

warning "Backend update (October 2026)" GitHub Models inference was retired on July 30, 2026. Any references below are retained only for historical context while this lab is migrated. Use Lab 013 ? Free LLM backends to configure Foundry Local or Ollama before continuing.

Lab 036: Prompt Injection Defense & Agent Security

Level: L300 Path: Pro Code Time: ~40 min ? Cost: GitHub Free

What You'll Learn

What prompt injection is and why it's dangerous in agentic systems The difference between direct and indirect injection attacks Practical defenses: input sanitization, output validation, privilege separation Tool call guardrails ? preventing agents from taking destructive actions Security checklist for production agents

Introduction

A prompt injection attack manipulates an AI agent by embedding malicious instructions in content the agent processes. Unlike SQL injection (inject code into a query), prompt injection injects instructions into the LLM's context.

Why agents are especially vulnerable: They take actions (send emails, modify files, call APIs). A manipulated agent doesn't just produce bad text ? it does bad things.

Prerequisites

Python 3.11+ pip install openai pydantic GITHUB_TOKEN set

Lab Exercise

Step 1: Direct injection ? understand the attack

[diagram or code sample]

Step 2: Indirect injection ? the harder problem

Indirect injection occurs when malicious instructions are embedded in data the agent reads (web pages, documents, emails, database results).

[diagram or code sample]

Step 3: Defense 1 ? Input sanitization

[diagram or code sample]

Step 4: Defense 2 ? Privilege separation

The most effective defense: agents should only have access to what they need.

[diagram or code sample]

Step 5: Defense 3 ? Output validation

[diagram or code sample]

Agent Security Checklist

? | Defense | Implementation | ? | Input sanitization | Scan for injection patterns, strip HTML comments | ? | Privilege separation | Agents have only needed permissions | ? | Tool call allowlisting | Validate every tool call against role | ? | Structured output | Validate agent output shape before acting | ? | Human-in-the-loop | Require confirmation for irreversible actions | ? | Audit logging | Log all tool calls with user, timestamp, arguments | ? | Rate limiting | Prevent agents from taking too many actions too fast | ? | Separation of data and instructions | Use different prompt sections, mark user content | ? | Content filtering | Use Azure Content Safety / Responsible AI APIs | ? | Least-privilege secrets | Agent API keys have minimum necessary permissions |

Marking User Content Clearly

A simple but effective technique: make the LLM distinguish user content from instructions.

[diagram or code sample]

Next Steps

Evaluate your agent's robustness: ? Lab 035 ? Agent Evaluation Responsible AI fundamentals:

? Lab 008 ? Responsible AI for Agents

Supporting kit

The site build generates stable PDF and ZIP downloads. The ZIP contains the lab guide and every supporting file declared for this lab.

[Download generated PDF](#) [Download generated ZIP](#)

Supporting file: [CHEATSHEET.md](#)

Supporting files: Lab 036: Prompt Injection Defense & Agent Security

Start with the documented local or free path. Record assumptions and validation evidence.

Keep hosted services optional unless the lab marks them required.

Supporting file: [README.md](#)

Lab 036: Prompt Injection Defense & Agent Security

Use these files with the lab guide and generated kit.