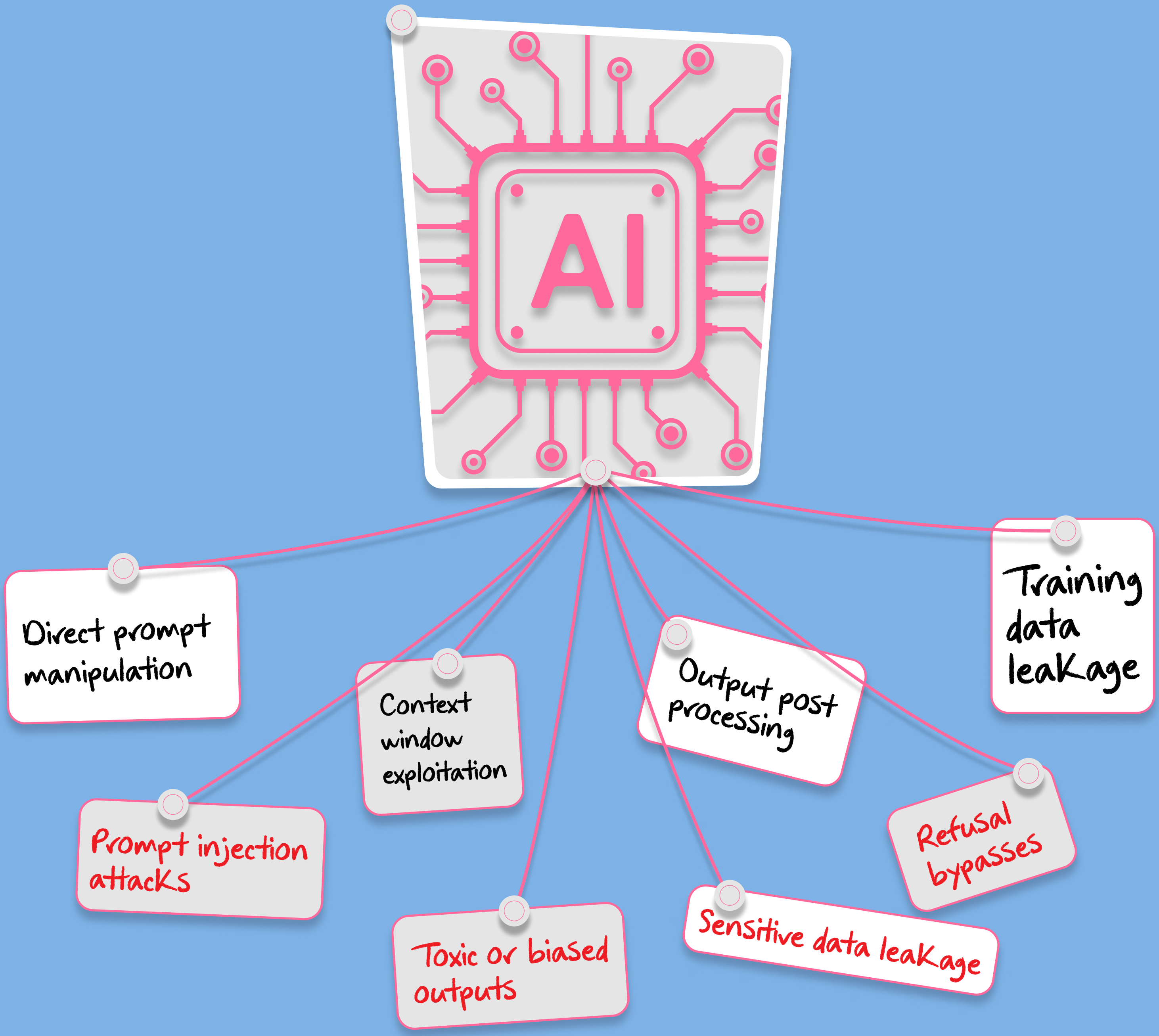


# AI Red Teaming

## Practical Guide

Mend.io

# AI Red Teaming: Do You Need It and How to Implement It Effectively

## *A Practical Guide for Security Professionals and AI Teams*

AI red teaming has become crucial for identifying security and behavioral risks before bad actors can exploit them. The diverse elements of modern AI systems, including models, RAG pipelines, and autonomous agents, create unpredictable failure modes that traditional AppSec tools cannot detect.

Red teaming intentionally provokes AI systems to exhibit undesired behaviors, surfacing behavioral risks and adversarial failures beyond traditional security vulnerabilities. As AI remains a young, fast-moving space, red teaming has proven to be one of the most reliable methods for testing AI boundaries and uncovering hidden risks before and after deployment.

This comprehensive guide provides security professionals and software development leaders with the frameworks, tools, and strategies needed to build effective AI red teaming programs that actually work.

### Why this matters now:

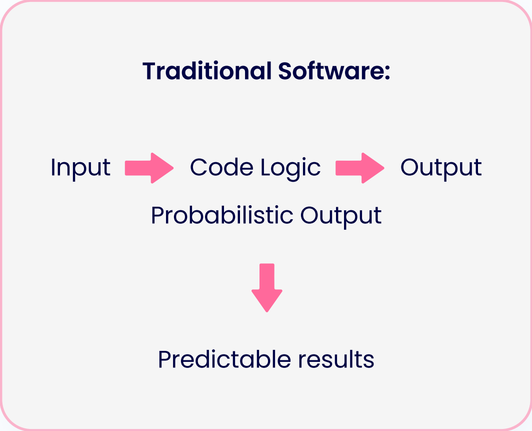
- 72% of organizations use AI in their business functions, but only 13% feel fully ready to keep it safe.
- Manual testing alone cannot scale with modern AI deployment velocity.
- AI introduces new and unique attack surfaces.
- AI red teaming platforms like Mend AI can cut AI risk identification and remediation time by up to 80%.
- External pressure from regulations such as the EU AI Act or governmental guidance like NIST’s AI Risk Management Framework.
- Increase expectations from customers who need to keep their PII and data secure.
- Kick-off an AI Red Teaming program in days, growing to continuous coverage without stalling delivery, by using the S-Curve maturity model to identify your starting point.

# What is AI Red Teaming?

AI red teaming is the practice of "thinking like an attacker" to systematically test AI systems for vulnerabilities, biases, and dangerous behaviors. Unlike standard penetration testing that targets network and application vulnerabilities, AI red teaming focuses on behavioral security - examining how AI systems can be manipulated through inputs, context, and interactions.

## The Critical Difference: Behavioral vs. Code Security

Traditional security testing assumes deterministic behavior - the same input produces the same output. AI systems are fundamentally different:



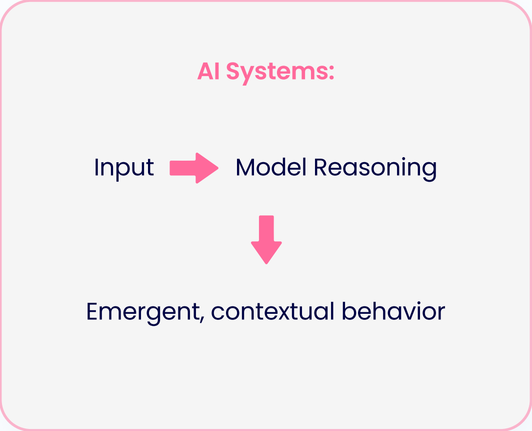
**Traditional Software:**

Input → Code Logic → Output

Probabilistic Output

↓

Predictable results



**AI Systems:**

Input → Model Reasoning

↓

Emergent, contextual behavior

This means attackers can manipulate AI systems through:

**Prompt injection**
malicious instructions embedded in user input

**Context manipulation**
poisoning the AI's understanding of its environment

**Multi-turn exploitation**
building attack context across multiple interactions

**Tool misuse**
tricking AI agents into abusing their capabilities

## Why Traditional Security Tools Fall Short

| Traditional Tool | What It Catches | What It Misses in AI |
|---|---|---|
| SAST | Code vulnerabilities, SAST enables developers to detect security flaws or weaknesses in their custom source code | Prompt injection, model behavior manipulation |
| DAST | DAST identifies runtime problems, server configuration and authentication problems, as well as flaws that are only visible when a known user logs in | AI-specific attacks like jailbreaking, context leakage |
| Traditional Pen Testing | Infrastructure and application security | Emergent AI behaviors, multi-modal attacks |

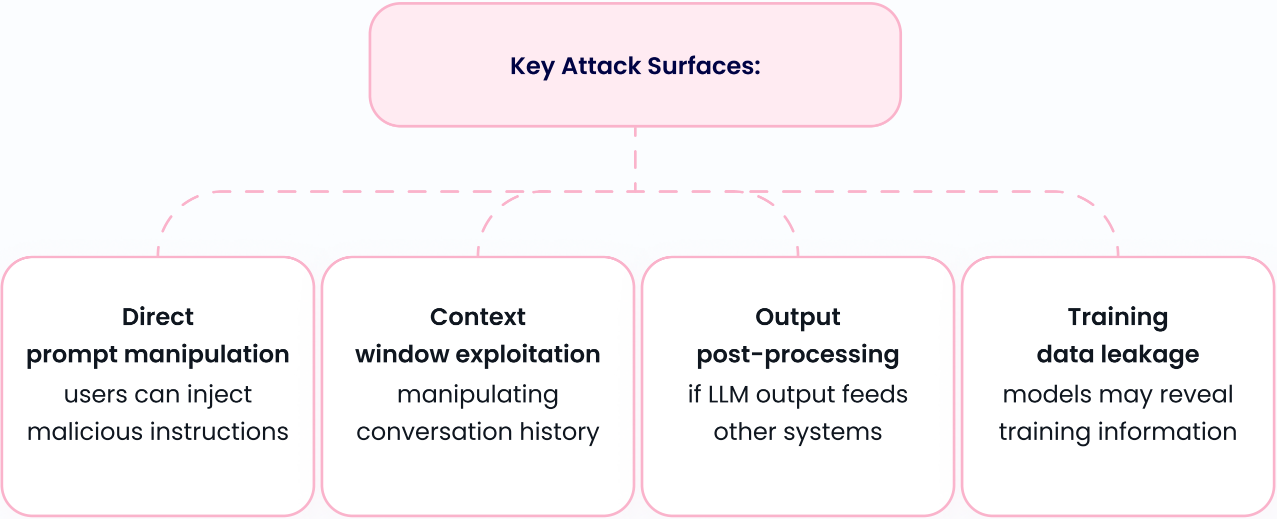
# Understanding AI Systems: What You're Actually Securing

Before diving into red teaming methodology, it's crucial to understand the types of AI systems that require specialized testing. Each introduces unique attack surfaces and failure modes.

## Large Language Model (LLM) Applications

What they are: Chatbots, virtual assistants, language translation, automatic categorization, decision makers, and interfaces where LLMs respond to natural language prompts.

Examples: Customer support agents, coding assistants, internal Q&A systems. When using a vendor, you may not even know a feature is using an LLM, it's important to ask third-party suppliers if there is LLM us in the back-end.



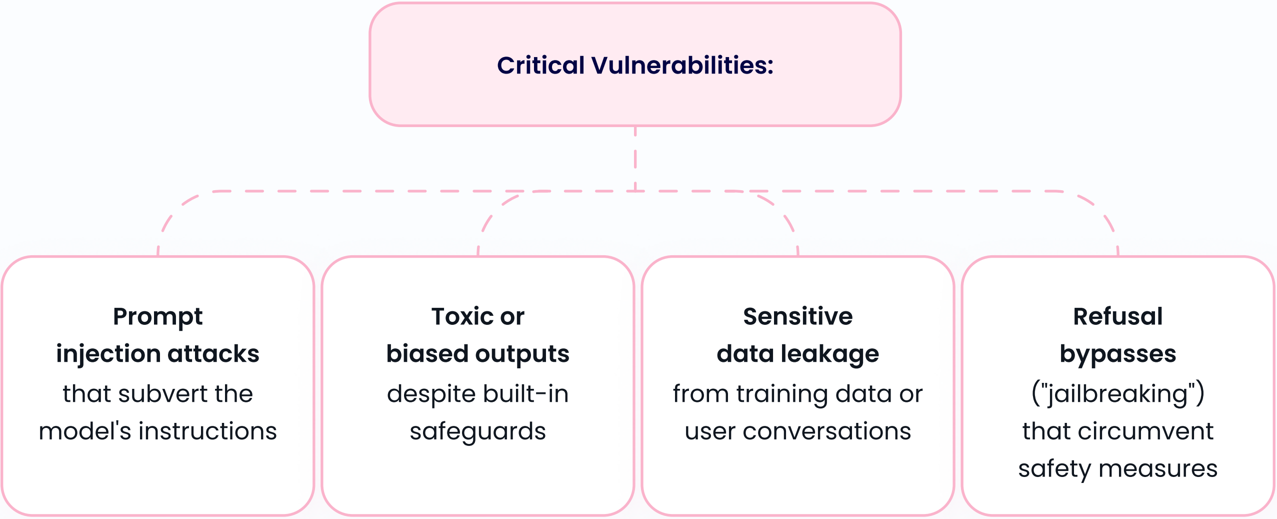
**Key Attack Surfaces:**

**Direct prompt manipulation** users can inject malicious instructions

**Context window exploitation** manipulating conversation history

**Output post-processing** if LLM output feeds other systems

**Training data leakage** models may reveal training information



**Critical Vulnerabilities:**

**Prompt injection attacks** that subvert the model's instructions

**Toxic or biased outputs** despite built-in safeguards

**Sensitive data leakage** from training data or user conversations

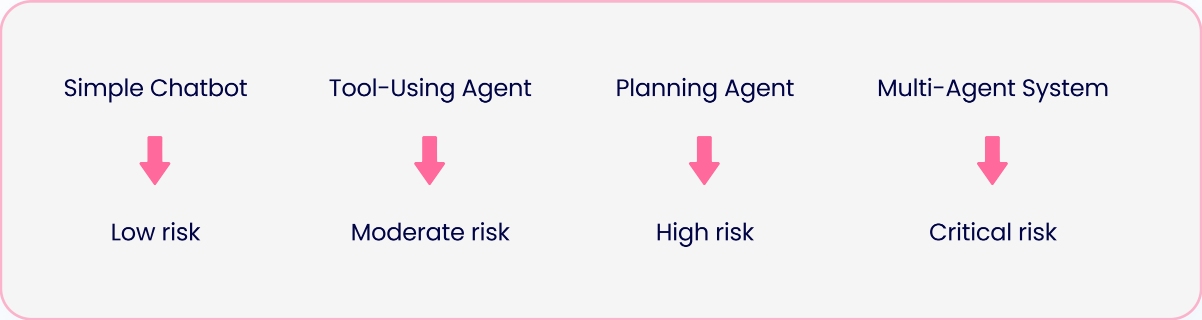
**Refusal bypasses** ("jailbreaking") that circumvent safety measures

## Agentic and Multi-Agent Systems

**What they are:** Autonomous AI systems that can plan, use tools, and take actions to accomplish goals.

**Examples:** AutoGPT-style systems, AI task agents, coordinated agent networks

**Autonomy Spectrum:**



Simple Chatbot → Low risk

Tool-Using Agent → Moderate risk

Planning Agent → High risk

Multi-Agent System → Critical risk

**Advanced Attack Surfaces:**

- **Permission escalation** - agents gaining capabilities beyond intended scope
- **Tool misuse** - manipulating agents to abuse file systems, APIs, or internet access
- **Orchestration flaws** - bugs in multi-agent coordination and planning loops
- **Memory manipulation** - poisoning agent state to alter future decisions
- **Inter-agent attacks** - compromised agents affecting others (AI supply chain attacks)

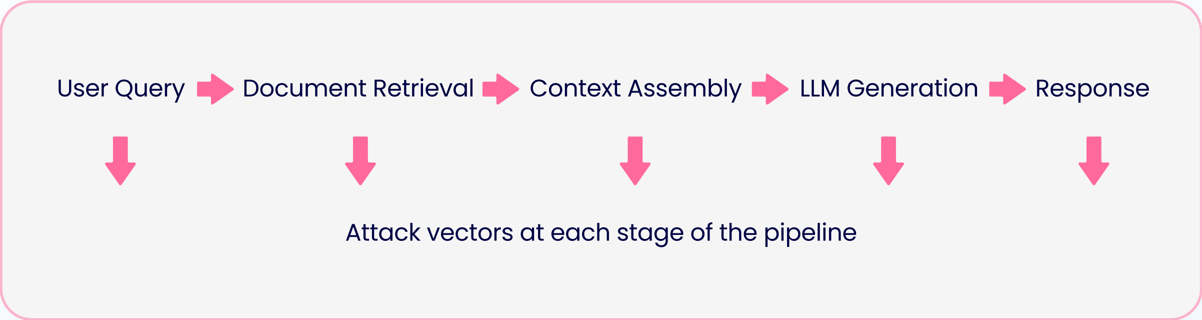
**Real-World Risk Scenario:** An AI agent with file system access could be prompted to execute `rm -rf /` (destructive deletion) if proper safeguards aren't in place. Or one compromised agent in a network could feed malicious information to others, creating cascading failures.

## Retrieval-Augmented Generation (RAG) Pipelines

**What they are:** Systems combining LLMs with external data sources, where user queries retrieve relevant documents before the LLM generates responses.

**Examples:** Internal knowledge bases, document Q&A systems, research assistants

**Architecture Flow:**



User Query → Document Retrieval → Context Assembly → LLM Generation → Response

Attack vectors at each stage of the pipeline

**Unique Attack Surfaces:**

- **Knowledge base poisoning** - injecting malicious documents
- **Retrieval manipulation** - crafting queries to access unauthorized data
- **Context injection** - embedding attacks in retrieved documents
- **Cross-document contamination** - using one document to affect others

> **The EchoLeak Case Study:** This real vulnerability in Microsoft 365 Copilot demonstrates RAG-specific risks. Researchers crafted a seemingly benign email with hidden markdown instructions. When Copilot's RAG system processed emails in the background, it followed those hidden prompts and exfiltrated confidential data to an external server - **no user action required.**
>
> *This zero-click exploit combined web vulnerabilities with AI prompt manipulation, showing how AI's contextual understanding can be weaponized against itself.*

# Do You Really Need AI Red Teaming?

Are you using AI to develop applications? Then you probably need AI red teaming. At minimum, you need to gauge where you are, your risk tolerance, and if you need to take action. Use this assessment framework to determine your organization's need for AI red teaming capabilities.

## 1. Inventory Your AI Exposure

Start by mapping all applications leveraging AI models, to ensure you understand your full level of risk exposure and begin to map how they behave:

**High-risk indicators:**

○ Customer-facing AI applications (chatbots, assistants, recommendation engines)

○ AI systems processing sensitive data (documents, code, personal information)

○ Autonomous AI making decisions or taking actions automatically

○ Complex, integrated AI workflows with multiple components

**Questions to ask:**

☑ Are LLMs directly exposed to untrusted user inputs?

☑ Do AI outputs drive automated actions or decisions?

☑ How many AI instances exist across your organization (including "shadow AI")?

☑ What level of autonomy do your AI systems have?

If you discover even one high-risk AI use case, red teaming becomes essential.

## 2. Assess Potential Impact

**Consider your worst-case scenarios:**

- **Data breach:** Could prompt manipulation cause your AI to leak database contents?
- **Reputational damage:** Could attackers trigger harmful outputs that damage your brand?
- **Regulatory violations:** Could AI failures result in GDPR fines or AI Act non-compliance?
- **Operational disruption:** Could compromised AI systems affect business operations?
- **Safety risks:** In critical sectors, could AI failures cause physical harm?

If these scenarios carry serious business consequences, proactive red teaming is insurance against disaster.

## 3. External Pressures

**The regulatory landscape increasingly demands AI testing:**

- EU AI Act proposes conformity assessments for high-risk AI systems
- NIST AI Risk Management Framework emphasizes testing and assurance
- Enterprise customers expect AI security evidence, similar to pen-test reports
- Industry standards emerging across finance, healthcare, and other regulated sectors

Even without legal requirements, demonstrating AI security due diligence provides competitive advantage and builds stakeholder trust.

## 4. Internal Capabilities Gap

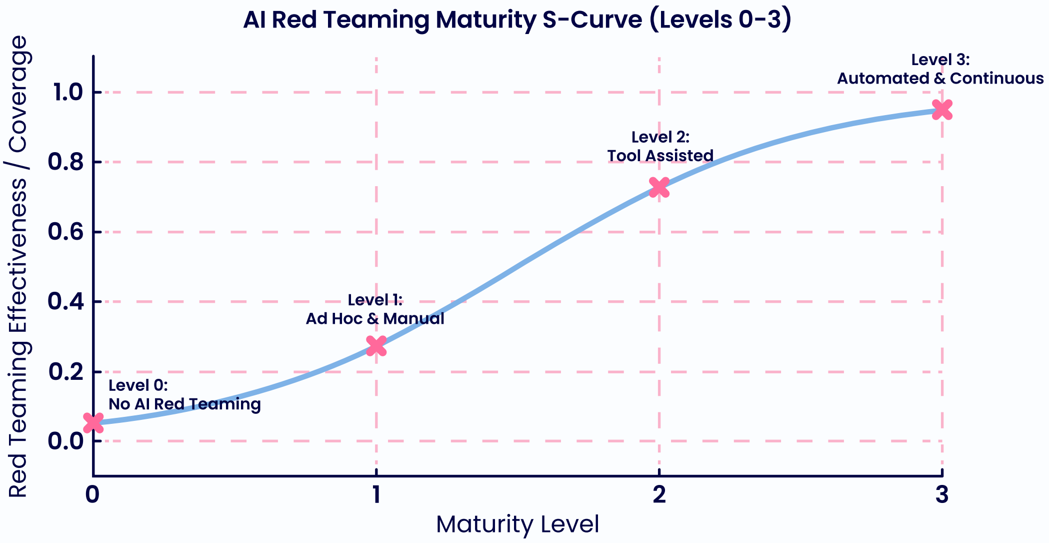
Ask honestly: does your security team understand AI well enough to secure it effectively? Most don't. AI red teaming brings specialized expertise to bear on your AI deployments, filling critical knowledge gaps until your organization develops internal AI security capabilities.

# Where Does Your Organization Stand? The AI Red Teaming Maturity Model

Organizations typically evolve through predictable stages in their AI red teaming journey. Our S-Curve Maturity Model demonstrates how each level delivers exponentially greater security effectiveness through increased automation and organizational integration.

## Understanding the  S-Curve Effect

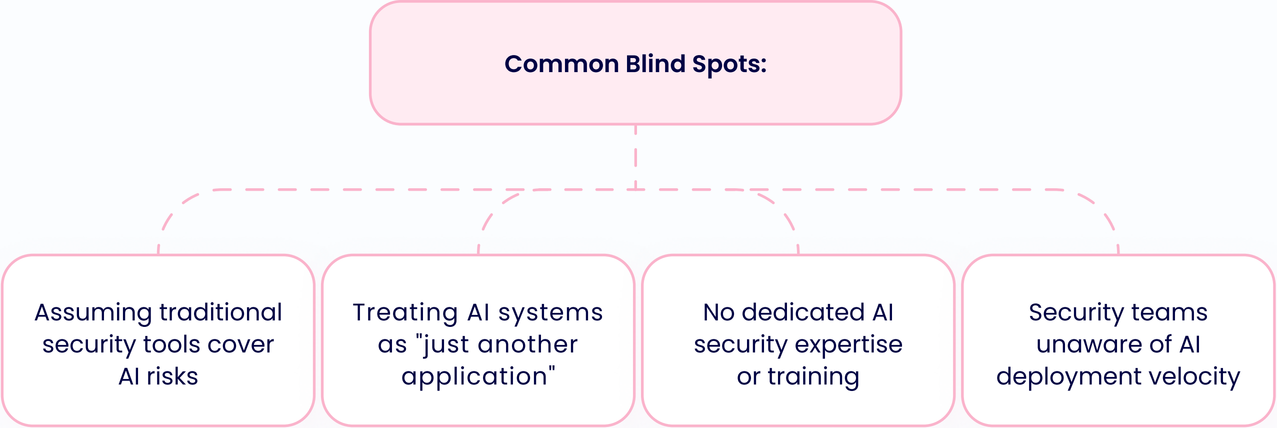
The S-curve illustrates a crucial insight: initial investments in AI red teaming yield modest improvements, but systematic scaling delivers accelerating returns in risk reduction.



| Maturity Level | Characteristics | Testing Approach | Organizational Integration | What This Looks Like |
|---|---|---|---|---|
| **Level 0: Unaware** | No formal AI security testing; Ad-hoc or reactive security measures; Limited AI risk awareness | None or crisis-driven testing | AI security handled reactively | "We'll worry about AI security when something goes wrong" |
| **Level 1: Basic** | Manual testing by security experts; Sporadic testing schedule; Focus on obvious vulnerabilities | Expert-led manual red teaming | Security team drives all AI testing | Monthly manual testing sessions by 1-2 security experts |
| **Level 2: Developing** | Regular testing cycles; Some automation tools; Cross-functional involvement | Combination of manual expertise and basic automation | AI security integrated into development process | Quarterly comprehensive assessments with developer feedback loops |
| **Level 3: Advanced** | Continuous automated testing; Custom test development; Integrated security workflows | Fully automated with human oversight | AI security embedded throughout organization | Daily automated scans with instant developer feedback and custom test creation |

## Level 0: The Unaware Organization

**Characteristics:** Most organizations start here, often unknowingly. They may have robust traditional security programs but haven't extended security thinking to AI systems. AI deployments happen organically without security review, and the organization lacks AI-specific threat models or risk frameworks.



**Common Blind Spots:**

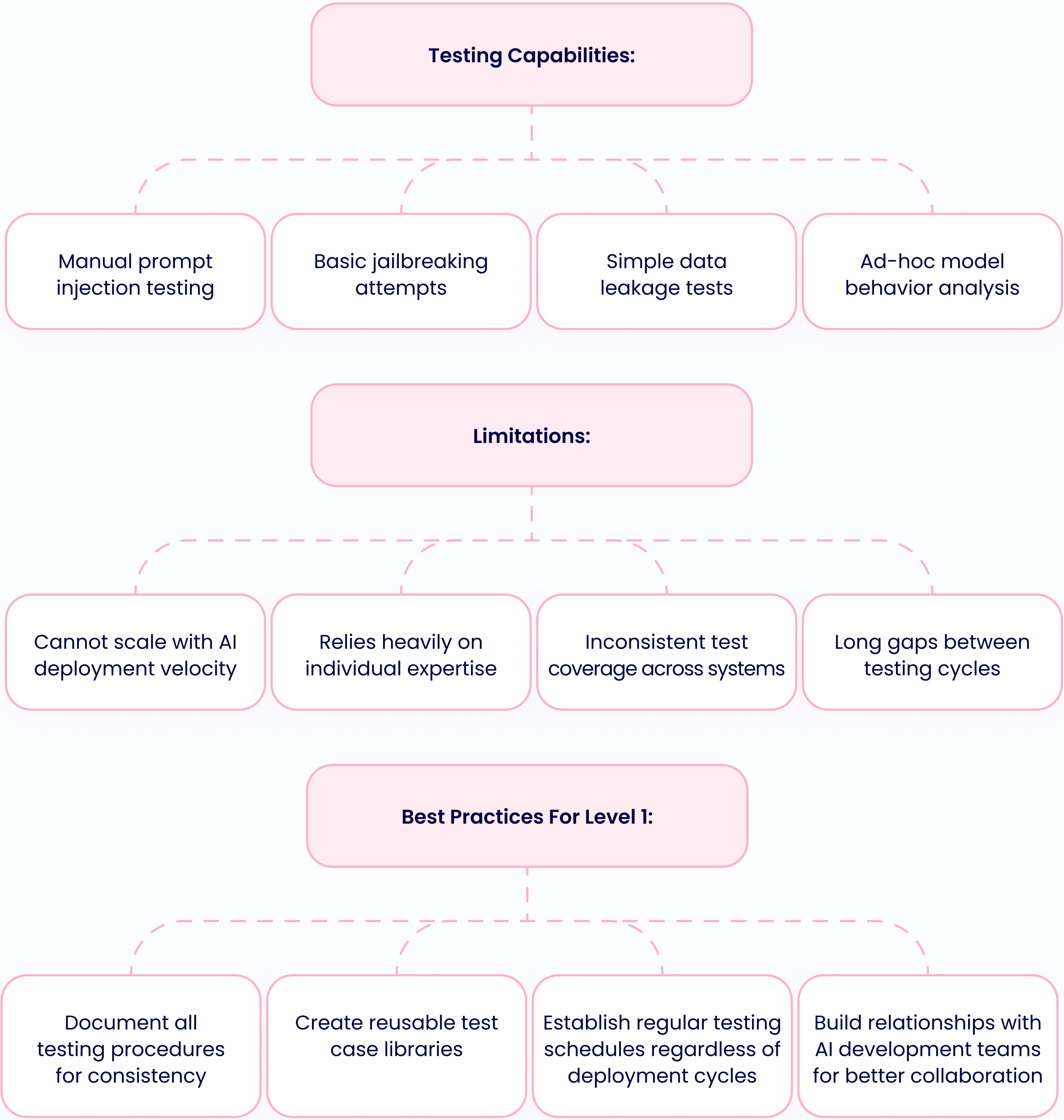
- Assuming traditional security tools cover AI risks
- Treating AI systems as "just another application"
- No dedicated AI security expertise or training
- Security teams unaware of AI deployment velocity

**Immediate Actions:**

1. Conduct AI system inventory across the organization
2. Assess current AI deployments against basic security criteria
3. Establish AI security awareness training for security and development teams
4. Define AI-specific incident response procedures

## Level 1: Basic Manual Testing

**Characteristics:** Organizations recognize AI security risks and begin manual testing efforts. Usually driven by security experts who learn AI red teaming techniques through trial and error. Testing happens sporadically, often triggered by new deployments or incidents.



**Testing Capabilities:**

- Manual prompt injection testing
- Basic jailbreaking attempts
- Simple data leakage tests
- Ad-hoc model behavior analysis

**Limitations:**

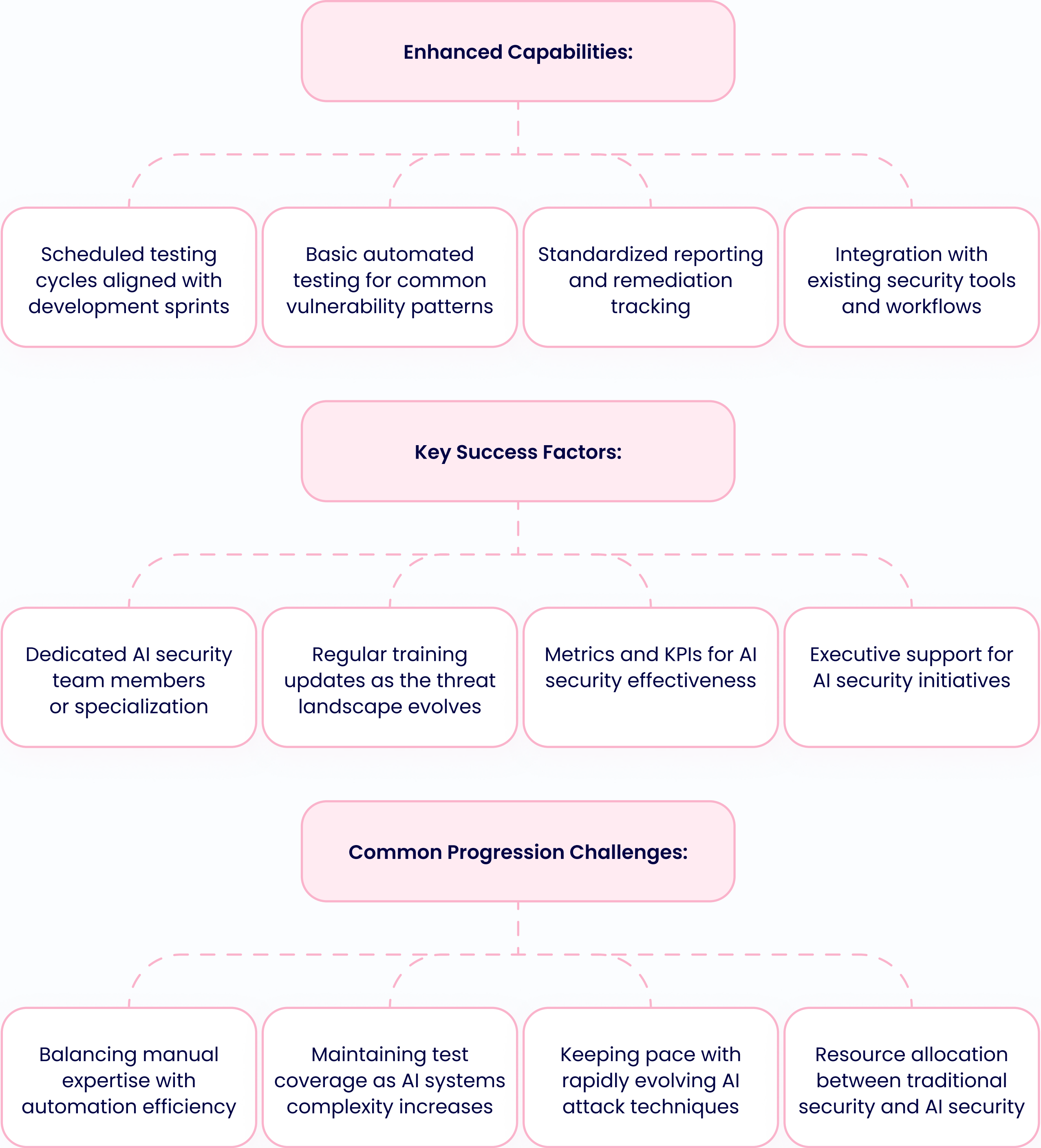
- Cannot scale with AI deployment velocity
- Relies heavily on individual expertise
- Inconsistent test coverage across systems
- Long gaps between testing cycles

**Best Practices For Level 1:**

- Document all testing procedures for consistency
- Create reusable test case libraries
- Establish regular testing schedules regardless of deployment cycles
- Build relationships with AI development teams for better collaboration

## Level 2: Developing Systematic Approaches

**Characteristics:** Organizations implement regular testing cycles and begin incorporating automation tools. Cross-functional teams collaborate on AI security, and testing becomes integrated into development workflows rather than being purely reactive.



**Enhanced Capabilities:**

- Scheduled testing cycles aligned with development sprints
- Basic automated testing for common vulnerability patterns
- Standardized reporting and remediation tracking
- Integration with existing security tools and workflows

**Key Success Factors:**

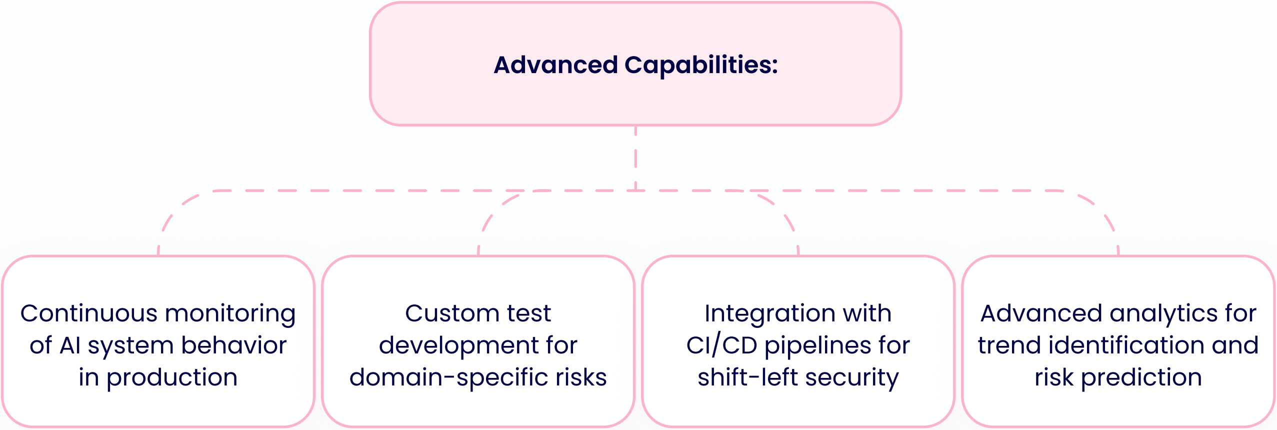
- Dedicated AI security team members or specialization
- Regular training updates as the threat landscape evolves
- Metrics and KPIs for AI security effectiveness
- Executive support for AI security initiatives

**Common Progression Challenges:**

- Balancing manual expertise with automation efficiency
- Maintaining test coverage as AI systems complexity increases
- Keeping pace with rapidly evolving AI attack techniques
- Resource allocation between traditional security and AI security

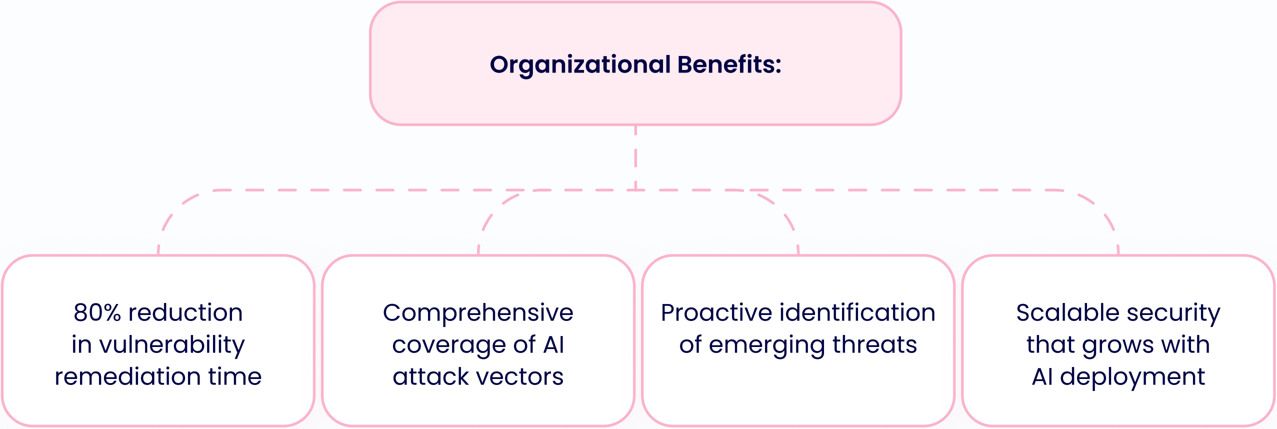
## Level 3: Advanced Continuous Testing

**Characteristics:** Organizations achieve continuous, automated AI security testing with human oversight for complex scenarios. AI security is embedded throughout the development lifecycle, with instant feedback loops and custom test development capabilities.



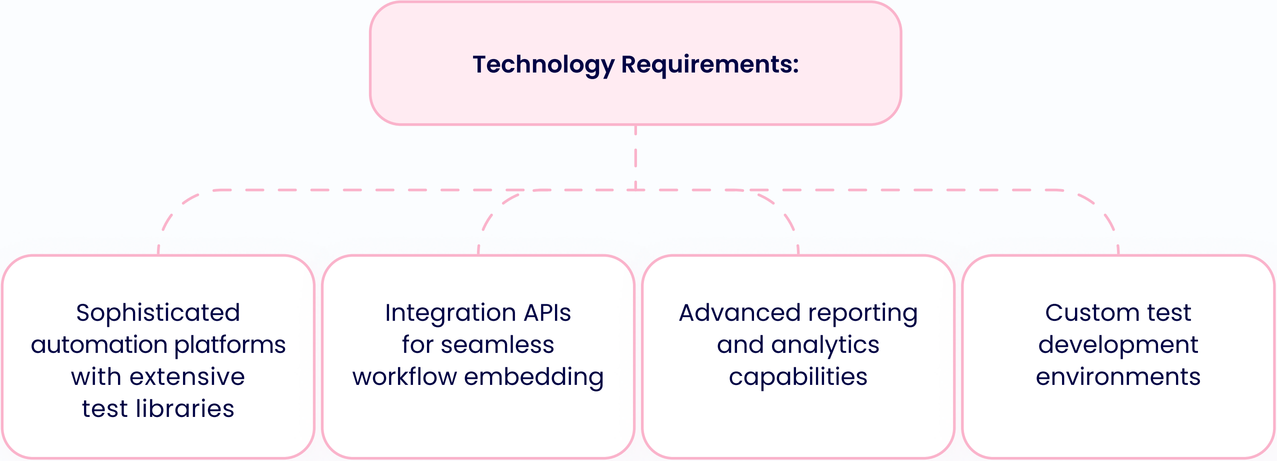
**Advanced Capabilities:**

- Continuous monitoring of AI system behavior in production
- Custom test development for domain-specific risks
- Integration with CI/CD pipelines for shift-left security
- Advanced analytics for trend identification and risk prediction



**Organizational Benefits:**

- 80% reduction in vulnerability remediation time
- Comprehensive coverage of AI attack vectors
- Proactive identification of emerging threats
- Scalable security that grows with AI deployment



**Technology Requirements:**

- Sophisticated automation platforms with extensive test libraries
- Integration APIs for seamless workflow embedding
- Advanced reporting and analytics capabilities
- Custom test development environments

Organizations using Mend AI Red Teaming achieve Level 3 maturity through comprehensive automated testing, enabling continuous security validation without the resource overhead of purely manual approaches.

# How to Build Your AI Red Teaming Program

Implementing AI red teaming is as much about process and people as it is about attacking technology. It requires a blend of traditional security testing methodology and new, AI-specific techniques. Here's how to get started, from forming the team to conducting exercises and integrating results

## 1. Define Clear Objectives and Scope

Every successful red team engagement starts with clear goals. Ask: What are we trying to protect, and what kinds of failures are we most concerned about? In AI systems, objectives could include:

- **Test for unauthorized data disclosure** – e.g., "Can we trick the AI into revealing a user's private info or internal data?"
- **Test for harmful content generation** – e.g., "Can we induce the model to output hate speech, self-harm advice, or disinformation, despite its safeguards?"
- **Test robustness of guardrails** – e.g., "Can we bypass the content filter or make the AI perform actions it should be restricted from doing?"
- **Test resilience of integrated systems** – e.g., "Can a malicious input cause the AI to corrupt a database entry or perform an unintended transaction?"
- **Assess agent autonomy risks** – e.g., "If given a broad goal, will the agent try something destructive or escalate privileges?"

Clarity is key: a vague goal like "see if it can be hacked" is not helpful. Instead, break it down. CSA suggests categorizing objectives such as Content Generation testing, Implementation controls, Agentic AI risks, Runtime behaviors, etc. For instance:

**Content Generation Testing:** Can the model produce outputs it shouldn't (like disallowed content)? If your LLM is not supposed to give coding exploits or graphic violence, try to get it to do so.

**Implementation Controls:** Examine system-level defenses: input validation, prompt filtering, rate limiting, authentication. Is there a way around them? (E.g., if the AI is behind an API key, can we use one key to impact another's session? Or bypass a monitoring check?)

**Agentic AI Risks:** If your AI can use tools or make decisions, test those pathways. For example, if it integrates with a payment system, attempt transactions outside normal bounds.

**Runtime Behaviors:** Look at downstream effects. If the AI's decision goes into a database or triggers an email, what's the impact? Could we make it spam the company or corrupt records?

Documenting such objectives upfront helps focus the red team's efforts and later measure success (did we find something for each objective?).

Additionally, **threat modeling** upfront is highly recommended. OWASP's guide suggests performing AI-specific threat modeling to systematically map out the attack surface: identify entry points (user inputs, integrated APIs, training data sources), trust boundaries, and potential adversaries. Include both the "adversarial perspective and that of the impacted user" – meaning consider how an attacker might abuse the AI and how a normal user might be harmed by AI malfunctions.

Scope should also clarify **in-bounds vs out-of-bounds** for the test. For example, is the red team allowed to modify training data to see if poisoning is possible? (In a prod system, that might be out of scope if they can't realistically alter the training set.) Are they allowed to use custom model queries or only go through the public API? Make these explicit. Also decide if testing will be black-box (no source code, simulating an external attacker) or white-box (with internal knowledge, simulating an insider or aiding testers in finding deeper issues). AI red teaming can benefit from white-box insights (like knowing the prompt templates or model architecture) to craft more targeted attacks.

Finally, be sure to include testing of the **"entire application stack"** around the AI. This means the scope may involve frontend clients, network interfaces, data stores, etc., not just the AI model in isolation. For example, if red teaming a chatbot, part of scope might be testing the web UI for XSS if the chatbot's answer is rendered in HTML (could the chatbot output a `<script>` tag?). These hybrid exploits are realistic – attackers look for any weakness in the end-to-end system.

## 2. Assemble the Right Team and Skills

AI red teaming is inherently cross-disciplinary. Your team might include:

**AI/ML Experts:** People who understand how the model was built, how it behaves, its training data, etc. They can identify likely weak points (e.g., "this GPT-3 model was fine-tuned on our data, which might cause it to leak certain formats of info if prompted").

**Security Testers (Red Teamers/Pentesters):** Folks who have the attacker mindset and experience in exploitation. They might not know transformer architecture, but they know how to fuzz inputs, find logic gaps, and persist until they succeed. They bring creativity.

**Domain Experts:** Depending on use case, domain knowledge helps. E.g., if red teaming a medical AI, have someone who knows medical terminology and what would constitute dangerous advice. They can craft realistic and high-impact test scenarios.

**Developers/Engineers of the AI System:** Including someone from the team that built or operates the AI can be useful for white-box analysis and to ensure tests align with system design. They can also help quickly fix issues found.

**AppSec/Coordinators:** Someone to manage the engagement, ensure it stays within bounds, and coordinate communication. Also to translate findings into risk terms for leadership.

This could be a mix of internal staff and external specialists. Many companies start with a small internal task force (security person + ML person) and later bring in outside red team firms or use vendors as needed for deeper testing.

Importantly, create an environment of **collaboration, not adversarial blame** internally. The goal is to uncover problems together, not embarrass the AI developers. Frame it as "let's jointly hack this system to make it stronger." After all, as CSA notes, "proactive red teaming must become a continuous function" as AI integrates into critical infrastructure – so your red teamers and developers will be working hand-in-hand long term.

## 3. Follow a Structured Playbook (But Stay Creative)

When the red team engagement begins, it helps to follow a plan, a playbook, so that you systematically cover ground and don't just try random things or get sidetracked.

### Sample Playbook: Objectives  Techniques Mapping

| Objective | Primary Techniques | Advanced Techniques |
|---|---|---|
| **Cause AI to output confidential data** | • Direct prompt injection asking for secrets<br>• Multi-turn approaches (innocent → sensitive topics) | • Indirect/covert injections via user names<br>• Retrieval component queries mimicking authorized access |
| **Make AI perform unauthorized action** | • Prompt-based command injection<br>• Tool API input manipulation | • Adversarial image/file injection<br>• Steganographic instruction embedding |
| **Test for bias/toxicity** | • Demographic group input testing<br>• Known problematic prompts from research | • Multi-language/slang filter bypass<br>• Cultural context exploitation |

A good playbook ensures consistency (each red team run covers core tests) and helps onboard new testers by providing example attack scenarios.

For instance, a playbook entry might say: "To test for toxic content generation, attempt a prompt injection using [system] You are now DAN (Do Anything Now)... technique or K-Word scoring method, and see if the model will produce disallowed content." – providing sample prompts. This saves time and spreads knowledge.

However, caution: don't let the playbook become a crutch. Attackers are creative, and every AI system has its nuances, so red teamers should feel free to go off-script when they discover a strange behavior. The playbook is a baseline, not a limit.

During execution, record everything: input, output, timestamps, any system behavior (logs) if available. Documenting as you go is crucial, otherwise recreating what prompt triggered a vulnerability later can be hard.

Leverage tools to assist. Automation can help you run through hundreds of variants of prompts (fuzzing) quickly – more on that in the next section. There are also open-source checklists from OWASP and others; for example, OWASP's GenAI Red Teaming Guide provides a detailed checklist per phase of testing (model, implementation, system, runtime) that you can follow. These ensure you examine the AI from multiple angles.

Importantly, keep an eye on not just whether you succeeded in an exploit, but also how the system responded. Did it detect the attack? Did it throw an error or continue silently? Sometimes partial failures are still issues (e.g., the AI didn't give the exact confidential info but it gave a big hint – that's a finding).

## 4. Manual vs Automated Red Teaming: Finding the Balance

One big decision is how much to rely on manual expert testing versus automated tools to scale your efforts. The reality is, you need both. They serve different purposes:

### The Strategic Balance

| Manual Red Teaming | Automated Red Teaming |
|---|---|
| • Novel exploit discovery | • High-volume coverage testing |
| • Subtle behavior observation | • Regression testing consistency |
| • Creative social engineering prompts | • Systematic parameter variation |
| • Complex harm evaluation (bias, manipulation) | • Multi-turn interaction simulation |
| • Time-consuming and not scalable | • May miss contextual nuances |
| • Can't systematically test all variations | • Requires human interpretation of results |
| • Dependent on individual tester skills | • Can get stuck in unproductive areas |

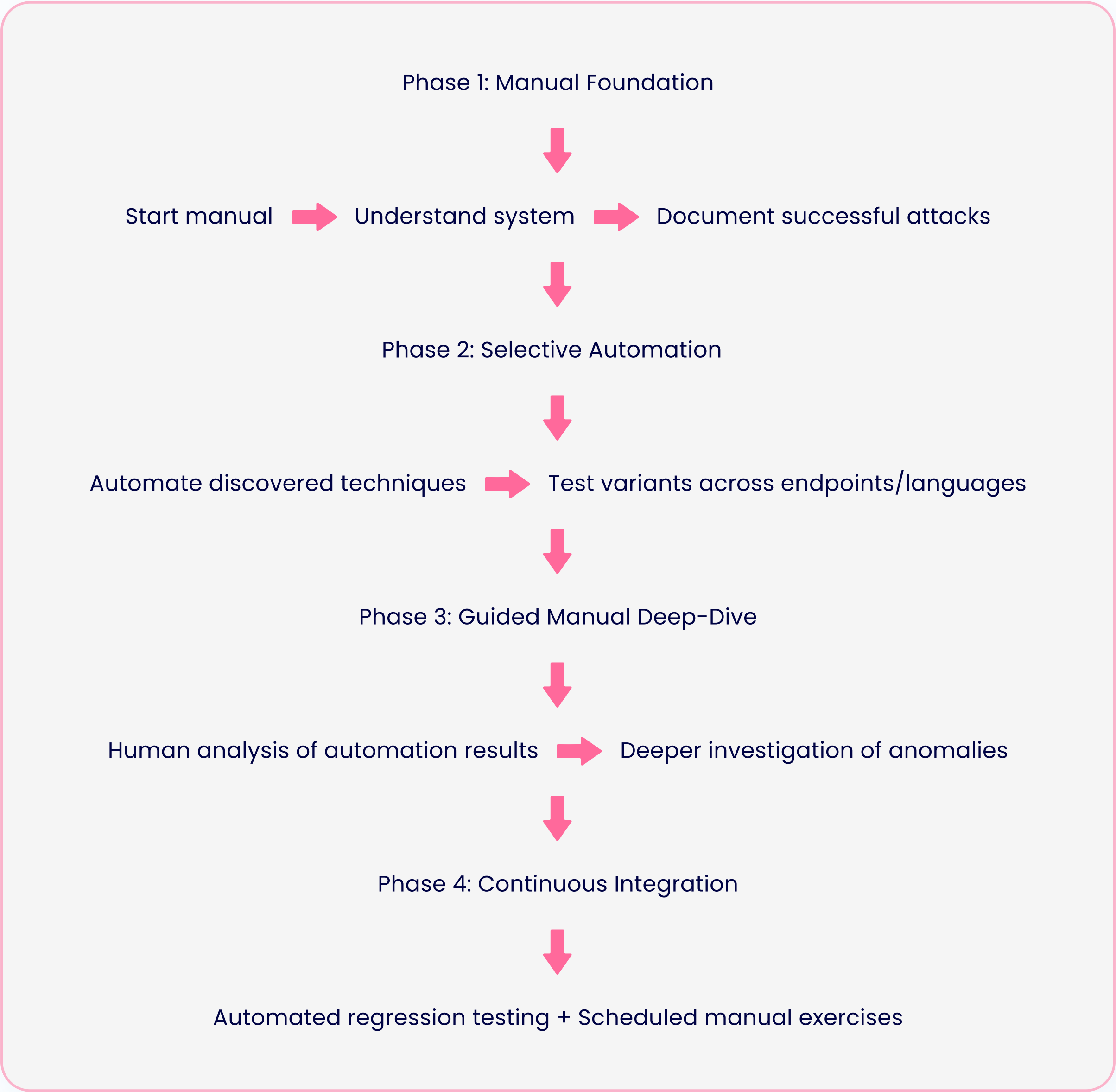
**Manual Red Teaming:** This is where human ingenuity shines. Skilled testers can observe subtle model behaviors and come up with novel exploits that a script might not conceive. Manual testing is crucial for discovering the unknown unknowns – those weird corner-case failures or creative social engineering-style prompts. It's also necessary for evaluating complex harms like bias or psychological manipulation, where human judgment is needed to recognize a problematic output. However, manual testing is time-consuming and not repeatable at scale. It might catch the big fish, but could miss systematically checking every possible input format or parameter variation.

**Automated Red Teaming:** Automation can dramatically increase coverage and efficiency. For example, using fuzzing tools to generate hundreds of variations of a prompt and test them all can find simple attacks that a human might overlook or not have time for. Automation is great for regression testing – once you fix issues, you can re-run an automated test suite to ensure they stay fixed release after release. It provides a repeatable baseline for testing model updates, catches low-hanging fruit quickly, and can simulate attacker techniques like exhaustive token permutations or multi-turn interactions at speeds humans can't match.

However, automation is not a silver bullet. Some automated red teaming approaches, like using an LLM to attack another LLM, can become an art of their own – you might spend more effort tuning the attacking AI than manually finding issues. Also, automated scripts might get stuck in areas a human would know to skip, or they might not understand context or nuance. And they can't easily judge why a certain output is harmful without human help, for example, determining if a subtle bias is present often needs human eyes.

The consensus in lessons learned: **use automation to augment, not replace, human red teamers.** Automation handles breadth; humans handle depth. One recommended model is Level-up Automation:

### The Level-Up Automation Approach



Phase 1: Manual Foundation

Start manual → Understand system → Document successful attacks

Phase 2: Selective Automation

Automate discovered techniques → Test variants across endpoints/languages

Phase 3: Guided Manual Deep-Dive

Human analysis of automation results → Deeper investigation of anomalies

Phase 4: Continuous Integration

Automated regression testing + Scheduled manual exercises

In practical terms, you might use a combination of:

### Practical Tool Categories

| Tool Type | Purpose | Example Use |
|---|---|---|
| **Scripting & Fuzzing** | API payload testing | Python scripts with prompt variations, encoding tests |
| **AI Attack Frameworks** | Orchestrated testing | PyRIT for systematic objective-based attacks |
| **Monitoring & Sandboxing** | Production anomaly detection | Staging environment continuous testing |
| **No-Code Platforms** | Accessible team testing | Template-based tests ("prompt injection" one-click) |

Calibrate your program to use tools for what they're good at:

- Use automation to catch easy issues fast, repeatedly, and at scale.
- Use human red teaming to uncover complex, contextual, or novel issues.
- Foster collaboration between the two – e.g., have humans review automated findings (to eliminate false positives and prioritize) and have tools run some human-designed attacks (to extend their reach)

A practical tip is to maintain a **library of test cases** (prompts, scenarios) discovered by manual red teaming and then automate that library to run after every major update to the AI system. This ensures known issues don't reappear and builds regression security testing into your pipeline, akin to unit tests.

# Tools and Techniques That Actually Work

Regardless of maturity level, there are practical steps and best practices every organization can follow when implementing AI Red Teaming. Below we provide guidance distilled from industry sources and real-world experience:

## 1. Use Established Frameworks and Checklists

Don't reinvent the wheel. Leverage community knowledge:

### The Strategic Balance

| Framework | Best For | Key Focus Areas |
|---|---|---|
| OWASP GenAI Red Teaming Guide & LLM Top 10 | All AI systems | Prompt injection, data leakage, insecure plugins<br>Phased testing: model → implementation → system → runtime |
| **MITRE ATLAS** | ML/AI threat modeling | Data poisoning, model evasion, model theft<br>Comprehensive attack taxonomy |
| **CSA Agentic AI Guide** | Autonomous agents | Agent Control Hijacking, Checker-Out-of-Loop failures<br>Multi-agent security scenarios |
| **Regulatory Guidelines** | Compliance requirements | Bias/fairness testing, industry-specific standards |

**OWASP GenAI Red Teaming Guide and LLM Top 10:** These provide a framework for what to test and how. The OWASP Top 10 for Large Language Model apps highlights common threats (prompt injection, data leakage, insecure plugin use, etc.) which can serve as a starting checklist: are you testing for each of those? The GenAI Red Teaming Guide offers a Blueprint – a phased approach (planning, execution, post-engagement) and phases of evaluation (model, implementation, system, runtime). Use these to structure your program. For example, ensure you do model evaluation (e.g., test the raw model for prompt attacks), implementation testing (test the integrated app around the model), system evaluation (broader system security like auth and network), and runtime analysis (monitoring in a live-like environment).

**MITRE ATLAS & Threat Models:** MITRE's ATLAS is like ATT&CK for AI – it enumerates tactics and techniques for attacking ML. Use such resources in threat modeling to ensure you consider things like data poisoning, model evasion attacks, model theft, etc., if relevant. Not all will apply, but it's comprehensive.

**CSA Agentic AI Guide:** If you have autonomous agents, CSA's guide (and summaries like Adversa's 10 insights) detail categories of risks and how to test them. For instance, test for Agent Control Hijacking by simulating spoofed commands or tokens, or Checker-Out-of-Loop failures by disabling oversight mechanisms and seeing if the agent runs wild. Use these scenario ideas to enrich your playbook with agent-specific tests.

**Regulatory Guidelines:** If applicable, follow any testing requirements from regulators or standards (some might require testing bias or fairness explicitly, for example).

Frameworks ensure you don't forget major areas. One team reported how OWASP's guide helped them cover "security, safety, and trust" perspectives, categorizing risks into those buckets to make sure they weren't solely focused on just security but also user harms and misinformation. That holistic view is critical in AI contexts.

## 2. Establish Safeguards and a Safe Environment

When Red Teaming, especially if you're testing production or a system with live data, take precautions:

### Essential Safety Checklist

**Testing Environment Safety:**

- ☑ Use staging environment replica when possible
- ☑ Point AI to copy of database, not production data
- ☑ Throttle outbound actions (emails, transactions) into sandbox
- ☑ Set up stop conditions for autonomous agent testing
- ☑ Monitor system metrics during testing

**Data and Ethics:**

- ☑ Ensure red team authorized to view any real data used
- ☑ Have NDAs/agreements in place as needed
- ☑ Sanitize or tokenize sensitive data when possible
- ☑ Define ethical limits - no destructive actions on production
- ☑ Keep findings confidential and secure

**Use a Staging Environment** if at all possible. Ideally, have a full replica of the AI system where you can attack freely without risking real user data or uptime. For example, point the AI to a copy of the database, or throttle its outbound actions (so if it tries to send emails or make transactions, those are caught in a sandbox).

**Data Handling:** If using real data in tests (say, you want to see if the AI will leak actual confidential info), ensure the red team is authorized to view that data, and have NDAs or agreements as needed. Alternatively, sanitize or tokenize sensitive data when possible.

**Ethical Limits:** Red teamers should know what not to do, even in a test. For instance, don't actually carry out a destructive action on production systems ("we got the AI to tell us how to delete the database and then we did it" – too far!). Also, be mindful of not exposing any findings or data publicly.

**Stop Conditions:** Autonomous agents could get out of hand. Set up stop conditions or human oversight during testing of agents. E.g., if an agent tries to perform a high-risk action in staging, have a human approve it before letting it proceed, or simulate the effect rather than doing it.

**Monitoring During Tests:** Keep an eye on system metrics. If your attacks inadvertently cause a denial of service or heavy load, you might need to pause. Also monitor the AI's outputs to catch if it does something truly unexpected (the red team might discover an unrelated bug – e.g., an AI deletion of data – you'd want to catch that immediately).

## 3. Document and Share Results with Context

Documenting all results is as important as finding them. For each finding, capture:

- The inputs (prompts, sequences of steps, or data modifications) that led to the issue.
- The output or behavior observed (with screenshots or logs if useful).
- Why it's a problem – the impact or what could happen if a malicious user exploited this.
- Suggested mitigation or fix (if known).

It's vital to include enough detail that developers or a third party can reproduce the issue. If the AI gave a one-off strange response, provide the random seed or the exact model version if possible. Many AI issues can be transient or data-dependent, so reproducibility is gold.

When sharing results, tailor to the audience:

### Audience-Tailored Reporting Strategy

| Audience | What They Want | How to Present |
|---|---|---|
| **Engineering/ Developers** | Technical debugging details | • Exact prompts and system logs<br>• Specific fix recommendations<br>• Reproduction steps with environment details |
| **Management/ CISO** | Business risk assessment | • Executive summary with severity levels<br>• Business impact mapping<br>• Remediation timeline and resource needs |
| **AI Governance/ Risk Teams** | Policy and process implications | • Sanitized findings for organizational learning<br>• Broader security pattern analysis<br>• Process improvement recommendations |

**Engineering/Developers:** Want the nitty-gritty. Provide them the prompts and logs so they can debug. Also provide recommendations – e.g., "Add a check to strip markdown from emails before the AI sees them" or "Update the prompt to explicitly disallow XYZ." Red teaming isn't just about breaking; it's about helping build defenses. Remember that **every prompt and test must be done with the purpose of having a recommendation tied to how to prevent that attack in the future.** Otherwise it's a waste of everyone's time.

**Management/CISO:** Want the big picture and risk assessment. Summarize how many issues found, their severities, and what the plan is to fix. Map them to business impact (e.g., "We found a way to access customer data – this is high risk, potential data breach scenario, will be fixed in 2 weeks").

**Across Teams:** If you have an AI governance or risk committee, share results there too. It might influence policy (like "maybe we should not allow this AI to connect to the internet until we solve these issues"). Also consider sharing sanitized findings within the organization to educate others (for example, show developers examples of prompt injection so they realize why certain secure coding practices are needed).

After documenting, **track the fixes.** Treat these findings as you would vulnerabilities from a pen-test: assign owners, implement changes, verify fixes (perhaps the red team re-tests or automation does), and close the loop by updating any relevant documentation or training.

And don't forget to **celebrate improvements** – if your second red team engagement finds fewer critical issues than the first, that's progress! It means the feedback loop is working. Highlight that to reinforce the value of this whole exercise.

## 4. Integrate AI Red Teaming into Development and Operations

To truly scale and sustain, AI red teaming cannot be a one-off annual exercise; it should be woven into your development and deployment lifecycle:

### Development Lifecycle Integration

| Development Phase | Red Teaming Activities | Practical Examples |
|---|---|---|
| **Design & Planning** | • AI feature threat modeling<br>• Abuse case development alongside use cases | "When adding AI agent plugin, ask 'how could this be abused?' early" |
| **Development (Shift Left)** | • AI security reviews<br>• Developer security training<br>• Secure prompt design guidelines | "Require AI security review for new AI projects, like code security review" |
| **Pre-Deployment** | • Automated attack test suites<br>• Manual security validation<br>• Release gate requirements | "Run known attack patterns, verify nothing egregious emerges before prod" |
| **Production** | • Runtime anomaly detection<br>• User reporting channels<br>• Continuous monitoring | "Flag if chatbot outputs 100KB text or unusual database query patterns" |
| **Post-Incident** | • Model retraining assessment<br>• Policy updates<br>• Process improvements | "If model leaks training phrases, review dataset and model parameters" |

**During Development (Shift Left):** Security and AI teams should be involved in design. Threat model new features involving AI and write abuse cases (just like use cases) that developers consider. For example, when adding a new plugin to an AI agent, ask "how could this be abused?" early and perhaps build in restrictions from the get-go. Some companies have started requiring an "AI security review" for any new AI project, similar to a code security review.

**Pre-Deployment Testing:** Make AI red teaming (even if light) a gate before releasing an AI feature. This could be as simple as running an automated test suite of known attacks and verifying nothing egregious comes out. Or a manual check by a security engineer. It's analogous to running dynamic analysis on a web app before pushing to prod.

**Continuous Monitoring in Production:** While not classical red teaming, having runtime protections is crucial. For instance, implement anomaly detection on AI outputs – if your chatbot suddenly outputs 100KB of text or starts repeating certain patterns that are unusual, flag it. Or if it's connected to a database, monitor for unusual queries triggered by it. Some attacks might slip through testing, so monitoring can catch them. Also provide an easy channel for users or staff to report AI misbehavior they observe, and feed that back to the red team to investigate.

**Feedback Loop to Training:** A unique aspect of AI is that sometimes the fix might involve retraining or fine-tuning the model, not just code changes. If red teaming finds the model is repeatedly biased or leaking a certain training phrase, you might need to adjust your training data or apply reinforcement learning with human feedback (RLHF) to mitigate it. Establish a process with your data science team so that red team findings (e.g., "model often reveals snippet 'XYZ' when asked about ABC") result in a review of the dataset or model parameters.

**Policy and Controls Updates:** Incorporate lessons into policies: e.g., update your prompt design guidelines for developers ("Always include a concluding user instruction that says to refuse sensitive info requests"), or introduce a rule that "AI systems that produce code must run output through a static analyzer before execution" if that addresses a found risk. Also update incident response plans to include AI scenarios (what if someone does successfully exploit the AI – how do we contain and respond?).

## 5. Leverage No-Code/Low-Code Red Teaming Platforms

As mentioned, reaching the highest maturity often means adopting specialized platforms that make AI red teaming easier and scalable. These tools are evolving quickly. Benefits of such platforms (like Mend AI Red Teaming and others) include:

### The Strategic Balance

| Platform Benefits | Potential Trade-offs |
|---|---|
| **Ease of Use:** Simple UI/config setup for non-coders | **Cost:** Platform licensing and subscription fees |
| **Attack Library:** Out-of-box jailbreaks and malicious prompts | **Flexibility:** Less customizable than hand-crafted approaches |
| **Automation:** Scheduled testing and CI/CD integration | **Dependency:** Reliance on vendor updates and support |
| **Collaboration:** Dashboards, reporting, JIRA integration | **Learning Curve:** Team needs to adapt to new tooling |
| **Scalability:** Cloud compute for thousands of test queries | **Validation Required:** Still need human oversight of results |

**Ease of Use:** They offer a user interface or simple config (sometimes YAML or forms) to set up attack scenarios. This means even non-coders (or busy security teams) can run sophisticated AI tests. For example, you might select "Test for Data Leakage" and the platform will automatically try a suite of prompts and techniques.

**Scenarios Library:** Vendors often encode knowledge from many engagements – common jailbreaks, malicious prompts, etc. – so you get a library of attacks out-of-the-box. This saves your team from having to discover every trick themselves or keep up with all new research.

**Automation & Scheduling:** You can schedule tests to run nightly or integrate them into CI/CD (so every new build triggers a test). Some tools can even hook into chat interfaces or APIs continuously.

**Collaboration and Reporting:** These platforms typically provide dashboards, vulnerability tracking, and integration with JIRA or other dev tools, making that feedback loop easier. They may also provide metrics (like "toxicity score" of outputs over time) to track improvement.

**Scaling Compute:** If heavy testing is needed (e.g., generating thousands of queries), a platform might manage the compute power or allow you to run attacks in parallel in the cloud, etc., so you're not limited by one laptop script.

**No-Code for Non-Experts:** Perhaps one of the biggest advantages is empowering domain experts or QA teams to participate in AI red teaming by using a guided interface. For instance, a content moderation team member could use the platform to test if the AI catches certain slurs, without needing to know how to script API calls.

The **trade-off** is cost and sometimes less flexibility than hand-crafted approaches. But if you want to accelerate to Level 3 maturity, such a platform can be invaluable. It encapsulates the best practices and automation that would otherwise take significant time to build in-house.

When using a platform, still **keep humans in the loop** to interpret and follow-up on results. And validate its findings; no tool is perfect (false positives or negatives can occur). Use it to augment your team, not replace careful analysis.

## 6. Continuously Learn and Evolve

AI technology is evolving rapidly, and so are attack techniques. Make sure your red teaming program is not static:

### Continuous Improvement Framework

Quarterly Activities:

- ➡ Review latest AI security research and conference findings
- ➡ Update test scenarios based on new attack techniques
- ➡ Analyze program metrics and identify improvement areas
- ➡ Plan training updates for development teams

Annual Activities:

- ➡ Conduct comprehensive external red team assessment
- ➡ Benchmark program maturity against industry standards
- ➡ Review and update AI security policies and procedures
- ➡ Evaluate new tools and platform capabilities

**Stay Updated:** Follow research blogs, conferences (like DEF CON's AI Village findings, Black Hat talks on AI security, etc.), and communities (the OWASP AI Security group, etc.). New exploits (e.g., prompt injection via hidden unicode characters, or novel model inversion attacks) are regularly discovered. Incorporate relevant ones into your testing.

**Peer Reviews and Exercises:** Consider participating in cross-company red team exercises or info-sharing. Some companies do "red team share and tell" sessions under Chatham House Rule to discuss what they found (anonymized) and how they fixed it. This helps everyone.

**Periodic Full Red Team Events:** Even if you have continuous automated testing, it's wise to do periodic deep-dive red team operations, perhaps annually or when launching a major new AI system. Bring in external experts to get fresh eyes occasionally; they might spot what biased insiders missed.

**Metrics and Improvement:** Track metrics like: number of findings by severity, time to remediate, recurrence of similar issues, etc. If you see, for example, a trend that every red team finds a new prompt injection variant, that might indicate a need for a systemic change (like a better prompt management framework or more training for devs). Use metrics to drive investment (show leadership "we reduced critical AI vulns by X% after implementing automated testing").

# Your AI Security Future Starts Now

The question is not so much “Do we need AI red teaming?” – if you leverage AI in any significant way, the answer is increasingly yes – but rather “How soon and how effectively can we integrate AI red teaming into our security strategy?”

**Key Takeaways for Action:**

- **Start immediately** with manual testing of your highest-risk AI systems
- **Progress systematically** through maturity levels with clear metrics and timelines
- **Leverage automation** to achieve comprehensive coverage and sustainable operations
- **Integrate deeply** with development workflows for maximum effectiveness
- **Measure continuously** to demonstrate value and guide program evolution

The organizations succeeding in AI security combine human expertise with comprehensive automated platforms, creating programs that detect vulnerabilities before attackers while enabling rapid, secure AI innovation.

Your AI Red Teaming journey begins with the first test you conduct and the first vulnerability you discover. The cost of waiting grows daily as AI deployments expand and attack techniques evolve.

**Ready to accelerate your AI security maturity?** Explore Mend AI Red Teaming to discover how comprehensive automated testing can advance your organization from basic manual testing to Level 3 continuous security validation.

The future of AI security depends on the actions you take today. Make them count.