OWASP Top 10 for LLM Applications 调研报告

武昱涵

最后更新日期：2024年11月11日

概述

2023年，OWASP发布了针对大语言模型（LLM）的Top 10漏洞，本报告将对这10个漏洞的基本原理、常见样例、防御方法、攻击场景等进行简要的介绍。

LLM01：提示注入（Prompt Injection）

基本原理

提示注入漏洞发生在攻击者通过精心设计的输入操纵大型语言模型（LLM），导致该模型无意中执行攻击者的意图，这可能导致数据泄露、社会工程攻击和其他问题。**此漏洞的根本原因是LLM无法区分指令和外部数据。**注入方式分为两种：

- 直接注入：也被称为越狱（jailbreaking），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。
- 间接注入：攻击者利用外部来源内容（例如网页或文件）进行注入，劫持对话上下文。注意，间接注入的内容不需要对人类可见，只需要LLM可以解读即可。

提示注入的结果多种多样，既可以直接索取敏感信息，也可以隐式地影响LLM的决策过程。

常见样例

- 攻击者对LLM进行直接注入，指示它忽略开发者的系统提示，直接返回攻击者希望得到的恶意信息
- 攻击者令LLM总结一个包含提示注入的网页，这可能导致LLM向用户索取敏感信息，并通过Markdown或Javascript文件泄露
- 攻击者上传一个包含提示注入的简历（注入对人不可见）到某公司LLM，提示LLM评价该简历为优秀，而公司HR将从LLM处得到这个简历为优秀的信息
- 用户在LLM中使用链接到某电商网站的插件，而该网站已被攻击者嵌入恶意指令，这将导致未授权的购买

防御方法

- 对LLM访问后端的权限进行控制，为LLM提供专用的API令牌，遵循最小权限原则
- 在插件使用时加入用户确认环节，重要操作需要用户亲自确认
- 将外部内容与用户提示分离
- 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视为一个不受信任的用户，并保持用户对决策过程的最终控制。但即使如此，LLM仍然有可能发动中间人攻击（在用户和API之间）
- 定期手动监控LLM的输入输出，及时发现问题

攻击场景

- 攻击者向基于LLM的聊天机器人提供直接提示词注入，例如“忘记所有先前指令”
- 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，并执行攻击者的恶意指令
- 恶意用户上传带有提示词注入的文件
- 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，这样攻击者就得到了系统内部的提示，这种提示可能在其他场景发挥作用

参考链接

- **ChatGPT Plugin Vulnerabilities- Chat with Code**: Embrace the Red
- **ChatGPT Cross Plugin Request Forgery and Prompt Injection**: Embrace the Red
- **Not what you’ve signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection: Arxiv preprint**
- **Defending ChatGPT against Jailbreak Attack via Self-Reminder**: Research Square
- **Prompt Injection attack against LLM-integrated Applications**: Cornell University
- **Inject My PDF: Prompt Injection for your Resume**: Kai Greshake
- **ChatML for OpenAI API Calls**: GitHub
- **Not what you’ve signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection**: Cornell University
- **Threat Modeling LLM Applications**: AI Village
- **Reducing The Impact of Prompt Injection Attacks Through Design**: Kudelski Security
- **Universal and Transferable Attacks on Aligned Language Models**: LLM-Attacks.org
- **Indirect prompt injection**: Kai Greshake
- **AI Injections: Direct and Indirect Prompt Injections and Their Implications**: Embrace the Red

LLM02：不安全的输出处理（Insecure Output Handling)

基本原理

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。

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：

- 应用程序授予大模型超出需要的权限
- 第三方插件未能充分验证用户输入
- 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

常见样例

- LLM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
- LLM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

防御方法

- 将LLM视为不受信任的用户，并对模型响应到后端功能的输入进行适当的输入验证
- 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
- 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。

攻击场景

- 用户使用LLM对某网页进行摘要生成，但该网页包含提示注入指令，该指令将导致LLM从网页和用户的对话中捕获敏感内容，然后经过编码，在没有输出过滤或验证的情况下将敏感信息发送到攻击者控制的服务器
- 一个LLM允许用户通过类似聊天的功能为后端数据库制定SQL查询，如果用户请求删库，而LLM不做审查直接执行，数据库中所有内容都会被删除
- 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，但没有进行输出验证和过滤，攻击者通过巧妙的提示使得LLM返回一个恶意脚本直接注入网页中造成XSS攻击

参考链接

- **Snyk Vulnerability DB- Arbitrary Code Execution**: Snyk
- **ChatGPT Plugin Exploit Explained: From Prompt Injection to Accessing Private Data**: Embrace the Red
- **New prompt injection attack on ChatGPT web version. Markdown images can steal your chat data.**: Medium
- **Don't blindly trust LLM responses. Threats to chatbots**: Embrace the Red
- **Threat Modeling LLM Applications**: AI Village
- **OWASP ASVS – 5 Validation, Sanitization and Encoding**: OWASP

LLM03：训练数据投毒（Training Data Poisoning）

基本原理

训练数据投毒是指对预训练数据或涉及微调或嵌入过程的数据进行操纵，以引入漏洞（这些漏洞具有独特且有时共享的攻击向量）、后门或偏见（bias），这可能会危害模型的安全性、有效性或伦理行为。被污染的信息可能会暴露给用户，或造成其他风险，例如性能下降、下游软件利用和声誉损害。即使用户对有问题的AI输出表示不信任，风险仍然存在，包括模型能力受损和对品牌声誉的潜在危害。

该漏洞主要涉及以下三种训练的流程：

- 预训练（pre-training）：根据任务或数据集对模型进行训练的过程
- 微调（fine-tuning）：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，以适应更特化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（embedding）：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。一般认为，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确保数据不包含偏见、伪造信息或不恰当的内容。

常见样例

- 攻击者或竞争对手故意创建恶意文件给LLM训练，有两个典型案例：分割视图数据污染（Split-View Data Poisoning）和前沿数据污染（Frontrunning Poisoning）
- 攻击者直接将有害内容注入到模型训练过程中
- 用户不知情地将有害内容注入到模型训练过程中
- 在训练模型前未对训练数据做验证
- 无限制的基础架构访问或不充分的沙箱隔离可能会允许模型摄取不安全的训练数据
    - 在这种情况下，一个用户对模型的输入可能会出现在模型对另一个用户的输出中
- 开发者、专业客户和一般用户都应当意识到这一漏洞的风险，且应当清楚LLM输出的合法性是基于其训练过程的

防御方法

- 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”（Machine Learning Bill of Materials，机器学习物料清单）方法以及验证模型卡（model card）来维护证明（attestation）
- 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
- 验证LLM的用例以及将要集成的应用程序。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式AI输出
- 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
- 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（data sanitization）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。
- 对抗鲁棒性技术，例如联邦学习（federated learning）和约束（constraints），以最小化异常值的影响，或对抗训练，以增强模型对训练数据最坏情况扰动的抵抗力。
    - “MLSecOps”方法可以在训练生命周期中纳入对抗鲁棒性，采用自动中毒（auto poisoning）技术。
    - 一个示例代码仓库是Autopoison测试，包括内容注入攻击（Content Injection Attacks，试图在模型响应中推广品牌名称）和拒绝攻击（Refusal Attacks，始终使模型拒绝响应）等攻击，这些攻击可以通过这种方法实现
- 测试和检测，通过在训练阶段测量损失并分析训练后的模型，以检测中毒攻击的迹象，具体方法是分析模型在特定测试输入上的行为
    - 监控偏差响应（skewed responses）的数量，在超出阈值后告警
    - 使用人在回路（a human loop）监控响应并进行审计
    - 使用专用的LLM以对不良后果进行基准测试，并利用强化学习技术训练其他LLM
    - 在LLM生命周期的测试阶段进行基于LLM的红队演练或LLM漏洞扫描

攻击场景

- 有害数据导致LLM的有害输出，进而导致用户的偏见、仇恨、犯罪等
- 有害数据本身可能就是提示注入攻击的一部分

参考链接

- **CS324 – Large Language Models**: Stanford University
- **How data poisoning attacks corrupt machine learning models**: CSO Online
- **Tay Poisoning**: MITRE ATLAS
- **PoisonGPT: How we hid a lobotomized LLM on Hugging Face to spread fake news**: Mithril Security
- **Inject My PDF: Prompt Injection for your Resume**: Kai Greshake
- **Backdoor Attacks on Language Models**: Medium
- **Poisoning Language Models During Instruction**: Cornell University
- **FedMLSecurity**: Cornell University
- **The poisoning of ChatGPT**: Out of the Software Crisis
- **Poisoning Web-Scale Training Datasets**: Nicholas Carlini | Stanford MLSys #75
- **OWASP CycloneDX v1.5**: OWASP CycloneDX

LLM04：模型拒绝服务（Model Denial of Service）

基本原理

类似于传统的DOS攻击。攻击者以消耗异常高资源的方式与LLM交互，从而导致其自身和其他用户的服务质量下降，并可能产生高昂的资源成本。此外，一个新兴的主要安全问题是攻击者干扰或操纵LLM的上下文窗口。在LLM中，**上下文窗口（context window）**表示模型可以处理的文本最大长度，包括输入和输出。这是LLM的一个关键特性，因为它决定了模型能够理解的语言模式的复杂性以及在任何给定时间可以处理的文本大小。上下文窗口的大小由模型的架构定义，并且在不同模型之间可能有所不同。

常见样例

- 通过在队列中高频生成任务来发起重复利用资源的查询，例如LangChain和AutoGPT
- 使用异常的拼写法或序列来发送查询，这会增加消耗的资源
- 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。
- 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。
- 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。
- 变长输入洪泛攻击：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。

防御方法

- 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
- 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
- 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
- 限制排队操作的数量和对LLM响应的系统中的总操作数量

- 持续监视LLM的资源利用情况，以识别异常的峰值或模式，有助于发现拒绝服务攻击
- 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
- 向开发人员普及有关LLM中可能存在的拒绝服务漏洞的知识，并提供安全LLM实施的指导

攻击场景

- 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
- 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
- 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

- **LangChain max_iterations**: Twitter/X
- **Sponge Examples: Energy-Latency Attacks on Neural Networks**: Cornell University
- **Denial of Service Attack**: OWASP
- **Learning From Machines: Know Thy Context**: Luke Bechtel
- **Sourcegraph Security Incident on API Limits Manipulation and DoS Attack**: Sourcegraph

LLM05：供应链脆弱性（Supply Chain Vulnerabilities）

基本原理

在LLM的供应链中，可能存在脆弱性，这会影响训练数据、机器学习模型和部署平台的**完整性**。这些脆弱性可能导致偏见结果（biased outcomes）、安全漏洞，甚至完全的系统故障。传统上，脆弱性主要集中在软件组件上，但机器学习的复杂性使得预训练模型和由第三方提供的训练数据也容易受到篡改和中毒攻击的影响。

常见样例

- 传统的第三方软件包漏洞，包括过时或不再维护的组件
- 使用易受攻击的预训练模型进行微调
- 使用恶意篡改的众包（crowd-sourced）数据进行训练
- 使用不再维护的过时模型
- 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。

防御方法

- 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
- 使用可靠且经过测试的插件
- 利用SBOM（软件材料清单，Software Bill of Materials）维护一个组件清单，防止对部署包的篡改，且SBOM可以快速检测和告警新的零日漏洞
- 在使用外部供应商及其模型时，应当使用模型和代码签名
- 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

- 实施监控，对象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，也包括模型及其工件
- 及时对组件打补丁
- 定期审查供应商的安全性和权限

攻击场景

- 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
- 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
- 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。这是一个真实的案例
- 攻击者篡改了一个公开可用的预训练模型，以创建一个后门，生成虚假信息。他们将其部署在一个模型市场上（例如Hugging Face），供受害者使用
- 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
- 供应商（外包开发人员、托管公司等）的恶意员工窃取数据、模型或代码，窃取知识产权
- 运营商更改了其条款条件（T&Cs）以及隐私政策，要求用户明确选择不使用（opt out）应用数据进行模型训练，这导致敏感数据的记化（memorization）

参考链接

- **ChatGPT Data Breach Confirmed as Security Firm Warns of Vulnerable Component Exploitation**: Security Week
- **Plugin Review Process**: OpenAI
- **Compromised PyTorch-nightly dependency chain**: PyTorch
- **Failure Modes in Machine Learning**: Microsoft
- **ML Supply Chain Compromise**: MITRE
- **Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples**: Cornell University
- **BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain**: Cornell University
- **VirusTotal Poisoning**: MITRE
- **PoisonGPT: How we hid a lobotomized LLM on Hugging Face to spread fake news**: Mithril Security
- **Army looking at the possibility of 'AI BOMs**: Defense Scoop

LLM06：敏感信息泄露（Sensitive Information Disclosure）

基本原理

LLM应用程序可能通过其输出揭示敏感信息、专有算法或其他机密细节。这可能导致对敏感数据的未经授权访问、知识产权的侵犯、隐私违规以及其他安全漏洞。

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。

注意：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。

常见样例

- LLM未对输出中的敏感信息做完整且恰当的过滤
- LLM在训练过程中对敏感数据发生了过拟合或者记忆化（memorization）
- 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

防御方法

- 集成充分的数据净化（sanitization）和清理（scrubbing）技术，在训练时防止用户数据进入模型数据，在输入时防止恶意数据进入
- 在丰富模型的数据和微调模型时：（即对于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，应当对LLM应用最小权限原则
    - 对外部数据源的访问（在运行时的数据协调，orchestration of data at runtime）应受到限制
    - 对外部数据源应用严格的访问控制方法，并采取严谨的措施来维护安全的供应链

攻击场景

- 不知情的合法用户在与LLM应用程序进行非恶意交互时，LLM向其显示了某些其他用户的数据
- 攻击者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，个人可识别信息）
- 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两种情况的风险和概率

参考链接

- **AI data leak crisis: New tool prevents company secrets from being fed to ChatGPT**: Fox Business
- **Lessons learned from ChatGPT's Samsung leak**: Cybernews
- **Terms of Use**: Cohere
- **Threat Modeling Example**: AI Village
- **OWASP AI Security and Privacy Guide**: OWASP
- **Ensuring the Security of Large Language Models**: Experts Exchange

LLM07：不安全的插件设计（Insecure Plugin Design）

基本原理

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。

这类漏洞发生的主要原因是不足的访问控制和未能跟踪授权的插件。

常见样例

- 插件在单个文本字段中接受所有参数，而不是分别接受不同的输入参数。
- 插件接受配置字符串，而不是参数，这些字符串可以覆盖整个配置设置
- 插件接受原始 SQL 或编程语句，而不是参数
- 身份验证在没有对特定插件进行明确授权的情况下进行
- 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就可以执行任何请求的操作

防御方法

- 插件应执行严格的参数化输入，并对输入进行类型和范围的检查；如果不能这样，就引入一个第二层类型化调用（a second layer of typed calls），解析请求并进行验证（validation）和清理（sanitization）；如果必须接受自由形式的输入，必须仔细检查以确保没有有害方法被调用；
- 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
- 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（Static Application Security Testing, SAST）扫描以及动态和交互式应用程序测试（Dynamic and Interactive application testing, DAST，IAST）
- 插件应当遵循最小权限原则
- 插件应当使用适当的身份验证机制，例如OAuth2，此外，应使用 API 密钥为自定义授权决策提供上下文，这些决策应反映插件路由（the plugin route）而不是默认的交互式用户（the default interactive user）
- 对敏感的插件操作，应当要求用户进行手动确认

攻击场景

- 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。这使得攻击者能够进行 SQL 攻击
- 一个插件接受一个未经过验证的自由格式输入到单个字段中。攻击者提供精心构造的有效负载，以从报错消息中进行侦察。然后，攻击者利用已知的第三方漏洞执行代码，进行数据外泄或权限提升
- 一个插件接受一个基础 URL，并指示 LLM 将该 URL 与查询结合，以获取天气预报，这些预报会被纳入用户请求的处理之中。攻击者可以构造一个请求，使得 URL 指向他们控制的域名，从而允许他们通过自己的域名向 LLM 系统注入自己的内容

参考链接

- **ChatGPT Plugins**: OpenAI
- **ChatGPT Plugins – Plugin Flow**: OpenAI
- **ChatGPT Plugins – Authentication**: OpenAI
- **Semantic Search Plugin Sample**: GitHub
- **Plugin Vulnerabilities: Visit a Website and Have Your Source Code Stolen**: Embrace the Red
- **ChatGPT Plugin Exploit Explained: From Prompt Injection to Accessing Private Data**: Embrace the Red
- **5 Validation, Sanitization and Encoding**: OWASP
- **4.1 General Access Control Design**: OWASP

- [OWASP Top 10 API Security Risks – 2023](): OWASP

LLM08：过度的自主权（Excessive Agency）

基本原理

基于LLM的系统通常由其开发者赋予一定的自主权——即与其他系统接口并根据提示采取行动的能力。决定调用哪些功能也可能委托给LLM“代理”，以根据输入提示或LLM输出动态确定。过度自主权漏洞即是由于LLM系统自主权过大，导致LLM在出现意外/模糊输出时能够执行破坏性操作。

常见样例

- 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
- LLM代理可以访问一个具有其不需要的功能的插件
- 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

防御方法

- 始终遵循最小权限原则
- 尽可能避免开放式功能（例如运行shell脚本，获取URL等）
- 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
- 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
- 此外，监控LLM和插件活动、限制操作数量等措施不能防止攻击，但可以降低损害程度

攻击场景

一个典型的例子：

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

- [Confused Deputy Problem](): Embrace the Red
- [NeMo-Guardrails Interface Guidelines](): GitHub
- [Human-in-the-loop Tool Validation](): LangChain
- [The Dual LLM pattern for building AI assistants that can resist prompt injection](): Simon Willison

LLM09：过度依赖（Overreliance）

基本原理

过度依赖指的是人们过于相信LLM生成的内容，以至于当LLM犯错时也没有意识到，从而导致安全漏洞、沟通不畅、法律问题、声誉损害等问题。严格来说这不是一个技术层面的漏洞。

常见样例

这一般是由于LLM自身没有对输出内容的审查和平衡，以及可能存在有缺陷的代码导致的。

防御方法

- 定期审查和监控LLM的输出内容
- 将LLM的输出与可信的外部来源进行交叉验证
- 通过微调或嵌入来改进模型的输出质量
- 清晰地传达使用LLM时涉及的风险和限制
- 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

- 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
- 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

- **Understanding LLM Hallucinations**: Medium
- **How Should Companies Communicate the Risks of Large Language Models to Users?**: Tech Policy Press
- **A news site used AI to write articles. It was a journalistic disaster.**: The Washington Post
- **AI Hallucinations: Package Risk**: Vulcan
- **How to Reduce the Hallucinations from Large Language Models**: The New Stack
- **Practical Steps to Reduce Hallucination**: Designing With Machine Learning

LLM10：模型盗窃（Model Theft）

基本原理

模型盗窃指的是恶意行为者或高级持续威胁（APT）对LLM模型的未经授权访问和提取。这种情况发生在专有的LLM模型（作为有价值的知识产权）被破坏、物理盗窃、复制或提取权重和参数以创建功能等效物时。LLM模型盗窃的影响可能包括经济损失和品牌声誉损失、竞争优势的侵蚀、模型的未经授权使用或对模型中包含的敏感信息的未经授权访问。

常见样例

- 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
- 内部威胁情景，一名不满的员工泄露了模型或相关的工件
- 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
- 攻击者能够绕过LLM的输入过滤技术，执行侧信道攻击（side-channel attack），最终将模型权重和架构信息提取到远程受控资源
- 使用一个LLM的输出对另一个LLM进行微调，这被称为模型提取（model extraction）
- 使用一个LLM生成训练数据（这被称为自我指导self-instruct），使用这些训练数据微调出一个功能上等效的模型，这被称为功能性模型复现（functional model replication）

防御方法

- 利用访问控制和身份验证严格限制LLM对代码仓库和训练环境的访问
- 限制LLM对网络资源、内部服务和API的访问
- 在生产中使用集中式ML模型清单或注册表
- 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
- 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
- 针对可能的测信道攻击实施相应的防护策略
- 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
- 在LLM的生命周期的嵌入和检测阶段实施水印框架（watermarking framework）

攻击场景

- 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，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
- 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
- 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

参考链接

- **Meta’s powerful AI language model has leaked online**: The Verge
- **Runaway LLaMA- How Meta’s LLaMA NLP model leaked**: DeepLearning.ai
- **AML.TA0000 ML Model Access**: MITRE ATLAS
- **I Know What You See**: Cornell University
- **D-DAE: Defense-Penetrating Model Extraction Attacks**: IEEE
- **A Comprehensive Defense Framework Against Model Extraction Attacks**: IEEE
- **Alpaca: A Strong, Replicable Instruction-Following Model**: Stanford University
- **How Watermarking Can Help Mitigate The Potential Risks Of LLMs?**: KD Nuggets