INSTALACIÓN Y AUDITORÍA DE MODELOS LLM EN ENTORNOS LOCALES

CAPTURA Y MONITORIZACIÓN DE TRÁFICO DE MODELOS GRANDES DE LENGUAJE (LLMs) EJECUTÁNDOSE SOBRE OLLAMA, VLLM Y TRANSFORMERS

octubre de 2025

Financiado por la Unión Europea NextGenerationEU

GOBIERNO DE ESPAÑA
MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES

Plan de Recuperación, Transformación y Resiliencia

incibe_
INSTITUTO NACIONAL DE CIBERSEGURIDAD

ÍNDICE

ÍNDICE DE ILUSTRACIONES

GLOSARIO DE TÉRMINOS Y ACRÓNIMOS

Término	Significado
LLM	*Large Language Model*
API	*Application Programming Interface*
CPU	*Central Processing Unit*
UUID	*Universal Unique Identifier*
GPU	Graphical Processing Unit
TLS	*Transport Layer Security*
SSL	*Secure Sockets Layer*, protocolo predecesor de TLS
HTTPS	Protocolo HTTP cifrado mediante TLS/SSL
REST	*Representational State Transfer*
Handshake	Proceso inicial de negociación entre cliente y servidor para establecer claves y parámetros de cifrado
Ciphersuite	Conjunto de algoritmos acordado en el *handshake* (cifrado, autenticación, *hash*)
Master Secret	Clave derivada durante el *handshake* usada para generar las claves de sesión en TLS ≤ 1.2
Traffic Secret	Clave usada en TLS 1.3 para cifrar el tráfico de aplicación
SSLKEYLOGFILE	Archivo que registra secretos de sesión TLS para permitir el descifrado de tráfico en auditorías
Client Random	Valor aleatorio del cliente que identifica la sesión y sirve para derivar claves
Session Keys	Claves de sesión únicas generadas por cada conexión TLS
Pcap	Archivo de captura de paquetes de red (*Packet Capture*)
Tcpdump	Herramienta de línea de comandos para capturar y analizar el tráfico de red en sistemas Unix/Linux
Wireshark	Herramienta de análisis de tráfico capaz de descifrar TLS usando el SSLKEYLOGFILE
MitM (*Man-in-the-Middle*)	Técnica de interceptación del tráfico, usada en auditorías para inspeccionar comunicaciones cifradas en entornos controlados
Telemetría	Datos de uso o diagnóstico enviados automáticamente por un *software*
Data Exfiltration	Transmisión no autorizada de información fuera del sistema

Docker Bridge	Interfaz virtual que conecta contenedores en una red interna y permite capturar su tráfico
Hex	Abreviatura del formato numérico hexadecimal
FastAPI	*Framework web* en Python usado por vLLM para ofrecer su API HTTPS
PagedAttention	Técnica de optimización en vLLM que mejora la inferencia mediante gestión eficiente de memoria
LoRA (*Low-Rank Adaptation*)	Método de ajuste fino para modelos LLM que reduce el coste computacional
Inference Request	Solicitud enviada al modelo para generar una respuesta o inferencia
Telemetry Opt-out	Mecanismo de configuración para desactivar el envío de estadísticas o datos de uso
Bfloat16	Formato numérico de 16 *bits* usado para acelerar cálculos de IA con menor consumo de memoria
Caché KV	Mecanismo de almacenamiento temporal de claves y valores usado en LLMs para acelerar inferencias
Container Network	Red lógica creada por Docker para interconectar contenedores de forma aislada
Log	Registro de un evento
Audit Logging	Registro detallado de eventos y comunicaciones durante una auditoría de tráfico
Decrypt Traffic	Proceso de descifrar comunicaciones TLS utilizando las claves extraídas o registradas

1. INTRODUCCIÓN

La adopción masiva de LLMs (*Large Language Models*) en entornos empresariales y de investigación ha impulsado el desarrollo de soluciones locales como Ollama, vLLM y despliegues personalizados basados en la librería Transformers [1]. Estos gestores permiten ejecutar modelos de IA de forma local, evitando la dependencia de APIs (*Application Programming Interface*) externas y reduciendo riesgos de privacidad y confidencialidad. Sin embargo, la interacción con estos modelos puede generar tráfico de red que, si no se gestiona adecuadamente, puede convertirse en un vector de filtración de información sensible.

El presente informe analiza el tráfico generado por los principales gestores de modelos LLM, evaluando su rendimiento, los riesgos de filtración de datos y las técnicas de interceptación de comunicaciones HTTPS mediante captura de claves en entornos Docker. El objetivo es identificar vulnerabilidades, proponer medidas de mitigación y establecer buenas prácticas para la protección de datos en entornos de producción.

1.1. Objetivos del informe

- Analizar el tráfico de red generado por los gestores de modelos LLM como Ollama [2], vLLM [3] y entornos personalizados con Transformers.
- Evaluar técnicas de interceptación y descifrado de comunicaciones HTTPS en Docker.
- Identificar vectores de filtración de información sensible durante la interacción con los modelos.
- Proponer medidas de seguridad para minimizar riesgos y garantizar la confidencialidad de los datos.

Este informe incluye un análisis técnico del protocolo TLS (***handshake***, secretos de sesión y diferencias entre TLS 1.2 y TLS 1.3), y documenta las implicaciones operativas y de confidencialidad asociadas a la generación de **SSLKEYLOGFILE** en entornos de contenedor.

2. ESTADO DEL ARTE

El auge de los **LLM** ha transformado el panorama de la inteligencia artificial aplicada. Estos modelos, basados en arquitecturas Transformer y entrenados con volúmenes masivos de texto, han demostrado capacidades cada vez mayores en generación de lenguaje natural, razonamiento y adaptación a múltiples tareas. La facilidad para integrar estas tecnologías en entornos productivos ha favorecido su adopción, tanto en investigación como en la industria, con un impacto comparable al de otras tecnologías disruptivas como el ***cloud computing***.

En este contexto, la forma de acceso a los modelos se ha convertido en un eje estratégico. Por un lado, los proveedores en la **nube** ofrecen servicios de inferencia escalables y optimizados, con costes variables y mantenimiento delegado. Este enfoque facilita la puesta en marcha inmediata, pero implica la transferencia de datos potencialmente sensibles a infraestructuras externas, con los consiguientes riesgos en materia de privacidad, confidencialidad y cumplimiento normativo. Por otro lado, los despliegues locales u ***on-premise*** han ganado relevancia como alternativa para organizaciones que buscan control total sobre la información, reducción de la latencia y mayor soberanía tecnológica, aunque a cambio de mayores costes en infraestructura y operación.

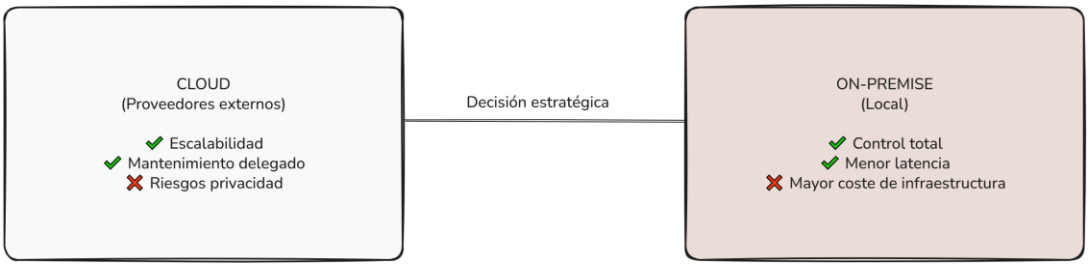

Ilustración 2.1: Decisión entre LLM Cloud y On-Premise

En cuanto a los gestores y motores de ejecución de LLM, se observa un ecosistema en rápida consolidación. **Ollama** se ha posicionado como una herramienta orientada a la **usabilidad**, ofreciendo la posibilidad de descargar, gestionar y servir modelos en entornos locales con una experiencia cercana a la de las APIs comerciales. **vLLM**, en cambio, se ha desarrollado con un enfoque de **rendimiento**, optimizando el uso de memoria y la concurrencia para permitir inferencias a gran escala en clústeres o contenedores. Finalmente, la librería **Transformers** de Hugging Face sigue siendo la referencia en **investigación y despliegues personalizados**, al proporcionar flexibilidad para diseñar arquitecturas específicas y combinarse con *frameworks* de *serving* o plataformas de orquestación.

Pese a que los LLM locales reducen la dependencia de terceros, **no están exentos de riesgos de seguridad**. La interacción con los modelos implica tráfico de red asociado a la descarga de parámetros, la comunicación entre contenedores o la exposición de APIs internas. Además, el uso de protocolos cifrados como **TLS** introduce desafíos en el análisis de tráfico: aunque protegen la confidencialidad de los datos en tránsito, también dificultan la monitorización y pueden abrir vectores de filtración si no se gestionan adecuadamente. En este ámbito, técnicas como la utilización del fichero **SSLKEYLOGFILE** en entornos Docker permiten **capturar secretos de sesión** y descifrar flujos TLS con fines de auditoría.

De este modo, el estado actual muestra una tensión entre la búsqueda de eficiencia y escalabilidad en el despliegue de LLM, y la necesidad de reforzar medidas de seguridad que aseguren la confidencialidad de los datos. Este equilibrio es especialmente crítico en entornos productivos, donde las fugas de información pueden tener consecuencias operativas, legales y reputacionales significativas.

3. FUNDAMENTOS TÉCNICOS

3.1. Arquitectura de los gestores de modelos LLM

3.1.1. Ollama

Ollama permite el despliegue local de modelos como Llama o Mistral mediante una API REST, utilizando HTTP/HTTPS para la comunicación. Su arquitectura modular facilita la integración con diferentes *backends*, pero la configuración por defecto puede exponer metadatos y datos sensibles si no se implementan controles de seguridad adecuados.

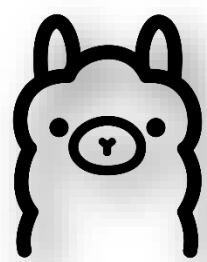

Ilustración 3.1: Logotipo de Ollama

Ollama se distribuye bajo la licencia MIT (Massachusetts Institute of Technology), una licencia de *software* libre reconocida por su simplicidad y flexibilidad. Esta licencia permite utilizar, copiar, modificar, fusionar, publicar, distribuir, sublicenciar y vender el *software*, incluso dentro de proyectos propietarios, siempre que se mantenga el aviso de *copyright* y la licencia en todas las copias o partes sustanciales del *software*. [4]

El *software* se proporciona “tal cual”, sin garantía expresa ni implícita, incluyendo, pero sin limitarse a, garantías de comerciabilidad o idoneidad para un propósito particular. Los autores no se responsabilizan de ningún daño o reclamo que pueda surgir del uso del *software*.

3.1.2. vLLM

vLLM está optimizado para inferencia de alta velocidad, gracias a técnicas como PagedAttention [5]. Su API, basada en FastAPI, soporta HTTPS, pero la configuración de TLS queda a cargo del usuario, lo que puede resultar en implementaciones inseguras si no se siguen buenas prácticas.

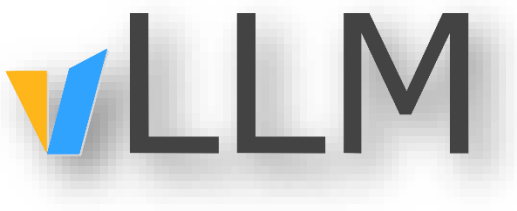

Ilustración 3.2: Logotipo de vLLM

vLLM se distribuye bajo Apache License 2.0, una licencia de *software* libre que permite usar, modificar, redistribuir y sublicenciar el *software*, incluso dentro de proyectos comerciales o propietarios, siempre que se mantengan los avisos de *copyright* y la propia licencia. El *software* se proporciona "tal cual", sin garantías de funcionamiento ni responsabilidad de los autores por daños derivados de su uso. [6]

3.1.3. Entornos personalizados con Transformers

Los despliegues personalizados con Transformers [7] ofrecen máxima flexibilidad, pero requieren una configuración manual de la infraestructura de red y seguridad, lo que incrementa el riesgo de errores y exposición de datos.

Ilustración 3.3: Logotipo de Transformers

La librería Transformers, desarrollada por Hugging Face, se distribuye bajo la misma licencia que vLLM, es decir, Apache License 2.0.

Los modelos pre-entrenados disponibles en Hugging Face pueden tener licencias diferentes, por lo que es necesario revisar la licencia específica de cada modelo antes de su uso, especialmente si se planea integrarlos en proyectos comerciales o entornos de producción. [8]

3.2. HTTPS / TLS: *Handshake* y secretos de sesión

3.2.1. Resumen

HTTPS es HTTP sobre TLS. TLS proporciona confidencialidad, integridad y autenticación entre cliente y servidor mediante un proceso de negociación (*handshake*) que acuerda parámetros criptográficos y deriva los secretos de sesión empleados para cifrar los datos de aplicación.

3.2.2. Objetivos de TLS

- **Confidencialidad**: que sólo cliente y servidor puedan leer el contenido intercambiado.
- **Integridad**: detección de modificaciones en tránsito.
- **Autenticación**: verificación de la identidad del servidor (y opcionalmente del cliente).

3.2.3. TLS 1.2: resumen del *handshake* (esquema)

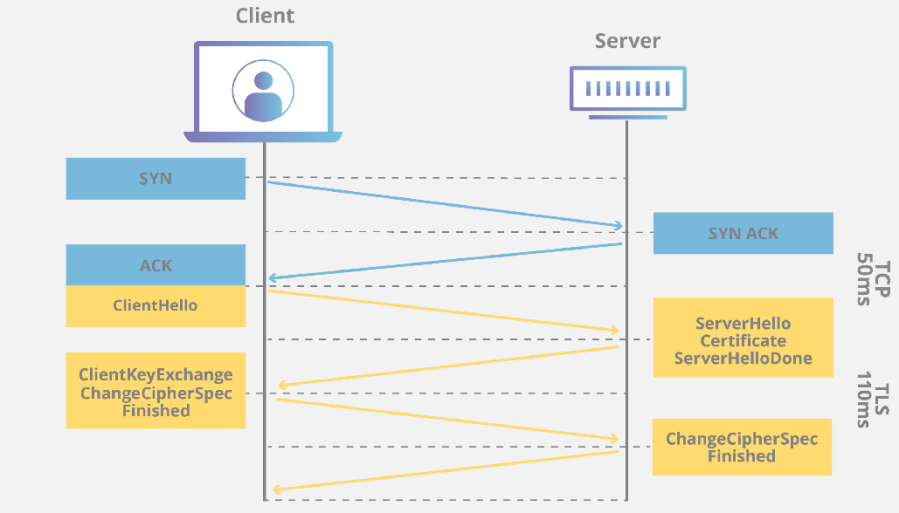

Ilustración 3.4: SSL Handshake [9]

1. ***ClientHello***: el cliente anuncia versiones, *ciphersuites* y envía *ClientRandom* (32 *bytes*).
2. ***ServerHello***: el servidor selecciona la *ciphersuite* y envía *ServerRandom*.
3. Intercambio de parámetros para generar el **secreto compartido.** Se pueden utilizar los siguientes protocolos criptográficos:
   - **RSA** o Rivest, Shamir y Adleman, (uso menos habitual o en desuso): el cliente cifraría un *premaster secret* con la clave pública del servidor.
   - **ECDHE** o Elliptic-curve Diffie-Hellman Ephimeral (uso más habitual): cliente y servidor intercambian *ephemeral key shares* para calcular un valor secreto compartido o DH.
4. Derivación del ***master secret***: a partir de *ClientRandom, ServerRandom y el premaster/DH* compartido, mediante una familia de funciones pseudoaleatorias.
5. Derivación de las **claves de sesión**: el *master secret* se usa para generar claves de cifrado y MACs (*client/server write keys*).

6. ***ChangeCipherSpec / Finished***: se verifica que ambas partes comparten las mismas claves y se comienza a cifrar la aplicación de datos.

Conocer el *master secret* permite reconstruir (o derivar) las claves que cifran la sesión TLS con versión ≤1.2, por lo que su divulgación equivale a la pérdida de confidencialidad de esa sesión.

3.2.4. TLS 1.3: diferencias clave

- ***Key schedule HKDF-based***: TLS 1.3 define una derivación escalonada de secretos (*early, handshake, application traffic secrets*).
- ***Forward Secrecy* forzada**: ECDHE es obligatorio; comprometer la clave privada del servidor no permite descifrar sesiones previas si no se han revelado los secretos efímeros.
- **Separación de secretos por fase**: TLS 1.3 genera distintos *traffic secrets (handshake vs application*), lo que limita el alcance de la exposición si sólo un subconjunto se filtra.

3.2.5. Qué secretos se generan y por qué son relevantes

- ***ClientRandom / ServerRandom***: entradas de entropía pública usadas en la derivación.
- ***Premaster / Master secret*** (TLS ≤1.2): núcleo de la confidencialidad de la sesión.
- ***Traffic secrets*** (TLS 1.3): secretos específicos para cifrar el tráfico en cada fase.
- ***Keys & IVs***: derivados de los secretos anteriores para cifrado simétrico y autenticación.

3.2.6. Riesgo operacional asociado a la exposición de secretos

Si una implementación o proceso vuelca esos secretos a un fichero legible (p. ej. mediante **SSLKEYLOGFILE**), cualquiera con acceso a ese fichero y a una captura *pcap* podrá descifrar el tráfico TLS correspondiente. Por tanto, cualquier *keylog* debe tratarse como información altamente sensible y sólo permitirse en entornos controlados con acceso restringido y eliminación segura tras su uso.

3.3. Ejecución de pruebas

En las siguientes páginas de este informe se detalla una **metodología para efectuar las pruebas sobre el gestor de modelos vLLM**. No se incluye información para efectuar las pruebas con el gestor de Ollama ya que, en las pruebas realizadas con este, no se descubrió ninguna evidencia de generación de tráfico, y tampoco se detallan para Transformers ya que, en ese caso, es el desarrollador el que decide lo que se envía o no.

4. METODOLOGÍA DE CAPTURA Y ANÁLISIS DE TRÁFICO EN VLLM

4.1. Requerimientos

Para llevar a cabo la captura y posterior análisis es necesario capturar las claves iniciales de SSL/TLS que se almacenan en el sistema que instale cada *software*. La forma más sencilla de llevar a cabo esta tarea es mediante *dockerización*, estableciendo una ruta estática para el almacenamiento de dichas claves y permitiendo la captura de tráfico por una interfaz lógica creada previamente a la primera inicialización del sistema operativo.

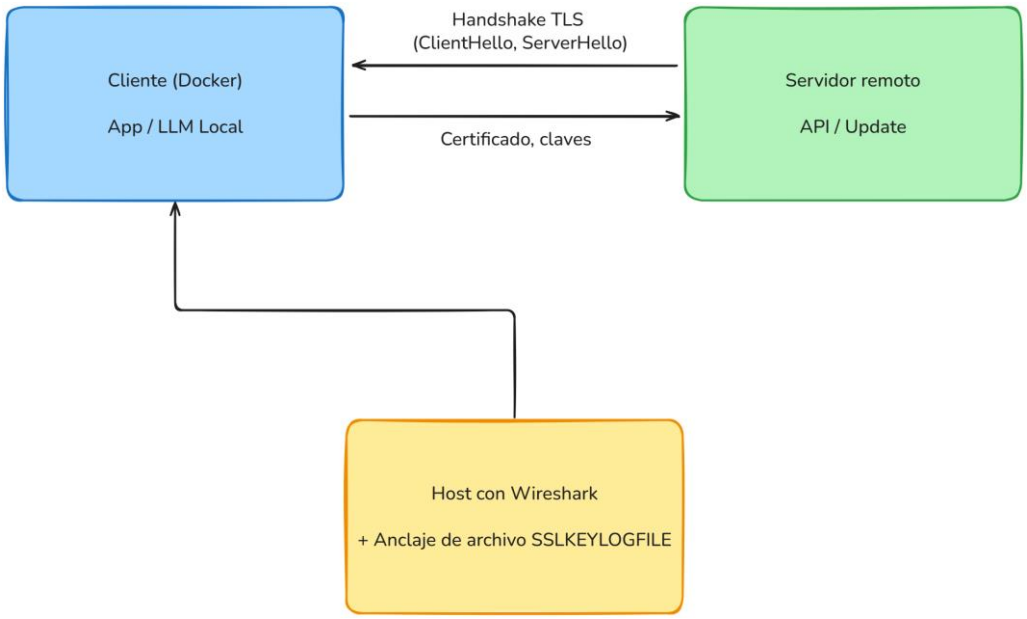

Ilustración 4.1: Esquema de captura de claves SSL

4.2. Configuración de la interfaz de red en Docker

Para analizar el tráfico generado por el gestor vLLM, se creó una red Docker tipo *bridge* dedicada, lo que permite aislar el tráfico del contenedor y facilitar su captura. Los comandos utilizados fueron los siguientes:

```
docker network create --driver bridge vllmnet

BRIDGE_NAME="br-$(docker network inspect vllmnet --format '{{ .Id }}' | cut -c1-12)"

echo $BRIDGE_NAME
```

- **"docker network create"**: Crea una red llamada **vllmnet** con *driver bridge*.
- **"BRIDGE_NAME"**: Extrae el identificador de la interfaz de red virtual creada por Docker, necesario para la captura de tráfico con tcpdump.

4.3. Captura de tráfico con tcpdump

Una vez identificada la interfaz de red virtual, se procede a capturar todo el tráfico que circula por ella, incluyendo las comunicaciones HTTPS. El comando utilizado fue:

```
sudo tcpdump -i $BRIDGE_NAME -w vllm_traffic_full.pcap
```

- **"-i $BRIDGE_NAME"**: Especifica la interfaz de red virtual de Docker como fuente de captura.
- **"-w vllm_traffic_full.pcap"**: Guarda el tráfico capturado en un archivo *pcap* para su posterior análisis con herramientas como Wireshark.

Nota: La captura se realizó en modo promiscuo, lo que permite registrar todo el tráfico que pasa por la interfaz, incluyendo paquetes que no están destinados al *host*.

4.4. Despliegue del contenedor vLLM

El contenedor de vLLM se desplegó utilizando Docker Compose, con una configuración específica para habilitar el *logging* de claves SSL y facilitar el descifrado del tráfico HTTPS. El archivo "docker-compose.yml" utilizado fue:

```
services:
 vllm:
  image: vllm/vllm-openai:latest
  container_name: vllm
  ports:
   - "8000:8000"
  runtime: nvidia
  environment:
   NVIDIA_VISIBLE_DEVICES: all
   VLLM_LOGGING_LEVEL: DEBUG
   SSLKEYLOGFILE: /tmp/sslkeys.log
  volumes:
   - ./sslkeys:/tmp
  command: >
   --model facebook/opt-125m
   --host 0.0.0.0
   --port 8000
networks:
 vllmnet:
  external: true
```

4.4.1. Detalles de la configuración:

- **"image: vllm/vllm-openai:latest":** Utiliza la imagen oficial de vLLM con soporte para OpenAI API.

- **"runtime: nvidia":** Habilita el uso de GPUs NVIDIA para la inferencia.
- **"SSLKEYLOGFILE: /tmp/sslkeys.log":** Guarda las claves de sesión TLS en un archivo, lo que permite descifrar el tráfico HTTPS capturado.
- **"volumes: - ./sslkeys:/tmp":** Monta un volumen local para persistir el archivo de claves SSL.
- **"command":** Especifica el modelo a cargar ("facebook/opt-125m") y la configuración de red ("*host* 0.0.0.0", "*port* 8000").

El contenedor se inicia con el siguiente comando:

```
docker compose up
```

4.5. Análisis del tráfico capturado

4.5.1. SSLKEYLOGFILE: formato, causas y consecuencias

4.5.1.1. Contexto en este análisis

Durante el análisis del contenedor vLLM se observó la existencia de un archivo de claves en la ruta montada "./sslkeys" (ver configuración en sección 4.4). A continuación, se detalla por qué ese fichero permite descifrar trazas y qué debe documentarse en una auditoría.

4.5.1.2. ¿Qué es SSLKEYLOGFILE?

SSLKEYLOGFILE es una convención (variable de entorno) que algunas librerías y binarios reconocen para volcar secretos de sesión en texto. Las entradas típicas siguen el patrón "LABEL <*client_random*> <*secret_hex*>" (por ejemplo, *CLIENT_RANDOM* ...). En TLS 1.3 los campos o *labels* hacen referencia a *traffic secrets* específicos (*handshake* / *application*). Este formato está pensado para la depuración y el aseguramiento de la calidad en entornos controlados.

4.5.1.3. Formato del SSLKEYLOGFILE

Cada línea del fichero sigue el formato:

```
<label> <client_random_hex> <secret_hex>\n
```

- ***label***: cadena textual que identifica el tipo de secreto. Ejemplos comunes:
  - CLIENT_RANDOM (TLS ≤ 1.2, típico).
  - CLIENT_HANDSHAKE_TRAFFIC_SECRET (TLS 1.3).
  - SERVER_HANDSHAKE_TRAFFIC_SECRET (TLS 1.3).
  - CLIENT_TRAFFIC_SECRET_0 (TLS 1.3, tráfico de aplicación).
  - SERVER_TRAFFIC_SECRET_0 (TLS 1.3).
  - EARLY_EXPORTER_SECRET, EXPORTER_SECRET y variantes según biblioteca o versión (menos frecuentes).
- ***client_random_hex***: *hex* del *ClientRandom* (valor de 32 *bytes* = 64 *hex* dígitos) que identifica la sesión o *handshake* con el que el secreto se corresponde.
- ***secret_hex****: secret* en *hex* cuyo tamaño varía con la función *hash* del *key schedule* (por ejmplo, SHA-256 → 32 *bytes* → 64 *hex* dígitos; SHA-384 → 48 *bytes* → 96 *hex*

dígitos). En TLS ≤1.2, para *CLIENT_RANDOM* este *secret* suele ser el *master secret*; mientras que en TLS 1.3 será el *secret* concreto (*traffic secret*) nombrado por el campo *label*.

4.5.1.4. Archivo capturado en la prueba

```
# TLS secrets log file, generated by OpenSSL / Python

SERVER_HANDSHAKE_TRAFFIC_SECRET
9c126547458a7a311175a1a28c4836170c7d3d73840b9a34af9a0f0433928<...>

EXPORTER_SECRET 9c126547458a7a311175a1a28c4836170c7d3d73840b9<...>

SERVER_TRAFFIC_SECRET_0 9c126547458a7a311175a1a28c4836170c7d3d73840b9a34af9<...>

CLIENT_HANDSHAKE_TRAFFIC_SECRET
9c126547458a7a311175a1a28c4836170c7d3d73840b9a34af9a0f0433928<...>

CLIENT_TRAFFIC_SECRET_0 9c126547458a7a311175a1a28c4836170c7d3d73840b9a34af9<...>
```

4.5.1.5. ¿Por qué aparece en entornos Docker?

La variable pudo activarse por varias opciones: (a) ajuste deliberado en el "docker-compose.yml", (b) una imagen con depuración o *debugging* habilitado, o (c) una práctica de pruebas que montó un volumen para dar persistencia al *keylog*.

Si el fichero se monta en un volumen compartido con el *host*, su persistencia y exposición aumentan el riesgo (por ejemplo, la ruta "/tmp" montado del *host*).

4.5.1.6. Consecuencias prácticas para la confidencialidad

- **Descifrado retroactivo**: si un atacante obtiene el *keylog* y posee un fichero *pcap* del tráfico, podrá descifrar sesiones pasadas recogidas en el *pcap*.
- **Erosión del sistema de cifrado PFS** (Perfect Forward Secrecy) en la práctica: aunque TLS 1.3 ofrezca *forward secrecy* contra compromisos de la clave privada, un *keylog* que contenga *traffic secrets* efímeros anula este beneficio para las sesiones explícitamente volcadas.
- **Implicación para la auditoría**: la presencia del fichero debe figurar en el informe con: ruta, permisos, propietario, fecha/hora de creación y si el fichero fue copiado o transmitido.

4.5.2. Descifrado de la traza

Dentro de Wireshark es necesario acceder al menú de preferencias para importar las claves de SSL/TLS.

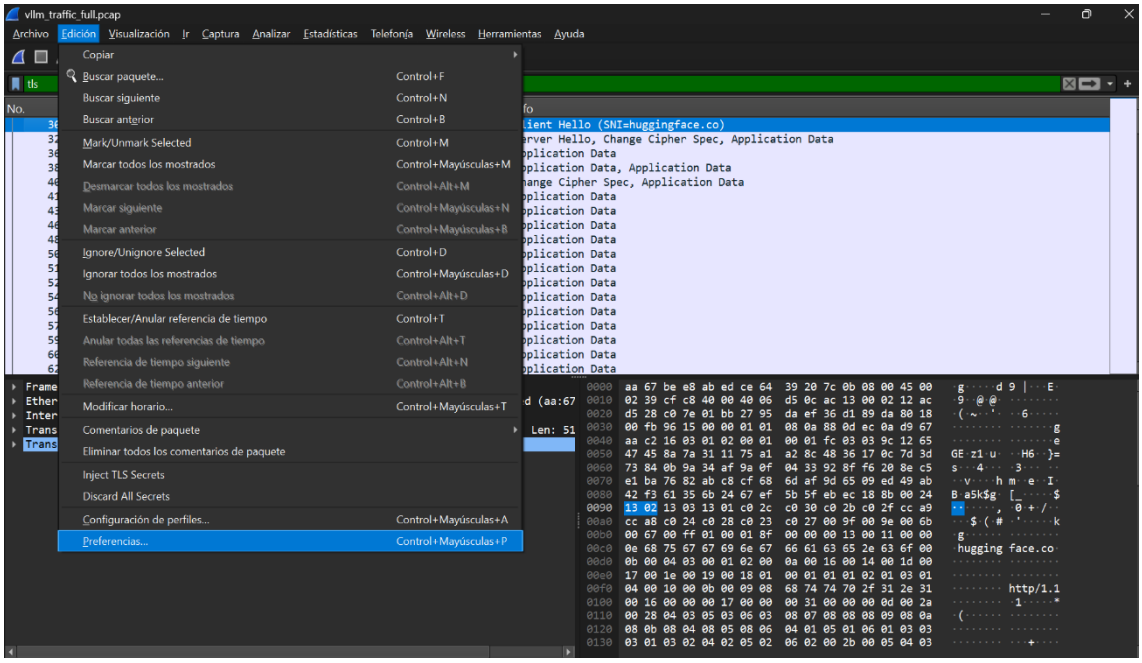

Ilustración 4.2: Menú de preferencias de Wireshark

Dentro del menú de preferencias se selecciona la pestaña de TLS, y se selecciona "Explorar" en el apartado de *(Pre)-Master-Secret log filename*, seleccionando así el archivo de claves.

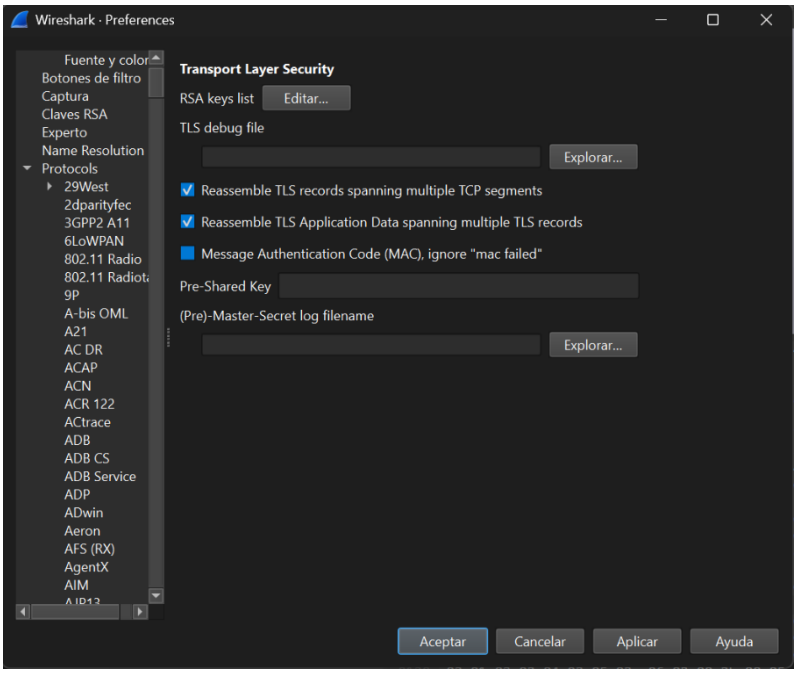

Ilustración 4.3: Menú de Transport Layer Security de Wireshark

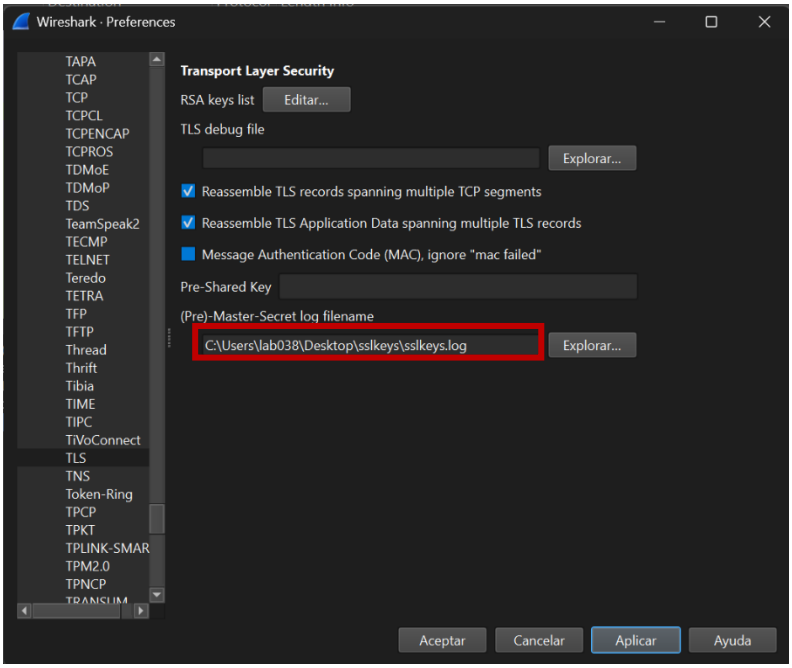

Ilustración 4.4: Carga de archivo de claves maestras en Wireshark

A continuación, se hace clic sobre “Aplicar” y “Aceptar”, de manera que Wireshark es capaz de descifrar el tráfico TLS de la captura.

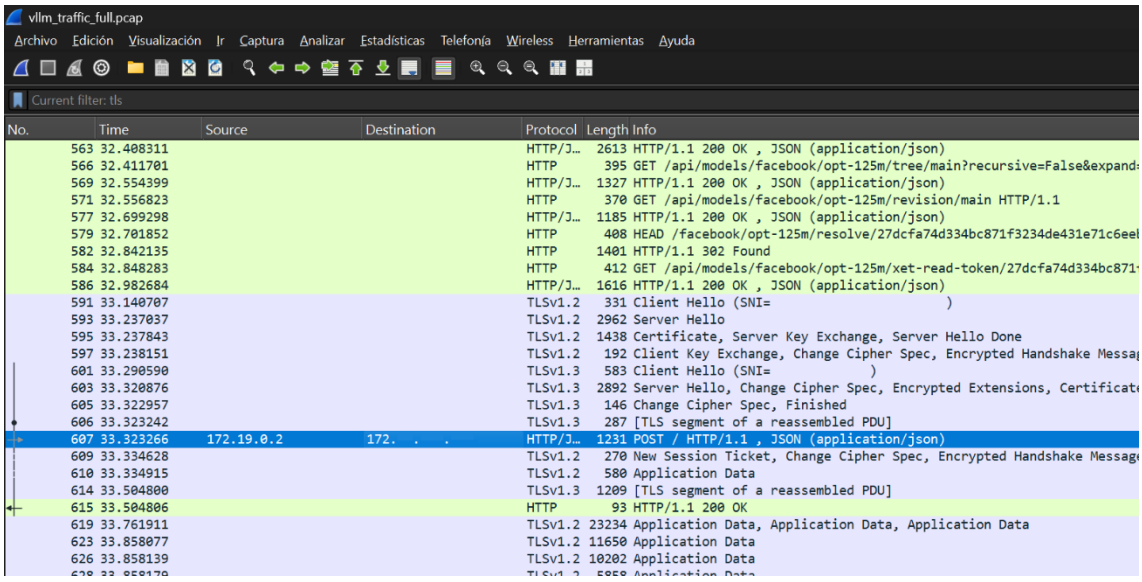

Ilustración 4.5: Trazas HTTPS descifradas en Wireshark

El análisis del archivo “vllm_traffic_full.pcap” reveló que, además del tráfico esperado entre el cliente y la API de vLLM, el contenedor envía telemetría y metadatos del sistema a un servidor externo ubicado en San Francisco. Esta comunicación se realiza mediante TLS/HTTPS (JSON sobre HTTP seguro), lo que inicialmente dificultó su identificación. Sin embargo, gracias a la captura de claves SSL (“sslkeys.log”), fue posible descifrar y analizar el contenido de estos paquetes.

4.5.3. Información filtrada en la traza

La traza descifrada contiene los siguientes datos sensibles del sistema y configuración de vLLM (Los datos a continuación han sido aleatorizados para preservar la confidencialidad):

4.5.3.1. Información general del sistema

- **UUID del sistema**: 600c2d56-0482-44d8-a6bb-eff74515a2f1
- **Proveedor**: CENSURADO
- **Fuente**: production-docker-image
- **Contexto**: ENGINE_CONTEXT
- **Tiempo de *log***: 1756885827045234944

4.5.3.2. CPU

- **Núcleos:** 48
- **Modelo**: Intel Xeon Platinum 8490H
- **Familia/Modelo/*Stepping***: 6, 143, 1
- **Arquitectura**: x86_64

4.5.3.3. Memoria

- **Memoria RAM total**: 2251799813685248 *bytes* (≈2.25 TB)

4.5.3.4. Sistema y plataforma

- **Plataforma**: Linux-5.15.0-1050-generic-x86_64-with-glibc2.35
- **Versión vLLM**: 0.9.2

4.5.3.5. Configuración del modelo vLLM

- **Arquitectura del modelo**: LLaMAForCausalLM
- **Tipo de datos**: torch.bfloat16
- **Tamaño de tensor paralelo**: 2
- **Tamaño de bloque**: 32
- **Cuantización**: null
- **Tipo de caché KV**: auto
- **LoRA habilitado**: false
- **Caché de prefijos habilitado**: true
- **Modo *eager* forzado**: false
- ***Custom Reduce* deshabilitado**: false

4.5.3.6. Variables de entorno (Parcial)

```
{
  "VLLM_USE_MODELSCOPE": false,
  "VLLM_USE_TRITON_FLASH_ATTN": true,
  "VLLM_ATTENTION_BACKEND": null,
  "VLLM_USE_FLASHINFER_SAMPLER": null,
  "VLLM_PP_LAYER_PARTITION": null,
```

```
  "VLLM_USE_TRITON_AWQ": false
}
```

4.5.3.7. Detalles de red

- Protocolo: TLS/HTTPS
- IP origen: 172.19.0.2 (red interna del contenedor)
- IP destino: 172.XXX.XXX.XXX (servidor externo en San Francisco)
- Puerto destino: 443 (HTTPS)

Investigando la IP en un motor de búsqueda de activos en Internet como Shodan, se encuentra la siguiente información:

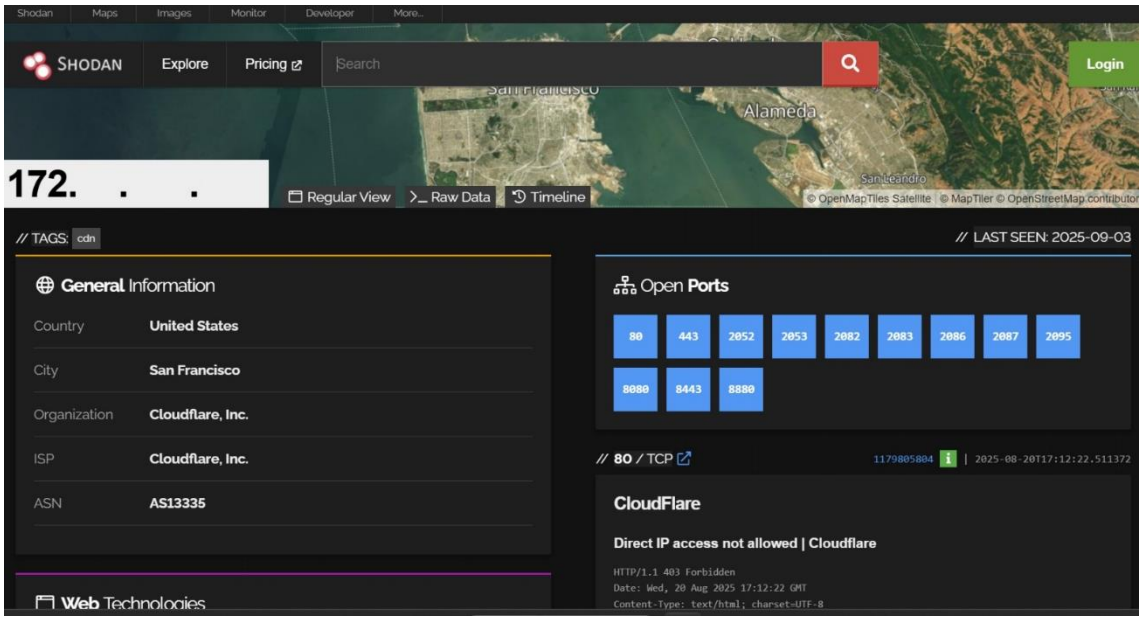

Ilustración 4.6: IP del servidor de destino en Shodan

4.5.4. Implicaciones de seguridad

La filtración de estos datos representa un riesgo de privacidad y seguridad del usuario, ya que expone:

- **Información única del *hardware*** (UUID, modelo de CPU/GPU, memoria), que podría utilizarse para identificar o rastrear el sistema.
- **Configuración detallada del entorno de ejecución**, información que podría utilizarse para la explotación de vulnerabilidades específicas.
- **Metadatos de la infraestructura**, que podrían ser utilizados para perfilar el entorno de producción.

4.6. Consideraciones de seguridad y privacidad según los datos obtenidos

4.6.1. Prevención del uso de telemetría

Según la documentación oficial de vLLM [10], el comportamiento de envío de datos se puede desactivar mediante la siguiente configuración de variables de entorno:

```
# Any of the following methods can disable usage stats collection
export VLLM_NO_USAGE_STATS=1
export DO_NOT_TRACK=1
mkdir -p ~/.config/vllm && touch ~/.config/vllm/do_not_track
```

4.6.2. Recomendaciones adicionales

- Deshabilitar la telemetría siempre en entornos de producción o con datos sensibles.
- Auditar el tráfico saliente de los contenedores para detectar comunicaciones no autorizadas.
- Utilizar *firewalls* o políticas de red para bloquear el tráfico a direcciones IP no esenciales.
- Revisar periódicamente las actualizaciones de vLLM y su documentación, ya que los comportamientos de telemetría pueden cambiar entre versiones.
- Cifrar y proteger los archivos de *logs* y claves SSL (“sslkeys.log”), y eliminarlos una vez finalizado el análisis.

ANEXO A: FORMATO Y EJEMPLOS DE SSLKEYLOGFILE

Formato (ejemplos conceptuales)

- TLS ≤ 1.2 (ejemplo):

```
CLIENT_RANDOM <client_random_hex> <master_secret_hex>
```

- TLS 1.3 (ejemplo):

```
CLIENT_HANDSHAKE_TRAFFIC_SECRET <client_random_hex> <secret_hex>
SERVER_TRAFFIC_SECRET_0 <client_random_hex> <secret_hex>
CLIENT_TRAFFIC_SECRET_0 <client_random_hex> <secret_hex>
```

Las etiquetas exactas pueden variar según la implementación y la versión de la librería. Tratar este fichero como alto secreto.

ANEXO B

Herramientas utilizadas para el análisis

- **Wireshark**: Para inspeccionar y descifrar el tráfico *pcap* utilizando el archivo "ssl-keys.log".
- **jq**: Para analizar y formatear los *payloads* JSON extraídos.
- **Docker**: Para lanzar el sistema operativo que contiene vLLM y poder capturar el tráfico por su interfaz.

BIBLIOGRAFÍA

[1] HuggingFace, "HuggingFace - The AI community," [Online]. Available: https://huggingface.co/. [Accessed 03 09 2025].

[2] Ollama, "Download Ollama on Linux," [Online]. Available: https://ollama.com/download/linux. [Accessed 03 09 2025].

[3] vLLM Team, "Quickstart vLLM," [Online]. Available: https://docs.vllm.ai/en/v0.9.2/getting_started/quickstart.html#openai-completions-api-with-vllm. [Accessed 03 09 2025].

[4] Ollama, "Github Ollama," [Online]. Available: https://github.com/ollama/ollama?tab=MIT-1-ov-file#readme. [Accessed 29 09 2025].

[5] vLLM Team, "Paged Attention - vLLM," 25 08 2025. [Online]. Available: https://docs.vllm.ai/en/latest/design/paged_attention.html. [Accessed 03 09 2025].

[6] vLLM, "Github vLLM," [Online]. Available: https://github.com/vllm-project/vllm?tab=Apache-2.0-1-ov-file#readme. [Accessed 29 09 2025].

[7] HuggingFace, "Transformers - HuggingFace," [Online]. Available: https://huggingface.co/docs/transformers/index. [Accessed 03 09 2025].

[8] HuggingFace, "Github Transformers," [Online]. Available: https://github.com/huggingface/transformers?tab=Apache-2.0-1-ov-file#readme. [Accessed 29 09 2025].

[9] Cloudflare, "¿Qué ocurre durante un protocolo de enlace TLS?," [Online]. Available: https://www.cloudflare.com/es-es/learning/ssl/what-happens-in-a-tls-handshake/. [Accessed 15 09 2025].

[10] vLLM Team, "Usage Stats Collection - vLLM," 28 08 2025. [Online]. Available: https://docs.vllm.ai/en/latest/usage/usage_stats.html#opting-out. [Accessed 03 09 2025].