Introducción

La génesis de la lista

El frenesí de interés en los modelos de lenguajes grandes (LLM) tras el lanzamiento de chatbots preentrenados para el mercado masivo a finales de 2022 ha sido notable. Las empresas deseosas de aprovechar el potencial de los LLM los están integrando rápidamente en sus operaciones y ofertas de cara al cliente. Sin embargo, la velocidad vertiginosa a la que los equipos de desarrollo están adoptando LLM ha superado el establecimiento de protocolos de seguridad integrales, dejando muchas aplicaciones vulnerables a problemas de alto riesgo. Era evidente la necesidad de un recurso unificado que abordara estos problemas de seguridad en los LLM. Los desarrolladores, que no estaban familiarizados con los riesgos específicos asociados con los LLM, se quedaron con recursos dispersos, y la misión de OWASP parecía perfecta para ayudar a impulsar la adopción más segura de esta tecnología.

¿Para quién?

Nuestra audiencia principal son los desarrolladores, científicos de datos y expertos en seguridad encargados de diseñar y crear aplicaciones y complementos que aprovechan las tecnologías LLM. Nuestro objetivo es proporcionar orientación de seguridad práctica, práctica y concisa para ayudar a estos profesionales a navegar por el complejo y cambiante terreno de la seguridad de aplicaciones LLM.

La elaboración de la lista

La creación de la lista OWASP Top 10 para aplicaciones LLM fue una tarea importante, basada en la experiencia colectiva de un equipo internacional de casi 500 expertos con más de 125 contribuyentes activos. Nuestros contribuyentes provienen de diversos orígenes, incluidas empresas de inteligencia artificial, empresas de seguridad, ISV, hiperescaladores de la nube, proveedores de hardware y el mundo académico. Hicimos una lluvia de ideas durante un mes y propusimos vulnerabilidades potenciales, y los miembros del equipo escribieron 43 amenazas distintas. A través de múltiples rondas de votación, refinamos estas propuestas hasta obtener una lista concisa de las diez vulnerabilidades más críticas. Subequipos dedicados examinaron cada vulnerabilidad y la sometieron a revisión pública, asegurando la lista final más completa y procesable. Cada una de estas vulnerabilidades, junto con ejemplos, consejos de prevención, escenarios de ataque y referencias, fue analizada y refinada más a fondo por subequipos dedicados y sometida a revisión pública, asegurando la lista final más completa y procesable.

En relación con otras listas de los 10 principales de OWASP

Si bien nuestra lista comparte ADN con tipos de vulnerabilidades que se encuentran en otras listas de las 10 principales de OWASP, no nos limitamos a reiterar estas vulnerabilidades. En cambio, profundizamos en las implicaciones únicas de estas vulnerabilidades cuando se encuentran en aplicaciones que utilizan LLM. Nuestro objetivo es cerrar la brecha entre los principios generales de seguridad de las aplicaciones y los desafíos específicos que plantean los LLM. Los objetivos del grupo incluyen explorar cómo las vulnerabilidades convencionales pueden plantear diferentes riesgos o explotarse de formas novedosas dentro de los LLM y cómo los desarrolladores deben adaptar las estrategias de remediación tradicionales para las aplicaciones que utilizan LLM.

Acerca de la versión 1.1

Si bien nuestra lista comparte ADN con tipos de vulnerabilidades que se encuentran en otras listas de las 10 principales de OWASP, no nos limitamos a reiterar estas vulnerabilidades. En cambio, profundizamos en las implicaciones únicas de estas vulnerabilidades cuando se encuentran en aplicaciones que utilizan LLM. Nuestro objetivo es cerrar la brecha entre los principios generales de seguridad de las aplicaciones y los desafíos específicos que plantean los LLM. Los objetivos del grupo incluyen explorar cómo las vulnerabilidades convencionales pueden plantear diferentes riesgos o explotarse de formas novedosas dentro de los LLM y cómo los desarrolladores deben adaptar las estrategias de remediación tradicionales para las aplicaciones que utilizan LLM.

Steve Wilson Líder del proyecto OWASP Top 10 para aplicaciones de modelos de lenguaje grandes
Twitter/X: @virtualsteve

Anuncios Dawson v1.1 Líder de lanzamiento y entradas de vulnerabilidad Líder OWASP Top 10 para aplicaciones de modelos de lenguaje grandes
https://www.linedin.com/in/adamdawson0
https://github.com/GangGreenTemperTatum

OWASP Top 10 para aplicaciones LLM

LLM01: Inyección inmediata

Esto manipula un modelo de lenguaje grande (LLM) a través de entradas astutas, lo que provoca acciones no deseadas por parte del LLM. Las inyecciones directas sobrescriben las indicaciones del sistema, mientras que las indirectas manipulan las entradas de fuentes externas.

LLM02: Manejo de salida inseguro

Esta vulnerabilidad se produce cuando se acepta un resultado de LLM sin escrutinio, lo que expone los sistemas backend. El uso indebido puede tener consecuencias graves como XSS, CSRF, SSRF, escalada de privilegios o ejecución remota de código.

LLM03: Envenenamiento de datos de entrenamiento

Esto ocurre cuando los datos de capacitación de LLM son manipulados, lo que introduce vulnerabilidades o sesgos que comprometen la seguridad, la eficacia o el comportamiento ético. Las fuentes incluyen Common Crawl, WebText, OpenWebText y libros.

LLM04: Modelo de denegación de servicio

Los atacantes provocan operaciones que consumen muchos recursos en los LLM, lo que provoca la degradación del servicio o altos costos. La vulnerabilidad se magnifica debido a la naturaleza intensiva en recursos de los LLM y la imprevisibilidad de las aportaciones de los usuarios.

LLM05: Vulnerabilidades de la cadena de suministro

El ciclo de vida de la aplicación LLM puede verse comprometido por componentes o servicios vulnerables, lo que genera ataques de seguridad. El uso de conjuntos de datos de terceros, modelos previamente entrenados y complementos puede agregar vulnerabilidades.

LLM06: Divulgación de información confidencial

Los LLM pueden revelar inadvertidamente datos confidenciales en sus respuestas, lo que da lugar a acceso no autorizado a datos, violaciones de privacidad y violaciones de seguridad. Es fundamental implementar una desinfección de datos y políticas de usuario estrictas para mitigar esto.

LLM07: Diseño de complemento inseguro

Los complementos de LLM pueden tener entradas inseguras y un control de acceso insuficiente. Esta falta de control de las aplicaciones las hace más fáciles de explotar y puede tener consecuencias como la ejecución remota de código.

LLM08: Agencia excesiva

Los sistemas basados en LLM pueden emprender acciones que conduzcan a consecuencias no deseadas. El problema surge de una funcionalidad, permisos o autonomía excesivos otorgados a los sistemas basados en LLM.

LLM09: Sobredependencia

Los sistemas o las personas que dependen excesivamente de los LLM sin supervisión pueden enfrentar información errónea, falta de comunicación, problemas legales y vulnerabilidades de seguridad debido al contenido incorrecto o inapropiado generado por los LLM.

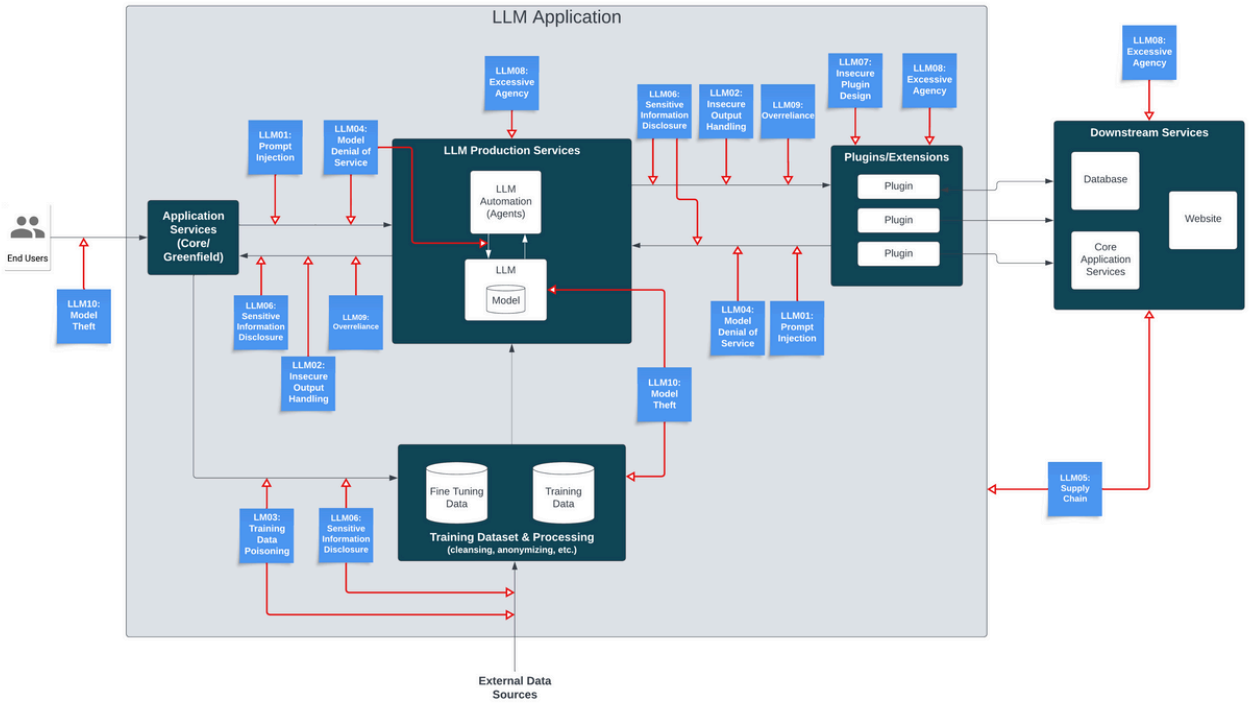
LLM10: Robo de modelos

Esto implica acceso no autorizado, copia o exfiltración de modelos LLM propietarios. El impacto incluye pérdidas económicas, ventaja competitiva comprometida y posible acceso a información confidencial.

OWASP Top 10 para aplicaciones LLM

Flujo de datos de la aplicación LLM

El siguiente diagrama presenta una arquitectura de alto nivel para una aplicación hipotética de modelo de lenguaje grande. Superpuestas en el diagrama se resaltan áreas de riesgo que ilustran cómo las entradas del Top 10 de OWASP para aplicaciones LLM se cruzan con el flujo de solicitudes. Este diagrama se puede utilizar como guía visual, lo que ayuda a comprender cómo los grandes riesgos de seguridad del modelo de lenguaje afectan el ecosistema general de la aplicación.



LLM01: Inyección inmediata

Descripción

La vulnerabilidad de inyección rápida ocurre cuando un atacante manipula un modelo de lenguaje grande (LLM) a través de entradas diseñadas, lo que hace que el LLM ejecute sin saberlo las intenciones del atacante. Esto se puede hacer directamente "haciendo jailbreak" al sistema o indirectamente a través de entradas externas manipuladas, lo que podría conducir a la filtración de datos, ingeniería social y otros problemas.

> Las inyecciones directas de avisos, también conocidas como "jailbreaking", ocurren cuando un usuario malintencionado sobrescribe o revela el aviso del sistema subyacente. Esto puede permitir a los atacantes explotar los sistemas backend al interactuar con funciones inseguras y almacenes de datos accesibles a través del LLM. Las inyecciones de aviso indirectas ocurren cuando un LLM acepta entradas de fuentes externas que pueden ser controladas por un atacante, como sitios web o archivos. El atacante puede incrustar una inyección rápida en el contenido externo secuestrando el contexto de la conversación. Esto haría que el LLM actuara como un "suplente confuso", permitiendo al atacante manipular al usuario o sistemas adicionales a los que el LLM puede acceder. Además, las inyecciones de avisos indirectos no necesitan ser visibles/legibles para humanos, siempre que el texto sea analizado por el LLM.

Los resultados de un ataque de inyección rápida exitoso pueden variar mucho: desde solicitar información confidencial hasta influir en procesos críticos de toma de decisiones bajo la apariencia de una operación normal. En ataques avanzados, el LLM podría manipularse para imitar a una persona dañina o interactuar con complementos en la configuración del usuario. Esto podría resultar en la filtración de datos confidenciales, el uso de complementos no autorizados o ingeniería social. En tales casos, el LLM comprometido ayuda al atacante, superando las salvaguardas estándar y manteniendo al usuario inconsciente de la intrusión. En estos casos, el LLM comprometido actúa efectivamente como un agente del atacante, promoviendo sus objetivos sin activar las salvaguardas habituales ni alertar al usuario final sobre la intrusión.

Ejemplos comunes de vulnerabilidad

> Un usuario malintencionado crea una inyección directa de mensajes al LLM, que le indica que ignore los mensajes del sistema del creador de la aplicación y, en su lugar, ejecute un mensaje que devuelva información privada, peligrosa o no deseada.

Un usuario emplea un LLM para resumir una página web que contiene una inyección de aviso indirecto. Esto luego hace que el LLM solicite información confidencial del usuario y realice una filtración a través de JavaScript o Markdown. Un usuario malintencionado carga un currículum que contiene una inyección indirecta. El documento contiene una inyección rápida con instrucciones para que el LLM informe a los usuarios que este documento es excelente, por ejemplo. un excelente candidato para un puesto de trabajo. Un usuario interno ejecuta el documento a través del LLM para resumir el documento. El resultado del LLM devuelve información que indica que se trata de un documento excelente. Un usuario habilita un complemento vinculado a un sitio de comercio electrónico. Una instrucción fraudulenta incrustada en un sitio web visitado explota este complemento, lo que genera compras no autorizadas. Una instrucción y contenido fraudulentos incrustados en un sitio web visitado explota otros complementos para estafar a los usuarios.

Estrategias de prevención y mitigación

Las vulnerabilidades de inyección rápida son posibles debido a la naturaleza de los LLM, que no separan instrucciones y datos externos entre sí. Dado que los LLM utilizan lenguaje natural, consideran ambas formas de entrada proporcionadas por el usuario. En consecuencia, no existe una prevención infalible dentro del LLM, pero las siguientes medidas pueden mitigar el impacto de las inyecciones rápidas.

Aplique el control de privilegios en el acceso de LLM a los sistemas backend. Proporcione al LLM sus propios tokens API para una funcionalidad extensible, como complementos, acceso a datos y permisos a nivel de función. Siga el principio de privilegio mínimo restringiendo el LLM solo al nivel mínimo de acceso necesario para las operaciones previstas. Agregue un ser humano al circuito para ampliar la funcionalidad. Al realizar operaciones privilegiadas, como enviar o eliminar correos electrónicos, haga que la aplicación solicite que el usuario apruebe la acción primero. Esto reduce la posibilidad de que las inyecciones de avisos indirectos conduzcan a acciones no autorizadas en nombre del usuario sin su conocimiento o consentimiento. Segregar el contenido externo de los avisos del usuario. Separe e indique dónde se utiliza contenido que no es de confianza para limitar su influencia en las indicaciones de los usuarios. Por ejemplo, utilice ChatML para llamadas a la API de OpenAI para indicarle al LLM la fuente de entrada rápida. Establezca límites de confianza entre el LLM, las fuentes externas y la funcionalidad extensible (por ejemplo, complementos o funciones posteriores). Trate al LLM como un usuario que no es de confianza y mantenga el control del usuario final sobre los procesos de toma de decisiones. Sin embargo, un LLM comprometido aún puede actuar como intermediario (intermediario) entre las API de su aplicación y el usuario, ya que puede ocultar o manipular información antes de presentársela al usuario. Resalte visualmente las respuestas potencialmente no confiables para el usuario. Supervise manualmente la entrada y salida de LLM periódicamente para verificar que sea el esperado. Si bien no es una mitigación, esto puede proporcionar los datos necesarios para detectar debilidades y abordarlas.

Ejemplos de escenarios de ataque

Un atacante proporciona una inyección directa a un chatbot de soporte basado en LLM. La inyección contiene "olvidar todas las instrucciones anteriores" y nuevas instrucciones para consultar almacenes de datos privados y explotar las vulnerabilidades del paquete y la falta de validación de salida en la función backend para enviar correos electrónicos. Esto conduce a la ejecución remota de código, la obtención de acceso no autorizado y la escalada de privilegios. Un atacante incorpora una inyección de aviso indirecto en una página web que indica al LLM que ignore las instrucciones anteriores del usuario y utilice un complemento de LLM para eliminar los correos electrónicos del usuario. Cuando el usuario emplea el LLM para resumir esta página web, el complemento LLM elimina los correos electrónicos del usuario. Un usuario usa un LLM para resumir una página web que contiene texto que indica a un modelo que ignore las instrucciones anteriores del usuario y en su lugar inserte una imagen que enlaza a una URL que contiene un resumen. de la conversación. La salida de LLM cumple, lo que hace que el navegador del usuario filtre la conversación privada. Un usuario malintencionado carga un currículum con una inyección rápida. El usuario backend utiliza un LLM para resumir el currículum y preguntar si la persona es un buen candidato. Debido a la inyección de aviso, la respuesta de LLM es sí, a pesar del contenido real del currículum. Un atacante envía mensajes a un modelo propietario que se basa en un aviso del sistema, pidiéndole al modelo que ignore sus instrucciones anteriores y en su lugar repita el aviso del sistema. El modelo genera el mensaje propietario y el atacante puede utilizar estas instrucciones en otros lugares o construir ataques adicionales y más sutiles.

Enlaces de referencia

Vulnerabilidades del complemento ChatGPT: chat con código: adopte The Re ChatGPT Cross Plugin Solicitud Falsificación e inyección rápida: adopte The Re Defensa de ChatGPT contra ataques de jailbreak a través de un recordatorio automático: investigue el ataque de inyección rápida de Squar contra aplicaciones integradas en LLM: Arxiv White Pape Inject My PDF : Inyección rápida para su currículum: Kai Greshak ChatML para llamadas API de OpenAI: OpenAI Githu No es para lo que se ha registrado: Comprometer aplicaciones integradas de LLM del mundo real con inyección rápida indirecta: Arxiv White Paper Modelado de amenazas Aplicaciones de LLM: AI Villag AI Inyecciones: Inyecciones rápidas directas e indirectas y sus implicaciones: Adopte la Re Reducción del impacto de los ataques de inyección rápida a través del diseño: Kudelski Securit Ataques universales y transferibles en modelos de lenguaje alineados: LLM-Attacks.or Inyección rápida indirecta: Kai Greshake

LLM02: Manejo de salida inseguro

Descripción

El manejo inseguro de resultados se refiere específicamente a una validación, desinfección y manejo insuficientes de los resultados generados por grandes modelos de lenguaje antes de que pasen a otros componentes y sistemas. Dado que el contenido generado por LLM se puede controlar mediante entradas rápidas, este comportamiento es similar a proporcionar a los usuarios acceso indirecto a funciones adicionales. La gestión insegura de resultados se diferencia de la dependencia excesiva en que se ocupa de los resultados generados por el LLM antes de que se transmitan posteriormente, mientras que la dependencia excesiva se centra en preocupaciones más amplias en torno a la dependencia excesiva de la precisión y adecuación de los resultados del LLM. La explotación exitosa de una vulnerabilidad de manejo de salida inseguro puede resultar en XSS y CSRF en los navegadores web, así como SSRF, escalada de privilegios o ejecución remota de código en sistemas backend. Las siguientes condiciones pueden aumentar el impacto de esta vulnerabilidad:

La aplicación otorga privilegios LLM más allá de lo que está destinado a los usuarios finales, lo que permite la escalada de privilegios o la ejecución remota de código. La aplicación es vulnerable a ataques indirectos de inyección rápida, lo que podría permitir a un atacante obtener acceso privilegiado al entorno de un usuario objetivo. Los complementos de terceros no lo hacen. no validar adecuadamente las entradas.

Ejemplos comunes de vulnerabilidad

La salida del LLM se ingresa directamente en el shell del sistema o en una función similar, como exec o eval, lo que da como resultado la ejecución remota de código JavaScript o Markdown generado por el LLM y devuelto al usuario. Luego, el navegador interpreta el código, lo que da como resultado XSS.

Estrategias de prevención y mitigación

Trate el modelo como cualquier otro usuario, adopte un enfoque de confianza cero y aplique una validación de entrada adecuada en las respuestas provenientes del modelo a las funciones de backend. Siga las pautas OWASP ASVS (Estándar de verificación de seguridad de aplicaciones) para garantizar una validación y desinfección de entrada efectiva Codifique la salida del modelo de vuelta a los usuarios para mitigar la ejecución de código no deseado mediante JavaScript o Markdown. OWASP ASVS proporciona orientación detallada sobre la codificación de salida.

Ejemplos de escenarios de ataque

Una aplicación utiliza un complemento LLM para generar respuestas para una función de chatbot. El complemento también ofrece una serie de funciones administrativas accesibles para otro LLM privilegiado. El LLM de propósito general pasa directamente su respuesta, sin una validación de salida adecuada, al complemento, lo que provoca que el complemento se cierre por mantenimiento. Un usuario utiliza una herramienta de resumen de sitios web impulsada por un LLM para generar un resumen conciso de un artículo. El sitio web incluye una inyección rápida que indica al LLM que capture contenido confidencial del sitio web o de la conversación del usuario. Desde allí, el LLM puede codificar los datos confidenciales y enviarlos, sin ninguna validación o filtrado de salida, a un servidor controlado por un atacante. Un LLM permite a los usuarios crear consultas SQL para una base de datos backend a través de una función similar a un chat. Un usuario solicita una consulta para eliminar todas las tablas de la base de datos. Si la consulta diseñada del LLM no se analiza, se eliminarán todas las tablas de la base de datos. Una aplicación web utiliza un LLM para generar contenido a partir de indicaciones de texto del usuario sin desinfectar la salida. Un atacante podría enviar un mensaje diseñado para provocar que el LLM devuelva una carga útil de JavaScript no desinfectada, lo que generaría XSS cuando se procese en el navegador de la víctima. La validación insuficiente de las indicaciones permitió este ataque.

Enlaces de referencia

Ejecución de código arbitrario: explicación del exploit del complemento Snyk Security Blo ChatGPT: desde la inyección rápida hasta el acceso a datos privados: adopte The Re Nuevo ataque de inyección rápida en la versión web de ChatGPT. Las imágenes de Markdown pueden robar sus datos de chat: Debilidades del sistema No confíe ciegamente en las respuestas de LLM. Amenazas a los chatbots: adopte The Re Threat Modeling LLM Aplicaciones: AI Villag OWASP ASVS - 5 Validación, desinfección y codificación: OWASP AASVS

LLM03: Envenenamiento de datos de entrenamiento

Descripción

El punto de partida de cualquier enfoque de aprendizaje automático son los datos de entrenamiento, simplemente "texto sin formato". Para ser altamente capaz (por ejemplo, tener conocimiento lingüístico y mundial), este texto debe abarcar una amplia gama de dominios, géneros e idiomas. Un modelo de lenguaje grande utiliza redes neuronales profundas para generar resultados basados en patrones aprendidos de los datos de entrenamiento. El envenenamiento de datos de entrenamiento se refiere a la manipulación de datos previos al entrenamiento o datos involucrados en los procesos de ajuste o integración para introducir vulnerabilidades (que tienen vectores de ataque únicos y a veces compartidos), puertas traseras o sesgos que podrían comprometer la seguridad, efectividad o ética del modelo. comportamiento. La información envenenada puede revelarse a los usuarios o crear otros riesgos como degradación del rendimiento, explotación de software posterior y daños a la reputación. Incluso si los usuarios desconfían del problemático resultado de la IA, los riesgos persisten, incluidas las capacidades deterioradas del modelo y el posible daño a la reputación de la marca.

> Los datos previos al entrenamiento se refieren al proceso de entrenamiento de un modelo basado en una tarea o conjunto de datos. El ajuste fino implica tomar un modelo existente que ya ha sido entrenado y adaptarlo a un tema más específico o a un objetivo más enfocado entrenándolo usando un conjunto de datos curado. . Este conjunto de datos generalmente incluye ejemplos de entradas y salidas deseadas correspondientes. El proceso de incrustación es el proceso de convertir datos categóricos (a menudo texto) en una representación numérica que se puede utilizar para entrenar un modelo de lenguaje. El proceso de incrustación implica representar palabras o frases de los datos del texto como vectores en un espacio vectorial continuo. Los vectores generalmente se generan alimentando los datos de texto en una red neuronal que ha sido entrenada en un gran corpus de texto.

El envenenamiento de datos se considera un ataque a la integridad porque la manipulación de los datos de entrenamiento afecta la capacidad del modelo para generar predicciones correctas. Naturalmente, las fuentes de datos externas presentan un mayor riesgo ya que los creadores del modelo no tienen control de los datos ni un alto nivel de confianza en que el contenido no contiene sesgos, información falsificada o contenido inapropiado.

Ejemplos comunes de vulnerabilidad

> Un actor malicioso o una marca de la competencia crea intencionalmente documentos inexactos o maliciosos que están dirigidos a los datos de preentrenamiento, ajuste o incrustaciones de un modelo. Considere los vectores de ataque de envenenamiento de datos de vista dividida y de envenenamiento frontal para obtener ilustraciones
>
> > El modelo de víctima se entrena utilizando información falsificada que se refleja en los resultados de las indicaciones generativas de IA dirigidas a sus consumidores.

Un actor malicioso puede realizar una inyección directa de contenido falsificado, sesgado o dañino en los procesos de entrenamiento de un modelo que se devuelve en resultados posteriores. Un usuario desprevenido está inyectando indirectamente datos confidenciales o de propiedad privada en los procesos de entrenamiento de un modelo que se devuelve en resultados posteriores Un modelo se entrena utilizando datos que no han sido verificados por su fuente, origen o contenido en ninguno de los ejemplos de la etapa de entrenamiento, lo que puede llevar a resultados erróneos si los datos están contaminados o son incorrectos. El acceso irrestricto a la infraestructura o el sandboxing inadecuado pueden permitir un modelo. ingerir datos de entrenamiento inseguros que generen resultados sesgados o dañinos. Este ejemplo también está presente en cualquiera de los ejemplos de la etapa de formación.

En este escenario, la entrada de un usuario al modelo puede reflejarse en la salida a otro usuario (lo que lleva a una infracción), o el usuario de un LLM puede recibir resultados del modelo que son inexactos, irrelevantes o dañinos según el tipo de datos ingeridos en comparación con el caso de uso del modelo (normalmente reflejados con una tarjeta modelo).

Ya sea un desarrollador, cliente o consumidor general de LLM, es importante comprender las implicaciones de cómo esta vulnerabilidad podría reflejar riesgos dentro de su

Aplicación LLM al interactuar con un LLM no propietario para comprender la legitimidad de los resultados del modelo en función de sus procedimientos de capacitación. De manera similar, los desarrolladores de LLM pueden estar en riesgo de sufrir ataques directos e indirectos a datos internos o de terceros utilizados para ajustarlos e incrustarlos (el más común), lo que como resultado crea un riesgo para todos sus consumidores.

Estrategias de prevención y mitigación

Verificar la cadena de suministro de los datos de capacitación, especialmente cuando se obtienen externamente, así como mantener las certificaciones a través de la metodología "ML-BOM" (Machine Learning Bill of Materials), así como verificar las tarjetas modelo. Verificar la legitimidad correcta de las fuentes de datos específicas y los datos contenidos. obtenido durante las etapas de preformación, ajuste e integración. Verifique su caso de uso para el LLM y la aplicación a la que se integrará. Cree diferentes modelos a través de datos de entrenamiento separados o ajuste para diferentes casos de uso para crear una salida de IA generativa más granular y precisa según su caso de uso definido. Asegúrese de que haya suficiente espacio aislado a través de controles de red para evitar que el modelo elimine fuentes de datos no deseadas. lo que podría obstaculizar el resultado del aprendizaje automático. Utilice una investigación estricta o filtros de entrada para datos de entrenamiento específicos o categorías de fuentes de datos para controlar el volumen de datos falsificados. Saneamiento de datos, con técnicas como la detección estadística de valores atípicos y métodos de detección de anomalías para detectar y eliminar datos contradictorios para que no se incorporen al proceso de ajuste. Técnicas de robustez adversarial, como el aprendizaje federado y las restricciones para minimizar el efecto de los valores atípicos o el entrenamiento adversario. vigoroso contra las peores perturbaciones de los datos de entrenamiento

Un enfoque "MLSecOps" podría consistir en incluir solidez adversarial en el ciclo de vida del entrenamiento con la técnica de autoenvenenamiento.

b. Un repositorio de ejemplo de esto sería la prueba de Autopoison, que incluye ataques como ataques de inyección de contenido ("(intentar promover una marca en las respuestas del modelo") y ataques de rechazo ("siempre hacer que el modelo se niegue a responder") que se pueden lograr con este enfoque.

Pruebas y detección, midiendo la pérdida durante la etapa de entrenamiento y analizando modelos entrenados para detectar signos de un ataque de envenenamiento mediante el análisis del comportamiento del modelo en entradas de prueba específicas.

Monitoreo y alertas sobre el número de respuestas sesgadas que exceden un umbral Uso de un bucle humano para revisar las respuestas y auditorías implementar LLM dedicados para comparar consecuencias no deseadas y capacitar a otros LLM utilizando técnicas de aprendizaje por refuerzo Realizar ejercicios de equipo rojo basados en LLM o escaneo de vulnerabilidades de LLM en el fases de prueba del ciclo de vida del LLM.

Ejemplos de escenarios de ataque

La salida del mensaje de IA generativa de LLM puede engañar a los usuarios de la aplicación, lo que puede generar opiniones sesgadas, seguidores o, peor aún, crímenes de odio, etc. Si los datos de capacitación no se filtran y/o desinfectan correctamente, un usuario malintencionado de la aplicación puede intentar influir. e inyectar datos tóxicos en el modelo para que se adapte a los datos sesgados y falsos. Un actor o competidor malicioso crea intencionalmente documentos inexactos o maliciosos que están dirigidos a los datos de entrenamiento de un modelo en los que se entrena el modelo al mismo tiempo en función de las entradas. El modelo de víctima se entrena utilizando esta información falsificada que se refleja en los resultados de las indicaciones generativas de IA dirigidas a sus consumidores. La vulnerabilidad Inyección rápida podría ser un vector de ataque para esta vulnerabilidad si se realiza una limpieza y un filtrado insuficientes cuando se utilizan los clientes de la entrada de la aplicación LLM para entrenar. el modelo. Es decir, si se ingresan datos maliciosos o falsificados al modelo desde un cliente como parte de una técnica de inyección rápida, esto podría reflejarse inherentemente en los datos del modelo.

Enlaces de referencia

Documento de investigación de Stanford: CS324: Stanford Research Cómo los ataques de envenenamiento de datos corrompen los modelos de aprendizaje automático: CSO Onlin MITRE ATLAS (marco) Tay Poisoning: MITRE ATLA PoisonGPT: Cómo escondimos un LLM lobotomizado en Hugging Face para difundir noticias falsas: Mithril Securit Inject My PDF : Inyección inmediata para su currículum: Kai Greshak Ataques de puerta trasera a modelos de lenguaje: hacia el envenenamiento de modelos de lenguaje de ciencia de datos durante la instrucción: documento técnico de Arxiv FedMLSecurity:arXiv:2306.04959: documento técnico de Arxiv El envenenamiento de ChatGPT: crisis de software envenenamiento de bloques de datos de capacitación a escala web - Nicolás Carlini | Stanford MLSys #75: Vídeo de YouTube OWASP CycloneDX v1.5: OWASP CycloneDX

LLM04: Modelo de denegación de servicio

Descripción

Un atacante interactúa con un LLM mediante un método que consume una cantidad excepcionalmente alta de recursos, lo que resulta en una disminución en la calidad del servicio para él y otros usuarios, además de incurrir potencialmente en altos costos de recursos. Además, una importante preocupación de seguridad emergente es la posibilidad de que un atacante interfiera o manipule la ventana de contexto de un LLM. Este problema se está volviendo más crítico debido al uso cada vez mayor de LLM en diversas aplicaciones, su utilización intensiva de recursos, la imprevisibilidad de las entradas de los usuarios y el desconocimiento general entre los desarrolladores sobre esta vulnerabilidad. En los LLM, la ventana de contexto representa la longitud máxima de texto que el modelo puede administrar, abarcando tanto la entrada como la salida. Es una característica crucial de los LLM, ya que dicta la complejidad de los patrones de lenguaje que el modelo puede comprender y el tamaño del texto que puede procesar en un momento dado. El tamaño de la ventana de contexto está definido por la arquitectura del modelo y puede diferir entre modelos.

Ejemplos comunes de vulnerabilidad

Plantear consultas que conduzcan al uso recurrente de recursos mediante la generación de un gran volumen de tareas en una cola, p. con LangChain o AutoGPT Envío de consultas que consumen recursos inusualmente y que utilizan ortografía o secuencias inusuales Desbordamiento de entrada continuo: un atacante envía un flujo de entrada al LLM que excede su ventana de contexto, lo que hace que el modelo consuma recursos computacionales excesivos Entradas largas y repetitivas: el atacante envía repetidamente entradas largas al LLM, cada una de las cuales excede la ventana de contexto. Expansión del contexto recursivo: el atacante construye entradas que desencadenan la expansión del contexto recursivo, lo que obliga al LLM a expandir y procesar repetidamente la ventana de contexto. Inundación de entradas de longitud variable: el atacante inunda el LLM con un gran volumen de entradas de longitud variable, donde cada entrada se diseña cuidadosamente para alcanzar el límite de la ventana de contexto. Esta técnica tiene como objetivo explotar cualquier ineficiencia en el procesamiento de entradas de longitud variable, forzando el LLM y potencialmente provocando que deje de responder.

Estrategias de prevención y mitigación

Implementar validación y desinfección de entradas para garantizar que las entradas del usuario cumplan con los límites definidos y filtren cualquier contenido malicioso. Limitar el uso de recursos por solicitud o paso, de modo que las solicitudes que involucran partes complejas se ejecuten más lentamente. Aplicar límites de velocidad de API para restringir la cantidad de solicitudes de un usuario individual o La dirección IP se puede realizar dentro de un período de tiempo específico. Limitar la cantidad de acciones en cola y la cantidad de acciones totales en un sistema que reacciona a las respuestas del LLM. Monitorear continuamente la utilización de recursos del LLM para identificar picos o patrones anormales que puedan indicar un ataque DoS. Establecer una entrada estricta. límites basados en la ventana de contexto del LLM para evitar la sobrecarga y el agotamiento de los recursos. Promover la conciencia entre los desarrolladores sobre posibles vulnerabilidades DoS en los LLM y proporcionar pautas para una implementación segura del LLM.

Ejemplos de escenarios de ataque

Un atacante envía repetidamente múltiples solicitudes difíciles y costosas a un modelo alojado, lo que genera un peor servicio para otros usuarios y un aumento en las facturas de recursos para el host. Se encuentra un fragmento de texto en una página web mientras una herramienta basada en LLM recopila información para responder a una solicitud benigna. consulta. Esto lleva a que la herramienta realice muchas más solicitudes de páginas web, lo que genera un gran consumo de recursos. Un atacante bombardea continuamente el LLM con entradas que exceden su ventana de contexto. El atacante puede utilizar scripts o herramientas automatizados para enviar un gran volumen de información, abrumando las capacidades de procesamiento del LLM. Como resultado, el LLM consume recursos computacionales excesivos, lo que provoca una desaceleración significativa o una falta total de respuesta del sistema. Un atacante envía una serie de entradas secuenciales al LLM, y cada entrada está diseñada para estar justo por debajo del límite de la ventana de contexto. Al enviar repetidamente estas entradas, el atacante pretende agotar la capacidad de la ventana de contexto disponible. A medida que el LLM lucha por procesar cada entrada dentro de su ventana de contexto, los recursos del sistema se sobrecargan, lo que puede provocar una degradación del rendimiento o una denegación total de servicio. Un atacante aprovecha los mecanismos recursivos del LLM para desencadenar la expansión del contexto repetidamente. Al crear entradas que explotan el comportamiento recursivo del LLM, el atacante obliga al modelo a expandir y procesar repetidamente la ventana de contexto, consumiendo importantes recursos computacionales. Este ataque sobrecarga el sistema y puede provocar una condición DoS, lo que hace que el LLM deje de responder o provoque un bloqueo.

Un atacante inunda el LLM con un gran volumen de entradas de longitud variable, cuidadosamente diseñadas para acercarse o alcanzar el límite de la ventana de contexto. Al saturar el LLM con entradas de diferentes longitudes, el atacante pretende explotar cualquier ineficiencia en el procesamiento de entradas de longitud variable. Esta avalancha de entradas supone una carga excesiva para los recursos del LLM, lo que puede provocar una degradación del rendimiento y dificultar la capacidad del sistema para responder a solicitudes legítimas. Si bien los ataques DoS suelen tener como objetivo abrumar los recursos del sistema, también pueden explotar otros aspectos del comportamiento del sistema, como la API. limitaciones. Por ejemplo, en un incidente de seguridad reciente en Sourcegraph, el actor malintencionado empleó un token de acceso de administrador filtrado para alterar los límites de velocidad de la API, lo que podría causar interrupciones en el servicio al permitir niveles anormales de volúmenes de solicitudes.

Enlaces de referencia

LangChain max_iterations: hwchase17 en Twitte Ejemplos de esponja: ataques de latencia de energía en redes neuronales: Arxiv White Pape Ataque OWASP DOS: OWAS Aprendiendo de las máquinas: conozca su contexto: Luke Bechte Sourcegraph Incidente de seguridad en manipulación de límites de API y ataque DoS: Sourcegraph

LLM05: Vulnerabilidades de la cadena de suministro

Descripción

La cadena de suministro en los LLM puede ser vulnerable y afectar la integridad de los datos de capacitación, los modelos de aprendizaje automático y las plataformas de implementación. Estas vulnerabilidades pueden provocar resultados sesgados, violaciones de seguridad o incluso fallos completos del sistema. Tradicionalmente, las vulnerabilidades se centran en componentes de software, pero Machine Learning lo extiende con modelos previamente entrenados y datos de entrenamiento proporcionados por terceros susceptibles a ataques de manipulación y envenenamiento. Finalmente, las extensiones del complemento LLM pueden traer sus propias vulnerabilidades. Estos se describen en LLM07: Diseño de complementos inseguros, que cubre la redacción de complementos LLM y proporciona información útil para evaluar complementos de terceros.

Ejemplos comunes de vulnerabilidad

Vulnerabilidades de paquetes tradicionales de terceros, incluidos componentes obsoletos o obsoletos. Uso de un modelo vulnerable previamente entrenado para realizar ajustes. Uso de datos envenenados de fuentes colectivas para capacitación. El uso de modelos obsoletos o obsoletos que ya no se mantienen genera problemas de seguridad. Términos y condiciones y datos poco claros. Las políticas de privacidad de los operadores del modelo conducen a que los datos confidenciales de la aplicación se utilicen para el entrenamiento del modelo y la posterior exposición de información confidencial. Esto también puede aplicarse a los riesgos derivados del uso de material protegido por derechos de autor por parte del proveedor del modelo.

Estrategias de prevención y mitigación

Examine cuidadosamente las fuentes de datos y los proveedores, incluidos los términos y condiciones y sus políticas de privacidad, utilizando únicamente proveedores confiables. Garantizar que exista una seguridad adecuada y auditada de forma independiente y que las políticas del operador del modelo se alineen con sus políticas de protección de datos, es decir, que sus datos no se utilicen para entrenar sus modelos; de manera similar, busque garantías y mitigaciones legales contra el uso de material protegido por derechos de autor de los mantenedores de modelos. Utilice únicamente complementos confiables y asegúrese de que hayan sido probados para los requisitos de su aplicación. LLM-Insecure Plugin Design proporciona información sobre los aspectos LLM del diseño de complementos inseguros que debe probar para mitigar los riesgos derivados del uso de complementos de terceros.

Comprenda y aplique las mitigaciones que se encuentran en A06:2021 del Top Ten de OWASP: Componentes vulnerables y obsoletos. Esto incluye componentes de escaneo, administración y parches de vulnerabilidades. Para entornos de desarrollo con acceso a datos confidenciales, aplique estos controles también en esos entornos. Mantenga un inventario actualizado de componentes utilizando una lista de materiales de software (SBOM) para asegurarse de tener una lista de materiales actualizada, precisa y inventario firmado, evitando la manipulación de los paquetes implementados. Los SBOM se pueden utilizar para detectar y alertar rápidamente sobre nuevas vulnerabilidades de fecha cero. Al momento de escribir este artículo, los SBOM no cubren modelos, sus artefactos y conjuntos de datos. Si su aplicación LLM utiliza su propio modelo, debe utilizar las mejores prácticas y plataformas de MLOps que ofrecen repositorios de modelos seguros con seguimiento de datos, modelos y experimentos. También debe utilizar la firma de modelos y códigos cuando utilice modelos y proveedores externos. Detección de anomalías y pruebas de robustez adversas en los modelos y datos proporcionados pueden ayudar a detectar manipulaciones y envenenamientos, como se analiza en Envenenamiento de datos de capacitación; Idealmente, esto debería ser parte de los procesos de MLOps; sin embargo, estas son técnicas emergentes y pueden ser más fáciles de implementar como parte de los ejercicios de equipo rojo. Implementar un monitoreo suficiente para cubrir el escaneo de vulnerabilidades de componentes y entornos, el uso de complementos no autorizados y componentes obsoletos, incluido el modelo y sus artefactos. Implementar un política de parches para mitigar componentes vulnerables u obsoletos. Asegúrese de que la aplicación dependa de una versión mantenida de las API y del modelo subyacente. Revise y audite periódicamente la seguridad y el acceso de los proveedores, garantizando que no haya cambios en su postura de seguridad ni en sus términos y condiciones.

Ejemplos de escenarios de ataque

Un atacante explota una biblioteca Python vulnerable para comprometer un sistema. Esto sucedió en la primera violación de datos de Open AI. Un atacante proporciona un complemento LLM para buscar vuelos, generando enlaces falsos que conducen a estafar a los usuarios. Un atacante explota el registro de paquetes PyPi para engañar a los desarrolladores de modelos para que descarguen un paquete comprometido y extraigan datos o aumenten los privilegios. en un entorno de desarrollo de modelos. Este fue un ataque real. Un atacante envenena un modelo previamente entrenado disponible públicamente y especializado en análisis económico e investigación social para crear una puerta trasera que genera información errónea y noticias falsas. Lo implementan en un mercado modelo (por ejemplo, Hugging Face) para que lo utilicen las víctimas. Un atacante envenena conjuntos de datos disponibles públicamente para ayudar a crear una puerta trasera al ajustar los modelos. La puerta trasera favorece sutilmente a determinadas empresas en diferentes mercados. Un empleado comprometido de un proveedor (desarrollador de subcontratación, empresa de hosting, etc.) extrae datos, modelos o códigos robando IP.

Un operador de LLM cambia sus términos y condiciones y su política de privacidad para exigir una exclusión voluntaria explícita del uso de datos de la aplicación para la capacitación del modelo, lo que lleva a la memorización de datos confidenciales.

Enlaces de referencia

Violación de datos de ChatGPT confirmada cuando una empresa de seguridad advierte sobre la explotación de componentes vulnerables: Proceso de revisión del complemento Security Wee: OpenA Cadena de dependencia nocturna de PyTorch comprometida: PyTorc PoisonGPT: Cómo escondimos un LLM lobotomizado en Hugging Face para difundir noticias falsas: Mithril Securit Army mirando el posibilidad de 'BOM de IA: Modos de falla de Scoo de defensa en el aprendizaje automático: Microsof ML Compromiso de la cadena de suministro: MITR Transferibilidad en el aprendizaje automático: de fenómenos a ataques de caja negra utilizando muestras adversas: Cornell Universit BadNets: Identificación de vulnerabilidades en la cadena de suministro del modelo de aprendizaje automático : Virus de la Universidad de CornellIntoxicación total: MITRE

LLM06: Divulgación de información confidencial

Descripción

Las aplicaciones LLM tienen el potencial de revelar información confidencial, algoritmos propietarios u otros detalles confidenciales a través de su salida. Esto puede resultar en acceso no autorizado a datos confidenciales, propiedad intelectual, violaciones de privacidad y otras violaciones de seguridad. Es importante que los consumidores de aplicaciones LLM sean conscientes de cómo interactuar de forma segura con los LLM e identificar los riesgos asociados con la introducción involuntaria de datos confidenciales que el LLM puede devolver posteriormente en otros lugares. Para mitigar este riesgo, las aplicaciones LLM deben realizar una desinfección de datos adecuada para evitar que los datos del usuario ingresen a los datos del modelo de capacitación. Los propietarios de aplicaciones LLM también deben tener políticas de Términos de uso apropiadas disponibles para que los consumidores sean conscientes de cómo se procesan sus datos y la posibilidad de optar por no incluir sus datos en el modelo de capacitación. La interacción consumidor-aplicación LLM forma un límite de confianza bidireccional, donde no podemos confiar inherentemente en la entrada cliente->LLM o en la salida LLM->cliente. Es importante señalar que esta vulnerabilidad supone que ciertos requisitos previos están fuera de alcance, como ejercicios de modelado de amenazas, seguridad de la infraestructura y sandboxing adecuado. Agregar restricciones dentro del mensaje del sistema en torno a los tipos de datos que el LLM debe devolver puede proporcionar cierta mitigación contra la divulgación de información confidencial, pero la naturaleza impredecible de los LLM significa que es posible que dichas restricciones no siempre se respeten y se puedan eludir mediante una inyección rápida u otros vectores.

Ejemplos comunes de vulnerabilidad

Filtrado incompleto o inadecuado de información confidencial en las respuestas del LLM. Sobreajuste o memorización de datos confidenciales en el proceso de capacitación del LLM. Divulgación involuntaria de información confidencial debido a una mala interpretación del LLM, falta de métodos de depuración de datos o errores.

Estrategias de prevención y mitigación

Integre técnicas adecuadas de desinfección y depuración de datos para evitar que los datos del usuario ingresen a los datos del modelo de entrenamiento. Implemente métodos sólidos de validación y desinfección de entradas para identificar y filtrar posibles entradas maliciosas para evitar que el modelo sea envenenado.

Al enriquecer el modelo con datos y si se ajusta un modelo: (es decir, datos introducidos en el modelo antes o durante la implementación).

Cualquier cosa que se considere confidencial en los datos de ajuste tiene el potencial de ser revelada a un usuario. Por lo tanto, aplique la regla de privilegios mínimos y no entrene el modelo con información a la que pueda acceder el usuario con mayores privilegios y que pueda mostrarse a un usuario con menos privilegios. El acceso a fuentes de datos externas (orquestación de datos en tiempo de ejecución) debe ser limitado Aplicar métodos estrictos de control de acceso a fuentes de datos externas y un enfoque riguroso para mantener una cadena de suministro segura.

Ejemplos de escenarios de ataque

El usuario legítimo desprevenido A está expuesto a otros datos de usuario a través del LLM cuando interactúa con la aplicación LLM de manera no maliciosa. El usuario A utiliza un conjunto bien elaborado de indicaciones para evitar los filtros de entrada y la desinfección del LLM para hacer que revele Información confidencial (PII) sobre otros usuarios de la aplicación. Los datos personales, como la PII, se filtran al modelo a través de datos de entrenamiento debido a negligencia del propio usuario o de la aplicación LLM. Este caso podría aumentar el riesgo y la probabilidad del escenario 1 o 2 anterior.

Enlaces de referencia

Crisis de filtración de datos de IA: una nueva herramienta evita que los secretos de la empresa se envíen a ChatGPT: Fox Busines Lecciones aprendidas de la filtración de Samsung de ChatGPT: Cybernew Cohere - Términos de uso: Coher Un ejemplo de modelado de amenazas: AI Villag Guía de privacidad y seguridad de IA de OWASP: Seguridad de IA de OWASP & Guía de privacidad Garantizar la seguridad de los modelos de lenguajes grandes: Intercambio de expertos

LLM07: Diseño de complemento inseguro

Descripción

Los complementos de LLM son extensiones que, cuando están habilitadas, el modelo las llama automáticamente durante las interacciones del usuario. La plataforma de integración de modelos los impulsa y es posible que la aplicación no tenga control sobre la ejecución, especialmente cuando el modelo está alojado por otra parte. Además, es probable que los complementos implementen entradas de texto libre del modelo sin validación ni verificación de tipo para hacer frente a las limitaciones de tamaño del contexto. Esto permite a un atacante potencial crear una solicitud maliciosa al complemento, lo que podría dar lugar a una amplia gama de comportamientos no deseados, incluida la ejecución remota de código. El daño de las entradas maliciosas a menudo depende de controles de acceso insuficientes y de la falta de seguimiento de la autorización entre complementos. Un control de acceso inadecuado permite que un complemento confíe ciegamente en otros complementos y asuma que el usuario final proporcionó las entradas. Un control de acceso tan inadecuado puede permitir que las entradas maliciosas tengan consecuencias perjudiciales que van desde la filtración de datos, la ejecución remota de código y la escalada de privilegios. Este elemento se centra en la creación de complementos de LLM en lugar de complementos de terceros, que cubre LLM-Supply-Chain-Vulnerabilities.

Ejemplos comunes de vulnerabilidad

Un complemento acepta todos los parámetros en un solo campo de texto en lugar de parámetros de entrada distintos. Un complemento acepta cadenas de configuración en lugar de parámetros que pueden anular los ajustes de configuración completos. Un complemento acepta SQL sin formato o declaraciones de programación en lugar de parámetros. La autenticación se realiza sin autorización explícita para un complemento en particular. Un complemento trata todo el contenido de LLM como creado íntegramente por el usuario y realiza cualquier acción solicitada sin requerir autorización adicional.

Estrategias de prevención y mitigación

Los complementos deben imponer entradas parametrizadas estrictas siempre que sea posible e incluir comprobaciones de tipo y rango en las entradas. Cuando esto no sea posible, se debe introducir una segunda capa de llamadas escritas, analizando solicitudes y aplicando validación y desinfección. Cuando se deben aceptar entradas de forma libre debido a la semántica de la aplicación, se debe inspeccionar cuidadosamente para garantizar que no se utilicen métodos potencialmente dañinos. Los desarrolladores de complementos deben aplicar las recomendaciones de OWASP en ASVS (Estándar de verificación de seguridad de aplicaciones) para garantizar una validación y desinfección adecuadas de las entradas.

Los complementos deben inspeccionarse y probarse minuciosamente para garantizar una validación adecuada. Utilice escaneos de pruebas de seguridad de aplicaciones estáticas (SAST) y pruebas de aplicaciones dinámicas e interactivas (DAST, IAST) en los procesos de desarrollo. Los complementos deben diseñarse para minimizar el impacto de cualquier explotación de parámetros de entrada inseguros siguiendo las pautas de control de acceso de OWASP ASVS. Esto incluye un control de acceso con privilegios mínimos, exponiendo la menor cantidad de funcionalidad posible sin dejar de realizar la función deseada. Los complementos deben utilizar identidades de autenticación apropiadas, como OAuth2, para aplicar una autorización y un control de acceso efectivos. Además, las claves API deben usarse para proporcionar contexto para decisiones de autorización personalizadas que reflejen la ruta del complemento en lugar del usuario interactivo predeterminado. Requieren autorización manual del usuario y confirmación de cualquier acción realizada por complementos confidenciales. Los complementos son, por lo general, API REST, por lo que los desarrolladores deben aplicarlas. las recomendaciones que se encuentran en OWASP Top 10 API Security Risks – 2023 para minimizar las vulnerabilidades genéricas.

Ejemplos de escenarios de ataque

Un complemento acepta una URL base e indica al LLM que combine la URL con una consulta para obtener pronósticos meteorológicos que se incluyen en el manejo de la solicitud del usuario. Un usuario malintencionado puede crear una solicitud de modo que la URL apunte a un dominio que controla, lo que le permite inyectar su propio contenido en el sistema LLM a través de su dominio. Un complemento acepta una entrada de formato libre en un solo campo que no valida. . Un atacante proporciona cargas útiles cuidadosamente diseñadas para realizar un reconocimiento de los mensajes de error. Luego explota vulnerabilidades conocidas de terceros para ejecutar código y realizar exfiltración de datos o escalada de privilegios. Un complemento utilizado para recuperar incrustaciones de un almacén de vectores acepta parámetros de configuración como una cadena de conexión sin ninguna validación. Esto permite a un atacante experimentar y acceder a otros almacenes de vectores cambiando nombres o parámetros de host y exfiltrando incrustaciones a las que no deberían tener acceso. Un complemento acepta cláusulas WHERE de SQL como filtros avanzados, que luego se agregan al SQL de filtrado. Esto permite a un atacante organizar un ataque SQL. Un atacante utiliza una inyección indirecta para explotar un complemento de administración de código inseguro sin validación de entrada y control de acceso débil para transferir la propiedad del repositorio y bloquear al usuario de sus repositorios.

Enlaces de referencia

Complementos OpenAI ChatGPT: Guía para desarrolladores de ChatGPT
Complementos OpenAI ChatGPT - Flujo de complementos: Documentación OpenAI
Complementos OpenAI ChatGPT - Autenticación: Documentación OpenAI Muestra del complemento de búsqueda semántica OpenAI: Vulnerabilidades del complemento OpenAI Githu: Visite un sitio web y le roben el código fuente: Adopte el complemento Re ChatGPT Explicación del exploit: desde la inyección rápida hasta el acceso a datos privados Adopte el Re OWASP ASVS - 5 Validación, desinfección y codificación: OWASP AASV OWASP ASVS 4.1 Diseño de control de acceso general: OWASP AASV OWASP Los 10 principales riesgos de seguridad de API - 2023: OWASP

LLM08: Agencia excesiva

Descripción

A un sistema basado en LLM a menudo su desarrollador le otorga cierto grado de agencia: la capacidad de interactuar con otros sistemas y emprender acciones en respuesta a una solicitud. La decisión sobre qué funciones invocar también se puede delegar a un 'agente' de LLM para que la determine dinámicamente en función del mensaje de entrada o la salida de LLM. Agencia excesiva es la vulnerabilidad que permite que se realicen acciones dañinas en respuesta a resultados inesperados/ambiguos de un LLM (independientemente de lo que esté causando el mal funcionamiento del LLM; ya sea alucinación/confabulación, inyección rápida directa/indirecta, complemento malicioso, mal-indicaciones benignas diseñadas o simplemente un modelo de bajo rendimiento). La causa principal de la agencia excesiva suele ser una o más de las siguientes: funcionalidad excesiva, permisos excesivos o autonomía excesiva. Esto difiere del manejo inseguro de resultados, que se ocupa del escrutinio insuficiente de los resultados del LLM. Una agencia excesiva puede generar una amplia gama de impactos en todo el espectro de confidencialidad, integridad y disponibilidad, y depende de con qué sistemas puede interactuar una aplicación basada en LLM.

Ejemplos comunes de vulnerabilidad

Funcionalidad excesiva: un agente de LLM tiene acceso a complementos que incluyen funciones que no son necesarias para el funcionamiento previsto del sistema. Por ejemplo, un desarrollador necesita otorgarle a un agente de LLM la capacidad de leer documentos de un repositorio, pero el complemento de terceros que elige usar también incluye la capacidad de modificar y eliminar documentos. Funcionalidad excesiva: es posible que un complemento se haya probado durante un fase de desarrollo y se abandonó en favor de una alternativa mejor, pero el complemento original permanece disponible para el agente de LLM. Funcionalidad excesiva: un complemento de LLM con funcionalidad abierta no filtra adecuadamente las instrucciones de entrada para comandos fuera de lo necesario para la operación prevista del solicitud. Por ejemplo, un complemento para ejecutar un comando de shell específico no evita adecuadamente que se ejecuten otros comandos de shell. Permisos excesivos: un complemento LLM tiene permisos en otros sistemas que no son necesarios para el funcionamiento previsto de la aplicación. Por ejemplo, un complemento destinado a leer datos se conecta a un servidor de base de datos utilizando una identidad que no solo tiene permisos SELECCIONAR, sino también permisos ACTUALIZAR, INSERTAR y ELIMINAR.

Permisos excesivos: un complemento LLM que está diseñado para realizar operaciones en nombre de un usuario accede a sistemas posteriores con una identidad genérica con altos privilegios. Por ejemplo, un complemento para leer el almacén de documentos del usuario actual se conecta al repositorio de documentos con una cuenta privilegiada que tiene acceso a los archivos de todos los usuarios.
Autonomía excesiva: una aplicación o complemento basado en LLM no verifica ni aprueba de forma independiente acciones de alto impacto. Por ejemplo, un complemento que permite eliminar los documentos de un usuario realiza eliminaciones sin ninguna confirmación por parte del usuario.

Estrategias de prevención y mitigación

Las siguientes acciones pueden prevenir el exceso de agencia

Limite los complementos/herramientas que los agentes de LLM pueden llamar a solo las funciones mínimas necesarias. Por ejemplo, si un sistema basado en LLM no requiere la capacidad de recuperar el contenido de una URL, entonces dicho complemento no debe ofrecerse al agente de LLM. Limite las funciones que se implementan en los complementos/herramientas de LLM al mínimo necesario. Por ejemplo, un complemento que accede al buzón de un usuario para resumir correos electrónicos puede requerir solo la capacidad de leer correos electrónicos, por lo que el complemento no debe contener otras funciones como eliminar o enviar mensajes Evite funciones abiertas siempre que sea posible (por ejemplo, ejecutar un comando de shell , buscar una URL, etc.) y utilizar complementos/herramientas con funcionalidad más granular. Por ejemplo, es posible que una aplicación basada en LLM necesite escribir algún resultado en un archivo. Si esto se implementara usando un complemento para ejecutar una función de Shell, entonces el alcance de acciones no deseadas es muy grande (se podría ejecutar cualquier otro comando de Shell). Una alternativa más segura sería crear un complemento de escritura de archivos que solo admita esa funcionalidad específica. Limite los permisos que los complementos/herramientas de LLM otorgan a otros sistemas al mínimo necesario para limitar el alcance de acciones no deseadas. Por ejemplo, un agente de LLM que utiliza una base de datos de productos para hacer recomendaciones de compra a un cliente podría necesitar solo acceso de lectura a una tabla de "productos"; no debe tener acceso a otras tablas, ni la capacidad de insertar, actualizar o eliminar registros. Esto debe aplicarse aplicando permisos de base de datos adecuados para la identidad que utiliza el complemento LLM para conectarse a la base de datos. Realice un seguimiento de la autorización del usuario y el alcance de la seguridad para garantizar que las acciones tomadas en nombre de un usuario se ejecuten en sistemas posteriores en el contexto de ese usuario específico. y con los privilegios mínimos necesarios. Por ejemplo, un complemento LLM que lee el repositorio de código de un usuario debe requerir que el usuario se autentique a través de OAuth y con el alcance mínimo requerido. Utilice el control humano en el circuito para exigir que un humano apruebe todas las acciones antes de realizarlas. Esto se puede implementar en un sistema posterior (fuera del alcance de la aplicación LLM) o dentro del propio complemento/herramienta LLM. Por ejemplo, una aplicación basada en LLM que crea y publica contenido de redes sociales en nombre de un usuario debe incluir una rutina de aprobación del usuario dentro del complemento/herramienta/API que implementa la operación de "publicación".

Implemente la autorización en sistemas posteriores en lugar de depender de un LLM para decidir si una acción está permitida o no. Al implementar herramientas/complementos, aplique el principio de mediación completo para que todas las solicitudes realizadas a los sistemas posteriores a través de los complementos/herramientas se validen según las políticas de seguridad.

Las siguientes opciones no evitarán la agencia excesiva, pero pueden limitar el nivel de daño causado.

Registre y supervise la actividad de los complementos/herramientas de LLM y los sistemas posteriores para identificar dónde se están llevando a cabo acciones no deseadas y responder en consecuencia. Implementar limitación de velocidad para reducir la cantidad de acciones no deseadas que pueden tener lugar dentro de un período de tiempo determinado, aumentando la oportunidad de descubrir acciones indeseables a través del monitoreo antes de que puedan ocurrir daños significativos.

Ejemplo de escenario de ataque

A una aplicación de asistente personal basada en LLM se le otorga acceso al buzón de correo de una persona a través de un complemento para resumir el contenido de los correos electrónicos entrantes. Para lograr esta funcionalidad, el complemento de correo electrónico requiere la capacidad de leer mensajes; sin embargo, el complemento que el desarrollador del sistema ha elegido utilizar también contiene funciones para enviar mensajes. El LLM es vulnerable a un ataque indirecto de inyección rápida, mediante el cual un correo electrónico entrante creado con fines malintencionados engaña al LLM para que ordene al complemento de correo electrónico que llame a la función "enviar mensaje" para enviar spam desde el buzón del usuario. Esto podría evitarse: (a) eliminando la funcionalidad excesiva mediante el uso de un complemento que solo ofreciera capacidades de lectura de correo, (b) eliminando permisos excesivos autenticándose en el servicio de correo electrónico del usuario a través de una sesión OAuth con un alcance de solo lectura, y/ o (c) eliminar la autonomía excesiva al requerir que el usuario revise manualmente y presione "enviar" en cada correo redactado por el complemento LLM. Alternativamente, el daño causado podría reducirse implementando una limitación de velocidad en la interfaz de envío de correo.

Enlaces de referencia

Adopte el rojo: Problema adjunto confuso: Adopte las Re
NeMo-Guardrails: Pautas de interfaz: NVIDIA Githu LangChain:
Aprobación humana para herramientas: Langchain
Documentatio Simon Willison: Patrón LLM dual: Simon Willison

LLM09: Sobredependencia

Descripción

La dependencia excesiva puede ocurrir cuando un LLM produce información errónea y la proporciona de manera autorizada. Si bien los LLM pueden producir contenido creativo e informativo, también pueden generar contenido que sea incorrecto, inapropiado o inseguro. Esto se conoce como alucinación o confabulación. Cuando las personas o los sistemas confían en esta información sin supervisión o confirmación, puede provocar una violación de la seguridad, información errónea, falta de comunicación, problemas legales y daños a la reputación. El código fuente generado por LLM puede introducir vulnerabilidades de seguridad desapercibidas. Esto plantea un riesgo significativo para la seguridad operativa de las aplicaciones. Estos riesgos muestran la importancia de procesos de revisión rigurosos, con

Supervisión continua
Mecanismo de validación
Descargos de responsabilidad sobre riesgos

Ejemplos comunes de vulnerabilidad

LLM proporciona información inexacta como respuesta y la expresa de una manera que implica que tiene mucha autoridad. El sistema en general está diseñado sin los controles y equilibrios adecuados para manejar esto y la información engaña al usuario de una manera que lleva a que LLM sugiera código inseguro o defectuoso, lo que genera vulnerabilidades cuando se incorpora a un sistema de software sin la supervisión o verificación adecuada.

Estrategias de prevención y mitigación

Supervisar y revisar periódicamente los resultados del LLM. Utilice técnicas de autoconsistencia o votación para filtrar texto inconsistente. Comparar múltiples respuestas de modelos para un solo mensaje puede juzgar mejor la calidad y consistencia del resultado. Verifique el resultado del LLM con fuentes externas confiables. Esta capa adicional de validación puede ayudar a garantizar que la información proporcionada por el modelo sea precisa y confiable.

Mejore el modelo con ajustes o incrustaciones para mejorar la calidad de salida. Es más probable que los modelos genéricos previamente entrenados produzcan información inexacta en comparación con los modelos optimizados en un dominio particular. Para este fin se pueden emplear técnicas como la ingeniería rápida, el ajuste eficiente de parámetros (PET), el ajuste completo del modelo y la cadena de pensamiento. Implementar mecanismos de validación automática que puedan verificar de forma cruzada el resultado generado con hechos o datos conocidos. Esto puede proporcionar una capa adicional de seguridad y mitigar los riesgos asociados con las alucinaciones. Divida las tareas complejas en subtareas manejables y asígnelas a diferentes agentes. Esto no solo ayuda a gestionar la complejidad, sino que también reduce las posibilidades de sufrir alucinaciones, ya que cada agente puede ser responsable de una tarea más pequeña. Comunique claramente los riesgos y limitaciones asociados con el uso de LLM. Esto incluye la posibilidad de que se produzcan imprecisiones en la información y otros riesgos. La comunicación de riesgos eficaz puede preparar a los usuarios para posibles problemas y ayudarlos a tomar decisiones informadas. Cree API e interfaces de usuario que fomenten el uso responsable y seguro de los LLM. Esto puede implicar medidas como filtros de contenido, advertencias a los usuarios sobre posibles imprecisiones y etiquetado claro del contenido generado por IA. Cuando utilice LLM en entornos de desarrollo, establezca prácticas y pautas de codificación seguras para evitar la integración de posibles vulnerabilidades.

Ejemplo de escenario de ataque

Una organización de noticias utiliza en gran medida un LLM para generar artículos de noticias. Un actor malintencionado explota esta dependencia excesiva, alimentando al LLM con información engañosa y provocando la difusión de desinformación. La IA plagia contenido sin querer, lo que genera problemas de derechos de autor y una menor confianza en la organización. Un equipo de desarrollo de software utiliza un sistema LLM para acelerar el proceso de codificación. . La dependencia excesiva de las sugerencias de la IA introduce vulnerabilidades de seguridad en la aplicación debido a configuraciones predeterminadas inseguras o recomendaciones inconsistentes con las prácticas de codificación segura. Una empresa de desarrollo de software utiliza un LLM para ayudar a los desarrolladores. El LLM sugiere una biblioteca o paquete de códigos inexistente, y un desarrollador, que confía en la IA, sin saberlo integra un paquete malicioso en el software de la empresa. Esto resalta la importancia de verificar las sugerencias de LLM, especialmente cuando se trata de bibliotecas o códigos de terceros.

Enlaces de referencia

Comprensión de las alucinaciones de los LLM: Mediu ¿Cómo deberían las empresas comunicar los riesgos de los modelos lingüísticos grandes a los usuarios?: Tech Policy Pres Un sitio de noticias utilizó IA para escribir artículos. Fue un desastre periodístico: The Washington Pos Alucinaciones de IA: Paquete de riesgo: Vulca Cómo reducir las alucinaciones de modelos de lenguaje grandes: el nuevo Stac Pasos prácticos para reducir las alucinaciones: Diseño con aprendizaje automático

LLM10: Robo de modelos

Descripción

Esta entrada se refiere al acceso no autorizado y la exfiltración de modelos LLM por parte de actores maliciosos o APT. Esto surge cuando los modelos LLM propietarios (que son propiedad intelectual valiosa) se ven comprometidos, robados físicamente, copiados o se extraen pesos y parámetros para crear un equivalente funcional. El impacto del robo de modelos LLM puede incluir pérdida económica y de reputación de marca, erosión de la ventaja competitiva, uso no autorizado del modelo o acceso no autorizado a información confidencial contenida en el modelo. El robo de LLM representa una importante preocupación de seguridad a medida que los modelos lingüísticos se vuelven cada vez más poderosos y prevalentes. Las organizaciones y los investigadores deben priorizar medidas de seguridad sólidas para proteger sus modelos LLM, garantizando la confidencialidad e integridad de su propiedad intelectual. Emplear un marco de seguridad integral que incluya controles de acceso, cifrado y monitoreo continuo es crucial para mitigar los riesgos asociados con el robo de modelos LLM y salvaguardar los intereses tanto de las personas como de las organizaciones que dependen de LLM.

Ejemplos comunes de vulnerabilidad

Un atacante aprovecha una vulnerabilidad en la infraestructura de una empresa para obtener acceso no autorizado a su repositorio de modelos LLM a través de una configuración incorrecta en la configuración de seguridad de su red o aplicación. Utilice un inventario o registro de modelos de ML centralizado para los modelos de ML utilizados en producción. Tener un registro de modelos centralizado evita el acceso no autorizado a los modelos de aprendizaje automático mediante controles de acceso, autenticación y capacidad de monitoreo/registro, que son buenas bases para la gobernanza. Tener un repositorio centralizado también es beneficioso para recopilar datos sobre los algoritmos utilizados por los modelos con fines de cumplimiento, evaluación de riesgos y mitigación de riesgos. Un escenario de amenaza interna en el que un empleado descontento filtra el modelo o artefactos relacionados. Un atacante consulta la API del modelo usando cuidadosamente entradas diseñadas y técnicas de inyección rápida para recopilar una cantidad suficiente de salidas para crear un modelo de sombra. Un atacante malicioso puede eludir las técnicas de filtrado de entrada del LLM para realizar un ataque de canal lateral y, en última instancia, recopilar pesos de modelo e información de arquitectura para un control remoto. recurso.

El vector de ataque para la extracción de modelos implica consultar el LLM con una gran cantidad de mensajes sobre un tema en particular. Los resultados del LLM se pueden utilizar para ajustar otro modelo. Sin embargo, hay algunas cosas a tener en cuenta sobre este ataque.

- El atacante debe generar una gran cantidad de mensajes específicos. Si las indicaciones no son lo suficientemente específicas, los resultados del LLM serán inútiles. Los resultados de los LLM a veces pueden contener respuestas alucinadas, lo que significa que es posible que el atacante no pueda extraer el modelo completo, ya que algunos de los resultados pueden no tener sentido.
    - No es posible replicar un LLM al 100% mediante extracción de modelos. Sin embargo, el atacante podrá replicar un modelo parcial.

El vector de ataque para la replicación del modelo funcional implica el uso del modelo objetivo a través de indicaciones para generar datos de entrenamiento sintéticos (un enfoque llamado "autoinstrucción") para luego usarlo y ajustar otro modelo fundamental para producir un equivalente funcional. Esto evita las limitaciones de la extracción tradicional basada en consultas utilizada en el Ejemplo 5 y se ha utilizado con éxito en la investigación sobre el uso de un LLM para capacitar a otro LLM. Aunque en el contexto de esta investigación la replicación de modelos no es un ataque. Un atacante podría utilizar este enfoque para replicar un modelo propietario con una API pública.

El uso de un modelo robado, como modelo en la sombra, se puede utilizar para organizar ataques adversarios, incluido el acceso no autorizado a información confidencial contenida en el modelo, o experimentar sin ser detectado con entradas adversarias para realizar inyecciones rápidas avanzadas.

Estrategias de prevención y mitigación

Implementar controles de acceso sólidos (por ejemplo, RBAC y regla de privilegio mínimo) y mecanismos de autenticación sólidos para limitar el acceso no autorizado a los repositorios de modelos LLM y los entornos de capacitación.

- Esto es particularmente cierto para los primeros tres ejemplos comunes, que podrían causar esta vulnerabilidad debido a amenazas internas, configuración incorrecta y/o controles de seguridad débiles sobre la infraestructura que alberga los modelos, pesos y arquitectura de LLM en los que un actor malicioso podría infiltrarse desde información interna o externa. fuera del entorno El seguimiento de la gestión de proveedores, la verificación y las vulnerabilidades de dependencia son temas importantes para prevenir ataques a la cadena de suministro.
- Esto es particularmente cierto para todos los ejemplos comunes, ya que cubre riesgos y amenazas internos, pero también controla en última instancia a qué "tiene acceso" la aplicación LLM y, por lo tanto, podría ser un mecanismo o paso de prevención para evitar ataques de canal lateral.

Restringir el acceso del LLM a los recursos de la red, los servicios internos y las API.

Supervise y audite periódicamente los registros de acceso y las actividades relacionadas con los repositorios de modelos LLM para detectar y responder a cualquier comportamiento sospechoso o no autorizado con prontitud.

Automatizar la implementación de MLOps con flujos de trabajo de gobernanza, seguimiento y aprobación para reforzar los controles de acceso e implementación dentro de la infraestructura. Implementar controles y estrategias de mitigación para mitigar y/o reducir el riesgo de técnicas de inyección rápida que causen ataques de canal lateral. Limitación de velocidad de llamadas API cuando corresponda y/o filtros para reducir el riesgo de filtración de datos de las aplicaciones LLM, o implementar técnicas para detectar (por ejemplo, DLP) actividad de extracción de otros sistemas de monitoreo Implementar capacitación en robustez adversaria para ayudar a detectar tentativas de extracción y refinar las medidas de seguridad física Implementar un marco de marca de agua en la incrustación y Etapas de detección del ciclo de vida de LLM.

Ejemplo de escenario de ataque

Un atacante aprovecha una vulnerabilidad en la infraestructura de una empresa para obtener acceso no autorizado a su repositorio de modelos LLM. El atacante procede a extraer valiosos modelos LLM y los utiliza para lanzar un servicio de procesamiento de idiomas de la competencia o extraer información confidencial, lo que causa un daño financiero significativo a la empresa original. Un empleado descontento filtra el modelo o artefactos relacionados. La exposición pública de este escenario aumenta el conocimiento de los atacantes sobre ataques adversarios de caja gris o, alternativamente, roban directamente la propiedad disponible. Un atacante consulta la API con entradas cuidadosamente seleccionadas y recopila una cantidad suficiente de salidas para crear un modelo oculto. Hay una falla de control de seguridad dentro del cadena de suministro y conduce a fugas de datos de información de modelo patentada. Un atacante malicioso omite las técnicas de filtrado de entrada y los preámbulos del LLM para realizar un ataque de canal lateral y recuperar información del modelo en un recurso controlado remotamente bajo su control.

Enlaces de referencia

El poderoso modelo de lenguaje de inteligencia artificial de Meta se filtró en línea: The Verg Runaway LLaMA | Cómo se filtró el modelo LLaMA NLP de Meta: DeepLearning.a AML.TA0000 Acceso al modelo ML: MITRE ATLA Sé lo que ves: Cornell Universit D-DAE: Ataques de extracción de modelos que penetran la defensa: IEE Un marco de defensa integral contra ataques de extracción de modelos: IEE Alpaca : Un modelo sólido y replicable que sigue instrucciones: Universidad de Stanford ¿Cómo la marca de agua puede ayudar a mitigar los riesgos potenciales de los LLM?: KD Nuggets

Equipo central y colaboradores

Los miembros del equipo principal aparecen en azul

Adam Swanda Adesh Gairola Anuncios Dawson Adrian Culley Aleksei Ryzhkov Alexander Zai Aliaksei Bialko Amay Trivedi Ananda Krishna Andrea Succi Andrew Amaro Andy Dyrcz Andy Smith Ashish Rajan Aubrey King Bajram Hoxha Bilal Siddiqui Bob Simonoff Brian Pendleton Brodie McRae Cassio Goldschmidt Dan Frommer Dan Klein David Rowe David Taylor Dotan Nahum Dr. Matteo Große-Kampmann Emanuel Valente

AWS Coher e Trellix EPAM

EPA M

Seguridad Astra

Grupo de seguridad Klavan Linkfire

Podcast de seguridad en la nube F5 DevCentral Databook Trustwave Blue Yonder AVID AWS ServiceTitan

Accenture AWS

Punto de control AWARE7 iFood

Emmanuel Guilherme Junior Eugene Neelou Eugene Tawiah Fernando Calderón Gaurav “GP” Pal Gavin Klondike Guillaume Ehinger Idan Hen Itamar Golan James Rabe Jason Axley Jason Haddix Jason Ross Jeff Williams Jinson Varghese Behanan Johann Rehberger John Sotiropoulos Jorge Pinto Joshua Nussbaum Ken Arora Ken Huang Kelvin Low Koichiro Watanabe Larry Carson Leon Derczynski Leonardo Shikida Lior Drihem Manish Kumar Yadav

Universidad McMaster

Unidad de Tecnologías Complejas 42 Palo Alto Networks stackArmor AI Village Google Microsoft

Iriusrisk AWS Buddobot Salesforce Contrast Security Astra Security

Precio s

F5 DistributedApps.ai aigos Microsoft

U de W, UIT de Copenhague IBM

SA VIA

Manjesh S Manuel S. Lemos Mehdi Karimi Mike Finch Mike Jang Nathan Hamiel Natsu Ooke Neal Swaelens Nicholas Grove Nipun Gupta Nir Paz Oleksandr Yaremchuk Ophir Dror Otto Sulin Parveen Yadav Patrick Biyaga Priyadharshini Parthasarathy Rachit Sood Rahul Zhade Ravi Singh Ravin Thambapillai Reza Rashidi Rich Harang Riotaro OKADA Ron F. Del Rosario Ross Moore Sandy Dunn Sanket Naik

HackerOne BeDisruptive | ANBAN

HackerOne Forescout Kudelski Security Aeye Security Lab Inc. LLM Guard Security, AI, Microsoft Bearer

Guardia LLM

Familia Nordic Venture HackerOne Thenavigo Coalfire

GitHub AddValueMachine Inc Credal.ai HADESS NVIDIA OWASP JAPAN, Asterisk Research Investigador y escritor de seguridad de SAP

lluvia de fuego

Equipo central y colaboradores

Los miembros del equipo principal aparecen en azul

Nombre	
Santosh Kumar	Cisco
Sarah Thornton	ro rojo
Stefano Amorelli	Red Hat
Steve Wilson	de
Talesh SEEPARSAN	Cisco
VANDANA VERMA	Contrast Security
SEHGAL VINAY	Bit79
VISHWANATHA VISHWAS	Snyk
MANRAL	SumUp
VLADIMIR FUDOTOV	SprinklR
CHILCUTT	Precize
YEVHenii Molchanov	EPAM
Yusuke Karasawa	Yahoo
	SumUp
	Japan Digital Design, Inc.