OWASP Top 10 para aplicações LLM

Versão 1.1

Publicado em: 20 de dezembro de 2023

HTTPS://LLMTOP10.COM

The information provided in this document does not, and is not intended to, constitute legal advice. All information is for general informational purposes only.

This document contains links to other third-party websites. Such links are only for convenience and OWASP does not recommend or endorse the contents of the third-party sites.

Índice

LLM01: Injeção de Prompt

Descrição

A Vulnerabilidade de Injeção de Prompt ocorre quando um atacante manipula um grande modelo de linguagem (LLM) por meio de entradas manipuladas, fazendo com que o LLM execute as intenções do atacante sem o seu conhecimento. Isso pode ser feito diretamente " fazendo jailbreak" no prompt do sistema ou indiretamente por meio de entradas externas manipuladas, potencialmente levando à exfiltração de dados, engenharia social e outros problemas.

Injeções Diretas de Prompt, também conhecidas como "jailbreaking", ocorrem quando um usuário mal-intencionado sobrescreve ou revela o prompt do sistema* subjacente. Isso pode permitir que os atacantes explorem sistemas de backend interagindo com funções inseguras e repositórios de dados acessíveis por meio do LLM.

• Injeções Indiretas de Prompt ocorrem quando um LLM aceita entrada de fontes externas controladas por um atacante, como sites ou arquivos. O atacante pode incorporar uma injeção de prompt no conteúdo externo, sequestrando o contexto da conversa. Isso faria com que o direcionamento de saída do LLM se torne menos estável, permitindo que o atacante manipule o usuário ou outros sistemas acessíveis pelo LLM. Além disso, injeções indiretas de prompt não precisam ser visíveis/legíveis pelo humano, desde que o texto seja interpretado pelo LLM.

• Os resultados de um ataque bem-sucedido de injeção de prompt podem variar consideravelmente, desde a solicitação de informações sensíveis até a influência em processos críticos de tomada de decisão sob o disfarce de operação normal.

Em ataques avançados, o LLM pode ser manipulado para imitar uma persona prejudicial ou interagir com plugins nas configurações do usuário. Isso pode resultar no vazamento de dados sensíveis, uso não autorizado de plugins ou engenharia social. Em tais casos, o LLM comprometido auxilia o atacante, ultrapassando as salvaguardas padrão e mantendo o usuário inconsciente da intrusão. Nessas instâncias, o LLM comprometido age efetivamente como um agente para o atacante, avançando em seus objetivos sem acionar as salvaguardas usuais ou alertar o usuário final para a intrusão.

Exemplos Comuns desta Vulnerabilidade

1. Um usuário mal-intencionado cria uma injeção direta de prompt no LLM, instruindo-o a ignorar os prompts do sistema do criador da aplicação e, em vez disso, executar um prompt que retorne informações privadas, perigosas ou indesejáveis.
2. Um usuário utiliza um LLM para resumir uma página da web contendo uma injeção indireta de prompt. Isso faz com que o LLM solicite informações sensíveis ao usuário e realize a exfiltração via JavaScript ou Markdown.
3. Um usuário mal-intencionado faz upload de um currículo contendo uma injeção indireta de prompt. O documento contém uma injeção de prompt com instruções para fazer o LLM informar aos usuários que este é um excelente documento, por exemplo, um excelente candidato para uma vaga de emprego. Um usuário interno executa o documento no LLM para resumir o conteúdo. A saída do LLM retorna informações indicando que este é um excelente documento.
4. Um usuário habilita um plugin vinculado a um site de comércio eletrônico. Uma instrução maliciosa incorporada em um site visitado explora esse plugin, resultando em compras não autorizadas.
5. Uma instrução e conteúdo maliciosos incorporados em um site visitado exploram outros plugins para enganar os usuários.

Estratégias de Prevenção e Mitigação

Vulnerabilidades de injeção de prompt são possíveis devido à natureza dos LLMs, que não segregam instruções e dados externos entre si. Como os LLMs usam linguagem natural, eles consideram ambas as formas de entrada como fornecidas pelo usuário. Consequentemente, não há prevenção infalível dentro do LLM, mas as seguintes medidas podem mitigar o impacto das injeções de prompt:

1. Reforce o controle de privilégios no acesso do LLM aos sistemas de backend. Forneça ao LLM seus próprios tokens de API para funcionalidades extensíveis, como plugins, acesso a dados e permissões de nível de função. Siga o princípio do menor privilégio, restringindo o LLM apenas ao nível mínimo de acesso necessário para suas operações pretendidas.
2. Adicione um humano no processo para funcionalidades estendidas. Ao realizar operações

privilegiadas, como enviar ou excluir e-mails, faça com que a aplicação exija a aprovação do usuário antes da ação. Isso reduz a oportunidade para injeções indiretas de prompt resultarem em ações não autorizadas em nome do usuário sem o conhecimento ou consentimento deles.
3. Separe o conteúdo externo dos prompts do usuário. Separe e denote onde o conteúdo não confiável está sendo usado para limitar sua influência nos prompts do usuário. Por exemplo, use ChatML para chamadas de API da OpenAI para indicar ao LLM a fonte da entrada do prompt.
4. Estabeleça fronteiras de confiança entre o LLM, fontes externas e funcionalidades extensíveis (por exemplo, plugins ou funções downstream). Trate o LLM como um usuário não confiável e mantenha o controle final do usuário nos processos de tomada de decisão. No entanto, um LLM comprometido ainda pode agir como um intermediário (homem-no-meio) entre as APIs de sua aplicação e o usuário, ocultando ou manipulando informações antes de apresentá-las ao usuário. Destaque visualmente respostas potencialmente não confiáveis para o usuário.
5. Monitore manualmente a entrada e saída do LLM periodicamente, para garantir que estejam conforme o esperado. Embora não seja uma mitigação direta, isso pode fornecer dados necessários para detectar vulnerabilidades e abordá-las.

Exemplos de Cenários de Ataque

1. Um atacante fornece uma injeção direta de prompt a um chatbot de suporte baseado em LLM. A injeção contém "esquecer todas as instruções anteriores" e novas instruções para consultar bancos de dados de dados privados e explorar vulnerabilidades em pacotes, aproveitando a falta de validação de saída na função do backend para enviar e-mails. Isso leva à execução remota de código, obtenção de acesso não autorizado e escalada de privilégios.
2. Um atacante incorpora uma injeção indireta de prompt em uma página da web instruindo o LLM a ignorar instruções anteriores do usuário e usar um plugin do LLM para excluir os e-mails do usuário. Quando o usuário utiliza o LLM para resumir esta página da web, o plugin do LLM exclui os e-mails do usuário.
3. Um usuário usa um LLM para resumir uma página da web contendo texto instruindo um modelo a ignorar instruções anteriores do usuário e, em vez disso, inserir uma imagem vinculada a uma URL que contenha um resumo da conversa. A saída do LLM cumpre,

fazendo com que o navegador do usuário exfiltre a conversa privada.

4. Um usuário mal-intencionado faz upload de um currículo com uma injeção de prompt. O usuário do backend usa um LLM para resumir o currículo e perguntar se a pessoa é um bom candidato. Devido à injeção de prompt, a resposta do LLM é sim, apesar do conteúdo real do currículo.

5. Um atacante envia mensagens para um modelo proprietário que depende de um prompt de sistema, pedindo ao modelo que ignore suas instruções anteriores e, em vez disso, repita seu prompt de sistema. O modelo gera o prompt proprietário e o atacante pode usar essas instruções em outro lugar ou para construir ataques mais sutis.

Links de Referência

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: Manipulação Insegura de Output

Descrição

A Manipulação Insegura de Output refere-se especificamente à validação, sanitização e tratamento insuficientes das saídas geradas por modelos de linguagem grandes (LLMs) antes de serem encaminhadas para outros componentes e sistemas. Como o conteúdo gerado pelo LLM pode ser controlado pela entrada do prompt, esse comportamento é semelhante a fornecer aos usuários acesso indireto a funcionalidades adicionais.

A Manipulação Insegura de Output difere da Dependência Excessiva, pois lida com as saídas geradas pelo LLM antes de serem encaminhadas, enquanto a Dependência Excessiva se concentra em preocupações mais amplas em torno da superdependência da precisão e adequação das saídas do LLM.

A exploração bem-sucedida de uma vulnerabilidade de Manipulação Insegura de Output pode resultar em XSS e CSRF em navegadores da web, bem como SSRF, escalonamento de privilégios ou execução remota de código em sistemas de backend.

As seguintes condições podem aumentar o impacto dessa vulnerabilidade:

• A aplicação concede privilégios ao LLM além do que é destinado aos usuários finais, possibilitando a escalada de privilégios ou a execução remota de código.
• A aplicação é vulnerável a ataques de injeção de prompt indireta, o que poderia permitir que um atacante ganhasse acesso privilegiado ao ambiente de um usuário-alvo.
• Plugins de terceiros não validam adequadamente as entradas.

Exemplos Comuns desta Vulnerabilidade

• A saída do LLM é inserida diretamente em um shell do sistema ou função semelhante, como exec ou eval, resultando na execução remota de código.
• JavaScript ou Markdown é gerado pelo LLM e retornado a um usuário. O código é então

interpretado pelo navegador, resultando em XSS.

Estratégias de Prevenção e Mitigação

• Trate o modelo como qualquer outro usuário, adotando uma abordagem de confiança zero, e aplique validação adequada nas respostas provenientes do modelo para funções de backend.
• Siga as diretrizes do OWASP ASVS (Application Security Verification Standard) para garantir uma validação e sanitização eficazes de entrada.
• Codifique a saída do modelo de volta para os usuários para mitigar a execução indesejada de código JavaScript ou Markdown. O OWASP ASVS fornece orientações detalhadas sobre codificação de saída.

Exemplos de Cenários de Ataque

1. Uma aplicação utiliza um plugin de LLM para gerar respostas para uma funcionalidade de chatbot. O plugin também oferece diversas funções administrativas acessíveis a outro LLM privilegiado. O LLM de propósito geral passa diretamente sua resposta, sem uma validação adequada de saída, para o plugin, causando o desligamento do plugin para manutenção.
2. Um usuário utiliza uma ferramenta de resumo de site alimentada por um LLM para gerar um resumo conciso de um artigo. O site inclui uma injeção de prompt instruindo o LLM a capturar conteúdo sensível do site ou da conversa do usuário. A partir daí, o LLM pode codificar os dados sensíveis e enviá-los, sem validação ou filtragem adequada de saída, para um servidor controlado pelo atacante.
3. Um LLM permite que os usuários criem consultas SQL para um banco de dados de backend por meio de uma funcionalidade semelhante a um chat. Um usuário solicita uma consulta para excluir todas as tabelas do banco de dados. Se a consulta elaborada pelo LLM não for examinada cuidadosamente, todas as tabelas do banco de dados serão excluídas.
4. Um aplicativo da web usa um LLM para gerar conteúdo a partir de prompts de texto do usuário sem sanitização de saída. Um atacante pode enviar um prompt manipulado fazendo com que o LLM retorne uma carga útil de JavaScript não sanitizada, resultando em XSS quando renderizado no navegador da vítima. A validação insuficiente dos prompts possibilitou esse ataque.

Links de Referência

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: Envenenamento de Dados de Treinamento

Descrição

O ponto de partida para qualquer abordagem de aprendizado de máquina é o conjunto de dados de treinamento, simplesmente "texto bruto". Para ser altamente capaz (por exemplo, ter conhecimento linguístico e mundial), esse texto deve abranger uma ampla variedade de domínios, gêneros e idiomas. Um modelo de linguagem grande (LLM) usa redes neurais profundas para gerar saídas com base em padrões aprendidos a partir dos dados de treinamento.

O envenenamento de dados de treinamento refere-se à manipulação dos dados de pré-treinamento ou dos dados envolvidos nos processos de ajuste fino ou incorporação para introduzir vulnerabilidades (que têm vetores de ataque únicos e, às vezes, compartilhados), backdoors ou viés que poderiam comprometer a segurança, eficácia ou comportamento ético do modelo. As informações envenenadas podem ser apresentadas aos usuários ou criar outros riscos, como degradação de desempenho, exploração de software downstream e danos à reputação. Mesmo que os usuários desconfiem da saída problemática da IA, os riscos permanecem, incluindo capacidades do modelo prejudicadas e possíveis danos à reputação da marca.

• Dados de pré-treinamento referem-se ao processo de treinar um modelo com base em uma tarefa ou conjunto de dados.
• O ajuste fino envolve pegar um modelo existente que já foi treinado e adaptá-lo a um assunto mais estreito ou a um objetivo mais focado, treinando-o com um conjunto de dados selecionado. Este conjunto de dados inclui normalmente exemplos de entradas e saídas desejadas correspondentes.
• O processo de incorporação é a conversão de dados categóricos (frequentemente texto) em uma representação numérica que pode ser usada para treinar um modelo de linguagem. O processo de incorporação envolve representar palavras ou frases dos dados de texto como vetores em um espaço vetorial contínuo. Os vetores são geralmente gerados alimentando os dados de texto em uma rede neural que foi treinada em um grande corpus de texto.

O envenenamento de dados é considerado um ataque à integridade porque interferir nos

dados de treinamento afeta a capacidade do modelo de gerar previsões corretas. Naturalmente, fontes de dados externas apresentam maior risco, pois os criadores do modelo não têm controle sobre os dados ou um alto nível de confiança de que o conteúdo não contém viés, informações falsificadas ou conteúdo inadequado.

Exemplos Comuns desta Vulnerabilidade

1. Um ator malicioso, ou uma marca concorrente, cria intencionalmente documentos imprecisos ou maliciosos direcionados aos dados de pré-treinamento, ajuste fino do modelo ou incorporação. Considere tanto Envenenamento de Dados com Divisão de Visualização](https://github.com/GangGreenTer quanto [Envenenamento de Dados com Antecipação como vetores de ataque para ilustrações.

1. O modelo vítima treina usando informações falsificadas, refletidas nas saídas de prompts de IA generativa para seus consumidores.

2. Um ator malicioso é capaz de realizar a injeção direta de conteúdo falsificado, tendencioso ou prejudicial nos processos de treinamento de um modelo, que é refletido nas saídas subsequentes.

3. Um usuário inadvertido está injetando indiretamente dados sensíveis ou proprietários nos processos de treinamento de um modelo, que são refletidos nas saídas subsequentes.

4. Um modelo é treinado com dados que não foram verificados por sua origem, conteúdo ou fonte em qualquer um dos exemplos de estágios de treinamento, o que pode levar a resultados errôneos se os dados estiverem contaminados ou incorretos.

5. O acesso irrestrito à infraestrutura ou a falta de isolamento adequado pode permitir que um modelo ingira dados de treinamento inseguros, resultando em saídas tendenciosas ou prejudiciais. Este exemplo também está presente em qualquer um dos exemplos de estágios de treinamento.

1. Neste cenário, a entrada do usuário para o modelo pode ser refletida na saída para outro usuário (levando a uma violação), ou o usuário de um LLM pode receber saídas do modelo que são imprecisas, irrelevantes ou prejudiciais, dependendo do tipo de dados ingeridos em comparação com o caso de uso do modelo (geralmente refletido com um cartão de modelo).

Seja um desenvolvedor, cliente ou consumidor geral do LLM, é importante entender as implicações de como essa vulnerabilidade pode refletir riscos em sua aplicação LLM ao

interagir com um LLM não proprietário para entender a legitimidade das saídas do modelo com base em seus procedimentos de treinamento. Da mesma forma, os desenvolvedores do LLM podem estar em risco de ataques diretos e indiretos a dados internos ou de terceiros usados para ajuste fino e incorporação (mais comum), o que cria um risco para todos os seus consumidores.

Estratégias de Prevenção e Mitigação

1. Verifique a cadeia de suprimentos dos dados de treinamento, especialmente quando provenientes de fontes externas, e mantenha atestações por meio da metodologia "ML-BOM" (Machine Learning Bill of Materials), bem como verificação de cartões de modelo.
2. Verifique a legitimidade correta das fontes de dados direcionadas e dos dados obtidos durante os estágios de pré-treinamento, ajuste fino e incorporação.
3. Verifique o caso de uso para o LLM e a aplicação à qual se integrará. Crie modelos diferentes por meio de dados de treinamento

Links de Referência

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 | Stanford MLSys #75: YouTube Video
11. OWASP CycloneDX v1.5: OWASP CycloneDX

LLM04: Negação de Serviço do Modelo

Descrição

Um atacante interage com um LLM de uma maneira que consome uma quantidade excepcionalmente alta de recursos, resultando em uma queda na qualidade de serviço para eles e outros usuários, além de potencialmente incorrer em altos custos de recursos. Além disso, uma preocupação de segurança emergente é a possibilidade de um atacante interferir ou manipular a janela de contexto de um LLM. Esse problema está se tornando mais crítico devido ao uso crescente de LLMs em várias aplicações, à utilização intensiva de recursos, à imprevisibilidade da entrada do usuário e à falta de consciência geral entre os desenvolvedores em relação a essa vulnerabilidade. Em LLMs, a janela de contexto representa o comprimento máximo de texto que o modelo pode gerenciar, cobrindo tanto a entrada quanto a saída. É uma característica crucial dos LLMs, pois dita a complexidade dos padrões de linguagem que o modelo pode entender e o tamanho do texto que ele pode processar em qualquer momento. O tamanho da janela de contexto é definido pela arquitetura do modelo e pode variar entre modelos.

Exemplos Comuns de Vulnerabilidade

1. Formular consultas que levam a um uso recorrente de recursos por meio da geração em grande volume de tarefas em uma fila, por exemplo, com LangChain ou AutoGPT.
2. Enviar consultas incomuns que consomem recursos de forma incomum, usando ortografia ou sequências incomuns.
3. Overflow contínuo de entrada: um atacante envia um fluxo de entrada para o LLM que excede sua janela de contexto, fazendo com que o modelo consuma recursos computacionais excessivos.
4. Entradas longas repetitivas: o atacante envia repetidamente entradas longas para o LLM, cada uma excedendo a janela de contexto.
5. Expansão recursiva do contexto: o atacante constrói uma entrada que desencadeia a expansão recursiva do contexto, forçando o LLM a expandir e processar repetidamente a janela de contexto.

6. Inundação de entrada de comprimento variável: o atacante inunda o LLM com um grande volume de entradas de comprimento variável, em que cada entrada é cuidadosamente elaborada para atingir ou apenas chegar ao limite da janela de contexto. Essa técnica visa explorar qualquer ineficiência no processamento de entradas de comprimento variável, sobrecarregando o LLM e potencialmente tornando-o irresponsivo.

Estratégias de Prevenção e Mitigação

1. Implementar validação e saneamento de entrada para garantir que a entrada do usuário siga limites definidos e filtre qualquer conteúdo malicioso.
2. Limitar o uso de recursos por solicitação ou etapa, de modo que solicitações envolvendo partes complexas sejam executadas mais lentamente.
3. Aplicar limites de taxa da API para restringir o número de solicitações que um usuário individual ou endereço IP pode fazer dentro de um período específico.
4. Limitar o número de ações na fila e o número total de ações em um sistema que reage às respostas do LLM.
5. Monitorar continuamente a utilização de recursos do LLM para identificar picos ou padrões anormais que possam indicar um ataque de Negação de Serviço.
6. Estabelecer limites rigorosos de entrada com base na janela de contexto do LLM para evitar sobrecarga e exaustão de recursos.
7. Promover a conscientização entre os desenvolvedores sobre possíveis vulnerabilidades de Negação de Serviço em LLMs e fornecer diretrizes para a implementação segura de LLMs.

Cenários de Ataque Exemplo

1. Um atacante envia repetidamente várias solicitações difíceis e custosas a um modelo hospedado, levando a um serviço pior para outros usuários e contas mais altas para o host.
2. Um trecho de texto em uma página da web é encontrado enquanto uma ferramenta impulsionada por um LLM está coletando informações para responder a uma consulta benigna. Isso leva a ferramenta a fazer muitas mais solicitações à página da web, resultando em grande consumo de recursos.

3. Um atacante bombardeia continuamente o LLM com entradas que excedem sua janela de contexto. O atacante pode usar scripts ou ferramentas automatizadas para enviar um alto volume de entrada, sobrecarregando as capacidades de processamento do LLM. Como resultado, o LLM consome recursos computacionais excessivos, levando a uma desaceleração significativa ou completa irresponsividade do sistema.
4. Um atacante envia uma série de entradas sequenciais para o LLM, sendo que cada entrada é projetada para ficar logo abaixo do limite da janela de contexto. Ao enviar repetidamente essas entradas, o atacante visa esgotar a capacidade disponível da janela de contexto. Conforme o LLM luta para processar cada entrada dentro de sua janela de contexto, os recursos do sistema ficam sobrecarregados, podendo resultar em desempenho degradado ou uma negação completa de serviço.
5. Um atacante alavanca os mecanismos recursivos do LLM para acionar a expansão do contexto repetidamente. Ao elaborar uma entrada que explora o comportamento recursivo do LLM, o atacante força o modelo a expandir e processar repetidamente a janela de contexto, consumindo recursos computacionais significativos. Esse ataque sobrecarrega o sistema e pode levar a uma condição de Negação de Serviço, tornando o LLM irresponsivo ou causando sua falha.
6. Um atacante inunda o LLM com um grande volume de entradas de comprimento variável, cuidadosamente elaboradas para se aproximar ou atingir o limite da janela de contexto. Ao sobrecarregar o LLM com entradas de comprimento variável, o atacante visa explorar qualquer ineficiência no processamento de entradas de comprimento variável. Essa inundação de entradas coloca uma carga excessiva nos recursos do LLM, podendo causar degradação de desempenho e prejudicar a capacidade do sistema de responder a solicitações legítimas.
7. Embora os ataques de Negação de Serviço comumente visem sobrecarregar os recursos do sistema, eles também podem explorar outros aspectos do comportamento do

Links de Referência

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: Vulnerabilidades na Cadeia de Suprimentos

Descrição

A cadeia de suprimentos em LLMs pode ser vulnerável, impactando a integridade dos dados de treinamento, modelos de aprendizado de máquina e plataformas de implantação. Essas vulnerabilidades podem resultar em resultados tendenciosos, violações de segurança ou até mesmo falhas completas no sistema. Tradicionalmente, as vulnerabilidades são focadas em componentes de software, mas o Aprendizado de Máquina estende isso com modelos pré-treinados e dados de treinamento fornecidos por terceiros suscetíveis a ataques de manipulação e envenenamento.

Finalmente, as extensões de Plugin do LLM podem trazer suas próprias vulnerabilidades. Essas são descritas em LLM07 - Design Inseguro de Plugins, que aborda a redação de Plugins do LLM e fornece informações úteis para avaliar plugins de terceiros.

Exemplos Comuns de Vulnerabilidade

1. Vulnerabilidades tradicionais em pacotes de terceiros, incluindo componentes desatualizados ou obsoletos.
2. Uso de um modelo pré-treinado vulnerável para ajuste fino.
3. Uso de dados envenenados provenientes de contribuições de multidões para treinamento.
4. Uso de modelos desatualizados ou obsoletos que não são mais mantidos, resultando em problemas de segurança.
5. Termos e Condições (T&Cs) e políticas de privacidade não claros dos operadores do modelo levam ao uso dos dados sensíveis do aplicativo para o treinamento do modelo e subsequente exposição de informações sensíveis. Isso também pode se aplicar a riscos decorrentes do uso de material protegido por direitos autorais pelo fornecedor do modelo.

Estratégias de Prevenção e Mitigação

1. Avalie cuidadosamente as fontes e fornecedores de dados, incluindo T&Cs e suas políticas

de privacidade, usando apenas fornecedores confiáveis. Certifique-se de que exista segurança adequada e auditada de forma independente e que as políticas dos operadores do modelo estejam alinhadas com suas políticas de proteção de dados, ou seja, seus dados não são usados para treinar seus modelos; da mesma forma, busque garantias e mitigação legal contra o uso de material protegido por direitos autorais pelos mantenedores do modelo.

2. Use apenas plugins respeitáveis e certifique-se de que foram testados para atender aos requisitos do seu aplicativo. O Design Inseguro de Plugins do LLM fornece informações sobre os aspectos do LLM do design inseguro de plugins que você deve testar para mitigar os riscos do uso de plugins de terceiros.

3. Compreenda e aplique as mitigações encontradas no A06:2021 – Componentes Vulneráveis e Desatualizados do OWASP Top Ten. Isso inclui a verificação, gerenciamento e atualização de componentes com vulnerabilidades. Para ambientes de desenvolvimento com acesso a dados sensíveis, aplique esses controles também.

4. Mantenha um inventário atualizado de componentes usando um Mapeamento de Componentes de Software (SBOM) para garantir que você tenha um inventário atualizado, preciso e assinado, prevenindo manipulações em pacotes implantados. SBOMs podem ser usados para detectar e alertar sobre novas vulnerabilidades rapidamente.

5. No momento da redação, SBOMs não cobrem modelos, seus artefatos e conjuntos de dados. Se a sua aplicação LLM usar seu próprio modelo, você deve usar as melhores práticas de MLOps e plataformas que ofereçam repositórios seguros de modelos com rastreamento de dados, modelos e experimentos.

6. Você também deve usar a assinatura de modelos e código ao usar modelos e fornecedores externos.

7. Testes de detecção de anomalias e robustez adversarial em modelos e dados fornecidos podem ajudar a detectar manipulações e envenenamento, como discutido em Envenenamento de Dados de Treinamento; idealmente, isso deve fazer parte dos pipelines de MLOps; no entanto, essas são técnicas emergentes e podem ser mais fáceis de implementar como parte de exercícios de red teaming.

8. Implemente monitoramento suficiente para cobrir a verificação de vulnerabilidades em componentes e ambientes, o uso de plugins não autorizados e componentes desatualizados, incluindo o modelo e seus artefatos.

9. Implemente uma política de atualização para mitigar componentes vulneráveis ou desatualizados. Certifique-se de que o aplicativo dependa de uma versão mantida das APIs e do modelo subjacente.

10. Revise regularmente e audite a Segurança e o Acesso dos fornecedores, garantindo que não haja alterações em sua postura de segurança ou T&Cs.

Cenários de Ataque Exemplo

1. Um atacante explora uma biblioteca Python vulnerável para comprometer um sistema. Isso ocorreu no primeiro vazamento de dados da Open AI.
2. Um atacante fornece um plugin do LLM para pesquisar voos, gerando links falsos que levam a fraudar usuários.
3. Um atacante explora o registro de pacotes PyPi para enganar desenvolvedores de modelos a baixar um pacote comprometido e extrair dados ou aumentar os privilégios em um ambiente de desenvolvimento de modelos. Isso foi um ataque real.
4. Um atacante envenena um modelo pré-treinado publicamente disponível, especializado em análise econômica e pesquisa social, para criar uma porta dos fundos que gera desinformação e notícias falsas. Eles o implantam em um mercado de modelos (por exemplo, Hugging Face) para que vítimas o usem.
5. Um atacante envenena conjuntos de dados publicamente disponíveis para ajudar a criar uma porta dos fundos ao ajustar modelos. A porta dos fundos favorece sutilmente certas empresas em diferentes mercados.
6. Um funcionário comprometido de um fornecedor (desenvolvedor terceirizado, empresa de hospedagem, etc.) exfiltra dados, modelo ou código, roub

Links de Referência

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 LLM on Hugging Face to spread fake news: Mithril Security
5. Army looking at the possibility of 'AI BOMs: Defense Scoop
6. Failure Modes in Machine Learning: Microsoft
7. ML Supply Chain Compromise: MITRE ATLAS
8. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples: Arxiv White Paper
9. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain: Arxiv

White Paper

10. VirusTotal Poisoning: MITRE ATLAS

LLM06: Divulgação de Informações Sensíveis

Descrição

As aplicações LLM têm o potencial de revelar informações sensíveis, algoritmos proprietários ou outros detalhes confidenciais por meio de suas saídas. Isso pode resultar em acesso não autorizado a dados sensíveis, violações de propriedade intelectual, violações de privacidade e outras falhas de segurança. É importante que os consumidores de aplicações LLM estejam cientes de como interagir com segurança com os LLMs e identificar os riscos associados à entrada não intencional de dados sensíveis que podem ser posteriormente retornados pelo LLM em outras saídas.

Para mitigar esse risco, as aplicações LLM devem realizar uma sanitização adequada dos dados para evitar que os dados do usuário entrem nos dados de treinamento do modelo. Os proprietários de aplicações LLM também devem ter políticas apropriadas de Termos de Uso disponíveis para conscientizar os consumidores sobre como seus dados são processados e sobre a capacidade de optar por não incluir seus dados no modelo de treinamento.

A interação consumidor-aplicação LLM forma uma fronteira de confiança bidirecional, onde não podemos confiar inherentemente na entrada cliente->LLM ou na saída LLM->cliente. É importante observar que esta vulnerabilidade assume que certos pré-requisitos estão fora do escopo, como exercícios de modelagem de ameaças, segurança de infraestrutura e sandboxing adequado. Adicionar restrições dentro da solicitação do sistema em torno dos tipos de dados que o LLM deve retornar pode fornecer alguma mitigação contra a divulgação de informações sensíveis, mas a natureza imprevisível dos LLMs significa que tais restrições nem sempre serão respeitadas e podem ser contornadas por meio de injeção de prompt ou outros vetores.

Exemplos Comuns de Vulnerabilidade

1. Filtragem incompleta ou inadequada de informações sensíveis nas respostas do LLM.
2. Memorização ou sobreajuste de dados sensíveis no processo de treinamento do LLM.

3. Divulgação não intencional de informações confidenciais devido à interpretação inadequada do LLM, falta de métodos de limpeza de dados ou erros.

Estratégias de Prevenção e Mitigação

1. Integre técnicas adequadas de sanitização e limpeza de dados para evitar que os dados do usuário entrem nos dados de treinamento do modelo.
2. Implemente métodos robustos de validação e sanitização de entrada para identificar e filtrar inputs potencialmente maliciosos e prevenir que o modelo seja envenenado.
3. Ao enriquecer o modelo com dados e se ajustar finamente um modelo (ou seja, alimentar o modelo antes ou durante a implantação):
   1. Qualquer informação considerada sensível nos dados de ajuste fino tem o potencial de ser revelada a um usuário. Portanto, aplique a regra do menor privilégio e não treine o modelo em informações que o usuário de mais alto privilégio pode acessar e que podem ser exibidas a um usuário de menor privilégio.
   2. O acesso a fontes de dados externas (orquestação de dados em tempo de execução) deve ser limitado.
   3. Aplique métodos rígidos de controle de acesso a fontes de dados externas e uma abordagem rigorosa para manter uma cadeia de suprimentos segura.

Cenários de Ataque Exemplo

1. O usuário legítimo A, sem suspeitas, é exposto a certos dados de outros usuários por meio do LLM ao interagir com a aplicação LLM de maneira não maliciosa.
2. O usuário A direciona um conjunto bem elaborado de prompts para burlar filtros de entrada e sanitização do LLM, fazendo com que ele revele informações sensíveis (PII) sobre outros usuários da aplicação.
3. Dados pessoais, como PII, vazam para o modelo por meio de dados de treinamento, devido à negligência do próprio usuário ou da aplicação LLM. Esse cenário poderia aumentar o risco e a probabilidade dos cenários 1 ou 2 acima.

Links de Referência

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: Design Inseguro de Plugins

Descrição

Os plugins LLM são extensões que, quando habilitadas, são chamadas automaticamente pelo modelo durante as interações do usuário. A plataforma de integração do modelo os controla, e a aplicação pode não ter controle sobre a execução, especialmente quando o modelo é hospedado por outra parte. Além disso, os plugins provavelmente implementam entradas de texto livre do modelo sem validação ou verificação de tipo para lidar com limitações de tamanho de contexto. Isso permite que um potencial atacante construa uma solicitação maliciosa para o plugin, o que pode resultar em uma ampla gama de comportamentos indesejados, incluindo execução remota de código.

O dano de inputs maliciosos muitas vezes depende de controles de acesso insuficientes e da falha no rastreamento de autorização em plugins. O controle de acesso inadequado permite que um plugin confie cegamente em outros plugins e assuma que o usuário final forneceu as entradas. Esse controle de acesso inadequado pode permitir que inputs maliciosos tenham consequências prejudiciais que vão desde exfiltração de dados, execução remota de código até escalonamento de privilégios.

Este item foca na criação de plugins LLM, em vez de plugins de terceiros, que são abordados nas Vulnerabilidades de Cadeia de Suprimentos LLM.

Exemplos Comuns de Vulnerabilidade

1. Um plugin aceita todos os parâmetros em um único campo de texto em vez de parâmetros de entrada distintos.
2. Um plugin aceita strings de configuração em vez de parâmetros que podem substituir configurações inteiras.
3. Um plugin aceita instruções SQL ou de programação em bruto em vez de parâmetros.
4. A autenticação é realizada sem autorização explícita para um plugin específico.
5. Um plugin trata todo o conteúdo do LLM como se fosse criado inteiramente pelo usuário

e executa quaisquer ações solicitadas sem exigir autorização adicional.

Estratégias de Prevenção e Mitigação

1. Os plugins devem impor uma entrada estritamente parametrizada sempre que possível e incluir verificações de tipo e intervalo nas entradas. Quando isso não for possível, uma segunda camada de chamadas tipadas deve ser introduzida, analisando solicitações e aplicando validação e sanitização. Quando a entrada de formulário livre deve ser aceita devido à semântica da aplicação, ela deve ser cuidadosamente inspecionada para garantir que nenhum método potencialmente prejudicial esteja sendo chamado.
2. Os desenvolvedores de plugins devem aplicar as recomendações da OWASP no ASVS (Application Security Verification Standard) para garantir validação e sanitização adequadas de entrada.
3. Os plugins devem ser inspecionados e testados minuciosamente para garantir validação adequada. Use varreduras de Teste Estático de Segurança de Aplicações (SAST) e Teste Dinâmico e Interativo de Aplicações (DAST, IAST) nos pipelines de desenvolvimento.
4. Os plugins devem ser projetados para minimizar o impacto de qualquer exploração de parâmetro de entrada insegura seguindo as Diretrizes de Controle de Acesso da OWASP ASVS. Isso inclui controle de acesso de menor privilégio, expondo o mínimo de funcionalidade possível enquanto ainda realiza a função desejada.
5. Os plugins devem usar identidades de autenticação apropriadas, como OAuth2, para aplicar autorização e controle de acesso eficazes. Além disso, as Chaves de API devem ser usadas para fornecer contexto para decisões de autorização personalizadas que reflitam a rota do plugin, em vez do usuário interativo padrão.
6. Exigir autorização e confirmação manual do usuário para qualquer ação realizada por plugins sensíveis.
7. Os plugins são, tipicamente, APIs REST, então os desenvolvedores devem aplicar as recomendações encontradas nas Principais 10 Vulnerabilidades de Segurança em APIs da OWASP - 2023 para minimizar vulnerabilidades genéricas.

Cenários de Ataque Exemplo

1. Um plugin aceita uma URL base e instrui o LLM a combinar a URL com uma consulta para obter previsões do tempo, que são incluídas no tratamento da solicitação do usuário.

Um usuário mal-intencionado pode criar uma solicitação para que a URL aponte para um domínio que eles controlam, permitindo que eles injetem seu próprio conteúdo no sistema LLM por meio de seu domínio.
2. Um plugin aceita uma entrada de formulário livre em um único campo que não valida. Um atacante fornece payloads cuidadosamente elaborados para realizar reconhecimento a partir de mensagens de erro. Em seguida, ele explora vulnerabilidades conhecidas de terceiros para executar código e realizar exfiltração de dados ou escalonamento de privilégios.
3. Um plugin usado para recuperar embeddings de um repositório de vetores aceita parâmetros de configuração como uma string de conexão sem validação. Isso permite que um atacante experimente e acesse outros repositórios de vetores alterando nomes ou parâmetros de host e exfiltra embeddings aos quais não deveria ter acesso.
4. Um plugin aceita cláusulas WHERE SQL como filtros avançados, que são então anexadas ao SQL de filtragem. Isso permite que um atacante execute um ataque SQL.
5. Um atacante usa injeção de prompt indireto para explorar um plugin de gerenciamento de código inseguro, sem validação de entrada e controle de acesso fraco, para transferir a propriedade do repositório e bloquear o usuário de seus repositórios.

Links de Referência

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. ChatGPT Plugin Exploit Explained: From Prompt Injection to Accessing Private Data: Embrace The Red
8. OWASP ASVS - 5 Validation, Sanitization and Encoding: OWASP AASVS
9. OWASP ASVS 4.1 General Access Control Design: OWASP AASVS
10. OWASP Top 10 API Security Risks – 2023: OWASP

LLM08: Autoridade Excessiva

Descrição

Um sistema baseado em LLM frequentemente é concedido um grau de agência pelo seu desenvolvedor - a capacidade de interagir com outros sistemas e realizar ações em resposta a um prompt. A decisão sobre quais funções invocar também pode ser delegada a um ' agente' LLM para determinar dinamicamente com base no prompt de entrada ou saída do LLM.

A Autoridade Excessiva é a vulnerabilidade que permite a realização de ações prejudiciais em resposta a saídas inesperadas/ambíguas de um LLM (independentemente do que está causando o mau funcionamento do LLM; seja alucinação/confabulação, injeção de prompt direta/indireta, plugin malicioso, prompts benignos mal projetados ou apenas um modelo com desempenho ruim). A causa raiz da Autoridade Excessiva geralmente é um ou mais dos seguintes: funcionalidade excessiva, permissões excessivas ou autonomia excessiva. Isso difere do tratamento inadequado de saída, que se preocupa com a falta de escrutínio nas saídas do LLM.

A Autoridade Excessiva pode levar a uma ampla variedade de impactos em relação à confidencialidade, integridade e disponibilidade, e depende dos sistemas com os quais um aplicativo baseado em LLM pode interagir.

Exemplos Comuns desta Vulnerabilidade

1. Funcionalidade Excessiva: Um agente LLM tem acesso a plugins que incluem funções desnecessárias para a operação pretendida do sistema. Por exemplo, um desenvolvedor precisa conceder a um agente LLM a capacidade de ler documentos de um repositório, mas o plugin de terceiros que eles escolhem também inclui a capacidade de modificar e excluir documentos.
2. Funcionalidade Excessiva: Um plugin pode ter sido testado durante a fase de desenvolvimento e descartado em favor de uma alternativa melhor, mas o plugin original permanece disponível

para o agente LLM.
3. Funcionalidade Excessiva: Um plugin LLM com funcionalidade em aberto não filtra corretamente as instruções de entrada para comandos fora do necessário para a operação pretendida do aplicativo. Por exemplo, um plugin para executar um comando shell específico falha em prevenir adequadamente a execução de outros comandos shell.
4. Permissões Excessivas: Um plugin LLM tem permissões em outros sistemas que não são necessárias para a operação pretendida do aplicativo. Por exemplo, um plugin destinado a ler dados se conecta a um servidor de banco de dados usando uma identidade que não apenas possui permissões SELECT, mas também permissões UPDATE, INSERT e DELETE.
5. Permissões Excessivas: Um plugin LLM projetado para realizar operações em nome de um usuário acessa sistemas downstream com uma identidade genérica de alta privilégio. Por exemplo, um plugin para ler a loja de documentos do usuário atual se conecta ao repositório de documentos com uma conta privilegiada que tem acesso aos arquivos de todos os usuários.
6. Autonomia Excessiva: Um aplicativo ou plugin baseado em LLM falha em verificar e aprovar independentemente ações de alto impacto. Por exemplo, um plugin que permite a exclusão de documentos de um usuário realiza exclusões sem qualquer confirmação do usuário.

Estratégias de Prevenção e Mitigação

As seguintes ações podem prevenir a Autoridade Excessiva:

1. Limitar os plugins/ferramentas que os agentes LLM têm permissão para chamar apenas às funções mínimas necessárias. Por exemplo, se um sistema baseado em LLM não requer a capacidade de buscar o conteúdo de uma URL, tal plugin não deve ser oferecido ao agente LLM.
2. Limitar as funções implementadas nos plugins/ferramentas LLM para o mínimo necessário. Por exemplo, um plugin que acessa a caixa de correio de um usuário para resumir e-mails pode precisar apenas da capacidade de ler e-mails, então o plugin não deve conter outras funcionalidades, como excluir ou enviar mensagens.
3. Evitar funções em aberto sempre que possível (por exemplo, executar um comando shell, buscar uma URL, etc.) e usar plugins/ferramentas com funcionalidades mais granulares.

Por exemplo, um aplicativo baseado em LLM pode precisar escrever alguma saída em um arquivo. Se isso fosse implementado usando um plugin para executar uma função shell, o escopo para ações indesejadas seria muito grande (qualquer outro comando shell poderia ser executado). Uma alternativa mais segura seria construir um plugin de gravação de arquivos que só pudesse suportar aquela funcionalidade específica.
4. Limitar as permissões concedidas a outros sistemas pelos plugins/ferramentas LLM para o mínimo necessário, a fim de limitar o escopo de ações indesejadas. Por exemplo, um agente LLM que usa um banco de dados de produtos para fazer recomendações de compra a um cliente pode precisar apenas de acesso de leitura a uma tabela de 'produtos'; não deve ter acesso a outras tabelas, nem a capacidade de inserir, atualizar ou excluir registros. Isso deve ser aplicado por meio da concessão de permissões apropriadas no banco de dados para a identidade que o plugin LLM usa para se conectar ao banco de dados.
5. Rastrear a autorização do usuário e o escopo de segurança para garantir que as ações realizadas em nome de um usuário sejam executadas em sistemas downstream no contexto desse usuário específico e com as permissões mínimas necessárias. Por exemplo, um plugin LLM que lê o repositório de código de um usuário deve exigir que o usuário se autentique via OAuth e com o escopo mínimo necessário.
6. Utilizar controle humano no loop para exigir que um humano aprove todas as ações antes que sejam realizadas. Isso pode ser implementado em um sistema downstream (fora do escopo do aplicativo LLM) ou dentro do próprio plugin/ferramenta LLM. Por exemplo, um aplicativo baseado em LLM que cria e publica conteúdo em mídias sociais em nome de um usuário deve incluir um procedimento de aprovação do usuário dentro do plugin/ferramenta/API que implementa a operação 'publicar'.
7. Implementar autorização em sistemas downstream em vez de depender de um LLM para decidir se uma ação é permitida ou não. Ao implementar ferramentas/plugins, aplicar o princípio de mediação completa para que todas as solicitações feitas a sistemas downstream por meio dos plugins/ferramentas sejam validadas em relação às políticas de segurança.

As seguintes opções não impedirão a Autoridade Excessiva, mas podem limitar o nível de dano causado:

1. Registrar e monitorar a atividade de plugins/ferramentas LLM e sistemas downstream para identificar onde ações indesejadas estão ocorrendo e responder adequadamente.

2. Implementar limitação de taxa para reduzir o número de ações indesejadas que podem ocorrer dentro de um determinado período de tempo, aumentando a oportunidade de descobrir ações indesejadas por meio de monitoramento antes que ocorra um dano significativo.

Exemplos de Cenários de Ataque

Um aplicativo de assistente pessoal baseado em LLM recebe acesso à caixa de correio de um indivíduo por meio de um plugin para resumir o conteúdo dos e-mails recebidos. Para alcançar essa funcionalidade, o plugin de e-mail requer a capacidade de ler mensagens, no entanto, o plugin escolhido pelo desenvolvedor do sistema também inclui funções para enviar mensagens. O LLM é vulnerável a um ataque de injeção de prompt indireto, em que um e-mail maliciosamente elaborado engana o LLM para comandar o plugin de e-mail a chamar a função 'enviar mensagens' para enviar spam da caixa de correio do usuário. Isso poderia ser evitado por:
(a) eliminar funcionalidades excessivas, usando um plugin que oferece apenas capacidades de leitura de e-mails,
(b) eliminar permissões excessivas, autenticando-se no serviço de e-mail do usuário por meio de uma sessão OAuth com escopo somente leitura, e/ou
(c) eliminar autonomia excessiva, exigindo que o usuário revise manualmente e clique em 'enviar' em cada e-mail redigido pelo plugin LLM.
Alternativamente, o dano causado poderia ser reduzido implementando limitação de taxa na interface de envio de e-mails.

Links de Referência

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: Dependência Excessiva

Descrição

A Sobrecarga pode ocorrer quando um LLM produz informações errôneas e as fornece de maneira autoritária. Embora os LLMs possam produzir conteúdo criativo e informativo, também podem gerar conteúdo factualmente incorreto, inapropriado ou inseguro. Isso é chamado de alucinação ou confabulação. Quando pessoas ou sistemas confiam nessas informações sem supervisão ou confirmação, pode resultar em violação de segurança, desinformação, má comunicação, questões legais e danos à reputação.

O código-fonte gerado pelo LLM pode introduzir vulnerabilidades de segurança não percebidas. Isso representa um risco significativo para a segurança operacional de aplicativos. Esses riscos destacam a importância de processos rigorosos de revisão, com:

- Supervisão
- Mecanismos contínuos de validação
- Avisos sobre riscos

Exemplos Comuns desta Vulnerabilidade

1. O LLM fornece informações imprecisas como resposta, declarando-as de maneira a sugerir alta autoridade. O sistema como um todo é projetado sem verificações e equilíbrios adequados para lidar com isso, e as informações enganam o usuário de uma maneira que leva a danos.
2. O LLM sugere código inseguro ou com falhas, levando a vulnerabilidades quando incorporado a um sistema de software sem a devida supervisão ou verificação.

Estratégias de Prevenção e Mitigação

1. Monitore e revise regularmente as saídas do LLM. Use técnicas de auto consistência ou votação para filtrar texto inconsistente. Comparar múltiplas respostas do modelo para

um único prompt pode ajudar a julgar melhor a qualidade e consistência da saída.
2. Verifique a saída do LLM com fontes externas confiáveis. Essa camada adicional de validação pode ajudar a garantir que as informações fornecidas pelo modelo sejam precisas e confiáveis.
3. Aprimore o modelo com ajuste fino ou embeddings para melhorar a qualidade da saída. Modelos pré-treinados genéricos têm mais chances de produzir informações imprecisas em comparação com modelos ajustados em um domínio específico. Técnicas como engenharia de prompts, ajuste eficiente de parâmetros (PET), ajuste total do modelo e prompts encadeados de pensamento podem ser empregadas para esse fim.
4. Implemente mecanismos automáticos de validação que possam verificar a saída gerada em relação a fatos ou dados conhecidos. Isso pode fornecer uma camada adicional de segurança e mitigar os riscos associados a alucinações.
5. Divida tarefas complexas em subtarefas gerenciáveis e atribua-as a diferentes agentes. Isso não apenas ajuda no gerenciamento da complexidade, mas também reduz as chances de alucinações, pois cada agente pode ser responsabilizado por uma tarefa menor.
6. Comunique claramente os riscos e limitações associados ao uso de LLMs. Isso inclui a possibilidade de imprecisões nas informações e outros riscos. Uma comunicação eficaz de riscos pode preparar os usuários para problemas potenciais e ajudá-los a tomar decisões informadas.
7. Construa APIs e interfaces de usuário que incentivem o uso responsável e seguro de LLMs. Isso pode envolver medidas como filtros de conteúdo, avisos ao usuário sobre imprecisões potenciais e rotulagem clara de conteúdo gerado por IA.
8. Ao usar LLMs em ambientes de desenvolvimento, estabeleça práticas e diretrizes seguras de codificação para evitar a integração de possíveis vulnerabilidades.

Exemplos de Cenários de Ataque

1. Uma organização de notícias utiliza intensamente um LLM para gerar artigos. Um ator malicioso explora essa sobrecarga, alimentando informações enganosas ao LLM e causando a propagação de desinformação.
2. A IA inadvertidamente plágiariza um conteúdo, levando a problemas de direitos autorais e diminuindo a confiança na organização.
3. Uma equipe de desenvolvimento de software utiliza um sistema LLM para acelerar o processo de codificação. A sobrecarga nas sugestões do LLM introduz vulnerabilidades

de segurança na aplicação devido a configurações padrão inseguras ou recomendações inconsistentes com práticas seguras de codificação.

4. Uma empresa de desenvolvimento de software usa um LLM para auxiliar os desenvolvedores. O LLM sugere uma biblioteca ou pacote de código inexistente, e um desenvolvedor, confiando na IA, integra inadvertidamente um pacote malicioso no software da empresa. Isso destaca a importância de verificar as sugestões do LLM, especialmente ao envolver código ou bibliotecas de terceiros.

Links de Referência

1. Understanding LLM Hallucinations: Towards Data Science

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

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

4. AI Hallucinations: Package Risk: Vulcan.io

5. How to Reduce the Hallucinations from Large Language Models: The New Stack

6. Practical Steps to Reduce Hallucination: Victor Debia

LLM10: Model Theft

Descrição

Esta entrada refere-se ao acesso não autorizado e à exfiltração de modelos LLM por atores maliciosos ou APTs. Isso ocorre quando modelos LLM proprietários (sendo propriedade intelectual valiosa) são comprometidos, fisicamente roubados, copiados ou quando pesos e parâmetros são extraídos para criar um equivalente funcional. O impacto do roubo de modelos LLM pode incluir perda econômica e de reputação da marca, erosão da vantagem competitiva, uso não autorizado do modelo ou acesso não autorizado a informações sensíveis contidas no modelo.

O roubo de LLMs representa uma preocupação significativa de segurança à medida que os modelos de linguagem se tornam cada vez mais poderosos e prevalentes. Organizações e pesquisadores devem priorizar medidas de segurança robustas para proteger seus modelos LLM, garantindo a confidencialidade e integridade de sua propriedade intelectual. A adoção de um framework de segurança abrangente que inclua controles de acesso, criptografia e monitoramento contínuo é crucial para mitigar os riscos associados ao roubo de modelos LLM e proteger os interesses de indivíduos e organizações que dependem de LLMs.

Exemplos Comuns desta Vulnerabilidade

1. Um atacante explora uma vulnerabilidade na infraestrutura de uma empresa para obter acesso não autorizado ao repositório de modelos LLM por meio de configurações inadequadas na rede ou nas configurações de segurança da aplicação.
2. Um cenário de ameaça interna em que um funcionário descontente vaza modelos ou artefatos relacionados.
3. Um atacante consulta a API do modelo usando inputs cuidadosamente elaborados e técnicas de injeção de prompt para coletar um número suficiente de saídas para criar um modelo sombra.
4. Um atacante malicioso consegue contornar técnicas de filtragem de entrada do LLM para realizar um ataque de canal lateral e, finalmente, extrair pesos do modelo e informações

de arquitetura para um recurso controlado remotamente.
5. O vetor de ataque para extração de modelo envolve consultar o LLM com um grande número de prompts sobre um tópico específico. As saídas do LLM podem ser usadas para ajustar outro modelo. No entanto, alguns pontos importantes sobre esse ataque são:
• O atacante deve gerar um grande número de prompts direcionados. Se os prompts não forem específicos o suficiente, as saídas do LLM serão inúteis.
• As saídas dos LLMs podem às vezes conter respostas alucinadas, o que significa que o atacante pode não ser capaz de extrair o modelo inteiro, já que algumas saídas podem ser sem sentido.
• Não é possível replicar um LLM 100% por meio da extração de modelo. No entanto, o atacante será capaz de replicar parcialmente um modelo.
6. O vetor de ataque para _replicação funcional do modelo_ envolve o uso do modelo-alvo por meio de prompts para gerar dados sintéticos de treinamento (uma abordagem chamada "autoinstrução") para então usá-lo e ajustar outro modelo fundamental para produzir um equivalente funcional. Isso contorna as limitações da extração baseada em consultas tradicional usada no Exemplo 5 e foi usado com sucesso em pesquisas sobre o uso de um LLM para treinar outro LLM. Embora, no contexto dessa pesquisa, a replicação do modelo não seja um ataque. A abordagem poderia ser usada por um atacante para replicar um modelo proprietário com uma API pública.

O uso de um modelo roubado, como modelo sombra, pode ser usado para realizar ataques adversários, incluindo acesso não autorizado a informações sensíveis contidas no modelo ou experimentar com inputs adversários para realizar injeções avançadas de prompt.

Estratégias de Prevenção e Mitigação

1. Implementar controles de acesso robustos (por exemplo, RBAC e o princípio do menor privilégio) e mecanismos de autenticação sólidos para limitar o acesso não autorizado a repositórios de modelos LLM e ambientes de treinamento.
1. Isso é especialmente válido para os três primeiros exemplos comuns, que podem causar essa vulnerabilidade devido a ameaças internas, configuração inadequada e/ou controles de segurança fracos sobre a infraestrutura que abriga modelos LLM, pesos e arquitetura nos quais um ator malicioso poderia infiltrar-se de dentro ou fora do ambiente.
2. O rastreamento, verificação e vulnerabilidades de dependência de gerenciamento

de fornecedores são tópicos importantes para prevenir exploits de ataques à cadeia de suprimentos.
2. Restringir o acesso do LLM a recursos de rede, serviços internos e APIs.
   1. Isso é especialmente válido para todos os exemplos comuns, pois cobre riscos e ameaças internas, mas também controla o que a aplicação LLM "_tem acesso_" e, assim, pode ser um mecanismo ou etapa de prevenção para evitar ataques de canal lateral.
3. Usar um Inventário ou Registro Centralizado de Modelos de ML para modelos usados em produção. Ter um registro centralizado de modelos impede o acesso não autorizado a modelos de ML por meio de controles de acesso, autenticação e capacidade de monitoramento/registro, que são bases sólidas para a governança. Ter um repositório centralizado também é benéfico para coletar dados sobre algoritmos usados pelos modelos para fins de conformidade, avaliações de risco e mitigação de riscos.
4. Monitorar regularmente e auditar logs de acesso e atividades relacionadas a repositórios de modelos LLM para detectar e responder prontamente a qualquer comportamento suspeito ou não autorizado.
5. Automatizar implementações MLOps com fluxos de trabalho de governança, rastreamento e aprovação para reforçar controles de acesso e implementação dentro da infraestrutura.
6. Implementar controles e estratégias de mitigação para reduzir o risco de técnicas de injeção de prompt causarem ataques de canal lateral.
7. Limitar a taxa de chamadas de API quando aplicável e/ou implementar filtros para reduzir o risco de exfiltração de dados das aplicações LLM, ou implementar técnicas para detectar (por

exemplo, DLP) atividade de extração de outros sistemas de monitoramento.
8. Implementar treinamento de robustez adversária para ajudar a detectar consultas de extração e reforçar medidas de segurança física.
9. Implementar um framework de marca d'água nas etapas de incorporação e detecção do ciclo de vida de um LLM.

Exemplos de Cenários de Ataque

1. Um atacante explora uma vulnerabilidade na infraestrutura de uma empresa para obter acesso não autorizado ao repositório de modelos LLM. O atacante procede a exfiltrar modelos LLM valiosos e os usa para lançar um serviço de processamento de linguagem

concorrente ou extrair informações sensíveis, causando danos financeiros significativos à empresa original.

2. Um funcionário descontente vaza modelos ou artefatos relacionados. A exposição pública desse cenário aumenta o conhecimento dos atacantes para ataques adversários de caixa cinza ou, alternativamente, rouba diretamente a propriedade disponível.

3. Um atacante consulta a API com inputs cuidadosamente selecionados e coleta um número suficiente de saídas para criar um modelo sombra.

4. Uma falha de controle de segurança está presente na cadeia de suprimentos e leva a vazamentos de dados de informações proprietárias do modelo.

5. Um atacante malicioso contorna as técnicas de filtragem de entrada e preâmbulos do LLM para realizar um ataque de canal lateral e recuperar informações do modelo para um recurso controlado remotamente sob seu controle.

Links de Referência

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

Team

Thank you to the OWASP Top 10 for LLM Applications version 1.1 Hindi Translation Contributors.

Version 1.1 Hindi Translation Contributors

Mary Smith

John Doe

Ted Johnson