AI Red Teaming for AI and Machine Learning Systems: Testing LLMs, Agents, and Models

Compliance Guide for AI/Machine Learning Teams

Prepared by Jacobian Engineering | 2026-02-09

This guide is for informational purposes only and does not constitute legal advice.

Executive Summary

AI systems fail in ways that traditional software tests do not always catch. Large language models can be manipulated with prompt injection. Agents can be tricked into making tool calls that expose data. Classifiers can be fooled by adversarial inputs. Even when security controls exist, unsafe behavior can slip through because the model itself is a moving part.

AI red teaming is a structured testing approach designed to uncover these failures before customers do. It blends adversarial thinking, security testing, and model behavior evaluation. This guide explains how to plan and run AI red teaming for machine learning systems, including LLMs and agentic workflows. It also explains how to turn the results into fixes, monitoring, and evidence that supports customer trust.

What AI red teaming is and what it is not

AI red teaming is adversarial testing focused on AI specific risks. It looks for ways a model can be manipulated, misused, or produce harmful results. It is not a replacement for standard penetration testing. You still need to test APIs, authentication, and infrastructure. Red teaming adds another layer by testing the model and the surrounding workflow as a complete system.

How AI red teaming differs from traditional testing

- **Behavioral focus:** It tests outputs and decision paths, not only code paths.
- **Adversarial inputs:** It assumes attackers will craft inputs to break safety and security boundaries.
- **System perspective:** It evaluates the combination of model, prompts, retrieval data, tools, and user interface.

Why does this matter. If an attacker can cause a model to reveal secrets, call internal tools, or generate unsafe content, the impact can be real even when the infrastructure is patched and configured correctly.

When AI teams should run red teaming

Red teaming is most effective when it is scheduled and repeatable. Waiting until a customer reports an issue is too late. A practical program runs tests at defined points in the lifecycle.

- **Before first launch:** Validate the basic safety and security posture of the AI feature.
- **After major model changes:** New fine tuning, new retrieval sources, or new tool integrations change the threat surface.
- **Before entering regulated markets:** When the model influences sensitive decisions, testing should be deeper and documented.
- **After incidents or near misses:** Use red teaming to validate fixes and to search for similar weaknesses.

Define scope and objectives

Scope is the difference between useful testing and chaos. Start by defining the system boundary. Which model versions are in scope. Which user interfaces. Which tools can the model call. Which data sources can it retrieve. Which logs capture prompts and outputs.

Then define objectives. Are you primarily testing data leakage. Are you testing misuse. Are you testing decision integrity. Are you testing regulatory expectations. A good test plan states what success looks like.

Questions that help define scope

- **Data:** What sensitive data could be exposed through prompts, retrieval, or outputs.
- **Tools:** What actions can the model take, and what can go wrong if it is tricked into taking them.
- **Users:** Who can access the system, and how could access be abused.
- **Environment:** Is testing done in production, staging, or a dedicated sandbox.

Common AI red team test categories

Prompt injection and instruction hierarchy attacks

Prompt injection is a technique where an attacker crafts input to override system instructions or to manipulate the model into doing something unintended. This is especially important in systems that retrieve external content or allow users to upload documents. The model may treat untrusted content as instruction.

- **Direct injection:** User input tries to override safety or policy constraints.
- **Indirect injection:** Content retrieved from a document or web source includes hidden instructions.
- **Tool abuse:** Injection triggers tool calls that expose data or change state.

Data leakage and privacy exposure

AI systems can leak sensitive data through outputs, logs, or embeddings. Red teaming tests whether the system reveals secrets, personal data, or customer content. It also tests whether memory features and conversation history behave as intended.

- **Secrets exposure:** API keys, tokens, internal URLs, or system prompts appearing in outputs.
- **Cross tenant leakage:** One customer can retrieve another customer’s content through prompts or retrieval.
- **Training data memorization:** The model reproduces sensitive training examples.

Model misuse and abuse scenarios

Attackers may use AI systems for fraud, phishing, malware assistance, or harassment. Red teaming tests whether guardrails are effective and whether the system can be coaxed into producing disallowed

content.

Model extraction and intellectual property risk

In some cases, attackers may try to reconstruct a model or steal its capabilities through repeated queries. Red teaming can evaluate whether rate limits, monitoring, and output controls reduce this risk.

Data poisoning and supply chain risk

If your system learns from user feedback or ingests third party data, poisoning is a risk. Red teaming can test the controls around data validation, labeling workflows, and retraining triggers.

Agents and tool calling risks

Agentic systems introduce risks beyond text output. If a model can call tools, it can change state. It can create tickets, send emails, modify records, or query internal systems. Red teaming should treat tool access like privileged access. What is the least privilege tool set the model needs. What guardrails exist to prevent dangerous sequences.

- **Tool permission review:** Verify each tool has only the permissions required for its function.
- **Input validation:** Validate what is sent to tools, not only what the model outputs.
- **Approval gates:** Require human confirmation for high impact actions.
- **Audit logs:** Ensure tool calls are logged with correlation to user sessions and model versions.

Retrieval augmented generation and untrusted content

RAG systems expand the attack surface because they ingest documents and retrieve content dynamically. That content can contain malicious instructions or sensitive data. Red teaming should test how the system handles untrusted documents and whether it can be tricked into revealing content outside the user’s permissions.

- **Content injection:** Hidden instructions in uploaded files or retrieved pages.
- **Permission bypass:** Prompts that attempt to retrieve content from other tenants or internal sources.
- **Embedding leakage:** Attempts to reconstruct sensitive documents from vector stores.

Model serving and MLOps attack paths

Red teaming should also consider the operational pipeline. If an attacker can change the model artifact, poison training data, or modify configuration, they may control outputs without touching the application code.

- **Model registry controls:** Who can publish or promote models, and how approvals are recorded.
- **CI/CD integrity:** How model and code artifacts are built, signed, and deployed.
- **Secrets and credentials:** Whether model serving uses properly managed secrets and rotation.

Mitigation patterns to test and validate

Red teaming is more valuable when it validates mitigation strategies. Common patterns include prompt hardening, output filtering, retrieval filtering, strict tool schemas, and rate limiting. The key question is whether the mitigation holds under adversarial pressure. If a filter works for normal misuse but fails under crafted prompts, it needs improvement.

Severity, prioritization, and ownership

AI findings can feel subjective. To keep remediation practical, define a severity approach. Consider impact on confidentiality, integrity, availability, and user harm. Tie each finding to an owner and a target timeline. If you cannot assign ownership, the issue will linger.

Testing safely without leaking real data

AI red teaming should be done with care. Use staging environments when possible. Use synthetic or anonymized data sets for retrieval and fine tuning. Limit who can run tests and where results are stored. If the red team discovers sensitive outputs, handle them like security incidents.

Methodology for running an AI red teaming engagement

Phase 1: Threat modeling and test design

Start with a threat model that covers the model, the application, and the business context. Identify assets, threats, and potential impacts. Then design test cases that target the highest risk areas.

Phase 2: Manual adversarial testing

Manual testing is essential because real attackers are creative. Testers explore prompts, tool chains, and edge cases. They try to break safety constraints, extract hidden instructions, and trigger unintended actions.

Phase 3: Automated testing and regression suites

Automation helps scale coverage. Build a suite of adversarial prompts and abuse scenarios that can be run regularly. Keep the tests versioned so you can see whether a change improved or degraded behavior.

Phase 4: Findings, remediation, and retesting

Findings should be actionable. Each issue should include reproduction steps, impact, and recommended mitigations. After fixes are applied, retest to confirm the issue is resolved. Record the results so you can prove improvement over time.

Phase 5: Operationalize monitoring

Red teaming should feed monitoring. If a prompt pattern represents an attack, create detection signals. If a tool call is risky, add alerting and approval gates. The goal is to make adversarial behavior visible in production.

Building a sustainable red teaming program

A one time engagement can find issues, but AI systems change quickly. The strongest programs treat red teaming like a cycle. Teams run smaller tests frequently and deeper tests at milestones. This approach reduces the cost of each test because the system is already instrumented and the team already knows how to respond.

Create a living test library

Capture prompts, adversarial examples, and abuse scenarios in a versioned library. Link them to findings and mitigations. When a model changes, rerun the library. If a past issue returns, you will catch it early.

Assign a response owner

When the red team finds an issue, someone must own the fix and the retest. That sounds obvious, but it often breaks down in cross functional teams. Define ownership in advance. Is it the AI platform team. The application team. The security team. A clear answer speeds remediation.

Connect red teaming to release management

For high impact systems, add red team checks to release gates. You do not need to retest everything for every change. You do need a trigger based approach. New tools and new retrieval sources should trigger deeper tests. Minor prompt changes may trigger a lighter suite.

Use metrics that improve over time

Track measurable improvements such as reduced high severity findings, faster remediation times, and fewer production incidents. Metrics are not the goal, but they help leadership understand whether the program is working.

Turning red team results into defensible evidence

Customers and auditors often ask, “How do you test your AI system.” Red teaming can provide a strong answer if it is documented. Keep a report that includes scope, methodology, summary of findings, remediation actions, and retest results. Tie the work to your broader security program. If your organization has an incident response plan, show how AI incidents fit into it.

Business benefits

AI red teaming reduces the chance of embarrassing failures and costly incidents. It can also shorten customer reviews because you can explain your testing approach clearly. For products entering regulated environments, red teaming supports risk management and quality expectations. A disciplined program also helps engineering teams ship faster because they know what they must validate before release.

How Jacobian Engineering supports AI red teaming

Jacobian Engineering performs AI red teaming as part of a broader security testing practice. The team tests the AI system and the surrounding application, including APIs, authentication, and cloud configuration. The engagement typically includes threat modeling, adversarial testing, remediation guidance, and retesting. Jacobian can also help teams build repeatable test suites and monitoring

coverage so red teaming becomes part of an ongoing program.

Conclusion

AI red teaming is a practical way to understand how your system behaves under adversarial pressure. It is not a marketing exercise. It is a way to find weaknesses, fix them, and build confidence that the system is safe to operate. If you run tests before launch, after major changes, and as part of continuous monitoring, you will reduce risk and improve trust.

If you need a structured red teaming approach that fits your product and your risk profile, Jacobian Engineering can help you plan, execute, and operationalize AI red teaming for your AI and machine learning systems.