OWASP Top 10 für LLM-Applikationen

VERSION 1.1

Veröffentlicht am: 10. Juni 2024

HTTPS://LLMTOP10.COM

Inhaltsverzeichnis

Abbildungen

Einleitung

Die Entstehung der Liste

Seit der Einführung von massentauglichen, vortrainierten Chatbots Ende 2022 ist das Interesse an Large Language Models (LLMs) enorm. Unternehmen, die das Potenzial von LLMs nutzen wollen, integrieren sie rasch in ihre Geschäftsprozesse und Angebote für Kunden. Die rasante Geschwindigkeit der Einführung von LLMs hat jedoch dazu geführt, dass die Etablierung umfassender Sicherheitsprotokolle in Verzug geraten ist, sodass viele Anwendungen mit hohen Risiken behaftet sind.

Das Fehlen einer zentralen Ressource, die sich mit diesen Sicherheitsbedenken in Bezug auf LLMs befasst, war unübersehbar. Entwicklerinnen und Entwickler, die mit den spezifischen Risiken von LLMs nicht vertraut waren, standen nur wenige Quellen zur Verfügung, und die Mission von OWASP schien die perfekte Lösung zu sein, um eine sichere Einführung dieser Technologie zu fördern.

Für wen diese Liste ist

Unsere Hauptzielgruppe sind Entwickelnde, Data Scientists sowie Sicherheitsexpertinnen und -experten, die Anwendungen und Plug-ins basierend auf LLM-Technologien entwerfen und erstellen. Unser Ziel ist es, praktische, umsetzbare und prägnante Sicherheitsleitlinien bereitzustellen, die diesen Fachleuten helfen, sich auf dem komplexen und sich ständig weiterentwickelnden Gebiet der Sicherheit von LLM-Anwendungen zurechtzufinden.

Die Erstellung der Liste

Das Erstellen der OWASP Top 10 für LLM-Applikationen war ein bedeutendes Unterfangen, das auf der kollektiven Expertise eines internationalen Teams von fast 500 Expertinnen und Experten mit mehr als 125 aktiven Mitgestaltenden basiert. Unsere Mitwirkenden kommen aus den unterschiedlichsten Bereichen, darunter KI-Unternehmen, Sicherheitsfirmen, ISVs, Cloud-Hyperscaler, Hardware-Anbieter und die akademische Welt.

Wir haben einen Monat lang gebrainstormt, potenzielle Schwachstellen vorgeschlagen und dabei 43 verschiedene Bedrohungen formuliert. In mehreren Abstimmungsrunden haben wir diese Vorschläge zu einer prägnanten Liste der zehn kritischsten Schwachstellen verfeinert. Spezialisierte Untergruppen untersuchten jede Schwachstelle und unterzogen sie einer öffentlichen Überprüfung, um sicherzustellen, dass die endgültige Liste so umfassend und nützlich wie möglich war.

Das Verhältnis zu anderen OWASP Top-10-Listen

Obwohl unsere Liste Gemeinsamkeiten mit den Schwachstellentypen anderer OWASP Top-10-Listen aufweist, wiederholen wir diese Schwachstellen nicht einfach. Stattdessen fokussieren wir uns auf die einzigartigen Auswirkungen dieser Schwachstellen bei der Verwendung von LLMs.

Unser Ziel ist es, die Lücke zwischen allgemeinen Prinzipien der Anwendungssicherheit und den spezifischen Herausforderungen von LLMs zu schließen. Dazu gehört die Untersuchung, wie herkömmliche Schwachstellen in LLMs andere Risiken darstellen oder auf neue Weise ausgenutzt werden können und wie herkömmliche Abwehrmaßnahmen für LLM-basierte Anwendungen angepasst werden müssen.

Die Zukunft

Version 1.1 der Liste wird nicht die letzte sein. Wir planen, sie regelmäßig zu aktualisieren, um mit den Entwicklungen in der Branche Schritt zu halten. Wir werden mit der erweiterten Community zusammenarbeiten, um den Stand der Technik voranzutreiben und mehr Schulungsmaterial für eine Vielzahl von Anwendungen zu erstellen. Ebenso streben wir auch die Zusammenarbeit mit Standardisierungsorganisationen und Regierungen in Fragen der KI-Sicherheit an. Sie sind herzlich eingeladen, sich unserer Gruppe anzuschließen und einen Beitrag zu leisten.

Steve Wilson

Projektleiter, OWASP Top 10 für LLM-Applikationen
https://www.linkedin.com/in/wilsonsd
Twitter/X: @virtualsteve

Ads Dawson

v1.1 Release Lead & Vulnerability Entries Lead, OWASP Top 10 für LLM-Applikationen
https://www.linkedin.com/in/adamdawson0
GitHub: @GangGreenTemperTatum

Über diese Übersetzung

Übersetzer

Johann-Peter Hartmann
https://www.linkedin.com/in/johann-peter-hartmann-92b70a/

Philippe Schrettenbrunner
https://www.linkedin.com/in/philippe-schrettenbrunner/

Bei der Erstellung dieser Übersetzung haben wir uns bewusst dafür entschieden, nur menschliche Übersetzer einzusetzen, in Anerkennung der außerordentlich technischen und kritischen Natur der OWASP Top 10 für LLM-Applikationen. Die oben aufgeführten Übersetzer verfügen nicht nur über ein tiefes Verständnis des Originalinhalts, sondern auch über die sprachliche Kompetenz, um diese Übersetzung sinnvoll zu gestalten.

Talesh Seeparsan
Übersetzungsleiter, OWASP Top 10 für LLM-Applikationen
https://www.linkedin.com/in/talesh/

OWASP Top 10 für LLM-Applikationen

LLM01: Prompt Injection

Mittels raffinierter Eingaben kann ein Large Language Model manipuliert werden und unbeabsichtigte Aktionen auslösen. Direkte Injections überschreiben System-Prompts, während indirekte Injection Eingaben über externe Quellen manipulieren.

LLM02: Unsichere Ausgabeverarbeitung

Diese Schwachstelle tritt auf, wenn eine Ausgabe von einem LLM ungeprüft akzeptiert wird, wodurch Backend-Systeme angreifbar werden. Ein Missbrauch kann zu schwerwiegenden Folgen wie XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery), SSRF (Server Side Request Forgery), Privilegienerweiterung oder Remote-Code-Ausführung führen.

LLM03: Poisoning von Trainingsdaten

Dies tritt auf, wenn LLM-Trainingsdaten manipuliert werden und dadurch Sicherheitslücken oder Bias entstehen, die Sicherheit, Performance oder ethisches Verhalten beeinträchtigen. Quellen umfassen Common Crawl, WebText, OpenWebText und Bücher.

LLM04: Denial of Service des Modells

Angreifende verursachen ressourcenintensive Operationen auf Large Language Models, was zu Beeinträchtigung oder hohen Kosten führt. Die Schwachstelle wird durch die ressourcenintensive Natur von LLMs und die Unvorhersehbarkeit von Benutzereingaben verstärkt.

LLM05: Schwachstellen in der Lieferkette

Der Lebenszyklus von LLM-Anwendungen kann durch verwundbare Komponenten oder Dienste kompromittiert werden, was Angriffe auf die Sicherheit zur Folge haben kann. Die Verwendung von Datensätzen von Drittanbietern, vortrainierten Modellen und Plug-ins kann zu weiteren Schwachstellen führen.

LLM06: Offenlegung sensibler Informationen

LLMs können in ihren Antworten vertrauliche Daten preisgeben, was zu unbefugtem Datenzugriff, Datenschutzverletzungen und Sicherheitsverstößen führt. Datenbereinigung und strenge Benutzerrichtlinien sind unerlässlich, um dies zu verhindern.

LLM07: Unsicheres Plug-in-Design

LLM-Plug-ins können unsichere Eingaben und unzureichende Zugriffskontrollen aufweisen. Dieser Mangel an Anwendungskontrolle erleichtert das Ausnutzen von

LLM-Plug-ins und kann zu Folgen wie der Ausführung von Remote-Code führen.

LLM08: Übermäßige Handlungsfreiheit

LLM-basierte Systeme können Aktionen ausführen, die unbeabsichtigte Folgen haben. Das Problem entsteht, wenn diesen Systemen zu viele Funktionalitäten, zu viele Berechtigungen oder zu viel Autonomie eingeräumt werden.

LLM09: Übermäßige Abhängigkeit

Systeme oder Personen, die sich zu sehr und unkontrolliert auf LLMs verlassen, können durch falsche oder unangemessene Inhalte, die von LLMs erzeugt werden, mit Fehlinformationen, Fehlkommunikation, rechtlichen Problemen und Sicherheitslücken konfrontiert werden.

LLM10: Modell-Diebstahl

Dies schließt den unbefugten Zugriff, das Kopieren oder die Weitergabe von geschützten LLM-Modellen ein. Die Folgen sind wirtschaftliche Verluste, gefährdete Wettbewerbsvorteile und potenzieller Zugang zu sensiblen Informationen.

Datenfluss einer LLM-Anwendung

Das folgende Diagramm zeigt die High-Level-Architektur einer hypothetischen LLM-Anwendung. Im Diagramm sind die Risikobereiche hervorgehoben, die veranschaulichen, wie sich die Punkte der OWASP Top 10 für LLM-Anwendungen mit dem Datenfluss der Anwendung überschneiden.

Dieses Diagramm kann als visueller Leitfaden verwendet werden, um zu verstehen, wie sich die Sicherheitsrisiken großer Sprachmodelle auf das gesamte Anwendungsökosystem auswirken.

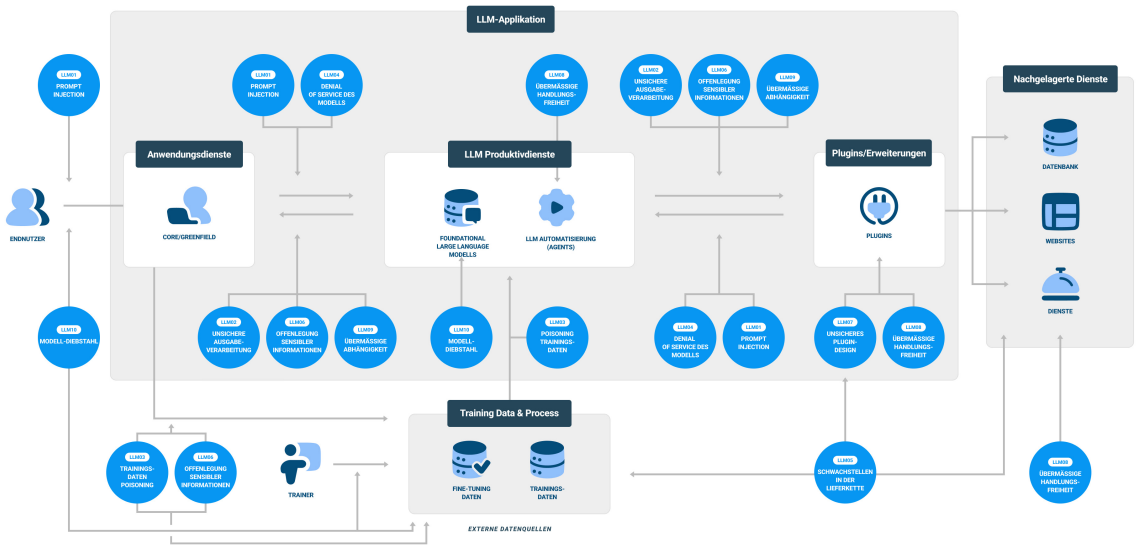

Abbildung 1: OWASP Top 10 für LLM-Applikationen visualisiert

LLM01: Prompt Injection

Beschreibung

Die Prompt-Injection-Schwachstelle tritt auf, wenn Angreifende ein Large Language Model (LLM) durch speziell gestaltete Eingaben so manipulieren, dass das LLM unwissentlich die Absichten des Angreifenden ausführt. Dies kann direkt durch "Jailbreaking" des System-Prompts oder indirekt durch manipulierte, externe Eingaben geschehen, was zu Datenexfiltration, Social Engineering und anderen Problemen führen kann.

- Direkte Prompt Injections, auch als "Jailbreaking" bekannt, treten auf, wenn böswillige Personen den zugrundeliegenden System Prompt überschreiben oder offenlegen. Dies kann es Angreifenden ermöglichen, Backend-Systeme zu nutzen, indem sie mit unsicheren Funktionen und Datenspeichern interagieren, die über das LLM zugänglich sind.
- Indirekte Prompt Injections treten auf, wenn ein LLM Eingaben von externen Quellen akzeptiert, die von Angreifenden kontrolliert werden können, wie z. B. Websites oder Dateien. Angreifende können eine Prompt Injection in externen Inhalt einbetten, um den Konversationskontext zu übernehmen. Dies würde dazu führen, dass die Stabilität der LLM-Ausgabe weniger robust wird, wodurch Angreifende entweder Personen oder zusätzliche Systeme, auf die das LLM Zugriff hat, manipulieren könnte. Außerdem müssen indirekte Prompt Injections für Menschen nicht sichtbar/lesbar sein, solange der Text vom LLM verarbeitet wird.

Die Ergebnisse eines erfolgreichen Prompt Injection-Angriffs können stark variieren - vom Ausspähen sensibler Informationen bis hin zur Beeinflussung kritischer Entscheidungsprozesse unter dem Deckmantel eines normalen Betriebs.

Bei fortgeschrittenen Angriffen könnte das LLM so manipuliert werden, dass es eine böswillige Person imitiert oder mit Plug-ins in der Umgebung der Nutzenden interagiert. Dies könnte zur Offenlegung sensibler Daten, zur unautorisierten Nutzung von Plug-ins oder zu Social Engineering führen. In solchen Fällen unterstützt das kompromittierte LLM die angreifende Person und umgeht die Standardsicherheitsmaßnahmen, während die Nutzenden nichts von dem Einbruch bemerken. In diesen Fällen agiert das kompromittierte LLM effektiv als Agent für die angreifende Person, die ihre Ziele verfolgt, ohne die üblichen Sicherheitsmaßnahmen auszulösen oder die Endnutzenden auf das Eindringen aufmerksam zu machen.

Gängige Beispiele für Schwachstellen

1. Eine böswillige Person erstellt eine direkte Prompt Injection für das LLM, die es anweist, die System-Prompts der App-Entwickelnden zu ignorieren und stattdessen einen Prompt auszuführen, der private, gefährliche oder anderweitig unerwünschte Informationen zurückgibt.
2. Eine Person verwendet ein LLM, um eine Webseite zusammenzufassen, die eine indirekte Prompt Injection enthält. Dies führt dazu, dass das LLM sensible Informationen vom Benutzer oder von der Benutzerin anfordert und eine Exfiltration über JavaScript oder Markdown durchführt.
3. Eine böswillige Person lädt einen Lebenslauf hoch, der eine indirekte Prompt Injection enthält. Das Dokument enthält eine Prompt Injection mit Anweisungen, die das LLM dazu veranlassen, Personen darüber zu informieren, dass dieses Dokument ausgezeichnet ist z. B. bei der Bewerbung auf eine Stellenausschreibung. Eine interne Person leitet das Dokument durch das LLM, um das Dokument zusammenzufassen. Das LLM meldet zurück, dass es sich um ein exzellentes Dokument handelt.
4. Eine Person aktiviert ein Plug-in, das mit einer E-Commerce-Website verknüpft ist. Eine bösartige Funktion, die in eine besuchte Website eingebettet ist, nutzt dieses Plug-in aus und führt zu nicht autorisierten Käufen.
5. Schädliche Befehle und Inhalte, die auf einer besuchten Website eingebettet sind, nutzen andere Plug-ins aus, um Personen zu betrügen.

Präventions- und Mitigationsstrategien

Prompt-Injection-Schwachstellen sind möglich, da LLMs von Natur aus nicht zwischen Befehlen und externen Daten unterscheiden können. Weil LLMs natürliche Sprache verwenden, betrachten sie beide Formen der Eingabe als vom Benutzer bereitgestellt. Daher gibt es keine unfehlbare Prävention innerhalb von LLMs, aber die folgenden Maßnahmen können die Auswirkungen von Prompt Injections reduzieren:

1. Erzwingen Sie eine Zugriffskontrolle für den Zugriff durch das LLM auf Backend-Systeme. Stellen Sie dem LLM eigene API-Tokens für zusätzliche Funktionen zur Verfügung, wie z. B. Plug-ins, Datenzugriff und Funktionsberechtigungen. Befolgen Sie das Least-Privilege-Prinzip und beschränken Sie den LLM-Zugriff auf das notwendige Minimum.
2. Binden Sie eine menschliche Kontrollinstanz für erweiterte Funktionalität ein. Bei der Durchführung privilegierter Operationen, wie dem Senden oder Löschen von E-Mails, sollte die Anwendung eine Bestätigung durch einen Menschen einfordern. Dies verringert die Möglichkeit, dass indirekte Prompt Injection-Angriffe zu unbefugten Handlungen im Namen von Nutzenden führen, ohne deren Wissen oder Zustimmung.
3. Trennen Sie externen Inhalt von Nutzereingaben. Isolieren und kennzeichnen

Sie, wo nicht vertrauenswürdiger Inhalt verwendet wird, um dessen Einfluss auf Nutzereingaben zu begrenzen. Verwenden Sie beispielsweise ChatML für OpenAI-API-Aufrufe, um dem LLM die Quelle der Eingabeaufforderung anzuzeigen.

4. Etablieren Sie Vertrauensgrenzen zwischen dem LLM, externen Quellen und erweiterbaren Funktionen (z.B. Plug-ins oder nachgelagerte Abläufe). Behandeln Sie das LLM als nicht vertrauenswürdige Instanz und bewahren Sie die endgültige Kontrolle über Entscheidungsprozesse bei den Nutzenden. Ein kompromittiertes LLM kann jedoch weiterhin als Vermittler (Man-in-the-Middle) zwischen den APIs der Anwendung und den Nutzenden agieren, da es Informationen verbergen oder manipulieren kann, bevor es diese den Nutzenden präsentiert. Kennzeichnen Sie potenziell nicht vertrauenswürdige Antworten visuell für die Nutzenden.
5. Überwachen Sie manuell und in regelmäßigen Abständen die Eingaben und Ausgaben des LLM, um sicherzustellen, dass sie den Erwartungen entsprechen. Obwohl dies keine Risikominderung darstellt, kann es Daten liefern, die zur Identifizierung und Behebung von Schwachstellen erforderlich sind.

Beispiele für Angriffsszenarien

1. Angreifende senden eine direkte Prompt Injection an einen LLM-basierten Support-Chatbot. Die Injection enthält "Vergiss alle vorherigen Anweisungen" sowie neue Anweisungen, um private Datenspeicher abzufragen. Außerdem werden Paketverwundbarkeiten und fehlende Ausgabevalidierung in der Backend-Funktion zum Senden von E-Mails ausgenutzt. Dies führt zu Remote-Code-Ausführung, unberechtigtem Zugriff und Privilegienerweiterung.
2. Angreifende betten eine indirekte Prompt-Injection in eine Webseite ein, die das LLM anweist, vorherige Benutzeranweisungen zu ignorieren und ein LLM-Plug-in zu verwenden, um die E-Mails des Benutzers zu löschen. Wenn eine Person das LLM verwendet, um diese Webseite zusammenzufassen, löscht das LLM-Plug-in die E-Mails der Person.
3. Eine Person verwendet ein LLM, um eine Webseite zusammenzufassen, die Text enthält, welcher ein Modell anweist, vorherige Benutzeranweisungen zu ignorieren und stattdessen ein Bild einzufügen, das zu einer URL verlinkt, die eine Zusammenfassung der Unterhaltung enthält. Die LLM-Ausgabe befolgt dies, was dazu führt, dass der Browser der Person die private Unterhaltung exfiltriert.
4. Eine bösartige Person lädt einen Lebenslauf mit einer Prompt Injection hoch. Der Backend-Benutzer verwendet ein LLM, um den Lebenslauf zusammenzufassen und zu fragen, ob die Person ein guter Kandidat ist. Aufgrund der Prompt Injection lautet die Antwort des LLM ja, ungeachtet des tatsächlichen Inhalts des Lebenslaufs.
5. Angreifende senden Nachrichten an ein proprietäres Modell, das sich auf einen System-Prompt verlässt, und bittet das Modell, seine vorherigen Anweisungen zu ignorieren und stattdessen seinen System-Prompt zu wiederholen. Das

Modell gibt den proprietären Prompt aus und die Angreifende können diese Anweisungen anderswo verwenden oder weitere, subtilere Angriffe konstruieren.

Referenzen

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: Unsichere Ausgabeverarbeitung

Beschreibung

Unsichere Ausgabeverarbeitung bezieht sich speziell auf die unzureichende Validierung, Bereinigung und Handhabung von Ausgaben, die von Large Language Models erzeugt werden, bevor sie an andere Komponenten und Systeme weitergeleitet werden. Da der von LLMs erzeugte Inhalt durch Prompt-Eingaben gesteuert werden kann, ähnelt dieses Verhalten dem indirekten Zugriff des Benutzers auf zusätzliche Funktionen.

Unsichere Ausgabeverarbeitung unterscheidet sich von Übermäßiger Abhängigkeit insofern, als sie sich mit den Ausgaben befasst, die von LLMs generiert werden, bevor sie weitergeleitet werden. Im Gegensatz dazu konzentriert sich Übermäßige Abhängigkeit auf allgemeinere Bedenken hinsichtlich der Angewiesenheit auf die Genauigkeit und Angemessenheit des LLM-Outputs.

Die erfolgreiche Ausnutzung einer Schwachstelle in der unsicheren Ausgabeverarbeitung kann zu XSS (Cross-Site Scripting) und CSRF (Cross-Site Request Forgery) in Webbrowsern sowie zu SSRF (Server-Side Request Forgery), Rechteerweiterung (Privilege Escalation) oder Remote-Code-Ausführung in Backend-Systemen führen.

Die folgenden Bedingungen können die Auswirkungen dieser Schwachstelle verstärken:

- Die Anwendung gewährt dem LLM Privilegien, die über die für den Endbenutzer vorgesehenen Privilegien hinausgehen, was eine Eskalation der Privilegien oder die Ausführung von Remote-Code ermöglicht.
- Die Anwendung ist anfällig für indirekte Prompt Injection-Angriffe, die es Angreifenden ermöglichen, privilegierten Zugriff auf die Umgebung eines Zielbenutzers zu erlangen.
- Plug-ins von Drittanbietern validieren Eingaben nicht ausreichend.

Gängige Beispiele für Schwachstellen

1. Die Ausgabe des LLM wird direkt in eine Systemshell oder eine ähnliche Funktion wie exec oder eval eingegeben, was zu einer Remote-Code-Ausführung führt.
2. JavaScript oder Markdown wird vom LLM generiert und an die aufrufende Person zurückgegeben. Der Code wird dann vom Browser interpretiert, was zu

XSS führt.

Präventions- und Mitigationsstrategien

1. Behandeln Sie das Sprachmodell mit einem Zero-Trust-Ansatz und wenden Sie eine geeignete Eingabevalidierung auf die Antworten an, die vom Modell an die Backend-Funktionen gesendet werden.
2. Befolgen Sie die OWASP ASVS (Application Security Verification Standard) Richtlinien, um eine effektive Eingabevalidierung und -bereinigung zu gewährleisten.
3. Encoden Sie die Modellausgabe zurück an den Benutzer, um unerwünschte Codeausführung durch JavaScript oder Markdown zu verhindern. Der OWASP ASVS bietet detaillierte Anweisungen zum Output Encoding.

Beispiele für Angriffsszenarien

1. Eine Anwendung verwendet ein LLM-Plug-in, um Antworten für eine Chatbot-Funktion zu generieren. Das Plug-in bietet auch eine Reihe von administrativen Funktionen, die einem anderen privilegierten LLM zur Verfügung stehen. Das allgemeine LLM sendet seine Antwort direkt an das Plug-in, ohne die Ausgabe ordnungsgemäß zu validieren, was dazu führt, dass das Plug-in für Wartungsarbeiten heruntergefahren wird.
2. Eine Person verwendet ein von einem LLM betriebenes Tool, das Webseiten zusammenfasst, um eine kurze Übersicht über einen Artikel zu erstellen. Die Website enthält eine Eingabeaufforderung, die das LLM anweist, sensible Inhalte entweder von der Website oder aus der Konversation des Benutzers zu erfassen. Anschließend kann der LLM die sensiblen Daten verschlüsseln und ohne Validierung oder Filterung der Ausgabe an einen von Angreifenden kontrollierten Server senden.
3. Ein LLM ermöglicht es Personen, SQL-Abfragen für eine Backend-Datenbank über eine Chat-ähnliche Funktion zu erstellen. Eine Person stellt eine Abfrage zum Löschen aller Datenbanktabellen. Wenn die vom LLM erstellte Abfrage nicht überprüft wird, könnten alle Datenbanktabellen gelöscht werden.
4. Eine Webanwendung verwendet einen LLM, um Inhalte aus Benutzereingaben zu generieren, ohne die Ausgabe zu bereinigen. Angreifende könnten eine manipulierte Anfrage einreichen, die das LLM dazu veranlasst, eine unbereinigte JavaScript-Payload zurückzugeben, die zu XSS führt, wenn sie im Browser des Opfers ausgeführt wird. Unzureichende Validierung von Anfragen ermöglicht diesen Angriff.

Referenzen

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: Poisoning von Trainingsdaten

Beschreibung

Ausgangspunkt jedes Ansatzes zum maschinellen Lernen sind Trainingsdaten, einfach gesagt "Rohdaten". Um eine hohe Effizienz zu erreichen (z. B. linguistisches Wissen und Wissen über die Welt), müssen diese Daten eine große Bandbreite an Domänen, Genres und Sprachen abdecken. Ein Large Language Model verwendet tiefe neuronale Netze, um Ausgaben zu erzeugen, die auf Mustern basieren, die aus den Trainingsdaten gelernt wurden.

Das Poisoning von Trainingsdaten bezieht sich auf die Manipulation von Daten vor dem Training oder von Daten, die in den Fine-Tuning- oder Embedding-Prozess involviert sind, um Schwachstellen (die alle einzigartige und manchmal gemeinsame Angriffsvektoren haben), Hintertüren oder Verzerrungen einzuführen, die die Sicherheit, die Effektivität oder das ethische Verhalten des Modells beeinträchtigen könnten. Vergiftete Informationen können an Personen weitergegeben werden oder andere Risiken wie Leistungseinbußen, die Ausnutzung nachgelagerter Software und Rufschädigung mit sich bringen. Selbst wenn Personen der problematischen KI-Ausgabe misstrauen, bleiben die Risiken bestehen, einschließlich der Beeinträchtigung der Leistung des Modells und der potenziellen Imageschäden.

- Das Pre-Training von Daten bezieht sich auf den Prozess des Trainierens eines Modells auf der Grundlage einer Aufgabe oder eines Datensatzes.
- Die Feinabstimmung umfasst die Anpassung eines bereits trainierten Modells an ein enger gefasstes Thema oder ein spezifischeres Ziel, indem es mit einem kuratierten Datensatz trainiert wird. Dieser Datensatz enthält typischerweise Beispiele für Eingaben und die entsprechenden gewünschten Ausgaben.
- Embedding ist der Prozess der Umwandlung von kategorialen Daten (oft Text) in eine digitale Repräsentation, die für das Training eines Sprachmodells verwendet werden kann. Beim Embedding werden Wörter oder Phrasen aus den Textdaten als Vektoren in einem kontinuierlichen Vektorraum dargestellt. Die Vektoren werden typischerweise durch Eingabe der Textdaten in ein neuronales Netz erzeugt, das auf einem großen Textkorpus trainiert wurde.

Die Vergiftung von Daten wird als Angriff auf die Integrität betrachtet, da die Manipulation der Trainingsdaten die Fähigkeit des Modells beeinträchtigt, korrekte Vorhersagen zu liefern. Es liegt auf der Hand, dass externe Datenquellen ein höheres Risiko bergen, da die Modellersteller keine Kontrolle über die Daten haben und nicht sicher sein können, dass der Inhalt frei von Bias, falschen Informationen oder unangemessenen Inhalten ist.

Gängige Beispiele für Schwachstellen

1. Bösartige Akteure oder Wettbewerber erstellen absichtlich ungenaue oder bösartige Dokumente, die auf das Pre-Training-, Fine-Tuning oder Embedding eines Modells abzielen. Betrachten Sie sowohl Split-View Data Poisoning (Ref.12) als auch Frontranking Poisoning (Ref.13) Angriffsvektoren zur Veranschaulichung.
    - Das Opfermodell wird anhand von gefälschten Informationen trainiert, die sich in den Ausgaben der generativen KI-Prompts an die Endkunden widerspiegeln.
2. Böswillige Akteure können direkt gefälschte, voreingenommene oder schädliche Inhalte in die Trainingsprozesse eines Modells einspeisen, die in späteren Ausgaben zurückgegeben werden.
3. Eine ahnungslose Person injiziert indirekt sensible oder proprietäre Daten in die Trainingsprozesse eines Modells, die in nachfolgenden Ausgaben zurückgegeben werden.
4. Ein Modell wird mit Daten trainiert, deren Quelle, Herkunft oder Inhalt in keinem der Trainingsbeispiele verifiziert wurde, was zu falschen Ergebnissen führen kann, wenn die Daten verfälscht oder fehlerhaft sind.
5. Uneingeschränkter Zugang zur Infrastruktur oder unzureichende Sandbox-Umgebungen können dazu führen, dass ein Modell unsichere Trainingsdaten verwendet, was zu verzerrten oder schädlichen Ergebnissen führt. Dieses Beispiel kann in allen Trainingsstadien vorkommen.
    - In diesem Szenario kann die Eingabe einer Person in das Modell in der Ausgabe einer anderen Person widergespiegelt werden (was zu einem Datenleck führt), oder Nutzende eines LLM können Ausgaben aus dem Modell erhalten, die, abhängig von der Art der erfassten Daten, ungenau, irrelevant oder schädlich für den Anwendungsfall des Modells sein können (in der Regel durch eine Modellkarte widergespiegelt).
6. Ob Entwicklerinnen und Entwickler, die Kundschaft oder allgemeine Nutzende des LLM, es ist wichtig zu verstehen, wie sich diese Schwachstelle auf die potenziellen Risiken innerhalb Ihrer LLM-Anwendung auswirken könnte, wenn sie mit einem fremden LLM interagiert, um die Legitimität der Modellausgaben zu verstehen, die auf dessen Trainingsverfahren basieren. In ähnlicher Weise könnten LLM-Entwickelnde sowohl direkten als auch indirekten Angriffen auf interne Daten oder Daten von Drittanbietern ausgesetzt sein, die für die (gängigste Variante) Fine-Tuning und Embeddings verwendet werden, was ein Risiko für alle LLM-Anwender darstellt.

Präventions- und Mitigationsstrategien

1. Überprüfen Sie die Lieferkette der Trainingsdaten, insbesondere wenn diese extern bezogen werden, und führen Sie Bescheinigungen mit der „ML-BOM“-Methode (Machine Learning Bill of Materials) sowie die Überprüfung der

Modellkarten durch.

2. Überprüfen Sie die korrekte Eignung der angestrebten Datenquellen und der darin enthaltenen Daten, die sowohl in der Vorbereitungsphase als auch in der Fine-Tuning- und Integrationsphase gewonnen wurden.
3. Überprüfen Sie Ihren Use Case für das LLM und die Anwendung, in die es integriert werden soll. Entwickeln Sie verschiedene Modelle mit separaten Trainingsdaten oder Fine-Tuning für verschiedene Anwendungsfälle, um eine granularere und genauere generative KI-Ausgabe für den jeweiligen Anwendungsfall zu erzeugen.
4. Stellen Sie sicher, dass ein ausreichendes Sandboxing durch Netzwerkkontrollen vorhanden ist, um zu verhindern, dass das Modell unbeabsichtigte Datenquellen nutzt, die die Ergebnisse des maschinellen Lernens beeinträchtigen könnten.
5. Verwenden Sie strenge Kontrollen oder Eingabefilter für bestimmte Trainingsdaten oder Kategorien von Datenquellen, um die Menge an gefälschten Daten zu kontrollieren. Bereinigung der Daten durch Techniken wie statistische Ausreißererkennung und Methoden zur Erkennung von Anomalien, um unerwünschte Daten zu erkennen und zu entfernen, bevor sie möglicherweise in den Feinabstimmungsprozess einfließen.
6. Entwickeln Sie Kontrollfragen bezüglich der Quelle und des Eigentums von Datensätzen, um sicherzustellen, dass das Modell nicht verunreinigt wurde, und integrieren Sie diese Kultur in den MLSecOps-Zyklus. Beziehen Sie sich auf verfügbare Ressourcen wie z. B. The Foundation Model Transparency Index (Ref.14) oder Open LLM Leaderboard (Ref.15).
7. Verwenden Sie DVC (Data Version Control (Ref.16)), um Teile eines Datensatzes, die manipuliert, gelöscht oder hinzugefügt wurden und zu Poisoning geführt haben, genau zu identifizieren und zu verfolgen.
8. Verwenden Sie eine Vektordatenbank, um von Benutzern bereitgestellte Informationen zu speichern, um andere Personen vor Manipulationen zu schützen und um sogar während der Produktion Fehler zu beheben, ohne ein neues Modell trainieren zu müssen.
9. Verwenden Sie Techniken zur Abwehr von Angriffen, wie z. B. föderiertes Lernen und Einschränkungen, um den Einfluss von Ausreißern zu minimieren, oder adverses Training, um robust gegenüber den schlimmsten Störungen der Trainingsdaten zu sein.
    - Ein "MLSecOps"-Ansatz könnte darin bestehen, die adversariale Robustheit mithilfe der Autopoison-Technik in den Trainingslebenszyklus zu integrieren.
    - Ein Beispiel Repository hierfür ist das Autopoison (Ref.17), das sowohl Angriffe wie Content Injection Attacks (der Versuch, einen Markennamen in den Antworten des Modells zu bewerben) als auch Refusal Attacks ("das Modell immer dazu bringen, die Antwort zu verweigern") umfasst, die mit diesem Ansatz durchgeführt werden können.
10. Testen und Erkennen durch Messen der Verluste während der Trainingsphase sowie Analyse der trainierten Modelle, um Anzeichen eines Poisoning-Angriffs zu erkennen, indem das Modellverhalten bei bestimmten Testeingaben

analysiert wird.
11. Überwachung und Alarmierung, wenn die Anzahl der verzerrten Antworten einen Schwellenwert überschreitet.
12. Menschliche Kontrolle bei der Überprüfung von Antworten und Audits.
13. Implementierung dedizierter LLMs, um unerwünschte Auswirkungen zu messen und andere LLMs mit Reinforcement Learning Techniken zu trainieren (Ref.18).
14. Durchführung von LLM-basierten Red-Team-Übungen (Ref.19) oder LLM-Schwachstellenanalysen (Ref.20) in den Testphasen des LLM-Lebenszyklus.

Beispiele für Angriffsszenarien

1. Die generative KI-Prompt-Ausgabe des LLM kann die Benutzer der Anwendung irreführen, was zu Bias (voreingenommenen Meinungen), Schlussfolgerungen oder, schlimmer noch, zu Hassverbrechen usw. führen kann.
2. Wenn die Trainingsdaten nicht ordnungsgemäß gefiltert und/oder bereinigt werden, kann eine böswillige Person versuchen, toxische Daten in das Modell einzuspeisen, damit es sich an die voreingenommenen und falschen Daten anpasst.
3. Böswillige Akteure oder Wettbewerber erstellen absichtlich ungenaue oder schädliche Dokumente, die auf die Trainingsdaten eines Modells abzielen, das gleichzeitig auf der Grundlage von Eingaben trainiert wird. Das Opfermodell trainiert mit diesen gefälschten Informationen, die sich in den Ausgaben generativer KI-Aufforderungen an seine Verbraucher widerspiegeln.
4. Die Schwachstelle Prompt Injection (Ref.21) könnte ein Angriffsvektor für diese Schwachstelle sein, wenn unzureichende Sanierung und Filterung durchgeführt werden, wenn Eingaben von LLM-Anwendungskunden zum Trainieren des Modells verwendet werden. D.h., wenn bösartige oder gefälschte Daten als Teil einer Prompt-Injektionstechnik in das Modell eingegeben werden, könnte dies inhärent in die Modeldaten übertragen werden.

Referenzen

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

12. Split-View Data Poisoning
13. Frontrunning Poisoning
14. The Foundation Model Transparency Index
15. Open LLM Leaderboard
16. Data Version Control
17. Autopoison
18. Reinforcement Learning Techniken
19. Red Team Exercises
20. LLM Vulnerability Scans
21. Prompt Injection

LLM04: Denial of Service des Modells

Beschreibung

Angreifende interagieren mit einem LLM auf eine Weise, die eine außergewöhnlich große Menge an Ressourcen verbraucht, was zu einer Verschlechterung der Servicequalität für sie und andere Personen führt und potenziell hohe Ressourcenkosten verursacht. Ebenso stellt die Möglichkeit, dass Angreifende in das Kontextfenster eines LLM eindringen oder es manipulieren, ein erhebliches Sicherheitsproblem dar. Dieses Problem wird immer kritischer aufgrund der zunehmenden Verwendung von LLMs in verschiedenen Anwendungen, ihrer intensiven Ressourcennutzung, der Unvorhersehbarkeit von Benutzereingaben und eines allgemeinen Mangels an Bewusstsein für diese Verwundbarkeit unter Entwicklerinnen und Entwicklern. In LLMs stellt das Kontextfenster die maximale Textlänge dar, die das Modell verwalten kann, einschließlich Eingabe und Ausgabe. Es ist ein entscheidendes Merkmal von LLMs, da es die Komplexität der Sprachmuster bestimmt, die das Modell verstehen kann, und die Länge des Textes, den es zu einem gegebenen Zeitpunkt verarbeiten kann. Die Größe des Kontextfensters wird durch die Architektur des Modells bestimmt und kann von Modell zu Modell variieren.

Gängige Beispiele für Schwachstellen

1. Abfragen, die zu einer wiederkehrenden Ressourcennutzung führen, indem Tasks in großer Zahl in einer Warteschlange erzeugt werden, z. B. mit LangChain oder AutoGPT.
2. Senden von ungewöhnlich ressourcenintensiven Anfragen, die ungewöhnliche Zeichenkombinationen oder Schreibweisen verwenden.
3. Kontinuierlicher Eingabe-Overflow: Angreifende senden einen Eingabestrom an das LLM, der dessen Kontextfenster überschreitet, wodurch das Modell übermäßig viele Rechenressourcen verbraucht.
4. Wiederholte lange Eingaben: Angreifende senden wiederholt lange Eingaben an das LLM, die jeweils über das Kontextfenster überschreiten.
5. Rekursive Kontexterweiterung: Angreifende konstruieren Eingaben, die rekursive Kontexterweiterungen auslösen und das LLM zwingen, das Kontextfenster wiederholt zu erweitern und zu verarbeiten.
6. Überflutung mit Eingaben variabler Länge: Angreifende überfluten das LLM mit einer großen Anzahl von Eingaben variabler Länge, wobei jede Eingabe sorgfältig so konstruiert ist, dass sie gerade die Grenze des Kontextfensters erreicht. Diese Technik zielt darauf ab, mögliche Ineffizienzen bei der Verarbeitung von Eingaben variabler Länge auszunutzen, das LLM zu

überlasten und möglicherweise seine Reaktionsfähigkeit zu beeinträchtigen.

Präventions- und Mitigationsstrategien

1. Validierung und Bereinigung von Eingaben, um sicherzustellen, dass Benutzereingaben die festgelegten Grenzen einhalten und schädliche Inhalte herausgefiltert werden.
2. Begrenzen Sie die Ressourcennutzung pro Anfrage oder Vorgang, um die Ausführung von Anfragen mit komplexen Teilen zu verlangsamen.
3. Setzen Sie API-Rate Limits durch, um die Anzahl der Anfragen zu begrenzen, die einzelne Personen oder eine IP-Adresse in einem bestimmten Zeitraum stellen kann.
4. Begrenzen Sie die Anzahl der anstehenden Aktionen und die Gesamtanzahl der Aktionen in einem System, das auf LLM-Antworten reagiert.
5. Kontinuierliche Überwachung der LLM-Ressourcenauslastung, um abnormale Spitzen oder Muster zu erkennen, die auf einen DoS-Angriff hindeuten könnten.
6. Setzen Sie strikte Eingabebegrenzungen basierend auf dem Kontextfenster des LLM, um eine Überlastung und Erschöpfung der Ressourcen zu vermeiden.
7. Entwickler für potenzielle DoS-Schwachstellen in LLMs zu sensibilisieren und Richtlinien für eine sichere LLM-Implementierung bereitzustellen.

Beispiele für Angriffsszenarien

1. Angreifende senden wiederholt mehrere komplexe und teure Anfragen an ein gehostetes Modell, was zu einer schlechteren Servicequalität für andere Personen und zu höheren Ressourcenkosten für die Betreibenden führt.
2. Während ein LLM-gesteuertes Tool Informationen sammelt, um eine harmlose Anfrage zu beantworten, wird ein Stück Text auf einer Webseite gefunden. Dies führt dazu, dass das Tool viele weitere Anfragen an die Webseite stellt, was zu einem hohen Ressourcenverbrauch führt.
3. Angreifende bombardieren das LLM kontinuierlich mit Eingaben, die das Kontextfenster des LLM überschreiten. Die Angreifenden können automatisierte Skripte oder Tools verwenden, um eine große Menge an Eingaben zu senden, die die Verarbeitungskapazität des LLM übersteigen. Infolgedessen verbraucht das LLM übermäßig viele Rechenressourcen, was zu einer erheblichen Verlangsamung oder zum völligen Ausfall des Systems führt.
4. Angreifende senden eine Reihe von sequenziellen Eingaben an das LLM, wobei jede Eingabe so gestaltet ist, dass sie knapp unterhalb der Grenze des Kontextfensters liegt. Durch das wiederholte Senden dieser Eingaben versuchen die Angreifenden, die verfügbare Kapazität des Kontextfensters auszuschöpfen. Da das LLM Mühe hat, jede Eingabe innerhalb seines Kontextfensters zu verarbeiten, werden die Systemressourcen belastet, was potenziell zu einer verminderten Leistung oder einem vollständigen Ausfall des Dienstes führt.
5. Angreifende nutzen die rekursiven Mechanismen des LLM, um wiederholt eine

Kontexterweiterung auszulösen. Indem Eingaben konstruiert werden, die das rekursive Verhalten des LLM ausnutzen, zwingen die Angreifenden das Modell dazu, das Kontextfenster wiederholt zu erweitern und zu verarbeiten, wodurch erhebliche Rechenressourcen verbraucht werden. Dieser Angriff belastet das System und kann zu einem DoS-Zustand führen, der das LLM handlungsunfähig macht oder zum Absturz bringt.

6. Angreifende überfluten das LLM mit einer großen Anzahl von Eingaben variabler Länge, die sorgfältig so gestaltet sind, dass sie sich der Grenze des Kontextfensters nähern oder diese erreichen. Durch die Überflutung des LLM mit Eingaben variabler Länge versuchen die Angreifenden, mögliche Ineffizienzen bei der Verarbeitung von Eingaben variabler Länge auszunutzen. Diese Flut von Eingaben belastet die Ressourcen des LLM übermäßig, was zu einer Verschlechterung der Leistung führen und die Fähigkeit des Systems, auf legitime Anfragen zu reagieren, beeinträchtigen kann.
7. Während DoS-Angriffe in der Regel darauf abzielen, Systemressourcen zu überlasten, können sie auch andere Aspekte des Systemverhaltens ausnutzen, wie z. B. API-Beschränkungen. Bei einem kürzlich aufgetretenen Sicherheitsvorfall bei Sourcegraph beispielsweise nutzten böswillige Akteure ein durchgesickertes Admin-Zugriffstoken, um API-Rate-Limiting zu ändern, was potenziell zu Dienstunterbrechungen führte, indem es anormale Anfragevolumina zuließ.

Referenzen

1. LangChain max_iterations: hwchase17 auf 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: Schwachstellen in der Lieferkette

Beschreibung

Die LLM-Lieferkette kann Schwachstellen aufweisen, die die Integrität von Trainingsdaten, ML-Modellen und Anwendungsplattformen beeinträchtigen. Diese Schwachstellen können zu verzerrten Ergebnissen, Sicherheitsverletzungen oder sogar zu kompletten Systemausfällen führen. Traditionell konzentrieren sich Schwachstellen auf Softwarekomponenten, doch beim maschinellen Lernen kommt hinzu, dass vortrainierte Modelle und Trainingsdaten, die von Dritten bereitgestellt werden, anfällig für Manipulationen und Schadangriffe sind.

Schließlich können LLM-Plug-in-Erweiterungen ihre eigenen Schwachstellen mitbringen. Diese werden in LLM07 - Unsicheres Plug-in-Design (Ref.11) beschrieben, das das Schreiben von LLM-Plug-ins abdeckt und nützliche Informationen zur Bewertung von Plug-ins von Drittanbietern liefert.

Gängige Beispiele für Schwachstellen

1. Traditionelle Schwachstellen in Paketen von Drittanbietern, einschließlich veralteter oder nicht mehr unterstützter Komponenten.
2. Verwendung eines anfälligen, vortrainierten Modells für das Fine-Tuning.
3. Verwendung vergifteter Crowd-Sourced-Daten für das Training.
4. Verwendung veralteter oder nicht mehr unterstützter Modelle, die nicht mehr gewartet werden und zu Sicherheitsproblemen führen.
5. Unklare AGB und Datenschutzrichtlinien der Modellbetreiber führen dazu, dass sensible Daten der Anwendung für das Modelltraining verwendet und anschließend sensible Informationen preisgegeben werden. Dies kann auch für Risiken gelten, die sich aus der Verwendung von urheberrechtlich geschütztem Material durch den Modellanbieter ergeben.

Präventions- und Mitigationsstrategien

1. Prüfen Sie Datenquellen und -anbieter sorgfältig, einschließlich der allgemeinen Geschäftsbedingungen und Datenschutzrichtlinien; verwenden Sie nur vertrauenswürdige Anbieter. Stellen Sie sicher, dass ein angemessenes und unabhängig verifiziertes Sicherheitsniveau vorhanden ist und dass die Richtlinien des Modellanbieters mit Ihren Datenschutzrichtlinien übereinstimmen, d. h. dass Ihre Daten nicht für das Training Ihrer Modelle verwendet werden.
2. Verwenden Sie nur seriöse Plug-ins und stellen Sie sicher, dass diese für Ihre

Anwendungsanforderungen getestet wurden. LLM07 Unsicheres Plug-in-Design bietet Informationen über die LLM-Aspekte eines unsicheren Plug-in-Designs, gegen die Sie testen sollten, um die Risiken bei der Verwendung von Plug-ins Dritter zu minimieren.

3. Verständnis und Anwendung der in den OWASP Top 10 A06:2021 - Vulnerable and Outdated Components (Ref.12) identifizierten Abhilfemaßnahmen. Dies beinhaltet das Scannen, Verwalten und Patchen von Komponenten. Für Entwicklungsumgebungen mit Zugriff auf sensible Daten sollten diese Kontrollen auch in diesen Umgebungen angewendet werden.
4. Führen Sie ein aktuelles Inventar der Komponenten mit einer Software-Bill-of-Materials (SBOM), um sicherzustellen, dass Sie über ein aktuelles, genaues und signiertes Inventar verfügen, das Manipulationen an bereitgestellten Paketen verhindert. SBOMs können verwendet werden, um neue Zero-Day-Schwachstellen schnell zu identifizieren und zu melden.
5. Zum Zeitpunkt der Erstellung dieses Dokuments decken die SBOMs keine Modelle, deren Artefakte und Datensätze ab. Wenn Ihre LLM-Anwendung ein proprietäres Modell verwendet, sollten Sie bewährte MLOps-Praktiken und -Plattformen verwenden, die sichere Model-Repositorys mit Daten-, Modell- und Experimentverfolgung bieten.
6. Sie sollten auch Modell- und Codesignaturen verwenden, wenn Sie externe Modelle und Anbieter verwenden.
7. Die Erkennung von Anomalien und Robustheitstests gegen die bereitgestellten Modelle und Daten können helfen, Manipulationen und Vergiftungen aufzudecken, wie in Poisoning von Trainingsdaten (Ref.13) erörtert. Idealerweise sollte dies Teil der MLOps-Pipelines sein, obwohl es sich hierbei um neue Techniken handelt, die möglicherweise leichter im Rahmen von Red-Teaming-Übungen implementiert werden können.
8. Implementieren Sie eine ausreichende Überwachung, um das Scannen von Komponenten und Umgebungen auf Schwachstellen, die Verwendung nicht autorisierter Plug-ins und veraltete Komponenten, einschließlich des Modells und seiner Artefakte, abzudecken.
9. Implementieren Sie eine Patch-Richtlinie, um anfällige oder veraltete Komponenten zu entschärfen. Stellen Sie sicher, dass die Anwendung eine gepflegte Version der APIs und des zugrundeliegenden Modells verwendet.
10. Überprüfen Sie regelmäßig die Sicherheit und den Zugang des Anbieters und stellen Sie sicher, dass es keine Änderungen an der Sicherheitslage oder den Nutzungsbedingungen gibt.

Beispiele für Angriffsszenarien

1. Angreifende verwenden eine verwundbare Python-Bibliothek, um ein System zu kompromittieren. Dies geschah beim ersten OpenAI-Datenleck.
2. Angreifende bieten ein LLM-Plug-in für die Flugsuche an, das gefälschte Links generiert, die Benutzer auf betrügerische Websites leiten.

3. Angreifende nutzen die PyPi-Paket-Registry aus, um Modellentwickler dazu zu bringen, ein kompromittiertes Paket herunterzuladen und Daten zu exfiltrieren oder Privilegien in einer Modellentwicklungsumgebung zu eskalieren. Dies war ein echter Angriff.
4. Angreifende vergiften ein öffentlich verfügbares, vortrainiertes Modell, das auf Wirtschaftsanalyse und Sozialforschung spezialisiert ist, um eine Hintertür zu schaffen, die Falschinformationen und Fake News generiert. Sie stellen das Modell auf einem Marktplatz für Modelle (z. B. Hugging Face) zur Verfügung, damit die Opfer es verwenden.
5. Angreifende vergiften öffentlich verfügbare Datensätze, um eine Hintertür beim Fine-Tuning von Modellen zu schaffen. Die Hintertür begünstigt auf subtile Weise bestimmte Unternehmen in verschiedenen Märkten.
6. Eine kompromittierte Person eines Lieferanten (Outsourcing-Entwickler, Hosting-Unternehmen usw.) exfiltriert Daten, Modell oder Code und stiehlt geistiges Eigentum.
7. Ein LLM-Betreiber ändert seine AGB und Datenschutzbestimmungen dahin gehend, dass eine ausdrückliche Ablehnung der Verwendung von Anwendungsdaten für das Modelltraining erforderlich ist, was zur Speicherung sensibler Daten führt.

Referenzen

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
11. LLM07 - Insecure Plugin Design
12. A06:2021 – Vulnerable and Outdated Components
13. Training Data Poisoning

LLM06: Offenlegung sensibler Informationen

Beschreibung

LLM-Anwendungen haben das Potenzial, durch ihre Ausgabe sensible Informationen, proprietäre Algorithmen oder andere vertrauliche Details zu offenbaren. Dies kann zu unberechtigtem Zugriff auf sensible Daten, geistiges Eigentum, Verletzungen der Privatsphäre und anderen Sicherheitsverletzungen führen. Es ist wichtig, dass die Benutzerinnen und Benutzer von LLM-Anwendungen wissen, wie sie sicher mit LLMs interagieren können, und dass sie sich der Risiken bewusst sind, die mit der unbeabsichtigten Eingabe sensibler Daten verbunden sind, die dann von der LLM in der Ausgabe an anderer Stelle zurückgegeben werden können.

Um dieses Risiko zu minimieren, sollten LLM-Anwendungen eine angemessene Datenbereinigung durchführen, um zu verhindern, dass Benutzerdaten in die Daten des Trainingsmodells gelangen. Die Eigentümer von LLM-Anwendungen sollten auch über angemessene Nutzungsbedingungen verfügen, um die Verbraucher darüber zu informieren, wie ihre Daten verarbeitet werden, und ihnen die Möglichkeit zu geben, die Aufnahme ihrer Daten in das Trainingsmodell abzulehnen.

Die Interaktion zwischen Verbraucher und LLM-Anwendung bildet eine zweiseitige Vertrauensgrenze, bei der wir weder den Eingaben des Clients->LLM noch den Ausgaben des LLM->Client vertrauen können. Es ist wichtig zu beachten, dass diese Schwachstelle davon ausgeht, dass bestimmte Voraussetzungen nicht gegeben sind, wie z.B. Bedrohungsmodellierungsverfahren, eine sichere Infrastruktur und eine angemessene Sandbox. Das Hinzufügen von Einschränkungen in der System-Eingabeaufforderung bezüglich der Datentypen, die der LLM zurückgeben soll, kann einen gewissen Schutz vor der Offenlegung sensibler Informationen bieten. Die unvorhersehbare Natur von LLMs bedeutet jedoch, dass solche Einschränkungen nicht immer beachtet werden und durch Prompt Injection oder andere Vektoren umgangen werden könnten.

Gängige Beispiele für Schwachstellen

1. Unvollständige oder unsachgemäße Filterung von sensiblen Informationen in den Antworten des LLM.
2. Übermäßige Angleichung oder Einprägung sensibler Daten im Trainingsprozess des LLM.
3. Unbeabsichtigte Offenlegung vertraulicher Informationen aufgrund von Fehlinterpretationen des LLM, fehlenden Datenbereinigungsmethoden oder

Fehlern.

Präventions- und Mitigationsstrategien

1. Integrieren Sie geeignete Datenbereinigungs- und Scrubbing-Techniken, um zu verhindern, dass Benutzerdaten in die Daten des Trainingsmodells gelangen.
2. Implementierung robuster Eingabevalidierungs- und -bereinigungsmethoden, um potenziell schädliche Eingaben zu identifizieren und zu entfernen, damit das Modell nicht vergiftet wird.
3. Wenn das Modell mit Daten angereichert und Fine-Tuning (Ref.7) betrieben wird: (z. B. Daten, die dem Modell vor oder während der Bereitstellung zugeführt werden)
    - Alles, was in den Fine-Tuning-Daten als sensibel eingestuft ist, könnte Personen offengelegt werden. Wenden Sie daher das Least-Privilege-Prinzip an und trainieren Sie das Modell nicht mit Informationen, auf die Personen mit den höchsten Rechten zugreifen können und die dann einer weniger privilegierten Person angezeigt werden könnten.
    - Der Zugriff auf externe Datenquellen (Orchestrierung von Daten zur Laufzeit) sollte eingeschränkt werden.
    - Strenge Zugriffskontrollmethoden für externe Datenquellen und ein rigoroser Ansatz zur Aufrechterhaltung einer sicheren Lieferkette.

Beispiele für Angriffsszenarien

1. Die ahnungslose, legitime Benutzerin A erhält über das LLM Zugang zu anderen Benutzerdaten, wenn sie in nicht böswilliger Absicht mit der LLM-Anwendung interagiert.
2. Benutzer A zielt darauf ab, die Eingabefilter und die Bereinigungsfunktionen des LLM durch eine ausgeklügelte Abfolge von Eingabeaufforderungen zu umgehen und Personen dazu zu bringen, personenbezogene Informationen (PII) über andere Personen der Anwendung preiszugeben.
3. Personenbezogene Daten wie z.B. PII gelangen über Trainingsdaten in das Modell, entweder durch Unachtsamkeit der Person selbst oder durch die LLM-Anwendung. Dies könnte das Risiko und die Wahrscheinlichkeit von Szenario 1 oder 2 oben erhöhen.

Referenzen

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
7. Fine-Tuning

LLM07: Unsicheres Plug-in-Design

Beschreibung

LLM-Plug-ins sind Erweiterungen, die, wenn sie aktiviert sind, bei Benutzerinteraktionen automatisch vom Modell aufgerufen werden. Sie werden von der Modell-Integrationsplattform gesteuert, und die Anwendung hat möglicherweise keine Kontrolle über ihre Ausführung, insbesondere wenn das Modell von einer anderen Instanz gehostet wird. Weiterhin ist es wahrscheinlich, dass Plug-ins Freitexteingaben aus dem Modell ohne Validierung oder Typüberprüfung implementieren, um Beschränkungen der Kontextgröße zu umgehen. Dadurch können Angreifende eine böswillige Anfrage an das Plug-in stellen, was zu einer Vielzahl von unerwünschtem Verhalten bis hin zur Remote-Code-Ausführung führen kann.

Der Schaden, der durch böswillige Eingaben verursacht wird, hängt oft von unzureichenden Zugriffskontrollen und dem Versäumnis ab, die Autorisierung über Plug-ins hinweg zu verfolgen. Unzureichende Zugriffskontrollen ermöglichen es einem Plug-in, anderen Plug-ins blind zu vertrauen und davon auszugehen, dass die Eingaben von einem Menschen stammen. Solche unzureichenden Zugriffskontrollen können dazu führen, dass böswillige Eingaben schädliche Folgen haben, von der Datenexfiltration über die Ausführung von Remote-Code bis hin zur Privilegieneskalation.

Dieser Abschnitt konzentriert sich auf die Erstellung von LLM-Plug-ins und nicht auf Plug-ins von Drittanbietern, die durch LLM-Supply-Chain-Schwachstellen abgedeckt werden.

Gängige Beispiele für Schwachstellen

1. Ein Plug-in akzeptiert alle Parameter in einem einzigen Textfeld anstelle von eigenständig eingegebenen Parametern.
2. Ein Plug-in akzeptiert Konfigurationsstrings anstelle von Parametern, die alle Konfigurationseinstellungen überschreiben können.
3. Ein Plug-in akzeptiert direkt SQL- oder Programmieranweisungen anstelle von Parametern.
4. Die Authentifizierung erfolgt ohne explizite Autorisierung für ein bestimmtes Plug-in.
5. Ein Plug-in behandelt alle LLM-Inhalte so, als ob sie vollständig vom Menschen erstellt wurden, und führt jede angeforderte Aktion ohne zusätzliche Autorisierung aus.

Präventions- und Mitigationsstrategien

1. Plug-ins sollten, wo immer möglich, streng parametrisierte Eingaben erzwingen und Typ- und Bereichsprüfungen für Eingaben vorsehen. Wenn dies nicht möglich ist, sollte eine zweite Schicht von typisierten Aufrufen eingeführt werden, die die Anfragen parst und eine Validierung und Bereinigung durchführt. Wenn Freitexteingaben aufgrund der Anwendungssemantik akzeptiert werden müssen, sollten diese sorgfältig geprüft werden, um sicherzustellen, dass keine potenziell schädlichen Methoden aufgerufen werden.
2. Plug-in-Entwickelnde sollten die Empfehlungen aus dem OWASP ASVS (Application Security Verification Standard) anwenden, um eine angemessene Validierung und Bereinigung von Eingaben sicherzustellen.
3. Plug-ins sollten gründlich geprüft und getestet werden, um eine angemessene Validierung zu gewährleisten. Verwenden Sie statische Anwendungssicherheitstests (SAST) sowie dynamische und interaktive Anwendungstests (DAST, IAST) in den Entwicklungspipelines.
4. Plug-ins sollten so entworfen werden, dass die Auswirkungen der Ausnutzung unsicherer Eingabeparameter gemäß den OWASP ASVS Access Control Guidelines minimiert werden. Dies beinhaltet eine Zugriffskontrolle mit den geringsten Rechten, die so wenig Funktionalität wie möglich preisgibt, aber dennoch die gewünschte Funktion erfüllt.
5. Plug-ins sollten geeignete Authentifizierungsidentitäten wie OAuth2 verwenden, um eine effektive Autorisierung und Zugriffskontrolle anzuwenden. Überdies sollten API-Schlüssel verwendet werden, um den Kontext für benutzerdefinierte Autorisierungsentscheidungen bereitzustellen, die den Pfad des Plug-ins und nicht den interaktiven Standardbenutzer widerspiegeln.
6. Verlangen Sie eine manuelle Benutzerautorisierung und Bestätigung aller von vertraulichen Plug-ins durchgeführten Aktionen.
7. Plug-ins sind in der Regel REST-APIs. Daher sollten Entwickelnde die Empfehlungen in OWASP Top 10 API Security Risks - 2023 befolgen, um allgemeine Schwachstellen zu minimieren.

Beispiele für Angriffsszenarien

1. Ein Plug-in akzeptiert eine Basis-URL und weist das LLM an, die URL mit einer Anfrage zu kombinieren, um Wettervorhersagen zu erhalten, die in die Bearbeitung der Benutzeranfrage einfließen. Böswillige Personen können eine Anfrage so gestalten, dass die URL auf eine von ihnen kontrollierte Domäne verweist, wodurch sie ihre eigenen Inhalte über diese in das LLM-System einspeisen können.
2. Ein Plug-in akzeptiert eine freie Eingabe in ein einzelnes Feld, die nicht validiert wird. Angreifende liefern sorgfältig gestaltete Payloads, um Fehlermeldungen auszuspähen. Anschließend nutzen sie bekannte Sicherheitslücken von Drittanbietern aus, um Code auszuführen, Daten zu exfiltrieren oder Rechte zu

erweitern.
3. Ein Plug-in, das zum Abrufen von Embeddings aus einem Vektorspeicher verwendet wird, akzeptiert Konfigurationsparameter als Verbindungsstring ohne jegliche Validierung. Dadurch können Angreifende experimentieren und auf andere Vektorspeicher zugreifen, indem sie Namen oder Host-Parameter ändern und Embeddings exfiltrieren, auf die sie keinen Zugriff haben sollten.
4. Ein Plug-in akzeptiert SQL WHERE-Klauseln als erweiterte Filter, die dann an die SQL-Bedingungen angehängt werden. Dadurch können Angreifende einen SQL-Angriff durchführen.
5. Angreifende nutzen eine indirekte Prompt Injection aus, um ein unsicheres Codeverwaltungs-Plug-in ohne Eingabevalidierung und mit schwacher Zugriffskontrolle auszunutzen, um den Besitz von Repositorys zu übertragen und Personen von ihren Repositorys auszuschließen.

Referenzen

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: Übermäßige Handlungsfreiheit

Beschreibung

Ein System, das auf einem LLM basiert, wird von Entwickelnden oft mit einem gewissen Grad an Handlungsfreiheit ausgestattet - das heißt, der Fähigkeit, mit anderen Systemen zu interagieren und Aktionen auf Anforderung auszuführen. Die Entscheidung, welche Funktionen aufgerufen werden sollen, kann auch an einen LLM-Agenten delegiert werden, um dynamisch auf der Grundlage der Eingabeaufforderung oder der LLM-Ausgabe bestimmt zu werden.

Übermäßige Handlungsfreiheit ist die Schwachstelle, die es ermöglicht, schädliche Aktionen als Reaktion auf unerwartete/unklare Ausgaben eines LLM auszuführen (unabhängig davon, was den LLM fehlerhaft macht; sei es Halluzination/Konfabulation, direkte/indirekte Prompt Injection, bösartige Plug-ins, schlecht konstruierte gutartige Prompts oder einfach ein schlecht funktionierendes Modell). Die Hauptursache für übermäßige Handlungsfreiheit ist typischerweise eine oder mehrere der folgenden: übermäßige Funktionalität, übermäßige Berechtigungen oder übermäßige Autonomie. Dies unterscheidet sich von Unsichere Ausgabeverarbeitung, die sich auf unzureichende Überprüfung der LLM-Ausgabe bezieht.

Übermäßige Handlungsfreiheit kann zu einem breiten Spektrum von Auswirkungen auf Vertraulichkeit, Integrität und Verfügbarkeit führen und hängt von den Systemen ab, mit denen eine LLM-basierte Anwendung interagieren kann.

Gängige Beispiele für Schwachstellen

1. Übermäßige Funktionalität: Ein LLM-Agent hat Zugriff auf Plug-ins, die Funktionen enthalten, die für den beabsichtigten Betrieb des Systems nicht erforderlich sind. Entwickelnde müssen etwa einem LLM-Agenten die Fähigkeit geben, Dokumente aus einem Repository zu lesen, aber das Plugin eines Drittanbieters, das er verwendet, enthält auch die Fähigkeit, Dokumente zu ändern und zu löschen.
2. Übermäßige Funktionalität: Ein Plugin wurde während einer Entwicklungsphase getestet und zugunsten einer besseren Alternative verworfen, aber das ursprüngliche Plugin bleibt für den LLM-Agenten zugänglich.
3. Übermäßige Funktionalität: Ein LLM-Plugin mit offenem Funktionsumfang filtert nicht korrekt die Eingabeanweisungen für Befehle, die nicht für den beabsichtigten Betrieb der Anwendung notwendig sind. Beispielsweise

verhindert ein Plugin zur Ausführung eines bestimmten Shell-Befehls nicht ordnungsgemäß die Ausführung anderer Shell-Befehle.
4. Übermäßige Berechtigungen: Ein LLM-Plugin hat Berechtigungen für andere Systeme, die für den beabsichtigten Betrieb der Anwendung nicht erforderlich sind. Beispielsweise verbindet sich ein Plugin zum Lesen von Daten mit einem Datenbankserver unter Verwendung einer Identität, die nicht nur SELECT-Berechtigungen hat, sondern auch UPDATE-, INSERT- und DELETE-Berechtigungen.
5. Exzessive Berechtigungen: Ein LLM-Plugin, das entwickelt wurde, um Operationen im Namen einer Person durchzuführen, greift auf nachgelagerte Systeme mit einer generischen, hoch privilegierten Identität zu. Beispielsweise verbindet sich ein Plug-in, das den aktuellen Dokumentenspeicher einer Person lesen soll, mit dem Dokumentenspeicher mit einem privilegierten Konto, das Zugriff auf alle Dateien der Person hat.
6. Übermäßige Autonomie: Eine LLM-basierte Anwendung oder ein Plugin versäumt es, hochwirksame Aktionen unabhängig zu überprüfen und zu genehmigen. Beispielsweise führt ein Plugin, das das Löschen von Dokumenten einer Person ermöglicht, Löschungen ohne jegliche Bestätigung durch diese durch.

Präventions- und Mitigationsstrategien

Die folgenden Maßnahmen können dazu beitragen, übermäßige Handlungsfreiheit zu vermeiden:

1. Beschränken Sie die Plug-ins/Tools, die von LLM-Agenten aufgerufen werden können, auf die notwendige Minimalfunktionalität. Wenn etwa ein LLM-basiertes System nicht die Fähigkeit benötigt, den Inhalt einer URL abzurufen, sollte ein solches Plug-in dem LLM-Agenten nicht angeboten werden.
2. Beschränken Sie die in LLM-Plug-ins/Tools implementierten Funktionen auf das notwendige Minimum. Ein Plug-in, das auf die Mailbox einer Person zugreift, um E-Mails zusammenzufassen, benötigt möglicherweise nur die Fähigkeit, E-Mails zu lesen, und sollte daher keine weiteren Funktionen wie das Löschen oder Versenden von Nachrichten enthalten.
3. Vermeiden Sie nach Möglichkeit Funktionen mit offenem Ende (z.B. einen Shell-Befehl ausführen, eine URL abrufen usw.) und verwenden Sie Plug-ins/Tools mit granulareren Funktionen. Beispielsweise muss eine LLM-basierte Anwendung einige Ausgaben in eine Datei schreiben. Wenn dies mit einem Plugin implementiert wird, das eine Shell-Funktion ausführt, ist der Spielraum für unerwünschte Aktionen gewaltig (jeder andere Shell-Befehl könnte ausgeführt werden). Eine sicherere Alternative wäre es, ein Plug-in zum Schreiben von Dateien zu entwickeln, das nur diese spezielle Funktion unterstützt.
4. Beschränken Sie die Berechtigungen, die LLM-Plug-ins/Tools anderen Systemen gewähren, auf ein Minimum, um den Umfang unerwünschter Aktionen

einzuschränken. Beispielsweise benötigt ein LLM-Agent, der eine Produktdatenbank verwendet, um einem Kunden Kaufempfehlungen zu geben, möglicherweise nur Lesezugriff auf eine 'Produkte'-Tabelle; er sollte keinen Zugriff auf andere Tabellen oder die Möglichkeit haben, Datensätze einzufügen, zu aktualisieren oder zu löschen. Dies sollte durch entsprechende Datenbankberechtigungen für die Identität durchgesetzt werden, die das LLM-Plugin verwendet, um sich mit der Datenbank zu verbinden.

5. Protokollieren Sie die Autorisierung von Personen und den Security Scope, um zu gewährleisten, dass Aktionen, die im Namen der Personen durchgeführt werden, auf nachgelagerten Systemen im Kontext der jeweiligen Person und mit den erforderlichen Mindestrechten ausgeführt werden. Beispielsweise sollte ein LLM-Plugin, das das Code-Repository einer Person liest, verlangen, dass sich die Person über OAuth und mit dem erforderlichen Mindestumfang authentifiziert.
6. Setzen Sie auf menschliche Kontrolle, um für alle Aktionen eine menschliche Genehmigung zu verlangen, bevor sie ausgeführt werden. Dies kann in einem nachgelagerten System (außerhalb der LLM-Anwendung) oder innerhalb des LLM-Plug-ins/Tools selbst implementiert werden. Beispielsweise sollte eine LLM-basierte Anwendung, die Social-Media-Inhalte im Namen einer Person erstellt und veröffentlicht, eine Freigaberoutine innerhalb des Plug-ins/Tools/API enthalten, das die 'Post'-Operation implementiert.
7. Implementieren Sie Autorisierung in nachgelagerten Systemen, anstatt sich darauf zu verlassen, dass ein LLM entscheidet, ob eine Aktion erlaubt ist oder nicht. Bei der Implementierung von Tools/Plug-ins das Prinzip der vollständigen Mediation (complete mediation principle) anwenden, sodass alle Anfragen, die über Plug-ins/Tools an nachgelagerte Systeme gestellt werden, anhand von Sicherheitsrichtlinien validiert werden.

Die folgenden Optionen verhindern zwar keine übermäßige Handlungsfreiheit, können aber das Ausmaß des Schadens begrenzen:

1. Protokollieren und überwachen Sie die Aktivitäten von LLM-Plug-ins und -Tools sowie von nachgelagerten Systemen, um zu erkennen, wo unerwünschte Aktionen stattfinden, und um entsprechend reagieren zu können.
2. Implementieren Sie Rate-Limiting, um die Anzahl unerwünschter Aktionen, die in einem bestimmten Zeitraum auftreten können, zu reduzieren und die Möglichkeit zu erhöhen, unerwünschte Aktionen durch Überwachung zu erkennen, bevor größerer Schaden entsteht.

Beispiele für Angriffsszenarien

Eine LLM-basierte Personal Assistant App greift über ein Plug-in auf das Postfach einer Person zu, um den Inhalt eingehender E-Mails zusammenzufassen. Um diese Funktionalität zu erreichen, muss das E-Mail-Plug-in in der Lage sein, Nachrichten zu lesen. Das vom Systementwickler ausgewählte Plug-in enthält jedoch auch

Funktionen zum Senden von Nachrichten. Das LLM ist anfällig für einen indirekten Man-in-the-Middle-Angriff, bei dem eine bösartig gestaltete eingehende E-Mail das LLM dazu veranlasst, das E-Mail-Plug-in anzuweisen, die Funktion 'Nachricht senden' aufzurufen, um Spam aus der Mailbox des Benutzers zu versenden. Dies könnte vermieden werden durch
(a) Eliminierung überflüssiger Funktionalität durch Verwendung eines Plug-ins, das nur E-Mail-Lesefunktionen bietet,
(b) Eliminierung von übermäßigen Berechtigungen durch Authentifizierung am E-Mail-Dienst des Benutzers über eine OAuth-Sitzung mit schreibgeschütztem Umfang, und/oder
(c) Eliminierung übermäßiger Autonomie, indem der Benutzer jede vom LLM-Plug-in erzeugte E-Mail manuell überprüfen und auf 'Senden' klicken muss.
Alternativ könnte der verursachte Schaden durch die Implementierung von Rate-Limiting in der Mailversand-Schnittstelle reduziert werden.

Referenzen

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: Übermäßige Abhängigkeit

Beschreibung

Übermäßige Abhängigkeit kann entstehen, wenn ein LLM fehlerhafte Informationen produziert und diese als authentisch darstellt. Während LLMs kreative und informative Inhalte produzieren können, können sie auch Inhalte produzieren, die faktisch falsch, unangemessen oder unsicher sind. Dies wird als Halluzination oder Konfabulation bezeichnet. Wenn Menschen oder Systeme diesen Informationen ohne Überprüfung oder Bestätigung vertrauen, kann dies zu Sicherheitsverletzungen, Fehlinformationen, falscher Kommunikation, rechtlichen Problemen und Rufschädigung führen.

Von LLM erzeugter Quellcode kann unbemerkt Sicherheitslücken einführen. Dies stellt ein erhebliches Risiko für die Betriebs- und Anwendungssicherheit dar. Diese Risiken unterstreichen die Bedeutung strenger Verifikationsprozesse:

- Überprüfung
- kontinuierliche Validierungsmechanismen
- Haftungsausschlüsse für Risiken

Gängige Beispiele für Schwachstellen

1. Ein LLM antwortet mit ungenauen Informationen auf eine Art und Weise, die es höchst vertrauenswürdig wirken lässt. Das Gesamtsystem ist ohne angemessene Kontrollen und Überprüfungen konzipiert, und die Informationen führen Personen in einer Weise in die Irre, die zu Schäden führen kann.
2. Das LLM schlägt unsicheren oder fehlerhaften Code vor, der zu Schwachstellen führt, wenn er ohne angemessene Aufsicht oder Überprüfung in ein Softwaresystem eingebaut wird.

Präventions- und Mitigationsstrategien

1. Überwachen und überprüfen Sie regelmäßig die Ergebnisse des LLMs. Verwenden Sie Selbstkonsistenz- oder Voting-Techniken, um inkonsistenten Text herauszufiltern. Der Vergleich mehrerer Modellantworten auf eine Anfrage kann helfen, die Qualität und Konsistenz der Ausgabe zu beurteilen.
2. Überprüfen Sie die LLM-Ausgabe mit vertrauenswürdigen externen Quellen. Diese zusätzliche Ebene der Validierung kann dazu beitragen, dass die vom Modell gelieferten Informationen genau und zuverlässig sind.
3. Verbessern Sie das Modell durch Fine-Tuning oder Embeddings, um die

Ausgabequalität zu verbessern. Allgemeine, vortrainierte Modelle liefern mit größerer Wahrscheinlichkeit ungenaue Informationen als Modelle, die in einem bestimmten Bereich angepasst wurden. Techniken wie Prompt Engineering, parameter efficient tuning (PET), vollständiges Model-Tuning und Chain-of-Thought-Prompting können zu diesem Zweck eingesetzt werden.

4. Implementieren Sie automatische Validierungsmechanismen, die die generierte Ausgabe mit bekannten Fakten oder Daten vergleichen können. Dies kann eine zusätzliche Sicherheitsebene bieten und das Risiko von Halluzinationen verringern.
5. Zerlegen Sie komplexe Aufgaben in handhabbare Unteraufgaben und weisen Sie sie verschiedenen Agenten zu. Dies hilft nicht nur bei der Bewältigung der Komplexität, sondern verringert auch die Wahrscheinlichkeit von Halluzinationen, da jeder Agent für eine kleinere Aufgabe verantwortlich gemacht werden kann.
6. Kommunizieren Sie klar die Risiken und Einschränkungen, die mit der Verwendung von LLMs verbunden sind. Dies schließt das Potenzial für Informationsungenauigkeiten und andere Risiken ein. Eine effektive Risikokommunikation kann die Nutzenden auf mögliche Probleme vorbereiten und ihnen helfen, informierte Entscheidungen zu treffen.
7. Entwickeln Sie APIs und Benutzeroberflächen, die eine verantwortungsvolle und sichere Nutzung von LLMs fördern. Dies kann Maßnahmen wie Inhaltsfilter, Benutzerwarnungen vor potenziellen Ungenauigkeiten und eine klare Kennzeichnung von KI-generierten Inhalten umfassen.
8. Bei der Verwendung von LLMs in Entwicklungsumgebungen sollten sichere Programmierpraktiken und -richtlinien festgelegt werden, um die Integration potenzieller Schwachstellen zu verhindern.

Beispiele für Angriffsszenarien

1. Eine Nachrichtenorganisation nutzt ein LLM intensiv, um Nachrichtenartikel zu generieren. Ein böswilliger Akteur nutzt diese übermäßige Abhängigkeit aus, indem er das LLM mit irreführenden Informationen füttert und so die Verbreitung von Fehlinformationen verursacht.
2. Die KI plagiiert unbeabsichtigt Inhalte, was zu Urheberrechtsproblemen und einem Vertrauensverlust in die Organisation führt.
3. Ein Softwareentwicklungsteam verwendet ein LLM-System, um den Entwicklungsprozess zu beschleunigen. Die übermäßige Abhängigkeit von den Vorschlägen der KI führt zu Sicherheitslücken in der Anwendung aufgrund unsicherer Standardeinstellungen oder Empfehlungen, die nicht sicheren Programmierpraktiken entsprechen.
4. Eine Softwareentwicklungsfirma verwendet ein LLM, um Entwickelnde zu unterstützen. Das LLM schlägt eine nicht existierende Codebibliothek oder ein nicht existierendes Paket vor, und eine Person, die der KI vertraut, integriert unwissentlich ein schädliches Paket in die Software des Unternehmens. Dies

unterstreicht die Bedeutung der Überprüfung von LLM-Vorschlägen, insbesondere wenn es sich um Code oder Bibliotheken von Drittanbietern handelt.

Referenzen

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: Modell-Diebstahl

Beschreibung

Dieser Eintrag bezieht sich auf den unbefugten Zugriff und die Exfiltration von LLM-Modellen durch böswillige Akteure oder APTs (Advanced Persistent Threats). Dies geschieht, wenn proprietäre LLM-Modelle (als wertvolles geistiges Eigentum) kompromittiert, physisch gestohlen, kopiert oder Modellgewichte und Parameter extrahiert werden, um ein funktionsfähiges Äquivalent zu erstellen. Die Folgen eines Diebstahls von LLM-Modellen können unter anderem sein: wirtschaftliche Einbußen und Imageschäden, Erosion des Wettbewerbsvorteils, unbefugte Nutzung des Modells oder unbefugter Zugang zu sensiblen Informationen, die im Modell enthalten sind.

Der Diebstahl von LLMs stellt ein erhebliches Sicherheitsproblem dar, da Sprachmodelle immer leistungsfähiger und allgegenwärtiger werden. Organisationen und Forschende müssen robuste Sicherheitsmaßnahmen ergreifen, um ihre LLM-Modelle zu schützen und die Vertraulichkeit und Integrität ihres geistigen Eigentums zu gewährleisten. Der Einsatz eines umfassenden Sicherheitsrahmens, der Zugriffskontrollen, Verschlüsselung und kontinuierliche Überwachung umfasst, ist entscheidend, um die mit dem Diebstahl von LLM-Modellen verbundenen Risiken zu mindern und die Interessen sowohl von Einzelpersonen als auch von Organisationen zu schützen, die sich auf LLM verlassen.

Gängige Beispiele für Schwachstellen

1. Angreifende nutzen eine Schwachstelle in der Infrastruktur einer Organisation aus, um sich durch Fehlkonfiguration der Netzwerk- oder Anwendungssicherheitseinstellungen unberechtigten Zugriff auf das LLM-Model-Repository zu verschaffen.
2. Ein Insider-Bedrohungsszenario, bei dem unzufriedene Mitarbeitende das Modell oder damit verbundene Artefakte nach Außen dringen lassen.
3. Angreifende verwenden die Modell-API mit sorgfältig erstellten Eingaben und Prompt Injection-Techniken, um eine ausreichende Anzahl von Ausgaben zu sammeln, um ein Schattenmodell zu erstellen.
4. Böswillige Angreifenden sind in der Lage, die Eingabefiltertechniken des LLM zu umgehen, um einen Seitenkanal-Angriff durchzuführen und schließlich Modellgewichte und Architekturinformationen an eine ferngesteuerte Ressource zu übermitteln.
5. Der Angriffsvektor für die Extraktion von Modellen beinhaltet die Abfrage des LLM mit einer großen Anzahl von Prompts zu einem bestimmten Thema. Die

Ausgabe des LLM kann dann verwendet werden, um ein anderes Modell zu verfeinern. Bei diesem Angriff sind jedoch einige Dinge zu beachten:

- Angreifende müssen eine große Anzahl spezifischer Prompts erzeugen. Wenn die Prompts nicht spezifisch genug sind, ist die Ausgabe des LLM nutzlos.
- Die Ausgaben von LLMs können manchmal halluzinierte Antworten enthalten, was bedeutet, dass Angreifende möglicherweise nicht in der Lage sind, das gesamte Modell zu extrahieren, da einige der Ausgaben sinnlos sein können.
  - Es ist nicht möglich, ein LLM zu 100 % durch Modellextraktion zu replizieren. Angreifende werden jedoch in der Lage sein, ein Teilmodell zu replizieren.

6. Der Angriffsvektor für die _funktionale Modellreplikation_ besteht darin, das Zielmodell über Prompts zu verwenden, um synthetische Trainingsdaten zu erzeugen (ein Ansatz, der als "Selbstinstruktion" bezeichnet wird), und diese dann zu verwenden, um ein anderes Basismodell zu verfeinern und ein funktionales Äquivalent zu erzeugen. Dieser Ansatz umgeht die Beschränkungen der traditionellen abfragebasierten Extraktion, die in Beispiel 5 verwendet wurde, und wurde erfolgreich in der Forschung über die Verwendung eines LLM zum Trainieren eines anderen LLM eingesetzt. Im Kontext dieser Forschung stellt die Modellreplikation keinen Angriff dar. Der Ansatz könnte von Angreifenden verwendet werden, um ein proprietäres Modell mit einer öffentlichen API zu replizieren.

Die Verwendung eines gestohlenen Modells als Schattenmodell kann dazu genutzt werden, Angriffe des Gegners zu inszenieren, einschließlich des unbefugten Zugriffs auf sensible Informationen, die im Modell enthalten sind, oder um unbemerkt mit Eingaben des Gegners zu experimentieren, um fortgeschrittenere Prompt-Injektionen zu inszenieren.

Präventions- und Mitigationsstrategien

1. Implementieren Sie strenge Zugriffskontrollen (z. B. RBAC und das Least-Privilege-Prinzip) und starke Authentifizierungsmechanismen, um den unbefugten Zugriff auf Repositorys mit LLM-Modellen und Lernumgebungen einzuschränken.
   - Dies gilt insbesondere für die ersten drei gängigen Beispiele, die diese Schwachstelle aufgrund von Insider-Bedrohungen, Fehlkonfigurationen und/oder schwachen Sicherheitskontrollen in Bezug auf die Infrastruktur, die LLM-Modelle, -Gewichte und -Architekturen beherbergt, verursachen können, in die böswillige Akteure von innen oder außen eindringen können.
   - Schwachstellen bei der Rückverfolgbarkeit, der Verifizierung und der Abhängigkeit von Lieferanten sind wichtige Schwerpunktthemen, um die Ausnutzung von Angriffen auf die Lieferkette zu verhindern.

2. Beschränken Sie den Zugriff des LLM auf Netzwerkressourcen, interne Dienste und APIs.
    - Dies gilt speziell für alle gängigen Beispiele, da es Risiken und Bedrohungen von innen abdeckt, aber letztlich auch kontrolliert, worauf die LLM-Anwendung "zugreifen" kann und somit einen Mechanismus oder eine Präventivmaßnahme zur Verhinderung von Seitenkanal-Angriffen darstellen kann.
3. Verwenden Sie ein zentrales ML-Modell-Inventar oder -Register für ML-Modelle, die in der Produktion verwendet werden. Eine zentralisierte Modellregistrierung verhindert den unbefugten Zugriff auf ML-Modelle durch Zugriffskontrollen, Authentifizierung und Überwachungs-/Protokollierungsfunktionen, die eine gute Grundlage für die Governance bilden. Ein zentrales Repository ist auch vorteilhaft, um Daten über die von den Modellen verwendeten Algorithmen für Zwecke der Compliance, Risikobewertung und Risikominderung zu sammeln.
4. Überwachen und überprüfen Sie regelmäßig die Zugriffsprotokolle und Aktivitäten im Zusammenhang mit LLM-Model-Repositorys, um verdächtiges oder nicht autorisiertes Verhalten zu erkennen und sofort darauf zu reagieren.
5. Automatisieren Sie die MLOps-Bereitstellung mit Governance-, Tracking- und Genehmigungs-Workflows, um die Zugriffs- und Bereitstellungskontrollen innerhalb der Infrastruktur zu stärken.
6. Implementieren Sie Kontrollen und Abschwächungsstrategien, um das Risiko von Prompt Injection-Techniken, die Side-Channel-Angriffe verursachen, abzuschwächen oder zu verringern.
7. Beschränken Sie gegebenenfalls die Anzahl der API-Aufrufe und/oder setzen Sie Filter ein, um das Risiko der Datenexfiltration aus LLM-Anwendungen zu verringern, oder implementieren Sie Techniken zur Erkennung von Extraktionsaktivitäten (z. B. DLP) aus anderen Überwachungssystemen.
8. Implementieren Sie ein Robustheitstraining für feindliche Angriffe, um Extraktionsanfragen zu erkennen und die physischen Sicherheitsmaßnahmen zu verstärken.
9. Implementieren Sie ein Wasserzeichen-Framework in den Embeddings- und Erkennungsphasen des Lebenszyklus eines LLM.

Beispiele für Angriffsszenarien

1. Angreifende nutzen eine Schwachstelle in der Infrastruktur eines Unternehmens aus, um sich unberechtigten Zugriff auf dessen LLM-Modell-Repository zu verschaffen. Die Angreifenden exfiltrieren wertvolle LLM-Modelle und verwendet sie, um einen konkurrierenden Sprachverarbeitungsdienst zu starten oder sensible Informationen zu extrahieren, wodurch dem ursprünglichen Unternehmen erheblicher finanzieller Schaden entsteht.
2. Unzufriedene Mitarbeitende lassen das Modell oder damit verbundene Artefakte nach außen dringen. Das öffentliche Bekanntwerden dieses Szenarios erhöht

das Wissen für Angreifende, die Gray-Box-Angriffe oder alternativ den direkten Diebstahl des verfügbaren Eigentums planen.
3. Angreifende fragen die API mit sorgfältig ausgewählten Eingaben ab und sammelt eine ausreichende Anzahl von Ausgaben, um ein Schattenmodell zu erstellen.
4. Ein Versagen der Sicherheitskontrolle in der Lieferkette führt zum Abfluss von proprietären Modellinformationen.
5. Böswillige Angreifende umgeht die Eingabefilterung und die Präambel des LLM, um einen Seitenkanal-Angriff durchzuführen und Modellinformationen über eine ferngesteuerte Ressource unter seiner Kontrolle zu erlangen.

Referenzen

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