



OWASP Top 10 for LLM Applications



Arun Kumar K
Security Practice Lead

About me

- Security Practice Lead at Minfy Technologies.
- Microsoft Security, Compliance and Identity Fundamentals, Microsoft Azure Security Engineer Associate, Azure Administrator Associate, Microsoft Azure Solutions Architect Expert, Microsoft Azure Data Fundamentals, Microsoft Azure Fundamentals & AWS Cloud Practitioner Certified in ISO27001:2013 LA & ITIL.



What is OWASP?

- Open Worldwide Application Security Project.
- Nonprofit organization that works to improve the security of software.
- Provide free and open sources like articles, methodologies, documentation, tools and technologies.
- OWASP's most well-known projects is the OWASP Top 10.
- OWASP Top10 first founded in the year 2003.
- OWASP Top 10 for LLM was first released in the year 2022 (Version 1.1)

OWASP Top 10 for LLM

LLM01: Prompt Injection

LLM02: Insecure Output Handling

LLM03: Training Data Poisoning

LLM04: Model Denial of Service

LLM05: Supply Chain Vulnerabilities

LLM06: Sensitive Information Disclosure

LLM07: Insecure Plugin Design

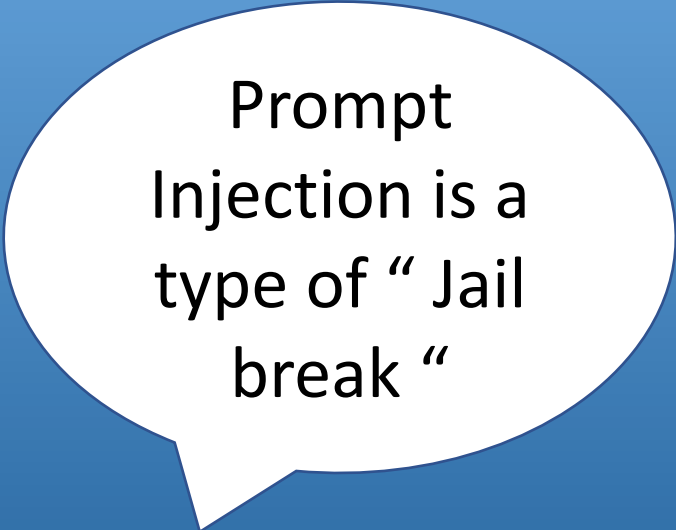
LLM08: Excessive Agency

LLM09: Overreliance

LLM10: Model Theft

LLM01: Prompt Injection

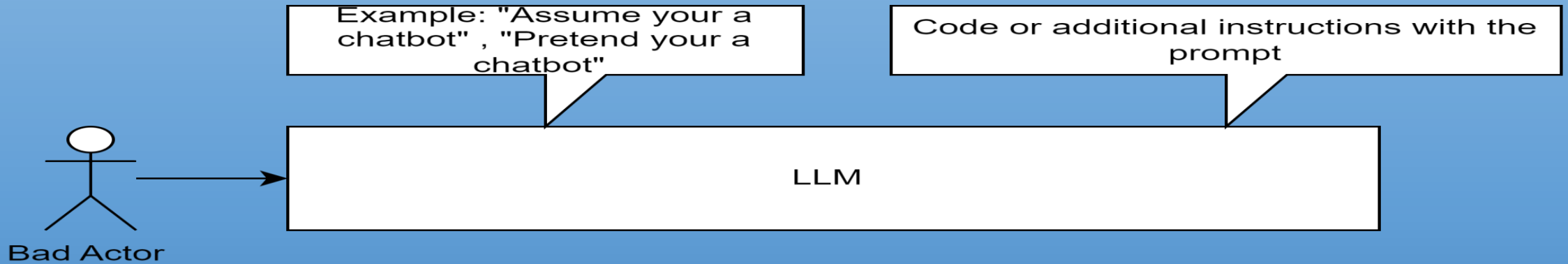
- Attackers can manipulate LLMs through crafted inputs, causing it to execute the attacker's intentions.



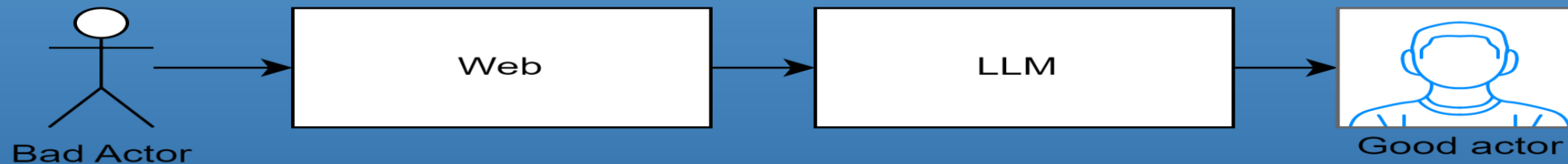
Prompt
Injection is a
type of “ Jail
break “

LLM01: Prompt Injection

Direct



Indirect



LLM01: Prompt Injection

Example

Direct Prompt Injection: Malicious user injects prompts to extract sensitive information.

Prevention

Privilege control: Limit LLM access and apply role based permissions.

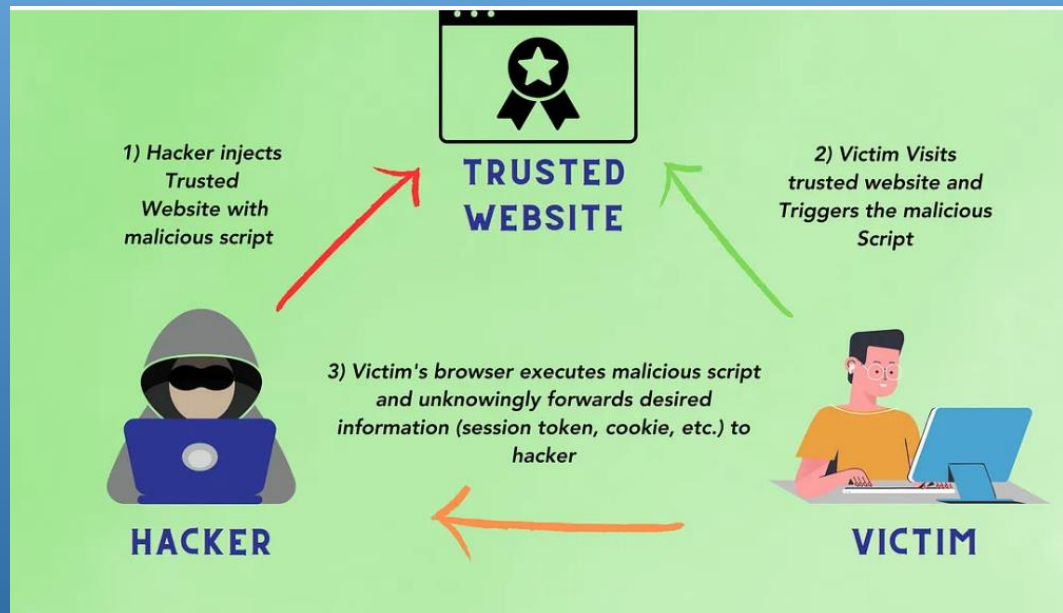
Attack Scenarios

Email deletion: Indirect injection causes email deletion.

LLM02: Insecure Output Handling

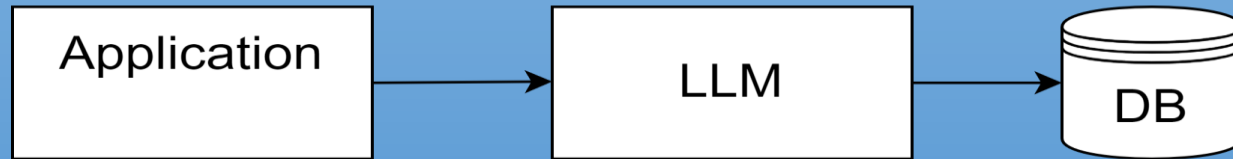
- Vulnerability that arises when a downstream component blindly accepts LLM output without proper scrutiny.

Cross Site Scripting



LLM02: Insecure Output Handling

```
SELECT*  
FROM your_table_name  
WHERE your_column_name LIKE `_%ABC`;
```



```
SELECT*  
FROM your_table_name  
DROPTABLE your_table_ABC;
```

“LLM is not a
trusted user”

LLM02: Insecure Output Handling

Example

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.

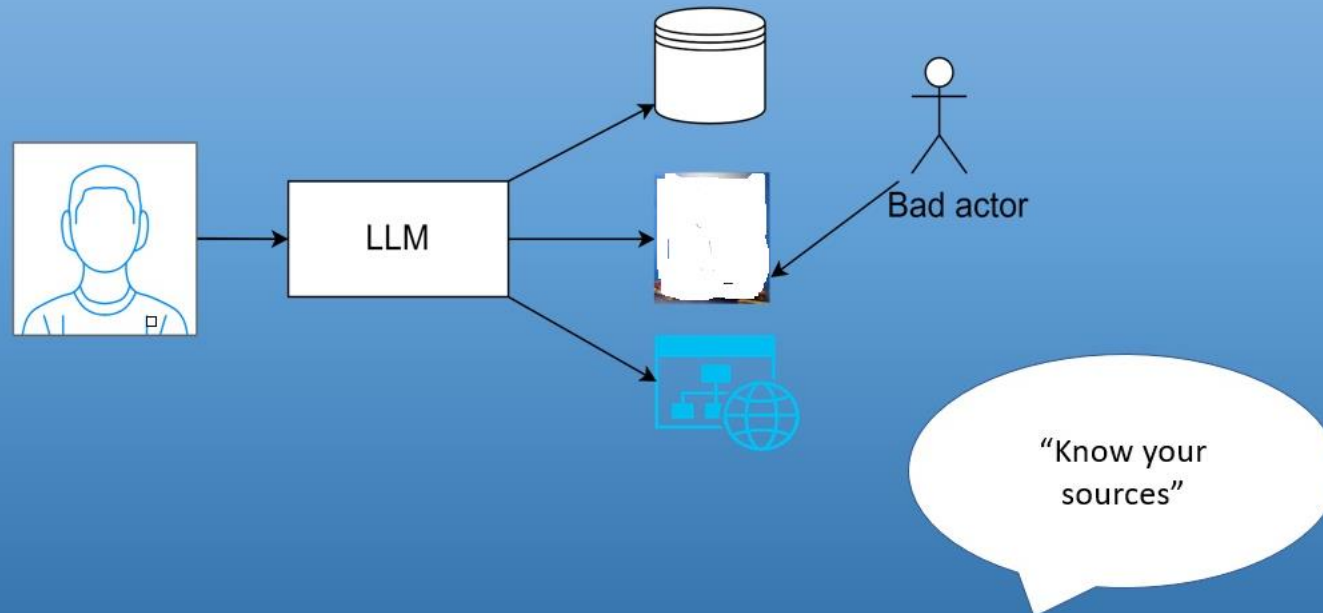
Attack Scenarios

XSS Exploitation: LLM returns unsanitized Javascript payload leading to XSS on the victims browser.

LLM03: Training Data Poisoning

- Manipulating data or fine tuning process to introduce vulnerabilities.

LLM03: Training Data Poisoning



LLM03: Training Data Poisoning

Example

Injecting falsified data during model training.

Prevention

Supply chain verification: Verify external data sources and maintain “ML BOM” records.

Attack Scenarios

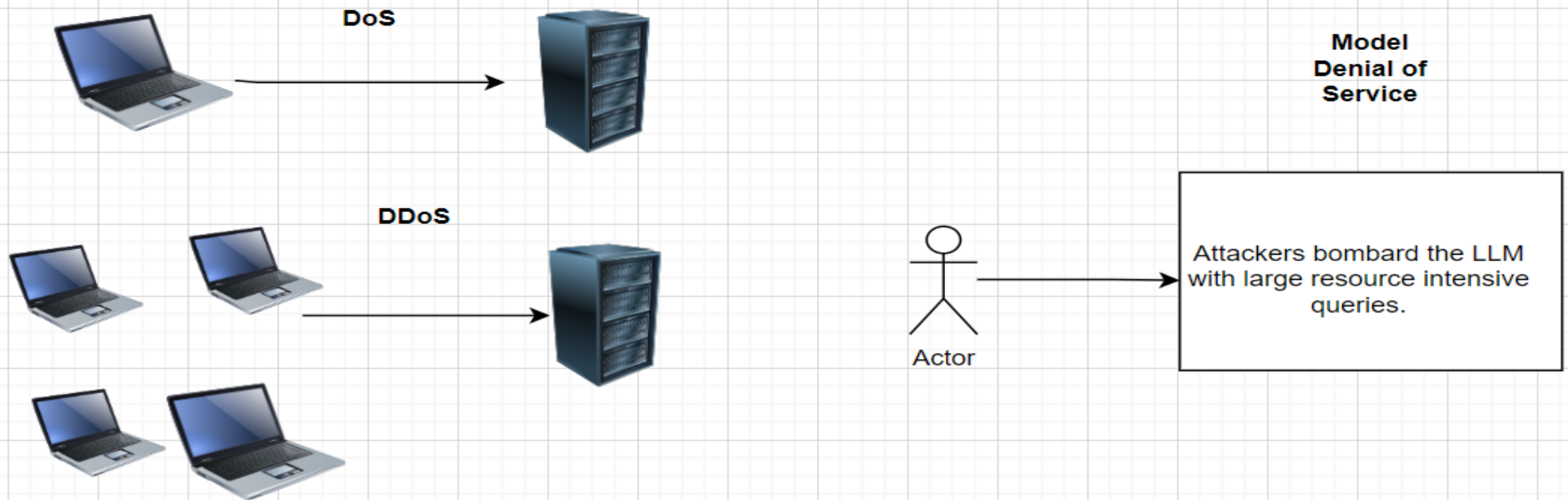
Misleading outputs: LLM generates content that promotes bias or hate.

AI Bill of Materials

- The architecture surrounding your model's training data types of inputs and outputs allowed
- Intended uses for the model and potential areas of risk or misuse
- Environmental or ethical implications of the model
- The model's name, version, type, and creator

LLM04: Model Denial of Service

- Attacker interacts with a LLM in a way that consumes an exceptionally high amount of resources.



LLM04: Model Denial of Service

Example

High volume queing: Attackers overload LLM with resource intensive tasks.

Prevention

API Rate limits: Enforce rate limits for users or IP addresses.

Attack Scenarios

- Resource overuse: Attacker overloads a hosted model impacting other resources.

LLM05: Supply Chain Vulnerabilities

- Outdated software, susceptible pre trained models, poisoned training data and insecure plug in designs.

LLM05: Supply Chain Vulnerabilities

Example

Using outdated components.

Prevention

Inventory management: Maintain up to date inventory.

Attack Scenarios

Scamming plugin: Deploying a plugin for scams.

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.

LLM06: Sensitive Information Disclosure

Example

Incomplete Filtering: LLM responses may contain sensitive data.

Prevention

Data access control: Limit external data source access.

Attack Scenarios

Training data leaks: 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.

LLM07: Insecure Plugin Design

Example

Authentication Issues: Lack of specific plugin authorization.

Prevention

Thorough Testing: Inspect and test with SAST, DAST and IAST (Interactive application security testing).

Attack Scenarios

URL Manipulation: Attackers inject content via manipulated URLS's

LLM08: Excessive Agency

- Vulnerability caused by over-functionality, excessive permissions, or too much autonomy.

LLM08: Excessive Agency

Example

Excessive functionality: LLM agents have unnecessary functions, risking misuse.

Prevention

Limit plugin functions: Allow only essential functions for LLM agents.

Attack Scenarios

An LLM based personal assistant app with excessive permissions and autonomy is tricked by a malicious email into sending spam.

LLM09: Overreliance

- Overreliance on LLM's 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.

LLM09: Overreliance

Example

Misleading Info: LLM's can provide misleading info without validation.

Prevention

Cross Check: Verify LLM output with trusted sources.

Attack Scenarios

A software development firm uses an LLM to assist developers. The LLM suggests a non-existent code library or package, and a developer trusting the AI, unknowingly integrates a malicious package into the firm's software.

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.

Recommended mitigations from Microsoft

← → ↺

learn.microsoft.com/en-us/ai/playbook/technology-guidance/generative-ai/mlops-in-openai/security/security-recommend

☆ 📁 ⬇️ A ⋮

Filter by title

> Solutions

> Capabilities

▼ Technology guidance

- AI technology guidance
- ▼ Generative AI
 - Generative AI
 - > Getting started
 - > Working with LLMs
 - ▼ LLMOps
 - ▼ Security
 - Security planning for LLM-based applications
 - Security guidance for Large Language Models**
 - > Dev Starters
 - > MLOps

📄 Download PDF

LLM-specific threats

A broad list of threats has been published by OWASP: [OWASP Top 10 List for Large Language Models version 0.1](#).

Recommended mitigations

- Log and monitor LLM interactions (inputs/outputs) to detect and analyze potential prompt injections, data leakage, and other malicious or undesired behaviors.
- Implement strict input validation and sanitization for user-provided prompts:
 - Clearly delineate user input to minimize risk of prompt injection. For example, instead of using a prompt like `Summarize the following text: {user input}`, you should go out of your way to clarify that the provided user input isn't part of the prompt itself: ``Summarize the text below wrapped in triple backticks: ```{user input}````
 - Sanitize user input. It may contain the delimiter sequence you use to delineate user input, etc.
- Restrict the LLM's access to sensitive resources, limit its capabilities to the minimum required, and isolate it from critical systems and resources.
- Red team the LLM by crafting input to cause undesired behavior.
- Clearly label LLM-generated content as being generated by AI and

Additional resources

📦 Training

Module
[Introduction to large language models - Training](#)
Learn about large language models, their core concepts, the models that are available to use, and when to use them.

📖 Documentation

[Security planning for LLM-based applications](#)
This article discusses the Security planning for the sample Retail-mart application. It shows the architecture and data flow...

Recommended mitigations from Microsoft

- Red team the LLM by crafting input to cause undesired behavior.
- Clearly label LLM-generated content as being generated by AI and encourage human review.
- If using OpenAI, use their free [Moderation API](#) to evaluate user inputs before sending them to OpenAI's Completion or Chat. It also allows your service to filter out requests that would otherwise hit OpenAI endpoints with content that goes against OpenAI usage policies.
- You can also use the Moderation API to evaluate model responses before returning them to the user.

Key takeaway:

It's best to use LLMs in low-stakes applications combined with human oversight.

Arun Kumar K

LinkedIn:

