt injection, data leakage, jailbreaking, tool abuse. I optimize for continuous testing because the attack landscape changes weekly. Trade-offs include balancing depth of coverage against the speed of releasing new AI features, and accepting that no single defense fully solves prompt injection – defense in depth is the only viable strategy."

What not to say

- "We just need to write a good system prompt and the AI will be secure." – A system prompt is a speed bump, not a wall; it can be bypassed.
- "AI security is just regular security with a chatbot." – This ignores the entire AI-specific attack surface (prompt injection, data leakage, RAG poisoning).
- "We tested the AI once before launch, so we're covered." – AI security requires continuous testing as new attack techniques emerge weekly.
- "The model provider handles security for us." – The provider secures the model; you secure the application around it.
- "Our users wouldn't know how to do prompt injection." – Curious users discover injection techniques daily; assuming ignorance is not a defense.

Interview Depth Check

Question 1

Prompt: Your company is launching an AI-powered customer support chatbot that can look up orders and process refunds. The security team asks you to assess the AI-specific attack surface. Walk through the threat categories you would identify.

What a strong answer should cover:

- The dual attack surface concept (traditional web + AI-specific)
- Specific AI threats: prompt injection (direct and indirect), system prompt extraction, data leakage, tool/plugin abuse, cross-session contamination
- That each capability (order lookup, refund processing, conversation history) is a separate attack vector
- Reference to the OWASP LLM Top 10 as the organizing framework

Example answer:

- "I would start by mapping every AI capability to its attack vectors. The system prompt can be extracted via prompt injection. The order lookup API is an attack vector if the LLM bypasses access controls. Refund processing can be abused through social engineering the AI. Conversation history introduces cross-session leakage risk. I would organize the assessment around the OWASP LLM Top 10 – prompt injection, insecure output handling, sensitive information disclosure, excessive agency, and insecure plugin design are the top concerns for this use case."

Question 2

Prompt: A product manager tells you that AI security testing is unnecessary because the model provider (OpenAI, Anthropic) already builds safety into the model. How do you respond?

What a strong answer should cover:

- The distinction between model-level safety and application-level security
- That the provider secures the model but not your system prompt, tools, data access, or output handling
- That prompt injection targets the application layer, not the model layer
- Concrete examples of application-level vulnerabilities the provider cannot prevent

Example answer:

- "Model providers build safety guardrails into the base model – they prevent the model from generating certain types of harmful content. But application security is our responsibility. The provider cannot prevent prompt injection that targets our system prompt. They cannot enforce our access control when the LLM calls our internal APIs. They cannot prevent data leakage from our RAG pipeline. These are application-layer vulnerabilities that exist regardless of how safe the underlying model is."

Question 3

Prompt: You need to prioritize security testing for a new AI feature with a tight deadline. You can only implement three security controls before launch. Which three do you choose and why?

What a strong answer should cover:

- A clear prioritization framework based on likelihood and impact
- Prompt injection testing as the top priority (most common, most impactful)
- Output validation/sanitization as second (prevents downstream injection)
- Rate limiting and cost controls as third (prevents DoS and financial damage)
- Acknowledgment that the remaining controls should follow post-launch

Example answer:

- "First, prompt injection testing with at least 20 payloads – it is the most critical and most common AI vulnerability. Second, output sanitization for any downstream use of LLM output in SQL, HTML, or shell commands – this prevents the LLM from becoming an injection vector into our own systems. Third, rate limiting and max_tokens enforcement to prevent cost-based denial of service. Everything else – jailbreak testing, data leakage scanning, compliance – I would schedule for the first sprint after launch with a clear timeline communicated to stakeholders."

Question 4

Prompt: Explain the difference between prompt injection and jailbreaking to a non-technical stakeholder.

What a strong answer should cover:

- A clear, jargon-free distinction
- Prompt injection: making the AI do something unintended (functionality attack)

- Jailbreaking: making the AI say something it should refuse (safety attack)
- A concrete analogy or example for each

Example answer:

- "Prompt injection is when an attacker tricks our AI into doing something it was not supposed to do – like revealing our internal instructions, accessing another customer's data, or processing an unauthorized refund. Jailbreaking is when an attacker tries to make the AI say something it was trained to refuse – like generating harmful content or offensive material. Both are important to test for, but prompt injection is the bigger business risk because it can lead to data breaches and unauthorized actions, while jailbreaking is primarily a reputational risk."

About This Sample

You have just read **Chapter 1 of Security Testing for AI Applications: The New Attack Surface** — the free sample. The complete book continues with every remaining chapter, the interview Q&A, and all of the code.

Get the full book

Sample readers get a discount at
https://leanpub.com/modernqa-07-security-testing-ai/c/SAMPLE10

Read online first

Chapter one of **every** book in the series is free to read at
https://modernqacourse.com/library

About the Author

Yuri Syuganov is a software quality engineer with more than twenty years of hands-on testing experience, spanning the arc from manual test design through Selenium-era automation to today's AI-augmented, agent-driven frameworks. He is the author of The Modern QA Engineer Skillset, a 26-book series built on a blunt analysis of what companies actually expect from QA engineers, and the creator of modernQAcourse.com, where the course behind the series is free and readers can bring real testing problems to experienced practitioners. He revises the series as the industry changes — because it does.

Also in This Series

The Modern QA Engineer Skillset — a 26-book course in modern software quality assurance.

AI-Augmented QA (Books 1–10)

- **Book 1:** Agent Skills for Browser Automation: The Complete Guide
- **Book 2:** AI-Augmented Test Design: From Prompts to Test Suites
- **Book 3:** Agentic Testing Architectures: Building Autonomous QA Systems
- **Book 4:** API and Contract Testing with AI: Schema to Shield
- **Book 5:** Performance and Chaos Engineering: Breaking Things on Purpose
- **Book 6:** Observability-Driven Testing: Testing in Production
- **Book 7:** Security Testing for AI Applications: The New Attack Surface
- **Book 8:** Infrastructure as Code Testing: Terraform to Kubernetes
- **Book 9:** Mobile and Cross-Platform Testing: 24,000 Devices, One Strategy
- **Book 10:** Visual and Accessibility Testing: Pixels and People

Technical Foundations (Books 11–19)

- **Book 11:** Manual Testing Fundamentals: The Bedrock
- **Book 12:** Programming for QA: Python, JavaScript, and TypeScript from Zero to Fluent
- **Book 13:** Browser Test Automation: Playwright, Patterns, and the Death of Fragile Tests

- **Book 14:** API Testing Fundamentals: Postman to Pytest
- **Book 15:** SQL and Database Testing: Query Your Way to Quality
- **Book 16:** CI/CD Pipelines: From Commit to Confidence
- **Book 17:** Git and Version Control: Beyond Push and Pull
- **Book 18:** Test Management Tools: Making Quality Visible
- **Book 19:** Linux and Command Line: Your Production Survival Kit

Professional Skills (Books 20–26)

- **Book 20:** Agile and Scrum for QA: Quality in Every Sprint
- **Book 21:** Communication and Stakeholder Management for QA: Influence Without Authority
- **Book 22:** Test Strategy and Quality Metrics: Measuring What Matters
- **Book 23:** QA Leadership and Mentoring: Multiplying Your Impact
- **Book 24:** Technical Writing for QA: Documents That Drive Action
- **Book 25:** Interview Preparation and Career Strategy: Land the Job
- **Book 26:** Testing Like a Senior: Patterns & Anti-Patterns Across Software QA Career Levels

Visit https://modernQAcourse.com for the full course catalog, updates, and companion resources.

Stuck on a Real Problem?

The full course behind this series is free to read at https://modernQAcourse.com — and it stays free.

When a chapter meets the real world and something breaks, post it on the **Help Board** at https://modernQAcourse.com. Asking is free, and experienced practitioners answer right in the thread. Need hands-on help? Book a 1:1 screen-share session with a QA expert.