radware

Guide

How to Guard Against the OWASP LLM Top Ten

Table of Contents

Introduction

Large language models (LLMs) follow instructions without concern about the consequences your organization will face. Learn about the latest risks involved in LLMs, how savvy hackers exploit agentic AI use and the best ways to set up guardrails to keep your organization safe.

LLM01: 2025 Prompt Injection

Attackers can manipulate prompts to override built-in safety rules and force the model to take unintended actions. How does this happen? Large language model (LLM) prompts are built to accept open-ended input and create a natural, conversation-like experience for users. Because they are optimized to satisfy user requests, they often attempt to carry out any instruction they are given—whether appropriate, safe or accurate.

Risks:

Data leaks, privilege escalation, malicious outputs

How to Mitigate

Preventing prompt injections requires an in-depth defensive approach that treats the LLM as an untrusted component. Start by constraining model behavior through detailed system prompts that define clear boundaries and ignore modification attempts. Implement strict input/output filtering with semantic analysis and the RAG Triad framework to catch malicious content. Enforce least privilege access by managing API tokens in code rather than exposing them to the model, and require human approval for high-risk actions. Segregate external content clearly, and conduct regular adversarial testing to ensure your trust boundaries and access controls actually hold up under attack.

Key Approaches:

Constrain model behavior, validate output formats, implement semantic filtering, enforce privilege control, require human approval for high-risk actions, conduct regular penetration testing

LLM02: 2025 Sensitive Information Disclosure

LLMs can unintentionally expose private data like personal identifiable information (PII), business secrets or proprietary algorithms. This can result in unauthorized data access, privacy violations, and intellectual property breaches. Plus, exposure of personal data violates privacy laws (e.g., GDPR), causing legal and reputational risks.

Risks:

Privacy violations, IP theft, compliance breaches

How to Mitigate

Preventing sensitive data leakage demands robust sanitization and strict access controls throughout your LLM pipeline. Implement data sanitization techniques to scrub sensitive content before it enters training models, and apply rigorous input validation to catch harmful inputs early. Use federated learning and differential privacy to minimize centralized data exposure, while educating users about safe interaction practices. You can secure your system configuration by concealing system preambles and following OWASP API security guidelines. Finally, consider advanced techniques like homomorphic encryption for processing sensitive data and tokenization with pattern matching to detect and redact confidential information before the model ever sees it.

Key Approaches:

Integrate data sanitization, enforce strict access controls, utilize federated learning and differential privacy, educate users, implement tokenization and redaction, use secure system configuration

LLM03: 2025 Supply Chain

The complex process of creating LLMs often depends on third-party technology. Vulnerabilities in these third-party tools, plugins or model components can open the door to threats including biased LLM outputs, security breaches or system failures. The emergence of open-access and on-device LLMs only increases the attack surface and risk of attack.

Risks:

Malware injection, dependency hijacking

How to Mitigate

Supply chain security for LLMs requires rigorous vetting and continuous monitoring of every external component. Carefully evaluate all data sources and suppliers, including their terms and privacy policies, and regularly audit their security posture. Apply vulnerability scanning and patching from OWASP's A06:2021 guidance while conducting comprehensive AI red teaming on third-party models. Maintain an up-to-date Software Bill of Materials (SBOM) using tools like OWASP CycloneDX to track components and detect vulnerabilities. Implement code signing, integrity checks with file hashes, and strict monitoring for collaborative development environments. For edge deployments, encrypt models with integrity checks and vendor attestation APIs.

Key Approaches:

Vet data sources rigorously, conduct AI red teaming, maintain SBOM inventory, implement code signing and integrity checks, monitor collaborative environments, encrypt edge-deployed models

LLM04: 2025 Data and Model Poisoning

Data poisoning occurs when the pre-training, fine-tuning or embedding of data is manipulated to introduce vulnerabilities, backdoors or biases. Malicious data or poisoned models can skew outputs or embed harmful behaviors. This means degraded model performance, biased or toxic content, and exploitation of downstream systems.

Risks:

Corrupted decision-making, reputational damage

How to Mitigate

Protecting against data poisoning requires comprehensive provenance tracking and multi-layered validation. Use tools like OWASP CycloneDX or ML-BOM to track data origins and transformations throughout model development. Vet data vendors rigorously and validate outputs against trusted sources to detect poisoning signs. Implement strict sandboxing with anomaly detection to filter adversarial data, and tailor models using specific, verified datasets for fine-tuning. You can use data version control (DVC) to track dataset changes and store user-supplied information in vector databases for adjustments without full retraining. Conduct red team campaigns, monitor training loss for anomalies, and integrate Retrieval-Augmented Generation during inference to reduce hallucination risks.

Key Approaches:

Track data provenance, vet vendors and validate outputs, implement sandboxing and anomaly detection, use data version control, conduct adversarial testing, integrate RAG techniques

LLM05: 2025 Improper Output Handling

Improper output handling occurs when there is insufficient validation, sanitization and handling of the output generated by LLMs before it's passed to other components and systems. Since LLM-generated content can be controlled by prompt input, this behavior is like providing users with indirect access to additional functionality. Mishandling generated content can lead to harmful or misleading outputs reaching users.

Risks:

Phishing, misinformation, compliance violations

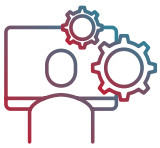
How to Mitigate

Preventing improper output handling means treating LLM responses with the same scrutiny as untrusted user input. Adopt a zero-trust approach by applying proper input validation on all model responses before passing them to backend functions, following OWASP ASVS guidelines. Implement context-aware output encoding based on where outputs will be used: HTML encoding for web content, SQL escaping for databases, etc. Use parameterized queries exclusively for database operations and employ strict content security policies to mitigate XSS risks. Establish robust logging and monitoring to detect unusual output patterns indicating exploitation attempts, and implement rate limiting to slow potential attacks while detection systems respond.

Key Approaches:

Apply zero-trust validation, implement context-aware output encoding, use parameterized queries, employ Content Security Policies, establish logging and monitoring, implement rate limiting

LLM06: 2025 Excessive Agency



Developers grant a certain amount of freedom to their LLM-based system. They can call functions or interface with other systems to fulfill a request from a prompt. Excessive agency happens when damaging actions occur in response to unexpected, ambiguous or manipulated outputs from an LLM.

Risks:

Unauthorized transactions, system changes, cascading failures

How to Mitigate

Controlling excessive agency requires carefully limiting what LLMs can do and ensuring human oversight for critical actions. Minimize available extensions to only what's essential, and limit functionality within those extensions to prevent scope creep. Avoid open-ended extensions like "run shell command" in favor of specific, granular functions. Restrict extension permissions to the minimum necessary using proper database permissions and identity management. Track user authorization to execute actions in the context of specific users with minimal privileges, and implement human-in-the-loop controls for high-impact operations. Apply complete mediation by enforcing authorization in downstream systems and follow secure coding practices with regular SAST and DAST testing.

Key Approaches:

Minimize extensions and functionality, avoid open-ended extensions, restrict permissions to minimum necessary, execute in user context, require human approval for high-impact actions, enforce complete mediation

LLM07: 2025 System Prompt Leakage

System prompt leakage happens when the system prompts or LLM instructions use to direct the model also contain sensitive information not intended to go public. System prompts are designed to guide the model's output, but sometime also contain secret information that can be discovered, reverse-engineered and used in other attacks.

Risks:

Reveal of sensitive information, internal rules, filtering criteria or roles and permissions

How to Mitigate

Preventing system prompt leakage starts with recognizing that prompts aren't secrets and shouldn't contain sensitive information. Separate all sensitive data like API keys, credentials, and database names from system prompts, externalizing them to systems the model doesn't directly access. Avoid relying on system prompts for strict behavior control since they're vulnerable to prompt injection—instead, implement guardrails outside the LLM itself to inspect outputs independently. Ensure security controls like privilege separation and authorization are enforced deterministically in external systems, not delegated to the LLM. When agents need different access levels, use multiple agents each configured with least privileges rather than one overprivileged agent.

Key Approaches:

Separate sensitive data from prompts, implement external guardrails, enforce security controls independently from LLM, use multiple least-privileged agents, avoid behavior control through prompts alone

LLM08: 2025 Vector and Embedding Weaknesses

Vectors and embeddings in retrieval augmented generation (RAG)-based LLMs can create major security gaps. Attackers exploit weaknesses in the ways data is processed, stored or retrieved to inject malicious content, manipulate AI response or steal sensitive information. These vulnerabilities threaten both data integrity and system reliability in LLM applications.

Risks:

Poisoned embeddings, data exfiltration

How to Mitigate

Securing RAG systems requires fine-grained controls and rigorous data validation throughout your vector pipeline. Implement permission-aware vector databases with strict logical partitioning to prevent cross-context leakage in multi-tenant environments. Establish robust data validation pipelines that accept only verified sources and regularly audit for hidden codes and poisoning. Thoroughly review combined datasets from different sources, tagging and classifying data to control access levels. Maintain detailed immutable logs of retrieval activities to detect suspicious behavior, and validate all input documents before adding them to your knowledge base. Monitor how RAG affects foundational model behavior to ensure qualities like empathy aren't inadvertently diminished.

Key Approaches:

Implement permission-aware vector stores, validate data sources rigorously, review and classify combined datasets, maintain immutable logs, monitor behavioral changes from RAG implementation

LLM09: 2025 Misinformation

LLMs create critical misinformation risks that threaten security, reputation and legal compliance. The models produce false but credible-sounding content through hallucinations or fabricated responses based on statistical patterns rather than factual knowledge. Biased training data and information gaps also compromise accuracy. Organizations face significant exposure when deploying LLM applications without proper safeguards.

Risks:

Reputational harm, regulatory issues

How to Mitigate

Combating misinformation requires technical controls combined with user education and transparency. Implement Retrieval-Augmented Generation to ground responses in verified external sources, and enhance models through fine-tuning with techniques like parameter-efficient tuning. Encourage cross-verification of outputs with trusted sources and implement human oversight with properly trained reviewers to avoid overreliance. Deploy automatic validation mechanisms for high-stakes outputs and clearly communicate limitations to users. Establish secure coding practices to prevent integration of vulnerable suggestions, and design user interfaces that encourage responsible use through content filters and AI-generated content labels. Provide comprehensive training on LLM limitations and the critical importance of independent verification.

Key Approaches:

Use Retrieval-Augmented Generation, implement cross-verification and human oversight, deploy automatic validation, communicate risks clearly, establish secure coding practices, educate users on limitations

LLM10: 2025 Unbounded Consumption

Uncontrolled resource use by LLMs can lead to outages and skyrocketing costs. Unbounded consumption attacks exploit LLM systems by triggering excessive, uncontrolled interfaces that drain computational resources. Attackers leverage this vulnerability to launch DDoS attacks, inflate operational costs, steal model behavior or degrade service performance.

Risks:

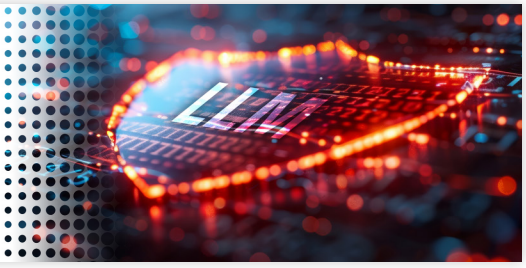
Denial of service, budget overruns

How to Mitigate

Managing unbounded consumption requires comprehensive resource controls and monitoring across your entire LLM infrastructure. Implement strict input validation with size limits and apply rate limiting with user quotas to prevent resource exhaustion. Monitor resource allocation dynamically and set timeouts for resource-intensive operations. Use sandboxing to restrict LLM access to network resources and internal services—this is critical for preventing both external attacks and insider threats. Deploy comprehensive logging with anomaly detection to identify unusual consumption patterns, and implement watermarking to detect unauthorized model use. Design for graceful degradation under load, limit queued actions, and use adversarial robustness training to detect extraction attempts and malicious queries.

Key Approaches:

Implement input validation and rate limiting, monitor resource usage dynamically, use sandboxing techniques, deploy anomaly detection, implement watermarking, design for graceful degradation, train for adversarial robustness



RWI6252 I 11.18.2025