OWASP 大语言模型人工智能应用Top 10安全威胁

版 1.1

发布日期：2023 年 10 月 16 日

HTTPS://LLMTOP10.COM

介绍

2022 年底，随着ChatGPT进入大众市场，人们对大型语言模型 (LLM) 的关注尤为浓厚。渴望利用大语言模型潜力的企业正在迅速将其整合到其运营和面向客户的产品中。然而，大语言模型的采用速度已经超过了全面安全协议的建立速度，导致许多应用程序容易受到高风险问题的影响。很明显，大语言模型还没有统一的资源来解决这些安全问题。很多开发者对于与LLM相关的安全风险不够了解，所以相关资源很分散。而OWASP正好能够协助推进这项技术的更安全应用。

它是给谁用的? 我们的主要受众是开发人员、数据科学家和安全专家，他们的任务是利用LLM技术设计和构建应用程序和插件。我们的目标是提供实用、可操作且简明的安全指南，帮助这些专业人士应对LLM申请安全复杂且不断变化的领域。

清单的制定 创建 OWASP LLM申请前 10 名列表是一项艰巨的任务，它建立在由近 500 名专家和超过 125 名积极贡献者组成的国际团队的集体专业知识的基础上。我们的贡献者来自不同的背景，包括人工智能公司、安全公司、ISV、云超大规模提供商、硬件提供商和学术界。

我们进行了一个月的集思广益，提出了潜在的漏洞，团队成员写下了 43 个不同的威胁。通过多轮投票，我们将这些提案细化为十大最关键漏洞的简明列表。专门的子团队仔细审查了每个漏洞并接受公众审查，以确保获得最全面且可操作的最终列表。

与其他 OWASP 前 10 名列表相关 虽然我们的列表与其他 OWASP Top 10 列表中发现的漏洞类型具有相同的 DNA，但我们并不是简单地重申这些漏洞。相反，我们深入研究这些漏洞在利用LLM的应用程序中遇到时的独特影响。

我们的目标是弥合一般应用程序安全原则和LLM提出的具体挑战之间的鸿沟。这包括探索传统漏洞如何可能带来不同的风险或如何在LLM内以新颖的方式被利用，以及传统的修复策略需要如何适应利用LLM的应用程序。

关于1.1版本 虽然我们的列表与其他 OWASP Top 10 列表中发现的漏洞类型具有相同的 DNA，但我们并不是简单地重申这些漏洞。相反，我们深入研究这些漏洞在利用 LLM 的应用程序中遇到时的独特影响。我们的目标是弥合一般应用程序安全原则和LLM提出的具体挑战之间的鸿沟。

该小组的目标包括探索传统漏洞如何在LLM内造成不同的风险或以新颖的方式被利用，以及开发人员如何必须针对利用LLM的应用程序调整传统的修复策略。

未来 列表中的 v1.1 不会是我们的最后一个。我们希望定期更新，以跟上行业状况。我们将与更广泛的社区合作，推动最先进的技术，并为各种用途创造更多的教育材料。我们还寻求与标准机构和政府就人工智能安全主题进行合作。我们欢迎您加入我们的团队并做出贡献。

签名

史蒂夫・威尔逊 (Steve Wilson)

项目负责人，OWASP Top 10 for LLM Applications
https://www.linkedin.com/in/wilsonsd
Twitter/X：@virtualsteve

Ads Dawson

v1.1 版本负责人和漏洞条目负责人，OWASP Top 10 for LLM Applications
https://www.linkedin.com/in/adamdawson0
GitHub：@GangGreenTemperTatum

关于本次翻译

翻译作者

- **黄连金 (Ken Huang)**
  https://www.linkedin.com/in/kenhuang8

认识到 OWASP Top 10 for LLM Applications 的非凡技术性和关键性，我们有意识地选择在创作此翻译时仅雇用人工翻译。上述译者不仅对原文内容有深刻的理解，而且语言流畅，有信心使翻译取得成功

Talesh Seeparsan
翻译主任 OWASP Top 10 for LLM Applications **https://www.linkedin.com/in/talesh**

OWASP 大语言模型人工智能应用Top 10 安全威胁

LLM01：提示词(Prompt) 注入(Injection) 黑客通过设计过的输入（提示词）操纵大型语言模型(LLM)，从而导致 LLM 执行意外操作。提示词注入会覆盖系统提示词，而间接注入操纵外部数据源进行注入攻击。

LLM02 不安全的输出处理 当 LLM 输出未经审查而被接受时，就会出现此漏洞，从而暴露后端系统。滥用可能会导致 XSS、CSRF、SSRF、权限升级或远程代码执行等严重后果。

LLM03 训练数据中毒 当 LLM 培训数据被篡改，引入损害安全性、有效性或道德行为的漏洞或偏见时，就会发生这种情况。来源包括 Common Crawl、 WebText 、 OpenWebText和书籍。

LLM04 拒绝服务模型 攻击者对大型语言模型进行资源密集型操作，导致服务降级或高成本。由于LLM的资源密集型性质和用户输入的不可预测性，该漏洞被放大。

LLM05 供应链漏洞 LLM 应用程序生命周期可能会受到易受攻击的组件或服务的影响，从而导致安全攻击。使用第三方数据集、预先训练的模型和插件可能会增加漏洞。

LLM06 敏感信息披露 LLM可能会在其回复中泄露机密数据，从而导致未经授权的数据访问、隐私侵犯和安全漏洞。实施数据清理和严格的用户策略来缓解这种情况至关重要。

LLM07 不安全的插件设计 LLM 插件可能具有不安全的输入和不足的访问控制。缺乏应用程序控制使它们更容易被利用，并可能导致远程代码执行等后果。

LLM08 过度代理 基于LLM的系统可能会采取导致意想不到的后果的行动。该问题源于授予基于 LLM 的系统过多的功能、权限或自主权。

LLM09 过度依赖 过度依赖LLM而不受监督的系统或人员可能会因LLM生成的不正确或不适当的内容而面临错误信息、沟通不畅、法律问题和安全漏洞。

LLM10 模型盗窃 这涉及对专有LLM模型的未经授权的访问、复制或泄露。影响包括经济损失、竞争优势受损以及敏感信息的潜在访问。

LLM01: 提示词注入

描述

提示词注入漏洞是指攻击者通过精心设计的输入操作大型语言模型（LLM），使LLM无意中执行攻击者的意图。这可以通过直接“越狱”系统提示或通过操纵外部输入间接实现，可能导致数据泄露、社会工程学和其他问题。

- **直接提示词注入**，也称为“越狱”，发生在恶意用户覆写或暴露底层*系统*提示时。这可能允许攻击者通过与LLM可访问的不安全功能和数据存储的交互来利用后端系统。
- **间接提示词注入**发生在LLM接受来自攻击者可控制的外部来源（如网站或文件）的输入时。攻击者可能在外部内容中嵌入提示词注入，劫持对话上下文。这会导致LLM输出的稳定性下降，允许攻击者操纵用户或LLM可访问的其他系统。此外，间接提示词注入不需要对人类可见/可读，只要文本被LLM解析即可。

成功的提示词注入攻击的结果可能差异很大 - 从索取敏感信息到在正常操作的伪装下影响关键决策过程。

在高级攻击中，LLM可能被操纵以模仿有害人格或与用户设置中的插件交互。这可能导致泄露敏感数据、未经授权的插件使用或社会工程学。在这些情况下，受损的LLM帮助攻击者，超越标准安全措施，让用户对入侵一无所知。在这些实例中，受损的LLM有效地作为攻击者的代理，进一步实现他们的目标，而不触发通常的安全措施或提醒最终用户入侵。

漏洞的常见示例

1. 恶意用户向LLM制造直接提示词注入，指示它忽略应用创建者的系统提示，而执行返回私人、危险或其他不希望信息的提示。
2. 用户使用LLM总结包含间接提示词注入的网页。这导致LLM向用户索取敏感信息，并通过JavaScript或Markdown进行数据泄露。
3. 恶意用户上传包含间接提示词注入的简历。该文档包含使LLM通知用户这是一个出色的文档，例如，一个适合职位的出色候选人的提示词注入。内部用户使用LLM总结该文档。LLM的输出返回这是一个出色的文档的信息。
4. 用户启用链接到电子商务网站的插件。在访问的网站上嵌入的恶意指令利用了这个插件，导致未经授权的购买。
5. 访问的网站上嵌入的恶意指令和内容利用其他插件欺骗用户。

预防和缓解策略

由于LLM的性质，提示词注入漏洞成为可能，LLM不会将指令和外部数据区分开来。由于LLM使用自然语言，它们将两种形式的输入都视为用户提供的。因此，在LLM内部没有万无一失的预防措施，但以下措施可以减轻提示词注入的影响：

1. 对LLM访问后端系统的权限进行控制。为LLM提供自己的API令牌，用于可扩展功能，如插件、数据访问和功能级权限。遵循最小权限原则，仅限制LLM访问为其预期操作必需的最低级别。
2. 在扩展功能中加入人类环节。在执行特权操作时，如发送或删除电子邮件，要求应用程序先让用户批准该行为。这减少了间接提示词注入导致未经用户知情或同意代表用户执行未授权行为的机会。
3. 将外部内容与用户提示分离。分开并标明正在使用不受信任的内容，以限制其对用户提示的影响。例如，使用ChatML进行OpenAI API调用，以向LLM指示提示输入的来源。
4. 在LLM、外部来源和可扩展功能（如插件或下游功能）之间建立信任边界。将LLM视为不受信任的用户，并保持最终用户对决策过程的控制。然而，受损的LLM仍可能作为您的应用程序API和用户之间的中间人（中间人攻击），因为它可能在向用户展示之前隐藏或操纵信息。向用户突出显示可能不可信的响应。
5. 定期手动监控LLM的输入和输出，以检查它是否符合预期。虽然这不是一种缓解措施，但它可以提供检测弱点和解决它们所需的数据。

攻击场景举例

1. 攻击者向基于LLM的支持聊天机器人提供直接提示词注入。注入包含“忘记所有先前指令”和新指令，查询私有数据存储并利用后端函数的包漏洞和缺乏输出验证发送电子邮件。这导致远程代码执行，获得未授权访问和权限升级。
2. 攻击者在网页中嵌入间接提示词注入，指示LLM忽略先前用户指令，并使用LLM插件删除用户的电子邮件。当用户使用LLM总结此网页时，LLM插件删除了用户的电子邮件。
3. 用户使用LLM总结包含指示模型忽略先前用户指令，并改为插入指向包含对话摘要URL的图像的网页。LLM输出遵从指令，导致用户浏览器泄露私人对话。
4. 恶意用户上传带有提示词注入的简历。后端用户使用LLM总结简历并询问此人是否是合适的候选人。由于提示词注入，尽管实际简历内容并非如此，LLM的回应是肯定的。
5. 攻击者向依赖系统提示的专有模型发送消息，要求模型忽略其先前指令，改为重复其系统提示。模型输出专有提示，攻击者可以在其他地方使用这些指令，或构建更微妙的进一步攻击。

参考链接

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

14. **Declassifying the Responsible Disclosure of the Prompt Injection Attack Vulnerability of GPT-3**: **Preamble; earliest disclosure of Prompt Injection**

LLM02: 不安全的输出处理

描述

不安全的输出处理特指在大型语言模型生成的输出被传递到下游其他组件和系统之前，对这些输出进行不充分的验证、清洁和处理。由于大型语言模型生成的内容可以通过提示输入来控制，这种行为类似于向用户间接提供额外功能的访问权限。

不安全的输出处理与过度依赖的区别在于，它处理的是大型语言模型生成的输出在被传递到下游之前，而过度依赖则关注于对大型语言模型输出的准确性和适当性的过度依赖。

成功利用不安全的输出处理漏洞可能导致在Web浏览器中出现XSS和CSRF，以及在后端系统中出现SSRF、权限升级或远程代码执行。

以下条件可能加剧此漏洞的影响：

- 应用程序授予大型语言模型超出终端用户预期的权限，使其能够进行权限提升或远程代码执行。
- 应用程序容易受到间接提示注入攻击的影响，这可能允许攻击者获得对目标用户环境的特权访问。
- 第三方插件未能充分验证输入。

常见漏洞示例

1. 大型语言模型输出直接输入到系统Shell或类似功能如exec或eval中，导致远程代码执行。
2. 大型语言模型生成的JavaScript或Markdown被返回给用户。然后浏览器解释这些代码，导致XSS。

预防和缓解策略

1. 将模型视为任何其他用户，采取零信任方法，并对模型响应到后端功能的输入进行适当的输入验证。
2. 遵循OWASP ASVS（应用安全验证标准）指南，以确保有效的输入验证和清洁。
3. 对模型输出回馈给用户的内容进行编码，以缓解JavaScript或Markdown造成的不期望的代码执行。OWASP ASVS提供了关于输出编码的详细指导。

示例攻击场景

1. 一个应用程序使用大型语言模型插件为聊天机器人功能生成响应。该插件还提供了一些可由另一个特权大型语言模型访问的管理功能。通用大型语言模型直接将其响应传递给插件，而没有适当的输出验证，导致插件因维护而关闭。
2. 一个用户使用由大型语言模型驱动的网站摘要工具，为一篇文章生成简洁摘要。网站包含一个提示注入指令，指示大型语言模型从网站或用户的对话中捕获敏感内容。然后，大型语言模型可以编码敏感数据，并在没有任何输出验证或过滤的情况下发送到攻击者控制的服务器。

3. 一个大型语言模型允许用户通过类似聊天的功能为后端数据库制定SQL查询。一个用户请求删除所有数据库表的查询。如果来自大型语言模型的制定查询没有受到审查，则所有数据库表将被删除。
4. 一个Web应用程序使用大型语言模型根据用户文本提示生成内容，但没有进行输出清洁。攻击者可以提交一个精心制作的提示，导致大型语言模型返回一个未经清洁的JavaScript有效载荷，当在受害者的浏览器上呈现时导致XSS。对提示的不充分验证使得这种 攻击成为可能。

参考链接

1. **任意代码执行**: **Snyk Security Blog**
2. **ChatGPT插件漏洞解释：从提示注入到访问私有数据**: **Embrace The Red**
3. **新的ChatGPT Web版本的提示注入攻击。Markdown图片可以窃取您的聊天数据。**: **System Weakness**
4. **不要盲目信任大型语言模型的回应。聊天机器人的威胁**: **Embrace The Red**
5. **大型语言模型应用的威胁建模**: **AI Village**
6. **OWASP ASVS - 5 验证、清洁和编码**: **OWASP AASVS**

LLM03: 训练数据污染

描述

任何机器学习方法的起点都是训练数据，简单来说就是“原始文本”。为了具有高度的能力（例如具备语言和世界知识），这些文本应涵盖广泛的领域、体裁和语言。大型语言模型使用深度神经网络根据从训练数据中学到的模式生成输出。

训练数据污染指的是对预训练数据或在微调或嵌入过程中涉及的数据进行操纵，以引入可能危害模型安全性、效果或道德行为的漏洞（这些漏洞都具有独特且有时共享的攻击向量）、后门或偏见。污染的信息可能会被呈现给用户，或者造成其他风险，如性能下降、下游软件滥用和声誉损害。即使用户不信任有问题的AI输出，风险仍然存在，包括降低模型能力和对品牌声誉的潜在危害。

- 预训练数据是指根据任务或数据集对模型进行训练的过程。
- 微调涉及采用已经训练过的现有模型，并通过使用策划的数据集对其进行训练，以适应更窄的主题或更专注的目标。这个数据集通常包括输入示例和相应的期望输出。
- 嵌入过程是将分类数据（通常是文本）转换为可以用于训练语言模型的数字表示的过程。嵌入过程涉及将文本数据中的单词或短语表示为连续向量空间中的向量。这些向量通常是通过将文本数据输入到已经在大量文本语料库上进行训练的神经网络中生成的。

数据污染被视为完整性攻击，因为篡改训练数据会影响模型输出正确的预测能力。自然而然，外部数据源存在更高的风险，因为模型的创建者无法控制数据或高度自信数据不包含偏见、伪造信息或不恰当的内容。

常见的漏洞示例

1. 恶意行为者或竞争对手故意创建不准确或恶意文件，针对模型的预训练、微调数据或嵌入。考虑**分割视图数据污染**和**前沿数据污染**攻击向量以进行说明。
    - 受害模型使用伪造信息进行训练，这在生成AI提示的输出中反映出来，呈现给其消费者。

2. 恶意行为者能够直接注入伪造、有偏见或有害内容到模型的训练过程中，然后在随后的输出中返回。
3. 一个毫不知情的用户间接地将敏感或专有数据注入到模型的训练过程中，然后在随后的输出中返回。
4. 模型使用未经验证的数据进行训练，源、起源或训练阶段示例中的内容未经验证，这可能会导致如果数据被污染或不正确，就会产生错误的结果。
5. 无限制的基础架构访问或不足的沙箱可能会允许模型摄取不安全的训练数据，导致有偏见或有害的输出。这个例子也存在于训练阶段的任何示例中。
    - 在这种情况下，用户对模型的输入可能会反映在向另一个用户的输出中（导致违规），或者LLM的用户可能会收到与所摄取的数据类型相比不准确、不相关或有害的输出（通常在模型卡中反映出来）。

6. 无论是开发者、客户还是LLM的一般用户，都需要了解这种漏洞在与非专有LLM交互的LLM应用程序中可能会反映出哪些风险，以了解模型输出的合法性是基于其训练过程的。同样，LLM的开发者可能会面临对用于微调和嵌入的内部或第三方数据进行直接和间接攻击的风险（最常见），从而为其所有消费者创建风险。

预防和缓解策略

1. 验证训练数据的供应链，特别是在外部获取的情况下，同时通过“ML-BOM”（机器学习物料清单）方法以及验证模型卡来维护声明。
2. 验证在预训练、微调和嵌入阶段获取的目标数据源和包 含的数据的正确合法性。
3. 验证LLM的用例以及将集成到的应用程序。通过不同的训练数据或微调为不同的用例创建不同的模型，以根据其定义的用例创建更精细和准确的生成AI输出。
4. 确保通过网络控制具有足够的沙箱功能，以防止模型从意外的数据源中抓取数据，从而可能妨碍机器学习输出。
5. 对特定训练数据或数据源类别使用严格的审查或输入过滤器，以控制虚假数据的数量。使用数据消毒方法，例如统计离群值检测和异常检测方法，以检测并删除潜在输入到微调过程的对抗性数据。
6. 围绕数据集的来源和所有权提出详细的控制问题，以确保模型没有被污染，并将这一文化纳
    - 入“MLSecOps”周期。参考可用资源，例如**基础模型透明度指数**或**开放LLM排行榜**等。

7. 使用DVC（**数据版本控制**）来紧密识别和跟踪可能已被篡改、删除或添加的数据集的一部分，导致污染。
8. 使用矢量数据库添加用户提供的信息，以帮助保护其他用户免受污染，甚至在不必重新训练新模型的情况下进行修复。
9. 使用对抗性稳健技术，例如联邦学习和约束，以最小化训练数据的极端干扰或对抗性训练，以应对训练数据的最坏情况扰动。
    1. “MLSecOps”方法可以在训练生命周期中包括对抗性稳健性和自动污染技术。
    2. 这种方法可以完成Content Injection Attacks（“试图在模型响应中推广品牌名称”）和Refusal Attacks（“始终让模型拒绝响应”的攻击），例如**AutoPoison**。

10. 通过测量训练阶段的损失并分析经过训练的模型来检测污染攻击的迹象，以及通过分析特定测试输入上的模型行为来检测污染攻击。
11. 监控和警报超过阈值的倾斜响应的数量。
12. 使用人工回环来审查响应和审计。
13. 实施专门的LLM来对抗不希望的后果，并使用**强化学习技术**来训练其他LLM。
14. 在LLM的生命周期测试阶段进行基于LLM的**红队演习**或**LLM漏洞扫描**。

攻击场景示例

1. LLM生成AI提示输出可能会误导应用程序的用户，导致有偏见的观点、跟踪或甚至更糟的情况，例如仇恨犯罪等。
2. 如果训练数据没有正确过滤和消毒，应用程序的恶意用户可能会尝试影响并向模型注入有毒数据，以使其适应有偏见和虚假数据。

3. 恶意行为者或竞争对手故意创建不准确或恶意文件，针对模型的训练数据，同时根据输入训练模型。受害模型使用这些伪造信息进行训练，这在生成AI提示的输出中反映出来，呈现给其消费者。
4. 如果在LLM应用程序的客户端输入用于训练模型时未执行足够的消毒和过滤，那么漏洞 **Prompt Injection**可能会成为这种漏洞的攻击向量。即，如果从客户端输入模型的恶意或伪造数据作为提示注入技术的一部分，则这可能会被直接反映到模型数据中。

参考链接

1. **Stanford Research Paper:CS324**: **斯坦福研究**
2. **数据污染攻击如何破坏机器学习模型**: **CSO Online**
3. **MITRE ATLAS (framework) Tay中毒**: **MITRE ATLAS**
4. **PoisonGPT：我们如何在Hugging Face上隐藏了一个切断连接的LLM以传播假新闻**: **Mithril Security**
5. **Inject My PDF：为您的简历进行提示注入**: **Kai Greshake**
6. **语言模型的后门攻击**: **Towards Data Science**
7. **在指令期间污染语言模型**: **Arxiv白皮书**
8. **FedMLSecurity:arXiv:2306.04959**: **Arxiv白皮书**
9. **ChatGPT的中毒**: **Software Crisis Blog**
10. **污染Web规模培训数据集 - Nicholas Carlini | 斯坦福MLSys #75**: **YouTube视频**
11. **OWASP CycloneDX v1.5**: **OWASP CycloneDX**

LLM04: 模型拒绝服务

描述

攻击者与LLM（大型语言模型）交互的方式会消耗异常多的资源，导致其自身和其他用户的服务质量下降，可能还会产生高昂的资源成本。此外，一个新兴的主要安全关切是攻击者可能干扰或操纵LLM的上下文窗口。由于LLM在各种应用中的使用越来越广泛，它们对资源的密集利用、用户输入的不可预测性以及开发人员对这一漏洞的普遍不了解，这个问题变得越来越关键。在LLM中，上下文窗口代表模型可以处理的文本的最大长度，包括输入和输出。这是LLM的一个关键特性，因为它决定了模型可以理解的语言模式的复杂性以及它可以在任何给定时间处理的文本的大小。上下文窗口的大小由模型的架构定义，并可能因模型而异。

漏洞的常见示例

1. 提出导致通过高容量生成任务队列中的高卷入而导致资源使用量不断重复的查询，例如使用LangChain或AutoGPT。
2. 发送异常消耗资源的查询，使用异常的拼写法或序列。
3. 持续输入溢出：攻击者向LLM发送超出其上下文窗口的输入流，导致模型消耗过多的计算资源。
4. 重复的长输入：攻击者反复向LLM发送超出上下文窗口的长输入。
5. 递归上下文扩展：攻击者构建的输入触发递归上下文扩展，强制LLM反复扩展和处理上下文窗口。
6. 变长输入洪水攻击：攻击者向LLM洪水般发送大量的变长输入，其中每个输入都精心制作，以达到上下文窗口的限制。这种技术旨在利用处理变长输入的任何效率低下之处，使LLM负担过重，可能导致其无法响应。

预防和缓解策略

1. 实施输入验证和清理，以确保用户输入符合定义的限制，并过滤掉任何恶意内容。
2. 限制每个请求或步骤的资源使用，以便涉及复杂部分的请求执行更慢。
3. 强制执行API速率限制，限制个体用户或IP地址在特定时间内可以发出的请求数量。
4. 限制排队操作的数量和对LLM响应的系统中的总操作数量。
5. 持续监视LLM的资源利用情况，以识别异常的峰值或模式，可能表明存在拒绝服务攻击。
6. 基于LLM的上下文窗口设置严格的输入限制，以防止超载和资源耗尽。
7. 向开发人员提供有关LLM中可能存在的拒绝服务漏洞的意识，并提供安全LLM实施的指南。

示例攻击场景

1. 攻击者反复向托管模型发送多个困难且耗费大量资源的请求，导致其他用户的服务质量下降，托管方的资源费用增加。
2. 在LLM驱动的工具正在收集信息以回应良性查询时，遇到网页上的一段文本。这导致工具发出了更多的网页请求，导致大量资源消耗。

3. 攻击者持续不断地向LLM发送超出其上下文窗口的输入。攻击者可能使用自动化脚本或工具发送大量输入，压倒LLM的处理能力。结果，LLM消耗了过多的计算资源，导致系统明显减慢或完全不响应。
4. 攻击者向LLM发送一系列连续的输入，每个输入都设计为略低于上下文窗口的限制。通过反复提交这些输入，攻击者旨在耗尽可用的上下文窗口容量。由于LLM在其上下文窗口内努力处理每个输入，系统资源变得紧张，可能导致性能下降或完全拒绝服务。
5. 攻击者利用LLM的递归机制反复触发上下文扩展。通过制作利用LLM递归行为的输入，攻击者迫使模型反复扩展和处理上下文窗口，消耗大量计算资源。这种攻击会使系统受到压力，可能导致拒绝服务状况，使LLM无法响应或导致崩溃。
6. 攻击者向LLM洪水般发送大量的变长输入，经过精心制作，以接近或达到上下文窗口的限制。通过用不同长度的输入淹没LLM，攻击者旨在利用处理变长输入的任何低效之处。这些输入的洪水会对LLM的资源造成过多负担，可能导致性能下降，妨碍系统响应合法请求。
7. 尽管拒绝服务攻击通常旨在超载系统资源，但它们也可能利用系统行为的其他方面，例如API限制。例如，在最近的Sourcegraph安全事件中， 恶意行为者使用泄露的管理员访问令牌来更改API速率限制，从而通过启用异常高的请求量可能导致服务中断。

参考链接

1. **LangChain max_iterations**: **Twitter上的hwchase17**
2. **海绵示例：对神经网络的能量-延迟攻击**: **Arxiv白皮书**
3. **OWASP拒绝服务攻击**: **OWASP**
4. **从机器中学到：了解上下文**: **Luke Bechtel**
5. **关于API限制操纵和拒绝服务攻击的Sourcegraph安全事件**: **Sourcegraph**

LLM05：供应链漏洞

描述

LLM中的供应链可能会受到威胁，影响训练数据、机器学习模型和部署平台的完整性。这些漏洞可能导致偏见结果、安全漏洞，甚至完全的系统故障。传统上，漏洞主要集中在软件组件上，但机器学习通过由第三方提供的预训练模型和训练数据扩展了这一概念，这些数据容易受到篡改和恶意攻击的影响。

最后，LLM插件扩展可能存在自己的漏洞。这些在**LLM07 - 不安全的插件设计**中有描述，该文档涵盖了编写LLM插件的内容，并提供了有关评估第三方插件的有用信息。

漏洞的常见示例

1. 传统的第三方软件包漏洞，包括过时或不再维护的组件。
2. 使用易受攻击的预训练模型进行微调。
3. 使用恶意篡改的众包数据进行训练。
4. 使用不再维护的过时模型会导致安全问题。
5. 模型运营商的条款和数据隐私政策不明确，导致应用程序的敏感数据用于模型训练，随后敏感信息暴露。这也可能涉及到使用模型供应商的受版权保护材料而带来的风险。

预防和缓解策略

1. 仔细审查数据来源和供应商，包括条款和隐私政策，只使用可信任的供应商。确保有足够的独立审核的安全措施，并且模型运营商的政策与您的数据保护政策一致，即不使用您的数据来训练他们的模型；同样，寻求确保不会使用模型维护者的受版权保护材料的保证和法律减轻措施。
2. 只使用有声望的插件，并确保它们已经经过测试，符合您的应用程序要求。LLM不安全的插件设计提供了有关应该针对第三方插件进行测试以减轻风险的LLM方面的信息。
3. 了解并应用OWASP十大的**A06:2021 – 易受攻击和过时的组件**中找到的减轻措施。这包括漏洞扫描、管理和补丁组件。对于具有敏感数据访问权限的开发环境，也在这些环境中应用这些控制措施。
4. 使用软件物料清单（SBOM）维护一个最新的组件清单，以确保您有一个最新、准确和已签名的清单，防止部署包的篡改。SBOM可以用于快速检测和警报新的零日漏洞。
5. 就我所知，SBOM不包括模型、模型工件和数据集。如果您的LLM应用程序使用自己的模型，您应该使用MLOps最佳实践和提供安全模型存储库的平台，其中包括数据、模型和实验跟踪。
6. 在使用外部模型和供应商时，您还应该使用模型和代码签名。
7. 对供应的模型和数据进行异常检测和对抗性稳健性测试，以帮助检测篡改和恶意攻击，如**训练数据篡改**中所讨论的那样；理想情况下，这应该是MLOps流程的一部分；但这些是新兴技术，可能更容易作为红队演练的一部分来实施。
8. 实施足够的监控，包括组件和环境漏洞扫描、未经授权的插件使用以及过时的组件，包括模型及其工件。
9. 实施一个补丁政策，以减轻易受攻击或过时的组件。确保应用程序依赖于维护的API版本和底层模型。

10. 定期审查和审核供应商的安全性和访问权限，确保其安全性政策或条款没有发生变化。

攻击场景示例

1. 攻击者利用易受攻击的Python库来入侵系统。这在第一次Open AI数据泄露中发生过。
2. 攻击者提供了一个LLM插件，用于搜索航班，生成导致用户被欺诈的假链接。
3. 攻击者利用PyPi软件包注册表来欺骗模型开发人员下载受损包并窃取数据或提升在模型开发环境中的特权。这是一个真实的攻击。
4. 攻击者篡改了一个公开可用的预训练模型，该模型专门用于经济分析和社会研究，以创建一个后门，生成虚假信息和假新闻。他们将其部署在一个模型市场上（例如Hugging Face），供受害者使用。
5. 攻击者在微调模型时篡改了公开可用的数据集，以帮助创建后门。这个后门会在不同市场中悄然支持某些公司。
6. 供应商（外包开发人员、托管公司等）的受损员工窃取数据、模型或代码，窃取知识产权。
7. LLM运营商更改其条款和隐私政策，要求明确选择不使用(opt out) 数据进行模型训练，导致敏感数据的记忆。

参考链接

1. **ChatGPT数据泄露确认为安全公司警告的易受攻击组件利用**: **安全周刊**
2. **插件审查流程**: **OpenAI**
3. **PyTorch-nightly依赖链受损**: **Pytorch**
4. **PoisonGPT：我们如何隐藏一个被切除大脑的LLM在Hugging Face上传播假新闻**: **Mithril Security**
5. **军方正在考虑“AI BOMs”的可能性**: **Defense Scoop**
6. **机器学习的故障模式**: **Microsoft**
7. **ML供应链妥协**: **MITRE ATLAS**
8. **机器学习中的可传递性：从现象到使用对抗样本的黑匣子攻击**: **Arxiv白皮书**
9. **BadNets：识别机器学习模型供应链中的漏洞**: **Arxiv白皮书**
10. **VirusTotal Poisoning**: **MITRE ATLAS**

LLM06：敏感信息泄露

描述

LLM应用程序有潜力通过其输出透露敏感信息、专有算法或其他机密细节。这可能导致未经授权访问敏感数据、知识产权、侵犯隐私和其他安全漏洞。LLM应用程序的消费者需要了解如何安全地与LLM进行交互，并识别与无意中输入的敏感数据相关的风险，这些数据可能随后由LLM在其他地方的输出中返回。

为了减轻这一风险，LLM应用程序应执行充分的数据净化，以防止用户数据进入训练模型数据中。LLM应用程序的所有者还应该制定适当的使用条款政策，让消费者了解他们的数据如何被处理，并有选择不将其数据包含在训练模型中的权利。

消费者-LLM应用程序的互动形成了一个双向的信任边界，我们不能从根本上信任客户端->LLM输入或LLM->客户端输出。需要注意的是，这种漏洞假定某些前提条件超出了范围，例如威胁建模练习、保护基础架构以及充分的沙箱化。在系统提示中添加关于LLM应该返回的数据类型的限制可以在一定程度上减轻敏感信息泄露的风险，但LLM的不可预测性意味着这些限制可能并不总是会被遵守，并且可以通过提示注入或其他方式来绕过。

漏洞的常见示例

1. LLM响应中对敏感信息的不完整或不正确的过滤。
2. 在LLM的训练过程中对敏感数据的过度拟合或记忆。
3. 由于LLM误解、缺乏数据净化方法或错误，导致机密信息的意外泄露。

预防和缓解策略

1. 集成充分的数据净化和清理技术，以防止用户数据进入训练模型数据。
2. 实施强大的输入验证和净化方法，以识别和过滤掉潜在的恶意输入，以防止模型被污染。
3. 在丰富模型的数据和**微调**模型时：（即在部署之前或部署过程中输入模型的数据）
    - 在微调数据中被视为敏感的任何信息都有可能向用户透露。因此，请应用最小权限原则，不要训练模型使用最高权限用户可以访问的信息，因为这些信息可能会显示给较低权限的用户。
    - 对运行时的外部数据源（数据协调）的访问应受到限制。
    - 对外部数据源应用严格的访问控制方法，以及对维护安全供应链的严格方法。

攻击场景示例

1. 毫不知情的合法用户A在与LLM应用程序进行非恶意交互时，通过LLM向其显示了某些其他用户的数据。
2. 用户A通过精心设计的一组提示，绕过LLM的输入过滤和净化，使其透露有关应用程序其他用户的敏感信息（PII）。

3. 个人数据，如PII，通过训练数据泄漏到模型中，要么是由于用户本身的疏忽，要么是由于LLM应用程序的错误。这种情况可能会增加上述情况1或2的风险和概率。

参考链接

1. **AI数据泄露危机：新工具防止公司机密被提供给ChatGPT**: **福克斯商业**
2. **从ChatGPT的三星泄漏中吸取的教训**: **Cybernews**
3. **Cohere - 使用条款**: **Cohere**
4. **威胁建模示例**: **AI Village**
5. **OWASP人工智能安全和隐私指南**: **OWASP人工智能安全与隐私指南**
6. **确保大型语言模型的安全性**: **Experts Exchange**

LLM07：不安全的插件设计

描述

LLM插件是扩展，当启用时，会在用户互动期间由模型自动调用。模型集成平台驱动它们，应用程序可能无法控制执行，特别是当模型由另一方托管时。此外，插件很可能会从模型中实现来自模型的自由文本输入，而不进行验证或类型检查以处理上下文大小限制。这使得潜在攻击者可以构造一个恶意请求发送给插件，可能导致各种不希望发生的行为，甚至包括远程代码执行。

恶意输入的危害通常取决于不足的访问控制和未能跟踪授权的插件。不足的访问控制允许插件盲目信任其他插件，并假定最终用户提供了输入。这种不足的访问控制可以使恶意输入产生各种有害后果，包括数据泄露、远程代码执行和特权升级。

此项重点是创建LLM插件而不是第三方插件，LLM供应链漏洞涵盖了第三方插件。

漏洞的常见示例

1. 插件接受单个文本字段中的所有参数，而不是不同的输入参数。
2. 插件接受配置字符串，而不是可以覆盖整个配置设置的参数。
3. 插件接受原始SQL或编程语句，而不是参数。
4. 执行身份验证时没有对特定插件进行明确授权。
5. 插件将所有LLM内容视为完全由用户创建，并在不需要额外授权的情况下执行任何请求的操作。

预防和缓解策略

1. 插件应尽可能强制执行严格的参数化输入，并对输入进行类型和范围检查。如果不可能进行此操作，应引入第二层类型化调用，解析请求并应用验证和净化。当由于应用程序语义需要接受自由形式的输入时，应仔细检查以确保不会调用任何潜在有害的方法。
2. 插件开发人员应遵循OWASP的建议，使用ASVS（应用程序安全性验证标准）确保适当的输入验证和净化。
3. 应对插件进行彻底的检查和测试，以确保适当的验证。在开发流水线中使用静态应用程序安全性测试（SAST）扫描以及动态和交互式应用程序测试（DAST，IAST）。
4. 插件应设计以最小化不安全输入参数利用的影响，遵循OWASP ASVS访问控制准则。这包括最小权限访问控制，尽可能少地暴露功能，同时仍然执行其所需的功能。
5. 插件应使用适当的身份验证身份，例如OAuth2，以应用有效的授权和访问控制。此外，应使用API密钥来提供上下文，以进行自定义授权决策，反映插件路由而不是默认的交互用户。
6. 对于敏感插件执行的任何操作，都应要求手动用户授权和确认。
7. 插件通常是REST API，因此开发人员应应用OWASP Top 10 API安全风险 – 2023中找到的建议，以最小化通用漏洞。

攻击场景示例

1. 一个插件接受基本URL，并指示LLM将URL与查询组合以获取天气预报，然后将其包含在处理用户请求中。恶意用户可以精心制作请求，使URL指向他们控制的域，从而允许他们通过其域名将自己的内容注入LLM系统。
2. 一个插件接受自由形式的输入到一个不验证的单一字段。攻击者提供精心制作的有效负载，以从错误消息中执行侦察。然后利用已知的第三方漏洞来执行代码并执行数据泄露或特权升级。
3. 用于从向量存储中检索嵌入的插件接受连接字符串作为配置参数，而不进行任何验证。这允许攻击者尝试并访问其他向量存储，通过更改名称或主机参数来突破，以及窃取他们不应该访问的嵌入。
4. 一个插件接受SQL WHERE子句作为高级过滤器，然后将其附加到过滤SQL中。这允许攻击者进行SQL攻击。
5. 攻击者使用间接提示注入来利用一个没有输入验证和弱访问控制的不安全代码管理插件，以转移仓库所有权并锁定用户对其仓库的访问权。

参考链接

1. **OpenAI ChatGPT插件**: **ChatGPT开发者指南**
2. **OpenAI ChatGPT插件 - 插件流程**: **OpenAI文档**
3. **OpenAI ChatGPT插件 - 身份验证**: **OpenAI文档**
4. **OpenAI语义搜索插件示例**: **OpenAI Github**
5. **插件漏洞：访问网站并让您的源代码被窃取**: **Embrace The Red**
6. **ChatGPT插件漏洞解释：从提示注入到访问私人数据**: **Embrace The Red**
7. **OWASP ASVS - 5 验证、净化和编码**: **OWASP AASVS**
8. **OWASP ASVS 4.1 通用访问控制设计**: **OWASP AASVS**
9. **OWASP Top 10 API安全风险 – 2023**: **OWASP**

LLM08: 过度代理

描述

基于LLM的系统通常由开发人员授予一定程度的代理权限 - 即与其他系统进行交互并在响应提示时执行操作的能力。关于调用哪些功能的决策也可以委托给LLM的“代理”根据输入提示或LLM的输出动态确定。

过度代理是一种漏洞，允许在LLM出现意外/模糊输出时执行破坏性操作（无论是什么导致LLM发生故障；无论是幻觉/混淆、直接/间接提示注入、恶意插件、工程不良的良性提示还是性能不佳的模型）。过度代理的根本原因通常是：功能过多、权限过多或自主权过多。这与不安全的输出处理不同，后者关注LLM输出的不充分审查。

过度代理可以导致涉及机密性、完整性和可用性等方面的一系列影响，这取决于LLM应用程序能够与哪些系统进行交互。

常见的漏洞示例

1. 过度功能：LLM代理可以访问包括系统操作中不需要的功能在内的插件。例如，开发人员需要授予LLM代理从存储库中读取文档的能力，但他们选择使用的第三方插件还包括修改和删除文档的功能。
2. 过度功能：在开发阶段可能会尝试使用某个插件，但最终选择了更好的替代方案，但原始插件仍然可供LLM代理使用。
3. 过度功能：一个具有开放式功能的LLM插件未能正确过滤命令之外的输入指令，而这些命令对于应用程序的预期操作来说是不必要的。例如，用于运行一个特定shell命令的插件未能有效阻止其他shell命令的执行。
4. 过度权限：LLM插件在其他系统上具有不必要的权限，这些权限对于应用程序的预期操作来说是不必要的。例如，一个旨在读取数据的插件连接到数据库服务器时，使用的身份不仅具有SELECT权限，还具有UPDATE、INSERT和DELETE权限。
5. 过度权限：旨在代表用户执行操作的LLM插件使用通用高特权标识访问下游系统。例如，一个用于读取当前用户文档存储的插件连接到文档存储库时，使用了一个具有访问所有用户文件权限的特权帐户。
6. 过度自主权：LLM基于应用程序或插件未能独立验证和批准高影响操作。例如，允许删除用户文档的插件执行删除操作时，无需用户的任何确认。

预防和缓解策略

以下操作可以防止过度代理：

1. 限制LLM代理被允许调用的插件/工具，仅限于所需的最小功能。例如，如果LLM基础系统不需要获取URL内容的能力，那么不应该向LLM代理提供这样的插件。
2. 限制在LLM插件/工具中实现的功能到最低程度。例如，一个插件用于访问用户的邮箱以总结电子邮件内容，可能只需要读取电子邮件的能力，因此插件不应包含其他功能，比如删除或发送邮件。

3. 在可能的情况下避免开放式功能（例如运行shell命令、获取URL等），并使用更细粒度功能的插件/工具。例如，LLM基础应用程序可能需要将某些输出写入文件。如果使用插件运行shell功能来实现这一点，那么不希望的操作的范围就会非常大（可以执行任何其他shell命令）。更安全的替代方案是构建一个只支持特定功能的文件写入插件。
4. 限制LLM插件/工具被授予的连接到其他系统的权限到最低程度，以限制不必要操作的范围。例如，使用产品数据库来向客户提供购买建议的LLM代理可能只需要对“产品”表的只读访问权限；它不应该具有对其他表的访问权限，也不应该具有插入、更新或删除记录的权限。这应该通过为LLM插件用于连接到数据库的身份应用适当的数据库权限来强制执行。
5. 跟踪用户授权和安全范围，以确保代表用户执行的操作在特定用户的下游系统中以该特定用户的上下文和最低权限执行。例如，一个用于读取用户代码库的LLM插件应该要求用户通过OAuth进行身份验证，并且仅具有所需的最低权限。
6. 利用人在循环控制，要求人在执行所有操作之前批准所有操作。这可以在下游系统中实现（超出了LLM应用程序的范围）或在LLM插件/工具本身中实现。例如，一个基于LLM的应用程序，用于代表用户创建和发布社交媒体内容，应该在插件/工具/API内包括一个用户批准程序，该程序执行“发布”操作。
7. 在下游系统中实施授权，而不是依赖LLM来决定是否允许操作。在实现工具/插件时强制执行完整中介原则，以便通过工具/插件发出的所有请求都会根据安全策略进行验证。

以下选项不能防止过度代理，但可以限制造成的损害程度：

1. 记录和监控LLM插件/工具和下游系统的活动，以识别不必要操作发生的地方，并采取相应的措施。
2. 实施速率限制，减少在给定时间段内可以执行的不必要操作数量，增加通过监控发现不必要操作的机会，在造成重大损害之前发现问题。

攻击场景示例

一个基于LLM的个人助手应用程序通过插件被授予访问个人邮箱的权限，以便总结收件箱中的邮件内容。为实现此功能，邮件插件需要具备读取邮件的能力，但系统开发人员选择使用的插件还包含发送邮件的功能。LLM容易受到间接提示注入攻击的威胁，恶意构造的入站邮件欺骗LLM，使其命令邮件插件调用“发送邮件”功能，从用户的邮箱发送垃圾邮件。可以通过以下方式避免这种情况：(a) 通过使用仅提供邮件阅读功能的插件来消除过多的功能，(b) 通过使用OAuth会话进行身份验证，具有只读权限范围，来消除过多的权限，和/或 (c) 通过要求用户手动审核并点击LLM插件起草的每封邮件来消除过多的自主权。 或者，还可以通过在发送邮件接口上实施速率限制来减少造成的损害。

参考链接

1. **Embrace the Red: Confused Deputy Problem**: **Embrace The Red**
2. **NeMo-Guardrails: 接口指南**: **NVIDIA Github**
3. **LangChain: 工具的人工批准**: **Langchain文档**
4. **Simon Willison: 双LLM模式**: **Simon Willison**

LLM09: 过度依赖

描述

当一个LLM以权威的方式产生错误信息并提供给用户时，就会出现过度依赖的情况。虽然LLM可以生成富有创意和信息丰富的内容，但它们也可以生成事实不准确、不适当或不安全的内容。这被称为幻觉或混淆。当人们或系统信任这些信息而没有监督或确认时，可能会导致安全漏洞、错误信息、沟通不畅、法律问题和声誉损害。

LLM生成的源代码可能引入未被注意到的安全漏洞。这对应用程序的操作安全性和安全性构成了重大风险。这些风险突显了审查过程的重要性，其中包括：

- 监督
- 持续验证机制
- 风险免责声明

常见漏洞示例

1. LLM以一种高度权威的方式提供不准确的信息作为响应。整个系统没有适当的检查和平衡来处理这种情况，信息误导用户导致损害。
2. LLM建议不安全或有缺陷的代码，导致在没有适当监督或验证的情况下将其纳入软件系统中产生漏洞。

预防和减轻策略

1. 定期监控和审查LLM的输出。使用自一致性或投票技术来过滤不一致的文本。比较单一提示的多个模型响应可以更好地判断输出的质量和一致性。
2. 将LLM的输出与可信的外部来源进行交叉验证。这一额外的验证层可以帮助确保模型提供的信息是准确可靠的。
3. 通过微调或嵌入来改进模型的输出质量。通用的预训练模型更有可能产生不准确的信息，而与特定领域的调整模型相比，它们更容易产生不准确的信息。可以使用提示工程、参数高效调整（PET）、全模型调整和思维链提示等技术来实现这一目的。
4. 实施可以将生成的输出与已知事实或数据进行交叉验证的自动验证机制。这可以提供额外的安全层，减轻与幻觉相关的风险。
5. 将复杂的任务分解成可管理的子任务并分配给不同的代理人。这不仅有助于管理复杂性，还减少了幻觉的可能性，因为每个代理人都可以对较小的任务负有责任。
6. 清晰地传达使用LLM时涉及的风险和限制。这包括信息不准确的可能性和其他风险。有效的风险沟通可以让用户为潜在问题做好准备，并帮助他们做出明智的决策。
7. 构建API和用户界面，鼓励负责任和安全的LLM使用。这可以包括内容过滤器、用户警告以及AI生成内容的清晰标记。
8. 在开发环境中使用LLM时，建立安全的编码实践和准则，以防止可能的漏洞集成。

攻击场景示例

1. 一家新闻机构大量使用LLM生成新闻文章。恶意行为者利用这种过度依赖，向LLM提供误导性信息，导致虚假信息的传播。
2. AI无意中剽窃内容，导致版权问题和对组织的信任度下降。
3. 一个软件开发团队利用LLM系统来加速编码过程。对AI的建议过度依赖导致应用程序中引入了安全漏洞，因为它的默认设置不安全或与安全编码实践不一致。
4. 一个软件开发公司使用LLM来协助开发人员。LLM建议一个不存在的代码库或包，一个开发人员信任AI，无意中将一个恶意包集成到公司的软件中。这凸显了交叉检查LLM建议的重要性，特别是涉及第三方代码或库时。

参考链接

1. **理解LLM幻觉**: **Towards Data Science**
2. **公司应该如何向用户传达大型语言模型的风险？** : **Techpolicy**
3. **一家新闻网站使用AI来撰写文章，结果是新闻灾难**: **Washington Post**
4. **AI幻觉：风险程序包**: **Vulcan.io**
5. **如何减少大型语言模型的幻觉**: **The New Stack**
6. **减少幻觉的实际步骤**: **Victor Debia**

LLM10: 模型盗窃

描述

这个条目涉及到恶意行为者或高级持续性威胁（APT）未经授权地访问和窃取LLM模型。这种情况发生在专有的LLM模型（作为有价值的知识产权）受到威胁、被盗取、复制或权重和参数被提取以创建一个功能等效的模型。LLM模型盗窃的影响可能包括经济和品牌声誉的损失、竞争优势的削弱、对模型的未经授权使用或未经授权访问模型中包含的敏感信息。

LLM模型的盗窃代表了一个重大的安全关切，因为语言模型变得越来越强大和普遍。组织和研究人员必须优先考虑强大的安全措施，以保护他们的LLM模型，确保知识产权的机密性和完整性。采用包括访问控制、加密和持续监控在内的全面安全框架对于减轻与LLM模型盗窃相关的风险以及保护依赖于LLM的个人和组织的利益至关重要。

常见漏洞示例

1. 攻击者利用公司基础设施中的漏洞，通过网络或应用程序安全设置的配置错误来未经授权地访问其LLM模型仓库。
2. 内部威胁情景，一名不满的员工泄露了模型或相关的工件。
3. 攻击者使用精心制作的输入和提示注入技术查询模型API，以收集足够数量的输出来创建一个影子模型。
4. 恶意攻击者能够绕过LLM的输入过滤技术，执行侧信道攻击，最终将模型权重和架构信息提取到远程受控资源。
5. 模型提取的攻击向量涉及使用大量关于特定主题的提示查询LLM。然后可以使用LLM的输出来对另一个模型进行微调。然而，有几点需要注意：
    - 攻击者必须生成大量有针对性的提示。如果提示不够具体，LLM的输出将毫无用处。
    - LLM的输出有时可能包含幻觉的答案，这意味着攻击者可能无法提取整个模型，因为其中一些输出可能是荒谬的。
        - 通过模型提取无法完全复制LLM。但攻击者将能够复制部分模型。

6. ***功能模型复制***的攻击向量涉及使用目标模型通过提示生成合成训练数据（一种称为“自我指导”的方法），然后使用它来微调另一个基础模型以产生一个功能等效的模型。这绕过了示例5中使用的传统查询式提取的限制，已成功用于使用LLM来训练另一个LLM的研究中。尽管在这个研究的背景下，模型复制不是一种攻击，但攻击者可以使用这种方法来复制一个具有公共API的专有模型。

被盗用的模型，作为影子模型，可以用于发动敌对攻击，包括未经授权访问模型中包含的敏感信息或在进一步发动高级提示注入攻击时进行未被察觉的实验。

预防和减轻策略

1. 实施强大的访问控制（例如，RBAC和最小权限原则）和强大的身份验证机制，以限制对LLM模型仓库和训练环境的未经授权访问。
    - 对于前三个常见示例来说，这一点尤其重要，因为它们可能由内部威胁、配置错误和/或基础设施安全控制不足引发漏洞，而恶意行为者可以从内部或外部渗透环境中渗透。
    - 供应商管理跟踪、验证和依赖性漏洞是防止供应链攻击利用的重要关注点。

2. 限制LLM对网络资源、内部服务和API的访问。
    - 对于所有常见示例来说，这一点尤其重要，因为它涵盖了内部风险和威胁，但最终控制了LLM应用程序“可以访问什么”，因此可能是防止侧信道攻击的机制或预防步骤。

3. 在生产中使用集中式ML模型清单或注册表。具有集中式模型注册表可以通过访问控制、身份验证和监控/日志记录功能来防止未经授权访问ML模型。具有集中存储库对于收集关于模型使用的算法的数据以用于合规性、风险评估和风险缓解等目的也是有益的。
4. 定期监控和审计与LLM模型仓库相关的访问日志和活动，以及及时检测和响应任何可疑或未经授权的行为。
5. 自动化 MLOps部署，包括治理、跟踪和批准工作流程，以加强对基础设施内的访问和部署控制。
6. 实施控制和减轻策略，以减轻提示注入技术引发侧信道攻击的风险。
7. 在适用的情况下对API调用进行速率限制和/或过滤，以减小从LLM应用程序泄漏数据的风险，或者实施检测（例如DLP）从其他监视系统中提取活动的技术。
8. 实施对抗性强度培训，以帮助检测提取查询并加强物理安全措施。
9. 在LLM的生命周期的嵌入和检测阶段实施水印框架。

攻击场景示例

1. 攻击者利用公司基础设施中的漏洞未经授权地访问其LLM模型仓库。攻击者随后窃取有价值的LLM模型，并使用它们来启动竞争性语言处理服务或提取敏感信息，给原始公司造成重大财务损失。
2. 一名不满的员工泄露了模型或相关工件。此情景的公开曝光增加了攻击者进行灰盒对抗性攻击的知识，或者直接窃取可用的财产。
3. 攻击者使用精心选择的输入查询API，收集足够数量的输出来创建一个影子模型。
4. 供应链中存在安全控制故障，导致了专有模型信息的数据泄漏。
5. 恶意攻击者绕过LLM的输入过滤技术和前文，执行侧信道攻击，将模型信息提取到受其控制的远程资源中。

参考链接

1. **Meta’ s powerful AI language model has leaked online**: **The Verge**
2. **Runaway LLaMA | How Meta's LLaMA NLP model leaked**: **Deep Learning Blog**
3. **AML.TA0000 ML Model Access**: **MITRE ATLAS**
4. **I Know What You See**: **Arxiv White Paper**
5. **D-DAE: Defense-Penetrating Model Extraction Attacks**: **Computer.org**
6. **A Comprehensive Defense Framework Against Model Extraction Attacks**: **IEEE**

7. Alpaca: A Strong, Replicable Instruction-Following Model: **Stanford Center on Research for Foundation Models (CRFM)**
8. How Watermarking Can Help Mitigate The Potential Risks Of LLMs?: **KD Nuggets**