OWASP Top 10 per le applicazioni LLM

Versione 1.1

Pubblicato: 16 Ottobre 2023

HTTPS://LLMTOP10.COM

Sommario

Introduzione

L'introduzione sul mercato di massa dei chatbot pre-addestrati a fine 2022 ha innescato un'ondata di frenetico interesse per i modelli di linguaggio a grandi dimensioni (LLM). Le aziende, desiderose di sfruttare il potenziale degli LLM, li stanno integrando rapidamente nei loro sistemi e nelle offerte destinate ai clienti. Tuttavia, la velocità con cui gli LLM vengono adottati ha superato il tempo necessario per stabilire protocolli di sicurezza esaustivi, lasciando molte applicazioni vulnerabili a seri problemi di sicurezza.

Era evidente la necessità di una risorsa unificata per affrontare questi problemi di sicurezza degli LLM. Gli sviluppatori, non sempre avvezzi ai rischi associati agli LLM, si trovavano di fronte a risorse frammentate. La missione di OWASP sembrava quindi perfetta per guidare un'adozione sicura di questa tecnologia.

A chi si rivolge questo documento?

Il nostro pubblico principale sono gli sviluppatori, i data scientist e gli esperti di sicurezza incaricati di pianificare e costruire applicazioni e plugin basati su tecnologie LLM. Il nostro obiettivo è fornire una guida pratica e concisa per aiutare questi professionisti a muoversi nel terreno complesso e in continua evoluzione della sicurezza degli LLM.

La creazione della lista

La creazione dell'OWASP Top 10 per le applicazioni LLM ha richiesto un impegno significativo, realizzata grazie all'esperienza collettiva di un gruppo internazionale di quasi 500 esperti, con più di 125 contributori attivi. I nostri collaboratori provengono da contesti diversi, che includono aziende nel campo dell'intelligenza artificiale, aziende del settore della sicurezza, fornitori indipendenti di software, piattaforme cloud e hyperscale, e il mondo della ricerca accademica.

Nel corso di un mese, abbiamo discusso e proposto potenziali vulnerabilità e i membri del gruppo hanno considerato fino a 43 minacce distinte. Attraverso molteplici round di selezione, abbiamo ridotto queste proposte fino ad arrivare a una lista concisa delle 10 vulnerabilità più critiche.

Ognuna di queste vulnerabilità, congiuntamente agli esempi, ai suggerimenti relativi alla prevenzione, agli scenari di attacco e ai riferimenti, è stata ulteriormente esaminata e rifinita da sottogruppi specializzati e sottoposta a una revisione pubblica, per assicurare che la lista finale fosse il più possibile completa e

concretamente applicabile.

Relazione con le altre liste OWASP Top 10

Anche se la nostra lista condivide il DNA con i tipi di vulnerabilità che si possono trovare nelle altre liste OWASP Top 10, non ci limitiamo a reiterarle, ma analizziamo le implicazioni uniche che queste vulnerabilità hanno quando appaiono in applicazioni basate sugli LLM.

Il nostro obiettivo è di colmare la distanza tra i principi generali di sicurezza delle applicazioni e le sfide specifiche poste dagli LLM. Questo include l'esplorazione di come le vulnerabilità tradizionali possano porre rischi differenti o possano essere sfruttate in nuovi modi con gli LLM, e come i rimedi tradizionali debbano essere adattati alle applicazioni basate sugli LLM.

Riguardo alla versione 1.1

Anche se la nostra lista condivide il DNA con i tipi di vulnerabilità che si possono trovare nelle altre liste OWASP Top 10, non ci limitiamo a reiterarle, ma analizziamo le implicazioni uniche che queste vulnerabilità hanno quando appaiono in applicazioni basate sugli LLM.

Il nostro obiettivo è di colmare la distanza tra i principi generali di sicurezza delle applicazioni e le sfide specifiche poste dagli LLM. Questo include l'esplorazione di come le vulnerabilità tradizionali possano porre rischi differenti o possano essere sfruttate in nuovi modi con gli LLM, e come i rimedi tradizionali debbano essere adattati alle applicazioni basate sugli LLM.

Il futuro

La versione 1.1 di questa lista non sarà l'ultima. Ci aspettiamo di aggiornare questa lista periodicamente, per stare al passo con l'evoluzione del settore. Lavoreremo con la comunità per far evolvere la tecnologia e creare altro materiale di studio per una serie di casi d'uso. Miriamo inoltre a collaborare con gli organismi di standardizzazione e i governi a riguardo della sicurezza dell'intelligenza artificiale. Ti invitiamo a unirti al nostro gruppo e contribuire.

Steve Wilson
Responsabile del progetto OWASP Top 10 per le applicazioni LLM
https://www.linkedin.com/in/wilsonsd
Twitter/X: @virtualsteve

Ads Dawson
Responsabile della release 1.1 e responsabile voci di vulnerabilità per il progetto OWASP Top 10 per le applicazioni LLM
https://www.linkedin.com/in/adamdawson0
GitHub: @GangGreenTemperTatum

Riguardo alla traduzione

Traduttori

- ☐ Fabrizio Cilli
https://www.linkedin.com/in/fabriziocilli/
- ☐ Matteo Dora
https://www.linkedin.com/in/mattbit/
- ☐ Riccardo Sirigu
https://www.linkedin.com/in/riccardosirigu/
- ☐ Valerio Alessandroni
https://www.linkedin.com/in/valerio-alessandroni/

Nella realizzazione di questa traduzione, abbiamo scelto consapevolmente di impiegare solo traduttori umani, riconoscendo la natura eccezionalmente tecnica e critica dell'OWASP Top Ten per gli LLM. I traduttori elencati sopra non solo possiedono una profonda comprensione del contenuto originale, ma anche la fluidità per rendere questa traduzione un successo.

Talesh Seeparsan
Responsabile traduzioni, OWASP Top 10 per le applicazioni LLM
https://www.linkedin.com/in/talesh/

OWASP Top 10 per le applicazioni LLM

LLM01: Iniezione di Prompt

Input artificiosi possono manipolare un modello linguistico di grandi dimensioni, causando azioni non volute. Le iniezioni dirette sovrascrivono i prompt di sistema, mentre quelle indirette manipolano gli input provenienti da fonti esterne.

LLM02: Gestione Non Sicura dell'Output

Questa vulnerabilità si manifesta quando l'output del LLM è accettato senza previa verifica, esponendo i sistemi backend. L'abuso può portare a conseguenze gravi come XSS, CSRF, SSRF, escalation dei privilegi o esecuzione di codice remoto.

LLM03: Avvelenamento dei Dati di Apprendimento

Questo si verifica quando i dati di apprendimento del LLM vengono alterati, introducendo vulnerabilità o bias che ne compromettono la sicurezza, l'efficacia o il comportamento etico. Le fonti di dati includono Common Crawl, WebText, OpenWebText e libri.

LLM04: Denial of Service del Modello

Degli attaccanti causano operazioni che richiedono risorse elevate sui modelli linguistici di grandi dimensioni, portando a degrado del servizio o a costi elevati. La vulnerabilità è amplificata dalla natura intensiva delle risorse degli LLM e dall'imprevedibilità degli input dell'utente.

LLM05: Vulnerabilità della Supply-Chain

Il ciclo di vita dell'applicazione LLM può essere compromesso da componenti o servizi vulnerabili, portando ad attacchi di sicurezza. L'utilizzo di dataset, modelli pre-addestrati e plugin di terze parti può aggiungere altre vulnerabilità.

LLM06: Divulgazione di Informazioni Sensibili

Gli LLM possono rivelare dati confidenziali nelle risposte, portando ad accessi non autorizzati, violazioni della privacy e falle di sicurezza. Per mitigare questo rischio, è cruciale implementare un processo di sanitizzazione dei dati e politiche utente rigorose.

LLM07: Progettazione Insicura dei Plugin

I plugin LLM possono avere input non sicuri e controlli di accesso insufficienti. Questa mancanza di controllo dell'applicazione li rende più facili da sfruttare e può comportare conseguenze come l'esecuzione remota di codice.

LLM08: Eccessiva Autonomia

I sistemi basati sugli LLM possono intraprendere azioni che conducono a conseguenze non volute. Il problema nasce da funzionalità, permessi o autonomia eccessivi concessi a questi sistemi.

LLM09: Eccessivo Affidamento

Senza supervisione, sistemi o persone che fanno eccessivo affidamento sugli LLM possono incorrere in disinformazione, malfunzionamenti, problemi legali e vulnerabilità di sicurezza dovute a contenuti errati o inappropriati generati dagli LLM.

LLM10: Furto del Modello

Questa vulnerabilità consiste nell'accesso non autorizzato, la copia o l'esfiltrazione di modelli LLM proprietari. L'impatto include perdite economiche, compromissione del vantaggio competitivo e potenziale accesso a informazioni sensibili.

LLM01: Iniezione di Prompt

Descrizione

La vulnerabilità di tipo Iniezione di Prompt (inglese: prompt injection) si verifica quando un attaccante manipola un modello linguistico di grandi dimensioni (LLM) attraverso input fatti ad hoc, facendo in modo che il LLM risponda inconsapevolmente alle intenzioni dell'attaccante. Questo può essere fatto direttamente attraverso il "jailbreaking" (effrazione) del prompt di sistema, oppure indirettamente, manipolando gli input esterni, portando potenzialmente all'esfiltrazione di dati, all'ingegneria sociale e altre problematiche.

- L'Iniezione di Prompt diretta, conosciuta anche come "jailbreaking", si verifica quando un utente malintenzionato sovrascrive o rivela il prompt di sistema sottostante. Ciò può consentire agli attaccanti di sfruttare i sistemi backend interagendo con funzioni insicure e basi di dati accessibili tramite il LLM.

- L'Iniezione di Prompt indiretta si verifica quando un LLM accetta input da fonti esterne che possono essere controllate da un attaccante, come siti web o file. L'attaccante può incorporare un'Iniezione di Prompt nel contenuto esterno, dirottando il contesto della conversazione. Ciò causerebbe una minore stabilità dell'output del LLM, consentendo all'attaccante di manipolare l'utente o i sistemi aggiuntivi a cui il LLM può accedere. Inoltre, le iniezioni di prompt indirette non hanno bisogno di essere visibili o leggibili da un umano, purché il testo venga analizzato dal LLM.

I risultati di un attacco di Iniezione di Prompt di successo possono variare notevolmente - dalla richiesta di informazioni sensibili all'influenza su processi decisionali critici sotto mentite spoglie di normale funzionamento.

In attacchi avanzati, il LLM può essere manipolato per impersonare un personaggio malevolo o interagire con plugin nell'ambiente dell'utente. Ciò può portare alla divulgazione di dati sensibili, all'uso non autorizzato di plugin o all'ingegneria sociale. In tali casi, il LLM compromesso aiuta l'attaccante, aggirando i meccanismi di protezione standard e mantenendo l'utente all'oscuro dell'intrusione. In questi casi, il LLM compromesso agisce in sostanza come un agente per l'attaccante, perseguendo i suoi obiettivi senza innescare i normali meccanismi di protezione o senza segnalare l'intrusione all'utente finale.

Esempi comuni di vulnerabilità

1. Un utente malintenzionato crea un'Iniezione di Prompt diretta per il LLM, ordinandogli di ignorare i prompt di sistema del creatore dell'applicazione e invece eseguire un prompt che restituisce informazioni private, pericolose o in generale sgradite.
2. Un utente usa un LLM per riassumere una pagina web contenente un'Iniezione di Prompt indiretta. Ciò fa sì che il LLM richieda informazioni sensibili all'utente e le esfiltri tramite JavaScript o Markdown.
3. Un utente malintenzionato carica un curriculum contenente un'Iniezione di Prompt indiretta. Il documento contiene un'Iniezione di Prompt con istruzioni per far sì che il LLM informi gli utenti che questo documento è eccellente, ad esempio un candidato perfettamente compatibile per un ruolo lavorativo. Un utente interno analizza il documento tramite il LLM per riassumerne il contenuto. In conseguenza all'Iniezione di Prompt, l'output del LLM indica che il documento è eccellente.
4. Un utente abilita un plugin collegato a un sito di e-commerce. Un'istruzione malevola incorporata in un sito web visitato sfrutta questo plugin, portando ad acquisti non autorizzati.
5. Un'istruzione e del contenuto malevoli, incorporati in un sito web visitato, sfruttano altri plugin per truffare gli utenti.

Strategie di prevenzione e mitigazione

Le iniezioni di prompt sono possibili a causa della natura degli LLM, che non separano le istruzioni dai dati esterni. Poiché gli LLM utilizzano il linguaggio naturale, considerano entrambe le forme di input come fornite dall'utente. Di conseguenza, non esiste una prevenzione infallibile all'interno del LLM, ma le seguenti misure possono mitigare l'impatto delle iniezioni di prompt:

1. Applicare il controllo dei privilegi sull'accesso del LLM ai sistemi backend. Fornire al LLM i propri token API per funzionalità aggiuntive come plugin, accesso ai dati e autorizzazioni a livello di funzione. Seguire il principio del privilegio minimo restringendo i livelli di accesso per il LLM a quelli strettamente necessari per svolgere le operazioni previste.
2. Aggiungere un controllo umano (human in the loop) per funzionalità estese. Quando si eseguono operazioni privilegiate, come l'invio o l'eliminazione di e-mail, far sì che l'applicazione richieda all'utente di approvare l'azione. Ciò riduce l'opportunità per le iniezioni di prompt indirette di portare ad azioni non autorizzate senza il consenso dell'utente.
3. Separare il contenuto esterno dai prompt dell'utente. Separare e indicare i limiti del contenuto non attendibile per limitarne l'influenza sui prompt dell'utente. Ad esempio, utilizzare ChatML per le chiamate API di OpenAI per delineare la struttura del prompt.

4. Stabilire i confini di fiducia (trust boundary) tra il LLM, le fonti esterne e le funzionalità aggiuntive (per esempio plugin o funzioni a valle). Tuttavia, un LLM compromesso può comunque agire da intermediario (man-in-the-middle) tra le API dell'applicazione e l'utente, nascondendo o manipolando le informazioni prima di presentarle a quest'ultimo. Evidenziare visivamente le risposte potenzialmente non attendibili per l'utente.
5. Monitorare manualmente e periodicamente l'input e l'output del LLM, per verificare che sia conforme alle aspettative. Sebbene non sia una mitigazione, il monitoraggio può fornire i dati necessari per rilevare le debolezze e risolverle.

Esempi di scenario di attacco

1. Un attaccante effettua un'Iniezione di Prompt diretta a un chatbot di supporto basato su LLM. L'iniezione contiene "dimentica tutte le istruzioni precedenti", insieme a nuove istruzioni per interrogare basi di dati private e sfruttare le vulnerabilità dei pacchetti e la mancanza di validazione dell'output nella funzione backend per inviare e-mail. Ciò porta all'esecuzione di codice in remoto, all'accesso non autorizzato e all'escalation dei privilegi.
2. Un attaccante incorpora un'Iniezione di Prompt indiretta in una pagina web, istruendo il LLM a ignorare le istruzioni precedenti dell'utente e utilizzare un plugin LLM per eliminare le e-mail dell'utente. Quando l'utente utilizza il LLM per riassumere questa pagina web, il plugin LLM elimina le e-mail dell'utente.
3. Un utente usa un LLM per riassumere una pagina web il cui contenuto istruisce il modello a ignorare le precedenti istruzioni dell'utente e invece inserire un'immagine che rimanda a un URL che contiene un riassunto della conversazione. Il LLM esegue queste istruzioni, causando l'esfiltrazione della conversazione privata da parte del browser dell'utente.
4. Un utente malintenzionato carica un curriculum con un'Iniezione di Prompt. L'utente interno utilizza un LLM per riassumere il curriculum e chiedere se la persona è un buon candidato. A causa dell'Iniezione di Prompt, la risposta del LLM è sì, indipendentemente dal contenuto effettivo del curriculum.
5. Un attaccante invia un messaggio a un modello proprietario che si basa su un prompt di sistema, chiedendo al modello di ignorare le sue istruzioni precedenti e invece ripetere il suo prompt di sistema. Il modello restituisce il prompt proprietario e l'attaccante è in grado di utilizzare queste istruzioni altrove, o portare avanti attacchi ulteriori e più insidiosi.

Riferimenti e link (Inglese)

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: Gestione Non Sicura dell'Output

Descrizione

La Gestione Non Sicura dell'Output si riferisce nello specifico a una validazione, sanificazione e gestione insufficiente degli output generati da grandi modelli di linguaggio prima che vengano passati a valle ad altri componenti e sistemi. Poiché il contenuto generato da un LLM può essere controllato dal prompt in input, questo comportamento è comparabile a fornire agli utenti un accesso indiretto a funzionalità aggiuntive.

La Gestione Non Sicura dell'Output si differenzia dalla dipendenza eccessiva (LLM09) in quanto si occupa degli output generati da un LLM prima che vengano passati a valle, mentre la dipendenza eccessiva si concentra su questioni più ampie riguardanti l'eccessiva fiducia nell'accuratezza e nell'appropriatezza degli output di un LLM.

Un attacco che sfrutta la Gestione Non Sicura dell'Output può portare a XSS e CSRF nei browser web, nonché a SSRF, escalation dei privilegi o esecuzione di codice remoto (RCE) nei sistemi backend.

Le condizioni seguenti possono aumentare l'impatto di questa vulnerabilità:

- ☐ L'applicazione concede al LLM privilegi oltre a quelli previsti per gli utenti finali, consentendo l'escalation dei privilegi o l'esecuzione di codice remoto.
- ☐ L'applicazione è vulnerabile ad attacchi di iniezione di prompt indiretta, che potrebbero consentire a un attaccante di ottenere l'accesso privilegiato all'ambiente di un utente vittima.
- ☐ Plugin di terze parti non validano adeguatamente gli input.

Esempi comuni di vulnerabilità

1. L'output di un LLM viene inserito direttamente in una shell di sistema o in una funzione simile come exec o eval, causando l'esecuzione di codice remoto.
2. JavaScript o Markdown generati dal LLM vengono restituiti all'utente. Il codice viene quindi interpretato dal browser, causando un XSS.

Strategie di prevenzione e mitigazione

1. Trattare il modello come qualsiasi altro utente, adottando un approccio di zero-trust (fiducia zero), e applicare una corretta validazione degli input che vengono passati dal modello alle funzioni backend.

2. Seguire le linee guida OWASP ASVS (Application Security Verification Standard) per garantire una validazione e sanificazione efficace degli input.
3. Codificare l'output del modello che viene inviato agli utenti per mitigare l'esecuzione di codice indesiderato tramite JavaScript o Markdown. OWASP ASVS fornisce una guida dettagliata sulla codifica dell'output.

Esempi di scenari di attacco

1. Un'applicazione usa un plugin LLM per generare le risposte di un chatbot. Il plugin offre anche una serie di funzioni amministrative accessibili a un altro LLM privilegiato. Il LLM passa direttamente la sua risposta, senza una corretta validazione dell'output, al plugin causando l'arresto del plugin per manutenzione.
2. Un utente usa uno strumento di sintesi di siti web basato su un LLM per generare un riassunto conciso di un articolo. Il sito web include un'iniezione di prompt che istruisce il LLM a catturare contenuti sensibili dal sito web o dalla conversazione dell'utente. Il LLM può quindi codificare i dati sensibili e inviarli a un server controllato dall'attaccante, senza alcuna validazione o filtraggio dell'output.
3. Un LLM permette agli utenti di creare query SQL per un database nel backend attraverso una chat. Un utente richiede una query per eliminare tutte le tabelle del database. Se la query creata dal LLM non viene filtrata in nessun modo, allora tutte le tabelle del database verranno eliminate.
4. Un'applicazione web usa un LLM per generare contenuto a partire da prompt di testo inseriti dall'utente, senza sanificare l'output. Un attaccante potrebbe inviare un prompt creato ad arte che causa l'invio di un payload JavaScript non sanificato, portando a un XSS quando questo viene interpretato dal browser della vittima. La mancata validazione dei prompt rende possibile questo attacco.

Riferimenti e link (Inglese)

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: Avvelenamento dei Dati di Apprendimento

Descrizione

Il punto di partenza di qualsiasi approccio di machine learning (apprendimento automatico) è costituito dai dati di addestramento, semplicemente del "testo grezzo". Per essere un modello con capacità elevate (ad esempio, che abbia conoscenze linguistiche e del mondo), questo testo deve coprire un'ampia gamma di domini, generi e lingue. Un grande modello di linguaggio utilizza reti neurali profonde per generare output basati su pattern appresi dai dati di apprendimento.

Con avvelenamento dei dati si intende la manipolazione dei dati usati nel processo di pre-apprendimento (pre-training), di fine-tuning o di embedding, al fine di introdurre delle vulnerabilità (che hanno tutte vettori di attacco unici e a volte condivisi), backdoor o bias che potrebbero compromettere la sicurezza, l'efficacia o il comportamento etico del modello. Le informazioni avvelenate possono essere presentate agli utenti o creare altri rischi come la degradazione delle prestazioni, l'exploit del software downstream e danni alla reputazione. Anche se gli utenti non si fidano dell'output problematico dell'IA, i rischi rimangono, comprese le capacità compromesse del modello e potenziali danni alla reputazione del marchio.

- ☐ Il pre-apprendimento si riferisce al processo di addestramento di un modello basato su un compito o un set di dati.
- ☐ Il fine-tuning consiste nel prendere un modello esistente che è già stato addestrato e riadattarlo a un dominio più ristretto o a un obiettivo più focalizzato, addestrandolo utilizzando uno specifico set di dati. Questo set di dati include tipicamente esempi di input e i corrispondenti output desiderati.
- ☐ Il processo di embedding è il processo di conversione di dati categorici (spesso testo) in una rappresentazione numerica che può essere utilizzata per addestrare un modello di linguaggio. L'embedding consiste nella rappresentazione di parole o frasi prese dai dati testuali, come vettori in uno spazio vettoriale continuo. I vettori sono solitamente generati passando dati testuali attraverso una rete neurale che è stata addestrata su un ampio corpus di testo.

L'avvelenamento dei dati è considerato un attacco all'integrità perché la manipolazione dei dati di addestramento influisce sulla capacità del modello di produrre previsioni corrette. Naturalmente, le fonti di dati esterne presentano un rischio maggiore poiché i creatori del modello non hanno controllo su questi dati né la sicurezza che il contenuto non contenga bias, false informazioni o contenuti inappropriati.

Esempi comuni di vulnerabilità

1. Un attaccante o un'azienda concorrente crea intenzionalmente dei documenti inaccurati o malevoli, che sono indirizzati al pre-addestramento, fine-tuning o al processo di embedding del modello. Considerate entrambi i vettori di attacco di "avvelenamento da vista separata" (rif.12) e "avvelenamento per anticipazione" (rif.13) come esempi.
    - Il modello vittima viene addestrato utilizzando informazioni false che si riflettono negli output che il modello di IA generativa fornisce ai suoi fruitori.
2. Un attaccante è in grado di iniettare direttamente dei contenuti falsi, tendenziosi o pericolosi nei processi di addestramento di un modello, facendo così in modo che questi contenuti vengano poi restituiti dai suoi output.
3. Un utente ignaro inietta indirettamente dati sensibili o proprietari nei processi di addestramento di un modello, che vengono poi restituiti negli output successivi.
4. Un modello è addestrato usando dati la cui fonte, origine o contenuto non sono stati sottoposti a verifica in nessuna delle fasi di addestramento, il che può portare a risultati errati se i dati sono contaminati o non corretti.
5. L'accesso senza restrizioni all'infrastruttura o un sandboxing (isolamento) inadeguato possono consentire a un modello di acquisire dati di addestramento non sicuri, con conseguenti output tendenziosi o dannosi. Questo esempio può verificarsi in una qualsiasi delle fasi di addestramento.
    - In questo scenario, l'input che un utente fornisce al modello può essere riflesso nell'output di un altro utente (portando a una violazione), oppure l'utente di un LLM può ricevere dal modello output inesatti, irrilevanti o dannosi a seconda del tipo di dati ingeriti, confrontandoli ai casi d'uso del modello (di solito riportati nella model card).

Che siate sviluppatori, clienti o fruitori generici di un LLM, è importante comprendere come questa vulnerabilità potrebbe riflettersi sui rischi all'interno della vostra applicazione LLM quando questa interagisce con un LLM non proprietario, per comprendere la legittimità degli output del modello in base alle sue procedure di addestramento. Allo stesso modo, gli sviluppatori del LLM potrebbero essere a rischio di attacchi diretti e indiretti ai dati interni o di terze parti utilizzati per il fine-tuning e l'embedding (il più comune), comportando un rischio per tutti i suoi fruitori.

Strategie di prevenzione e mitigazione

1. Verificare la filiera di approvvigionamento dei dati utilizzati per l'addestramento, soprattutto quando sono ottenuti da fonti esterne, nonché mantenere attestazioni tramite la metodologia "ML-BOM" (Machine Learning Bill of Materials) e verificare le model card.

2. Verificare la legittimità delle fonti di dati e dei dati ottenuti durante le fasi di pre-apprendimento, fine-tuning ed embedding.
3. Verificare il caso d'uso del LLM e l'applicazione in cui verrà integrato. Sviluppare modelli specifici, utilizzando set di dati di addestramento distinti o tecniche di fine-tuning per ciascun caso d'uso, per affinare l'output dell'intelligenza artificiale generativa, rendendolo più dettagliato e preciso secondo il caso d'uso definito.
4. Assicurare un adeguato sandboxing attraverso controlli di rete rigorosi, per prevenire che il modello acceda a dati da fonti non autorizzate, evitando così il rischio di compromettere l'integrità dell'output del processo di apprendimento automatico.
5. Adottare un controllo rigoroso o filtri sull'input per dati di addestramento specifici o per determinate categorie di fonti di dati, per mantenere sotto controllo il volume di dati non genuini. Sanificare i dati con tecniche come la rilevazione statistica degli outlier o il rilevamento delle anomalie, per identificare e rimuovere i dati "ostili" che potrebbero essere inseriti nel processo di fine-tuning.
6. Elaborare domande di controllo riguardanti la fonte e la proprietà dei set di dati per garantire che il modello non sia stato avvelenato, e adottare questa cultura nel ciclo di vita "MLSecOps". Fare riferimento a risorse disponibili come ad esempio "The Foundation Model Transparency Index" (rif.14) o "Open LLM Leaderboard" (rif.15).
7. Usare "Data Version Control" (rif.16) per identificare e monitorare in modo rigoroso quella parte del set di dati che potrebbe essere stata manipolata, eliminata o aggiunta, portando all'avvelenamento dei dati.
8. Utilizzare database vettoriali per integrare informazioni fornite dall'utente, consentendo di tutelare gli utenti dall'avvelenamento dei dati e persino di apportare correzioni in ambiente di produzione senza la necessità di riaddestrare un nuovo modello.
9. Tecniche di robustezza avversaria come l'apprendimento federato (federated learning) e l'applicazione di vincoli per minimizzare gli effetti di outlier o dell'addestramento avversariale per aumentare la resistenza alle peggiori perturbazioni che possono presentarsi nei dati di addestramento.
    - Un approccio "MLSecOps" potrebbe essere quello di includere una fase di rafforzamento avversariale nel ciclo di vita dell'addestramento utilizzando ad esempio la tecnica del "auto-avvelenamento".
    - Un esempio di questo approccio è "Autopoison" (rif.17), che include test per attacchi come l'iniezione diretta di contenuto ("tentativo di promuovere un marchio nelle risposte del modello") e l'attacco di rifiuto ("fare in modo che il modello rifiuti sempre di rispondere").
10. Testare e rilevare, misurando la perdita durante la fase di addestramento e analizzando i modelli addestrati per rilevare gli indizi di un attacco di avvelenamento, controllando il comportamento del modello su input di test specifici.
11. Monitorare e segnalare il numero di risposte distorte che superano una certa

soglia.
12. Usare una validazione umana per fare un audit delle risposte.
13. Implementare LLM dedicati per mettere alla prova i modelli rispetto a conseguenze indesiderate e addestrare altri LLM usando tecniche di apprendimento con rinforzo (rif.18).
14. Eseguire esercitazioni di tipo red team (rif.19) basate su LLM o scansioni di vulnerabilità (rif.20) nelle fasi di test del ciclo di vita del LLM.

Esempi di scenario di attacco

1. L'output di un'IA generativa può indurre in errore gli utenti dell'applicazione, portando a opinioni tendenziose o, peggio ancora, reati d'odio, ecc. con scopi tendenziosi.
2. Se i dati di apprendimento non vengono correttamente filtrati e/o sanificati, un utente malevolo potrebbe tentare di influenzare il modello iniettando dati tossici per farlo adattare ai dati falsificati e tendenziosi.
3. Un utente malevolo o un concorrente crea deliberatamente documenti inaccurati o malevoli indirizzati ai dati di addestramento di un modello, che è in fase di addestramento al tempo stesso in base a questi input. Il modello vittima impara utilizzando informazioni falsificate che si riflettono negli output che il modello di IA generativa fornisce ai suoi fruitori.
4. L'iniezione di prompt (LLM01) potrebbe essere un vettore di attacco per questa vulnerabilità se non viene eseguita una sanitizzazione e un filtraggio adeguati quando gli input degli utenti sono usati per addestrare il modello. Ad esempio, se dati malevoli o falsificati vengono inseriti nel modello da un utente attraverso una tecnica di iniezione di prompt, questi potrebbero essere intrinsecamente rappresentati nei dati su cui il modello si basa.

Riferimenti e link (Inglese)

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. tecniche di apprendimento con rinforzo
19. red team
20. scansioni di vulnerabilità

LLM04: Denial of Service del Modello

Descrizione

Un attaccante può interagire con un LLM (Large Language Model) in modo tale da causare un consumo eccezionalmente alto di risorse, risultando in un deterioramento della qualità del servizio sia per sé stesso che per altri utenti, e potenzialmente causando un aumento dei costi per le risorse computazionali. Un altro aspetto critico per la sicurezza è la possibilità che un attaccante interferisca o manipoli la "finestra di contesto" di un LLM. Questa problematica sta diventando sempre più rilevante a causa dell'uso crescente degli LLM in diverse applicazioni, del loro intenso utilizzo di risorse, dell'imprevedibilità degli input degli utenti e di una generale mancanza di consapevolezza tra gli sviluppatori riguardo a questa vulnerabilità.
Negli LLM, la finestra di contesto rappresenta la lunghezza massima di testo che il modello può gestire, includendo sia l'input che l'output. Questa caratteristica è fondamentale per gli LLM, poiché determina la complessità dei costrutti linguistici che il modello può comprendere e la quantità di testo che può elaborare in un dato momento. La dimensione della finestra di contesto è definita dall'architettura del modello e può variare a seconda del modello specifico utilizzato.

Esempi comuni di vulnerabilità

1. Porre domande (prompts) che inducono a un utilizzo ripetitivo delle risorse attraverso la creazione di un elevato numero di compiti in coda, ad esempio impiegando strumenti come LangChain o AutoGPT.
2. Inviare interrogazioni insolitamente gravose in termini di risorse, utilizzando ortografie o sequenze di testo non convenzionali.
3. Sovraccarico continuo dell'input: un attaccante invia costantemente al LLM un flusso di input che eccede la finestra di contesto del modello, causando un consumo eccessivo di risorse computazionali.
4. Input lunghi ripetuti: l'attante invia al LLM input estesi in modo ripetitivo, ciascuno dei quali supera la capacità della finestra di contesto.
5. Espansione ricorsiva del contesto: l'attaccante formula un input che provoca un'espansione ricorsiva del contesto, obbligando il LLM a ingrandire e processare più volte la finestra di contesto.
6. Inondazione di input di lunghezza variabile: l'attaccante inonda il LLM con una grande quantità di input di lunghezze diverse, ognuno progettato per sfiorare il limite massimo della finestra di contesto. Questa tattica mira a sfruttare le inefficienze nella gestione degli input di dimensioni variabili, mettendo a dura prova il LLM e potenzialmente causandone il blocco.

Strategie di prevenzione e mitigazione

1. Implementare un controllo dell'input per assicurare che gli input degli utenti rispettino limiti predefiniti e siano privi di contenuti malevoli.
2. Impostare limiti alle risorse utilizzabili per ogni richiesta o passaggio, rallentando così l'esecuzione delle richieste più complesse.
3. Applicare limiti di frequenza alle chiamate API per contenere il numero di richieste che un singolo utente o indirizzo IP può fare entro un determinato lasso di tempo.
4. Limitare il numero di azioni in coda e il numero totale di azioni in un sistema terzo che interagisce con il LLM.
5. Monitorare continuamente l'utilizzo delle risorse del LLM per identificare picchi o schemi anomali che potrebbero indicare un attacco DoS.
6. Impostare limiti rigorosi sugli input in base alla finestra di contesto del LLM, per prevenire eccessivi sovraccarichi e l'esaurimento delle risorse.
7. Aumentare la consapevolezza tra gli sviluppatori sulle potenziali vulnerabilità ai DoS negli LLM e fornire indicazioni per un'implementazione sicura degli stessi.

Esempi di scenari di attacco

1. Un attaccante invia ripetutamente molteplici richieste complesse e onerose a un modello in hosting, portando a un deterioramento del servizio per gli altri utenti e a un aumento dei costi relative ai consumi delle risorse, nel servizio di hosting.
2. Durante l'elaborazione di un testo su una pagina web da parte di uno strumento basato su LLM che risponde a una query benigna, lo strumento si imbatte in un testo specifico e inizia a richiedere un numero eccessivo di pagine web, comportando un alto consumo di risorse.
3. Un attaccante bombarda continuamente il LLM con input che superano la sua finestra di contesto. L'attaccante può utilizzare script automatizzati o strumenti per inviare un alto volume di input, sovraccaricando le capacità di elaborazione del LLM. Di conseguenza, viene causato un eccessivo consumo di risorse computazionali, con conseguente rallentamento o inoperatività del sistema ospitante.
4. Un attaccante invia una serie di input sequenziali al LLM, in cui ciascun input è progettato per essere appena sotto il limite della finestra di contesto. Inviando ripetutamente questi input, l'attaccante mira a esaurire la capacità disponibile della finestra. Mentre il LLM fatica a elaborare ciascun input all'interno della sua finestra di contesto, le risorse del sistema si riducono drasticamente, risultando in un degrado delle prestazioni o in un completo diniego del servizio (DoS).
5. Un attaccante sfrutta i meccanismi ricorsivi del LLM per causare ripetutamente l'espansione del contesto. Creando un input che stimola il comportamento ricorsivo del LLM, l'attaccante costringe il modello a espandere e processare ripetutamente la finestra di contesto, consumando una significativa quantità di

risorse computazionali. Questo attacco mette sotto sforzo il sistema e può portare a una condizione di DoS, rendendo il LLM non reattivo o causandone il fermo totale.

6. Un attaccante inonda il LLM con un grande volume di input di lunghezza variabile, costruiti specificamente per avvicinarsi o raggiungere il limite della finestra di contesto. Sopraffacendo il LLM con input di lunghezze variabili, l'attaccante mira a sfruttare qualsiasi inefficienza nell'elaborazione di questo tipo di input. Questo sovraccarico pesa eccessivamente sulle risorse del LLM, provocando un degrado delle prestazioni e ostacolando la capacità del sistema di rispondere a richieste legittime.
7. Mentre gli attacchi di tipo Denial of Service (DoS) puntano generalmente a sovraccaricare le risorse di un sistema, possono anche mirare ad altri aspetti del suo funzionamento, come sfruttare le vulnerabilità nelle limitazioni dell'API. Per esempio, in un recente episodio di sicurezza che ha coinvolto Sourcegraph, un attore malevolo ha acquisito un token di accesso amministrativo e lo ha usato per modificare i limiti di frequenza delle chiamate API. Questa azione avrebbe potuto causare interruzioni del servizio, abilitando volumi insolitamente alti di richieste.

Riferimenti e link (Inglese)

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: Vulnerabilità della Supply-Chain

Descrizione

La catena di approvvigionamento nei modelli di linguaggio di grandi dimensioni (LLM) può essere soggetta a vulnerabilità, influenzando l'integrità dei dati di addestramento, dei modelli di machine learning (ML) e delle piattaforme di distribuzione. Queste vulnerabilità possono causare risultati distorti, violazioni della sicurezza o persino fallimenti completi del sistema. Tradizionalmente, le vulnerabilità si concentrano sui componenti software, ma il Machine Learning le estende anche ai modelli pre-addestrati e ai dati di addestramento forniti da terze parti, suscettibili ad attacchi di manomissione e avvelenamento.

Infine, le estensioni dei plugin LLM possono introdurre ulteriori vulnerabilità. Queste sono descritte in LLM07 - Progettazione Insicura dei Plugin (pp. 25-27), che tratta dello sviluppo di plugin LLM e fornisce informazioni utili per valutare i plugin di terze parti.

Esempi comuni di vulnerabilità

1. Vulnerabilità derivanti dai pacchetti di terze parti, che includono componenti obsoleti o deprecati.
2. Utilizzo di un modello pre-addestrato vulnerabile per il fine-tuning.
3. Uso di dati di addestramento distorti provenienti da fonti aperte non verificate.
4. L'utilizzo di modelli obsoleti o deprecati che non sono più mantenuti che portano a problemi di sicurezza.
5. Termini e Condizioni di Servizio (T&C) e politiche sulla privacy dei dati non trasparenti da parte degli operatori dei modelli possono portare all'utilizzo di dati sensibili di un'applicazione per l'addestramento del modello, e alla loro successiva esposizione. Questo può anche implicare problemi legali legati all'uso di materiale protetto da proprietà intellettuale, da parte del fornitore del modello.

Strategie di prevenzione e mitigazione

1. Valutare attentamente le fonti dei dati e i fornitori, inclusi i T&C e le loro politiche sulla privacy, scegliendo solo fornitori affidabili. Assicurarsi della presenza di una solida pratica di sicurezza certificata da terze parti e che le politiche degli operatori dei modelli siano allineate con le proprie politiche di protezione dei dati, ad esempio che i dati non vengano utilizzati per l'addestramento dei modelli senza consenso; non meno importante, predisporre

le necessarie garanzie legali e mitigazioni contro l'eventuale uso di materiale protetto da copyright da parte dei manutentori dei modelli.
2. Selezionare solo plugin di comprovata affidabilità e conformità ai requisiti specifici dell'applicazione. Il paragrafo LLM07 - Progettazione Insicura dei Plugin (rif.11) fornisce informazioni per comprendere le criticità nel design dei plugin e come testare quelli di terze parti per ridurre i rischi.
3. Esaminare e applicare le mitigazioni trovate nella Top Ten dell'OWASP "A06:2021 – Vulnerable and Outdated Components" (rif.11). Fra questi la scansione di vulnerabilità e l'aggiornamento di componenti, estendendo tali controlli anche agli ambienti di sviluppo che trattano dati sensibili.
4. Mantenere un inventario aggiornato dei componenti definendo una Software Bill of Materials (SBoM o lista artefatti) per assicurarsi di avere un inventario aggiornato, accurato e certificato, prevenendo così la manomissione dei pacchetti in produzione. Le liste artefatti software (SBoM) possono essere attivamente impiegate per rilevare e segnalare tempestivamente nuove vulnerabilità critiche (Zero-Day).
5. Al momento della scrittura, le liste SBoM non coprono i modelli, i loro artefatti e i set di dati. Se l'applicazione che fa uso di LLM utilizza un modello privato, è bene utilizzare le buone pratiche di "MLOps" e riferirsi a piattaforme che offrono repository di modelli sicuri con dati, modelli ed esperimenti identificabili e tracciabili.
6. Utilizzare modelli e codice certificati o firmati, quando si impiegano modelli e fornitori esterni.
7. L'esecuzione di test avversariali di robustezza e per la rilevazione di anomalie sui modelli e sui dati forniti possono aiutare a rilevare manomissioni e avvelenamento (poisoning) come discusso in "LLM03: Avvelenamento dei Dati di Apprendimento" (pp. 12-16); idealmente, questi test dovrebbero essere parte integrante della pipeline di MLOps; tuttavia, trattandosi di tecniche emergenti, potrebbero risultare più semplici da implementare come parte degli esercizi di red teaming.
8. Implementare un adeguato monitoraggio per assicurare la scansione delle vulnerabilità dei componenti e dell'ambiente che li ospita, l'uso di plugin non autorizzati e componenti obsoleti, inclusi il modello e i suoi artefatti.
9. Implementare una politica di aggiornamento (patching) per mitigare l'insorgenza di componenti vulnerabili o obsoleti. Assicurarsi che l'applicazione si basi su una versione costantemente manutenuta delle API e del modello sottostante.
10. Rivedere e controllare regolarmente la sicurezza e i criteri di accesso dei fornitori, assicurandosi che non ci siano cambiamenti nella loro postura di sicurezza o nelle loro condizioni di servizio o T&C.

Esempi di scenari di attacco

1. Un attaccante sfrutta una libreria Python vulnerabile per compromettere un

sistema. Questo è effettivamente accaduto nel primo data breach di Open AI.
2. Un attaccante fornisce un plugin LLM per la ricerca di voli, generando link falsi che portano a truffare gli utenti.
3. Un attaccante sfrutta il registro dei pacchetti PyPi per ingannare gli sviluppatori di modelli a scaricare un pacchetto compromesso ed esfiltrare dati o aumentare i propri privilegi nell'ambiente di sviluppo dei modelli. Questo scenario si basa su un attacco realmente avvenuto.
4. Un attaccante avvelena un modello pre-addestrato disponibile pubblicamente specializzato in analisi economica e ricerca sociale per creare una backdoor che genera disinformazione e fake news. Lo distribuisce su un marketplace di modelli (ad esempio, Hugging Face) per farlo utilizzare da vittime ignare.
5. Un attaccante avvelena set di dati pubblicamente disponibili per aiutare a creare una backdoor che acquisisce rilevanza quando si effettua il fine-tuning dei modelli. La backdoor favorisce in modo impercettibile alcune aziende in diversi mercati.
6. Un dipendente vittima di compromissione presso un fornitore (sviluppatore in outsourcing, azienda di hosting, ecc.) esfiltra dati, modelli o codice, di fatto trafugando proprietà intellettuale.
7. Un operatore LLM cambia le sue condizioni d'uso (T&C) e la Politica sulla Privacy (DPA) richiedendo un opt-out esplicito relativamente all'uso dei dati dell'applicazione per l'addestramento del modello, portando alla raccolta di dati sensibili.

Riferimenti e link (Inglese)

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. A06:2021 – Vulnerable and Outdated Components

LLM06: Divulgazione di Informazioni Sensibili

Descrizione

Le applicazioni LLM possono rivelare informazioni sensibili, algoritmi proprietari o altri dettagli confidenziali attraverso i loro output. Ciò può portare ad accessi non autorizzati, esposizione di dati sensibili, proprietà intellettuale, violazioni della privacy e altre problematiche di sicurezza. È importante che gli utenti delle applicazioni LLM siano consapevoli di come interagire in modo sicuro con gli LLM e identificare i rischi associati all'inserimento involontario di dati sensibili i quali potrebbero poi essere divulgati dal LLM in altri contesti.

Per mitigare questo rischio, le applicazioni LLM dovrebbero implementare un'adeguata sanificazione dei dati per impedire che i dati degli utenti entrino indiscriminatamente nei set di dati di addestramento del modello. I gestori di applicazioni LLM dovrebbero inoltre fornire dei Termini di Utilizzo adeguati, facilmente accessibili per informare gli utenti come vengono gestiti i loro dati, e l'opzione ben visibile per negare il consenso a includere i loro dati nei processi di addestramento del modello.

L'interazione tra l'utente e l'applicazione LLM instaura un contesto di fiducia reciproca, nel quale non possiamo fidarci intrinsecamente né dell'input utente->LLM né dell'output LLM->utente. È importante notare che questa vulnerabilità presuppone che certi prerequisiti siano assicurati al di fuori del presente ambito di analisi, fra questi gli esercizi di modellazione delle minacce (threat modeling), la protezione delle infrastrutture e una segregazione adeguata degli ambienti di esecuzione. Aggiungere restrizioni all'interno del prompt del sistema riguardo ai tipi di dati che il LLM dovrebbe restituire può fornire una mitigazione parziale contro la divulgazione di informazioni sensibili, tuttavia, data l'imprevedibilità degli LLM è possibile che tali restrizioni potrebbero non essere sempre efficaci e potrebbero essere aggirate tramite iniezione di prompt o altri metodi.

Esempi comuni di vulnerabilità

1. Filtraggio incompleto o inefficace delle informazioni sensibili presenti nelle risposte del LLM.
2. Sovradattamento o memorizzazione di dati sensibili nel processo di addestramento del LLM.
3. Divulgazione involontaria di informazioni confidenziali a causa di errata interpretazione da parte del LLM, mancanza di metodi di pulizia dei dati o errori.

Strategie di prevenzione e mitigazione

1. Integrare adeguate tecniche di sanificazione e pulizia dei dati per impedire che i dati degli utenti entrino nei set di dati di addestramento del modello.
2. Implementare metodi robusti di validazione e sanificazione degli input per identificare ed escludere potenziali input malevoli per prevenire l'avvelenamento (poisoning) del modello.
3. Quando si arricchisce il modello con dati e se si effettua il "fine-tuning" (rif.7) di un modello (ad esempio, dati inseriti nel modello prima o durante il rilascio):
    - Qualunque informazione sensibile nei dati di fine-tuning ha il potenziale di essere rivelato a un utente. Pertanto, si raccomanda di applicare la buona pratica di minimizzazione dei privilegi di accesso, e di non addestrare il modello su informazioni a cui un utente con privilegi elevati può accedere poiché potrebbero inavvertitamente essere mostrate a un utente con privilegi inferiori.
    - L'accesso a fonti di dati esterne (orchestrazione dei dati in tempo reale) dovrebbe essere limitato.
    - Applicare metodi stringenti di controllo degli accessi alle fonti di dati esterne e un approccio rigoroso nella gestione di una catena di approvvigionamento sicura.

Esempi di scenari di attacco

1. L'utente legittimo e ignaro A viene esposto a dati di altri utenti tramite il LLM quando interagisce con l'applicazione LLM in modo non malevolo.
2. L'utente A indirizza un insieme ben congegnato di prompt per bypassare i filtri di input e la sanificazione del LLM per far sì che divulghi informazioni sensibili (PII) su altri utenti dell'applicazione.
3. Dati personali (PII) vengono introdotti nel modello durante il processo di addestramento a causa di negligenza da parte dell'utente stesso o dell'applicazione LLM. Questo scenario potrebbe aumentare il rischio e la probabilità degli scenari 1 o 2 sopra descritti.

Riferimenti e link (Inglese)

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: Progettazione Insicura dei Plugin

Descrizione

I plugin LLM sono estensioni che, una volta attivate, vengono invocate automaticamente dal modello durante le interazioni con l'utente. La gestione di questi plugin è affidata alla piattaforma che integra il modello, e l'applicazione che lo utilizza potrebbe non avere il controllo diretto sulla loro esecuzione, in particolare quando il modello è gestito da un fornitore esterno. Inoltre, i plugin tendono ad accettare input di testo libero dal modello senza alcuna validazione o controllo sui tipi che gestisca le limitazioni sulla dimensione del contesto. Ciò consente a un potenziale attaccante di formulare una richiesta malevola al plugin, che potrebbe portare a una serie di comportamenti indesiderati, inclusa l'esecuzione di codice remoto.

Il danno causato da input malevoli dipende spesso dai controlli di accesso insufficienti e dalla mancata gestione nel tracciare le autorizzazioni tra i diversi plugin. Un controllo di accesso inadeguato permette a un plugin di fidarsi ciecamente di altri plugin e di presumere che gli input ricevuti siano stati forniti dall'utente finale. Tale controllo di accesso inadeguato può consentire agli input malevoli di avere conseguenze dannose che vanno dall'esfiltrazione di dati, all'esecuzione di codice remoto, fino all'acquisizione di privilegi non autorizzati.

Questa sezione si concentra sulla creazione di plugin specifici per LLM piuttosto che sui plugin di terze parti, coperti dalle Vulnerabilità della Catena di Approvvigionamento del LLM (Supply-Chain-Vulnerabilities).

Esempi comuni di vulnerabilità

1. Un plugin accetta tutti i parametri in un unico campo di testo anziché in parametri di input distinti.
2. Un plugin accetta stringhe di configurazione invece di parametri, che possono sovrascrivere intere impostazioni di configurazione.
3. Un plugin accetta comandi SQL o istruzioni di programmazione invece di parametri ristretti.
4. L'autenticazione è eseguita senza un'autorizzazione esplicita per un particolare plugin.
5. Un plugin tratta tutti i contenuti LLM come se fossero creati interamente dall'utente eseguendo qualsiasi azione richiesta senza chiedere ulteriori autorizzazioni.

Strategie di prevenzione e mitigazione

1. I Plugin dovrebbero accettare input che siano limitati e parametrizzati ove possibile e includere controlli sul tipo e la struttura dell'input. Dove non sia consentito, è opportuno inserire un secondo livello di chiamate fortemente tipizzate (controllo rigoroso sui tipi di parametro), analizzando le richieste e applicando validazione e sanificazione. Per gli input liberi, eventualmente necessari per alcune funzionalità dell'applicazione, è cruciale un'attenta revisione per prevenire l'invocazione di metodi dannosi.
2. Gli sviluppatori di plugin dovrebbero applicare le linee guida OWASP per gli standard di verifica della sicurezza applicativa (ASVS - Application Security Verification Standard) per assicurare adeguate validazione e sanitizzazione dell'input.
3. I Plugin dovrebbero essere ispezionati e testati in modo approfondito per assicurare adeguata validazione. Utilizzare sistemi di Scansione Statica del Codice (SAST) e Test Dinamici e Interattivi (DAST, IAST) all'interno dei processi (pipeline) di sviluppo.
4. I Plugin dovrebbero essere progettati con l'intento di minimizzare l'impatto dello sfruttamento di qualunque parametro di input insicuro come da linee guida sui Controlli di Accesso nello standard ASVS OWASP. Ciò prevede un controllo accessi che assicuri i privilegi strettamente necessari, e l'esposizione delle funzionalità strettamente necessarie al corretto funzionamento.
5. I Plugin dovrebbero utilizzare meccanismi di autorizzazione standardizzati come OAuth2, per consentire l'applicazione efficace dei controlli di autorizzazione e accesso. Inoltre le chiavi API dovrebbero essere utilizzate per fornire un contesto in cui applicare specifiche decisioni autorizzative che riflettano il flusso del plugin come chiaramente distinto da quello interattivo dell'utente.
6. Richiedere un intervento umano per l'autorizzazione e la conferma di ogni azione intrapresa da plugin particolarmente critici.
7. I Plugin sono tipicamente delle REST API, per cui si raccomanda agli sviluppatori l'applicazione delle raccomandazioni di cui alla lista OWASP Top 10 API Security Risks - 2023 per ridurre la presenza di vulnerabilità comuni.

Esempi di scenari di attacco

1. Un plugin accetta un URL base e istruisce il LLM nel combinare l'URL con una query per ottenere previsioni meteorologiche che sono incluse nell'elaborazione della richiesta dell'utente. Un utente malintenzionato può creare una richiesta in modo tale che l'URL punti verso un dominio sotto il suo controllo, permettendogli di iniettare il proprio contenuto nel modello LLM tramite il proprio dominio.
2. Un plugin accetta un input in forma libera in un unico campo che non viene validato. Un attaccante fornisce payload creati ad hoc per ottenere informazioni

utili a partire dai messaggi di errore. Poi sfrutta vulnerabilità conosciute nelle dipendenze di terze parti per eseguire del codice arbitrario ed effettuando un'esfiltrazione di dati o aumentando i propri di privilegi.
3. Un plugin utilizzato per recuperare delle inclusioni (embedding) da un base di dati vettoriale accetta parametri di configurazione come stringa di connessione senza effettuarne la validazione. Ciò permette a un attaccante di sfruttare questa problematica e accedere ad altre basi di dati vettoriali cambiando nomi o parametri host ed esfiltrare rappresentazioni vettoriali a cui non dovrebbe avere accesso.
4. Un plugin accetta direttamente clausole SQL "WHERE" come parte di filtri avanzati, che vengono poi aggiunti alla query SQL. Ciò permette all'attaccante di eseguire un attacco di iniezione SQL.
5. Un attaccante utilizza l'iniezione indiretta di prompt per attaccare un plugin di gestione del codice insicuro, privo di validazione dell'input e con controlli di accesso deboli, per trasferire la proprietà del repository (archivio) e bloccare l'utente dai propri.

Riferimenti e link (Inglese)

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

LLM08: Eccessiva Autonomia

Descrizione

Un sistema basato su LLM è spesso dotato di una certa autonomia dallo sviluppatore - la capacità di interfacciarsi con altri sistemi e prendere iniziative in risposta a un prompt. La decisione su quali funzioni invocare può anche essere delegata a un 'agente' per determinarla dinamicamente in base al prompt di input o all'output del LLM.

L'Eccessiva Autonomia è la vulnerabilità che permette di intraprendere azioni dannose in risposta a output inaspettati o ambigui generati da un LLM (indipendentemente dalla causa del malfunzionamento del LLM; sia essa allucinazione/confabulazione, iniezione di prompt diretta/indiretta, plugin malevolo, prompt mal progettati, o semplicemente un modello con prestazioni scadenti). La causa principale dell'Eccessiva Autonomia è tipicamente una o più delle seguenti: troppe funzionalità, permessi eccessivi o autonomia troppo elevata. Questo si distingue dalla Gestione Insicura dell'Output, che si riferisce invece alla mancanza di controlli adeguati sugli output generati dal LLM.

L'Eccessiva Autonomia può avere ripercussioni significative in termini di riservatezza, integrità e disponibilità, variando ampiamente a seconda dei sistemi con cui l'applicazione basata su LLM interagisce.

Esempi comuni di vulnerabilità

1. Funzionalità Eccessiva: Un agente LLM ha accesso a plugin che includono funzioni non strettamente necessarie. Ad esempio, uno sviluppatore deve concedere a un agente LLM la capacità di leggere documenti da un repository, ma il plugin di terze parti che sceglie di utilizzare consente anche la modifica o l'eliminazione dei documenti.
2. Funzionalità Eccessiva: Un plugin potrebbe essere stato utilizzato come prova durante una fase di sviluppo e poi scartato in favore di un'alternativa migliore, ma il plugin originale rimane disponibile all'agente LLM.
3. Funzionalità Eccessiva: Un plugin LLM con funzionalità aperte non riesce a filtrare adeguatamente le istruzioni di input consentendo l'esecuzione di comandi che superano le funzionalità previste dall'applicazione. Ad esempio, un plugin per eseguire un specifico comando shell non impedisce adeguatamente l'esecuzione di altri comandi shell.
4. Permessi Eccessivi: Un plugin LLM ha permessi su altri sistemi che non sono essenziali per il funzionamento previsto dell'applicazione. Ad esempio, un

plugin progettato per leggere dati si connette a una base di dati utilizzando un'utenza che non ha solo permessi SELECT, ma anche UPDATE, INSERT e DELETE.
5. Permessi Eccessivi: Un plugin LLM progettato per operare per conto di un utente accede a sistemi a valle con un'utenza che possiede privilegi elevati. Ad esempio, un plugin per leggere l'archivio documenti dell'utente corrente si connette al repository dei documenti con un account privilegiato che ha accesso ai file di tutti gli utenti.
6. Autonomia Eccessiva: Un'applicazione o plugin basato su LLM non verifica e approva in modo indipendente azioni ad alto impatto. Ad esempio, un plugin che consente di eliminare i documenti di un utente esegue cancellazioni senza esplicita conferma da parte dell'utente.

Strategie di prevenzione e mitigazione

Le seguenti azioni possono prevenire l'Eccessiva Autonomia:

1. Consentire agli agenti LLM l'utilizzo di plugin/strumenti che offrono le funzioni strettamente necessarie al loro funzionamento. Ad esempio, se un sistema basato su LLM non richiede la capacità di recuperare i contenuti di un URL, tale plugin non dovrebbe essere offerto all'agente LLM.
2. Limitare le funzioni implementate nei plugin/strumenti LLM al minimo necessario. Ad esempio, un plugin che accede alla casella di posta elettronica di un utente per riassumere le email dovrebbe limitarsi alla capacità di leggere le email, quindi il plugin non dovrebbe includere altre funzionalità come l'eliminazione o l'invio di messaggi.
3. Evitare, dove possibile, funzioni aperte (ad esempio, eseguire un comando shell, recuperare un URL, ecc.) e usare plugin/strumenti con funzionalità più granulari. Ad esempio, un'app basata su LLM potrebbe aver bisogno di scrivere alcuni output su un file. Se ciò venisse implementato utilizzando un plugin per eseguire una funzione shell, la superficie d'attacco sarebbe molto più ampia (potrebbe essere eseguito qualsiasi altro comando shell). Un'alternativa più sicura potrebbe essere la realizzazione di un plugin per la scrittura di file che supporta solo quella specifica funzionalità.
4. Limitare i permessi concessi ai plugin/strumenti LLM verso altri sistemi ai minimi necessari per limitare l'ambito di azioni indesiderate. Ad esempio, un agente LLM che utilizza una base di dati di prodotti per fornire suggerimenti d'acquisto necessita solo dell'accesso in lettura alla tabella 'prodotti'; non dovrebbe avere accesso ad altre tabelle, né la capacità di inserire, aggiornare o eliminare record. Ciò dovrebbe essere implementato applicando i permessi appropriati all'utenza che il plugin LLM utilizza per connettersi alla base di dati.
5. Monitorare le autorizzazioni dell'utente e il perimetro di sicurezza per garantire che le azioni intraprese per conto di un utente vengano eseguite sui sistemi a valle nel contesto previsto per quell'utente specifico e con i privilegi minimi

necessari. Ad esempio, un plugin LLM che legge il repository di codice di un utente dovrebbe richiedere all'utente di autenticarsi tramite OAuth e con l'ambito minimo richiesto per lo scopo.
6. Utilizzare il controllo umano nel processo decisionale (human-in-the-loop) per richiedere l'approvazione umana di tutte le azioni prima che queste vengano intraprese. Questo può essere implementato in un sistema "terzo" (al di fuori dell'ambito dell'applicazione LLM) o all'interno del plugin/strumento LLM stesso. Ad esempio, un'app basata su LLM che crea e pubblica contenuti sui social media per conto di un utente dovrebbe includere una routine che preveda l'esplicita approvazione di quest'ultimo all'interno del plugin/strumento/API che effettua l'operazione di pubblicazione.
7. Implementare meccanismi di autorizzazione nei sistemi esterni piuttosto che lasciar decidere al modello LLM se un'azione sia consentita o meno. Quando si implementano strumenti/plugin, applicare il principio di mediazione assoluta dei flussi per assicurare che tutte le richieste fatte ai sistemi a valle tramite i plugin/strumenti vengano validate rispetto alle politiche di sicurezza.

Le seguenti opzioni non prevengono l'Eccessiva Autonomia, ma possono limitare il livello di danno causato:

1. Registrare e monitorare l'attività dei plugin/strumenti LLM e dei sistemi da essi contattati per identificare eventuali azioni indesiderate, e rispondere di conseguenza.
2. Implementare un limite di frequenza (rate-limiting) per ridurre il numero di azioni indesiderate che possono verificarsi in un dato periodo di tempo, aumentando la probabilità di identificare azioni indesiderate tramite il monitoraggio prima che possano verificarsi danni significativi.

Esempi di scenari di attacco

Un'app assistente personale basata su LLM ottiene l'accesso alla casella di posta elettronica di un utente tramite un plugin per riassumere il contenuto delle email in arrivo. Per ottenere questa funzionalità, il plugin di posta elettronica necessita la capacità di leggere i messaggi, tuttavia il plugin scelto dallo sviluppatore del sistema include anche funzioni per l'invio di email. Il LLM è vulnerabile a un attacco di iniezione indiretta di prompt, in cui un'email malevola induce il LLM a comandare il plugin di posta elettronica per utilizzare la funzione 'invia messaggio' e inviare spam dalla casella di posta dell'utente. Questo scenario potrebbe essere evitato:
(a) eliminando la funzionalità eccessiva utilizzando un plugin che si limita esclusivamente alla lettura della posta,
(b) eliminando i permessi eccessivi autenticandosi al servizio email dell'utente tramite una sessione OAuth con privilegi di sola lettura, e/o
(c) eliminando l'autonomia eccessiva chiedendo all'utente di approvare manualmente ogni invio di email redatto dal plugin LLM.

In aggiunta, il danno causato potrebbe essere mitigato implementando un limite alla frequenza di invio sull'interfaccia che spedisce la posta elettronica.

Riferimenti e link (Inglese)

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: Eccessivo Affidamento

Descrizione

L'Eccessivo Affidamento si manifesta quando un LLM produce risultati erronei, presentati con una falsa aura di autorità.
Mentre gli LLM usualmente producono materiale creativo e informativo, questi possono anche generare contenuti fattualmente scorretti, inappropriati o pericolosi. Questo fenomeno è noto sotto il nome di allucinazione o confabulazione. Quando persone o sistemi si affidano incondizionatamente a queste informazioni, si possono generare violazioni della sicurezza, disinformazione, comunicazione errata, problemi legali, e danni reputazionali.

Il codice Sorgente generato da LLM, può introdurre silenziosamente delle vulnerabilità di sicurezza. Ciò espone a rischi significativi per l'integrità operativa, e la sicurezza delle applicazioni. Questi rischi sottolineano l'importanza di continui processi di verifica, attraverso:

- ☐ Supervisione
- ☐ Meccanismi di validazione continua
- ☐ Note agli utilizzatori sui rischi connessi all'utilizzo di tali tecnologie.

Esempi comuni di vulnerabilità

1. Un LLM fornisce informazioni inesatte come risposta, presentate in maniera molto autorevole. L'intero sistema è progettato senza adeguati controlli e bilanciamento per gestire queste situazioni e le informazioni generate possono traviare l'utente in maniera da generare potenziali danni.
2. Un LLM propone codice insicuro o difettoso, introducendo vulnerabilità quando incorporato in un sistema software senza controlli preventivi o verifiche puntuali su di esso.

Strategie di prevenzione e mitigazione

1. Monitoraggio costante e revisione degli output del LLM. Utilizzo di tecniche rivolte all'auto-consistenza o metodi di voto per filtrare testi incoerenti. Analizzare e confrontare le varie risposte fornite dal modello a un unico prompt può aiutare a valutare meglio la qualità e la coerenza degli output.
2. Controlli incrociati sul modello rispetto a fonti esterne certificate. Questo livello supplementare di validazione può aiutare ad assicurare che le informazioni prodotte dal modello siano accurate e affidabili.
3. Migliorare il modello con metodi di taratura fine (fine-tuning) o incorporazione

(embedding) per migliorare la qualità degli output. Modelli generici pre-addestrati possono generare informazioni meno accurate con maggiore probabilità rispetto a modelli addestrati su un dominio specifico. Tecniche come l'ingegnerizzazione dei prompt, ottimizzazione efficiente dei parametri (PET), ottimizzazione dell'intero modello, e prompting basato su catena di pensiero (chain of thoughts prompting) possono ottimizzarne le prestazioni.
4. Implementazione di meccanismi di validazione automatica che possono effettuare controlli comparativi sugli output generati, rispetto a fatti o dati consolidati. Questo può fornire un livello di sicurezza aggiuntiva e mitigare i rischi associati alle allucinazioni.
5. Suddivisione di compiti complessi in singole azioni parziali più gestibili, e assegnazione ad agenti specializzati. Questo non solo aiuta a gestire la complessità, ma anche a ridurre le probabilità di allucinazione essendo ogni agente responsabile di obiettivi più limitati e controllabili.
6. Comunicare in modo trasparente i rischi e le limitazioni associate con l'utilizzo degli LLM. Ciò include potenziale inesattezza delle informazioni, e altri rischi connessi. Una chiara comunicazione dei rischi prepara gli utenti a potenziali problemi e li aiuta a prendere decisioni in modo consapevole.
7. Costruire API e interfacce utente che incoraggino un utilizzo responsabile degli LLM. Questo include misure come filtri di contenuti, avvisi di potenziale inesattezza e chiara indicazione dei materiali generati dall'AI.
8. Quando si utilizzano LLM in ambienti di sviluppo, è necessario stabilire criteri di codifica sicura e linee guida che prevengano l'introduzione di potenziali vulnerabilità.

Esempi di scenari di attacco

1. Una testata giornalistica si affida eccessivamente a un LLM per generare i propri articoli e notizie. Un malintenzionato può sfruttare questa eccessiva dipendenza, iniettando nel LLM informazioni fuorvianti, che favoriscono la diffusione di notizie false o mirate.
2. L'intelligenza artificiale plagia involontariamente contenuti, causando problemi di proprietà intellettuale che minano la credibilità dell'organizzazione coinvolta.
3. Un team di sviluppo software utilizza un LLM per accelerare la scrittura di codice. Un'eccessiva dipendenza dai suggerimenti dell'AI introduce vulnerabilità nelle applicazioni generate a causa di parametri di default insicuro o raccomandazioni che non rispettano le pratiche di sviluppo sicuro.
4. Un'azienda che si occupa di sviluppo software utilizza un LLM per fornire assistenza agli sviluppatori. Il modello suggerisce una libreria o un pacchetto inesistenti e uno sviluppatore, affidandosi all'AI, integra senza saperlo un pacchetto malevolo nel software venduto dall'azienda. Questo esempio sottolinea l'importanza dei controlli incrociati sui suggerimenti del modello, specialmente in relazione a codice o librerie di terze parti.

Riferimenti e link (Inglese)

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: Furto del Modello

Descrizione

Questa voce si riferisce all'accesso non autorizzato e all'esfiltrazione di modelli LLM da parte di attori malintenzionati o Gruppi A.P.T. (Advanced Persistent Threat). Ciò si verifica quando i modelli LLM proprietari (rappresentando una proprietà intellettuale di valore) vengono compromessi, rubati fisicamente, copiati o se ne estraggono pesi e parametri per replicarne il funzionamento. Le conseguenze del furto di modelli LLM può includere perdite finanziarie, danni alla reputazione del marchio, erosione del vantaggio competitivo, uso non autorizzato del modello o accesso non autorizzato a informazioni sensibili contenute nel modello.

La crescente diffusione e potenza degli LLM, rende il furto di LLM una vulnerabilità dall'impatto significativo. Organizzazioni e ricercatori devono dare priorità a misure di sicurezza adeguate volte a proteggere i loro modelli LLM, garantendo la riservatezza e l'integrità della loro proprietà intellettuale. Per mitigare i rischi associati al furto del modello LLM e salvaguardare gli interessi di chi gli utilizza, è cruciale adottare un sistema di sicurezza integrato (es. sistema di gestione della sicurezza delle informazioni o altre pratiche equivalenti) che includa controlli di accesso, crittografia e monitoraggio continuo.

Esempi comuni di vulnerabilità

1. Un attaccante sfrutta una vulnerabilità nell'infrastruttura di un'azienda per accedere senza autorizzazione al loro archivio (repository) di modelli LLM a causa di configurazioni errate nella sicurezza della rete o delle applicazioni.
2. Uno scenario di minaccia interna in cui un dipendente insoddisfatto divulga modelli o artefatti correlati ai modelli.
3. Un attaccante interroga l'API del modello, utilizzando input realizzati ad hoc e tecniche di iniezione di prompt, per ottenere abbastanza output (risposte del modello) che permettano di ricreare successivamente un modello ombra (shadow model).
4. Un attaccante riesce a eludere i meccanismi di filtraggio degli input del LLM per effettuare un attacco laterale (side-channel) riuscendo così a raccogliere le informazioni sui pesi e sull'architettura del modello, per trasferirli verso una risorsa remota.
5. Il vettore d'attacco per l'estrazione del modello implica l'interrogazione del LLM con un gran numero di prompt, su un argomento specifico. Gli output del LLM possono a quel punto essere utilizzati per affinare un altro modello. Tuttavia, ci sono alcune considerazioni da fare su questo attacco:

- L'aggressore deve generare un gran numero di prompt mirati. Se i prompt non sono abbastanza specifici, gli output del LLM risulteranno essere di scarso valore.
- Gli output degli LLM possono talvolta contenere risposte considerate "allucinazioni", il che significa che l'attaccante potrebbe non essere in grado di estrarre l'intero modello, poiché alcuni output possono essere insensati.
- Non è possibile replicare un LLM al 100% tramite questo metodo di estrazione. Tuttavia l'attaccante sarà in grado di ottenere una replica parziale del modello.

6. Il vettore di attacco per creare una "replica funzionale del modello" si avvale dell'uso del modello target tramite prompt volti a generare dati di addestramento sintetici (un approccio chiamato "auto-istruzione"), da utilizzare in seguito per affinare un altro modello fondamentale (foundational model) in modo da produrre un'imitazione funzionale del modello target. Questa tecnica supera le limitazioni dell'estrazione tradizionale basata su query, illustrata nell'esempio precedente (5), ed è stata applicata con successo nella ricerca sull'uso di un LLM per addestrare un altro LLM. Sebbene in ambito di ricerca la replica del modello non è considerata malevola, lo stesso approccio potrebbe essere utilizzato da un attaccante per duplicare un modello proprietario con API pubblica.

L'impiego di un modello rubato, come un modello ombra (shadow model), può essere finalizzato a orchestrare attacchi informatici, incluso l'accesso non autorizzato a informazioni sensibili contenute nel modello stesso, o per condurre esperimenti su di esso in modo non rilevabile, con input avversi, per predisporre ulteriori iniezioni di prompt (prompt injection) più mirate.

Strategie di prevenzione e mitigazione

1. Implementare controlli di accesso robusti (ad esempio, RBAC e minimizzazione dei privilegi) accanto a meccanismi di autenticazione forti per limitare l'accesso non autorizzato ai repository di modelli LLM e agli ambienti di addestramento (pre-training e training).
   - Questo è particolarmente importante per i primi tre esempi esaminati, che possono causare questa vulnerabilità in caso di minacce interne (insider threat), configurazione errata e/o controlli di sicurezza deboli, relativi all'infrastruttura che ospita modelli, pesi e architettura del LLM, in cui un attore malintenzionato potrebbe guadagnare l'accesso dall'interno, o dall'esterno.
   - La gestione dei fornitori, il tracciamento, la verifica e le vulnerabilità delle dipendenze rappresentano argomenti di interesse fondamentali per prevenire lo sfruttamento degli attacchi alla catena di approvvigionamento.
2. Limitare l'accesso del LLM alle risorse di rete, ai servizi interni e alle API.
   - Questa pratica è efficace per tutti gli esempi di attacchi citati, proteggendo

dai rischi e dalle minacce interne (insider threat), e limitando allo stesso tempo ciò a cui l'applicazione LLM "ha accesso", rappresentando dunque un meccanismo per prevenire o interrompere attacchi laterali (side-channel attacks).

3. Utilizzare un Inventario o Registro centralizzato per i modelli di Machine Learning utilizzati in produzione. Avere un registro di modelli centralizzato aiuta a prevenire accessi non autorizzati ai modelli di ML attraverso controlli di accesso, autenticazione e grazie alla capacità di monitoraggio/registrazione, elementi fondamentali per la governance di questi processi. Avere un repository centralizzato offre anche il vantaggio di facilitare la raccolta di informazioni relative agli algoritmi utilizzati dai modelli per scopi di conformità, valutazione dei rischi e loro mitigazione.
4. Monitorare e verificare regolarmente i registri (log) di accesso e le attività relative agli archivi dei modelli LLM, per rilevare e rispondere tempestivamente a qualsiasi comportamento sospetto o non autorizzato.
5. Automatizzare l'implementazione delle cosiddette MLOps (Gestione Operativa dei Modelli di Machine Learning) attraverso flussi di lavoro dedicati alla governance, il tracciamento e l'approvazione, per rafforzare i controlli di accesso e rilascio all'interno dell'infrastruttura.
6. Implementare controlli e strategie di mitigazione per ridurre e/o contenere il rischio di attacchi indiretti (side channel) causati da tecniche di iniezione di prompt (prompt injection).
7. Limitare la frequenza delle chiamate API dove applicabile e/o utilizzare filtri per minimizzare il rischio di esfiltrazione dei dati dalle applicazioni LLM, o implementare tecniche (ad es. sistemi di Data Loss Prevention) per rilevare attività di estrazione anche da fonti alternative.
8. Implementare tecniche di addestramento avversariale mirato alla resilienza per aiutare il modello a rilevare query di estrazione e rafforzare inoltre le misure di sicurezza fisica.
9. Implementare un framework di marcatura modale e/o temporale (watermarking) nelle fasi di integrazione (input) e monitoraggio (output, trasformazione), del ciclo di vita di un LLM.

Esempi di scenari di attacco

1. Un attaccante abusa una vulnerabilità nell'infrastruttura di un'azienda per accedere senza autorizzazione al loro repository di modelli LLM. L'attaccante procede quindi all'esfiltrazione di modelli LLM di valore e li utilizza per lanciare un servizio concorrente di elaborazione del linguaggio o estrarre forzatamente informazioni sensibili, causando gravi danni finanziari all'azienda proprietaria del modello.
2. Un dipendente insoddisfatto divulga modelli o artefatti correlati. L'esposizione pubblica che questo scenario rappresenta aumenta la conoscenza a disposizione di possibili attaccanti, consentendo attacchi avversariali di tipo "gray box"

(conoscenza parziale) o, in alternativa, consentendo il furto diretto della proprietà intellettuale esposta.
3. Un attaccante interroga l'API con input creati ad hoc e raccoglie abbastanza output per creare un modello ombra (shadow).
4. Lacune nei controlli di sicurezza della catena di fornitura (supply chain) causano perdite di dati relativi a informazioni proprietarie del o sul modello.
5. Un attaccante elude le tecniche di filtraggio degli input e le istruzioni iniziali del LLM, per eseguire un attacco indiretto/laterale e recuperare informazioni sul modello caricandole su una risorsa remota sotto il suo controllo.

Riferimenti e link (Inglese)

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