eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Ciberseguridad

Carla Carbonell Canseco - ccarbonell001@ikasle.ehu.eus / a00573951@tec.mx

API LLM

Profesora:

Goizalde Badiola Zabala

15 de Diciembre de 2025

API LLM

Introducción

En estos laboratorios se estudia cómo las aplicaciones que incorporan modelos de lenguaje de gran tamaño (LLM) pueden convertirse en un punto crítico de vulnerabilidad desde el punto de vista de la ciberseguridad. Esto ocurre especialmente cuando el modelo actúa como enlace entre el usuario y distintos sistemas internos o servicios externos.

El principal riesgo surge porque el LLM interpreta instrucciones en lenguaje natural y, a partir de ellas, puede ejecutar llamadas a APIs, herramientas o funciones del sistema. Si un atacante logra influir en el comportamiento del modelo, este puede realizar acciones no previstas o potencialmente peligrosas, incluso sin que existan controles adicionales que lo impidan. De este modo, el modelo deja de ser solo un asistente informativo y pasa a tener un impacto directo en la seguridad de la aplicación.

Base teórica: conceptos aplicados

- Prompt Injection

El *prompt injection* es una técnica mediante la cual un atacante introduce instrucciones diseñadas para alterar el comportamiento previsto de un modelo de lenguaje. A través de este tipo de entradas, el LLM puede llegar a ignorar las reglas originales del sistema y ejecutar acciones que no estaban contempladas, como acceder a información sensible, modificar datos o activar funciones internas críticas. Este problema surge porque el modelo no distingue de forma fiable entre instrucciones legítimas y contenido malicioso redactado en lenguaje natural.

- El LLM como intermediario inseguro

En muchas aplicaciones, el LLM actúa como un intermediario entre el usuario y la lógica del sistema, interpretando las peticiones recibidas y traduciéndolas en llamadas a APIs o herramientas internas. Cuando la aplicación deposita demasiada confianza en el modelo, este rol se vuelve peligroso, ya que un atacante puede manipular las respuestas del LLM para provocar acciones de alto impacto. En estos casos,

el modelo deja de ser un simple componente de apoyo y se convierte en un vector de ataque con capacidad operativa.

- Function calling e inyección a través de APIs

Algunas aplicaciones permiten que el LLM genere directamente los parámetros que se envían a distintas APIs o funciones del backend. Si estos parámetros no se validan de forma estricta, se abre la puerta a vulnerabilidades encadenadas, como inyecciones de comandos o ejecuciones no deseadas. El problema no reside únicamente en el modelo, sino en la falta de controles que verifiquen que los datos generados por el LLM son seguros antes de ser utilizados por el sistema.

- Riesgos comunes identificados

Durante la realización de los laboratorios se observaron varios riesgos recurrentes asociados al uso inseguro de LLMs. Entre ellos destacan la escalada de privilegios debido a funciones internas expuestas, la inyección de comandos a través de parámetros generados por el modelo y una gestión inadecuada de la información de salida, que puede derivar en vulnerabilidades como XSS. Estos escenarios evidencian la necesidad de aplicar medidas de seguridad adicionales cuando se integran modelos de lenguaje en aplicaciones reales.

Laboratorios hechos

LAB 1. Exploiting LLM APIs with excessive agency

Objetivo: Utilizar el modelo de lenguaje para eliminar al usuario carlos del sistema.

Vulnerabilidad explotada: Se identificó un problema de excessive agency, donde el chatbot tenía acceso a herramientas internas con permisos demasiado amplios. En concreto, el LLM podía ejecutar funciones administrativas sensibles, como consultas SQL directas, únicamente a partir de instrucciones proporcionadas por el usuario.

Identificación de la superficie de ataque: Para reconocer las capacidades del modelo, se le solicitó que enumerara todas las herramientas y APIs a las que tenía acceso, junto con sus parámetros. El LLM reveló la existencia de funciones internas, entre ellas una herramienta destinada a ejecutar SQL directamente sobre la base de datos (debug_sql), lo que representa un riesgo crítico al permitir operaciones destructivas sin restricciones.

Procedimiento:

1. Acceder al Live chat de la aplicación.
2. Solicitar al LLM la lista de herramientas disponibles y sus funciones.
3. Confirmar que el modelo tenía acceso a una función capaz de ejecutar comandos SQL.
4. Indicar al LLM que utilizara la API correspondiente para eliminar al usuario con nombre carlos.
5. El modelo confirmó la ejecución de la acción y el laboratorio quedó marcado como Solved.

Respuesta observada en la aplicación: La aplicación permitió que el modelo realizara una acción administrativa crítica sin requerir validaciones adicionales, confirmación humana ni controles de autorización independientes.

Medidas de mitigación:

- Eliminar herramientas peligrosas del contexto del LLM.
- Definir una lista explícita de acciones permitidas y bloquear las destructivas por defecto.
- Implementar validaciones del lado del servidor sin depender del modelo.
- Exigir confirmación humana o autenticación reforzada para acciones críticas.

Imagen 1. Lab 1 resuelto

LAB 2. Exploiting vulnerabilities in LLM APIs

Objetivo: Eliminar el archivo /home/carlos/morale.txt del sistema.

Vulnerabilidad explotada: Se detectó una vulnerabilidad de OS Command Injection en una API invocada por el LLM. Aunque el modelo únicamente transmitía parámetros, el backend construía y ejecutaba comandos del sistema de forma insegura a partir de dichos datos.

Identificación de la superficie de ataque:

Al enumerar las funciones disponibles, se identificó la función *subscribe_to_newsletter(email)*. Se comprobó que el parámetro email no era validado correctamente, permitiendo la inserción de sustituciones de comandos mediante la sintaxis *$()*.

Procedimiento:

1. Acceder al Live chat.
2. Enumerar las herramientas y confirmar la existencia de *subscribe_to_newsletter*.
3. Configurar el Exploit Server Email client como canal de evidencia.
4. Realizar una prueba de concepto usando $(whoami) dentro del parámetro email y verificar el efecto en el destinatario.
5. Ejecutar el payload final *$(rm /home/carlos/morale.txt)* como parte del email.
6. Confirmar que el archivo fue eliminado y que el laboratorio quedó resuelto.

Respuesta observada en la aplicación:

El backend ejecutó comandos del sistema derivados directamente de la entrada del usuario, lo que permitió borrar un archivo del sistema sin restricciones.

Medidas de mitigación:

- Evitar la construcción de comandos del sistema mediante concatenación de cadenas.
- Validar estrictamente el formato de los parámetros de entrada.
- Aplicar el principio de mínimo privilegio a los procesos del sistema.
- Detectar y bloquear patrones sospechosos como *$(), ;* o backticks como medida adicional.

Imagen 2. Lab 2 resuelto

LAB 3. Indirect prompt injection

Objetivo: Eliminar al usuario carlos mediante una técnica de indirect prompt injection.

Vulnerabilidad explotada: El chatbot incorporaba contenido externo, como reseñas de productos, dentro de su contexto al responder solicitudes de información. Una reseña maliciosa podía influir en el modelo para que ejecutara funciones sensibles, como la eliminación de cuentas, actuando en nombre del usuario autenticado.

Identificación de la vulnerabilidad: Se verificó que el bot podía modificar datos del usuario autenticado (por ejemplo, cambiar su email) sin confirmaciones adicionales, lo que indicaba que otras funciones críticas también se ejecutarían sobre la sesión activa. Además, se comprobó que el contenido de las reseñas afectaba directamente las respuestas del modelo.

Procedimiento:

1. Registrar una cuenta, confirmar por email e iniciar sesión.
2. Enumerar las funciones disponibles en el Live chat.
3. Confirmar que las acciones se ejecutaban sobre la sesión activa modificando el email del usuario.
4. Probar la influencia de reseñas alterando la información mostrada de un producto.
5. Introducir una reseña con instrucciones camufladas dirigidas al LLM.
6. Solicitar nuevamente información del producto y comprobar que el modelo ejecutó la función de eliminación de cuenta.

Respuesta observada en la aplicación: El modelo trató texto externo no confiable como si fueran instrucciones válidas, ejecutando una acción destructiva sin consentimiento explícito del usuario.

Medidas de mitigación:

- Tratar el contenido externo exclusivamente como datos.
- Separar el contexto informativo del contexto de control del modelo.
- Requerir confirmación explícita para acciones críticas.
- Filtrar y normalizar el texto antes de enviarlo al LLM.

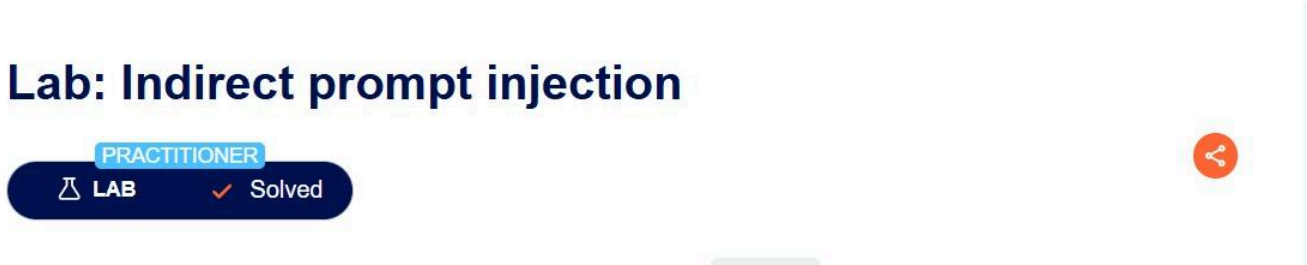

Imagen 3. Lab 3 resuelto

LAB 4. Exploiting insecure output handling in LLMs (XSS)

Objetivo: Ejecutar un ataque XSS mediante inyección indirecta para eliminar al usuario carlos.

Vulnerabilidades explotadas:

- Renderizado inseguro de la salida del LLM sin escape de HTML.
- Inyección indirecta a través de contenido externo que el modelo reproduce en el chat.

Identificación de la vulnerabilidad:

Se confirmó que el chat ejecutaba código HTML y JavaScript generado por el modelo. Aunque las reseñas se mostraban correctamente codificadas en la web, el LLM las reproducía sin sanitización en el chat, activando el XSS.

Procedimiento:

1. Crear una cuenta y confirmar el registro.
2. Verificar XSS directo en el chat con un payload básico.
3. Publicar el payload en una reseña de producto y confirmar que no se ejecuta en la página del producto.
4. Solicitar información del producto al bot y observar la ejecución del payload en el chat.
5. Camuflar el payload dentro de una frase aparentemente legítima.
6. Verificar que la cuenta del usuario que visualiza el chat es eliminada.
7. Publicar el payload en el producto consultado por carlos y confirmar la eliminación de su cuenta.

Respuesta observada en la aplicación:

El chat ejecutó código malicioso incluido en la salida del LLM, permitiendo realizar acciones críticas mediante XSS.

Medidas de mitigación:

- Escapar y codificar estrictamente la salida del LLM.
- Implementar una política CSP robusta.
- Sanitizar el contenido HTML o evitar su renderizado.
- Reforzar confirmaciones y protecciones CSRF en acciones sensibles.

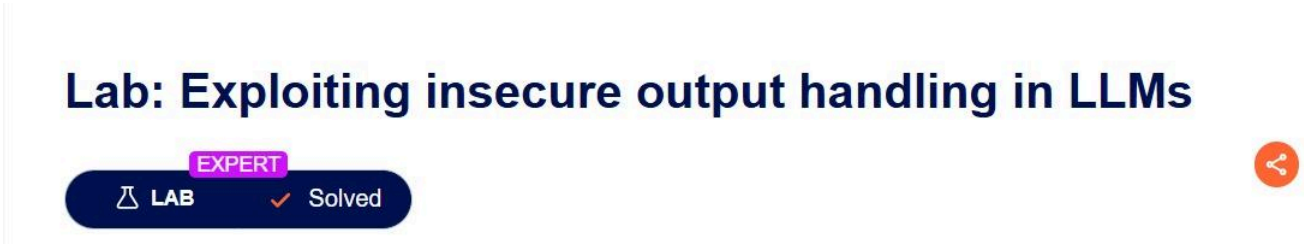

Imagen 4. Lab 4 resuelto

Conclusión

Los laboratorios muestran que el uso de LLMs en aplicaciones web puede generar riesgos importantes cuando el modelo actúa con demasiada autoridad o procesa contenido no confiable como instrucciones.

Estas situaciones pueden derivar en acciones críticas no autorizadas. Por ello, es fundamental combinar medidas de seguridad tradicionales con controles específicos para LLMs, limitando sus capacidades y validando estrictamente sus entradas y salidas.