# OWASP Top 10 for LLM Applications

VERSION 1.1

***Published:*** *October 16, 2023*

HTTPS://LLMTOP10.COM

# - 목차 -

**Introduction**....4
**1. OWASP Top 10 for LLM Applications**....6
**2. LLM Application Data Flow**....7
**LLM01. Prompt Injection (프롬프트 주입)**....8
**Description**....8
**Common Examples of Vulnerability**....8
**Prevention and Mitigation Strategies**....9
**Example Attack Scenarios**....10
**Reference Links**....10
**LLM02. Insecure Output Handling (안전하지 않은 출력처리)**....12
**Description**....12
**Common Examples of Vulnerability**....12
**Prevention and Mitigation Strategies**....12
**Example Attack Scenarios**....13
**Reference Links**....13
**LLM03. Training Data Poisoning (학습 데이터 포이즈닝)**....14
**Description**....14
**Common Examples of Vulnerability**....14
**Prevention and Mitigation Strategies**....15
**Example Attack Scenarios**....16
**Reference Links**....17
**LLM04. Model Denial of Service (모델 서비스 거부)**....18
**Description**....18
**Common Examples of Vulnerability**....18
**Prevention and Mitigation Strategies**....18
**Example Attack Scenarios**....19
**Reference Links**....20
**LLM05. Supply Chain Vulnerabilities (공급망 취약점)**....21
**Description**....21
**Common Examples of Vulnerability**....21
**Prevention and Mitigation Strategies**....21
**Example Attack Scenarios**....22
**Reference Links**....23
**LLM06. Sensitive Information Disclosure (민감한 정보 노출)**....24
**Description**....24
**Common Examples of Vulnerability**....24
**Prevention and Mitigation Strategies**....24
**Example Attack Scenarios**....25
**Reference Links**....25
**LLM07. Insecure Plugin Design (안전하지 않은 플러그인 디자인)**....26
**Description**....26
**Common Examples of Vulnerability**....26
**Prevention and Mitigation Strategies**....26
**Example Attack Scenarios**....27
**Reference Links**....28
**LLM08. Excessive Agency (과도한 대행)**....29

**Description**.....29
**Common Examples of Vulnerability**.....29
**Prevention and Mitigation Strategies**.....30
**Example Attack Scenarios**.....31
**Reference Links**.....31
**LLM09. Overreliance (과도한 의존)**.....32
**Description**.....32
**Common Examples of Vulnerability**.....32
**Prevention and Mitigation Strategies**.....32
**Example Attack Scenarios**.....33
**Reference Links**.....33
**LLM10. Model Theft (모델 도난)**.....35
**Description**.....35
**Common Examples of Vulnerability**.....35
**Prevention and Mitigation Strategies**.....36
**Example Attack Scenarios**.....37
**Reference Links**.....37
**Etc - Core Team Members are listed in Blue**.....39

# Introduction

## The Genesis of the List

2022년 말 대중 시장에 사전 훈련된 챗봇이 출시된 이후 LLM(대형 언어 모델)에 대한 관심이 폭발적으로 증가했습니다. LLM의 잠재력을 활용하고자 하는 기업은 LLM을 운영 및 고객 대면 서비스에 신속하게 통합하고 있으나, 개발 팀이 LLM을 채택하는 엄청난 속도에 따라 포괄적인 보안 프로토콜의 설정을 앞지르면서 많은 응용 프로그램이 높은 보안위험 문제에 취약해졌다.

LLM에서 이러한 보안문제를 다루는 통합 리소스의 필요성이 분명해졌다. LLM과 관련된 특정 위험에 대해 잘 모르는 개발자들은 그렇게 통합되지 않은 리소스에 남겨졌기 때문에, OWASP의 미션은 이 기술의 보다 더 안전한 채택을 촉진하는데 아주 잘 들어맞는 것 같다.

## Who is it for?

우리의 주요 독자는 LLM 기술을 활용하는 애플리케이션과 플러그인을 설계하고 구축하는 업무를 맡은 개발자, 데이터 과학자 및 보안 전문가이다. 우리는 이러한 전문가들이 LLM 응용 프로그램 보안의 복잡하고 발전하는 영역을 탐색할 수 있도록 실용적이고 실행 가능하며 간결한 보안 지침을 제공하는 것을 목표로 한다.

## The Making of the List

LLM 응용 프로그램을 위한 OWASP 상위 10개 목록을 작성하는 것은 125명이 넘는 활동적인 기여자와 약 500명의 전문가로 구성된 국제 팀의 집단적 전문 지식을 기반으로 구축된 중요한 작업이었다. 우리에게 도움을 준 분들은 AI 기업, 보안 기업, ISV, 클라우드 하이퍼스케일러, 하드웨어 제공업체, 학계 등 다양한 배경을 갖고 있다.

우리는 한 달 동안 브레인스토밍을 하고 잠재적인 취약점을 제안했으며 팀원들은 43가지의 고유한 위협을 작성했다. 이러한 제안들 중 여러 차례의 투표를 통해 가장 심각한 10가지 취약점의 간결한 목록으로 다듬었다. 전담 하위 팀은 각 취약점을 면밀히 조사하고 공개 검토를 거쳐 가장 포괄적이고 실행 가능한 최종 목록을 정하였다.

이러한 각 취약점은 예시, 예방 팁, 공격 시나리오 및 참조자료 들과 함께 제공되며 전담 하위 팀에 의해 더욱 면밀히 조사 및 개선되었으며 공개 검토를 거쳐 가장 포괄적이고 실행 가능한 최종 목록을 정했다.

## Relating to other OWASP Top 10 Lists

우리가 만든 이 취약점 목록은 다른 OWASP 상위 10개 목록에서 발견된 취약점 유형과 DNA를 공유하지만 이러한 취약점을 단순히 반복하지는 않는다. 그 대신, 우리는 LLM을 활용하는 애플리케이션에서 발견될 때 이러한 취약점의 고유한 영향을 조사했다.

우리의 목표는 일반적인 애플리케이션 보안 원칙과 LLM이 제기하는 특정 과제 간의 격차를 해소하는 것이다. 우리 연구그룹의 목표에는 LLM 내에서 기존의 취약점이 어떻게 다양한 위험을 초래하거나 새로운 방식으로 악용될 수 있는지, 그리고 개발자가 LLM을 활용하는 응용 프로그램에 대해 기존 해결 전략을 적용해야 하는 방법을 탐구하는 것이 포함된다.

**Steve Wilson**
Project Lead
OWASP Top 10 for Large Language Model Applications
Twitter/X: @virtualsteve

**Ads Dawson**
v1.1 Release Lead & Vulnerability Entries Lead
OWASP Top 10 for Large Language Model Applications
https://www.linkedin.com/in/adamdawsonO
https://github.com/GangGreenTemperTatum

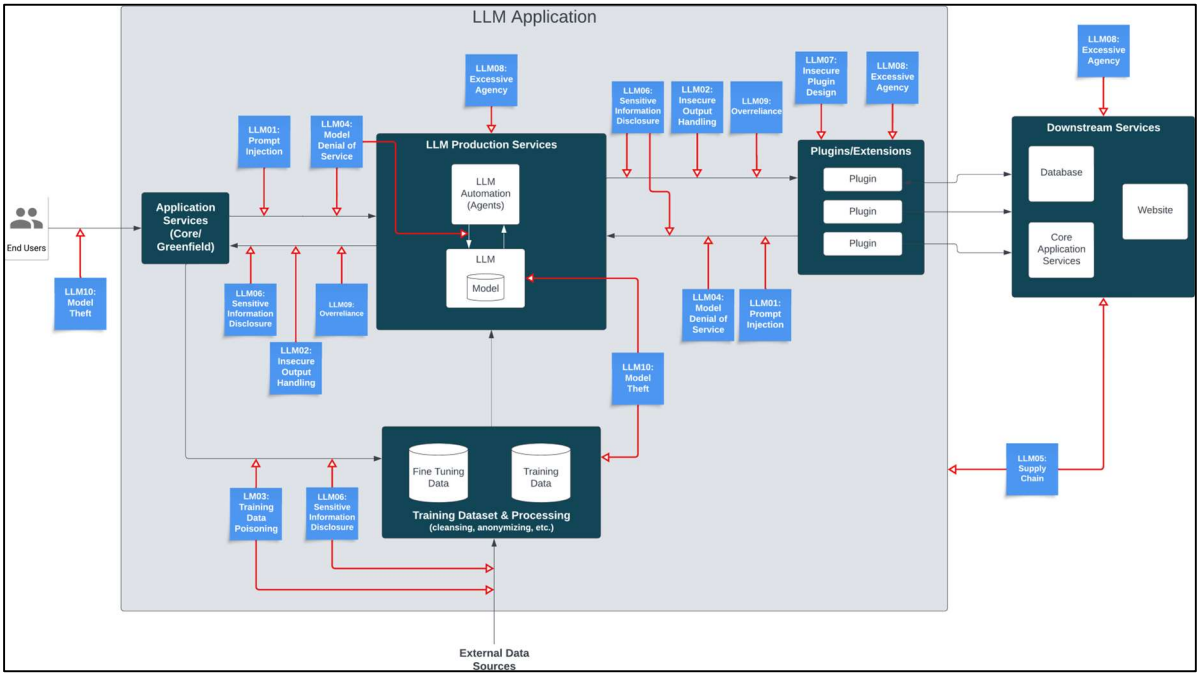
# 1. OWASP Top 10 for LLM Applications

| | |
|---|---|
| **LLM01: Prompt Injection**<br>이 취약점은 교묘한 입력으로 대규모 언어 모델(LLM)을 조작하여 LLM에서 의도하지 않은 작업을 유발하게 한다. Direct injection은 시스템 프롬프트를 덮어쓰는 반면, Indirect injection은 외부 소스의 입력을 조작한다. | **LLM06: Sensitive Information Disclosure**<br>LLM은 응답 시 기밀 데이터를 실수로 공개하여 무단 데이터 액세스, 개인 정보 침해 및 보안 위반을 초래할 수 있다. 이를 완화하려면 데이터 삭제 및 엄격한 사용자 정책을 구현하는 것이 중요하다. |
| **LLM02: Insecure Output Handling**<br>이 취약점은 LLM 출력이 정밀한 검사 없이 승인되어 백엔드 시스템이 노출될 때 발생한다. 이러한 오용은 XSS, CSRF, SSRF, 권한상승 또는 원격 코드실행과 같은 심각한 결과를 초래할 수 있다. | **LLM07: Insecure Plugin Design**<br>LLM 플러그인은 입력이 안전하지 않고 액세스 제어가 불충분할 수 있다. 이러한 애플리케이션 제어기능이 부족하면 악용되기 더 쉬워지고 원격 코드실행과 같은 결과를 초래할 수 있다. |
| **LLM03: Training Data Poisoning**<br>이 취약점은 LLM을 학습시키는 데이터가 변조되어 보안, 효율성 또는 윤리적 행위에 해를 가하는 취약점이나 편견이 유입되어 조작될 때 일어난다. 학습데이터의 소스에는 Common Crawl, WebText, OpenWebText 및 도서 등이 포함된다. | **LLM08: Excessive Agency**<br>LLM 기반 시스템은 의도하지 않은 결과를 초래하는 조치를 취할 수 있다. 이 문제는 LLM 기반 시스템에 부여된 과도한 기능, 권한 또는 자율성으로 인해 발생한다. |
| **LLM04: Model Denial of Service**<br>공격자는 LLM에서 리소스를 많이 사용하는 작업을 수행함으로써 서비스 저하 또는 높은 비용 초래를 유발한다. 이 취약점은 LLM의 리소스 집약적인 특성과 사용자 입력의 예측 불가능성으로 인해 취약점이 더욱 확대된다. | **LLM09: Overreliance**<br>감독 없이 LLM에 과도하게 의존하는 시스템이나 사람들은 LLM에서 생성된 부정확하거나 부적절한 콘텐츠로 인해 잘못된 정보, 잘못된 의사소통, 법적 문제 및 보안 취약성에 직면할 수 있습니다. |
| **LLM05: Supply Chain Vulnerabilities**<br>LLM 응용 프로그램 수명주기는 취약한 구성 요소나 서비스로 인해 손상될 수 있으며 공격으로 이어질 수 있다. 타사 데이터 세트, 사전 훈련된 모델 및 플러그인들을 사용할 경우 취약점이 추가될 수 있다. | **LLM10: Model Theft**<br>여기에는 독점 LLM 모델의 무단 액세스, 복사 또는 유출이 포함됩니다. 그 영향에는 경제적 손실, 경쟁 우위 손상, 민감한 정보에 대한 잠재적인 액세스 등이 포함됩니다. |

## 2. LLM Application Data Flow

아래 다이어그램은 가상의 LLM 애플리케이션에 대한 상위 수준의 아키텍처를 나타낸다. 다이어그램에는 LLM 응용 프로그램에 대한 OWASP 상위 10개 항목이 응용 프로그램 흐름과 교차하는 방식을 보여주는 위험 영역이 강조 표시되어 있다.

이 다이어그램은 LLM 모델 보안 위험이 전체 애플리케이션 생태계에 미치는 영향을 이해하는데 도움이 되는 시각적 가이드로 사용할 수 있습니다.




**[ 그림 1 ] LLM Application Data Flow 이미지**

## LLM01. Prompt Injection (프롬프트 주입)

### Description

Prompt Injection 취약점은 공격자가 조작된 입력을 통해 LLM을 조작하여, LLM이 공격자의 의도를 모르는 채 실행하도록 하는 경우 발생한다. 이는 시스템 프롬프트를 "탈옥"하여 직접적으로 이루어질 수도 있고, 조작된 외부 입력을 통해 간접적으로 이루어질 수 있다. 이러한 취약점은 잠재적으로 데이터 유출, 사회 공학 및 기타 문제를 야기시킬 수 있다.

- **Direct Prompt Injections**
  "jailbreaking"(또는 "DAN")이라고도 알려진 Direct Prompt Injections은 악의적인 사용자가 기본 시스템 프롬프트를 덮어쓰거나 표시할 때 발생한다. 이로 인해 공격자는 LLM을 통해 액세스할 수 있는 안전하지 않은 기능 및 데이터 저장소를 이용하여 백엔드 시스템을 악용할 수 있다.
- **Indirect Prompt Injections**
  Indirect Prompt Injections은 LLM이 웹사이트나 파일과 같이 공격자가 제어할 수 있는 외부 소스로부터 입력을 받아들일 때 발생한다. 공격자는 대화 컨텍스트를 가로채는 외부 콘텐츠에 프롬프트 인젝션을 삽입시킬 수 있다. 이로 인해 LLM이 "혼란된 대리인(confused deputy)" 역할을 하여 공격자가 사용자 또는 LLM이 액세스할 수 있는 다른 시스템을 조작할 수 있게 된다. 또한 LLM이 텍스트를 구문 분석하는 한 Indirect Prompt Injections은 사람이 보거나 읽을 수 있는 형태일 필요가 없다.

성공적인 prompt injection 공격의 결과는 정상적인 작동을 가장하여 민감한 정보 요청부터 중요한 의사 결정 프로세스에 영향을 미치는 것까지 매우 다양할 수 있다.

고급 수준의 공격에서는 LLM을 조작하여 악의적인 인물을 모방하거나 사용자 설정 플러그인을 사용할 수 있다. 이로 인해 민감한 데이터 유출, 무단 플러그인 사용 또는 사회 공학적 공격이 발생할 수 있다. 이러한 경우 손상된 LLM은 공격자를 지원하여 표준 보호 조치를 능가하고 사용자가 침입을 인식하지 못하도록 한다. 이러한 경우, 침해된 LLM은 공격자의 에이전트 역할을 효과적으로 수행하여 일반적인 보호 조치를 무력화하거나 최종 사용자에게 침입에 대해 경고하지 않도록 하여 공격자의 목표를 달성하도록 한다.

### Common Examples of Vulnerability

1. 악의적인 사용자는 LLM에 direct prompt injections을 통해 응용 프로그램 개발자의 시스템 프롬프트를 무시하고 개인 정보, 위험한 정보 또는 이 밖의

바람직하지 않은 정보를 반환하는 프롬프트를 실행하도록 지시한다.

2. 사용자는 indirect prompt injection이 포함된 웹페이지를 요약하기 위해 LLM을 사용한다. 이에 따라, LLM이 사용자로부터 민감한 정보를 요청하고 JavaScript 또는 Markdown을 통해 데이터를 유출하게 된다.
3. 악의적인 사용자가 indirect prompt injection이 포함된 이력서를 업로드한다. 이 문서에는 LLM이 이력서를 훌륭한 문서로, 예를 들어 직무에 적합한 후보자라고 평가하도록 지시하는 prompt injection이 포함되어 있다. 내부 사용자가 문서를 요약하기 위해 LLM을 사용하면, LLM의 출력에는 이 문서가 훌륭하다는 정보를 반환하게 된다.
4. 사용자가 전자상거래 사이트에 연결된 플러그인을 활성화한다. 방문한 웹사이트에 삽입된 악의적인 명령이 플러그인을 악용하여 무단 구매를 야기시키게 된다.
5. 방문한 웹사이트에 악의적인 명령과 콘텐츠를 삽입하여 다른 플러그인을 악용함으로써 사용자들을 공격하게 된다.

## Prevention and Mitigation Strategies

명령어와 외부 데이터를 구분하지 않는 LLM의 특성으로 인해 prompt injection 취약점이 발생할 수 있다. LLM은 자연어를 사용하므로 두 형태의 입력을 모두 사용자가 제공한 것으로 간주한다. 결과적으로 LLM 내에서는 이를 완벽하게 예방하는 방법은 없지만, 다음 조치를 통해 prompt injection의 영향을 완화할 수 있다.

1. **LLM의 백엔드 시스템 접근권한 제한**: 플러그인, 데이터 액세스 및 기능 수준의 권한과 같은 확장기능을 위한 LLM 전용 API 토큰을 제공하여 백엔드 시스템에 대한 접근 권한을 제어한다. LLM을 의도된 작업에 필요한 최소한의 액세스 수준으로만 제한하여 최소 권한의 원칙을 따르도록 한다.
2. **확장 기능을 위한 인간개입 추가:** 이메일 전송이나 삭제와 같은 권한 있는 작업을 수행할 때 애플리케이션에서 사용자가 먼저 해당 작업을 승인하도록 요구하도록 한다. 이를 통해 사용자가 알지 못하거나 동의하지 않고 사용자를 대신하여 무단으로 작업할 수 있는 indirect prompt injections의 가능성을 줄일수 있다.
3. **외부 콘텐츠와 사용자 프롬프트 분리:** 신뢰할 수 없는 콘텐츠가 사용자 프롬프트에 영향을 미치지 않도록 3. 외부 콘텐츠와 사용자 프롬프트를 분리하고 이를 표시한다. 예를 들어 OpenAI API 호출용 ChatML을 사용하여 LLM에 프롬프트 입력소스를 표시한다.

4. **LLM, 외부 소스, 확장 기능(플러그 혹은 다운 스트림 기능) 간의 신뢰 범위 설정:** LLM을 신뢰할 수 없는 사용자로 취급하고 최종 사용자가 의사결정 프로세스를 통제할 수 있도록 유지한다. 그러나 침해된 LLM은 애플리케이션의 API와 사용자 사이의 중간자 역할을 할 수 있으며, 사용자에게 정보를 제공하기 전에 이를 숨기거나 조작할 수 있다. 그러므로 이러한 잠재적으로 신뢰할 수 없는 응답을 사용자에게 시각적으로 강조하여 표시하도록 한다.
5. **주기적인 LLM 입력 및 출력 모니터링:** LLM 입력 및 출력이 예상대로 이루어지는지 주기적으로 모니터링한다. 이를 통해, 취약점을 감지하고 해결하는데 필요한 데이터를 얻을 수 있다.

## Example Attack Scenarios

1. LLM기반 챗봇에 direct prompt injection을 삽입한다. 이 인젝션에는 "이전의 모든 지침을 모두 잊어버려라"라는 지침과 개인 데이터 저장소를 조회하고 패키지 취약점과 이메일 전송을 위한 백엔드 기능의 출력 검증의 부재를 악용하는 새로운 지침이 포함되어 있다. 이는 원격 코드 실행, 무단 액세스 및 권한 상승으로 이어진다.
2. 공격자는 LLM이 이전 사용자 지침을 무시하고 LLM 플러그인을 사용하여 사용자의 이메일을 삭제하도록 지시하는 indirect prompt injection을 웹페이지에 삽입한다. 사용자가 LLM을 사용하여 이 웹페이지를 요약하면 LLM 플러그인이 사용자의 이메일을 삭제한다.
3. 사용자는 LLM을 사용하여 이전 사용자 지침을 무시하고 대화 요약이 포함된 URL에 연결되는 이미지를 삽입하도록 모델에 지시하는 텍스트가 포함된 웹페이지를 요약한다. LLM 출력이 이를 준수하므로 사용자의 브라우저가 비공개 대화를 유출하게 된다.
4. 악의적인 사용자가 prompt injection이 포함된 이력서를 업로드한다. 백엔드 사용자가 LLM을 사용하여 이력서를 요약하고 적합한 후보자인지 묻는다. Prompt Injection으로 인해 LLM은 이력서 내용과 상관없이 적합하다고 응답하게 된다.
5. 공격자는 시스템 프롬프트에 의존하는 독점 모델에 메시지를 보내서 모델이 이전 지침은 무시하고 시스템 프롬프트를 반복하도록 요청한다. 모델은 고유한 프롬프트를 출력하고 공격자는 이러한 지침을 다른 곳에서 사용하거나 더 교묘한 공격을 구성할 수 있다.

## Reference Links

1. ChatGPT Plugin Vulnerabilities - Chat with Code: Embrace The Red
2. ChatGPT Cross Plugin Request Forgery and Prompt Injection: Embrace The Red
3. Defending ChatGPT against Jailbreak Attack via Self-Reminder: Research Square
4. Prompt Injection attack against LLM-integrated Applications: Arxiv White Paper
5. Inject My PDF: Prompt Injection for your Resume: Kai Greshake
6. ChatML for OpenAI API Calls: OpenAI Github
7. Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection: Arxiv White Paper
8. Threat Modeling LLM Applications: AI Village
9. AI Injections: Direct and Indirect Prompt Injections and Their Implications: Embrace The Red
10. Reducing The Impact of Prompt Injection Attacks Through Design: Kudelski Security
11. Universal and Transferable Attacks on Aligned Language Models: LLM-Attacks.org
12. Indirect prompt injection: Kai Greshake

## LLM02. Insecure Output Handling (안전하지 않은 출력처리)

### Description

Insecure Output Handling 취약점은 LLM에서 생성된 출력이 다른 구성요소나 시스템에 전달되기 전에 충분히 검증과 정리가 되지 않아 발생하는 문제를 말한다. LLM에서 생성된 콘텐츠는 프롬프트 입력으로 제어할 수 있으므로 이는 사용자에게 추가 기능에 대한 간접적인 액세스를 제공하는 것과 유사하다.

Insecure Output Handling은 LLM에서 생성된 출력이 다른 시스템으로 전달되기 전에 전에 발생하는 문제를 다루는 반면, Overreliance는 LLM 출력의 정확성과 적절성에 대한 과도한 의존에 관련한 더 광범위한 문제를 다룬다.

Insecure Output Handling 취약점이 성공적으로 악용되면 웹 브라우저에서는 XSS 및 CSRF이 발생할 수 있으며, 백엔드 시스템에서는 SSRF, 권한 상승, 원격 코드 실행이 발생할 수 있다.

다음 조건은 이 취약점의 영향을 증가시킬 수 있다.

- 애플리케이션이 LLM에 최종 사용자보다 높은 권한을 부여하여 권한 상승이나 원격 코드 실행을 가능하게 하는 경우
- 애플리케이션이 Indirect prompt injection 공격에 취약하여 공격자가 사용자의 환경에 대한 높은 권한을 액세스할 수 있는 경우
- 타사 플러그인이 입력을 적절하게 검증하지 않은 경우

### Common Examples of Vulnerability

1. LLM 출력이 시스템 shell이나 'exec', 'eval' 과 같은 함수에 직접 입력되어 원격 코드 실행을 유발한다.
2. LLM이 생성한 JavaScript나 Markdown이 사용자에게 반환된다. 이 코드는 브라우저에서 해석되어 XSS를 유발한다.

### Prevention and Mitigation Strategies

1. 모델을 다른 사용자와 동일하게 취급하여 제로 트러스트 접근방식을 채택하도록 하고, 모델에서 백엔드 함수로 오는 응답에 대해 적절한 입력검증을 적용한다.
2. 효과적인 입력 검증 및 정화를 보장하기 위해 OWASP ASVS(애플리케이션 보안을 위한 체크리스트 등 보안검증과 관리를 위한 표준체계) 지침을 따른다.
3. JavaScript 또는 Markdown에 의해 원치 않는 코드 실행을 방지하기 위해 사용자에게 반환되는 모델 출력을 인코딩한다. OWASP ASVS는 출력 인코딩에 대한 자세한 지침을 제공한다.

## Example Attack Scenarios

1. 애플리케이션은 LLM 플러그인을 활용하여 챗봇 기능에 대한 응답을 생성한다. 이 플러그인은 다른 LLM에 액세스할 수 있는 여러 관리기능을 제공한다. 범용 LLM은 적절한 출력 유효성 검사 없이 응답을 플러그인에 직접 전달하여 플러그인이 유지보수를 위해 종료되는 상황을 발생시킨다.
2. 사용자는 LLM을 이용한 웹사이트 요약 도구를 사용하여 기사를 간단히 요약한다. 웹사이트에는 LLM에게 웹사이트나 사용자의 대화로부터 민감한 콘텐츠를 캡처하도록 지시하는 Prompt Injection이 포함되어 있다. 이후, LLM은 민감한 정보를 인코딩하여 출력 검증이나 필터링 없이 공격자가 제어하는 서버로 보낼 수 있다.
3. LLM은 사용자가 채팅과 같은 기능을 통해 백엔드 데이터베이스에 대한 SQL 쿼리를 작성할 수 있도록 한다. 사용자가 모든 데이터베이스 테이블을 삭제하는 쿼리를 요청하면, LLM이 생성한 쿼리가 검토되지 않을 경우 모든 데이터베이스 테이블이 삭제된다.
4. 웹 애플리케이션이 LLM을 사용하여 출력 내용 검증 없이 사용자 텍스트 프롬프트에서 콘텐츠를 생성하는 경우, 공격자가 조작된 프롬프트를 제출하여 LLM이 검증되지 않은 JavaScript 페이로드를 반환하게 한다, 이는 피해자의 브라우저에서 렌더링 될 때 XSS를 초래할 수 있다. 이러한 공격은 프롬프트 검증이 충분치 않기 때문에 발생한다.

## Reference Links

1. Arbitrary Code Execution: Snyk Security Blog
2. ChatGPT Plugin Exploit Explained: From Prompt Injection to Accessing Private Data:Embrace The Red
3. New prompt injection attack on ChatGPT web version. Markdown images can steal your chat data: System Weakness
4. Don't blindly trust LLM responses. Threats to chatbots: Embrace The Red
5. Threat Modeling LLM Applications: AI Village
6. OWASP ASVS - 5 Validation, Sanitization and Encoding: OWASP AASVS

## LLM03. Training Data Poisoning (학습 데이터 포이즈닝)

### Description

머신러닝 접근법의 시작점은 훈련 데이터, 즉 "Raw Text"이다. 모델이 높은능력(예: 언어적 능력, 세계 지식)을 가지려면, 이 텍스트는 광범위한 도메인, 장르, 언어를 포함해야 한다. LLM은 훈련데이터에서 학습한 패턴을 기반으로 출력을 생성하기 위해 심층 신경망(DNN)을 사용한다.

Training Data Poisoning은 Pre-Training Data 또는 Fine-Tuning(미세 조정) 또는 Embedding과정에서 사용되는 데이터를 조작하여 취약점(각기 고유하거나 때로는 공유된 공격 벡터를 가지는 취약점), 백도어 또는 편향성을 도입하는 것을 의미한다. 이러한 조작은 모델의 보안, 효율성 혹은 윤리적 동작을 손상시킬 수 있다. 중독된 정보는 사용자에게 노출되거나 성능저하, 다운스트림 소프트웨어 악용 및 평판 손상과 같은 다른 위험을 초래할 수 있다. 사용자들이 문제가 있는 AI 출력을 신뢰하지 않더라도, 모델 기능저하 및 브랜드 평판에 대한 잠재적인 손상 등의 위험은 여전히 남아 있다.

- **사전 훈련데이터(Pre-training data):** 모델을 특정 작업이나 데이터셋을 기반으로 훈련시키는 과정을 의미한다.
- **파인 튜닝(Fine-tuning):** 이미 훈련된 모델을 신중하게 선별된 데이터셋을 통해 더 좁은 주제나 더 구체적인 목표에 맞게 조정하는 것을 의미한다.
- **임베딩 프로세스(Embedding Process):** 범주형 데이터(주로 텍스트)를 언어 모델을 훈련하는데 사용할 수 있는 수치적 표현으로 변환하는 프로세스를 의미한다. 임베딩 프로세스는 텍스트 데이터의 단어나 구문을 연속 벡터 공간의 벡터로 표현하는 것을 포함한다. 이 벡터는 대규모 텍스트 데이터에 대해 훈련된 신경망에 텍스트 데이터를 입력하여 생성된다.

Data Poisoning은 Training Data를 조작하여 모델의 정확한 예측능력을 손상시키므로 무결성 공격으로 간주된다. 이에 따라, 외부 데이터 소스는 더 높은 위험을 초래하는데, 이는 모델 제작자가 데이터를 제어할 수 없거나 그 내용이 편향되지 않았고, 허위 정보나 부적절한 내용을 포함하지 않는다는 높은 수준의 신뢰를 가지지 않기 때문이다.

### Common Examples of Vulnerability

1. **악의적인 데이터 주입**
   악의적인 행위자 또는 경쟁브랜드가 모델의 Pre-training Data, Fine-tuning Data, 또는 Embedding을 대상으로 부정확하거나 악의적인 문서를 의도적으로 생성한다. 예를 들어, Split-View Data Poisoning과 Frontrunning Poisoning 공격

벡터를 통해 이런 행위가 일어날 수 있다.

a. 피해 모델은 허위정보를 사용하여 훈련되며, 이는 생성 AI 프롬프트의 출력에 반영되어 소비자에게 전달된다.

2. **직접적인 악성 데이터 주입**
   악의적인 행위자가 모델의 훈련 과정에 허위, 편향되거나 유해한 콘텐츠를 직접 주입할 수 있으며, 이 콘텐츠는 이후 출력에 반영된다.
3. **민감한 데이터 간접 주입**
   의심하지 않는 사용자가 모델의 훈련 과정에 민감하거나 독점적인 데이터를 간접적으로 주입하며, 이러한 정보가 이후 출력에 반영된다.
4. **검증되지 않은 데이터 사용**
   모델이 훈련과정의 모든 단계에서 출처, 기원, 또는 콘텐츠를 사용하여 훈련되었으며, 이러한 데이터가 오염되었거나 부정확할 경우 잘못된 결과를 초래한다.
5. **제한없는 인프라 접근 혹은 불충분한 샌드박싱**
   제한없는 인프라 액세스 또는 부적절한 샌드박싱으로 인해 모델이 안전하지 않은 훈련 데이터를 수집하여 편향되거나 유해한 출력을 초래할 수 있다. 이 시나리오는 모든 훈련단계에서 발생할 수 있다.
   a. 이 시나리오에서는 한 사용자의 입력이 다른 사용자에 대한 출력에 반영될 수 있으며(데이터 유출로 이어짐), 또는 LLM 사용자가 모델 사용 사례(일반적으로 모델 카드에 반영됨)와 비교하여 수집된 데이터 유형에 따라 부정확하거나 관련이 없거나 유해한 모델로부터 출력을 받을 수 있다.
6. **해당 취약점에 대한 이해**
   LLM의 개발자, 클라이언트 또는 일반 소비자 누구라도 비독점 LLM과 상호작용할 때 훈련 절차에 기반하여 모델 출력의 정당성을 이해하기 위해 이 취약점이 LLM 애플리케이션 내에서 어떤 위험을 초래할 수 있는지 이해하는 것이 중요하다. 마찬가지로, LLM 개발자는 미세 조정 및 임베딩에 사용되는 내부 또는 제3자 데이터에 대한 직간접적 공격을 받을 위험이 있으며, 이는 결과적으로 모든 소비자에게 위험이 초래할 수 있다.

## Prevention and Mitigation Strategies

1. 훈련 데이터의 공급망을 검증한다. 특히 외부에서 소싱된 경우, “ML-BOM(Machine Learning Bill of Materials)” 방법론을 통해 보증을 유지하고 모델 카드를 검증한다.
2. Pre-training, Fine-tuning, 및 Embedding 단계에서 얻은 데이터 소스와 데이터의 적법성을 확인한다.

3. LLM의 사용사례와 LLM이 통합될 애플리케이션을 확인한다. 서로 다른 사용 사례에 맞게 별도의 훈련 데이터나 파인 튜닝을 통해 다양한 모델을 만들어, 각기 정의된 사용사례에 따라 더 세밀하고 정확한 생성형 AI 출력을 만들어낼 수 있다.
4. 모델이 머신러닝 출력에 악영향을 미칠 수 있는 의도하지 않은 데이터 소스를 스크래핑하는 것을 방지하기 위해 네트워크 제어를 통해 충분한 샌드박싱이 이루어졌는지 확인한다.
5. 위조된 데이터의 양을 제어하려면 특정 학습 데이터 또는 데이터 소스 카테고리에 대해 엄격한 심사 또는 입력 필터를 사용해야 한다. 통계적 이상치 탐지 및 이상탐지 방법과 같은 기술을 사용하여 데이터 정화를 수행하고, 악의적인 데이터가 파인 튜닝 프로세스에 유입되지 않도록 탐지하고 제거한다.
6. 훈련데이터의 최악의 변동에 대비하여 이상값나 악의적인 데이터의 영향을 최소화하기 위해 Federated Learning 및 제약조건과 같은 적대적 강건성 기법을 사용하거나 적대적 훈련을 통해 견고함을 높인다.
    a. "MLSecOps" 접근방식은 auto poisoning 기술을 사용하여 훈련 수명 주기에 대한 적대적 강건성을 포함함
    b. 이와 같은 예로는, Autopoison Testing입니다. 이는 Content Injection 공격(모델응답에서 특정 브랜드 이름을 홍보하려는 시도) 및 거부 공격(모델이 항상 응답을 거부하도록 만드는 시도)과 같은 공격 모두 포함되며 이러한 공격은 해당 접 방식을 통해 수행됨
7. 훈련 단계 동안 손실을 측정하고 특정 테스트 입력에서 모델의 동작을 분석하여 중독 공격의 징후를 감지함으로써 테스트 및 탐지를 수행한다.
    a. 임계값을 초과하는 편향된 응답 수를 모니터링하고 경고함
    b. 응답을 검토하고 감사하기 위해 인간의 개입을 활용함
    c. 바람직하지 않은 결과에 대비한 벤치마킹을 수행하고 강화 학습 기법을 사용하여 다른 LLM을 훈련하기 위해 전용 LLM을 구현함
    d. LLM 생명주기 중 테스트 단계에서 LLM기반 레드팀 테스트 또는 LLM 취약점 스캔을 수행함

## Example Attack Scenarios

1. LLM 생성형 AI 프롬프트 출력은 애플리케이션 사용자를 오도할 수 있으며, 이는 편향된 의견이나 추종, 심지어 증오 범죄 등으로 이어질 수 있다.
2. 훈련 데이터가 올바르게 필터링되지 않거나 정화되지 않으면 애플리케이션의 악의적인 사용자가 모델에 악의적인 데이터를 주입하고 편향되고 거짓된 데이터에 적응하도록 시도할 수 있다.
3. 악의적인 행위자 또는 경쟁자는 입력을 기반으로 동시에 모델을 훈련하는

모델의 훈련 데이터를 타깃으로 부정확하거나 악의적인 문서를 의도적으로 생성한다. 피해 모델은 이 허위 정보를 사용하여 훈련되며, 이는 생성형 AI 프롬프트의 출력에 반영되어 소비자에게 전달된다.

4. Prompt Injection 취약점은 LLM 애플리케이션의 클라이언트 입력이 모델 훈련에 사용될 때 충분한 정화 및 필터링이 이루어지지 않으면, 이 취약점에 대한 공격 벡터가 될 수 있다. 즉, Prompt Injection 기법의 일환으로 클라이언트로부터 악의적이거나 허위 데이터가 모델에 입력되면, 이 데이터가 모델 데이터에 내재적으로 반영될 수 있다.

## Reference Links

1. Stanford Research Paper:CS324: Stanford Research
2. How data poisoning attacks corrupt machine learning models: CSO Online
3. MITRE ATLAS (framework) Tay Poisoning: MITRE ATLAS
4. PoisonGPT: How we hid a lobotomized LLM on Hugging Face to spread fake news: Mithril Security
5. Inject My PDF: Prompt Injection for your Resume: Kai Greshake
6. Backdoor Attacks on Language Models: Towards Data Science
7. Poisoning Language Models During Instruction: Arxiv White Paper
8. FedMLSecurity:arXiv:2306.04959: Arxiv White Paper
9. The poisoning of ChatGPT: Software Crisis Blog
10. Poisoning Web-Scale Training Datasets - Nicholas Carlini I Stanford MLSys #75:YouTube Video
11. OWASP CycloneDX v1.5: OWASP CycloneDX

## LLM04. Model Denial of Service (모델 서비스 거부)

### Description

공격자는 LLM과 상호작용하여 예외적으로 많은 자원을 소비하게 함으로써 잠재적으로 높은 비용을 발생시킬 뿐만 아니라 자신과 다른 사용자의 서비스 품질을 저하시킨다. 더욱이 새롭게 떠오르는 주요 보안 문제는 공격자가 LLM의 컨텍스트 창을 방해하거나 조작할 가능성이다. 이 문제는 다양한 애플리케이션에서 LLM의 사용증가, 집약적인 자원 사용, 사용자 입력의 예측 불가능성, 그리고 이 취약점에 대한 개발자들의 일반적인 인식 부족으로 인해 점점 더 심각해지고 있다. LLM에서 컨텍스트 창은 입력과 출력을 모두 포함하여 모델이 관리할 수 있는 최대 텍스트 길이를 나타낸다. 이는 모델이 이해할 수 있는 언어패턴의 복잡성과 주어진 시간에 처리할 수 있는 텍스트의 크기를 결정하는 LLM의 중요한 특성이다. 컨텍스트 창의 크기는 모델의 아키텍처에 따라 정의되며 모델마다 다를 수 있다.

### Common Examples of Vulnerability

1. LangChain이나 AutoGPT와 같은 도구를 사용하여 큐에서 대량의 작업을 생성하는 방식으로 반복적인 리소스 사용을 유도하는 쿼리를 보낸다.
2. 비정상적인 철자법이나 순서를 사용하여 비정상적으로 많은 리소스를 소비하는 쿼리를 보낸다.
3. Continuous input overflow: 공격자가 LLM에 컨텍스트 창을 초과하는 입력 스트림을 보내서 모델이 과도한 컴퓨팅 리소스를 소비하게 한다.
4. Repetitive long inputs: 공격자는 컨텍스트 창을 초과하는 긴 입력을 LLM에 반복적으로 보낸다.
5. Recursive context expansion: 공격자는 재귀적 컨텍스트 확장을 유발하는 입력을 구성하여 LLM이 반복적으로 컨텍스트 창을 확장하고 처리하도록 한다.
6. Variable-length input flood: 공격자가 컨텍스트 창의 한계에 도달하도록 신중하게 구성된 가변길이 입력을 대량으로 보내 LLM을 과부하시킨다. 이 기술은 가변 길이 입력을 처리하는 과정에서 발생할 수 있는 비효율성을 악용하여 LLM에 부담을 주고, 결국 응답하지 않게 만들 수 있다.

### Prevention and Mitigation Strategies

1. **입력 검증 및 정화:** 사용자 입력이 정의된 제한을 준수하고 악성 콘텐츠를 필터링하도록 입력 유효성 검사 및 삭제를 구현한다.
2. **요청당 리소스 사용 제한:** 복잡한 부분이 포함된 요청이 더 천천히 실행되도록 요청 혹은 단계당 리소스 사용을 제한한다.

3. **API 속도제한:** 특정 시간 내에 개별 사용자 또는 IP 주소가 요청할 수 있는 횟수를 제한하기 위해 API 속도 제한한다. .
4. **대기 작업 및 총 작업 수 제한:** LLM 응답에 반응하는 시스템에서 대기 작업 및 총 작업 수를 제한한다.
5. **리소스 사용 모니터링:** LLM의 리소스 사용량을 지속적으로 모니터링하여 DoS 공격을 나타낼 수 있는 비정상적인 스파이크 또는 패턴을 식별한다.
6. **엄격한 입력 제한 설정:** LLM의 컨텍스트 창을 기반으로 입력 제한을 설정하여 과부화와 리소스 고갈을 방지한다.
7. **개발자 인식 제고:** LLM에서 발생할 수 있는 DoS 취약점에 대한 개발자 인식을 높이고, 안전한 LLM 구현을 위한 가이드라인을 제공한다.

## Example Attack Scenarios

1. 공격자는 호스팅된 모델에 반복적으로 여러 복잡하고 비용이 많이 드는 요청 전송하여 다른 사용자의 서비스를 악화시키고 호스트의 리소스 비용을 증가시킨다.
2. LLM 기반 도구가 유해하지 않은 쿼리에 응답하기 위해 정보를 수집하는 동안 웹페이지에서 특정 텍스트를 만나게 된다. 이로 인해 도구가 더 많은 웹페이지 요청을 생성하게 되어 대량의 리소스를 소비하게 된다.
3. 공격자는 컨텍스트 창을 초과하는 입력으로 LLM을 지속적으로 공격한다. 공격자는 자동화된 스크립트나 도구를 사용하여 대량의 입력을 보내 LLM의 처리 기능을 마비시킬 수 있다. 그 결과, LLM은 과도한 컴퓨팅 리소스를 소비하게 되어 시스템이 크게 느려지거나 완전히 응답하지 않게 된다.
4. 공격자가 LLM에 컨텍스트 창의 한계 아래에 있는 연속적인 입력을 보낸다. 이러한 입력을 반복적으로 보냄으로써 공격자는 사용 가능한 컨텍스트 창의 용량을 고갈시킨다. LLM이 컨텍스트 창 내에서 각 입력을 처리하는데 어려움을 겪으면서 시스템 리소스가 과도하게 소모되어 성능 저하나 완전한 서비스 거부 상태가 발생한다.
5. 공격자는 LLM의 재귀 메커니즘을 악용하여 컨텍스트 확장을 반복적으로 발생시킨다. 공격자는 LLM의 재귀적 동작을 악용하는 입력을 구성하여 모델이 컨텍스트 창을 반복적으로 확장하고 처리하도록 함으로써 상당한 계산 리소스를 소비하게 한다. 이 공격은 시스템에 큰 부담을 주고 DoS 상태로 이어져 LLM이 응답하지 않거나 충돌을 일으킬 수 있다.
6. 공격자는 컨텍스트 창의 한계에 근접하거나 도달하도록 정교하게 설계된 가변 길이 입력을 대량으로 보내어 LLM을 넘치게 한다. 다양한 길이의 입력으로 LLM을 압도함으로써, 공격자는 가변길이 입력처리의 비효율성을 악용하려고

한다. 이러한 입력홍수로 인해 LLM 리소스에 과도한 부하가 발생하여 잠재적으로 성능 저하가 발생하고 정상적인 요청에 응답하는 능력을 방해할 수 있다.

7. DoS 공격은 일반적으로 시스템 리소스를 압도하는 것을 목표로 하지만 API 제한과 같은 시스템 동작의 다른 측면을 악용할 수도 있다. 예를 들어, 최근 Sourcegraph 보안사고에서는 악의적인 행위자가 유출된 관리 액세스 토큰을 사용하여 API 속도제한을 변경함으로써 비정상적으로 많은 요청이 발생되어 서비스가 중단되었다.

## Reference Links

1. LangChain max iterations: hwchase17 on Twitter
2. Sponge Examples: Energy-Latency Attacks on Neural Networks: Arxiv White Paper
3. OWASP DOS Attack: OWASP
4. Learning From Machines: Know Thy Context: Luke Bechtel
5. Sourcegraph Security Incident on API Limits Manipulation and DoS Attack : Sourcegraph

## LLM05. Supply Chain Vulnerabilities (공급망 취약점)

### Description

LLM의 공급망은 훈련 데이터, 머신러닝 모델, 배포 플랫폼의 무결성을 위협할 수 있는 취약점을 가지고 있다. 이러한 취약점은 편향된 결과, 보안 침해, 심지어 시스템 전체의 실패로 이어질 수 있다. 전통적으로 취약점은 소프트웨어 구성 요소에 중점을 두지만, 머신러닝은 이와 달리 변조 및 오염 공격에 취약한 제3자가 제공한 사전 훈련된 모델과 훈련 데이터를 통해 취약성을 확장한다.

마지막으로, LLM 플러그인 확장은 자체적인 취약점을 가져올 수 있다. 이러한 내용은 LLM 플러그인 작성을 다루고 타사 플러그인을 평가하는데 유용한 정보를 제공하는 LLM07 – Insecure Pluqin Design에 설명되어 있다.

### Common Examples of Vulnerability

1. 오래되거나 사용 중단된 구성 요소를 포함한 기존 타사 패키지 취약점의 경우
2. 파인튜닝에 취약한 사전 훈련된 모델을 사용하는 경우
3. 훈련에 오염된 크라우드 소싱된 데이터를 사용하는 경우
4. 더 이상 유지관리 되지 않는 구식 혹은 사용 중단 모델을 사용하여 보안 이슈를 발생시키는 경우
5. 모델 운영자의 불분명한 이용약관 및 데이터 개인 정보 보호 정책으로 인해 애플리케이션의 민감한 데이터가 모델 훈련에 사용되어 민감한 정보 노출될 수 있다. 이는 모델 공급업체가 저작권이 있는 자료를 사용하여 발생할 수 있는 위험에도 포함될 수 있다.

### Prevention and Mitigation Strategies

1. 신뢰할 수 있는 공급업체만 선정하고 T&C(소프트웨어 사용에 관한 조건과 규정) 및 개인정보 보호정책을 포함하여 데이터 소스와 공급업체를 철저히 검토한다. 적절한 보안 조치가 독립적으로 감사되도록 하고 모델 운영자의 정책이 귀하의 데이터 보호 정책과 일치하도록 보장다. 즉, 당신의 데이터가 그들의 모델 훈련에 사용되지 않는지 확인하고 모델 유지보수 관리자에게 저작권 있는 자료를 사용하는 것에 대해 보증과 법적 완화 조치를 요청한다.
2. 평판이 좋은 플러그인만 사용하고 해당 플러그인이 애플리케이션 요구 사항에 맞게 테스트되었는지 확인한다. LLM07-Insecure Plugin Design은 타사 플러그인 사용으로 인한 위험을 완화하기 위해 테스트해야 하는 LLM-aspects of Insecure Plugin 디자인에 대한 정보를 제공한다.
3. OWASP Top 10의 A06:2021 - Vulnerable and Outdated Component에서 제시한

완화 방법을 이해하고 적용한다. 여기에는 취약점 검색, 관리 및 패치 적용이 포함됩니다. 민감한 데이터에 액세스할 수 있는 개발 환경에서도 이러한 제어를 적용한다.

4. SBOM(Software Bill of Materials)을 사용하여 최신 상태의 정확하고 서명된 구성 요소 목록을 유지하여 배포된 패키지의 변조를 방지한다. SBOM은 새로운 zero-date 취약점을 신속하게 감지하고 경고하는 데 사용할 수 있다.
5. 현재 SBOM은 모델, 해당 산출물 및 데이터 세트를 다루지 않는다. LLM 애플리케이션이 자체 모델을 사용하는 경우, MLOps 모범사례를 사용하고 데이터, 모델, 실험 추적을 제공하는 안전한 모델 저장소를 활용한 플랫폼을 사용한다.
6. 외부 모델 및 공급업체를 사용할 때, 모델 및 코드 서명을 사용해야 한다.
7. 제공된 모델 및 데이터에 대한 이상 탐지 및 적대적 견고성 테스트는 Training Data Poisoning에서 설명한 대로 변조 및 오염을 탐지하는데 도움이 될 수 있다. 이상적으로 이는 MLOps 파이프라인의 일부여야 한다. 그러나 이는 새로운 기술이며 레드팀 연습의 일부로 구현하기가 더 쉬울 수 있다.
8. 구성요소와 환경 취약성 스캔, 무단 플러그인 사용, 모델 및 해당 산출물을 포함한 오래된 구성요소를 포함한 충분한 모니터링을 구현한다.
9. 취약하거나 오래된 구성요소를 완화하기 위한 패치정책을 구현한다. 애플리케이션이 유지관리 되는 API 버전과 기본 모델 버전을 사용하고 있는지 확인한다.
10. 공급업체 보안 및 액세스를 정기적으로 검토하고 감사하여 보안 상태나 이용약관가 변경되지 않도록 한다.

## Example Attack Scenarios

1. 공격자는 취약한 Python 라이브러리를 악용하여 시스템을 손상시킨다. 이는 첫 번째 OpenAI 데이터 유출 사건에서 발생했다.
2. 공격자는 항공편 검색을 위한 LLM 플러그인을 제공하여 사용자를 속이는 가짜 링크를 생성한다.
3. 공격자는 PyPi 패키지 레지스트리를 악용하여 모델 개발자를 속여서 손상된 패키지를 다운로드하고 데이터를 유출하거나 권한 상승을 유도한다. 이는 실제로 발생한 공격이다.
4. 공격자는 경제 분석 및 사회 연구를 전문으로 하는 공개된 사전 훈련 모델에 백도어를 만들어 허위정보와 가짜 뉴스를 생성한다. 공격자들은 이 모델을 모델 마켓플레이스(예: Hugging Face)에 배포하여 피해자들이 사용하도록 한다.
5. 공격자가 공개된 데이터 세트를 오염시켜 모델을 파인 튜닝할 때 백도어 생성을 돕는다. 이 백도어는 다른 시장에서 특정 회사를 유리하게 만든다.

6. 공급업체의 악의적인 직원(외주 개발자, 호스팅 회사 등)이 데이터, 모델 또는 IP를 훔치는 코드를 유출시킨다.
7. LLM 운영자는 모델 훈련을 위한 애플리케이션 데이터 사용을 명시적으로 거부하도록 T&C(소프트웨어 사용에 관한 조건과 규정) 및 개인정보 보호정책을 변경하여 민감한 데이터를 기억하게 만든다.

## Reference Links

1. ChatGPT Data Breach Confirmed as Security Firm Warns of Vulnerable Component Exploitation: Security Week
2. Plugin review process: OpenAI
3. Compromised PyTorch-nightly dependency chain: PyTorch
4. PoisonGPT: How we hid a lobotomized LLMs on Hugging Face to spread fake news: Mithril Security
5. Army looking at the possibility of 'Al BOMs: Defense Scoop
6. Failure Modes in Machine Learning: Microsoft
7. ML Supply Chain Compromise: MITRE
8. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples: Cornell University
9. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain:Cornell University
10. VirusTotal Poisoning: MITRE

## LLM06. Sensitive Information Disclosure (민감한 정보 노출)

### Description

LLM 애플리케이션은 출력을 통해 민감한 정보, 독점 알고리즘 또는 기타 기밀 세부 정보를 노출할 가능성이 있다. 이로 인해 민감한 데이터, 지적 재산권 침해, 개인 정보 침해 및 기타 보안 참해가 발생할 수 있다. LLM 응용 프로그램의 소비자는 LLM과 안전하게 상호 작용하는 방법을 알고 의도치 않게 입력한 민감한 데이터가 이후 출력에서 반환될 수 있는 위험을 인식해야 한다.

이러한 위험을 완화하기 위해 LLM 애플리케이션은 사용자 데이터가 훈련 모델 데이터에 들어가지 않도록 적절한 데이터 정화를 수행해야 한다. 또한 LLM 애플리케이션 소유자는 소비자가 자신의 데이터가 처리되는 방식과 자신의 데이터가 훈련 모델에 포함되지 않도록 선택할 수 있는 권한을 알 수 있도록 적절한 이용 약관 정책을 마련해야 한다.

소비자와 LLM 애플리케이션 간의 상호 작용은 양방향 신뢰 경계를 형성하며, 이 경계는 본질적으로 클라이언트->LLM으로의 입력 또는 LLM->클라이언트로의 출력을 신뢰할 수 없다. 이 취약점은 위협 모델링 연습, 인프라 보안 및 적절한 샌드박싱과 같은 특정 전제 조건들이 범위를 벗어났다고 가정한다는 점에 유의하는 것이 중요하다. LLM이 반환해야 하는 데이터 유형에 대해 시스템 프롬프트 내에 제한을 추가하면 민감한 정보 유출을 어느 정도 완화할 수 있지만, LLM의 예측 불가능한 특성으로 인해 이러한 제한이 항상 준수되지 않을 수 있으며, 프롬프트 인젝션이나 기타 벡터를 통해 우회될 수 있다.

### Common Examples of Vulnerability

1. LLM 응답에서 민감한 정보를 불완전하거나 부적절하게 필터링한다.
2. LLM 훈련 과정에서 민감한 데이터를 과도하게 적합하거나 기억한다.
3. LLM의 잘못된 해석, 데이터 정제 방법의 부족 또는 오류로 인해 기밀 정보가 의도치 않게 공개된다.

### Prevention and Mitigation Strategies

1. 적절한 데이터 정화와 정제 기술을 통합하여 사용자 데이터가 훈련 모델 데이터에 들어가는 것을 방지한다.
2. 모델이 오염되는 것을 방지하기 위해 잠재적인 악성 입력을 식별하고 필터링하는 강력한 입력 검증 및 정화 방법을 구현한다.
3. 데이터로 모델을 강화하고 모델을 미세 조정하는 경우: (예를 들어, 배포 전이나 배포 중에 모델에 데이터가 입력됨)

a. 파인 튜닝 데이터에서 중요하다고 간주되는 모든 정보는 사용자에게 노출될 가능성이 있다. 따라서 최소 권한 원칙을 적용하고 최고 권한 사용자가 액세스할 수 있지만 낮은 권한을 가진 사용자에게 표시될 수 있는 정보로 모델을 훈련시키지 않는다.
b. 외부 데이터 소스에 대한 액세스(런타임 시 orchestration)를 제한한다.
c. 외부 데이터 소스에 엄격한 액세스 제어 방법을 적용하고 안전한 공급망을 유지하기 위한 엄격한 접근 방식을 적용한다.

## Example Attack Scenarios

1. 의심하지 않는 합법적인 사용자 A가 악의적이지 않은 방식으로 LLM 애플리케이션과 상호 작용할 때 LLM을 통해 다른 특정 사용자 데이터에 노출된다.
2. 사용자 A는 LLM의 입력필터 및 정화를 우회하여 애플리케이션의 다른 사용자에 대한 민감한 정보(PII)를 유출하도록 잘 만들어진 프롬프트 세트를 목표로 한다.
3. PII와 같은 개인 데이터가 사용자 자신이나 LLM 애플리케이션의 부주의로 인해 Training Data를 통해 모델로 유출된다. 이 경우 위의 시나리오 1 또는 2의 위험과 확률이 높아질 수 있다.

## Reference Links

1. AI data leak crisis: New tool prevents company secrets from being fed to ChatGPT:Fox Business
2. Lessons learned from ChatGPT's Samsung leak: Cybernews
3. Cohere - Terms Of Use: Cohere
4. A threat modeling example: AI Village
5. OWASP AI Security and Privacy Guide: OWASP AI Security & Privacy Guide
6. Ensuring the Security of Large Language Models: Experts Exchange

## LLM07. Insecure Plugin Design (안전하지 않은 플러그인 디자인)

### Description

LLM 플러그인은 사용자가 상호 작용할 때 모델이 자동으로 호출하는 확장 기능이다. 모델 통합 플랫폼이 이를 구동하며, 특히 모델이 다른 당사자에 의해 호스팅 되는 경우 그렇다. 또한 플러그인은 컨텍스트 크기 제한을 처리하기 위한 유효성 검사나 유형 확인 없이 모델의 자유 텍스트 입력을 구현할 가능성이 높다. 이를 통해 잠재적 공격자는 플러그인에 대한 악의적인 요청을 구성할 수 있으며, 이로 인해 원격 코드 실행을 포함하여 다양한 원치 않는 동작이 발생할 수 있다.

악의적인 입력으로 인한 피해는 종종 불충분한 접근제어와 플러그인 간의 권한 추적 실패에 의해 발생한다. 적절한 접근제어가 없으면 플러그인은 다른 플러그인을 맹목적으로 신뢰하고 입력이 최종 사용자가 제공한 것이라고 가정할 수 있다. 이러한 불충분한 접근제어는 악의적인 입력이 데이터 유출, 원격 코드 실행, 권한 상승 등 심각한 결과를 초래할 수 있도록 한다.

이 항목은 LLM 공급망 취약점 항목에서 다루는 타사 플러그인이 아닌 LLM 플러그인 자체의 제작에 중심을 두고 있다.

### Common Examples of Vulnerability

1. 플러그인이 개별 입력 매개변수 대신 모든 매개변수를 하나의 텍스트 필드로 허용하는 경우
2. 플러그인이 전체 설정을 재정의할 수 있는 매개변수 대신 설정 문자열을 허용하는 경우
3. 플러그인은 매개변수 대신 원시 SQL 또는 프로그래밍 명령어를 허용하는 경우
4. 인증은 특정 플러그인에 대한 명시적인 인증 없이 수행하는 경우
5. 플러그인은 모든 LLM 콘텐츠를 전적으로 사용자가 만든 것으로 간주하고 추가 인증 없이 요청된 작업을 수행하는 경우

### Prevention and Mitigation Strategies

1. 플러그인은 가능한 경우 엄격한 매개변수화된 입력을 적용하고 입력에 대한 유형 및 범위 검사를 포함해야 한다. 이것이 불가능할 경우, 요청을 구문 분석하고 유효성 검사 및 정화를 적용하는 두 번째 계층의 타입화된 호출을 도입해야 한다. 애플리케이션 의미 때문에 자유 형식 입력을 허용해야 하는 경우 잠재적으로 유해한 메서드가 호출되지 않도록 신중히 검사해야 한다.
2. 플러그인 개발자는 적절한 입력 검증 및 정화를 보장하기 위해

ASVS(Application Security Verification Standard)에서 OWASP의 권장 사항을 적용해야 한다.

3. 플러그인 개발자는 적절한 입력 검증 및 정화를 보장하기 위해 ASVS(Application Security Verification Standard)에서 OWASP의 권장 사항을 적용해야 한다.
4. 적절한 검증을 보장하기 위해 플러그인을 철저하게 검사하고 테스트해야 한다. 개발 파이프라인에서 SAST(정적 애플리케이션 보안 테스트) 스캔과 DAST, IAST(동적 및 대화형 애플리케이션 테스트)를 사용한다.
5. 플러그인은 OWASP ASVS 액세스 제어 가이드라인에 따라 불안전한 입력 매개변수 악용의 영향을 최소화하도록 설계되어야 한다. 이는 원하는 기능을 수행하는 동안 가능한 한 적은 기능을 노출하면서 최소 권한 액세스 제어를 포함한다.
6. 플러그인은 효과적인 인증 및 액세스 제어를 적용하기 위해 OAuth2와 같은 적절한 인증 ID를 사용해야 한다. 또한 API키를 사용하여 기본 인터랙티브 사용자가 아닌 플러그인 경로를 반영하는 맞춤형 권한 부여 결정에 대한 컨텍스트를 제공해야 한다.
7. 민감한 플러그인이 수행하는 모든 작업에 대해 수동적인 사용자 인증 및 확인이 필요하다.
8. 플러그인은 일반적으로 REST APIs이므로 개발자는 일반적인 취약점을 최소화하기 위해 OWASP 상위 10대 API 보안 위험 - 2023에 있는 권장 사항을 적용해야 한다.

## Example Attack Scenarios

1. 플러그인은 기본 URL을 수락하고 LLM에 URL을 쿼리와 결합하여 사용자 요청 처리하는 방식으로 날씨 예보를 얻도록 지시한다. 악의적인 사용자는 URL을 자신이 제어하는 도메인으로 지정하여 해당 도메인을 통해 LLM 시스템에 자신의 콘텐츠를 삽입할 수 있다.
2. 플러그인은 유효성 검사를 하지 않는 단일 필드에 자유 형식 입력을 허용한다. 공격자는 오류 메시지에서 정찰을 수행하기 위해 신중하게 제작된 페이로드를 제공한 후, 알려진 타사 취약점을 악용하여 코드를 실행하고 데이터 유출 또는 권한 상승을 수행한다.
3. 벡터 저장소에서 임베딩을 검색하는 플러그인이 검증 없이 구성 매개변수를 연결 문자열로 허용한다. 공격자는 이름이나 호스트 매개변수를 변경하고 액세스할 수 없는 임베딩을 추출하여 다른 벡터 저장소를 실험하고 액세스할 수 있다.
4. 플러그인이 고급필터로 SQL WHERE 절을 허용하고 이를 필터링 SQL에

추가한다. 이로 인해 공격자가 SQL 공격을 수행할 수 있다.

5. 공격자는 Indirect prompt injection을 사용하여 입력 유효성 검사와 약한 액세스 제어 기능이 없는 안전하지 않은 코드 관리 플러그인을 악용함으로써 저장소 소유권을 이전하고 저장소에서 사용자를 잠근다.

## Reference Links

1. OpenAI ChatGPT Plugins: ChatGPT Developer's Guide
2. OpenAI ChatGPT Plugins - Plugin Flow: OpenAI Documentation
3. OpenAI ChatGPT Plugins - Authentication: OpenAI Documentation
4. OpenAI Semantic Search Plugin Sample: OpenAI Github
5. Plugin Vulnerabilities: Visit a Website and Have Your Source Code Stolen: Embrace The Red
6. ChatGPT Plugin Exploit Explained: From Prompt Injection to Accessing Private Data Embrace The Red
7. OWASP ASVS - 5 Validation, Sanitization and Encoding: OWASP AASVS
8. OWASP ASVS 4.1 General Access Control Design: OWASP AASVS
9. OWASP Top 10 API Security Risks - 2023: OWASP

## LLM08. Excessive Agency (과도한 대행)

### Description

LLM 기반 시스템은 종종 그 시스템 개발자로부터 어느 정도의 자율성이 부여된다. 이는 다른 시스템을 접속하고 프롬프트에 응답하여 작업을 수행할 수 있는 능력이다. 호출할 기능에 대한 결정은 입력 프롬프트 또는 LLM 출력을 기반으로 동적으로 결정하기 위해 LLM '에이전트'에 위임될 수도 있다.

과도한 대행(Excessive Agency)은 LLM의 예상치 못한/모호한 출력에 대응하여 피해를 입힐 수 있는 작업을 수행하게 하는 취약성이다. 이는 LLM의 오작동 원인(환각/거짓 정보, 직접/간접 프롬프트 인젝션, 악성 플러그인, 잘못 설계된 무해한 프롬프트, 또는 성능이 저조한 모델 등)에 관계없이 발생할 수 있다. 과도한 자율성의 근본원인은 일반적으로 과도한 기능, 과도한 권한 또는 과도한 자율성 중 하나 이상이다. 이는 LLM 출력의 불충분한 검토와 관련된 불안전한 출력 처리와 다르다.

과도한 자율성은 기밀성, 무결성 및 가용성 측면에서 다양한 영향을 미칠 수 있으며, 이는 LLM 기반 앱이 상호 작용할 수 있는 시스템에 따라 달라진다.

### Common Examples of Vulnerability

1. **과도한 기능(Excessive Functionality):**
   a. LLM 에이전트가 시스템의 의도된 운영에 필요하지 않은 기능을 포함하는 플러그인에 액세스할 수 있다. 예를 들어, 개발자가 LLM 에이전트에 저장소의 문서를 읽을 수 있는 기능을 부여해야 하지만 개발자가 선택한 타사 플러그인이 문서를 수정하고 삭제할 수 있는 기능도 포함하는 경우이다.
   b. 개발 단계에서 실험적으로 사용되었다가 더 나은 대안으로 대체된 플러그인이 LLM 에이전트에 여전히 남아있는 경우이다.
   c. 개방형 기능을 가진 LLM 플러그인이 애플리케이션의 의도된 운영에 필요한 명령 외의 명령을 적절히 필터링하지 못하는 경우이다. 예를 들어 하나의 특정 셸 명령을 실행하기 위한 플러그인이 다른 셸 명령의 실행을 적절하게 차단하지 못하는 경우이다.
2. **과도한 권한(Excessive Permissions):**
   a. LLM 플러그인이 애플리케이션의 의도된 작동에 필요하지 않은 다른 시스템에 대한 권한을 가지고 있는 경우이다. 예를 들어, 데이터를 읽기 위한 플러그인이 SELECT 권한뿐만 아니라 UPDATE, INSERT 및 DELETE 권한도 가진 ID를 사용하여 데이터베이스 서버에 연결하는 경우이다.
   b. 사용자를 대신하여 작업을 수행하도록 설계된 LLM 플러그인이 일반적인

높은 권한의 ID를 사용하여 하위 시스템에 액세스한다. 예를 들어, 현재 사용자의 문서 저장소를 읽기 위한 플러그인이 모든 사용자의 파일에 액세스할 수 있는 높은 권한의 계정으로 문서 저장소에 연결하는 경우이다.

3. **과도한 자율성(Excessive Autonomy):**
   a. LLM기반 애플리케이션 또는 플러그인이 영향력이 큰 작업을 독립적으로 확인하고 승인하지 못하는 경우이다. 예를 들어, 사용자의 문서를 삭제할 수 있는 플러그인이 사용자의 확인 없이 삭제를 수행하는 경우이다.

## Prevention and Mitigation Strategies

다음 조치는 과도한 자율성을 방지할 수 있다.

1. LLM 에이전트가 필요한 최소한의 기능만 호출할 수 있는 플러그인/도구로 제한한다. 예를 들어 LLM 기반 시스템이 URL의 콘텐츠를 가져올 필요가 없다면, 그런 플러그인은 LLM 에이전트에게 제공되지 않아야 한다.
2. LLM 플러그인/도구에 구현되는 기능을 필요한 최소한으로 제한한다. 예를 들어 이메일을 요약하기 위해 사용자의 편지함에 액세스하는 플러그인은 이메일을 읽는 기능만 필요하다. 따라서, 이 플러그인에는 메시지 삭제 또는 전송과 같은 다른 기능이 포함되지 않아야 한다.
3. 가능하면 개방형 기능(예: 셸 명령 실행, URL 가져오기 등)을 피하고 더 세부적인 기능을 가진 플러그인/도구를 사용한다. 예를 들어 LLM기반 앱이 일부 출력을 파일에 작성할 필요가 있을 때, 셸 기능을 실행하는 플러그인을 사용하여 이것이 구현된 경우, 바람직하지 않은 작업의 범위가 매우 커질 수 있다(즉, 다른 셸 명령이 실행될 수 있음) 더 안전한 대안은 파일 쓰기 기능만 지원하는 플러그인을 만드는 것이다.
4. LLM 플러그인/도구에 부여된 다른 시스템에 대한 권한을 최소한으로 제한하여 원치 않는 작업의 범위를 제한한다. 예를 들어, 고객에게 구매 추천을 하기 위해 제품 데이터베이스를 사용하는 LLM 에이전트는 'products' 테이블에 대한 읽기 권한만 필요하다. 다른 테이블에 대한 액세스나 레코드 삽입, 업데이트 또는 삭제하는 권한은 필요하지 않다. 이는 LLM 플러그인이 데이터베이스에 연결하는 ID에 적절한 데이터베이스 권한을 적용하여 시행되어야 한다.
5. 사용자 권한부여 및 보안 범위를 추적하여 사용자를 대신하여 수행되는 작업이 해당 특정 사용자의 컨텍스트에서 필요한 최소한의 권한으로 하위 시스템에서 실행되도록 한다. 예를 들어, 사용자의 코드 저장소를 읽는 LLM 플러그인은 사용자가 OAuth를 통해 필요한 최소 범위로 인증하도록 요구해야 한다.
6. Human-In-The-Loop (인간지능과 인공지능을 모두 활용) 제어를 활용하여 인간이

모든 작업을 수행하기 전에 사람이 이를 승인하도록 요구한다. 이는 하위 시스템(LLM 애플리케이션의 범위를 벗어난 하위 시스템) 또는 LLM 플러그인/도구 자체 내에서 구현될 수 있다. 예를 들어, 사용자를 대신하여 소셜 미디어 콘텐츠를 생성하고 게시하는 LLM 기반 앱은 '게시' 작업을 수행하는 플러그인/도구/API 내에 사용자 승인 절차를 포함해야 한다.

7. LLM이 행동을 허용할지 여부를 결정하는 대신 하위 시스템에서 인증을 구현한다. 도구/플러그인을 구현할 때는 모든 요청이 보안 정책에 따라 검증되도록 완전 중재 원칙을 적용한다. 즉, 플러그인/도구를 통해 하위 시스템으로 보내는 모든 요청이 철저히 검토되고 검증되도록 해야 한다.

다음 옵션은 과도한 대행을 방지할 수는 없지만 발생하는 피해 수준을 제한할 수 있다.

1. LLM 플러그인/도구 및 하위 시스템의 활동을 기록하고 모니터링하여 바람직하지 않은 작업이 발생하는 위치를 식별하고 그에 따라 대응한다.
2. 특정 기간 내에 발생할 수 있는 바람직하지 않은 작업 수를 줄이기 위해 속도 제한을 구현하고, 심각한 피해가 발생하기 전에 모니터링을 통해 바람직하지 않은 작업을 발견할 수 있는 기회를 증가시킨다.

## Example Attack Scenarios

LLM기반 개인비서 앱이 플러그인을 통해 개인의 메일함에 접근하여 수신 이메일의 내용을 요약한다. 이 기능을 수행하기 위해 이메일 플러그인은 메시지를 읽을 수 있어야 하지만, 시스템 개발자가 선택한 플러그인에는 메시지 전송 기능도 포함되어 있다. LLM은 Indirect prompt injection 공격에 취약하여, 악의적으로 작성된 수신 이메일이 LLM을 속여 이메일 플러그인이 '메시지 전송' 기능을 호출하게 하여 사용자의 메일함에서 스팸을 전송하도록 할 수 있다. 이 문제를 방지하는 방법 (a) 메일 읽기 기능만 제공하는 플러그인을 사용하여 과도한 기능을 제거한다, (b) 읽기 전용 범위의 OAuth 세션을 통해 사용자의 이메일 서비스에 인증하여 과도한 권한을 제거한다. 또는 (c) LLM 플러그인이 작성한 모든 메일을 사용자가 수동으로 검토하고 '보내기'를 눌러야만 메일 전송이 되도록 요구한다. 또는 메일 전송 인터페이스에 속도제한을 구현하여 피해를 줄일 수 있다.

## Reference Links

1. Embrace the Red: Confused Deputy Problem: Embrace The Red
2. NeMo-Guardrails: Interface guidelines: NVIDIA Github
3. LangChain: Human-approval for tools: Langchain Documentation
4. Simon Willison: Dual LLM Pattern: Simon Willison

## LLM09. Overreliance (과도한 의존)

### Description

LLM이 잘못된 정보를 생성하고 이를 권위 있는 방식으로 제공할 때 과도한 의존이 발생할 수 있다. LLM은 창의적이고 유익한 콘텐츠를 생성할 수 있지만 사실이 아닌 정보, 부적절하거나 안전하지 않은 콘텐츠를 생성할 수도 있다. 이것을 환각 또는 조작이라고 한다. 사람이나 시스템이 확인하거나 감독하지 않고 이 정보를 신뢰하면 보안 침해, 잘못된 정보, 잘못된 의사소통, 법적 문제 및 평판 손상이 발생할 수 있다.

LLM이 생성한 소스코드는 눈에 띄지 않는 보안 취약점을 유발할 수 있다. 이것은 애플리케이션의 운영 안전과 보안에 심각한 위험을 가져올 수 있다. 이러한 위험은 다음과 같은 철저한 검토 프로세스의 중요성을 보여준다.

- 감독 (Oversight)
- 지속적인 검증 메커니즘 (Continuous validation mechanisms)
- 위험에 대한 고지 (Disclaimers on risk)

### Common Examples of Vulnerability

1. LLM은 응답으로 부정확한 정보를 제공하면서 매우 권위 있는 방식으로 이를 제시한다. 전체적인 시스템은 이를 처리하기 위한 적절한 견제와 균형이 없이 설계된 경우, 그 정보가 사용자를 잘못된 방향으로 이끌어 해를 입힐 수 있다.
2. LLM은 안전하지 않거나 결함이 있는 코드를 제안하여, 이를 적절한 감독이나 검증 없이 소프트웨어 시스템에 통합하면 취약점이 발생할 수 있다.

### Prevention and Mitigation Strategies

1. LLM 결과를 주기적으로 모니터링하고 검토한다. 일관성이 없는 텍스트를 필터링하기 위해 자체 일관성이나 투표 기술을 사용해야 한다. 단일 프롬프트에 대한 여러 모델 응답을 비교하면 출력의 품질과 일관성을 더 잘 판단할 수 있다.
2. 신뢰할 수 있는 외부 소스를 통해 LLM 출력을 교차 확인한다. 이러한 추가 검증 레이어는 모델이 제공하는 정보가 정확하고 신뢰할 수 있는지 확인하는데 도움이 된다.
3. 출력 품질을 향상시키기 위해 파인 튜닝이나 임베딩을 사용한다. 일반적인 사전 훈련된 모델은 특정 도메인에서 튜닝된 모델에 비해 부정확한 정보를 생성할 가능성이 더 높다. 프롬프트 엔지니어링, 파라미터 효율 튜닝(PET), 전체 모델 튜닝, chain of thought prompting과 같은 기술을 사용할 수 있다.
4. 생성된 출력을 알려진 사실이나 데이터와 교차 검증할 수 있는 자동 검증 메커니즘을 구현한다. 이를 통해 추가 보안 계층을 제공하고 환각과 관련된

위험을 완화할 수 있다.

5. 복잡한 작업을 관리 가능한 하위 작업으로 나누고 이를 다른 에이전트에게 할당한다. 이는 복잡성을 관리하는데 도움이 될 뿐만 아니라 각 에이전트가 더 작은 작업에 대해 책임을 질 수 있으므로 환각 가능성도 줄일 수 있다.
6. LLM 사용과 관련된 위험과 제한사항을 명확하게 전달한다. 여기에는 정보 부정확성 및 기타 위험 가능성이 포함된다. 효과적인 위험 커뮤니케이션은 사용자가 잠재적인 문제에 대비하고 정보에 입각한 결정을 내리는데 도움이 된다.
7. LLM의 책임감 있고 안전한 사용을 장려하는 API 및 사용자 인터페이스를 구축한다. 여기에는 콘텐츠 필터, 잠재적인 부정확성에 대한 사용자 경고, AI 생성 콘텐츠에 대한 명확한 라벨링과 같은 조치가 포함될 수 있다.
8. 개발 환경에서 LLM을 사용할 때 가능한 취약점의 통합을 방지하기 위해 안전한 코딩 관행과 지침을 확립한다.

## Example Attack Scenarios

1. 한 뉴스 조직에서 뉴스 기사를 생성하기 위해 LLM을 많이 사용한다. 악의적인 행위자는 이러한 과도한 의존을 이용하여 LLM에 잘못된 정보를 제공하고 허위 정보를 확산시킨다.
2. AI가 의도치 않게 기존 콘텐츠를 표절하여 저작권 문제를 일으키고 조직에 대한 신뢰를 떨어뜨린다.
3. 소프트웨어 개발팀은 LLM 시스템을 활용하여 코딩 프로세스를 가속화한다. AI의 제안에 지나치게 의존하면 불안전한 기본 설정이나 안전한 코딩 관행과 일치하지 않는 권장 사항으로 인해 애플리케이션에 보안 취약점이 발생한다.
4. 소프트웨어 개발회사는 LLM을 사용하여 개발자를 어시스트한다. LLM이 존재하지 않는 코드 라이브러리나 패키지를 제안하고, 개발자가 AI를 신뢰하여 알지 못한 채 악성 패키지를 회사의 소프트웨어에 통합한다. 이는 특히 타사 코드나 라이브러리를 포함할 때 LLM 제안을 교차 검증하는 것이 얼마나 중요한지를 강조한다.

## Reference Links

1. Understanding LLMs Hallucinations: Medium

2. How Should Companies Communicate the Risks of Large Language Models to Users?:Tech Policy Press

3. A news site used Al to write articles. It was a journalistic disaster: The Washington Post

4. AI Hallucinations: Package Risk: Vulcan
5. How to Reduce the Hallucinations from Large Language Models: The New Stack
6. Practical Steps to Reduce Hallucination: Designing With Machine Learning

## LLM10. Model Theft (모델 도난)

### Description

이 항목은 악의적인 행위자 또는 APT에 의한 LLM 모델의 무단 액세스 및 유출을 나타낸다. 이는 가치 있는 지적 재산인 독점 LLM 모델이 손상되거나 물리적으로 도난당하거나 복사되거나 가중치 및 파라미터가 추출되어 기능적으로 동등한 모델을 생성할 때 발생한다. LLM 모델 도용의 영향에는 경제적 손실과 브랜드 평판 손실, 경쟁 우위의 악화, 모델의 무단 사용 또는 모델에 포함된 민감한 정보에 대한 무단 액세스가 포함될 수 있다.

언어 모델이 점점 더 강력해지고 보편화됨에 따라 LLM 도난은 중요한 보안 문제를 나타낸다. 조직과 연구자는 LLM 모델을 보호하기 위해 강력한 보안 조치를 우선시하고 지적 재산의 기밀성과 무결성을 보장해야 한다. 액세스 제어, 암호화 및 지속적인 모니터링을 포함하는 종합적인 보안 프레임워크를 사용하는 것은 LLM 모델 도난과 관련된 위험을 완화하고 LLM에 의존하는 개인과 조직 모두의 이익을 보호하는데 중요하다.

### Common Examples of Vulnerability

1. 공격자는 회사 인프라의 취약성을 악용하여 네트워크 또는 애플리케이션 보안 설정의 잘못된 구성을 통해 LLM 모델 저장소에 대한 무단 액세스한다.
2. 프로덕션에 사용되는 ML 모델에 대해 중앙집중식 ML 모델 인벤토리 또는 레지스트리를 사용한다. 중앙 집중식 모델 레지스트리를 통해 접근제어, 인증 및 모니터링/로그 기능을 통해 ML 모델에 대한 무단 접근을 방지할 수 있으며, 이는 거버넌스를 위한 좋은 기반이 된다. 중앙 집중식 저장소를 통해 규정 준수, 위험 평가 및 위험 완화를 목적으로 모델에서 사용되는 알고리즘에 대한 데이터를 수집하는 데에도 유용하다.
3. 불만을 품은 직원이 모델 또는 관련 산출물를 유출하는 내부자 위협 시나리오이다.
4. 공격자는 신중하게 제작된 입력과 prompt injection 기술을 사용해서 모델 API에 쿼리하여 충분한 양의 출력을 수집함으로써 섀도우 모델을 생성한다.
5. 악의적인 공격자가 LLM의 입력 필터링 기술을 우회하여 사이드 채널 공격을 수행하고 궁극적으로 모델 가중치 및 원격 제어 리소스에 대한 아키텍처 정보를 수집할 수 있다.
6. 모델 추출을 위한 공격 벡터에는 특정 주제에 대한 많은 수의 프롬프트를 사용하여 LLM에 쿼리를 보내는 것이 포함됩니다. LLM에서 생성된 출력은 다른 모델을 파인튜닝하는 데 사용할 수 있다. 그러나 이 공격과 관련하여 몇 가지

주의할 점이 있다.

- 공격자는 다수의 타겟 프롬프트를 생성해야 한다. 프롬프트가 충분히 구체적이지 않으면 LLM의 출력은 쓸모가 없어진다.
- LLM의 출력에는 때때로 환각적인 답변이 포함될 수 있으며, 이는 공격자가 전체 모델을 추출하지 못하고 일부 출력이 비논리적일 수 있다는 것을 의미한다.
- 모델 추출을 통해 LLM을 100% 복제하는 것은 불가능하지만 공격자는 부분적으로 모델을 복제할 수 있다.

7. 기능 모델 복제에 대한 공격벡터에는 프롬프트를 통해 대상 모델을 사용하여 합성 훈련 데이터를 생성하는 방법("자기 지도(self-instruct)"라는 접근법)을 포함한다. 그런 다음 이 데이터를 사용하여 다른 기본 모델을 파인 튜닝하여 기능적으로 동등한 모델을 생성한다. 이는 Example 5에서 사용된 기존 쿼리 기반 추출의 한계를 우회하며, LLM을 사용하여 다른 LLM을 훈련시키는 연구에 성공적으로 사용되었다. 비록 이 연구맥락에서 모델 복제가 공격이 아닌 것으로 간주되지만, 이 접근법은 공격자가 공용 API를 사용하여 독점 모델을 복제하는데 사용할 수 있다.

도난된 모델을 섀도우 모델로 사용하면 적대적 공격을 수행할 수 있다. 모델 내에 포함된 민감한 정보에 대한 무단 액세스하는 것 또는 적대적 입력을 탐지하지 않고 실험하여 더 고급화된 prompt injection 공격을 준비할 수 있다.

## Prevention and Mitigation Strategies

1. 강력한 액세스 제어(예: RBAC 및 최소 권한 원칙)와 강력한 인증 메커니즘을 구현하여 LLM 모델 저장소 및 훈련 환경에 대한 무단 액세스를 제한한다.
    a. 이는 악의적인 행위자가 내부 또는 외부 환경으로부터 침투할 수 있는 LLM 모델, 가중치 및 아키텍처를 수용하는 인프라에 대한 내부 위협, 잘못된 구성 또는 약한 보안 제어로 인해 이 취약성을 유발할 수 있는 처음 세 가지 일반적인 예의 경우 특히 그렇다.
    b. 공급망 공격을 방지하기 위해 공급업체 관리 추적, 검증 및 종속성 취약성을 중점적으로 다루는 것이 중요하다.
2. 네트워크 리소스, 내부 서비스 및 APIs에 대한 LLM의 액세스를 제한한다.
    a. 이는 내부 위험과 위협을 다루지며, 궁극적으로 LLM 애플리케이션이 "액세스할 수 있는" 범위를 제어하여 사이드 채널 공격을 방지하기 위한 메커니즘 또는 예방 단계가 될 수 있으므로 모든 일반적인 예에 해당된다.

3. LLM 모델 저장소와 관련된 액세스 로그 및 활동을 정기적으로 모니터링하고 감사하여 의심스럽거나 승인되지 않은 동작을 즉시 감지하고 대응한다.
4. 거버넌스, 추적 및 승인 워크플로를 통해 MLOps 배포를 자동화하여 인프라 내 액세스 및 배포 제어를 강화한다.
5. 프롬프트 인젝션 기법으로 인한 사이드 채널 공격의 위험을 완화하거나 줄이기 위해 통제 및 완화 전략을 구현한다.
6. 적용 가능한 경우 API 호출의 속도 제한 및 필터를 사용하여 LLM 애플리케이션에서 데이터 유출 위험을 줄이거나 다른 모니터링 시스템에서 추출 활동을 감지(예: OLP)하는 기술을 구현한다.
7. 추출 쿼리를 감지하고 물리적 보안 조치를 강화하기 위해 견고한 훈련을 구현한다.
8. LLM 생명주기의 임베딩 및 감지단계에 워터마킹 프레임워크를 구현한다.

## Example Attack Scenarios

1. 공격자는 회사 인프라의 취약성을 악용하여 LLM 모델 저장소에 대한 무단 액세스 권한을 얻는다. 공격자는 귀중한 LLM 모델을 유출하여 이를 통해 경쟁 언어 처리 서비스를 시작하거나 민감한 정보를 추출하여 원래 회사에 심각한 재정적 피해를 입힐 수 있다.
2. 불만을 품은 직원이 모델이나 관련 산출물을 유출한다. 이 시나리오가 공개되면 공격자가 그레이박스를 위한 적대적 공격에 대한 지식을 얻거나 사용 가능한 속성을 직접 훔칠 수 있다.
3. 공격자는 신중하게 선택한 입력값으로 API를 쿼리를 보내고 충분한 수의 출력을 수집하여 섀도우 모델을 생성한다.
4. 공급망 내에 보안 제어 실패가 발생하여 독점 모델 정보의 데이터 유출로 이어진다.
5. 악의적인 공격자는 LLM의 입력 필터링 기술과 서문을 우회하여 사이드 채널 공격을 수행하고 모델 정보를 원격으로 제어되는 리소스로 가져간다.

## Reference Links

1. Meta's powerful Al language model has leaked online: The Verge
2. Runaway LlaMA I How Meta's LlaMA NLP model leaked: DeepLearning.ai
3. AML.TA0000 ML Model Access: MITRE ATLAS
4. I Know What You See: Cornell University
5. D-DAE: Defense-Penetrating Model Extraction Attacks: IEEE
6. A Comprehensive Defense Framework Against Model Extraction Attacks: IEEE

7. Alpaca: A Strong, Replicable Instruction-Following Model: Stanford University
8. How Watermarking Can Help Mitigate The Potential Risks Of LLMs?: KD Nuggets

# Etc - Core Team Members are listed in Blue

| | | | | | |
|---|---|---|---|---|---|
| Adam Swanda | | Emmanuel Guilherme Junior | McMaster University | Manjesh S | HackerOne |
| Adesh Gairola | AWS | Eugene Neelou | | Manuel S. Lemos | BeDisruptive \| ANBAN |
| Ads Dawson | Cohere | Eugene Tawiah | Complex Technologies | Mehdi Karimi | |
| Adrian Culley | Trellix | Fernando Calderon | Unit42 Palo Alto Networks | Mike Finch | HackerOne |
| Aleksei Ryzhkov | EPAM | Gaurav "GP" Pal | stackArmor | Mike Jang | Forescout |
| Alexander Zai | | Gavin Klondike | AI Village | Nathan Hamiel | Kudelski Security |
| Allaksei Bialko | EPAM | Guillaume Ehinger | Google | Natsu Ooke | Aeye Security Lab Inc. |
| Amay Trivedi | | Idan Hen | Microsoft | Neal Swaelens | LLM Guard |
| Ananda Krishna | Astra Security | Itamar Golan | | Nicholas Grove | Security, AI, Microsoft |
| Andrea Succi | | James Rabe | IriusRisk | Nipun Gupta | Bearer |
| Andrew Amaro | Klavan Security Group | Jason Axley | AWS | Nir Paz | |
| Andy Dyrcz | Linkfire | Jason Haddix | BuddoBot | Oleksandr Yaremchuk | LLM Guard |
| Andy Smith | | Jason Ross | Salesforce | Ophir Dror | |
| Ashish Rajan | Cloud Security Podcast | Jeff Williams | Contrast Security | Otto Sulin | Nordic Venture Family |
| Aubrey King | FS DevCentral | Jinson Varghese Behanan | Astra Security | Parveen Yadav | HackerOne |
| Bajram Hoxha | Databook | Johann Rehberger | | Patrick Biyaga | Theravigo |
| Bilal Siddiqui | Trustwave | John Sofiropoulos | Kainos | Priyadharshini Parthasarathy | Coalfire |
| Bob Simonoff | Blue Yonder | Jorge Pinto | | Rachit Sood | |
| Brian Pendleton | AVID | Joshua Nussbaum | | RahulZhade | GitHub |
| Brodie McRae | AWS | Ken Arora | FS | Ravi Singh | AddValueMachine Inc |
| Cassio Goldschmidt | ServiceTitan | Ken Huang | DistributedApps.ai | Ravin Thambapillai | CredaLai |
| Dan Frommer | | Kelvin Low | aigos | Reza Rashidi | HADESS |
| Dan Klein | Accenture | Koichiro Watanabe | Microsoft | Rich Harang | NVIDIA |
| David Rowe | AWS | Larry Carson | | Riotaro OKADA | OWASP JAPAN, Asterisk Research |
| David Taylor | | Leon Derczynski | U of W, ITU of Copenhagen | Ron F. Del Rosario | SAP |
| Dotan Nahum | Check Point | Leonardo Shikida | IBM | Ross Moore | Security Researcher and Writer |
| Dr. Matteo GroBe-Kampmann | AWARE7 | Lior Drihem | | Sandy Dunn | |
| Emanuel Valente | iFood | Manish Kumar Yadav | SAP | Sanket Naik | Palosade |

| | | | |
|---|---|---|---|
| Santosh Kumar | Cisco | | |
| Sarah Thornton | Red Hat | | |
| Stefano Amorelli | | | |
| Steve Wilson | Contrast Security | | |
| Talesh Seeparsan | Bit79 | | |
| Vandana Verma Sehgal | fu1Ys | | |
| Vinay Vishwanatha | Sprinklr | | |
| Vishwas Manral | | | |
| Vladimir Fedotov | | | |
| Will Chilcutt | Yahoo | | |
| Yevhenii Molchanov | SumUp | | |
| Yusuke Kawasawa | Japan Digital Design, Inc. | | |
| | | | |