Defensive Adversarial AI

Using Adversarial Testing to Harden Organizational Language Models

Tom Olzak, MBA, CISSP, SecAI+
Online Faculty
University of Phoenix
August 2026

Abstract

Organizations increasingly use large language models (LLMs) for business operations and security. While this improves productivity by helping workers find information, write proposals, and detect and respond to cyber incidents, LLMs also expand an organization's attack surface more quickly. Threat actors can poison prompts and training data. They can manipulate LLMs to infer sensitive information or extract the organization's model. Defensive adversarial AI helps manage this expanded attack surface by intentionally subjecting AI systems to hostile prompts and operating conditions, testing whether the model is robust enough and has sufficient safeguards and guardrails to identify weaknesses before threat actors do. This paper explores how to use open testing frameworks and how to prioritize test results based on the associated risk without unacceptable reduction in model performance. The approach is based on the NIST AI Risk Management Framework, the NIST Generative AI Profile, NIST adversarial machine-learning guidance, NIST secure development guidance, OWASP's LLM risk model, and recent research on prompt injection and jailbreak testing (Autio et al., 2024; Booth et al., 2024; Vassilev et al., 2025). The paper's central theme is that no one filter, model, or benchmark can provide adequate protection. Effective model hardening requires continuous testing of the full AI-enabled system—including data, retrieval, tools, identity, application logic, and human oversight—followed by risk-based remediation and regression testing (Liu et al., 2024a; Olzak, 2026a; OWASP GenAI Security Project, 2024).

Paper Objectives

- Explain defensive adversarial AI and distinguish it from ordinary quality assurance, safety benchmarking, and traditional penetration testing.
- Build a threat-informed test plan for an LLM, retrieval-augmented generation (RAG) system, or agentic AI application.
- Select and apply appropriate manual and automated tools, including Dioptra, PyRIT, garak, and custom test harnesses.
- Prioritize findings by combining exploitability, exposure, business impact, data sensitivity, and the privileges available to the AI system.
- Choose remediation controls at the model, prompt/context, data/RAG, application, tool/agent, and governance layers.
- Verify fixes with controlled regression tests, negative testing, utility measurements, and production monitoring.

1. What Defensive Adversarial AI Is and Why It Matters

Defensive adversarial testing is the deliberate use of attacker-like behavior to discover how an AI system can be made to violate its intended security, privacy, safety, or operational boundaries. The defensive qualifier matters. The purpose is not to create harmful capability for its own sake; it is to learn where a deployed or deployable system can be manipulated and to convert that knowledge into stronger controls. NIST describes adversarial machine learning in terms of deliberate actions by motivated adversaries and organizes attacks by life-cycle stage, attacker knowledge, capability, objective, and consequence. Its current taxonomy includes evasion, poisoning, privacy, and misuse or abuse risks across predictive and generative systems (Autio et al., 2024; Tabassi, 2023; Vassilev et al., 2025).

For LLM applications, the attack surface extends beyond model weights. The model receives instructions from a system prompt, users, retrieved documents, memory, plugins, and application code; it may then return text to a user or initiate actions through tools and APIs. OWASP's 2025 LLM risk model therefore highlights

- Prompt injection
- Sensitive information disclosure
- Data and model poisoning
- Improper output handling
- Excessive agency
- System prompt leakage
- Vector and embedding weaknesses
- Misinformation
- Supply-chain weaknesses

- Unbounded consumption

Recent security research concludes that prompt injection is an application-level security problem because an attacker can inject instructions into the same context the model uses to follow legitimate instructions (Liu et al., 2024a; Olzak, 2026a; OWASP GenAI Security Project, 2024).

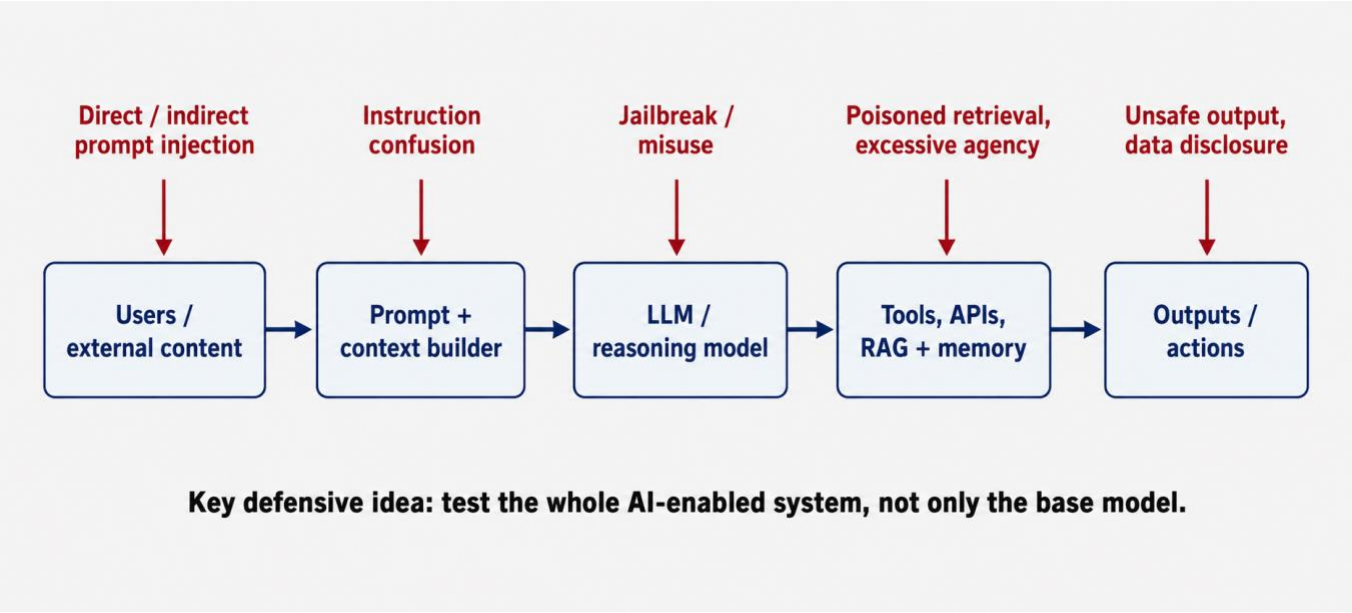

Figure 1. Simplified attack surface of an LLM-integrated application. The critical boundary is the entire system, not only the model.

This distinction changes how cybersecurity teams think about assurance. A conventional software defect is often deterministic: provide the same malformed input to the same vulnerable code path and the result repeats. LLM behavior is probabilistic, context-sensitive, and frequently altered by model updates. A defensive test program therefore needs repeated trials, multiple phrasing strategies, attack variants, and scoring rules. It must also track the exact model version, system prompt, tool configuration, retrieved content, temperature or sampling settings, and application build so that results are reproducible enough to support engineering decisions (Derczynski et al., 2024; Lopez Munoz et al., 2024; Yu et al., 2024).

The business reason for this work is clear. An LLM that only drafts low-risk text has a smaller potential impact than an agent that can read customer records, query production databases, create cloud resources, send email, or approve transactions. The same prompt-injection weakness can therefore have very different risk. NIST's AI RMF and Generative AI Profile emphasize mapping context and impacts before selecting measurements and controls, while secure-development guidance extends this thinking into the development life cycle. The security team should consequently ask not only, "Can the model be tricked?" but also, "What can happen if it is tricked here?" (Autio et al., 2024; Booth et al., 2024; Tabassi, 2023).

Table 1. Major adversarial AI risk categories and defensive test objectives

Risk category	Where it enters	Typical consequence	Defensive test objective
Prompt injection / jailbreak	User input, retrieved content, web pages, documents	Instruction override, policy bypass, unauthorized behavior	Determine whether untrusted language can redirect the system or suppress higher-priority instructions.
Poisoning / backdoor	Training, fine-tuning, embeddings, external knowledge sources	Targeted misbehavior or biased responses triggered by chosen conditions	Test provenance, training/retrieval integrity, trigger behavior, and resilience to corrupted samples.
Privacy / extraction	Queries, outputs, APIs, side channels	Disclosure of training data, secrets, system prompts, or model behavior	Probe for memorization, secret leakage, inference, extraction, and excessive observability.
Excessive agency	Tool calls, plugins, APIs, autonomous loops	Unauthorized transactions, data modification, lateral movement	Test least privilege, authorization boundaries, human approvals, and action allowlists.
Unsafe output handling	Generated code, HTML, SQL, commands, workflow fields	Downstream injection or execution in conventional systems	Verify that model output is treated as untrusted data and validated before use.

2. Planning Defensive Adversarial AI Testing

Good adversarial testing begins with a test charter, not a tool. The charter defines the

- System under test
- Business purpose
- Data classes
- Approved targets
- Prohibited actions
- Environments
- Owners
- Escalation paths
- Evidence requirements

This mirrors mature penetration-testing practice, but AI adds several variables:

- The exact model and version
- The system and developer instructions
- Retrieval sources
- Embeddings
- Memory
- Enabled tools
- Tool credentials
- Maximum autonomy
- Safety filters
- Evaluation model
- Sampling configuration

Planning these elements is necessary because changing any of them can change both attack success and normal utility (Autio et al., 2024; Booth et al., 2024; Lopez Munoz et al., 2024).

2.1 Map the System Before You Attack It

The first planning artifact should be a simple data-and-action flow. Identify every point where untrusted information can enter the AI context and every place where the AI can cause a consequential action. For a RAG application, this includes

- Ingestion pipelines
- Document repositories
- Chunking and embedding services
- Vector databases
- Retrieval filters
- Prompt construction
- The LLM endpoint
- Output parsers
- Downstream tools

For an agent, include service identities, API scopes, approval steps, memory, delegated subagents, and network destinations. This system view aligns with NIST's emphasis on mapping context and with OWASP's treatment of LLM risk as an application and integration problem rather than a model-only problem (Autio et al., 2024; Olzak, 2026a; OWASP GenAI Security Project, 2024).

2.2 Build a Threat Model and Abuse-Case Library

Next, translate the architecture into attacker objectives. Examples include

- Obtaining confidential data
- Changing a model-guided decision
- Causing the application to call an unauthorized tool
- Forcing the model to ignore policy

- Poisoning a retrieval source
- Extracting a hidden prompt
- Increasing cost
- Causing harmful code to reach a downstream interpreter

Each objective should have at least one abuse case and a measurable success condition. NIST's adversarial ML taxonomy provides vocabulary for attacker capability and knowledge, while OWASP provides LLM-specific application risks; combining the two keeps tests both technically rigorous and operationally relevant (Olzak, 2025; OWASP GenAI Security Project, 2024; Vassilev et al., 2025).

2.3 Define Baselines and Success Criteria

A test is useful only when the team can distinguish a pass from a failure. Establish a clean baseline for normal task accuracy, refusal behavior, latency, cost, and tool-call correctness before adversarial testing. Then define security metrics such as attack success rate, unauthorized tool-call rate, sensitive-data disclosure rate, jailbreak rate, poisoned-retrieval success rate, and percentage of adversarial cases detected or blocked. Research frameworks such as garak, PyRIT, and LLM-Fuzzer all emphasize repeatable probing and systematic scoring because one anecdotal jailbreak does not measure the overall exposure of a probabilistic system (Derczynski et al., 2024; Lopez Munoz et al., 2024; Yu et al., 2024).

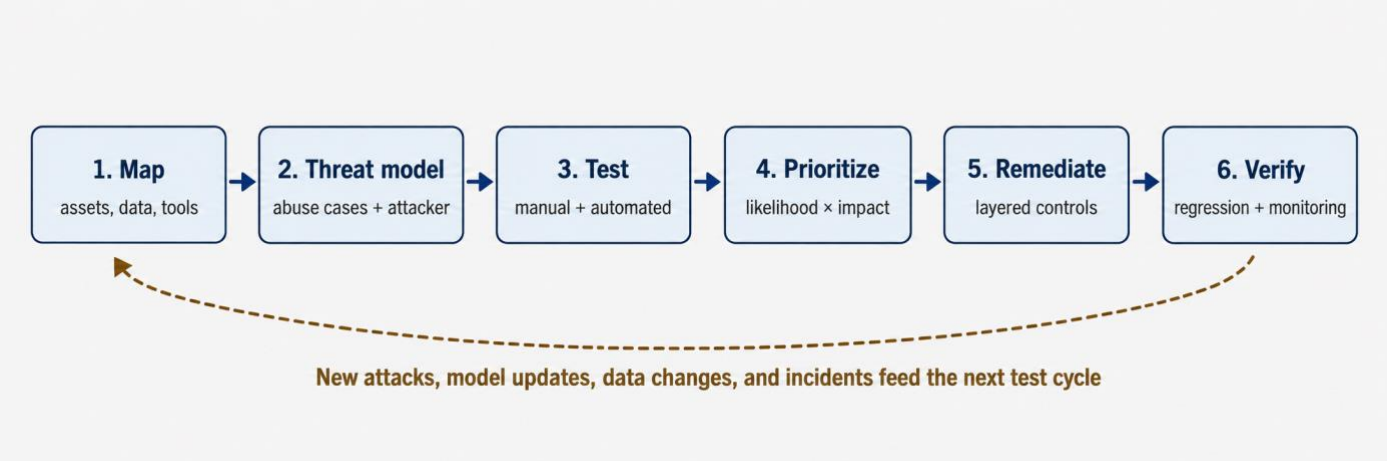

Figure 2. Defensive adversarial AI testing is a continuous assurance cycle rather than a one-time penetration test.

2.4 Rules of Engagement

The rules of engagement should explicitly protect people and production systems. Test accounts should use synthetic or approved data whenever possible. Tool permissions should be constrained so a successful adversarial prompt cannot make an uncontrolled change. If the objective requires production-like privileges, use a sandbox or a transaction simulator and record the intended effect instead of executing the real action. The team should also specify stop conditions, such as unexpected access to real personal data, uncontrolled external communication, high cost, or evidence that a test is affecting other tenants. This is consistent with secure-development and risk-management guidance that expects defined roles, controlled environments, documented changes,

and protection of sensitive information throughout testing (Autio et al., 2024; Booth et al., 2024; Tabassi, 2023).

3. Tools and Related Procedures

No single tool covers the full defensive adversarial AI problem. The practical approach is to combine manual exploratory testing with automation. Humans are better at understanding business context, chaining observations, and recognizing subtle policy failures. Automation is better at repetition, mutation, broad probe coverage, and regression. Microsoft's experience red teaming many generative AI products similarly emphasizes that automation expands coverage but does not replace human judgment; NIST and academic tooling likewise focus on reproducible experiments rather than a universal pass/fail scanner (Derczynski et al., 2024; Lopez Munoz et al., 2024; Olzak, 2026a).

Table 2. Representative defensive adversarial AI tools

Tool	Best fit	What it contributes	Practical procedure
NIST Dioptra	Predictive AI and controlled ML experiments	Reproducible, trackable workflows for attacks, defenses, datasets, models, and metrics	Create a versioned experiment; run clean baseline; apply attack/defense combinations; compare metrics; preserve artifacts for repeatability.
Microsoft PyRIT	Generative AI red teaming	Orchestration, attack strategies, converters, targets, memory, scorers, and multi-turn testing	Define target and objective; select attack strategy; configure scorers; run controlled campaigns; review transcripts and scored outcomes.
NVIDIA garak	Broad LLM vulnerability probing	Large probe library for jailbreaks, injection, leakage, hallucination, toxicity, and other failure modes	Scan a defined model/application interface; select relevant probes; capture JSONL results; triage failures; convert confirmed failures into regression cases.
Custom harness + human red team	Business-specific workflows and agent actions	High-context tests, domain abuse cases, authorization checks, and novel attack chains	Script expected workflow; inject adversarial variants at each trust boundary; inspect model reasoning artifacts only where permitted; verify actual downstream authorization.

3.1 A Repeatable Test Procedure

1. **Freeze the test configuration**. Record model identifier, prompt templates, guardrail versions, retrieval corpus snapshot, tool permissions, application build, and sampling parameters.
2. **Run the benign baseline**. Measure normal task quality, refusal behavior, response time, cost, and correct tool use.
3. **Run known attack classes**. Include direct and indirect injection, jailbreak variants, data leakage probes, poisoned retrieval, malformed outputs, tool-abuse attempts, and resource-consumption cases appropriate to the system.
4. **Mutate attacks**. Change language, encoding, role framing, multi-turn structure, document placement, context length, and ordering. Probabilistic systems should be tested across multiple trials.
5. **Score and confirm**. Automated scorers accelerate review, but high-risk findings should be manually confirmed to reduce false positives and evaluator-model bias.
6. **Preserve evidence**. Store the prompt, relevant retrieved content, model response, tool calls, logs, model/version metadata, scorer output, and reproduction steps.
7. **Convert confirmed failures to regression tests**. Every material vulnerability should become a repeatable test that can run after remediation and during future releases.

A useful test program also separates discovery from verification. During discovery, testers should encourage variation and experimentation because unusual failure modes matter. During verification, inputs, configurations, and expected outcomes should be tightly controlled so the team can compare before-and-after results. Recent prompt-injection benchmarking research shows why this distinction matters; defenses can perform very differently across attacks, tasks, and models, so a claim that a control “blocks prompt injection” is meaningful only relative to a defined test set and operating context (Liu et al., 2024a; Vassilev et al., 2025; Yu et al., 2024).

4. Prioritizing Findings Based on Risk

A jailbreak is not automatically a critical vulnerability. Risk depends on what the system protects and what the attacker gains. The same bypass may be low risk in an isolated model and critical in an enterprise agent that can retrieve regulated records and make authenticated changes. NIST’s risk-management approach therefore starts with context, affected stakeholders, and impact rather than a universal technical severity. In conventional vulnerability management, Olzak (2025) similarly emphasizes separating the probability of exploitation from the severity of impact; for AI findings, the same reasoning can be adapted without treating CVSS or EPSS as direct LLM scoring systems (Autio et al., 2024; Olzak, 2025; Tabassi, 2023).

4.1 A Practical AI Finding Score

A simple organizational scoring model can use five 1-to-5 factors: exploitability, exposure, impact, data sensitivity, and agent privilege.

- Exploitability asks how reliably the attack succeeds and how much expertise it requires.
- Exposure asks who can reach the vulnerable interface or poison the relevant data source.

- Impact measures mission harm to confidentiality, integrity, availability, safety, finance, or compliance.
- Data sensitivity reflects the classification of data available to the application.
- Agent privilege captures what the AI can do through tools and identities.

Teams can weight the factors to match their mission, but the scoring rubric should be written before triage begins so similar findings receive similar treatment (Autio et al., 2024; Olzak, 2025; Vassilev et al., 2025).

Table 3. Example risk prioritization for three AI security findings

Finding	Exploitability	Exposure	Impact	Data / privilege	Priority
Indirect prompt injection causes a public FAQ bot to mention an attacker-selected URL	4	5	2	Low / none	Moderate
Prompt injection causes an HR assistant to reveal restricted employee data from RAG	4	3	5	High / read	Critical
Agent follows injected instructions to issue an unapproved production configuration change	3	2	5	High / write	Critical

Two additional modifiers deserve attention. First, repeatability matters. A finding that succeeds in 1 of 100 trials may still be serious if the attacker can cheaply automate thousands of attempts, but it is different from a 95% success condition. Second, detection matters. A high-impact attack that produces a clear, immediate alert and requires approval before action is less likely to create loss than the same attack occurring silently. The final priority should therefore be a reasoned risk decision supported by evidence, not merely the output of a formula (Autio et al., 2024; Liu et al., 2024a; Olzak, 2026a).

5. Approaches to Remediate the Risk

The most important remediation principle is to fix the weakness at the correct layer. Prompt filtering can be useful, but it cannot compensate for an agent that has unnecessary administrative permissions. Model fine-tuning can improve refusal behavior, but it cannot make untrusted generated SQL safe to execute. Data cleaning can reduce poisoning risk, but it cannot prevent a web-connected RAG system from ingesting a newly compromised page. NIST, OWASP, and recent security research consistently point toward layered controls because individual defenses remain incomplete and attacks adapt (Booth et al., 2024; Liu et al., 2024a; Vassilev et al., 2025).

Figure 3. Layered remediation model. Strong AI security places controls around the model as well as inside it.

5.1 Model-Level Controls

Model-level controls include safety fine-tuning, adversarial training, preference optimization, refusal tuning, model editing, and, in predictive ML, robust training and certified defenses. These controls can reduce susceptibility to known attacks, but they should not be presented as proof of immunity. Jailbreak research continues to demonstrate that safety-aligned models can be manipulated through adversarial phrasing, multi-turn decomposition, or automated mutation. Use model controls to raise attacker cost and reduce attack success, then assume that some malicious inputs will still reach the next layer (Liu et al., 2024b; Vassilev et al., 2025; Yu et al., 2024).

5.2 Prompt and Context Controls

At the context layer, clearly separate trusted instructions from untrusted data, minimize unnecessary context, label retrieved content as data rather than instructions, and constrain the model to a narrow task. Input classifiers and prompt-injection detectors can provide another signal, but they should not be the only control because an adaptive attacker can alter wording, language, encoding, or placement. System prompts should also avoid embedding secrets; if a secret must remain secret, it belongs in an authorization system, not in text that the model receives (Liu et al., 2024a; Olzak, 2026a; OWASP GenAI Security Project, 2024).

5.3 RAG and Data Controls

RAG systems need conventional data security plus AI-specific provenance controls.

- Restrict who can add or modify source documents
- Preserve provenance metadata
- Validate content before indexing
- Apply access control before retrieval

- Prevent the model from receiving documents the user is not authorized to read
- Treat external web content as hostile

For high-value knowledge bases, monitor sudden embedding or document changes and maintain a known-good corpus for testing. These steps reduce both poisoning and indirect prompt-injection risk, and they align with secure-development, OWASP, and NIST generative-AI guidance (Autio et al., 2024; Booth et al., 2024; OWASP GenAI Security Project, 2024).

5.4 Agent and Tool Controls

Agentic systems demand the strongest authorization discipline because model output may become action. Give the agent the minimum permissions required for its task; separate read and write identities; use allowlisted tools and destinations; enforce parameter validation outside the model; and require human approval for high-consequence actions. A model should be able to propose an action without necessarily being authorized to execute it, keeping a human in the loop to verify that the action is needed and authorized. This separation reduces the blast radius when prompt injection or jailbreak succeeds (Autio et al., 2024; Olzak, 2026a; OWASP GenAI Security Project, 2024).

5.5 Application and Output Controls

LLM output is untrusted data. If generated text is inserted into HTML, SQL, shell commands, source code, email, or workflow fields, the receiving system must validate and encode it according to conventional secure-software rules. Structured output schemas, allowlisted values, parameterized APIs, escaping, sandboxed execution, and secondary policy checks reduce the chance that a successful model manipulation becomes a traditional injection vulnerability. NIST's SSDF profile explicitly brings generative AI into secure software-development practices, while OWASP treats improper output handling as a distinct LLM application risk (Booth et al., 2024; Olzak, 2024; OWASP GenAI Security Project, 2024).

5.6 Governance and Operational Controls

Some failures cannot be solved with a single technical patch. Organizations need

- Ownership
- Approved use cases
- Change control
- Incident response
- Logging
- Model and prompt inventories
- Third-party review
- Data classification
- Periodic reevaluation

Model providers can change behavior without an application-code release; retrieval sources change daily; and new attack techniques appear continuously. Governance therefore establishes when a

system must be retested: for example, after a model upgrade, tool-permission change, new data source, prompt redesign, major incident, or material change in business purpose (Autio et al., 2024; Booth et al., 2024; Tabassi, 2023).

6. Ensuring the Changes Work as Expected

Remediation is not complete when the code change is merged. It is complete when the organization has evidence that the risk was reduced without creating unacceptable new failure modes. The first step is exact replay: rerun the original exploit with the same configuration and enough trials to confirm that the success rate has materially changed.

The second step is adversarial variation; mutate the attack so the team does not merely prove that one string was blocked. The third step is utility regression; compare normal task quality, latency, cost, refusal rate, and correct tool behavior against the pre-change baseline. Security controls that prevent legitimate work may simply move risk elsewhere (Derczynski et al., 2024; Liu et al., 2024a; Lopez Munoz et al., 2024).

6.1 Verification Matrix

Table 4. Minimum verification evidence after remediation

Verification area	Question	Evidence
Original attack	Does the confirmed exploit still work?	Before/after attack success rate, transcripts, tool logs, and model/version metadata.
Attack variants	Can equivalent phrasing or a changed delivery channel bypass the fix?	Mutated prompts, indirect-document cases, multilingual/encoding variants, multi-turn cases.
Normal utility	Did the fix reduce legitimate performance?	Task-quality benchmark, false-refusal rate, latency, cost, correct tool-call rate.
Authorization	Can a compromised model exceed the intended blast radius?	Negative authorization tests, denied tool calls, approval logs, identity scopes.
Monitoring	Would the organization know if the attack reappeared?	Alert test, security telemetry, retained evidence, escalation workflow.
Change resilience	Will future updates trigger reevaluation?	Regression suite in CI/CD or release gates, model/prompt/data version tracking.

For probabilistic tests, retain distributions rather than a single pass/fail result. Suppose a prompt-injection attack succeeded in 38 of 50 trials before remediation and 2 of 50 after remediation. That is stronger evidence than “the prompt was blocked once”, although the residual 4% success rate may still be unacceptable for a high-impact agent. Establish thresholds that reflect system consequence:

a public information bot may tolerate more residual behavioral variance than a model that can expose health records or change infrastructure (Autio et al., 2024; Liu et al., 2024a; Tabassi, 2023).

Production monitoring closes the loop. Log security-relevant inputs and outcomes in a privacy-conscious manner, including

- Tool denials
- Policy-filter events
- Suspicious retrieval sources
- Repeated jailbreak patterns
- Unusual token consumption
- Authorization failures

A newly observed production attack should become a sanitized test case and feed the next red-team cycle. This converts incidents and near misses into organizational memory, which is essential because AI security is a moving target rather than a one-time certification state (Derczynski et al., 2024; Olzak, 2026a; Vassilev et al., 2025).

7. Applied and Real-World Examples

Example 1: Prompt Injection Against LLM-Integrated Applications

Prompt injection is not merely a laboratory concern. Liu et al. (2023) evaluated a black-box prompt-injection technique against 36 real LLM-integrated applications and reported that 31 were susceptible; multiple vendors validated the reported findings. For an enterprise RAG system, the defensive analog is to place controlled adversarial instructions in documents or web content that the application may retrieve, then measure whether the model follows those instructions, exposes unauthorized information, or causes a downstream action. The key lesson is that hostile language can arrive indirectly through data that the user never typed (Liu et al., 2023; Liu et al., 2024a; OWASP GenAI Security Project, 2024).

A durable remediation combines provenance and write controls on knowledge sources, retrieval-time access control, explicit separation of instructions from untrusted content, detector signals, and strict authorization on tools. Verification should replay the original injected documents and new variants rather than merely block one known string. If the model still occasionally follows malicious text but authorization prevents disclosure or action, the business impact is materially reduced (Autio et al., 2024; Booth et al., 2024; Olzak, 2026a).

Example 2: Red Teaming Generative AI at Product Scale

Microsoft researchers summarized lessons from red teaming more than 100 generative AI products and emphasized a pattern also reflected in PyRIT and garak; automation expands coverage, but human testers remain essential for understanding context, chaining weaknesses, and deciding what constitutes meaningful harm. In practice, a security team can use a human-discovered jailbreak or tool-abuse case as a seed, automate mutations across phrasing and multi-turn structures, score the

results, and then retain confirmed failures as regression tests (Bullwinkel et al., 2025; Derczynski et al., 2024; Lopez Munoz et al., 2024).

Risk still depends on deployment context. A policy bypass in a public support bot with no private data or tools is different from the same bypass in an agent that can read regulated records or make authenticated changes. Remediation may therefore combine model alignment, filters, rate controls, human escalation, and, most importantly for agentic systems, least privilege and external authorization. Verification should measure both adversarial success and false refusals on legitimate work (Autio et al., 2024; Olzak, 2026a; Vassilev et al., 2025).

Example 3: Backdoor Risk in Customized LLMs

USENIX Security research has demonstrated instruction backdoors against customized LLM applications, while NIST classifies poisoning as a core adversarial risk. A defensive test can introduce controlled poisoned examples into a lab copy, evaluate trigger-dependent behavior, and compare clean-task utility before and after poisoning. The corresponding controls are provenance, protected training pipelines, review of third-party models and datasets, isolated evaluation before promotion, and regression tests for known triggers (Booth et al., 2024; Vassilev et al., 2025; Zhang et al., 2024).

8. Building a Sustainable Defensive Adversarial AI Program

A mature program can start small.

1. Select one high-value AI application and create a threat model, a 20-to-50 case adversarial regression suite, a baseline utility benchmark, and a defined finding rubric.
2. Run manual exploration quarterly or around material changes and automate the stable regression cases in the release process.
3. Track the ratio of confirmed findings to false positives, time to reproduce, time to remediate, residual attack success after fixes, and whether new releases reintroduce closed weaknesses.

These measures turn red teaming from a demonstration into an assurance process (Autio et al., 2024; Derczynski et al., 2024; Lopez Munoz et al., 2024).

The program should also retain a human-centered operating model. Security engineers understand exploit chains and authorization boundaries; ML engineers understand model behavior and evaluation; application developers understand integration logic; data owners understand sensitivity and provenance; risk and privacy teams understand business consequence; and system owners decide whether residual risk is acceptable. My recent work on cybersecurity roles under AI adoption similarly argues that accountability, adversarial reasoning, governance, and secure system design become more, not less, important as AI automates routine technical tasks (Autio et al., 2024; Olzak, 2026b; Tabassi, 2023).

Finally, leadership should expect residual risk. NIST explicitly cautions that current adversarial defenses are incomplete, and empirical research repeatedly shows that defenses effective against one class of prompt or model may fail against another. The defensible objective is therefore not "make the model impossible to jailbreak." It is to reduce the probability of successful manipulation, limit the impact when manipulation occurs, detect abnormal behavior quickly, and maintain enough evidence to improve after change (Liu et al., 2024a; OWASP GenAI Security Project, 2024; Vassilev et al., 2025).

Conclusion

Defensive adversarial AI applies a familiar cybersecurity principle to a new class of systems: assume motivated adversaries will search for paths that designers did not intend, and search for those paths first. The work begins by mapping the complete AI-enabled system and its business consequences. It continues through threat-informed manual and automated testing, evidence-based risk prioritization, layered remediation, and disciplined verification. The strongest programs avoid two traps: treating the base model as the entire security boundary and treating a single successful defense test as proof of safety. NIST guidance, OWASP's application risks, and recent security research instead support continuous, system-level assurance in which models, data, prompts, retrieval, software, tools, identities, and people are tested together (Autio et al., 2024; Booth et al., 2024; Vassilev et al., 2025).

For organizations deploying LLMs and agents, the practical standard is resilience. A resilient AI system may still receive a successful adversarial prompt, but least privilege prevents a dangerous action; retrieval controls prevent unauthorized data from entering context; output validation blocks downstream execution; monitoring creates an alert; and regression tests ensure the incident becomes a permanent test case. That layered outcome is what adversarial testing is intended to produce: not the absence of attack, but a system that continues to protect the mission when attack occurs (Olzak, 2026a; OWASP GenAI Security Project, 2024; Tabassi, 2023).

References

AI Usage Disclosure: This document was created with the assistance of AI-enabled tools. ChatGPT was used to support background research and idea generation. Grammarly was used for formatting support, content rewrites, and to help ensure the final content is clear, consistent, and grammatically correct. All final decisions about content, interpretation, and presentation were made by the author.

Autio, C., Schwartz, R., Dunietz, J., Jain, S., Stanley, M., Tabassi, E., Hall, P., & Roberts, K. (2024). Artificial intelligence risk management framework: Generative artificial intelligence profile (NIST AI 600-1). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.AI.600-1

Booth, H., Souppaya, M., Vassilev, A., Ogata, M., Stanley, M., & Scarfone, K. (2024). Secure software development practices for generative AI and dual-use foundation models: An SSDF community profile (NIST SP 800-218A). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-218A

Bullwinkel, B., Minnich, A., Chawla, S., Lopez, G., Pouliot, M., Maxwell, W., de Gruyter, J., Pratt, K., Qi, S., Chikanov, N., Lutz, R., Dheekonda, R. S. R., Jagdagdorj, B.-E., Kim, E., Song, J., Hines, K., Jones, D., Severi, G., Lundeen, R., Vaughan, S., Westerhoff, V., Bryan, P., Siva Kumar, R. S., Zunger, Y., Kawaguchi, C., & Russinovich, M. (2025). Lessons from red teaming 100 generative AI products. arXiv. https://arxiv.org/abs/2501.07238

Derczynski, L., Galinkin, E., Martin, J., Majumdar, S., & Inie, N. (2024). garak: A framework for security probing large language models. arXiv. https://doi.org/10.48550/arXiv.2406.11036

Liu, Y., Deng, G., Li, Y., Wang, K., Wang, Z., Wang, X., Zhang, T., Liu, Y., Wang, H., Zheng, Y., & Liu, Y. (2023). Prompt injection attack against LLM-integrated applications. arXiv. https://arxiv.org/abs/2306.05499

Liu, T., Zhang, Y., Zhao, Z., Dong, Y., Meng, G., & Chen, K. (2024b). Making them ask and answer: Jailbreaking large language models in few queries via disguise and reconstruction. In 33rd USENIX Security Symposium (pp. 4711–4728). USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/liu-tong

Liu, Y., Jia, Y., Geng, R., Jia, J., & Gong, N. Z. (2024a). Formalizing and benchmarking prompt injection attacks and defenses. In 33rd USENIX Security Symposium (pp. 1831–1847). USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/liu-yupei

Lopez Munoz, G. D., Minnich, A. J., Lutz, R., Lundeen, R., Dheekonda, R. S. R., Chikanov, N., Jagdagdorj, B.-E., Pouliot, M., Chawla, S., Maxwell, W., Bullwinkel, B., Pratt, K., de Gruyter, J., Siska, C., Bryan, P., Westerhoff, T., Kawaguchi, C., Seifert, C., Siva Kumar, R. S., & Zunger, Y. (2024). PyRIT: A framework for security risk identification and red teaming in generative AI systems. arXiv. https://arxiv.org/abs/2410.02828

National Institute of Standards and Technology. (2024). Dioptra: Software test platform for assessing trustworthy characteristics of AI models. https://pages.nist.gov/dioptra/

Olzak, T. (2024). Injection attacks. LinkedIn. https://www.linkedin.com/pulse/injection-attacks-tom-olzak-gzmpc

Olzak, T. (2025). Cybersecurity risk analysis and management. ResearchGate. https://www.researchgate.net/publication/389652223_Cybersecurity_Risk_Analysis_and_Management

Olzak, T. (2026a). Guide for securing generative AI applications against the OWASP Top 10 for LLM Applications (2025): Threats, indicators, defense, and response. ResearchGate. https://www.researchgate.net/publication/405638802_Guide_for_Securing_Generative_AI_Applications_Against_the_OWASP_Top_10_for_LLM_Applications_2025_Threats_Indicators_Defense_and_Response

Olzak, T. (2026b). The impact of AI on the cybersecurity professions: Training, available roles, secure use of AI-enhanced safeguards, and role obsolescence. ResearchGate. https://www.researchgate.net/publication/407441824_The_Impact_of_AI_on_the_Cybersecurity_Professions_The_Impact_of_AI_on_the_Cybersecurity_Professions_Training_Available_Roles_Secure_Use_of_AI-Enhanced_Safeguards_and_Role_Obsolescence

OWASP GenAI Security Project. (2024). OWASP Top 10 for LLM applications 2025. https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/

Tabassi, E. (2023). Artificial intelligence risk management framework (AI RMF 1.0) (NIST AI 100-1). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.AI.100-1

Vassilev, A., Oprea, A., Fordyce, A., Anderson, H., Davies, X., & Hamin, M. (2025). Adversarial machine learning: A taxonomy and terminology of attacks and mitigations (NIST AI 100-2e2025). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.AI.100-2e2025

Yu, J., Lin, X., Yu, Z., & Xing, X. (2024). LLM-Fuzzer: Scaling assessment of large language model jailbreaks. In 33rd USENIX Security Symposium (pp. 4657–4674). USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/yu-jiahao

Zhang, R., Li, H., Wen, R., Jiang, W., Zhang, Y., Backes, M., Shen, Y., & Zhang, Y. (2024). Instruction backdoor attacks against customized LLMs. In 33rd USENIX Security Symposium (pp. 1849–1866). USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/zhang-rui