OWASP®

OWASP Top 10 for LLM

2023

OWASP.ORG/WWW-PROJECT-TOP-10-FOR-LARGE-LANGUAGE-MODEL-APPLICATIONS

OWASP Top 10 for LLM

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 add vulnerabilities.

LLM06

Sensitive Information Disclosure

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

LLM07

Insecure Plugin Design

LLM plugins can have insecure inputs and insufficient access control due to lack of application control. Attackers can exploit these vulnerabilities, resulting in severe 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

Prompt Injection allows attackers to manipulate Large Language Models (LLMs) through crafted inputs, leading to potential backend system exploitation or user interaction manipulation.

EXAMPLES

1. Attackers instruct LLM to return private information.
2. Hidden prompt injection in a webpage solicits sensitive information.
3. A document with a prompt injection manipulates LLM's output.
4. Rogue instruction on a website exploits a plugin, causing unauthorized actions.

PREVENTION

1. Restrict LLM's access to necessary operations.
2. Require user approval for privileged operations.
3. Limit untrusted content's influence on user prompts.
4. Establish trust boundaries and maintain user control.

ATTACK SCENARIOS

1. Adversarial prompt injection on a website causes unauthorized actions.
2. Hidden prompt injection in a resume manipulates LLM's output.
3. Direct prompt injection allows malicious user control over the LLM.

LLM02

Insecure Output Handling

Insecure Output Handling vulnerability occurs when a plugin or application accepts LLM output without scrutiny and passes it to backend or client-side functions. This can lead to XSS, CSRF, SSRF, privilege escalation, or remote code execution.

EXAMPLES

1. LLM output directly entered into a backend function, causing remote code execution.
2. JavaScript or Markdown generated by the LLM is interpreted by the browser, resulting in XSS.

PREVENTION

1. Apply input validation on responses from the model to backend functions.
2. Encode output from the model back to users to mitigate undesired code interpretations.

ATTACK SCENARIOS

1. Application passes LLM-generated response into an internal function without validation, leading to unauthorized access or modifications.
2. A website summarizer tool powered by an LLM captures sensitive content and sends it to an attacker-controlled server.
3. An LLM allows users to craft SQL queries for a backend database without scrutiny, leading to potential data loss.
4. A malicious user instructs the LLM to return a JavaScript payload back to a user, causing unsanitized XSS payload execution.

LLM03

Training Data Poisoning

This happens when an attacker manipulates the training data or fine-tuning procedures of an LLM, introducing vulnerabilities, backdoors, or biases that compromise the model's security, effectiveness, or ethical behavior. This can impact model performance, user trust, and brand reputation.

EXAMPLES

1. Malicious influence on model outputs via targeted, inaccurate documents.
2. Model training using unverified data.
3. Unrestricted dataset access by models leading to control loss.

PREVENTION

1. Verify training data supply chain and data source legitimacy.
2. Employ dedicated models per use-case.
3. Implement sandboxing, input filters, adversarial robustness.
4. Detect poisoning attacks via loss measurement and model analysis.

ATTACK SCENARIOS

1. Misleading LLM outputs result in biased opinions or hate crimes.
2. Injection of toxic data by malicious users.
3. Competitor manipulation of model's training data.

LLM04

Model Denial of Service

Model Denial of Service involves excessive resource consumption by an attacker's interaction with an LLM, leading to service quality decline and potential cost increases.

EXAMPLES

1. High-volume task generation through specific queries.
2. Unusually resource-consuming queries.
3. Continuous input overflow exceeding the LLM's context window.
4. Repeated long inputs or variable-length input floods.

PREVENTION

1. Implement input validation and sanitization.
2. Cap resource use per request.
3. Enforce API rate limits.
4. Monitor LLM resource utilization.
5. Set strict input limits based on the LLM’s context window.
6. Promote developer awareness about potential DoS vulnerabilities.

ATTACK SCENARIOS

1. Repeated requests to a hosted model, worsening service for other users.
2. Text on a webpage causing excessive web page requests.
3. Continuous input overflow or sequential inputs exhausting the context window.
4. Recursive context expansion or variable-length input floods.

LLM05

Supply Chain Vulnerabilities

Supply-chain vulnerabilities can impact the entire lifecycle of LLM applications, including third-party libraries/packages, docker containers, base images, and service suppliers. These vulnerabilities can lead to cyber-attacks, data disclosure, and tampering.

EXAMPLES

1. Use of vulnerable third-party components or base images.
2. Use of a tampered pre-built model for fine-tuning.
3. Use of poisoned external data sets for fine-tuning.

PREVENTION

1. Vet data sources and suppliers, including their T&Cs and privacy policies.
2. Use reputable plug-ins and ensure they have been tested for your application requirements.
3. Maintain an up-to-date inventory of components using a Software Bill of Materials (SBOM).

ATTACK SCENARIOS

1. Exploitation of a vulnerable or outdated package or base image.
2. Exploitation of a malicious or vulnerable ChatGPT plugin.
3. Exploitation of an outdated or deprecated model with vulnerabilities.

LLM06

Sensitive Information Disclosure

Sensitive Information Disclosure in LLMs refers to unintentional exposure of confidential details, including algorithms and user data, through system responses. This can lead to unauthorized access, privacy infringements, and other security breaches.

EXAMPLES

1. Malicious manipulation of model's training data.
2. Training models using unverified data.
3. Unrestricted model access to datasets.

PREVENTION

1. Utilize data sanitization and robust input validation.
2. Implement least privilege principle during fine-tuning.
3. Limit and control access to external data sources.

ATTACK SCENARIOS

1. Inadvertent user exposure to other user data via the LLM.
2. Bypassing input filters to trick LLM into leaking sensitive data.
3. Personal data leakage via training data.

LLM07

Insecure Plugin Design

Insecure LLM plugin design results in vulnerabilities due to insecure inputs and insufficient access control. Plugin integration APIs could permit free text inputs without validation, enabling potential malicious requests. Misconfigurations and poor access controls can have consequences.

EXAMPLES

1. Plugins accepting undifferentiated parameters.
2. Plugins taking URL strings instead of query parameters.
3. Plugins permitting raw SQL queries.
4. Lack of distinct authorizations for chained plugins.

PREVENTION

1. Enforce parameterized input with type and range checks.
2. Apply OWASP's recommendations for input validation.
3. Utilize least-privilege access control.
4. Use robust authentication like Oauth2.
5. Require user confirmation for sensitive plugins' actions.

ATTACK SCENARIOS

1. Malicious URL redirection for reconnaissance or content injection.
2. Exploitation of non-validated free-form input to perform harmful actions.
3. Unauthorized access through manipulation of configuration parameters.
4. SQL attacks via advanced filters.
5. Unsanctioned actions through insecure plugins.

LLM08

Excessive Agency

Excessive Agency in LLM refers to vulnerabilities enabling harmful actions due to unexpected LLM outputs, caused by excessive functionality, permissions, or autonomy.

EXAMPLES

1. Unnecessary or high-privilege plugin functions accessible to LLM.
2. Lack of proper input filtering in open-ended functions.
3. Over-granted permissions to LLM plugins.

PREVENTION

1. Limit plugin/tools accessible to LLM.
2. Implement only necessary functions in plugins.
3. Avoid open-ended functions, prefer granular functionality.
4. Limit LLM plugins' permissions on other systems.
5. Use OAuth for user authentication, granting minimum necessary privileges.
6. Require human approval for all actions.

ATTACK SCENARIOS

A personal assistant app with access to a user's mailbox is tricked into sending spam emails. Prevention: use a read-only plugin, authenticate with read-only scope, require user to manually send emails, or implement rate limiting on sending interface.

LLM09

Overreliance

Overreliance on LLMs refers to the vulnerability that arises when systems or individuals excessively trust LLMs for decision-making or content creation without appropriate oversight, leading to potential misinformation, miscommunication, or security risks.

EXAMPLES

1. Misinformation from incorrect LLM outputs.
2. Logically incoherent LLM outputs.
3. Confusion due to LLM merging varied sources.
4. LLM-suggested insecure code.
5. Inadequate LLM risk communication.

PREVENTION

1. Monitor LLM outputs, filter inconsistencies, and enhance with fine-tuning.
2. Verify LLM outputs with trusted sources.
3. Implement automatic validation mechanisms.
4. Break tasks into subtasks.
5. Communicate LLM-related risks clearly.
6. Develop safe interfaces and APIs.
7. Establish secure coding practices.

ATTACK SCENARIOS

1. False news spread due to AI misinformation.
2. Security vulnerabilities from AI coding suggestions.
3. Malicious package integration due to false LLM suggestion.

LLM10

Model Theft

LLM Model Theft refers to unauthorized access and extraction of Language Model models, leading to economic loss, competitive disadvantage, unauthorized model usage, and potential exposure of sensitive information.

EXAMPLES

1. External unauthorized access to LLM repositories.
2. Leaking models by insiders.
3. Network/application security misconfigurations.
4. Shared GPU services exploited for model access.
5. Replication of models via querying or prompt injection.
6. Side-channel attacks retrieving model data.

PREVENTION

1. Strong access controls/authentication for LLM repositories.
2. Limiting LLM's access to network resources.
3. Regular monitoring/auditing of LLM-related activities.
4. Automated MLOps deployment with governance.
5. Rate limiting and exfiltration detection techniques.

ATTACK SCENARIOS

1. Infrastructure vulnerability exploits.
2. Exploitation of shared GPU services.
3. Shadow model creation via API querying.
4. Insider-conducted side-channel attacks and employee leaks.

Key Reference Links

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