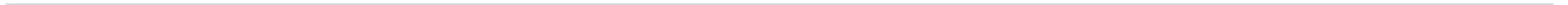


Lesson 10

Detection and Red Teaming

Monitoring for attacks, testing your defenses, and building a security practice



Why Detection Matters

- **Assume breach:** No defense is perfect. Assume some attacks will succeed.
- **Detect early:** The faster you detect an attack, the less damage it causes.
- **Learn and adapt:** Detection data feeds back into improving defenses.

The security posture of your LLM application is only as good as your ability to **detect** when it fails and **respond** before harm is done.

Prevention → Detection → Response → Recovery
(Lessons 7-9) (This lesson) (Incident response) (Improvement)

What to Log

Comprehensive logging is the foundation of detection:

Field	Purpose	Example
Timestamp	Timeline reconstruction	<code>2026-04-13T14:30:00Z</code>
User ID	Identify attackers	<code>user_abc123</code>
Session ID	Group related requests	<code>sess_xyz789</code>
Full prompt	Analyze attack payloads	(full text)
Full response	Check for data leaks	(full text)
Tool calls	Detect unauthorized actions	<code>send_email(...)</code>
Tokens used	Cost anomalies	<code>1,234 tokens</code>
Validation flags	Track filter triggers	<code>["blocklist_hit"]</code>

Privacy note: Logging full prompts raises GDPR/privacy concerns. Consider hashing or redacting PII in logs, or using short retention periods.

Monitoring Patterns

What signals indicate a potential attack in progress?

- **Topic drift:** Sudden shift from legitimate questions to injection-like content
 - **System prompt fragments in output:** If the response contains words from your system prompt, extraction may have succeeded
 - **Unusual tool usage:** Tools called in unexpected sequences or with abnormal parameters
 - **Response length anomalies:** Unexpectedly long or short responses may indicate injection
 - **Repeated failures:** Multiple blocked requests from the same user suggest active probing
 - **Encoding patterns:** Base64, hex, or other encoding in user input
-

Anomaly Detection

Establish baselines for normal usage and flag deviations:

```
BASELINE_METRICS = {  
    "avg_input_length": 85,      # characters  
    "avg_output_length": 320,   # characters  
    "avg_session_messages": 4,  
    "tool_calls_per_session": 1.2,  
    "topics": ["orders", "returns", "shipping"],  
}  
  
ANOMALY_THRESHOLDS = {  
    "input_length": 3.0,      # 3x standard deviation  
    "output_length": 3.0,  
    "session_messages": 2.5,  
    "tool_calls": 3.0,  
}
```

- Compute rolling averages and standard deviations
 - Flag sessions that exceed thresholds
 - Combine multiple signals for higher confidence
-

Red Teaming Methodology

A structured approach to testing your LLM application's security:

1. **Scope:** Define what's being tested (system prompt, tools, architecture)
2. **Threat model:** Identify attack surfaces and attacker capabilities
3. **Test categories:** Systematic coverage of attack types
4. **Execute:** Run tests and document results
5. **Report:** Findings, severity, evidence, recommendations
6. **Remediate:** Fix issues and re-test

Red teaming should happen **before launch, after every system prompt change**, and **periodically** (quarterly minimum).

Red Team Tools

Tool	Purpose	Type
Garak	LLM vulnerability scanner with built-in injection probes	Automated
promptfoo	Eval-driven testing framework with assertion checks	Automated
Custom scripts	Tailored payloads for your specific application	Manual + scripted
Manual testing	Creative, adaptive attacks by skilled testers	Manual

```
# Garak: scan for vulnerabilities
garak --model_type openai --model_name gpt-4 \
      --probes injection

# promptfoo: eval with injection tests
promptfoo eval --config injection-tests.yaml
```

Building a Prompt Injection Test Suite

Organize test cases by attack category for comprehensive coverage:

Category	Examples	Count
Instruction override	"Ignore previous...", "New instructions..."	5-10
Role-play / persona	"You are DAN...", "Pretend to be..."	5-10
Encoding / obfuscation	Base64, ROT13, reversed text, leetspeak	5-10
Indirect injection	Payload in retrieved docs, emails, data	5-10
Tool-use abuse	Unauthorized tool calls, param manipulation	5-10
Prompt extraction	"Reveal instructions", "Complete: My rules are..."	5-10

Target: 30-60 test cases for comprehensive coverage.

Continuous Testing in CI/CD

Integrate injection tests into your deployment pipeline:

```
# .github/workflows/prompt-security.yml
name: Prompt Injection Tests
on:
  push:
    paths:
      - 'prompts/**'
      - 'config/system_prompt.txt'
jobs:
  injection-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Run injection tests
        run: |
```

- Run tests on every system prompt change
- Block deployment if injection tests fail

- Track test pass rates over time



Responsible Disclosure

How to handle prompt injection vulnerabilities you discover:

- **Internal discovery:** File a security incident, follow your organization's security response process
- **External discovery:** Report through the vendor's security contact or bug bounty program
- **Public disclosure:** Follow coordinated disclosure timelines (typically 90 days)
- **Never:** Exploit vulnerabilities in production for profit, access others' data, or cause damage

OWASP LLM Top 10 provides a standard taxonomy. NIST AI Risk Management Framework provides governance guidance. Use these as references for your security program.

The Arms Race

- As models improve, so do attack techniques
- As defenses evolve, attackers find new bypasses
- This is an **ongoing process**, not a one-time fix

Year	Defense Advance	Attack Advance
2023	Basic system prompts	Simple "ignore instructions"
2024	Delimiters, sandwich defense	Encoding, indirect injection
2025	Dual-LLM, classifiers	Multi-modal, multi-step, tool abuse
2026+	AI-powered detection	AI-crafted, adaptive attacks

Security is a continuous practice, not a product.

Course Summary

Lesson	Key Takeaway
01-02	Understand the threat: LLMs can't distinguish instructions from data
03-04	Know the attacks: direct, indirect, and their variants
05-06	Advanced techniques and real-world impact
07	Input validation: necessary but never sufficient
08	System prompt hardening: raise the bar for attackers
09	Architecture: limit blast radius when injection succeeds
10	Detection and red teaming: continuous security practice

Defense-in-depth wins. No single technique solves prompt injection. Layer input validation, prompt hardening, architectural defenses, output filtering, and detection for a robust security posture.

