OWASP Top 10 for LLM Applications

VERSION 1.1

Published: *October 16, 2023*

HTTPS://LLMTOP10.COM

OWASP Top 10 for LLM Applications

LLM01

Prompt Injection

This manipulates a large language model (LLM) through crafty inputs, causing unintended actions by the LLM. Direct injections overwrite system prompts, while indirect ones manipulate inputs from external sources.

LLM02

Insecure Output Handling

This vulnerability occurs when an LLM output is accepted without scrutiny, exposing backend systems. Misuse may lead to severe consequences like XSS, CSRF, SSRF, privilege escalation, or remote code execution.

LLM03

Training Data Poisoning

This occurs when LLM training data is tampered, introducing vulnerabilities or biases that compromise security, effectiveness, or ethical behavior. Sources include Common Crawl, WebText, OpenWebText, & books.

LLM04

Model Denial of Service

Attackers cause resource-heavy operations on LLMs, leading to service degradation or high costs. The vulnerability is magnified due to the resource-intensive nature of LLMs and unpredictability of user inputs.

LLM05

Supply Chain Vulnerabilities

LLM application lifecycle can be compromised by vulnerable components or services, leading to security attacks. Using third-party datasets, pre- trained models, and plugins can add vulnerabilities.

LLM06

Sensitive Information Disclosure

LLMs may inadvertently reveal confidential data in its responses, leading to unauthorized data access, privacy violations, and security breaches. It's crucial to implement data sanitization and strict user policies to mitigate this.

LLM07

Insecure Plugin Design

LLM plugins can have insecure inputs and insufficient access control. This lack of application control makes them easier to exploit and can result in consequences like remote code execution.

LLM08

Excessive Agency

LLM-based systems may undertake actions leading to unintended consequences. The issue arises from excessive functionality, permissions, or autonomy granted to the LLM-based systems.

LLM09

Overreliance

Systems or people overly depending on LLMs without oversight may face misinformation, miscommunication, legal issues, and security vulnerabilities due to incorrect or inappropriate content generated by LLMs.

LLM10

Model Theft

This involves unauthorized access, copying, or exfiltration of proprietary LLM models. The impact includes economic losses, compromised competitive advantage, and potential access to sensitive information.

LLM01

Prompt Injection

Attackers can manipulate LLMs through crafted inputs, causing it to execute the attacker's intentions. This can be done directly by adversarially prompting the system prompt or indirectly through manipulated external inputs, potentially leading to data exfiltration, social engineering, and other issues.

EXAMPLES

- **Direct Prompt Injection**: Malicious user injects prompts to extract sensitive information.
- **Indirect Prompt Injection**: Users request sensitive data via webpage prompts.
- **Scam Through Plugins**: Websites exploit plugins for scams.

PREVENTION

- **Privilege Control**: Limit LLM access and apply role-based permissions.
- **Human Approval**: Require user consent for privileged actions.
- **Segregate Content**: Separate untrusted content from user prompts.
- **Trust Boundaries:** Treat LLM as untrusted and visually highlight unreliable responses.

ATTACK SCENARIOS

- **Chatbot Remote Execution**: Injection leads to unauthorized access via chatbot.
- **Email Deletion**: Indirect injection causes email deletion.
- **Exfiltration via Image**: Webpage prompts exfiltrate private data.
- **Misleading Resume**: LLM incorrectly endorses a candidate.
- **Prompt Replay**: Attacker replays system prompts for potential further attacks.

LLM02

Insecure Output Handling

Insecure Output Handling is a vulnerability that arises when a downstream component blindly accepts large language model (LLM) output without proper scrutiny. This can lead to XSS and CSRF in web browsers as well as SSRF, privilege escalation, or remote code execution on backend systems.

EXAMPLES

- **Remote Code Execution**: LLM output executed in system shell, leading to code execution.
- **Cross-Site Scripting (XSS)**: LLM-generated JavaScript or Markdown causes browser interpretation.

PREVENTION

- **Zero-Trust Approach:** Treat LLM output like user input; validate and sanitize it properly.
- **OWASP ASVS Guidelines**: Follow OWASP's standards for input validation and sanitization.
- **Output Encoding**: Encode LLM output to prevent code execution in JavaScript or Markdown.

ATTACK SCENARIOS

- **Chatbot Shutdown**: LLM output shuts down a plugin due to a lack of validation.
- **Sensitive Data Capture**: LLM captures and sends sensitive data to an attacker-controlled server.
- **Database Table Deletion**: LLM crafts a destructive SQL query, potentially deleting all tables.
- **XSS Exploitation**: LLM returns unsanitized JavaScript payload, leading to XSS on the victim's browser.

LLM03

Training Data Poisoning

Training Data Poisoning refers to manipulating the data or fine-tuning process to introduce vulnerabilities, backdoors or biases that could compromise the model's security, effectiveness or ethical behavior. This risks performance degradation, downstream software exploitation and reputational damage.

EXAMPLES

- **Malicious Data Injection:** Injecting falsified data during model training.
- **Biased Training Outputs:** Model reflects inaccuracies from tainted data.
- **Content Injection:** Malicious actors inject biased content into training.

PREVENTION

- **Supply Chain Verification**: Verify external data sources and maintain "ML-BOM" records.
- **Legitimacy Verification**: Ensure data legitimacy throughout training stages.
- **Use-Case Specific Training**: Create separate models for different use-cases.

ATTACK SCENARIOS

- **Misleading Outputs**: LLM generates content that promotes bias or hate.
- **Toxic Data Injection**: Malicious users manipulate the model with biased data.
- **Malicious Document Injection**: Competitors insert false data during model training.

LLM04

Model Denial of Service

Model Denial of Service occurs when an attacker interacts with a Large Language Model (LLM) in a way that consumes an exceptionally high amount of resources. This can result in a decline in the quality of service for them and other users, as well as potentially incurring high resource costs.

EXAMPLES

- **High-Volume Queuing**: Attackers overload LLM with resource-intensive tasks.
- **Resource-Consuming Queries**: Unusual queries strain system resources.
- **Continuous Input Overflow**: Flooding LLM with excessive input.
- **Repetitive Long Inputs**: Repeated long queries exhaust resources.
- **Recursive Context Expansion**: Attackers exploit recursive behavior.

PREVENTION

- **Input Validation**: Implement input validation and content filtering.
- **Resource Caps**: Limit resource use per request.
- **API Rate Limits**: Enforce rate limits for users or IP addresses.
- **Queue Management**: Control queued and total actions.
- **Resource Monitoring**: Continuously monitor resource usage.

ATTACK SCENARIOS

- **Resource Overuse**: Attacker overloads a hosted model, impacting other users.
- **Webpage Request Amplification:** LLM tool consumes excessive resources due to unexpected content.
- **Input Flood**: Overwhelm LLM with excessive input, causing slowdown.
- **Sequential Input Drain**: Attacker exhausts context window with sequential inputs.

LLM05

Supply Chain Vulnerabilities

Supply chain vulnerabilities in LLMs can compromise training data, ML models, and deployment platforms, causing biased results, security breaches, or total system failures. Such vulnerabilities can stem from outdated software, susceptible pre-trained models, poisoned training data, and insecure plugin designs.

EXAMPLES

- **Package Vulnerabilities**: Using outdated components.
- **Vulnerable Models**: Risky pre-trained models for fine-tuning.
- **Poisoned Data**: Tainted crowd-sourced data.
- **Outdated Models**: Using unmaintained models.
- **Unclear Terms**: Data misuse due to unclear terms.

PREVENTION

- **Supplier Evaluation**: Vet suppliers and policies.
- **Plugin Testing**: Use tested, trusted plugins.
- **OWASP A06**: Mitigate outdated component risks.
- **Inventory Management**: Maintain an up-to-date inventory.
- **Security Measures:** Sign models and code, apply anomaly detection, and monitor.

ATTACK SCENARIOS

- **Library Exploitation**: Exploiting vulnerable Python libraries.
- **Scamming Plugin**: Deploying a plugin for scams.
- **Package Registry Attack**: Tricking developers with a compromised package.
- **Misinformation Backdoor**: Poisoning models for fake news.
- **Data Poisoning:** Poisoning datasets during fine-tuning.

LLM06

Sensitive Information Disclosure

LLM applications can inadvertently disclose sensitive information, proprietary algorithms, or confidential data, leading to unauthorized access, intellectual property theft, and privacy breaches. To mitigate these risks, LLM applications should employ data sanitization, implement appropriate usage policies, and restrict the types of data returned by the LLM.

EXAMPLES

- **Incomplete Filtering:** LLM responses may contain sensitive data.
- **Overfitting**: LLMs memorize sensitive data during training.
- **Unintended Disclosure**: Data leaks due to misinterpretation or lack of scrubbing.

PREVENTION

- **Data Sanitization**: Use scrubbing to prevent user data in training.
- **Input Validation**: Filter malicious inputs to avoid model poisoning.
- **Fine-Tuning Caution**: Be careful with sensitive data in model fine-tuning.
- **Data Access Control**: Limit external data source access.

ATTACK SCENARIOS

- **Unintentional Exposure**: User A exposed to other user data.
- **Filter Bypass**: User A extracts PII by bypassing filters.
- **Training Data Leak**: Personal data leaks during training.

LLM07

Insecure Plugin Design

Plugins can be prone to malicious requests leading to harmful consequences like data exfiltration, remote code execution, and privilege escalation due to insufficient access controls and improper input validation. Developers must follow robust security measures to prevent exploitation, like strict parameterized inputs and secure access control guidelines.

EXAMPLES

- **Single Field Parameter**s: Plugins lack parameter separation.
- **Configuration Strings**: Configurations can override settings.
- **Authentication Issues:** Lack of specific plugin authorization.
- **Raw SQL or Code:** Unsafe acceptance of code or SQL.

PREVENTION

- **Parameter Control**: Enforce type checks and use a validation layer.
- **OWASP Guidance**: Apply ASVS recommendations.
- **Thorough Testing**: Inspect and test with SAST, DAST, IAST.
- **Least-Privilege**: Follow ASVS Access Control Guidelines.
- **Auth Identities**: Use OAuth2 and API Keys for custom authorization.
- **User Confirmation**: Require manual authorization for sensitive actions.

ATTACK SCENARIOS

- **URL Manipulation**: Attackers inject content via manipulated URLs.
- **Reconnaissance and Exploitation**: Exploiting lack of validation for code execution and data theft.
- **Unauthorized Access**: Accessing unauthorized data through parameter manipulation.
- **Repository Takeover:** Exploiting insecure code management plugin for repository takeover.

LLM08

Excessive Agency

Excessive Agency in LLM-based systems is a vulnerability caused by over-functionality, excessive permissions, or too much autonomy. To prevent this, developers need to limit plugin functionality, permissions, and autonomy to what's absolutely necessary, track user authorization, require human approval for all actions, and implement authorization in downstream systems.

EXAMPLES

- **Excessive Functionality**: LLM agents have unnecessary functions, risking misuse.
- **Excessive Permissions:** Plugins may have excessive access to systems.
- **Excessive Autonomy**: LLMs lack human verification for high-impact actions.

PREVENTION

- **Limit Plugin Functions**: Allow only essential functions for LLM agents.
- **Plugin Scope Control**: Restrict functions within LLM plugins.
- **Granular Functionality**: Avoid open-ended functions; use specific plugins.
- **Permissions Contro**l: Limit permissions to the minimum required.
- **User Authentication**: Ensure actions are in the user's context.
- **Human-in-the-Loop**: Require human approval for actions.
- **Downstream Authorization**: Implement authorization in downstream systems.

ATTACK SCENARIOS

An LLM-based personal assistant app with excessive permissions and autonomy is tricked by a malicious email into sending spam. This could be prevented by limiting functionality, permissions, requiring user approval, or implementing rate limiting.

LLM09

Overreliance

Overreliance on LLMs can lead to serious consequences such as misinformation, legal issues, and security vulnerabilities. It occurs when an LLM is trusted to make critical decisions or generate content without adequate oversight or validation.

EXAMPLES

- **Misleading Info**: LLMs can provide misleading info without validation.
- **Insecure Code**: LLMs may suggest insecure code in software.

PREVENTION

- **Monitor and Validate**: Regularly review LLM outputs with consistency checks.
- **Cross-Check**: Verify LLM output with trusted sources.
- **Fine-Tuning**: Enhance LLM quality with task-specific fine-tuning.
- **Auto Validation**: Implement systems to verify output against known facts.
- **Task Segmentation**: Divide complex tasks to reduce risks.
- **Risk Communication**: Communicate LLM limitations.
- **User-Friendly Interfaces**: Create interfaces with content filters and warnings.
- **Secure Coding**: Establish guidelines to prevent vulnerabilities.

ATTACK SCENARIOS

- **Disinfo Spread:** Malicious actors exploit LLM-reliant news organizations.
- **Plagiarism:** Unintentional plagiarism leads to copyright issues.
- **Insecure Software:** LLM suggestions introduce security vulnerabilities.
- **Malicious Package:** LLM suggests a non-existent code library.

LLM10

Model Theft

LLM model theft involves unauthorized access to and exfiltration of LLM models, risking economic loss, reputation damage, and unauthorized access to sensitive data. Robust security measures are essential to protect these models.

EXAMPLES

- **Vulnerability Exploitation**: Unauthorized access due to security flaws.
- **Central Model Registry:** Centralized security for governance.
- **Insider Threat**: Risk of employee model leaks.
- **Side-Channel Attack**: Extraction of model details through side techniques.

PREVENTION & MITIGATION

- **Access Control and Authentication**: Strong access controls and authentication.
- **Network Restrictions**: Limit LLM access to resources and APIs.
- **Monitoring and Auditing**: Regular monitoring of access logs.
- **MLOps Automation**: Secure deployment with approval workflows.

ATTACK SCENARIOS

- **Model Theft**: Unauthorized access and use for competition.
- **Employee Leak**: Exposure increases risks.
- **Shadow Model Creation**: Replicating models with queries.
- **Side-Channel Attack:** Extraction through side techniques.

Key Reference Links

- **Prompt Injection attack against LLM-integrated Applications**: Cornell University
- **Defending ChatGPT against Jailbreak Attack via Self-Reminder**: Research Square
- **OpenAI Chat Markup Language**: GitHub
- **Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection**: Cornell University
- **Threat Modeling LLM Applications**: AI Village
- **Safety Best Practices**: OpenAI
- **Arbitrary Code Execution**: Snyk
- **CS324 - Large Language Models:** Stanford University
- **How data poisoning attacks corrupt machine learning models**: CSO Online
- **ML Supply Chain Compromise**: MITRE
- **Tay Poisoning**: MITRE
- **Backdoor Attacks on Language Models: Can We Trust Our Model's Weights?**: Medium
- **Poisoning Language Models During Instruction Tuning**: Cornell University
- **ChatGPT Data Breach Confirmed as Security Firm Warns of Vulnerable Component Exploitation**: Security Week
- **What Happens When an AI Company Falls Victim to a Software Supply Chain Vulnerability**: Security Boulevard
- **Plugin Review Process**: OpenAI
- **Compromised PyTorch-nightly dependency chain**: PyTorch