Universidad de Valladolid

Escuela de Ingeniería de Segovia
Grado en Informática de Servicios y Aplicaciones

Detección de vulnerabilidades de dominios web

Alumno: Iván Jiménez Moreno

Tutores: Juan José Álvarez Sánchez

Detección de vulnerabilidades de dominios web

Iván Jiménez Moreno

Agradecimientos

“A mis padres y hermana, por apoyarme en todo el transcurso de mi carrera universitaria, en la que sin ellos y su gran apoyo que me han dado durante todos estos años, esta meta habría sido más dura y dificil de alcanzar”

Índice general

Lista de Imágenes

Lista de Tablas

1. Introducción

Las aplicaciones web son cada vez más complejas y están expuestas públicamente de manera constante, lo que las convierte en un objetivo perfecto para los atacantes. Detectar vulnerabilidades en este tipo de entornos es una tarea fundamental al igual que compleja si se quiere mantener la seguridad de los sistemas y evitar problemas que pueden terminar tanto en sanciones legales como en pérdidas económicas.

Aunque ya hay muchas herramientas que permiten hacer análisis de seguridad de forma automática, la mayoría se basan en patrones estáticos. Eso significa que, si se encuentran con un ataque que no está en su lista, probablemente no puedan llegar a ser detectados. Además, no suelen aprovechar las ventajas que ofrecen hoy en día la inteligencia artificial.

Este Trabajo de Fin de Grado propone el desarrollo de una aplicación web que permite detectar vulnerabilidades de forma automática, tanto en el dominio principal como en cada una de las subrutas que este pueda tener. Para ello, se combinan técnicas tradicionales de pentesting con modelos tanto de machine learning como de lenguaje natural. El sistema no se limita a escanear, sino que también analiza y genera recomendaciones usando un modelo de clasificación de payloads y un LLM (modelo de lenguaje) basado en LLaMA3, integrado a través del framework de Ollama.

1.2 Motivación del proyecto

La motivación principal que me impulsa a realizar este proyecto parte del interés por unir dos campos clave en la tecnología actual, que son la **seguridad ofensiva** y la **inteligencia artificial**. En el mundo profesional, hacer pruebas de seguridad (conocidas como tareas de pentesting) de forma manual requiere tiempo y conocimientos técnicos avanzados, lo cual puede suponer un problema para muchas organizaciones pequeñas que no cuentan con recursos suficientes para abordar estas tareas.

Este trabajo me ha servido como una oportunidad para aprender en profundidad sobre tecnologías actuales como FastAPI, Docker o MongoDB, y para explorar el procesamiento de texto web llevado a cabo con BeautifulSoup. También ha sido una forma práctica de entender cómo se pueden integrar modelos de lenguaje como los que ofrece Ollama, siempre teniendo en cuenta los aspectos éticos y responsables de su uso.

2. Ámbito de aplicación

2.1 Contexto del problema

Vivimos en una época en la que cada vez hay más servicios digitales y, con ellos, crece también el número de posibles puntos débiles en las aplicaciones web. Esto ha provocado que sea necesario contar con sistemas capaces de encontrar vulnerabilidades rápidamente y de forma precisa. Aunque existen herramientas clásicas como OWASP ZAP, Nikto o Sn1per, en muchos casos sus análisis se quedan cortos porque se ven limitados en algunas circunstancias.

Este proyecto parte precisamente de esa limitación. El objetivo es crear una solución más flexible y adaptativa, que no solo detecte vulnerabilidades conocidas, sino que también aprenda a reconocer nuevos patrones maliciosos gracias al uso de inteligencia artificial.

En resumen, buscamos que este nuevo sistema entienda mejor lo que ocurre en la web/dominio que analizamos, en lugar de simplemente buscar coincidencias exactas con patrones que ya existen dado a que puede dar lugar a confusión en algunos casos.

2.2 Entorno en el que se aplica

Este proyecto se sitúa dentro del campo del pentesting, es decir, en la parte de la ciberseguridad ofensiva que se encarga de detectar fallos en aplicaciones web antes de que lo haga un atacante. En concreto, la herramienta desarrollada está pensada para las fases de reconocimiento y explotación de vulnerabilidades.

El funcionamiento es bastante directo en donde el usuario encargado de interactuar con la aplicación introduce un dominio o una URL, y el sistema realiza un análisis semántico de las rutas que encuentra en esa página. A partir de ahí, se buscan posibles patrones que puedan indicar la presencia de una vulnerabilidad.

Para ello, utilizaremos un modelo de Machine Learning que ha sido entrenado previamente con descripciones y payloads reales, organizados por tipo de vulnerabilidad (por ejemplo, XSS, SQLi o RCE). Gracias a este modelo, el sistema puede identificar amenazas y clasificarlas automáticamente, sin necesidad de intervención humana en esa parte del proceso.

Pero el análisis no se queda ahí. Una vez detectadas las posibles vulnerabilidades, se genera también una recomendación personalizada gracias a un modelo de lenguaje natural (LLM) basado en LLaMA3. Este modelo se ejecuta de forma local a través de Ollama y se encarga de explicar el problema y sugerir una serie de mitigaciones recomendables que aconsejan y orientan al usuario de cara a poder resolver dichas vulnerabilidades que se han encontrado, todo en lenguaje claro y algo técnico.

2.3 Justificación de la utilidad

Este trabajo busca ir un paso más allá del típico escaneo de seguridad, apostando por una combinación entre análisis tradicional y técnicas de inteligencia artificial. Esta mezcla permite no solo encontrar más fallos, sino también entenderlos mejor y actuar más rápido.

Entre las ventajas más importantes que ofrece esta solución están:

- **Una mejor detección de patrones**: como el modelo se ha entrenado con más de 40.000 ejemplos únicos, es capaz de reconocer técnicas poco comunes o intentos de evasión que muchas herramientas tradicionales pasarían por alto.
- **Menor esfuerzo manual por parte del usuario**: gracias al modelo de machine learning, gran parte del trabajo de análisis se realiza de forma automática, lo que ahorra mucho tiempo y reduce la carga sobre los equipos de seguridad.
- **Recomendaciones útiles**: el sistema no solo detecta vulnerabilidades, sino que también explica cómo podremos actuar frente a estas, lo que resulta muy útil para quienes no tienen mucha experiencia en el tema y quieran poder solucionar dichas vulnerabilidades.
- **Escalabilidad**: como la base de datos de entrenamiento se puede ir actualizando, el sistema es capaz de adaptarse a nuevas amenazas que vayan apareciendo en el futuro.

Por ello, podemos verlo como que nuestro proyecto pone en práctica técnicas modernas de IA para resolver un problema real en el ámbito de la ciberseguridad ofensiva, demostrando cómo la tecnología puede mejorar tanto la detección como la interpretación de vulnerabilidades en aplicaciones web.

3. Cuestiones metodológicas

3.1 Enfoque y modelo de desarrollo

El enfoque que hemos seguido para la correcta elaboración de este proyecto ha sido un enfoque Agile de desarrollo iterativo e incremental. Esto quiere decir que, en lugar de intentar construir todo de una vez, se ha ido avanzando poco a poco, añadiendo funciones y validando los resultados en cada paso de la elaboración del proyecto.

Gracias a esto, nos ha sido más fácil identificar errores a tiempo, probar mejoras y adaptarnos a nuevas ideas o necesidades que surgían durante el desarrollo. Por ejemplo, en una de las primeras versiones del sistema, se planteó usar expresiones regulares para detectar vulnerabilidades, pero tras probarlo en varios dominios, se vio que no era suficientemente flexible y que los resultados obtenidos no eran los esperados.

Dado a este enfoque iterativo, se pudo sustituir esa técnica y vimos que sustituyendo esta forma de detectar por vulnerabilidades por un modelo de machine learning entrenado con patrones sintéticos resultó ser una solución óptima sin tener que rehacer todo el sistema desde cero.

A diferencia de otros modelos, como puede ser el modelo en cascada, que es más rígido, aquí se ha trabajado con más flexibilidad y margen de ajuste.

A nivel metodológico, se ha seguido una estructura basada en objetivos funcionales, y se han utilizado distintos diagramas para representar de forma visual el funcionamiento del sistema: casos de uso, diagramas de secuencia, etc. Todo esto ha ayudado a organizar y clarificar tanto el diseño como la implementación.

3.2 Fases del proyecto

El desarrollo del sistema se ha estructurado en varias fases, cada una con un objetivo claro. A continuación, procedemos a exponer cada una de las diferentes fases sobre las que hemos trabajado:

1. **Análisis del problema y recopilación de requisitos**
   En esta fase definimos claramente el propósito del sistema. Identificamos el ámbito de aplicación (el escaneo de vulnerabilidades en dominios web) y se analizaron las necesidades que debía cubrir.
   - Por ejemplo, se estableció que el sistema debía permitir introducir una URL, detectar subrutas internas y analizar su contenido en busca de fallos.
   - También se clasificaron los requisitos: funcionales (como registrar un dominio), no funcionales (por ejemplo, que sea usable desde cualquier navegador) y de información (qué datos debían recogerse y almacenarse).
2. **Implementación**
   El desarrollo comienza a realizarse en **Jupyter Notebook**, donde construiremos nuestro modelo de machine learning, usando tanto datos sintéticos como reales.
   - En esta fase probamos técnicas de vectorización y predicción para ver cómo respondía el modelo ante ejemplos de ataques como pueden ser ataques de XSS o SQLi.
     Después trasladamos la lógica al back-end, en donde se creó la interfaz en React(front-end) y se integró con MongoDB para guardar todo de forma estructurada.
3. **Diseño del sistema**
   Fase en la cual se elaboran varios diagramas para representar cómo debe comportarse el sistema. También se define su arquitectura para la posterior elaboración de la página web que iba a recoger lo que viene siendo la funcionalidad de nuestro proyecto: una parte constituye el back-end que se encargaría del análisis, seguidamente una interfaz web como es el front-end al que asociaremos con nuestro back-end y por último, una base de datos para guardar los resultados obtenidos.
   - Aquí, por ejemplo, se decidió usar **FastAPI** para la API, **React** para el front-end y **MongoDB** para la base de datos.
4. **Pruebas y validación**
   Aquí lo que hacemos es comprobar que cada parte del sistema funcione correctamente de forma individual y también que todo funcione de forma integrada.

- Por ejemplo, se probó que el sistema pudiera escanear un dominio real como *http://localhost/dvwa* y que el resultado apareciera correctamente en la interfaz, con su score/umbral de riesgo y su recomendación correspondiente.
  Además, se validaron los resultados del modelo usando casos reales y patrones conocidos.

5. **Documentación**
   Finalmente, elaboramos la memoria del proyecto, incluyendo diagramas, tablas de requisitos, capturas de pantalla y fragmentos de código.

   - Dentro de esta, destacar el apartado de manual de usuario para que cualquier usuario que emplease nuestra aplicación (o nuestro cuaderno de Jupyter) pudiera entender cómo usar la herramienta, incluso sin conocimientos técnicos avanzados.

4. Herramientas empleadas

Durante el desarrollo del proyecto se han utilizado diferentes herramientas, tanto para programar como para documentar, modelar o realizar pruebas. Todo el sistema se construyó en un entorno con Windows 10 como sistema operativo, y se utilizó Google Chrome como navegador principal. Este navegador no solo sirvió para acceder a documentación técnica durante el desarrollo, sino también para utilizar herramientas online como Mermaid Live Editor, que fue clave para crear varios de los diagramas que forman parte del proyecto de tal manera que son fáciles de interpretar de una forma rápida y visual.

El proyecto arrancó en un entorno de Jupyter Notebook, lo cual resultó ideal para la fase inicial de pruebas. Gracias a su entorno interactivo, fue posible ir probando pequeñas funciones, entrenar el modelo de Machine Learning y depurar errores sin necesidad de compilar todo el sistema. Para asegurar que no hubiese conflictos con versiones anteriores de librerías, se creó un entorno virtual exclusivo usando Anaconda Prompt, lo que permitió mantener las dependencias bajo control y evitar errores por incompatibilidad.

Para redactar esta memoria se empleó Microsoft Word, mientras que para representar gráficamente algunas de las funcionalidades del sistema se usó la herramienta StarUML. Esta herramienta fue útil para modelar concretamente el diagrama de casos de uso, en los que se muestran de forma clara las relaciones entre el usuario y las acciones que puede realizar en el sistema.

Una vez completadas las partes principales, el sistema se dividió en varios servicios utilizando Docker. Se crearon contenedores independientes para el backend y el frontend.

- Por ejemplo, el contenedor que ejecuta la interfaz web se desplegó en el puerto 8080, mientras que el backend desarrollado con FastAPI se expone en el puerto 8081. Esto permitió probar los endpoints del backend directamente desde Swagger UI, al mismo tiempo que se interactuaba con el frontend desde el navegador.

Finalmente, se eligió MongoDB como base de datos NoSQL para almacenar todos los datos relacionados con los análisis. En esta base se guarda información como los dominios escaneados, sus subrutas, las vulnerabilidades encontradas, los logs del proceso y las recomendaciones generadas automáticamente entre otros datos.

4.1 Lenguajes y librerías

El lenguaje principal utilizado en este proyecto ha sido Python, que ha servido tanto para construir el back-end como para desarrollar el modelo de machine learning y gestionar toda la lógica de análisis.

Entre las librerías de Python más importantes que se han empleado están:

- **pandas y numpy**: utilizadas para manipular y organizar los datos, especialmente durante el entrenamiento del modelo y la creación de los datasets.
- **scikit-learn**: clave en todo el proceso de preprocesamiento, entrenamiento y validación del modelo de clasificación.
- **joblib**: se ha usado para guardar el modelo entrenado en formato reutilizable, lo que permite cargarlo fácilmente desde el back-end.
- **requests y re**: para hacer peticiones HTTP y trabajar con expresiones regulares durante la exploración de contenido web.
- **beautifulsoup4**: muy útil para recorrer el HTML de los dominios escaneados y extraer las subrutas accesibles automáticamente.
- **pymongo**: encargada de gestionar la conexión entre Python y la base de datos MongoDB.
- **uvicorn y fastapi**: empleadas para levantar la API REST del sistema, con soporte asíncrono y documentación automática integrada.
- **os, json y dotenv**: usadas para gestionar rutas de archivos, leer configuraciones desde archivos .env y trabajar con estructuras de datos serializadas.

Por ejemplo, durante el proceso de crawling, se usó beautifulsoup4 para identificar todos los enlaces internos de un dominio, y luego se procesaron con re para comprobar su formato antes de analizarlos con el modelo.

4.2 Frameworks y tecnologías

El proyecto ha incorporado varias tecnologías clave para su desarrollo y despliegue:

- **FastAPI**: se ha usado como framework principal para construir el back-end, permitiendo probar los endpoints de forma rápida y clara, con documentación automática gracias a Swagger UI.
- **React**: esta biblioteca de JavaScript se ha utilizado para el desarrollo del front-end. Dado que no se había trabajado antes en el grado, su integración implicó un aprendizaje autodidacta, sobre todo en lo referente a la conexión con la API y el renderizado de resultados en tiempo real.

  También su uso se debe a que es uno de los mayores Frameworks a nivel de desarrollo web punteros dentro del mercado laboral.

- **Docker**: se ha empleado para contenerizar tanto el front-end como el back-end. Esto ha permitido ejecutar cada parte del sistema de forma aislada y sin conflictos, desplegando el front-end (en el puerto 8080) y la API (en el puerto 8081).
- **MongoDB**: base de datos NoSQL utilizada para almacenar todos los resultados del sistema, desde las rutas escaneadas hasta los logs y las recomendaciones generadas por el modelo.

5. Estado del arte

Existen diversas herramientas para la detección de vulnerabilidades en aplicaciones web, siendo algunas de las más conocidas **OWASP ZAP**, **Nikto** y **Sn1per**. Estas soluciones tradicionales están enfocadas en análisis clásicos basados en firmas conocidas, cabeceras HTTP y configuraciones incorrectas, lo que les permite encontrar problemas típicos como directorios expuestos, formularios sin validación o versiones de software con fallos conocidos.

- **OWASP ZAP** es una de las más utilizadas por la comunidad. Cuenta con una interfaz gráfica amigable, análisis pasivo y activo, e integración con navegadores. Es compatible con sistemas **Linux y Windows**, aunque en este último, algunos scripts que se incluyen en su descargable, pueden ser bloqueados por antivirus por considerarse "archivos sospechosos" al tratarse de herramientas de prueba de seguridad.

Figura 5.1: Herramienta OWASP ZAP

- **Nikto**, por su parte, es una herramienta basada en línea de comandos que analiza servidores web buscando configuraciones peligrosas, archivos sensibles o vulnerabilidades conocidas. Está diseñada principalmente para **entornos Linux**, aunque puede ejecutarse en Windows teniendo Perl instalado. Su funcionamiento en este sistema suele ser más limitado.

Figura 5.2: Herramienta Nikto

- **Sn1per** es una suite (conjunto de herramientas y aplicaciones diseñadas para automatizar procesos y tareas) que integra varias herramientas de reconocimiento, escaneo y generación de informes. Aunque potente, está pensada para entornos **Linux**, y solo puede usarse en Windows mediante WSL o entornos virtualizados. Además, al contener payloads y scripts ofensivos, suele generar alertas en los sistemas antivirus de los equipos.

Figura 5.3: Herramienta SN1PER

En cuanto al uso de inteligencia artificial, su aplicación en el ámbito de la ciberseguridad aún está en una fase relativamente inicial. Existen algunos enfoques experimentales basados en modelos de clasificación de texto o detección de anomalías, pero su integración directa en herramientas prácticas es todavía poco habitual. Además, los sistemas actuales no suelen ofrecer una capa de interpretación en lenguaje natural que facilite la comprensión de los resultados por parte del usuario.

En comparación con estas soluciones, el sistema que hemos desarrollado en este proyecto plantea un enfoque híbrido. Combina el escaneo semántico, aprendizaje automático y generación de recomendaciones automáticas mediante modelos de lenguaje (LLM), permitiendo una detección más flexible y comprensible. A diferencia de herramientas como ZAP o Nikto, no se limita a buscar coincidencias exactas, sino que interpreta el contenido de las páginas web en busca de patrones maliciosos utilizando embeddings y clasificación inteligente.

Además, la posibilidad de que el modelo de lenguaje explique el riesgo y sugiera acciones correctivas convierte este sistema en una herramienta más cercana a entornos reales.

Mostraremos a continuación una tabla en la que apreciaremos y compararemos las diferentes ventajas que tiene nuestro sistema respecto de los anteriormente mencionados:

Herramienta	Análisis clásico	Soporte IA	Generación de recomendaciones	Análisis semántico	Linux	Windows
OWASP ZAP	✓	✗	✗	✗	✓	✓
Nikto	✓	✗	✗	✗	✓	⚠
Sn1per	✓	✗	✗	✗	✓	⚠
Proyecto propio	✓	✓	✓	✓	✗	✓

Cuadro 5.1: Comparativa entre herramientas

(⚠ Nikto): Aunque puede ejecutarse en Windows con Perl instalado, está diseñado principalmente para Linux. En Windows suele requerir varias configuraciones extra.

(⚠ Sn1pper): Desarrollado principalmente para Linux. Puede usarse en Windows a través de WSL (Subsistema Linux), pero no se recomienda dado a su compatibilidad limitada.

6. Presupuesto

6.1 Estimación del esfuerzo

Para hacerme una idea del trabajo que ha supuesto este proyecto, hemos hecho una estimación de las horas que le hemos dedicado. Aunque no hemos tenido un control exacto del tiempo realizado por cada tarea, sí que hemos llevado un control más o menos aproximado según las semanas y las tareas que íbamos haciendo.

Durante el desarrollo, hemos estado trabajando unas 3-4 horas al día entre semana, más algunas sesiones intensas los findes cuando había que probar cosas o avanzar partes clave. En total, calculamos que hemos invertido unas **360 horas reales** a lo largo de los meses, repartidas entre código, pruebas, documentación y todas las reflexiones realizadas antes de ponernos con cada tarea.

6.2 Planificación temporal

Se muestra a continuación un cronograma temporal que refleja la división del tiempo que hemos empleado dentro de nuestro proyexto para la realización de cada una de las diferentes tareas llevadas a cabo dentro de cada una de las fases del proyecto.

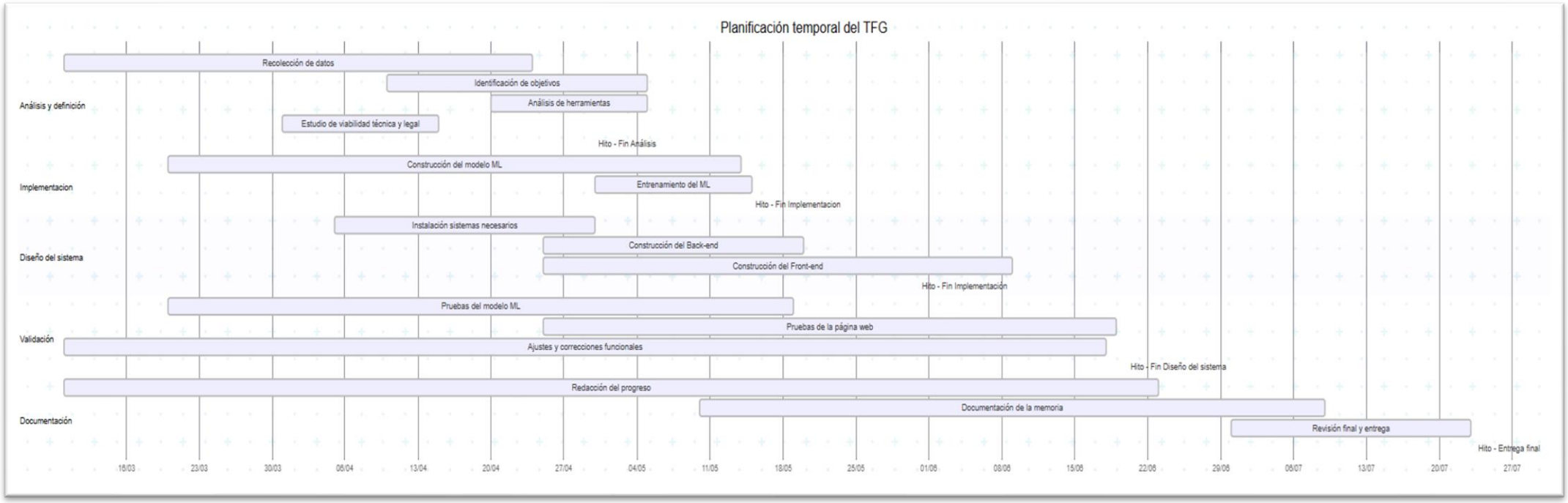


Figura 6.2.1: Cronograma de la planificación temporal

6.3 Presupuesto Económico

Para la estimación del presupuesto económico que se ha realizado en este proyecto, únicamente se contará de los recursos humanos en un ámbito real. Esto es porque se ha trabajado con herramientas libres u Open Source, accesible para cualquiera. Sólo existirán limitaciones de rendimiento del hardware con el que el trabajador desee trabajar.

6.4 Recursos Humanos

El trabajo realizado se refleja en 4 perfiles autónomos: Desarrollador de Front-end, Desarrollador de Back-end, Especialista en IA y Consultor de Ciberseguridad.

Podríamos considerar que los 2 primeros serían perfiles Junior mientras que los dos últimos serían perfiles Senior

Habrá que tener en cuenta también la diferencia entre sueldo Neto y Bruto, por lo que estimaremos lo siguiente:

- IRPF[20]: 10%
- Cuota de Autónomos: 80 €/mes

Y el tiempo de trabajo de media realizado es de 4h/día, haciendo que:

- Mes: 80h total, 20h/semana equitativamente.
- Proyecto: 360h equitativamente

Si esas **360 horas** se repartieran entre los **4 perfiles principales implicados** (desarrollador back-end, front-end, experto en IA y técnico en ciberseguridad), cada uno habría trabajado aproximadamente **90 horas**

En el caso del **Desarrollador de Back-End**, tendríamos que tener en cuenta las tarifas (Cuadro 6.1) con las que estimar:

	Junior	**Intermedio**	**Senior**
Tarifa Mínima €/h	18 €	35 €	68 €
Tarifa Máxima €/h	28 €	55 €	110 €

Cuadro 6.1: Tarifas para Desarrollador Back-End

A partir de la tarifa media y el tiempo de trabajo mencionado anteriormente, podremos estimar el coste total como Desarrollador de Back-End

Concepto	**Coste**
Tarifa Bruta Media	23 €/h
Total Bruto	2070€
Neto por hora (~90%)	~20.7€/h
Total Neto	~1550€

Cuadro 6.2: Estimación del coste total neto del Desarrollador Back-End

En el caso del **Desarrollador de Front-End**, tendríamos que tener en cuenta las tarifas (Cuadro 6.3) con las que estimar:

	Junior	**Intermedio**	**Senior**
Tarifa Mínima €/h	16 €	32 €	62 €
Tarifa Máxima €/h	25 €	50 €	95 €

Cuadro 6.3: Tarifas para Desarrollador Front-End

A partir de la tarifa media y el tiempo de trabajo mencionado anteriormente, podremos estimar el coste total como Desarrollador de Front-End

Concepto	**Coste**
Tarifa Bruta Media	20.5€/h
Total Bruto	1845€
Neto por hora (~90%)	~18.45€/h
Total Neto	~1350€

Cuadro 6.4: Estimación del coste total neto del Desarrollador Front-End

En el caso del **Especialista en Inteligencia Artificial**, tendríamos que tener en cuenta las tarifas (Cuadro 6.5) con las que estimar:

	Junior	**Intermedio**	**Senior**
Tarifa Mínima €/h	20 €	40 €	80 €
Tarifa Máxima €/h	35 €	60 €	120 €

Cuadro 6.5: Tarifas para Especialista en Inteligencia Artificial

A partir de la tarifa media y el tiempo de trabajo mencionado anteriormente, podremos estimar el coste total como Especialista en Inteligencia Artificial

Concepto	**Coste**
Tarifa Bruta Media	50 €/h
Total Bruto	4500 €
Neto por hora (~90%)	~45 €/h
Total Neto	~4050 €

Cuadro 6.6: Estimación del coste total neto del Especialista en Inteligencia Artificial

En el caso del **Consultor de Ciberseguridad**, tendríamos que tener en cuenta las tarifas (Cuadro 6.7) con las que estimar:

	Junior	**Intermedio**	**Senior**
Tarifa Mínima €/h	19 €	38 €	70 €
Tarifa Máxima €/h	30 €	58 €	100 €

Cuadro 6.7: Tarifas para Consultor de Ciberseguridad

A partir de la tarifa media y el tiempo de trabajo mencionado anteriormente, podremos estimar el coste total como Consultor de Ciberseguridad.

Concepto	**Coste**
Tarifa Bruta Media	48 €/h
Total Bruto	4320 €
Neto por hora (~90%)	~43,2 €/h
Total Neto	~3900 €

Cuadro 6.8: Estimación del coste total neto del Consultor de Ciberseguridad

6.5 Presupuesto Total

Una vez contabilizado lo anterior, procederemos a establecer cual sería el presupuesto total para nuestro proyecto:

	D.Front-End	**D.Back-End**	**Especialista IA**	**Consultor Ciber**	**Total**
Coste	1350€	1550€	4050 €	3900€	**10850 €**

Cuadro 6.9: Presupuesto total del proyecto

Estaríamos hablando de un presupuesto del trabajo realizado de unos **10850€**

7. Objetivos del sistema

7.1 Objetivo general

El objetivo genera de nuestro proyecto va a consistir en desarrollar una aplicación web capaz de escanear dominios, detectar vulnerabilidades comunes y ofrecer recomendaciones automatizadas, combinando técnicas de escaneo ofensivo con inteligencia artificial.

En este apartado se expondrán tanto los objetivos como los subobjetivos que se esperan alcanzar del proyecto que hemos desarrollado.

OBJ-01	**Gestión de los dominios**
Versión	1.0 (30/05/2025)
Autor	Iván Jiménez Moreno
Fuente	Iván Jiménez Moreno
Descripción	El sistema deberá gestionar adecuadamente la información relacionada con los dominios proporcionados por el usuario, permitiendo su registro, análisis y trazabilidad.
Subobjetivos	• **SOBJ-01:** Registro del dominio y sus rutas • **SOBJ-02:** Almacenamiento estructurado del análisis
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Cuando hablamos de la información de los dominios nos referimos a rutas, subrutas, vulnerabilidades, etc.

Cuadro 7.1: OBJ-01. Gestión de los dominios

SOBJ-01	**Registro del dominio y sus rutas**
Versión	1.0 (30/05/2025)
Autores	Iván Jiménez Moreno
Fuentes	Iván Jiménez Moreno
Descripción	El sistema deberá registrar el dominio introducido por el usuario y asociar las rutas web detectadas para su posterior análisis.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Se almacenarán los datos en la Base de Datos MongoDB con ID único por dominio

Cuadro 7.2: SOBJ-01. Registro del dominio y sus rutas

SOBJ-02	Almacenamiento estructurado del análisis
Versión	1.0 (30/05/2025)
Autores	Iván Jiménez Moreno
Fuentes	Iván Jiménez Moreno
Descripción	El sistema deberá almacenar de forma estructurada todos los datos recogidos durante el análisis del dominio, incluyendo rutas, logs y vulnerabilidades asociadas (y posibles mitigaciones en caso de que se encuentren vulnerabilidades en las rutas).
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Se almacenarán en la Base de Datos MongoDB en formato JSON

Cuadro 7.3: SOBJ-02. Almacenamiento estructurado del análisis

OBJ-02	Gestión de vulnerabilidades
Versión	1.0 (30/05/2025)
Autor	Iván Jiménez Moreno
Fuente	Iván Jiménez Moreno
Descripción	El sistema debe ser capaz de identificar y almacenar vulnerabilidades relevantes en función del análisis realizado sobre los dominios introducidos.
Subobjetivos	• **SOBJ-03:** Asociación semántica entre el texto de la web y las bases de datos OSINT • **SOBJ-04:** Relación entre vulnerabilidades y contexto web
Importancia	Muy Alta
Urgencia	Muy Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	-

Cuadro 7.4: OBJ-02. Gestión de vulnerabilidades

SOBJ-03	Asociación semántica entre web y bases OSINT
Versión	1.0 (30/05/2025)
Autores	Iván Jiménez Moreno
Fuentes	CVE, KEV, Exploit-DB, FAISS, Git, APIs de vulnerabilidades
Descripción	El sistema deberá analizar el contenido textual de la web y buscar coincidencias semánticas con vulnerabilidades existentes en bases OSINT utilizando embeddings (representaciones vectoriales de los datos).
Importancia	Muy Alta
Urgencia	Muy Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Requiere entrenamiento y evaluación del modelo de similitud

Cuadro 7.5: SOBJ-03. Asociación semántica entre web y bases OSINT

SOBJ-04	**Relación entre vulnerabilidades y contexto web**
Versión	1.0 (30/05/2025)
Autores	Iván Jiménez Moreno
Fuentes	CVE, KEV, Exploit-DB, Git, APIs de vulnerabilidades
Descripción	El sistema deberá establecer una relación entre los patrones detectados en los contenidos del sitio web y las descripciones/payloads de las fuentes OSINT, permitiendo identificar coincidencias relevantes mediante técnicas de similitud semántica
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	-

Cuadro 7.6: SOBJ-04. Relación entre vulnerabilidades y contexto web

OBJ-03	**Exploración automatizada de rutas y subrutas**
Versión	1.0 (30/05/2025)
Autor	Iván Jiménez Moreno
Fuente	Iván Jiménez Moreno
Descripción	El sistema deberá extraer de forma automática todas las rutas accesibles desde el HTML de una página web, como enlaces internos, y analizarlas individualmente.
Subobjetivos	• **SOBJ-05:** Detección de subrutas y formularios • **SOBJ-06:** Análisis individualizado por ruta
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Scraping mediante BeautifulSoup

Cuadro 7.7: OBJ-03. Exploración automatizada de rutas y subrutas

SOBJ-05	**Detección de subrutas y formularios**
Versión	1.0 (30/05/2025)
Autores	Iván Jiménez Moreno
Fuentes	HTML del dominio introducido, BeautifulSoup
Descripción	El sistema deberá recorrer el árbol DOM del sitio web para detectar enlaces internos, formularios y puntos de entrada potenciales.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Rutas almacenadas junto con estado HTTP y categoría

Cuadro 7.8: SOBJ-05. Detección de subrutas y formularios

SOBJ-06	Análisis individualizado por ruta
Versión	1.0 (30/05/2025)
Autores	Iván Jiménez Moreno
Fuentes	Rutas extraídas del scraping
Descripción	Cada ruta detectada será analizada de forma independiente por el motor de búsqueda semántica para encontrar vulnerabilidades asociadas.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Permite aislar rutas críticas de forma más efectiva

Cuadro 7.9: SOBJ-06 Análisis individualizado por ruta

OBJ-04	Generación de recomendaciones mediante LLM
Versión	1.0 (30/05/2025)
Autor	Iván Jiménez Moreno
Fuente	Iván Jiménez Moreno
Descripción	El sistema debe generar automáticamente recomendaciones preventivas basadas en los resultados del análisis y las vulnerabilidades detectadas, usando un modelo de lenguaje.
Subobjetivos	• **SOBJ-07:** Conversión de vulnerabilidades en consejos prácticos • **SOBJ-08**: Justificación técnica del riesgo detectado
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	LLM integrado con respuestas enriquecidas

Cuadro 7.10: OBJ-04. Generación de recomendaciones mediante LLM

SOBJ-07	Conversión de vulnerabilidades en consejos prácticos
Versión	1.0 (30/05/2025)
Autores	Iván Jiménez Moreno
Fuentes	LLM (Ollama, modelo local)
Descripción	El sistema generará automáticamente consejos de mitigación y recomendaciones prácticas a partir de los resultados de análisis.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	En lenguaje natural, con enfoque técnico y comprensible

Cuadro 7.11: SOBJ-07. Conversión de vulnerabilidades en consejos prácticos

SOBJ-08	Justificación técnica del riesgo detectado
Versión	1.0 (30/05/2025)
Autores	Iván Jiménez Moreno
Fuentes	Resultados del modelo LLM + logs de análisis
Descripción	El sistema explicará de forma técnica por qué una vulnerabilidad representa un riesgo, con base en los datos recogidos.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Mejora la trazabilidad y comprensión del riesgo real

Cuadro 7.12: SOBJ-08. Justificación técnica del riesgo detectado

OBJ-05	Recopilación y almacenamiento de datos
Versión	1.0 (04/06/2025)
Autor	Iván Jiménez Moreno
Fuente	Iván Jiménez Moreno
Descripción	El sistema debe permitir la recolección de datos desde múltiples fuentes (feeds, APIs, logs, repositorios) y almacenarlos correctamente para su uso posterior.
Subobjetivos	• **SOBJ-09:** Integración de fuentes heterogéneas • **SOBJ-010:** Validación y almacenamiento de datos recolectados
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Es el paso inicial para cualquier proceso de análisis o entrenamiento posterior.

Cuadro 7.13: OBJ-05. Recopilación y almacenamiento de datos

SOBJ-09	Integración de fuentes heterogéneas
Versión	1.0 (04/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Repositorios Git, APIs CVE/KEV, logs internos
Descripción	El sistema debe conectarse correctamente a fuentes variadas y heterogéneas, automatizando su integración para uso posterior.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Estas fuentes conforman la base de conocimiento del sistema.

Cuadro 7.14: SOBJ-09. Integración de fuentes heterogéneas

SOBJ-010	Validación y almacenamiento de datos recolectados
Versión	1.0 (04/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Datos obtenidos del proceso de scraping y feeds
Descripción	El sistema debe validar los datos recolectados (formato, integridad) y almacenarlos correctamente en una base de datos accesible.
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Aporta fiabilidad a los datos que usará el ML.

Cuadro 7.15: SOBJ-010. Validación y almacenamiento de datos recolectados

OBJ-06	Construcción y gestión del dataset para ML
Versión	1.0 (04/06/2025)
Autor	Iván Jiménez Moreno
Fuente	Iván Jiménez Moreno
Descripción	El sistema debe permitir construir datasets sintéticos etiquetados con ejemplos positivos y negativos a partir de los datos recolectados.
Subobjetivos	• **SOBJ-011:** Generación de muestras balanceadas • **SOBJ-012:** Enlace con la BBDD de conocimiento y etiquetas
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Este proceso es fundamental para alimentar el modelo de ML.

Cuadro 7.16: OBJ-06. Construcción y gestión del dataset para ML

SOBJ-011	Generación de muestras balanceadas
Versión	1.0 (04/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Datos procesados y etiquetados
Descripción	El sistema debe crear un conjunto de datos balanceado en cuanto a ejemplos positivos y negativos para evitar sesgos en el modelo.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Fundamental para mejorar el rendimiento del modelo.

Cuadro 7.17: SOBJ-011. Generación de muestras balanceadas

SOBJ-012	Enlace con la BBDD de conocimiento y etiquetas
Versión	1.0 (04/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Ontologías de CVEs, KEVs, y taxonomías internas
Descripción	El dataset debe estar alineado con la información contextual y etiquetas previamente guardadas en la base de datos.
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Mejora la semántica y contextualización del dataset.

Cuadro 7.18: SOBJ-012. Enlace con la BBDD de conocimiento y etiquetas

OBJ-07	Entrenamiento del modelo de aprendizaje automático
Versión	1.0 (04/06/2025)
Autor	Iván Jiménez Moreno
Fuente	Iván Jiménez Moreno
Descripción	El sistema debe entrenar un modelo de ML a partir del dataset generado, y permitir guardar su estado para su posterior inferencia.
Subobjetivos	• **SOBJ-013:** Entrenamiento con validación cruzada • **SOBJ-014:** Guardado de modelo y pipeline
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	El modelo es la base para la detección automática de vulnerabilidades.

Cuadro 7.19: OBJ-07. Entrenamiento del modelo de aprendizaje automático

SOBJ-013	Entrenamiento con validación cruzada
Versión	1.0 (04/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Dataset balanceado y estructurado
Descripción	El entrenamiento del modelo debe realizarse aplicando validación cruzada para evitar overfitting.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Garantiza robustez en la capacidad predictiva del sistema.

Cuadro 7.20: SOBJ-013. Entrenamiento con validación cruzada

SOBJ-014	Guardado de modelo y pipeline
Versión	1.0 (04/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Resultado del proceso de entrenamiento
Descripción	Una vez entrenado, el modelo y su pipeline deben ser almacenados de forma reutilizable (.joblib, etc.).
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Es esencial para la inferencia posterior durante el escaneo.

Cuadro 7.21: SOBJ-014. Guardado de modelo y pipeline

OBJ-08	Escaneo y predicción de vulnerabilidades en dominios
Versión	1.0 (04/06/2025)
Autor	Iván Jiménez Moreno
Fuente	Iván Jiménez Moreno
Descripción	El sistema debe permitir escanear dominios web y realizar predicciones sobre posibles vulnerabilidades con el modelo entrenado.
Subobjetivos	• **SOBJ-015:** Crawling automático de subrutas • **SOBJ-016:** Predicción con score de riesgo
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Es la funcionalidad principal orientada al usuario final.

Cuadro 7.22: OBJ-08. Escaneo y predicción de vulnerabilidades en dominios

SOBJ-015	Crawling automático de subrutas
Versión	1.0 (04/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Sitios web a escanear
Descripción	El sistema debe detectar y explorar automáticamente las subrutas de un dominio mediante crawling.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Aumenta la cobertura del escaneo.

Cuadro 7.23: SOBJ-015. Crawling automático de subrutas

SOBJ-016	Predicción con score de riesgo
Versión	1.0 (04/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Modelo entrenado + subrutas escaneadas
Descripción	El sistema debe asignar un score de riesgo a cada ruta escaneada, usando el modelo entrenado.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Es el núcleo de la función predictiva del sistema.

Cuadro 7.24: SOBJ-016. Predicción con score de riesgo

OBJ-09	Generación de recomendaciones mediante LLM
Versión	1.0 (04/06/2025)
Autor	Iván Jiménez Moreno
Fuente	Iván Jiménez Moreno
Descripción	El sistema debe almacenar los resultados de cada escaneo para su consulta, trazabilidad y auditoría futura.
Subobjetivos	• **SOBJ-017:** Guardado de predicciones y score • **SOBJ-018:** Consulta y recuperación por dominio o fecha
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Aporta trazabilidad completa del sistema.

Cuadro 7.25: OBJ-09. Generación de recomendaciones mediante LLM

SOBJ-017	Guardado de predicciones y score
Versión	1.0 (04/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Resultados del escaneo
Descripción	El sistema debe guardar cada predicción con su score asociado para futuras consultas.
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Aporta trazabilidad completa al sistema.

Cuadro 7.26: SOBJ-017. Guardado de predicciones y score

SOBJ-018	Consulta y recuperación por dominio o fecha
Versión	1.0 (04/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Historial almacenado en BBDD
Descripción	El sistema debe permitir recuperar escaneos por dominio, rango de fechas u otros filtros relevantes.
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Útil para auditorías, seguimiento o evolución de la seguridad.

Cuadro 7.27: SOBJ-018. Consulta y recuperación por dominio o fecha

8. Requisitos de información

Un requisito es:

1. Una condición o capacidad que necesita un usuario para resolver un problema o conseguir un objetivo.
2. Una condición o capacidad que debe cumplir o poseer un sistema (o componente del sistema) para satisfacer un contrato, estándar, especificación u otros documentos impuestos formalmente.

Los **requisitos de un sistema software** son las descripciones de aquello que el sistema tiene que hacer, los servicios que debe proveer y las restricciones que debe respetar en sus operaciones.

En este proyecto, los requisitos de información representan las descripciones estructuradas de los datos que el sistema debe gestionar para llevar a cabo el análisis de vulnerabilidades en dominios y subrutas web. A continuación, se detallan estos requisitos, clasificados según los bloques de información que forman parte de la solución propuesta.

IRQ-01	**Información relativa a los dominios**
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Input del usuario
Objetivos asociados	OBJ-01, SOBJ-01
Requisitos asociados	UC-01, CRQ-01, FRQ-01
Descripción	El sistema almacenará dominios introducidos por el usuario para realizar análisis posteriores.
Datos específicos	Nombre del dominio, fecha de análisis, ID único de escaneo
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Se enlaza con subrutas y vulnerabilidades detectadas

Cuadro 8.1: IRQ-01. Información relativa a los dominios

IRQ-02	Información relativa a las rutas/subrutas
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	HTML del sitio web, BeautifulSoup
Objetivos asociados	OBJ-03, SOBJ-05, SOBJ-06
Requisitos asociados	UC-03, FRQ-02, CRQ-02
Descripción	El sistema almacenará las subrutas accesibles obtenidas del HTML del dominio mediante crawling.
Datos específicos	URL completa
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Estas rutas son las que luego serán analizadas

Cuadro 8.2: IRQ-02. Información relativa a las rutas/subrutas

IRQ-03	Información relativa a las vulnerabilidades
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	CVE, KEV, Exploit-DB, Git repos
Objetivos asociados	OBJ-02, SOBJ-03, SOBJ-04
Requisitos asociados	UC-05, FRQ-03, CRQ-03
Descripción	El sistema guardará las vulnerabilidades extraídas de fuentes OSINT para relacionarlas con los datos obtenidos del análisis.
Datos específicos	ID único para la vulnerabilidad, categoria, patron, dato sintetico, tipo
Importancia	Muy Alta
Urgencia	Muy Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Sin filtrado por criticidad actualmente

Cuadro 8.3: IRQ-03. Información relativa a las vulnerabilidades

IRQ-04	Información relativa al modelo de entrenamiento
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Dataset generado desde patrones positivos y negativos
Objetivos asociados	OBJ-02, SOBJ-03
Requisitos asociados	UC-07, UC-08, FRQ-03
Descripción	Contiene los datos usados para entrenar el modelo de predicción de vulnerabilidades.
Datos específicos	Texto patrón, categoría, etiqueta, vector de embedding, fecha
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Guardado como archivo .joblib

Cuadro 8.4: IRQ-04. Información relativa al modelo de entrenamiento

IRQ-05	Información relativa al resultado del escaneo
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Resultado del modelo sobre rutas analizadas
Objetivos asociados	OBJ-02, OBJ-03, SOBJ-06
Requisitos asociados	UC-10, FRQ-03, CRQ-04
Descripción	Guarda el resultado de los patrones detectados tras analizar cada ruta con el modelo entrenado.
Datos específicos	Ruta analizada, score, patrón encontrado, fecha
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Si el score supera un umbral, activa la mitigación automática

Cuadro 8.5: IRQ-05. Información relativa al resultado del escaneo

IRQ-06	Información relativa a la mitigación generada
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	LLM (Llama3/Ollama)
Objetivos asociados	OBJ-04, SOBJ-07, SOBJ-08
Requisitos asociados	UC-11, FRQ-04, CRQ-05
Descripción	Contiene la recomendación generada automáticamente cuando el riesgo detectado supera el umbral.
Datos específicos	Dominio, vulnerabilidad asociada, recomendación generada, urgencia
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Justificación generada por el modelo con lenguaje técnico adaptado

Cuadro 8.6: IRQ-06. Información relativa a la mitigación generada

9. Restricciones de los requisitos de información

CRQ-01	Prevención de duplicados en los datos recopilados
Versión	1.1 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG - Escáner de vulnerabilidades
Objetivos asociados	OBJ-01, OBJ-02
Requisitos asociados	IRQ-01, FRQ-05
Descripción	El sistema debe garantizar que no se inserten datos duplicados procedentes de fuentes OSINT cuando todos sus atributos coincidan exactamente con un registro previamente almacenado.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Permite realizar múltiples escaneos sobre un mismo dominio, pero evita redundancia de datos si la información extraída es idéntica.

Cuadro 9.1: CRQ -01. Prevención de duplicados en los datos recopilados

CRQ-02	Formato válido de URLs
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG - Escáner de vulnerabilidades
Objetivos asociados	OBJ-03
Requisitos asociados	IRQ-02, FRQ-02
Descripción	Las subrutas obtenidas del crawling deben cumplir con un formato válido de URL para ser almacenadas y analizadas.
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	URLs mal formadas deben ser descartadas

Cuadro 9.2: CRQ -02. Formato válido de URLs

CRQ-03	Integridad de vulnerabilidades
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG - Escáner de vulnerabilidades
Objetivos asociados	OBJ-02
Requisitos asociados	IRQ-03, FRQ-03
Descripción	Los registros de vulnerabilidades deben contener como mínimo un ID CVE, descripción, fuente y categoría.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Campos obligatorios en la base de datos

Cuadro 9.3: CRQ -03. Integridad de vulnerabilidades

CRQ-04	Formato estructurado de predicción
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG - Escáner de vulnerabilidades
Objetivos asociados	OBJ-03
Requisitos asociados	IRQ-05, FRQ-03
Descripción	El resultado del análisis debe almacenarse en un formato estructurado que incluya ruta, score y patrón detectado.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Requerido para el análisis posterior

Cuadro 9.4: CRQ -04. Formato estructurado de predicción

CRQ-05	Texto explicativo en mitigaciones
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG - Escáner de vulnerabilidades
Objetivos asociados	OBJ-04
Requisitos asociados	IRQ-06, FRQ-04
Descripción	Las recomendaciones generadas deben incluir texto técnico claro y un campo de urgencia asociado.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Formato requerido por LLM

Cuadro 9.5: CRQ -05. Texto explicativo en mitigaciones

10. Requisitos funcionales

Los requisitos funcionales describen las funcionalidades específicas que el sistema debe implementar para satisfacer las necesidades del usuario y cumplir con los objetivos planteados.

Estos requisitos se derivan del análisis de los casos de uso y definen el comportamiento esperado del sistema en distintas situaciones.

Cada requisito funcional está vinculado a uno o más objetivos del sistema, a requisitos de información y a actores, y establece de forma detallada qué debe hacer el sistema en respuesta a una determinada acción o evento.

A continuación, se mostrarán los diferentes Requisitos Funcionales que hemos obtenido en relacion al proyecto planteado:

FRQ-01	**Registrar dominio a analizar**
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Iván Jiménez Moreno
Objetivos asociados	OBJ-01, SOBJ-01
Requisitos asociados	IRQ-01, UC-01
Descripción	El sistema debe permitir al usuario registrar un dominio que será utilizado para su posterior análisis de rutas y vulnerabilidades.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Primera interacción del usuario con el sistema

Cuadro 10.1: FRQ-01. Registrar dominio a analizar

FRQ-02	**Extraer subrutas mediante crawling**
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Iván Jiménez Moreno
Objetivos asociados	OBJ-03, SOBJ-05
Requisitos asociados	IRQ-02, UC-03
Descripción	El sistema debe extraer automáticamente todas las rutas accesibles desde el dominio introducido, recorriendo los enlaces internos detectados.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Implica análisis de HTML con BeautifulSoup

Cuadro 10.2: FRQ-02. Extraer subrutas mediante crawling

FRQ-03	Analizar contenido de cada subruta
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Iván Jiménez Moreno
Objetivos asociados	OBJ-03, SOBJ-06
Requisitos asociados	IRQ-02, IRQ-03, UC-04
Descripción	Cada subruta extraída será analizada individualmente para detectar posibles patrones relacionados con vulnerabilidades conocidas.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Funciona con embeddings + búsqueda semántica

Cuadro 10.3: FRQ-03. Analizar contenido de cada subruta

FRQ-04	Generar recomendaciones de mitigación
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Iván Jiménez Moreno
Objetivos asociados	OBJ-04, SOBJ-07, SOBJ-08
Requisitos asociados	IRQ-06, UC-11
Descripción	Si el análisis detecta una vulnerabilidad relevante, el sistema deberá generar una recomendación automática a través del modelo LLM.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Generación mediante LLaMA3/Ollama

Cuadro 10.4: FRQ-04. Generar recomendaciones de mitigación

FRQ-05	Validar datos duplicados
Versión	1.0 (03/06/2025)
Autor	Iván Jiménez Moreno
Fuentes	Iván Jiménez Moreno
Objetivos asociados	OBJ-01, OBJ-02
Descripción	El sistema debe comprobar si un registro OSINT con los mismos atributos ya existe en la base de datos antes de insertarlo.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Relacionado con la restricción CRQ-01. Evita redundancia innecesaria.

Cuadro 10.: FRQ-05. Validar datos duplicados

FRQ-06	Generar recomendaciones de mitigación
Versión	1.0 (03/06/2025)
Autor	Iván Jiménez Moreno
Fuentes	Iván Jiménez Moreno
Objetivos asociados	OBJ-02
Descripción	El sistema debe almacenar los resultados del escaneo en MongoDB, incluyendo rutas, vulnerabilidades y scores asociados.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Permite consultar posteriormente los análisis realizados.

Cuadro 10.6: FRQ-06. Generar recomendaciones de mitigación

FRQ-07	Guardar resultados en la base de datos
Versión	1.0 (03/06/2025)
Autor	Iván Jiménez Moreno
Fuentes	Iván Jiménez Moreno
Objetivos asociados	OBJ-02
Descripción	El sistema debe almacenar los resultados del escaneo en MongoDB, incluyendo rutas, vulnerabilidades y scores asociados.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Permite consultar posteriormente los análisis realizados.

Cuadro 10.7: FRQ-07. Guardar resultados en la base de datos

FRQ-08	Guardar modelo entrenado en .joblib
Versión	1.0 (03/06/2025)
Autor	Iván Jiménez Moreno
Fuentes	Iván Jiménez Moreno
Objetivos asociados	OBJ-03
Descripción	El sistema debe exportar el modelo entrenado a un archivo .joblib para su posterior uso en producción.
Importancia	Media
Urgencia	Media
Estado	Finalizado
Estabilidad	Alta
Comentarios	Permite separación entre entrenamiento e inferencia.

Cuadro 10.8: FRQ-08. Guardar modelo entrenado en .joblib

11. Requisitos no funcionales

Los requisitos no funcionales (NFR) son aquellos que describen las propiedades y cualidades del sistema más allá de lo que este debe hacer.
Mientras que los requisitos funcionales se centran en las acciones que debe realizar el sistema, los NFR definen **cómo debe comportarse** en términos de rendimiento, usabilidad, seguridad, escalabilidad, mantenibilidad, entre otros aspectos clave.
Este conjunto de requisitos es fundamental para asegurar que el sistema no solo cumpla su propósito, sino que lo haga de forma eficiente, segura, accesible y sostenible en el tiempo.

A continuación, se mostrarán una serie de Requisitos No Funcionales con una breve introducción, referentes a nuestro proyecto:

11.1 Mantenibilidad

Los requisitos de mantenibilidad hacen referencia a la facilidad con la que el sistema puede ser modificado para corregir errores, mejorar funcionalidades o adaptarse a nuevos entornos.
Un diseño modular y una documentación clara son esenciales para asegurar una evolución sostenible del sistema a largo plazo.

NFR-01	**Modularidad del sistema**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	General
Requisitos asociados	FRQ-01, FRQ-02
Descripción	El sistema debe estar estructurado en módulos independientes (crawling, entrenamiento, análisis) para facilitar su mantenimiento y actualización.
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Facilita el mantenimiento a futuro

Cuadro 11.1: NFR-01. Modularidad del sistema

NFR-02	**Código documentado**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	General
Requisitos asociados	Todos
Descripción	Todo el código fuente debe estar correctamente documentado mediante comentarios, nombres de funciones descriptivos y README explicativo.
Importancia	Media
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Mejora mantenibilidad futura

Cuadro 11.2: NFR-02. Código documentado

11.2 Usabilidad

Los requisitos de usabilidad establecen criterios para que el sistema sea fácil de utilizar, accesible y comprensible por parte del usuario final.
Una buena usabilidad reduce la curva de aprendizaje y mejora la interacción general con la plataforma.

NFR-03	**Interfaz sencilla de uso**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	OBJ-01, OBJ-04
Requisitos asociados	FRQ-01, FRQ-04
Descripción	El sistema debe contar con una interfaz clara y simple para facilitar la introducción de dominios y la consulta de resultados.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Relevante para usuarios sin perfil técnico

Cuadro 11.3: NFR-03. Interfaz sencilla de uso

NFR-04	**Respuestas comprensibles del LLM**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	OBJ-04
Requisitos asociados	FRQ-04
Descripción	Las recomendaciones generadas por el LLM deben estar escritas en lenguaje comprensible para usuarios sin perfil técnico.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Aplicable especialmente a mitigaciones

Cuadro 11.4: NFR-04. Respuestas comprensibles del LLM

11.3 Portabilidad

Los requisitos de portabilidad garantizan que el sistema pueda ejecutarse en distintos entornos o plataformas con el mínimo esfuerzo de adaptación.
Se busca que tanto la interfaz como el sistema interno sean independientes del sistema operativo o la infraestructura.

NFR-05	**Ejecución en entornos Docker**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	Todos
Requisitos asociados	FRQ-01 a FRQ-04
Descripción	El sistema debe ser portable y ejecutable en cualquier máquina compatible gracias al uso de contenedores Docker.
Importancia	Muy Alta
Urgencia	Alta
Estado	Implementado
Estabilidad	Alta
Comentarios	Facilita la portabilidad multiplataforma

Cuadro 11.5: NFR-05. Ejecución en entornos Docker

NFR-06	**Soporte multiplataforma**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	General
Requisitos asociados	FRQ-01
Descripción	La interfaz debe ser accesible desde cualquier sistema operativo compatible con navegador moderno.
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Interfaz Web React

Cuadro 11.6: NFR-06. Soporte multiplataforma

11.4 Disponibilidad

La disponibilidad se refiere a la capacidad del sistema para estar operativo y accesible en todo momento, sin interrupciones innecesarias.
Este tipo de requisito es fundamental en sistemas que deben estar continuamente disponibles para su uso.

NFR-07	**Accesibilidad del sistema en todo momento**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	OBJ-01
Requisitos asociados	FRQ-01, FRQ-02
Descripción	El sistema debe estar disponible para su uso sin restricciones de horario ni mantenimiento planificado durante la fase de pruebas.
Importancia	Media
Urgencia	Media
Estado	En desarrollo
Estabilidad	Media
Comentarios	Para entorno de pruebas

Cuadro 11.7: NFR-07. Accesibilidad del sistema en todo momento

NFR-08	**Sin interrupciones durante análisis**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	OBJ-03
Requisitos asociados	FRQ-02, FRQ-03
Descripción	Una vez iniciado el análisis, no debe detenerse por pérdida de conexión o timeout intermedio.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Evita errores durante crawling

Cuadro 11.8: NFR-08. Sin interrupciones durante análisis

11.5 Rendimiento

Los requisitos de rendimiento especifican el nivel de eficiencia que debe tener el sistema en cuanto a velocidad de ejecución, tiempo de respuesta o consumo de recursos. Su cumplimiento es clave para garantizar la escalabilidad y operatividad del sistema.

NFR-09	**Tiempo máximo de respuesta en escaneos**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	OBJ-03
Requisitos asociados	FRQ-02, FRQ-03
Descripción	El tiempo de respuesta desde que se lanza el escaneo hasta que se recibe el análisis de una subruta no debe superar los 10 segundos.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Evalúa eficiencia del sistema

Cuadro 11.9: NFR-09. Tiempo máximo de respuesta en escaneos

NFR-010	**Optimización del modelo de predicción**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	OBJ-03
Requisitos asociados	FRQ-03, IRQ-04
Descripción	El modelo entrenado debe estar optimizado en tamaño para poder cargarse en memoria sin ralentizar el análisis de rutas.
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Afecta al rendimiento general

Cuadro 11.10: NFR-010. Consulta y recuperación por dominio o fecha

11.6 Escalabilidad

La escalabilidad determina la capacidad del sistema para manejar incrementos en la carga de trabajo sin degradar su rendimiento.
Un sistema escalable puede crecer en complejidad o volumen de datos manteniendo su funcionalidad y eficiencia.

NFR-011	**Soporte para múltiples dominios concurrentes**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	OBJ-01, OBJ-03
Requisitos asociados	FRQ-01, FRQ-03
Descripción	El sistema debe poder procesar escaneos de múltiples dominios al mismo tiempo sin bloqueos.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Media
Comentarios	Asegura escalabilidad horizontal

Cuadro 11.11: NFR-011. Soporte para múltiples dominios concurrentes

NFR-012	**Capacidad de ampliación del dataset**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	OBJ-02
Requisitos asociados	IRQ-03, IRQ-04
Descripción	El sistema debe permitir añadir nuevas vulnerabilidades al dataset sin necesidad de reentrenar desde cero.
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Mejora el ciclo de evolución del modelo

Cuadro 11.12: NFR-012. Capacidad de ampliación del dataset

11.7 Seguridad

La seguridad comprende los mecanismos y controles necesarios para proteger el sistema frente a accesos no autorizados, manipulaciones o ataques.
Incluye validación de entradas, control de fuentes externas y salvaguarda de los datos almacenados.

NFR-013	**Protección frente a inputs maliciosos**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	OBJ-02
Requisitos asociados	IRQ-03
Descripción	Solo se deben considerar fuentes fiables y verificadas (CVE, KEV, Exploit-DB) para alimentar el sistema de detección.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Afecta directamente a la precisión del sistema

Cuadro 11.13: NFR-013. Protección frente a inputs maliciosos

NFR-014	**Protección frente a inputs maliciosos**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	OBJ-01
Requisitos asociados	FRQ-01, IRQ-01
Descripción	El sistema debe sanitizar y validar correctamente todos los inputs del usuario para evitar ataques como XSS o inyecciones.
Importancia	Muy Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Añade una capa de seguridad importante

Cuadro 11.14: NFR-014. Protección frente a inputs maliciosos

11.8 Trazabilidad

La trazabilidad implica la capacidad de rastrear el origen y evolución de los datos dentro del sistema.
Permite vincular resultados con sus entradas, auditar acciones del usuario y justificar decisiones tomadas automáticamente por el sistema.

NFR-015	**Registro de acciones del usuario**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	OBJ-01
Requisitos asociados	FRQ-01, IRQ-01
Descripción	El sistema debe guardar un log de todas las acciones realizadas por el usuario (introducción de dominio, escaneo, consultas, etc.).
Importancia	Alta
Urgencia	Media
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Facilita trazabilidad de actividades

Cuadro 11.15: NFR-015. Registro de acciones del usuario

NFR-016	**Trazabilidad entre resultados y fuentes**
Versión	1.0 (03/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Proyecto TFG
Objetivos asociados	OBJ-02
Requisitos asociados	IRQ-03, IRQ-05
Descripción	Cada resultado de análisis debe vincularse con la fuente de información que originó la vulnerabilidad detectada.
Importancia	Alta
Urgencia	Alta
Estado	En desarrollo
Estabilidad	Alta
Comentarios	Mejora transparencia del sistema

Cuadro 11.16: NFR-016. Trazabilidad entre resultados y fuentes

12. Diagramas de secuencia

A continuación, mostraremos el diagrama completo de secuencia referente a nuestro programa, de tal manera que de un solo golpe de vista podremos entender la funcionalidad de este.

Seguidamente diseccionaremos el diagrama en partes de tal manera que podamos explicar cada acción que se lleva a cabo por parte del usuario de una manera más limpia y clara.

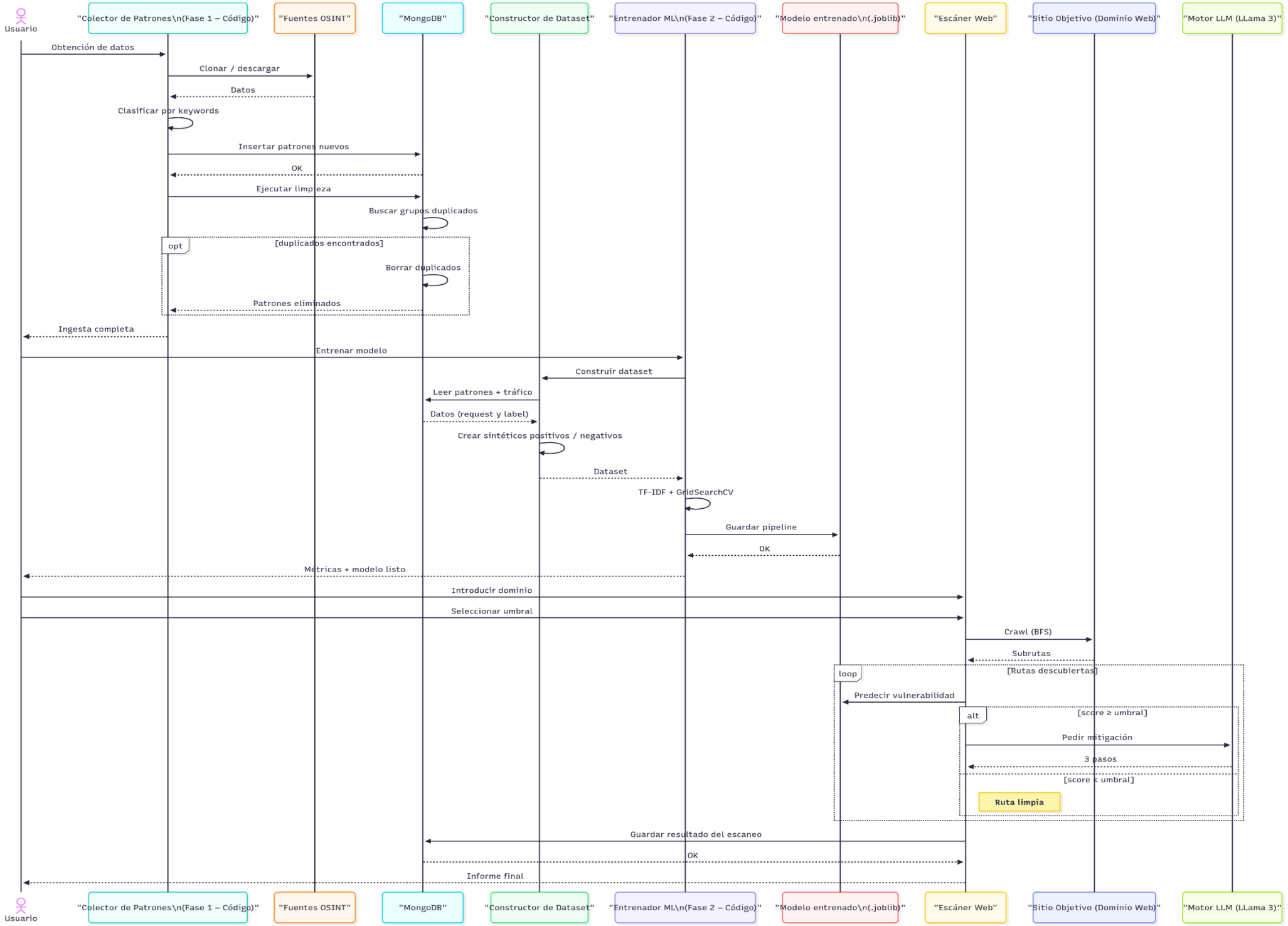

Usuario
"Colector de Patrones\n(Fase 1 – Código)"
"Fuentes OSINT"
"MongoDB"
"Constructor de Dataset"
"Entrenador ML\n(Fase 2 – Código)"
"Modelo entrenado\n(.joblib)"
"Escáner Web"
"Sitio Objetivo (Dominio Web)"
"Motor LLM (LLama 3)"
Obtención de datos
Clonar / descargar
Datos
Clasificar por keywords
Insertar patrones nuevos
OK
Ejecutar limpieza
Buscar grupos duplicados
opt
[duplicados encontrados]
Borrar duplicados
Patrones eliminados
Ingesta completa
Entrenar modelo
Construir dataset
Leer patrones + tráfico
Datos (request y label)
Crear sintéticos positivos / negativos
Dataset
TF-IDF + GridSearchCV
Guardar pipeline
OK
Métricas + modelo listo
Introducir dominio
Seleccionar umbral
Crawl (BFS)
Subrutas
[Rutas descubiertas]
loop
Predecir vulnerabilidad
alt
[score ≥ umbral]
Pedir mitigación
3 pasos
[score < umbral]
Ruta limpia
Guardar resultado del escaneo
OK
Informe final

Figura 12.1: Diagrama de secuencia

Tras mostrar el diagrama completo de nuestro programa podemos determinar que el sistema está compuesto por múltiples componentes distribuidos en diferentes fases, cada uno con una función específica dentro del flujo de detección y análisis de vulnerabilidades. Los principales actores que intervienen en el proceso son:

- **Usuario**: Inicia los procesos desde una interfaz o entorno de control. Es quien desencadena las acciones clave.
- **Colector de Patrones (Fase 1 – Código)**: Módulo encargado de obtener, clasificar y enviar patrones desde fuentes OSINT.
- **Fuentes OSINT**: Origen de los datos sin procesar (repositorios, bases públicas, etc.).
- **MongoDB**: Base de datos NoSQL que centraliza la información de patrones y resultados.
- **Constructor de Dataset**: Encargado de estructurar los datos de entrenamiento.
- **Entrenador ML (Fase 2 – Código)**: Módulo que entrena el modelo de clasificación a partir de los patrones almacenados.
- **Modelo entrenado (.joblib)**: Resultado del entrenamiento, listo para ser usado en la predicción.
- **Escáner Web**: Motor de análisis que explora el dominio objetivo y aplica el modelo.
- **Sitio Objetivo (Dominio Web)**: La página web que se desea analizar en busca de vulnerabilidades.
- **Motor LLM (LlaMA 3)**: Encargado de generar recomendaciones personalizadas en lenguaje natural usando Ollama.

12.1 Diagrama de secuencia diseccionado

En este apartado encontraremos las diferentes iteraciones que se encuentran dentro de nuestro diagrama de secuencia en donde explicaremos cada una para que se entienda el funcionamiento parte por parte de lo que viene siendo nuestro proyecto en su totalidad.

12.1.1 Iteración 1 – Obtención y limpieza de patrones

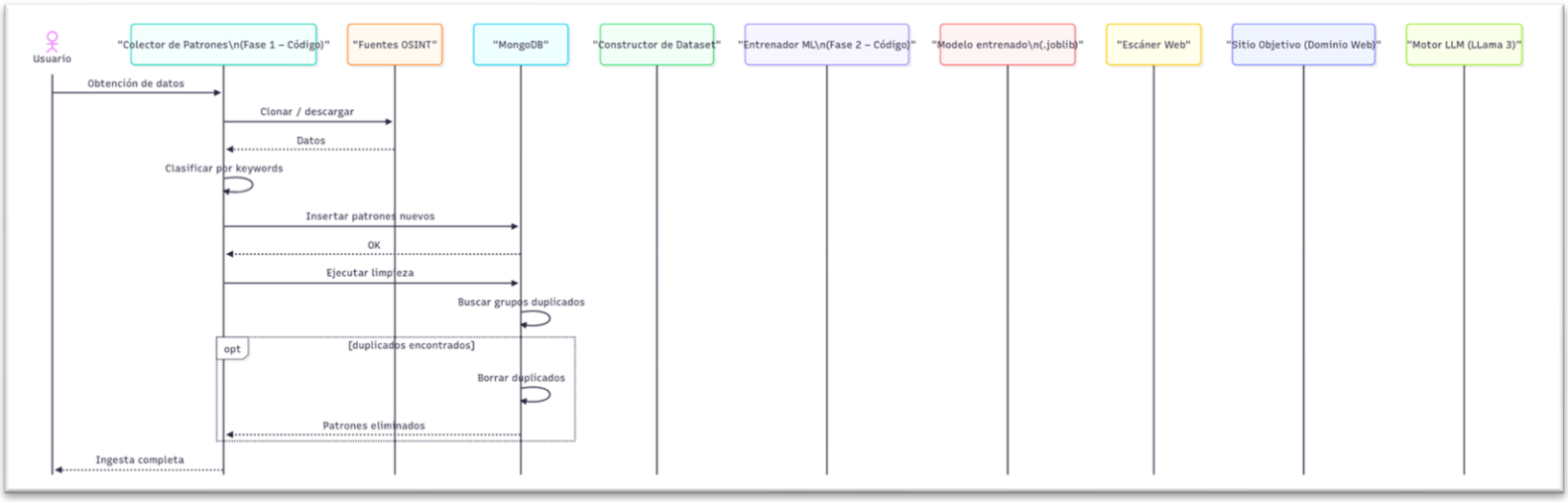


Figura 12.2: Primera iteración del diagrama de secuencia

En este diagrama podemos identificar un flujo en el que el usuario inicia en el sistema con la acción de obtener/solicitar datos de vulnerabilidades a través de fuentes abiertas (OSINT). El **Colector de Patrones** la función que desempeña es la de clonar o descargar los datos y clasificarlos según *keywords* específicas (palabras que nosotros establecemos a nivel de codigo que se localizarán dentro de los datos obtenidos).

Seguidamente lo que hará, será insertar los patrones nuevos en la base de datos MongoDB y una vez insertados, se ejecuta una limpieza que identifica y elimina todos aquellos campos de la base de datos que se encuentren duplicados (Esto lo hacemos porque puede haber algún que otro *log de datos* que aparezca en más de una fuente, de tal manera que asi reducimos el número de datos repetidos). Este proceso termina con la base de datos lista para la fase de entrenamiento.

Actores involucrados en esta primera iteración:

- Usuario
- Colector de Patrones
- Fuentes OSINT
- MongoDB

12.1.2 Iteración 2 – Entrenamiento del modelo de clasificación

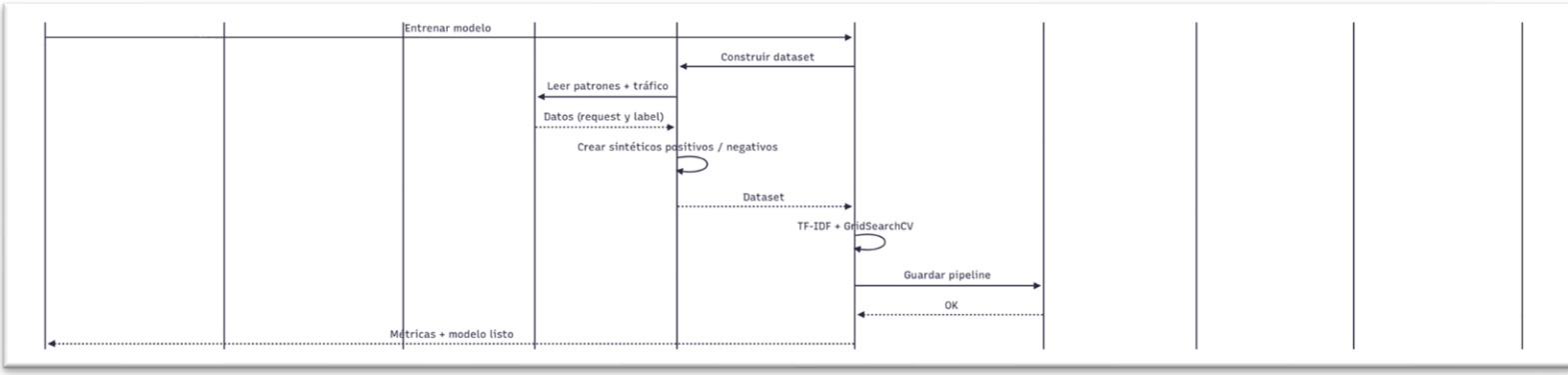


Figura 12.3: Segunda iteración del diagrama de secuencia

Tras completar la ingesta de patrones, el usuario comenzará con la construcción y entrenamiento del modelo de machine learning. El **Constructor de Dataset** accede a los patrones y genera ejemplos positivos y negativos para asi lograr un entrenamiento eficiente basado en posibles casos reales.

Luego, se transforma el contenido de los datos(patrones guardados en la Iteracion 1 + datos sintéticos creados en esta Iteración 2) en una representación matemática que pueda entender el modelo de inteligencia artificial, utilizando una técnica llamada **TF-IDF**, que básicamente se encarga de ayudar a identificar qué palabras son más importantes dentro de cada patrón. Seguidamente, se entrena el modelo probando diferentes combinaciones para encontrar los mejores parámetros posibles que le permitan aprender con la mayor precisión. Este proceso se realiza mediante una técnica llamada **GridSearchCV**, que actúa como una especie de prueba y error controlada, buscando la configuración óptima del modelo para que acierte lo máximo posible al clasificar los patrones de vulnerabilidad.

El modelo entrenado se guarda finalmente como un pipeline .joblib (en el directorio de nuestro sistema que nosotros le indiquemos en el código fuente), dejando el sistema listo para la predicción de vulnerabilidades.

Actores involucrados en esta segunda iteración:

- Usuario
- Constructor de Dataset
- Entrenador ML (Fase 2 - Código)
- Modelo entrenado (.joblib)
- MongoDB

12.1.3 Iteración 3 – Escaneo del dominio y predicción

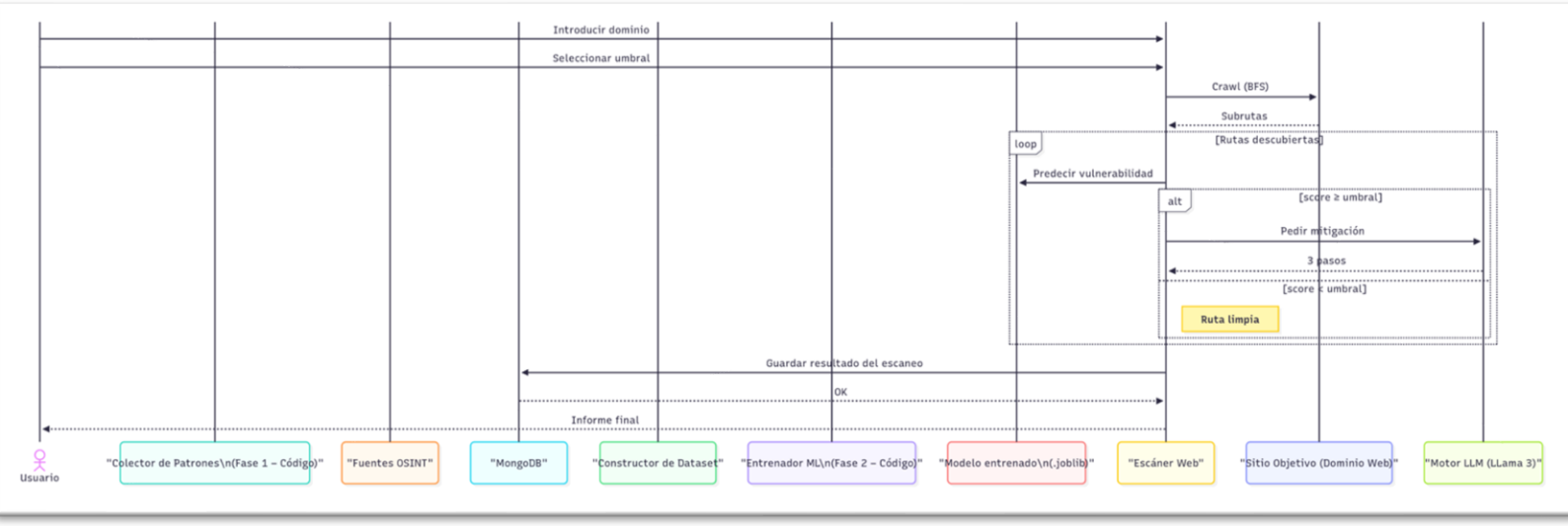


Figura 12.4: Tercera iteración del diagrama de secuencia

Para esta tercera interacción el usuario introduce un dominio a analizar. El **Escáner Web** recorre el sitio objetivo aplicando una estrategia de crawl BFS, extrayendo rutas y subruta del dicho dominio. Para cada ruta, se ejecuta una predicción con el modelo ML entrenado. Si el score de la vulnerabilidad es igual o superior a un umbral, el sistema consulta al **Motor LLM** (LLaMA3 vía Ollama) para generar una recomendación de mitigación para las vulnerabilidades que se hayan encontrado. En caso contrario, se marca como ruta limpia. Finalmente, se guarda el resultado en la base de datos y se genera un informe para el usuario.

Actores involucrados en esta tercera iteración:

- Usuario
- Escáner Web
- Sitio Objetivo (Dominio Web)
- Modelo entrenado (.joblib)
- Motor LLM (LLaMA 3)
- MongoDB

13. Diagrama de Casos de Uso

El diagrama de casos de uso representa de forma estructurada el funcionamiento del sistema de escaneo de vulnerabilidades en dominios web, así como la interacción del usuario con los distintos procesos internos.
A continuación podemos ver el diagrama de casos de uso referente a nuestro código realizado en el Jupyter Notebook:

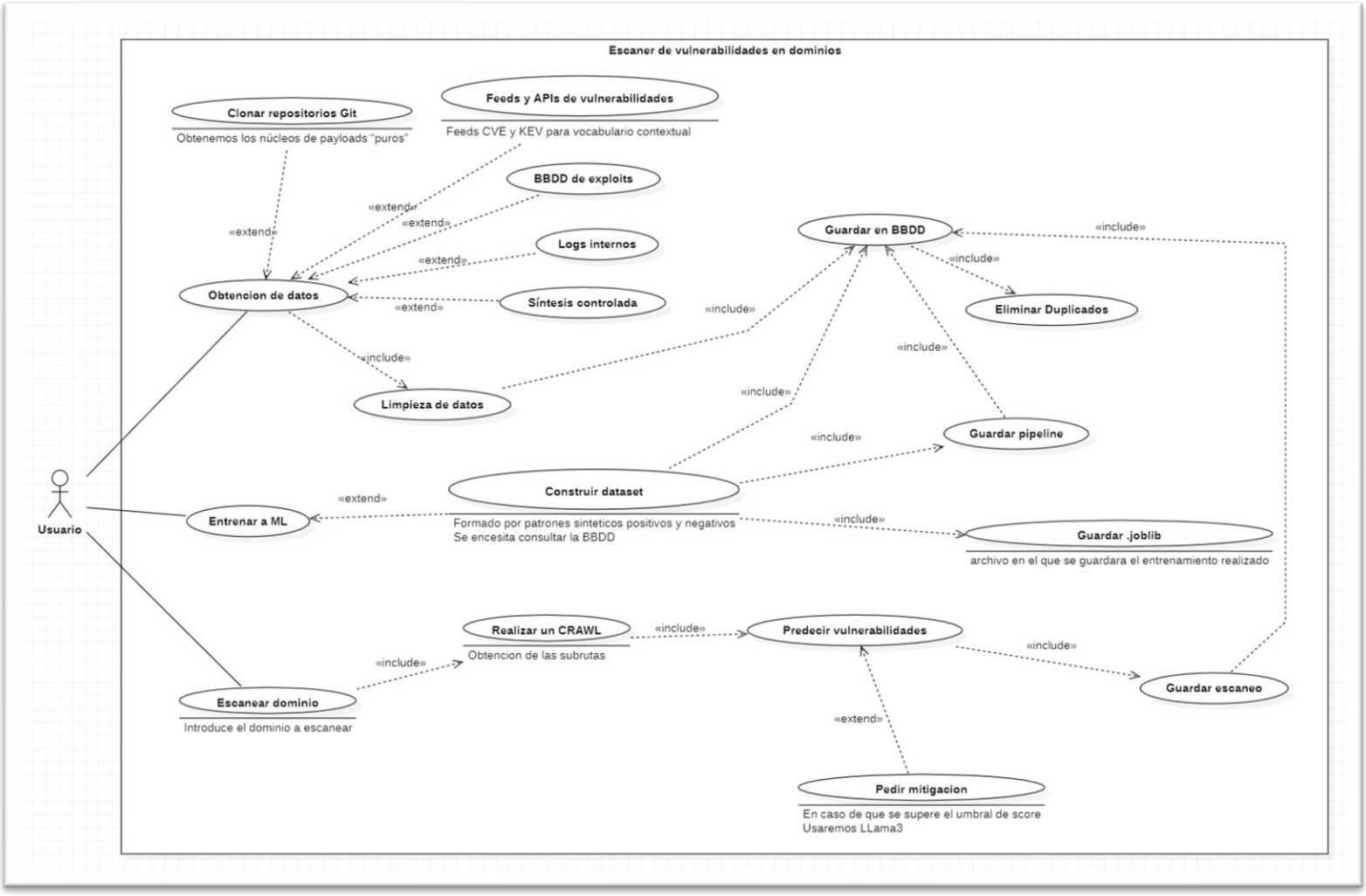


Figura 13.1: Diagrama de casos de uso

El usuario (“Usuario”) puede iniciar tanto el escaneo de un dominio como el entrenamiento del modelo de machine learning.

El proceso se inicia con la obtención de datos desde diversas fuentes (repositorios Git, APIs de vulnerabilidades, BBDD de exploits, logs internos o síntesis controlada), donde una vez que se obtienen dichos datos se realiza posteriormente una fase de limpieza previa al almacenamiento en la base de datos. Esta fase de limpieza se realiza debido a que puede haber información procedente de una fuente que se encuentre en otras de las

que también obtengamos información relevante.

Posteriormente, estos datos alimentan la construcción del dataset de entrenamiento que, una vez procesado, se guarda como archivo .joblib.

Cuando el usuario escanea un dominio, el sistema realiza un *crawling* para extraer subrutas, predice vulnerabilidades sobre estas y, si el riesgo supera un umbral, genera recomendaciones de mitigación mediante un modelo LLM (Llama3).

Durante todas estas fases, el sistema realiza tareas auxiliares como la eliminación de duplicados, el guardado de pipelines y la persistencia de resultados, garantizando trazabilidad y consistencia del flujo completo.

Actores

Un actor es una entidad externa al sistema que se modela y que puede interactuar con él. Puede ser una persona o un grupo de personas homogéneas, otro sistema o una máquina.
Los actores son externos al sistema que vamos a desarrollar. Por lo tanto, al identificarlos estamos comenzando a delimitar el sistema y a definir su alcance.

ACT-01	**Usuario**
Versión	1.0 (02/06/2025)
Autores	Iván Jiménez Moreno
Fuentes	Iván Jiménez Moreno
Descripción	Este actor representa al usuario responsable del uso del sistema. Puede introducir dominios, lanzar escaneos, consultar resultados y recibir recomendaciones.
Comentarios	Único usuario previsto; no hay distinción por roles en esta versión.

Cuadro 13.1: ACT-01. Usuario

14. Diagrama de clases (o arquitectura de módulos)

El **diagrama de clases** es un tipo de diagrama estructural de UML (Unified Modeling Language) que permite representar gráficamente la estructura del sistema en términos de **clases**, sus **atributos**, **métodos** y las **relaciones** existentes entre ellas. Este tipo de diagrama es fundamental en el diseño orientado a objetos, ya que proporciona una visión clara y estática de los componentes que conforman la aplicación, facilitando la comprensión, mantenimiento y escalabilidad del sistema.

En este proyecto, el diagrama de clases se ha elaborado para reflejar las entidades principales involucradas en el proceso de detección de vulnerabilidades en dominios web, desde la recopilación de patrones hasta la predicción y mitigación de vulnerabilidades mediante el uso de modelos de machine learning y lenguaje natural.

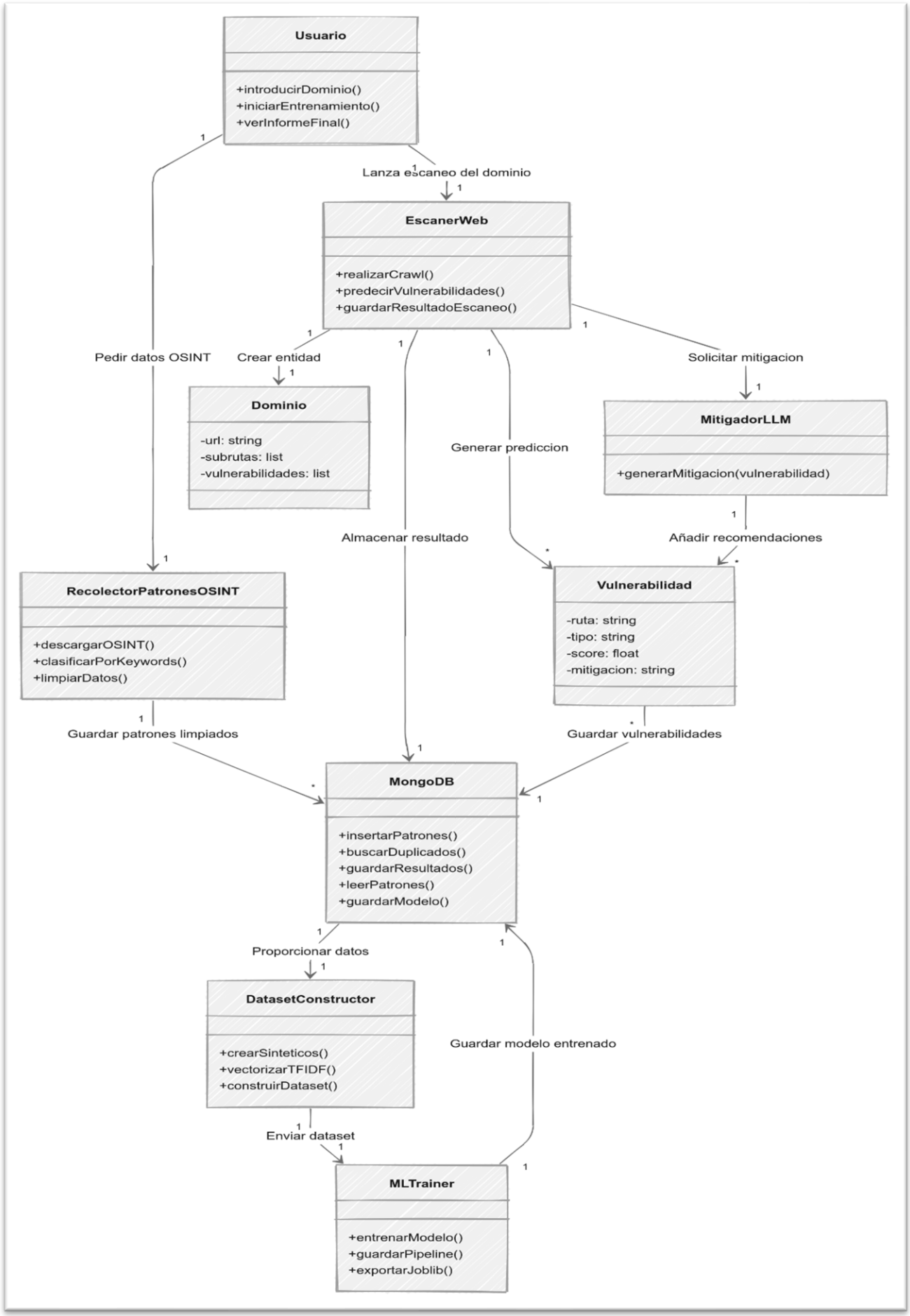


Figura 14.1: Diagrama de casos de clases

14.1 Explicación de los módulos

En este apartado explicaremos las principales clases y módulos que forman parte del sistema desarrollado. Cada una cumple una función concreta dentro del flujo de trabajo de escaneo, análisis y generación de recomendaciones. A continuación, se describen de forma resumida:

1. Usuario

Es el actor principal del sistema. Representa a la persona que interactúa directamente con la plataforma a través del navegador. Puede iniciar un análisis, entrenar el modelo o consultar resultados.
Métodos principales:

- introducirDominio(): inicia el proceso de escaneo sobre un dominio web.
- iniciarEntrenamiento(): lanza el entrenamiento del modelo de machine learning.
- verInformeFinal(): permite visualizar los resultados de un escaneo una vez finalizado.

2. RecolectorPatronesOSINT

Este componente se encarga de recopilar información externa útil para entrenar el modelo de Machine Learning. Sus funciones principales son la de extraer, clasificar y limpiar datos procedentes de fuentes públicas como CVE, GitHub, Exploit-DB, etc.
Métodos principales:

- descargarOSINT(): conecta con repositorios y descarga los datos.
- clasificarPorKeywords(): organiza los datos en categorías relevantes (por ejemplo: XSS, SQLi...).
- limpiarDatos(): filtra duplicados o datos irrelevantes para mantener calidad en el dataset.

3. MongoDB

Es el sistema gestor de base de datos utilizado. Almacena patrones, resultados de escaneos, logs y modelos entrenados.
Métodos principales:

- insertarPatrones(): guarda patrones nuevos recogidos de las fuentes OSINT.
- buscarDuplicados(): evita almacenar datos repetidos.
- guardarResultados(): guarda el resultado de un escaneo.
- leerPatrones(): recupera datos para el entrenamiento.
- guardarModelo(): almacena el modelo de ML ya entrenado.

4. DatasetConstructor

Se encarga de preparar los datos que se usarán para entrenar el modelo. Aquí se crean ejemplos sintéticos y se vectoriza el contenido.
Métodos principales:

- crearSinteticos(): genera ejemplos adicionales para balancear las clases del dataset.
- vectorizarTFIDF(): transforma texto en vectores numéricos que el modelo puede interpretar.
- construirDataset(): une todo el contenido preprocesado en un único conjunto listo para entrenamiento.

5. MLTrainer

Este módulo entrena el modelo de clasificación de vulnerabilidades usando el dataset anterior. Ajusta parámetros y guarda el modelo final para usarlo en producción.
Métodos principales:

- entrenarModelo(): ejecuta el proceso de entrenamiento completo.
- guardarPipeline(): almacena el pipeline con todas las transformaciones aplicadas.
- exportarJoblib(): exporta el modelo entrenado como archivo .joblib.

6. EscanerWeb

Realiza el trabajo principal de análisis del dominio. Detecta subrutas accesibles, ejecuta el modelo de predicción y guarda los resultados.
Métodos principales:

- realizarCrawl(): recorre el HTML de una web y extrae todas las rutas navegables.
- predecirVulnerabilidades(): analiza cada ruta usando el modelo de ML y calcula su score de riesgo.
- guardarResultadoEscaneo(): almacena los resultados obtenidos en la base de datos.

Por ejemplo, si una ruta presenta un patrón similar a un payload de XSS, esta clase se encarga de calcular el score y determinar si debe ser marcada como peligrosa

7. Dominio

Representa una web objetivo. Almacena su URL, las rutas internas que se han encontrado y las vulnerabilidades detectadas.
Atributos:

- url: la dirección del sitio.
- subrutas: lista de rutas detectadas mediante crawling.
- vulnerabilidades: fallos encontrados tras el análisis.

8. Vulnerabilidad

Modelo que representa una vulnerabilidad concreta detectada en una ruta.
Atributos:

- ruta: URL afectada.
- tipo: tipo de vulnerabilidad (por ejemplo, SQLi, XSS...).
- score: nivel de riesgo calculado por el modelo.
- mitigacion: recomendación generada por el sistema para corregir el fallo.

9. MitigadorLLM

Este componente conecta con el modelo de lenguaje LLaMA 3 a través de Ollama. Se encarga de redactar una recomendación práctica y entendible cuando se detecta una vulnerabilidad.

Método principal:

- generarMitigacion(vulnerabilidad): recibe una vulnerabilidad y devuelve un texto explicando cómo resolverla.

14.2 Relaciones y Multiplicidades

A continuación, se describen las relaciones existentes entre las distintas clases y módulos del sistema, así como las multiplicidades que definen cuántas instancias pueden intervenir en cada interacción. Esto permite entender de forma estructurada y clara, cómo se conectan los distintos elementos durante el flujo completo del escaneo y entrenamiento.

1. Usuario → EscanerWeb

Multiplicidad: 1 → 1

Un usuario puede lanzar un único escaneo a la vez. Cada instancia del escáner se activa como resultado directo de una acción manual (por ejemplo, al pulsar el botón "Escanear" en la interfaz).

2. Usuario → RecolectorPatronesOSINT

Multiplicidad: 1 → 1

El usuario también puede iniciar el proceso de recolección de datos desde fuentes OSINT como parte del flujo de entrenamiento. Cada ejecución de este módulo es gestionada de forma individual.

3. RecolectorPatronesOSINT → MongoDB

Multiplicidad: 1 → 1

Cada vez que se ejecuta el recolector, los patrones limpios obtenidos se guardan en una única colección de la base de datos MongoDB.

4. MongoDB → DatasetConstructor

Multiplicidad: 1 → 1

El constructor de datasets accede a los datos almacenados en MongoDB para generar el conjunto de entrenamiento. Este acceso es directo y único por ejecución.

5. DatasetConstructor → MLTrainer

Multiplicidad: 1 → 1

Una vez creado el dataset, este se pasa al componente de entrenamiento, que se encarga de construir el modelo de clasificación.

6. MLTrainer → MongoDB

Multiplicidad: 1 → 1

Al finalizar el entrenamiento, el modelo resultante se guarda como un archivo .joblib en la base de datos, asegurando su reutilización en escaneos futuros.

7. EscanerWeb → Dominio

Multiplicidad: 1 → 1

Cada ejecución del escáner se enfoca en un único dominio web. Es decir, se analiza una página concreta por cada acción de escaneo.

8. EscanerWeb → Vulnerabilidad

Multiplicidad: 1 → ***

Un mismo escaneo puede detectar múltiples vulnerabilidades, ya que cada subruta se analiza por separado y puede presentar distintos patrones sospechosos.

9. EscanerWeb → MitigadorLLM

Multiplicidad: 1 → 1

Cuando se detectan fallos, el escáner lanza una única consulta al módulo de mitigación con el objetivo de obtener recomendaciones automáticas.

10. MitigadorLLM → Vulnerabilidad

Multiplicidad: 1 → ***

El modelo LLM puede generar recomendaciones para varias vulnerabilidades dentro de un mismo escaneo, devolviendo sugerencias específicas para cada caso.

11. Dominio → RecolectorPatronesOSINT

Multiplicidad: 1 → 1

Aunque no es una relación directa en tiempo de ejecución, el dominio escaneado se beneficia de los patrones recogidos previamente por el recolector, lo que mejora la precisión del análisis.

12. Vulnerabilidad → MongoDB

Multiplicidad: * → 1

Todas las vulnerabilidades detectadas se almacenan en la base de datos. Aunque un escaneo genere múltiples fallos, todos se agrupan y guardan bajo una misma instancia de MongoDB.

15. Modelo Entidad/Relación de la base de datos

El modelo Entidad-Relación (E-R) representa un determinado dominio utilizando entidades y relaciones entre ellas.

Entidades:

- Cada entidad se caracteriza por el valor de sus atributos.
- Un atributo (clave) identifica unívocamente cada entidad.

Relaciones:

- Una relación describe una asociación entre varias entidades.
- Esta asociación determina el número de entidades (cardinalidad) relacionadas en la misma.

En el caso de este proyecto, el sistema se ha diseñado para la detección automatizada de vulnerabilidades en dominios web, incorporando mecanismos de entrenamiento de modelos de Machine Learning, análisis de patrones OSINT, y generación de recomendaciones con LLM.

Este diagrama es esencial para entender cómo se relacionan los componentes de la base de datos, y facilita tanto la implementación como el mantenimiento del sistema en el tiempo.

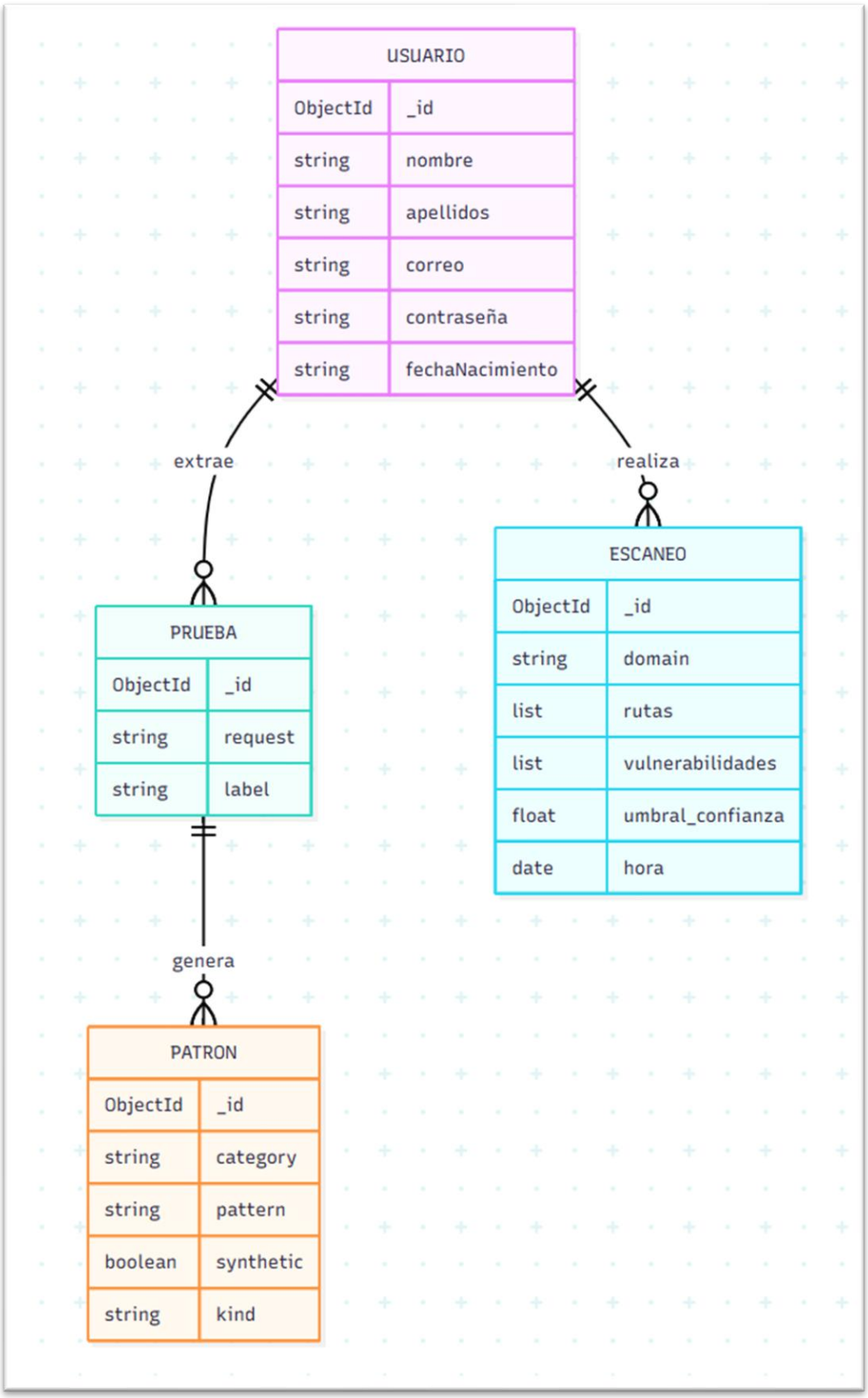


Figura 15.1: Diagrama de Entidad-Relación

15.1 Entidades del modelo

- **USUARIO**
    - Entidad encargada de representar al usuario del sistema, que ejecuta escaneos de dominios y extrae los patrones de las fuentes OSINT.
    - Atributos: _id, nombre, apellidos, correo, contraseña, fechaNacimiento.
- **PRUEBA**
    - Representa una entrada obtenida de fuentes OSINT (como payloads brutos).
    - Atributos: _id, request, label.
- **PATRON**
    - Representa un patrón sintético o real derivado de una prueba, utilizado para entrenar modelos de detección.
    - Atributos: _id, category, pattern, synthetic, kind.
- **ESCANEO**
    - Representa el resultado del análisis de un dominio ejecutado por un usuario.
    - Atributos: _id, domain, rutas, vulnerabilidades, umbral_confianza, hora.

15.2 Relaciones entre entidades

- **USUARIO -- extrae – PRUEBA**
    - Un usuario puede extraer múltiples pruebas desde fuentes OSINT.
    - Relación: 1:N
- **PRUEBA -- genera – PATRON**
    - A partir de una prueba pueden generarse múltiples patrones (sintéticos o reales).
    - Relación: 1:N
- **USUARIO -- realiza – ESCANEO**
    - Un usuario puede ejecutar múltiples escaneos de dominios.
    - Relación: 1:N

16. Diccionario de Datos

En esta sección de la memoria vamos a describir la estructura que tienen nuestras tablas dentro de la base de datos:

Tabla *tbl_users*				
Campo	**Tipo de Dato**	**Clave**	**Restricciones**	**Descripción**
_id	ObjectId	PK	NOT NULL	Identificador único del usuario
nombre	String		NOT NULL	Nombre del usuario
apellidos	String		NOT NULL	Apellidos del usuario
correo	String		ÚNICO, NOT NULL	Correo electrónico del usuario
contraseña	String		NOT NULL	Contraseña cifrada

Cuadro 16.1: NFR Tabla tbl_users

Tabla *tbl_resultados_escaneo*				
Campo	**Tipo de Dato**	**Clave**	**Restricciones**	**Descripción**
_id	ObjectId	PK	NOT NULL	Identificador del escaneo
domain	String		NOT NULL	URL del dominio escaneado
rutas	List[String]			Lista de subrutas obtenidas por crawling
vulnerabilidades	List[Object]			Lista de vulnerabilidades detectadas
umbral_confianza	Double			Umbral mínimo para considerar una predicción
hora	Date			Timestamp de cuándo se realizó el escaneo

Cuadro 16.2: Tabla tbl_resultados_escaneo

Tabla *tbl_resultados_patterns*				
Campo	**Tipo de Dato**	**Clave**	**Restricciones**	**Descripción**
_id	ObjectId	PK	NOT NULL	Identificador único del patrón
category	String			Categoría de la vulnerabilidad (ej. XSS, SQLi)
pattern	String			Texto del patrón/payload
synthetic	Boolean			Indica si el patrón es sintético o real
kind	String			Tipo de patrón (payload, description, unclear...)

Cuadro 16.3: Tabla tbl_resultados_patterns

Tabla *tbl_prueba*				
Campo	**Tipo de Dato**	**Clave**	**Restricciones**	**Descripción**
_id	ObjectId	PK	NOT NULL	Identificador único
request	String			URL objetivo que obtenemos de las fuentes OSINT
label	String			Etiqueta que cataloga la vulnerabilidad

Cuadro 16.4: Tabla tbl_prueba

17. Calidad de datos

Durante el desarrollo del sistema, se han manejado varios conjuntos de datos, correspondientes a diferentes etapas del procesamiento y análisis. Para garantizar la robustez del sistema, se ha realizado una evaluación exhaustiva de la **calidad de los datos**, atendiendo a aspectos como la integridad, consistencia, completitud y presencia de duplicados. A continuación, se describen los principales resultados obtenidos:

1. Proyecto.tbl_prueba.csv

Este dataset contiene un total de **131.043 registros** distribuidos en 3 columnas/registro (_id, request, label). Se trata de un conjunto de datos orientado al entrenamiento del modelo de machine learning. Podemos destacar algunas condiciones clave que se dan:

- **Ausencia de duplicados**: ninguno de los registros presentes en el dataset presenta un registro en donde sus atributos sean iguales al de otro registro.
- **Completitud**: todas las columnas, están completas, es decir, que todas ellas tienen un valor asociado distinto de *'null'*.
- **Tipos de datos**: podremos observar que hay diversos tipos de datos en nuestro dataset:
    - **Atributo _id**: atributo cuyo tipo de dato es ObjetId. Esto quiere decir que cuando se inserta un nuevo valor en esta colección se le asigna un id que será único en toda la colección de los datos.
    - **Atributo request**: atributo cuyo tipo de dato es String
    - **Atributo label:** atributo cuyo tipo de dato es String

Se considera un conjunto de entrenamiento válido, ya que posteriormente se emplearán los valores de dicha colección para la generación de datos sintéticos los cuales nos valdrán para poder entrenar a nuestro modelo de Machine Learning.

2. Proyecto.tbl_resultados_escaneo

Este dataset contiene **4 registros** y presenta las siguientes columnas principales (_id, domain, rutas, vulnerabilidades, umbral_confianza y hora). Algunas de estas columnas (como rutas y vulnerabilidades) contienen **arrays de objetos**, lo que permite registrar múltiples elementos por dominio escaneado. Podemos destacar algunas condiciones clave:

- **Ausencia de duplicados**: ninguno de los registros presentes en el dataset presenta un registro en donde sus atributos sean iguales al de otro registro.
- **Presencia de valores nulos**: El número de elementos presentes en los arrays de rutas[] y vulnerabilidades[] varía entre registros, en función de los datos encontrados para cada dominio. MongoDB almacena únicamente los elementos existentes, por lo que **no se generan índices vacíos ni se reservan posiciones nulas** si un dominio contiene, por ejemplo, solo 7 rutas o 3 vulnerabilidades (tenemos establecido que como máximo se puedan escanear un máximo de 50 subrutas y 7 vulnerabilidades).
- **Tipos de datos utilizados**:
    - **Atributo _id**: atributo cuyo tipo de dato es ObjetId. Esto quiere decir que cuando se inserta un nuevo valor en esta colección se le asigna un id que será único en toda la colección de los datos.
    - **Atributo domain**: tipo String, representando el dominio escaneado.
    - **Atributo rutas**: tipo Array de String, conteniendo las subrutas detectadas por crawling.
    - **Atributo vulnerabilidades**: tipo Array de objetos con campos anidados (ruta, tipo, score, mitigacion).
    - **Atributo umbral_confianza**: tipo Double, valor numérico que representa el nivel de score mínimo para considerar una predicción válida.
    - **Atributo hora**: tipo Date, usado para almacenar el timestamp del escaneo.

Este dataset es funcional y está bien estructurado para representar resultados de escaneo dinámicos. Su diseño con arrays permite la adaptabilidad a dominios con mayor o menor número de rutas, lo que resulta adecuado para la naturaleza variable del crawling web.

3. Proyecto.tbl_resultados_patterns

Con un total de **88.393 registros**, esta colección está compuesta por 5 campos principales (_id, category, pattern, synthetic y kind). Su propósito es almacenar patrones obtenidos del análisis semántico de payloads y clasificarlos según su tipo y origen. Las condiciones observadas son:

- **Alta completitud general**.
- **Ausencia de duplicados**: ninguno de los registros presentes en el dataset presenta un registro en donde sus atributos sean iguales al de otro registro.
- **Tipos de datos utilizados**:
    - **Atributo _id**: atributo cuyo tipo de dato es ObjetId. Esto quiere decir que cuando se inserta un nuevo valor en esta colección se le asigna un id que será único en toda la colección de los datos.
    - **Atributo category**: tipo String, que representa la categoría de la vulnerabilidad (por ejemplo, XSS, FILE_UPLOAD).
    - **Atributo pattern**: tipo String, contiene la descripción textual del patrón detectado.
    - **Atributo synthetic**: tipo Boolean, indica si el patrón ha sido generado de forma sintética o extraído de fuentes reales.
    - **Atributo kind**: tipo String, que describe la naturaleza del patrón (description, unclear, etc.).

La estructura de este dataset es adecuada para alimentar algoritmos que se vayan a usar para predicción o clasificación, dado que permite distinguir entre patrones reales y generados, así como categorizarlos por tipo y nivel de claridad. Por lo que nos será realmente útil para poder entrenar a nuestro modelo de Machine Learning.

Para entender esto mejor, vamos a mostrar un gráfico en el que podemos observar cómo varía la entropía en función del tipo de vulnerabilidad. La entropía es una métrica que nos ayuda a saber lo "complejo" o "mezclado" que está el contenido de la petición.

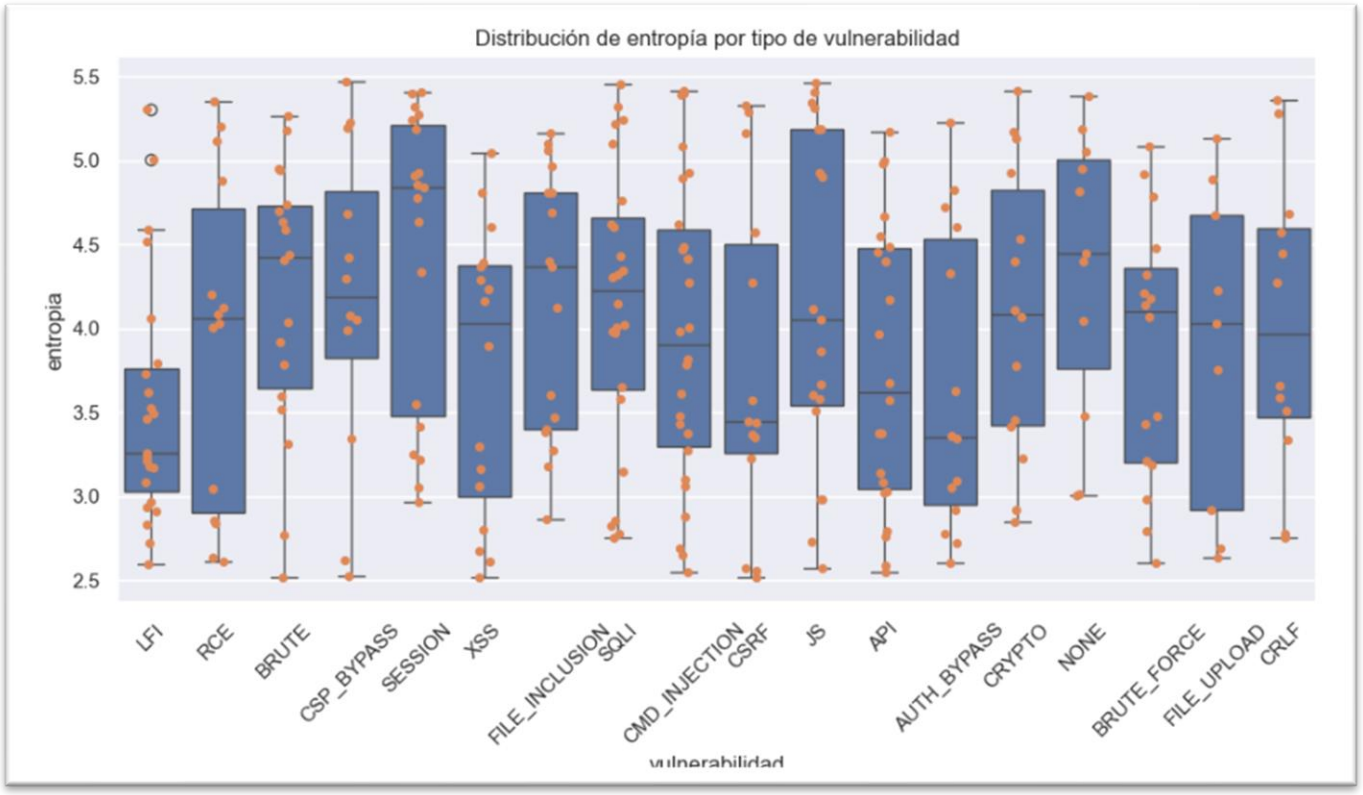


Figura 17.1: Distribución de la entropía por cada vulnerabilidad

Cada cajita representa una vulnerabilidad distinta (como SQLI, XSS, LFI, etc.) y dentro de ella vemos cómo se distribuyen sus valores de entropía. Cuanto más alta la caja, más variabilidad tiene esa vulnerabilidad. Además, los puntos naranjas encima nos enseñan los datos reales.

Lo interesante aquí es ver que algunas vulnerabilidades como SESSION, JS o CSP_BYPASS tienen entropías bastante altas, mientras que LFI o AUTH_BYPASS suelen estar más concentradas en valores bajos. Esto nos dice que la entropía sí puede ser útil para diferenciar ataques.

18. Entrenamiento del Modelo de Machine Learning

Para abordar la tarea de detección automática de vulnerabilidades a partir de peticiones HTTP, hemos optado por utilizar un modelo de **Regresión Logística**. Esta técnica se utiliza frecuentemente para problemas de clasificación supervisada y resulta muy adecuada cuando se trabaja con texto vectorizado y categorías bien definidas, como ocurre en este caso (por ejemplo: XSS, SQLi, CMD_INJECTION, etc.).

Una de las ventajas clave de la Regresión Logística es que ofrece resultados interpretables, maneja bien múltiples clases y tiene un rendimiento sólido en contextos donde la relación entre variables no es necesariamente lineal. Además, al tratarse de un modelo rápido y eficiente, se adapta bien al entorno de ejecución del proyecto, el cual está basado en máquinas locales y contenedores.

Seguidamente presentaremos una imagen en la que podemos ver cómo se comportan distintos modelos de machine learning al clasificar los datos. Cada recuadro representa un algoritmo diferente, y lo que se muestra es cómo cada uno divide el espacio según la clase.

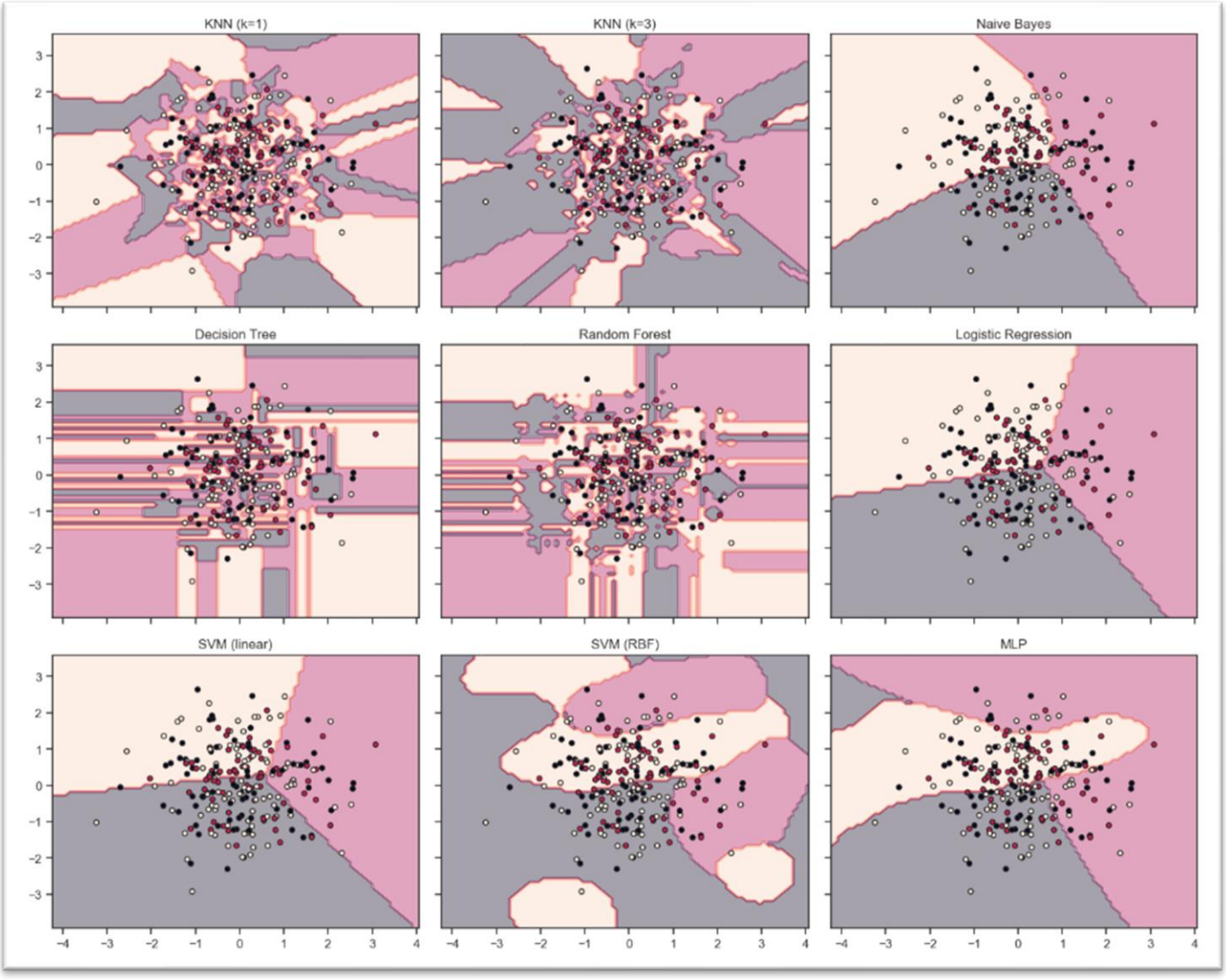


Figura 18.1: Entrenamientos de diferentes modelos de ML

A continuación, vamos a explicar de forma clara qué nos está mostrando la imagen anterior.
Si nos fijamos, podemos ver cómo distintos modelos de Machine Learning dividen el espacio para clasificar los datos. Lo primero que salta a la vista es que, según el modelo que usemos, las fronteras que separan las clases son más o menos limpias. Esto está relacionado con lo que llamamos **ruido**, que no es más que esas formas irregulares que aparecen en el gráfico y que realmente no representan un patrón real. El ruido suele aparecer cuando el modelo intenta **aprenderse de memoria** el conjunto de entrenamiento en vez de aprender la tendencia general.

Por ejemplo, modelos como **KNN con k=1** o **árboles de decisión** generan fronteras muy irregulares, mientras que otros como **Regresión Logística** o **SVM** hacen divisiones más suaves y ordenadas. Esta comparación nos sirve para ver de forma visual qué modelos generalizan mejor y cuáles tienden a sobreajustar demasiado.

El **color de fondo** indica qué clase predice el modelo en cada zona.

En el caso concreto de **Regresión Logística**, puede que veamos puntos fuera de su zona de color correcta. Esto es normal y está relacionado con que hemos usado **validación cruzada con K=3**: el dataset se divide en tres partes y, en cada iteración, una de ellas queda fuera para probar el modelo. Esos puntos de prueba no han sido vistos durante el entrenamiento, por lo que es lógico que algunos estén mal clasificados. Esto no es un fallo, sino una forma de medir cómo generaliza el modelo con datos nuevos.

Viendo el gráfico, se confirma que elegir **Regresión Logística** ha sido una buena decisión: es un modelo que no sobreajusta, genera una frontera limpia y clara y, además, es fácil de interpretar, algo muy útil para un sistema como el nuestro

18.1 Vectorización del texto

Antes de entrenar el modelo, las peticiones HTTP se transforman en vectores numéricos mediante la técnica **TF-IDF (Term Frequency – Inverse Document Frequency)**. Este método asigna mayor peso a los términos relevantes dentro del contexto del dataset, y reduce la importancia de palabras muy comunes o aquellas que ofrecen poca información.

El uso de TF-IDF permite que el modelo pueda captar la presencia de patrones sospechosos (como secuencias características de inyecciones SQL o etiquetas <script> en ataques XSS) con mayor precisión, sin necesidad de crear reglas manuales o listas predefinidas.

18.2 Ajuste mediante validación cruzada

Para asegurarnos de que el modelo de clasificación funciona bien no solo con los datos con los que se entrena, sino también con datos nuevos, hemos utilizado una técnica conocida como **validación cruzada por K particiones** (*K-Fold Cross-Validation*). Esta técnica consiste en dividir el conjunto de datos en varias partes o "folds" y entrenar el modelo varias veces, usando cada vez una parte distinta como test y el resto como entrenamiento.

En nuestro caso se ha utilizado un **K=3**, es decir, se han generado tres particiones diferentes del dataset. El modelo se ha entrenado y evaluado *k* veces(con esto entendemos que tenemos *k* iteraciones), intercambiando las particiones para que cada una se use como test en una de las vueltas. Esto permite obtener una medida más precisa y equilibrada del rendimiento, ya que se prueba el modelo sobre distintos subconjuntos del total.

Además, este proceso se ha combinado con una **búsqueda automática de hiperparámetros** mediante la herramienta **GridSearchCV**. Lo que hace es probar distintas configuraciones internas del modelo —por ejemplo, el número máximo de iteraciones— y quedarse con la que mejor resultado da según las métricas de validación cruzada.

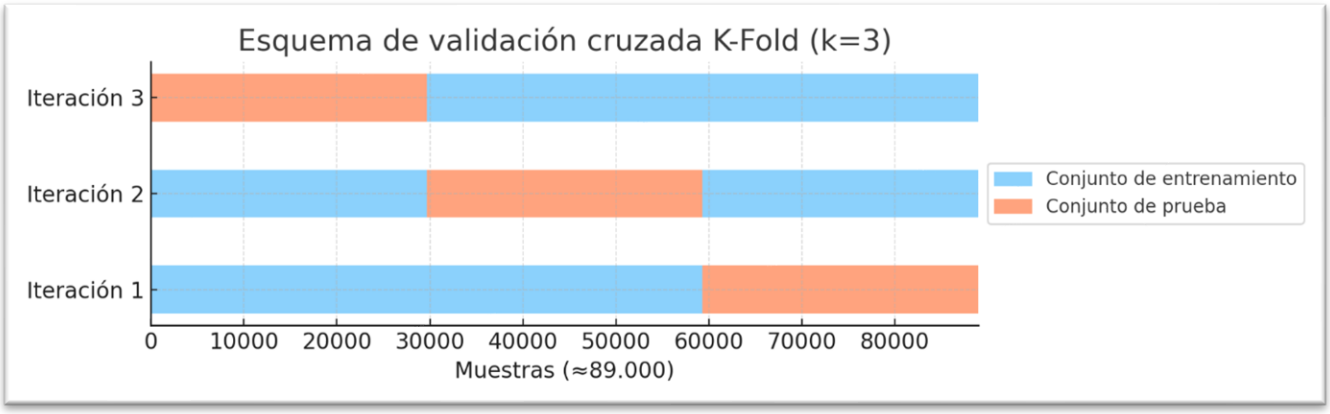


Figura 18.2: Técnica de validación cruzada

Una vez se encuentra la mejor combinación de parámetros, se guarda todo el pipeline completo (incluyendo tanto la transformación de texto como el modelo entrenado) en un archivo .joblib. Esto permite que el sistema pueda reutilizar ese modelo entrenado sin tener que repetir todo el proceso cada vez que se quiera hacer un análisis.

18.3 Resultados del entrenamiento

Una vez finalizamos el entrenamiento del modelo, tenemos varias representaciones que hemos obtenido y podemos obtener información bastante clara como es la siguiente:

```
✓ Cargados 88,393 patrones y 131,043 ejemplos reales

======== DISTRIBUCIÓN POR 'kind' ========
  payload     : 75,060  ( 84.9 %)
  description :  7,368  (  8.3 %)
  unclear     :  5,965  (  6.7 %)

✓ Payloads disponibles para sintéticos: 75,060
✓ Rutas reales válidas: 8
✓ Dataset final: 92,079 ejemplos (tras deduplicar)
✓ Train: 64,455  ·  Test: 27,624
Fitting 3 folds for each of 6 candidates, totalling 18 fits
✓ Best params: {'clf__estimator__C': 1, 'tfidf__max_features': 50000}

========== REPORT ==========
                precision    recall  f1-score   support

           API       0.80      0.85      0.82       401
   AUTH_BYPASS       0.04      0.05      0.05       400
         BRUTE       0.50      0.95      0.65      1264
   BRUTE_FORCE       0.73      0.28      0.40      1666
       CMD_INJ       1.00      0.64      0.78      1309
 CMD_INJECTION       0.77      0.78      0.77      1752
          CRLF       1.00      1.00      1.00      1346
        CRYPTO       0.10      0.17      0.13       406
    CSP_BYPASS       0.03      0.04      0.03       389
          CSRF       0.05      0.05      0.05       399
      FILE_INC       0.31      0.04      0.06       306
 FILE_INCLUSION       0.24      0.13      0.17       700
       FILE_UP       0.94      0.99      0.96      1338
   FILE_UPLOAD       0.05      0.02      0.03       471
            JS       0.97      0.96      0.97      1708
           LFI       0.92      0.99      0.95      4221
          NONE       1.00      0.64      0.78        22
           RCE       1.00      1.00      1.00      1258
       SESSION       0.05      0.06      0.05       400
          SQLI       0.83      0.88      0.85      3342
           XSS       0.99      0.99      0.99      4526

      accuracy                           0.78     27624
     macro avg       0.59      0.55      0.55     27624
  weighted avg       0.79      0.78      0.77     27624
```

Figura 18.3: Representación de los datos obtenidos tras el entrenamiento del modelo de ML

Durante el entrenamiento del modelo de Machine Learning se cargaron **88.393 patrones** y **131.043 ejemplos reales** extraídos de tráfico HTTP. Estos se agruparon por tipo (*kind*), siendo este un atributo que se encuentra dentro de una de las tablas de nuestra base de datos, en donde la mayoría de los ejemplos reales equivalen a *payloads* (**84,9 %**), seguidos de descripciones (**8,3 %**) y casos ambiguos (**6,7 %**). Para la generación de datos sintéticos se dispuso de **75.060 payloads** y se identificaron **8 rutas reales válidas**, es decir, que se han encontrado 8 rutas a las que todavia podríamos tener acceso hoy en día.

Tras la limpieza de duplicados, el dataset final quedó compuesto por **92.079 ejemplos**, de los cuales el **70 %** (**64.455**) se destinó a entrenamiento y el **30 %** restante (**27.624**) a prueba. El entrenamiento se realizó con validación cruzada **K-Fold** usando **K=3**, probando 6 combinaciones de hiperparámetros para un total de **18 ejecuciones**. La mejor configuración encontrada fue un valor de C=1 en el clasificador y un máximo de **50.000 características** para el algoritmo TF-IDF.

En los resultados, cada fila corresponde a una clase de vulnerabilidad y se muestran las siguientes métricas:

1. **precision** → cuántas veces acierta el modelo cuando predice una clase concreta.
2. **recall** → cuántos aciertos logra sobre todos los casos reales de esa clase.
3. **f1-score** → valor que combina *precision* y *recall* para equilibrar ambos.
4. **support** → número de ejemplos reales de esa clase presentes en el conjunto de prueba.

El modelo alcanzó una *accuracy* global del **78 %**, con un *macro average* de **0,59** y un *weighted average* de **0,79**.

Clases como **XSS, LFI, RCE o CRLF** destacan con *f1-scores* cercanos a 1, lo que indica un excelente rendimiento en su detección. En cambio, clases como **AUTH_BYPASS, CSP_BYPASS o FILE_UPLOAD** presentan valores bajos, lo que refleja la necesidad de disponer de más datos o de un mejor tratamiento de estas categorías para mejorar su identificación.

Una vez analizados los resultados numéricos del modelo y su capacidad de clasificación, pasamos ahora a estudiar cómo se comportan los distintos tipos de ataques en función de sus características internas.

En la gráfica que se presenta a continuación, nos hemos centrado solo en 4 vulnerabilidades ya que son de las más conocidas y de las que más vulnerabilidades puede haber: CMD_INJECTION, SQLI, XSS y LFI. La idea es ver si se comportan diferente entre sí en función de variables como la entropía, el número de parámetros o la longitud de la URL.

Los atributos que estudiaremos son los siguientes:

1. **longitud_request**
    - Longitud total de la solicitud HTTP (número de caracteres que tiene toda la request).
    - Las peticiones más largas suelen implicar inyecciones más complejas o múltiples parámetros.
2. **entropía**
    - Medida de lo "desordenada" o "aleatoria" que es la cadena de caracteres.
    - Un valor alto suele indicar cadenas con caracteres especiales, ofuscación o datos codificados, típico en intentos de evasión.
3. **num_parametros**
    - Cantidad de parámetros que se envían en la request (por ejemplo, en la URL o en el cuerpo de la petición).
    - Algunos ataques requieren más parámetros manipulados; otros pueden funcionar con uno solo.
4. **n_caracteres_especiales**
    - Número de caracteres especiales presentes (<, >, ', ", %, ;, {, }, etc.).
    - Un uso elevado puede ser indicio de inyecciones, scripts o intentos de romper la sintaxis.

Ahora procederemos a realizar el análisis de fila por fila para que quede clara la interpretación de los datos que estamos manejando, en este caso haciendo únicamente referencia a las clases SQLi, XSS, CMD_INJECTION y LFI:

El gráfico es una **matriz de dispersión (pairplot)**:

- **Diagonal principal**: Histogramas/densidades de cada atributo por clase.
- **Celdas fuera de la diagonal**: Comparaciones entre dos atributos diferentes (dispersión de puntos).

Fila 1: longitud_request

- **Diagonal (longitud_request vs longitud_request)**: Distribución de la longitud de las peticiones para cada tipo de ataque.
    - SQLI y XSS tienden a longitudes más variadas.
    - CMD_INJECTION y LFI más concentradas en un rango.
- **longitud_request vs entropía**
    - Vemos que en general, las peticiones con más longitud pueden tener entropías más altas, pero no siempre.
    - SQLI y XSS muestran más dispersión.
- **longitud_request vs num_parametros**
    - Los ataques no usan demasiados parámetros (se concentran en valores bajos)
    - Cuanta más longitud haya en la *request*, la posibilidad de que haya más parámetros aumenta.
- **longitud_request vs n_caracteres_especiales**
    - XSS y SQLI tienen más puntos con alto número de caracteres especiales, sobre todo en longitudes grandes.

Fila 2: entropía

- **entropía vs longitud_request**
    - (Comentado en Figura 1)
- **Diagonal (entropía vs entropía)**
    - XSS tiende a entropías ligeramente más altas.
    - CMD_INJECTION y LFI más concentrados en entropías medias.
- **entropía vs num_parametros**
    - No parece haber una correlación clara.
    - Casi todos los ataques tienen pocos parámetros, pero en XSS y SQLI hay más dispersión de entropía.
- **entropía vs n_caracteres_especiales**
    - Relación moderada: más caracteres especiales → más entropía.
    - XSS suele tener altos valores en ambos.
    -

Fila 3: num_parametros

- **num_parametros vs longitud_request**
    - (Comentado en Figura 1)
- **num_parametros vs entropía**
    - Sin patrón fuerte, salvo ligeros aumentos para SQLI.
- **Diagonal (num_parametros vs num_parametros)**
    - La mayoría de los ataques usan de 1 a 3 parámetros.
    - Pocos casos extremos con más de 5.
- **num_parametros vs n_caracteres_especiales**
    - Algunos ataques con más parámetros también tienen más caracteres especiales, pero no es la norma.

Fila 4: n_caracteres_especiales

- **n_caracteres_especiales vs longitud_request**
  - Relación clara: a más longitud, más caracteres especiales, sobre todo en XSS y SQLI.
- **n_caracteres_especiales vs entropía**
  - Fuerte relación positiva: más caracteres especiales conlleva a una mayor entropía.
- **n_caracteres_especiales vs num_parametros**
  - Relación débil, pero ataques con muchos parámetros pueden incluir más caracteres especiales.
- **Diagonal (n_caracteres_especiales vs n_caracteres_especiales)**
  - XSS y SQLI destacan con picos altos en caracteres especiales, mientras que CMD_INJECTION y LFI son más moderados.

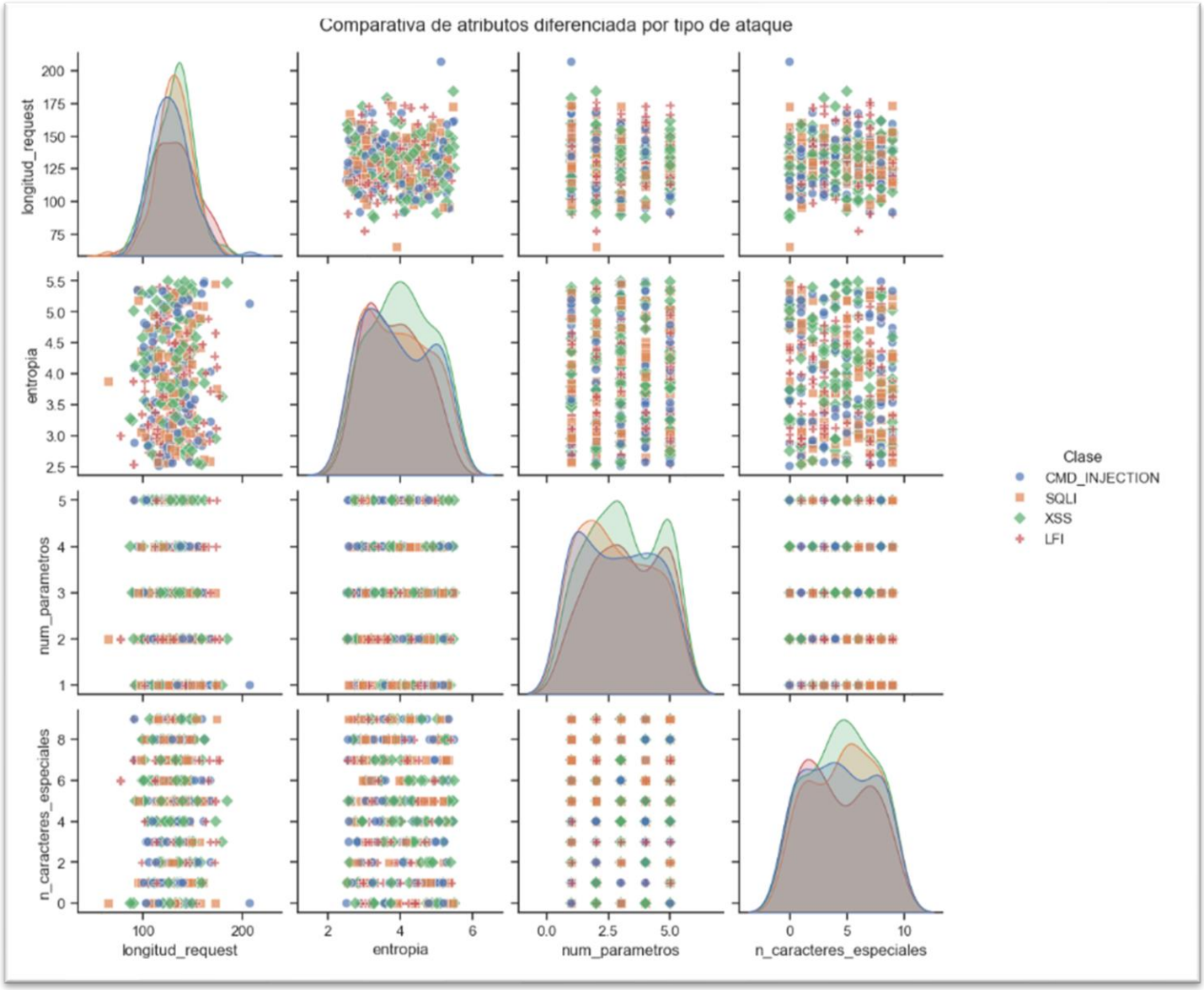


Figura 18.4: Comparativa de atributos entre diferentes vulnerabilidades

Al mirar los colores y formas de los puntos, vemos que cada tipo de ataque tiene ciertos patrones. Por ejemplo, XSS tiende a tener entropías más altas, mientras que CMD_INJECTION y LFI se reparten más en longitud. Este tipo de visual ayuda a entender si el modelo de ML va a ser capaz de distinguirlos bien con los atributos que hemos usado.

Una vez hecha esta comparativa, vamos a proceder a realizarla con todas las vulnerabilidades con la que nuestro modelo de ML se ha entrenado pudiendo obtener el siguiente gráfico:

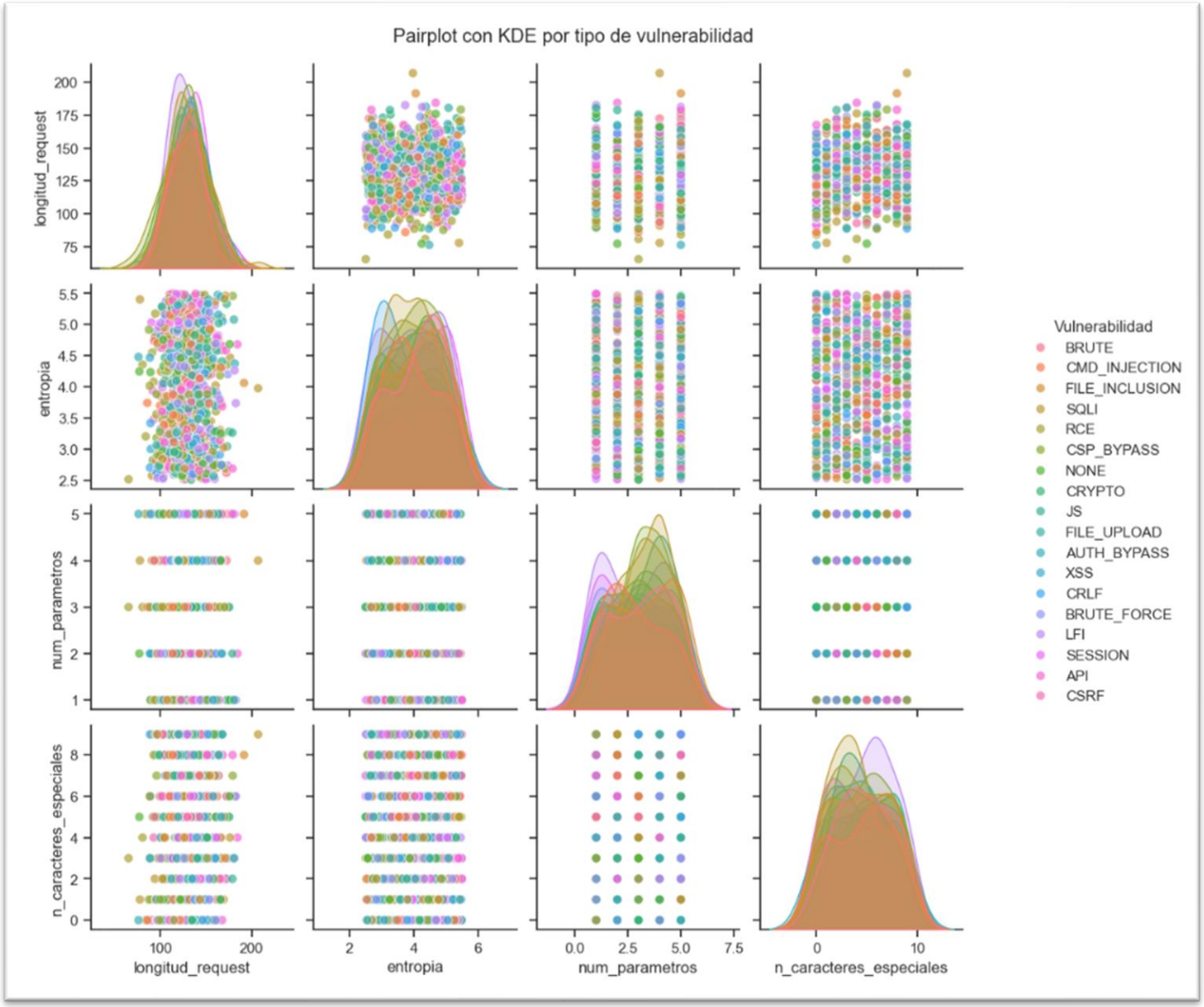


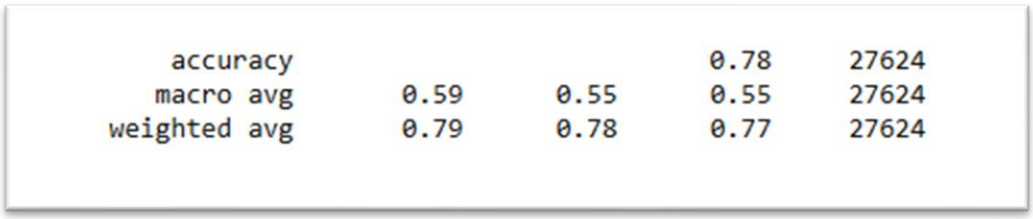
Figura 18.5: Comparativa de atributos entre todas las vulnerabilidades

Para cada par de atributos (por ejemplo, entropía vs longitud de la petición), podemos ver cómo se agrupan los puntos según el tipo de ataque. Las curvas suaves que se ven en la diagonal (gracias al KDE) nos muestran la densidad de cada clase: o sea, dónde se concentran más.

Esta visual viene genial para ver si hay solapamiento entre tipos de vulnerabilidades. Aunque hay bastante mezcla, se notan algunas diferencias claras en ciertas combinaciones, lo que da pistas al clasificador para que pueda hacer bien su trabajo.

18.4 Métricas de rendimiento

Haciendo referencia a la imagen de los resultados obtenidos en el entrenamiento vamos a destacar las ultimas 3 líneas que son las que equivalen a las métricas de rendimiento:



accuracy			0.78	27624
macro avg	0.59	0.55	0.55	27624
weighted avg	0.79	0.78	0.77	27624

Figura 18.6: Métricas de rendimiento

- **accuracy (0.78)**: es el porcentaje de predicciones correctas sobre el total de ejemplos del conjunto de prueba. En este caso, significa que el modelo acierta aproximadamente 78 de cada 100 predicciones.
- **macro avg (0.55)**: lo definimos como la media aritmética de las métricas (*precision*, *recall* o *f1-score*) calculadas **por clase**, sin tener en cuenta cuántos ejemplos tiene cada clase. Todas las clases pesan lo mismo

  Esto es útil para ver el rendimiento medio del modelo sin que las clases con más datos dominen el resultado.
- **weighted avg (0.77)**: corresponde con la media ponderada de las métricas (*precision*, *recall* o *f1-score*), **teniendo en cuenta el número de ejemplos de cada clase** (*support*).

Aquí las clases más frecuentes tienen más peso, por eso este valor suele ser más alto que el *macro average* si el modelo funciona bien en las clases mayoritarias.

Por eso vemos que el macro avg (0.55) es más bajo: el modelo falla más en clases minoritarias (por ejemplo, AUTH_BYPASS, CSRF, CSP_BYPASS) y eso tira hacia abajo el promedio simple. Pero en el weighted avg (0.77), como las clases más frecuentes tienen mejores resultados, el promedio sube.

Estas métricas indican que el modelo logra un rendimiento general bastante bueno, especialmente si se tiene en cuenta la amplia variedad de clases.

18.5 Matriz de confusión

A continuación, mostraremos la matriz de confusión, la cual está incluida en el entrenamiento y permite visualizar cómo el modelo ha clasificado cada tipo de ataque.

Una matriz de confusión es una tabla utilizada para evaluar el rendimiento de un modelo de clasificación, mostrando las predicciones correctas e incorrectas organizadas por clase. Cada fila representa las clases reales y cada columna las predicciones realizadas por el modelo. Los valores diagonales indican el número de aciertos (predicciones correctas), mientras que los valores fuera de la diagonal muestran las confusiones, es decir, casos en los que el modelo clasificó erróneamente un ejemplo como otra clase distinta.

Por ejemplo, se observa que la clase **SQLi** se ha clasificado correctamente en la mayoría de los casos (2940 aciertos), aunque ha tenido algunas confusiones con clases como **CMD_INJECTION** o **FILE_INCLUSION**. En el caso de XSS, el modelo ha logrado 1474 aciertos directos, con muy pocas confusiones, lo que refuerza su fiabilidad en este tipo de ataques.

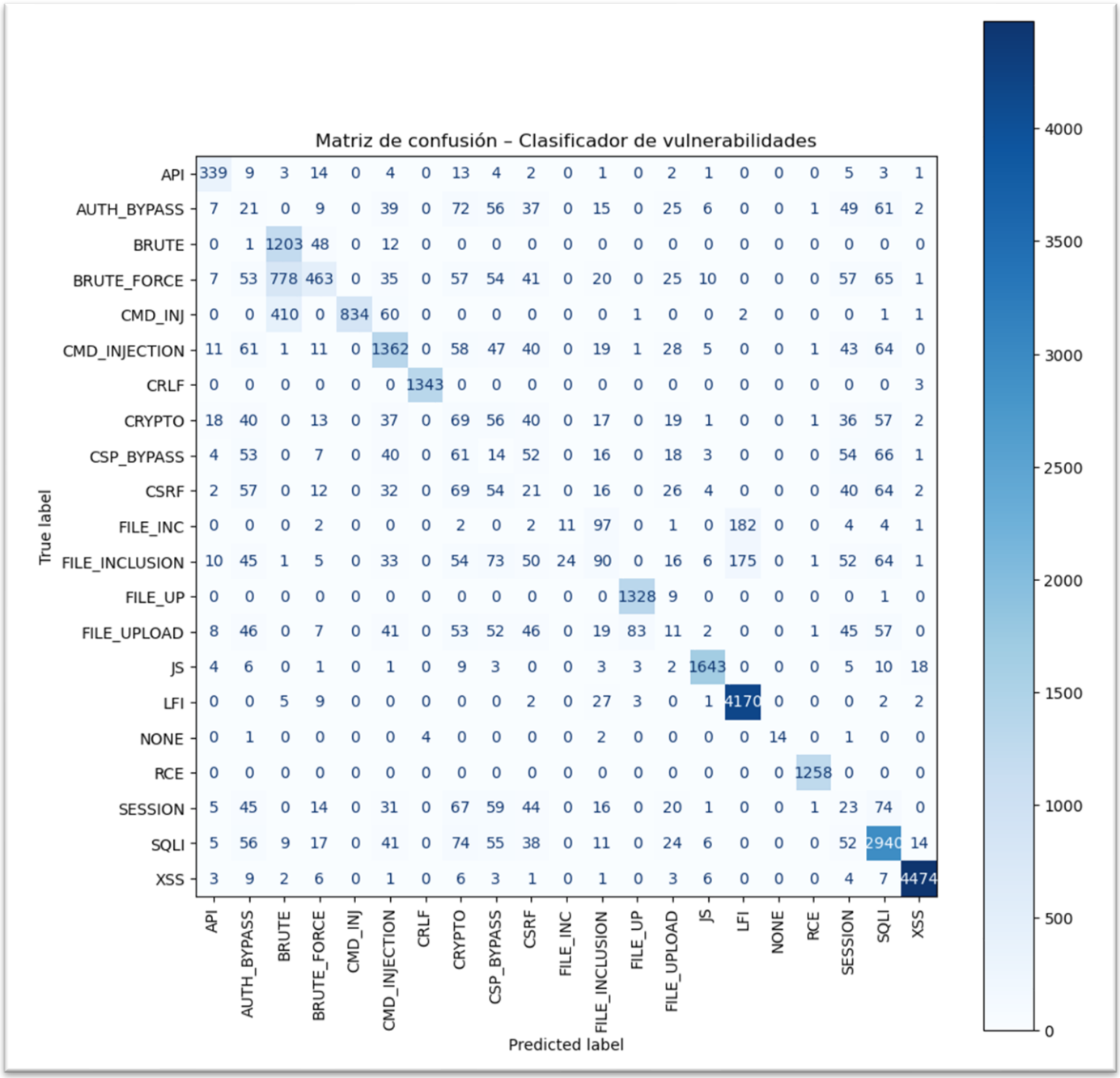


Figura 18.7: Matriz de confusión

19. Limitaciones legales/técnicas

A lo largo del desarrollo del proyecto nos hemos encontrado con varias limitaciones que han influido tanto en cómo se ha implementado el sistema como en la forma en la que se han podido hacer las pruebas.

1. Limitaciones legales

Una de las principales restricciones ha sido de tipo legal. En España, escanear dominios que no son tuyos sin permiso puede considerarse una actividad intrusiva e incluso ilegal, dependiendo de cómo y con qué intensidad se hagan las pruebas. Por ese motivo, se han seguido **una serie de principios de precaución muy claros**:

- No se ha escaneado ningún dominio real de terceros sin su autorización.
- Todas las pruebas se han realizado en **entornos controlados**, como DVWA o páginas vulnerables configuradas manualmente para simulación.
- En un escenario profesional, se entiende que este sistema **solo debería utilizarse con consentimiento por escrito** del propietario del dominio, dentro del marco del hacking ético.

Además, se ha tenido en cuenta la legislación vigente en materia de **protección de datos y ciberseguridad**, como el RGPD y la directiva NIS2. Esto ha reforzado la decisión de limitar el uso de la herramienta exclusivamente a contextos académicos o de pruebas bajo control.

2. Limitaciones técnicas

En el plano técnico también han aparecido algunos retos. El primero de ellos ha sido la **falta de acceso a datasets reales de ataques**, ya que por motivos de seguridad y privacidad no es fácil encontrar información de este tipo. Para suplirlo, se ha optado por **generar patrones sintéticos** o trabajar con fuentes públicas como CVE, KEV o Exploit-DB.

Otro aspecto importante ha sido la **capacidad limitada del entorno de ejecución**. El proyecto se ha desarrollado íntegramente en ordenadores personales y contenedores Docker locales, por lo que el volumen de escaneos y pruebas ha tenido que adaptarse a los recursos disponibles. Esto no ha impedido comprobar que el sistema funciona, pero sí ha limitado su escalado y pruebas con grandes volúmenes de datos.

20. Manual de usuario

20.1 Jupyter Notebook

Este manual está pensado para cualquier persona que quiera utilizar el sistema desde el entorno de **Jupyter Notebook**, donde se encuentra implementado el código Python principal. A lo largo de las siguientes fases se explica cómo preparar el entorno, entrenar el modelo y escanear dominios web en busca de posibles vulnerabilidades.

20.1.1 Fase 1: Preparación de los elementos a usar

Antes de empezar, asegúrate de tener a mano los siguientes elementos:

- **Anaconda** o **Jupyter Notebook** instalado en tu equipo.
- **Conexión a Internet**, ya que se descargan datasets públicos durante el proceso.
- Una **terminal limpia con entorno virtual activado** (idealmente creado con Anaconda para evitar conflictos entre librerías).
- Todas las librerías necesarias ya instaladas:
  pandas, tqdm, re, requests, scikit-learn, entre otras.

20.1.2 Fase 2: Obtención de datos

Abrimos el archivo “ML entrenado.ipynb” desde nuestro navegador (Para ello tendremos que iniciar previamente Jupyter Notebook). Seguidamente, dirígete a la sección titulada **"GENERACIÓN DE DATOS PARA PODER ENTRENAR A LA IA/ML"** y ejecuta las celdas que se encuentran debajo.

En este paso, el sistema descargará tráfico HTTP real desde fuentes públicas como **CSIC**, **ECML** o **CICIDS**. Este tráfico contiene tanto accesos válidos como simulaciones de ataques. Una vez descargado, el sistema utilizará expresiones regulares para clasificar automáticamente cada petición según el tipo de vulnerabilidad: **SQL Injection**, **XSS**, **File Upload**, etc.

20.1.3 Fase 3: Preparación del Dataset

Una vez clasificadas todas las peticiones, se organiza el contenido y se guarda en un archivo llamado “full_dataset.csv”. En esta fase:

- Se limpian los datos.
- Se eliminan entradas duplicadas.
- Se equilibra el número de ejemplos positivos y negativos para cada clase.

Este archivo será la base con la que se entrenará el modelo.

20.1.4 Fase 4: Entrenamiento del Modelo

Después de preparar el dataset, se continúa ejecutando las celdas siguientes del notebook. Aquí se realiza:

- **Vectorización** del texto usando técnicas como **TF-IDF**.
- Construcción de un **dataset sintético** con etiquetas para entrenamiento supervisado.
- Entrenamiento del modelo usando algoritmos como **Random Forest** o **Gradient Boosting**.
- Guardado del modelo entrenado en un archivo .joblib para su posterior uso.

Al finalizar, el notebook te mostrará métricas como **accuracy** y **precision**, lo que te permitirá evaluar la calidad del modelo.

20.1.5 Fase 5: Escaneo de un Dominio

Una vez el modelo está entrenado y almacenado, ejecuto la sección donde puedo introducir un dominio (por ejemplo, https://demo.owasp-juice.shop). El sistema realiza las siguientes acciones:

1. **Crawling del dominio**: busca todas las subrutas accesibles.
2. **Predicción de vulnerabilidades**: analiza el texto de cada subruta usando el modelo entrenado.
3. **Mitigación**: si el sistema detecta que una ruta es vulnerable, consulta a un modelo LLM (como LLaMA) para darnos una sugerencia de cómo mitigar esa vulnerabilidad.
4. **Almacenamiento de resultados**: todos los resultados se guardan automáticamente en la base de datos MongoDB.

20.1.6 Fase 6: Visualización y cierre

Al final del proceso podremos:

- Consultar el archivo .joblib generado.
- Ver los resultados del escaneo en MongoDB, incluyendo las rutas que han sido analizadas, las vulnerables encontradas y las recomendaciones generadas.

20.1.7 Recomendaciones

- Asegúrate de escanear solo dominios locales o que tengas autorización expresa.
- No modifiques las rutas del sistema a menos que sepas lo que estás haciendo.
- Puedes repetir el entrenamiento si agregas nuevos datos a los datasets OSINT.

20.2 Página Web

Este manual tiene como objetivo guiar al usuario paso a paso en la utilización de la aplicación web desarrollada para detectar vulnerabilidades en páginas web mediante crawling, análisis automático con modelos de Machine Learning y generación de recomendaciones con modelos de lenguaje (LLM). La aplicación ofrece una experiencia intuitiva que permite escanear dominios web y visualizar posibles fallos de seguridad detectados, sin necesidad de conocimientos técnicos avanzados.

20.2.1 Fase 1: Preparación del entorno

Antes de comenzar a utilizar la aplicación web, es necesario tener correctamente instalado y configurado el entorno que permite ejecutarla. La aplicación se encuentra contenida dentro de un conjunto de servicios orquestados con **Docker y Docker Compose**, lo cual facilita la instalación y despliegue, evitando problemas de dependencias.

Para ejecutar la aplicación, el usuario debe tener instalado:

- **Docker**: Para ejecutar los contenedores de back-end, front-end y base de datos.
- **Docker Compose**: Para levantar todos los servicios definidos en un único archivo docker-compose.yml.
- **Acceso a Internet**: Necesario para que el sistema pueda descargar los contenedores e interactuar con dominios externos.

Una vez cumplidos estos requisitos, basta con abrir una terminal en el directorio del proyecto y ejecutar el siguiente comando: ‘docker-compose up --build’.

Esto iniciará el sistema, incluyendo la API de análisis (FastAPI), la interfaz web (React) y la base de datos MongoDB, dejando la aplicación disponible en http://localhost:8080.

20.2.2 Fase 2: Acceso a la aplicación (Inicio de sesión)

Una vez iniciado el sistema, el usuario debe abrir el navegador web e ingresar a http://localhost:8080, donde se encontrará con una **pantalla de inicio de sesión**. Este formulario requiere que el usuario introduzca su **correo electrónico** y **contraseña**.

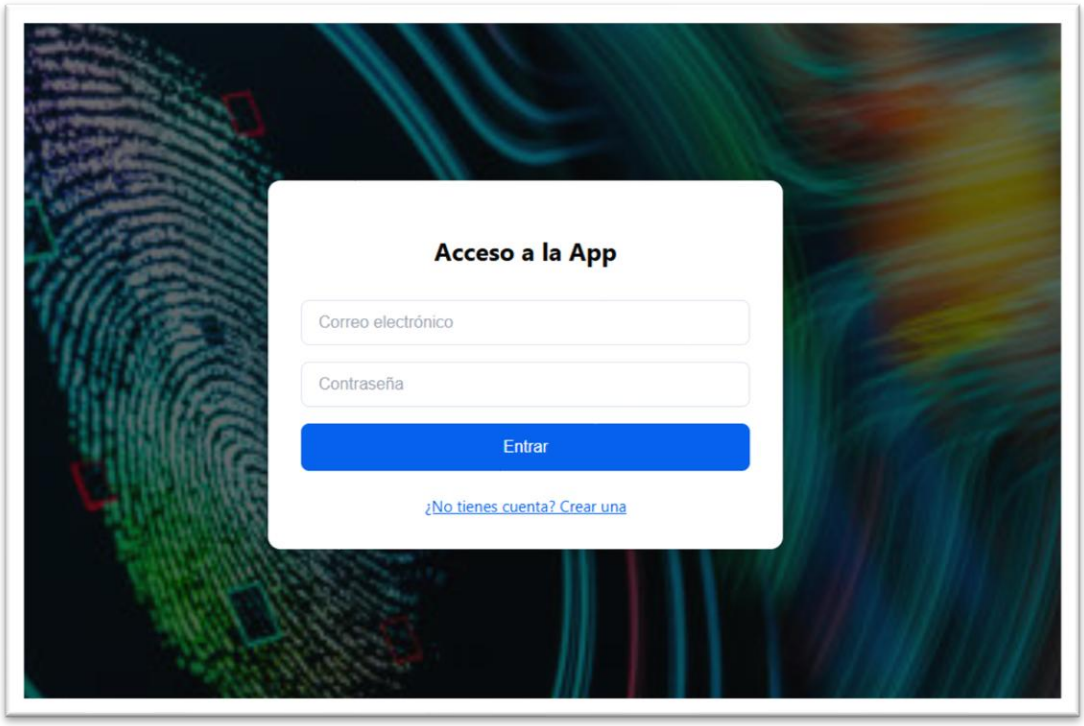


Figura 20.1: Página de login

En caso de no tener cuenta, la interfaz ofrece un enlace "¿No tienes cuenta? Crear una", que redirige al formulario de registro. Esta capa de autenticación garantiza que cada escaneo realizado quede asociado a un usuario concreto, y facilita el almacenamiento personalizado del historial de análisis.

20.2.3 Fase 3: Registro de usuario

La pantalla de registro solicita los siguientes datos personales:

- Nombre
- Apellidos
- Fecha de nacimiento
- Correo electrónico
- Contraseña

Una vez completado el formulario, se debe hacer clic en el botón **“Registrarse”**. Si todos los campos son válidos, el usuario será redirigido nuevamente a la pantalla de inicio de sesión para autenticarse con las credenciales recién creadas.

Este proceso garantiza que los datos almacenados en el sistema (escaneos realizados, historial de análisis, etc.) se asocien correctamente a un perfil de usuario concreto

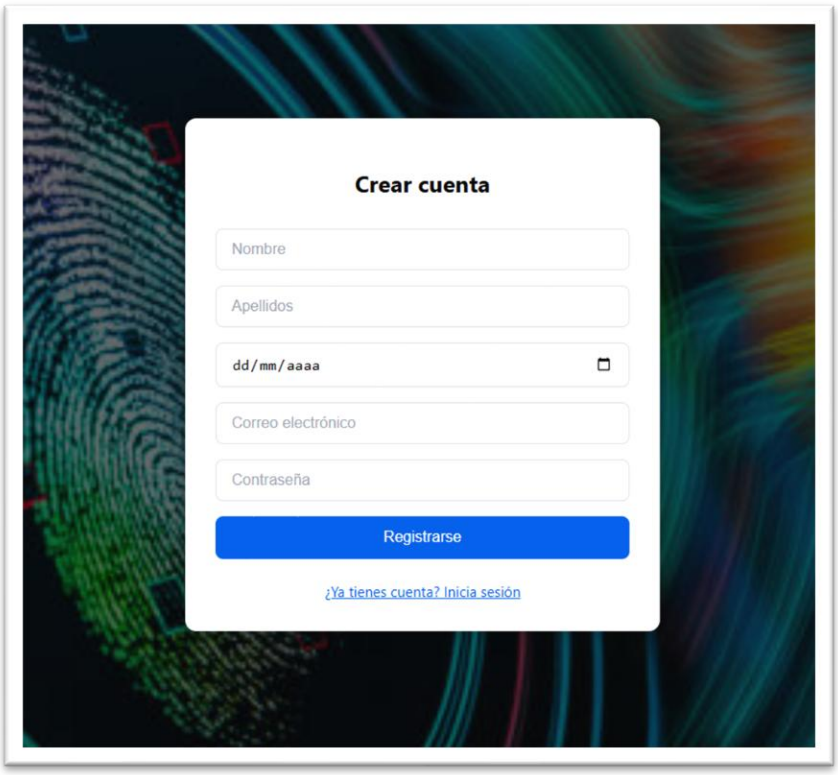


Figura 20.2: Página de registro de usuario

20.2.4 Fase 4: Escaneo de dominios

Tras iniciar sesión correctamente, el sistema redirige automáticamente al usuario a la **pantalla principal de escaneo**.

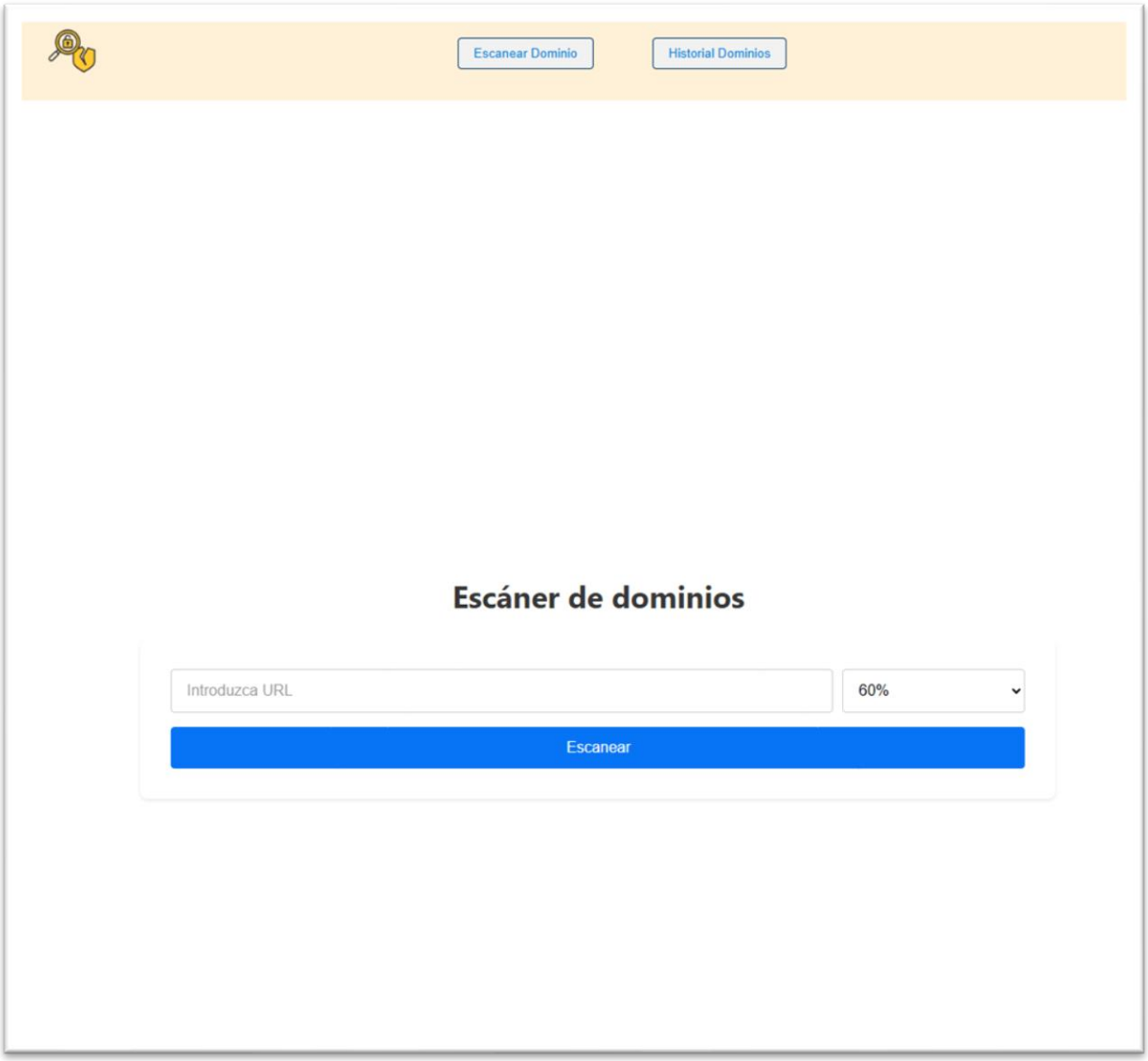


Figura 20.3: Página de inicio

Aquí se presenta un formulario sencillo que permite introducir:

- Una **URL o dominio** que se desea analizar (por ejemplo, https://demo.owasp-juice.shop)
- Un **umbral de confianza (%)**, que permite ajustar el nivel mínimo de certeza con el que el modelo debe identificar una vulnerabilidad para considerarla relevante.

Al pulsar el botón **“Escanear”**, se activa el flujo automatizado del sistema que consiste en:

1. **Crawling del dominio**: Se recorren y extraen automáticamente todas las rutas accesibles desde el dominio inicial.
2. **Análisis automático**: Cada ruta encontrada es evaluada por un modelo de Machine Learning que predice si contiene una posible vulnerabilidad, indicando además su tipo (por ejemplo, File Upload, SQLi, XSS…).
3. **Mitigación automática**: Si se detectan vulnerabilidades, el sistema consulta un modelo LLM (Large Language Model) como Ollama para generar recomendaciones detalladas de mitigación.
4. **Almacenamiento estructurado**: Todos los resultados (rutas analizadas, vulnerabilidades detectadas, confianza y mitigaciones generadas) se almacenan automáticamente en la base de datos MongoDB en la colección tbl_resultados_escaneo.

20.2.5 Fase 5: Visualización del resumen de escaneo

Una vez finalizado el análisis, el sistema redirige al usuario a una sección titulada **"Resumen del Último Escaneo"**, donde se visualizan los resultados del dominio más recientemente procesado. Esta vista contiene:

- La **URL escaneada** y la **fecha y hora del escaneo**.

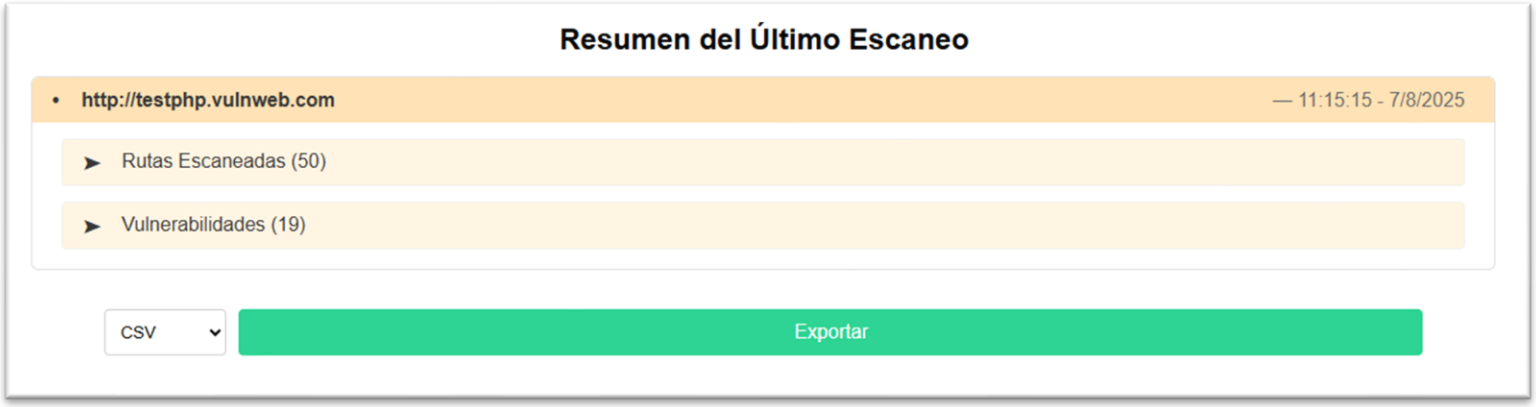


Figura 20.4: Página del resumen del dominio escaneado

- Una tabla detallada con todas las **rutas encontradas** durante el crawling.

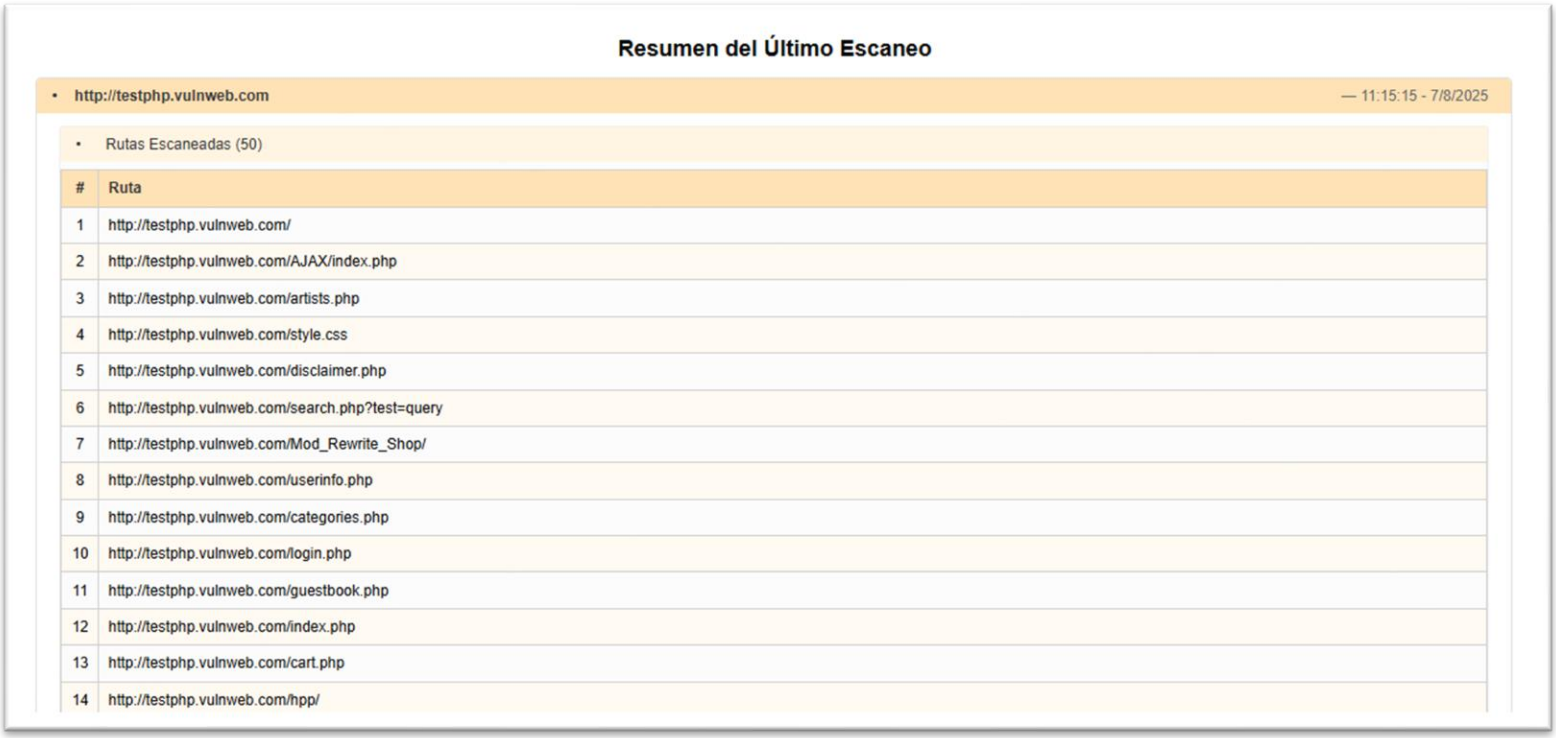
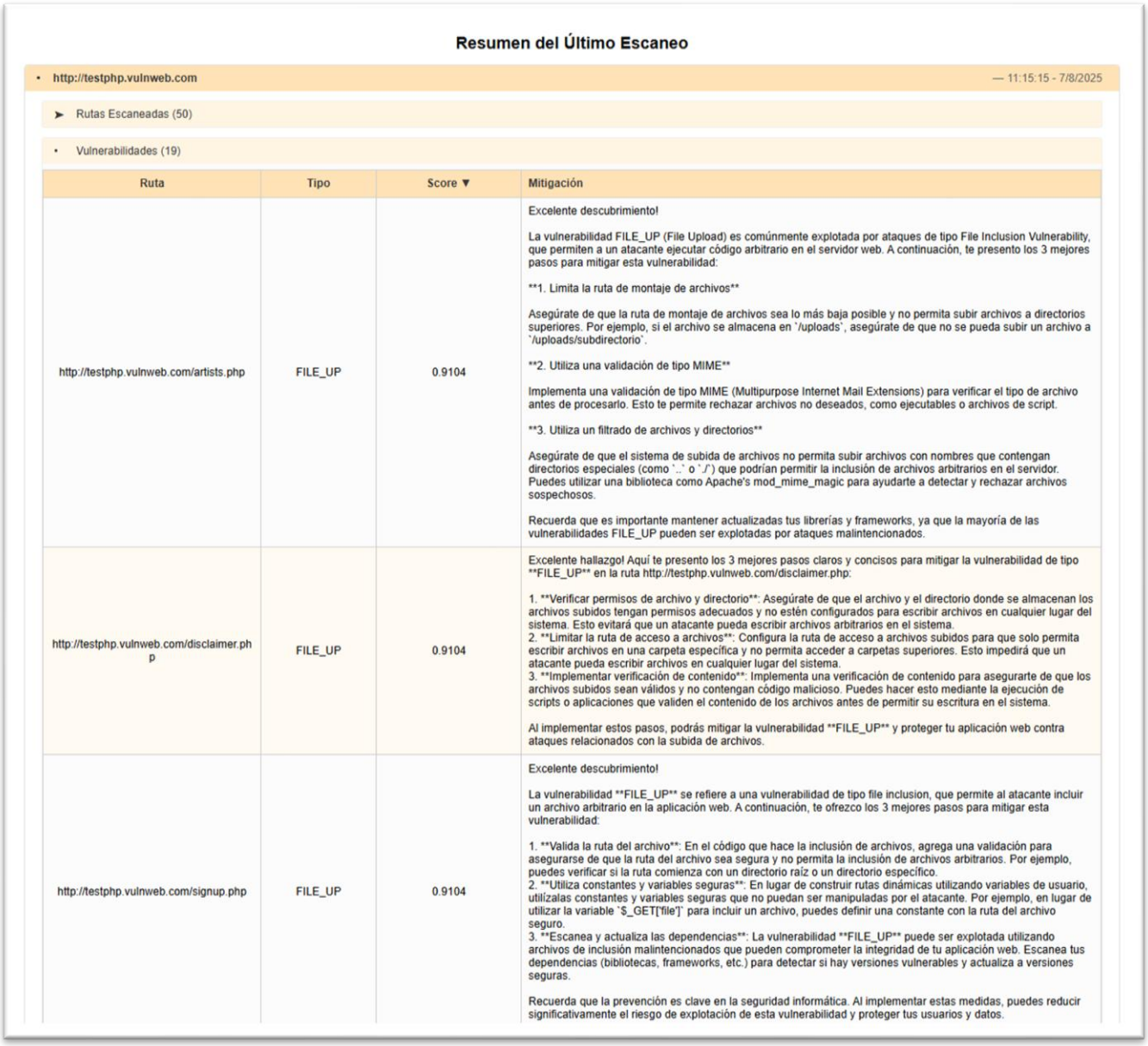


Figura 20.5: Pestaña en donde se muestran las subrutas escaneadas del dominio objetivo

- Una segunda tabla con las **vulnerabilidades detectadas**, que incluye:
    - Ruta afectada
    - Tipo de vulnerabilidad
    - Nivel de confianza (Score)
    - Mitigación generada por Llama3, explicada en lenguaje natural

Además, en la parte inferior se ofrece un botón de **exportación en formato CSV o PDF**, lo cual permite al usuario descargar el informe para su revisión o inclusión en auditorías.



Resumen del Último Escaneo

- http://testphp.vulnweb.com — 11:15:15 - 7/8/2025

➤ Rutas Escaneadas (50)

- Vulnerabilidades (19)

Ruta	Tipo	Score ▼	Mitigación
http://testphp.vulnweb.com/artists.php	FILE_UP	0.9104	Excelente descubrimiento! La vulnerabilidad FILE_UP (File Upload) es comúnmente explotada por ataques de tipo File Inclusion Vulnerability, que permiten a un atacante ejecutar código arbitrario en el servidor web. A continuación, te presento los 3 mejores pasos para mitigar esta vulnerabilidad: **1. Limita la ruta de montaje de archivos** Asegúrate de que la ruta de montaje de archivos sea lo más baja posible y no permita subir archivos a directorios superiores. Por ejemplo, si el archivo se almacena en `/uploads`, asegúrate de que no se pueda subir un archivo a `/uploads/subdirectorio`. **2. Utiliza una validación de tipo MIME** Implementa una validación de tipo MIME (Multipurpose Internet Mail Extensions) para verificar el tipo de archivo antes de procesarlo. Esto te permite rechazar archivos no deseados, como ejecutables o archivos de script. **3. Utiliza un filtrado de archivos y directorios** Asegúrate de que el sistema de subida de archivos no permita subir archivos con nombres que contengan directorios especiales (como `..` o `./`) que podrían permitir la inclusión de archivos arbitrarios en el servidor. Puedes utilizar una biblioteca como Apache's mod_mime_magic para ayudarte a detectar y rechazar archivos sospechosos. Recuerda que es importante mantener actualizadas tus librerías y frameworks, ya que la mayoría de las vulnerabilidades FILE_UP pueden ser explotadas por ataques malintencionados.
http://testphp.vulnweb.com/disclaimer.php	FILE_UP	0.9104	Excelente hallazgo! Aquí te presento los 3 mejores pasos claros y concisos para mitigar la vulnerabilidad de tipo **FILE_UP** en la ruta http://testphp.vulnweb.com/disclaimer.php: 1. **Verificar permisos de archivo y directorio**: Asegúrate de que el archivo y el directorio donde se almacenan los archivos subidos tengan permisos adecuados y no estén configurados para escribir archivos en cualquier lugar del sistema. Esto evitará que un atacante pueda escribir archivos arbitrarios en el sistema. 2. **Limitar la ruta de acceso a archivos**: Configura la ruta de acceso a archivos subidos para que solo permita escribir archivos en una carpeta específica y no permita acceder a carpetas superiores. Esto impedirá que un atacante pueda escribir archivos en cualquier lugar del sistema. 3. **Implementar verificación de contenido**: Implementa una verificación de contenido para asegurarte de que los archivos subidos sean válidos y no contengan código malicioso. Puedes hacer esto mediante la ejecución de scripts o aplicaciones que validen el contenido de los archivos antes de permitir su escritura en el sistema. Al implementar estos pasos, podrás mitigar la vulnerabilidad **FILE_UP** y proteger tu aplicación web contra ataques relacionados con la subida de archivos.
http://testphp.vulnweb.com/signup.php	FILE_UP	0.9104	Excelente descubrimiento! La vulnerabilidad **FILE_UP** se refiere a una vulnerabilidad de tipo file inclusion, que permite al atacante incluir un archivo arbitrario en la aplicación web. A continuación, te ofrezco los 3 mejores pasos para mitigar esta vulnerabilidad: 1. **Valida la ruta del archivo**: En el código que hace la inclusión de archivos, agrega una validación para asegurarse de que la ruta del archivo sea segura y no permita la inclusión de archivos arbitrarios. Por ejemplo, puedes verificar si la ruta comienza con un directorio raíz o un directorio específico. 2. **Utiliza constantes y variables seguras**: En lugar de construir rutas dinámicas utilizando variables de usuario, utilízalas constantes y variables seguras que no puedan ser manipuladas por el atacante. Por ejemplo, en lugar de utilizar la variable `$_GET['file']` para incluir un archivo, puedes definir una constante con la ruta del archivo seguro. 3. **Escanea y actualiza las dependencias**: La vulnerabilidad **FILE_UP** puede ser explotada utilizando archivos de inclusión malintencionados que pueden comprometer la integridad de tu aplicación web. Escanea tus dependencias (bibliotecas, frameworks, etc.) para detectar si hay versiones vulnerables y actualiza a versiones seguras. Recuerda que la prevención es clave en la seguridad informática. Al implementar estas medidas, puedes reducir significativamente el riesgo de explotación de esta vulnerabilidad y proteger tus usuarios y datos.

Figura 20.6: Pestaña en donde se muestran las vulnerabilidades junto con más datos del dominio objetivo

20.2.6 Fase 6: Historial de dominios escaneados

En el menú superior, el usuario puede acceder a la opción **"Historial Dominios"**, donde se muestran todos los dominios previamente escaneados por el usuario autenticado. En esta sección el usuario puede:

- Filtrar por nombre de dominio o por fecha.
- Consultar cada escaneo anterior haciendo clic en el botón correspondiente, lo cual redirige a su respectivo resumen detallado.
- Verificar el progreso histórico y evolución de los escaneos realizados sobre un mismo dominio.

Este historial ofrece una funcionalidad clave para usuarios que realizan pruebas periódicas o seguimiento de la seguridad web de sus sistemas.

Figura 20.6: Página del Historial de Dominios Escaneados

20.2.7 Recomendaciones

- Utiliza esta herramienta solo en dominios de tu propiedad o con autorización expresa para realizar análisis de seguridad.
- Elige umbrales de confianza altos (80% - 90%) si deseas enfocarte solo en posibles vulnerabilidades críticas.
- Repite los escaneos tras cualquier cambio en la infraestructura web, para mantener una visión actualizada de su estado de seguridad.
- Guarda los CSV o PDF exportados para documentar tus auditorías de seguridad.

21. Información sobre los datos

El punto de partida de este proyecto es una idea clave: **la calidad y variedad de los vectores de ataque influyen directamente en la capacidad del modelo para detectar vulnerabilidades reales** en aplicaciones web. Cuanto más diverso y representativo sea el conjunto de datos, mejores serán los resultados que puede ofrecer el sistema en escenarios reales.

Por eso, se ha optado por **fusionar múltiples fuentes** de información. Entre ellas se incluyen:

- Repositorios públicos de *payloads*.
- Bases oficiales de vulnerabilidades como CVE y KEV.
- Feeds de exploits publicados en sitios especializados.
- Tráfico generado en entornos de prueba y etiquetado manualmente.

Algunas de estas fuentes contienen directamente *payloads* ejecutables, otras ofrecen descripciones narrativas sobre cómo funciona una vulnerabilidad, y otras proporcionan estadísticas de uso o criticidad.

Todos estos datos se han unificado mediante un **proceso de ingesta y normalización**, almacenándolos finalmente en la colección tbl_resultados_patterns de la base de datos. A partir de ahí, se construyen los **datasets de entrenamiento** que alimentan el modelo de clasificación de vulnerabilidades.

21.1 Fuentes de datos empleadas

En esta sección se presentarán las distintas fuentes OSINT que hemos empleado para poder obtener los datos que tras un proceso de sanetizacion y limpieza serán los que acabemos usando para poder entrenar a nuestro modelo de Machine Learning

Fuente	Descripción resumida	Valor que aporta
FuzzDB (GitHub)	Colección histórica de listas .txt con payloads para SQLi, XSS, LFI, etc.	Payloads *puros* muy variados (inyecciones de uso real).
PayloadsAllTheThings	Repositorio colaborativo con payloads, cheatsheets y ejemplos de explotación.	Cadenas ejecutables + contexto didáctico.
Nuclei-templates	Plantillas YAML usadas por *ProjectDiscovery Nuclei*; contienen descripciones y a veces el payload dentro de matchers.	Sentencias realistas incrustadas en peticiones HTTP/JSON.
GitHub Advisory DB (GHSA)	Avisos de seguridad en formato JSON (uno por CVE o GHSA).	Descripciones detalladas de vulnerabilidades OSS.
OSS-Fuzz-Vulns	JSONs generados por Google OSS-Fuzz con fallos de memoria y casos edge.	Patrones poco comunes (heap-overflow, OOB-read).
NVD recent / modified	Feed oficial CVE en JSON comprimido.	Descripciones estándar + puntuaciones CVSS.
MITRE allitems.xml	Listado XML completo de CVE con su descripción.	Cobertura histórica completa (desde 1999).
Exploit-DB CSV	Índice CSV de exploits públicos.	Títulos con vectores y a veces el snippet de código.
PacketStorm	Índice web de exploits y advisories.	Titulares que indican rápidamente la técnica.
OWASP Labs (PortSwigger)	Catálogo HTML de laboratorios prácticos (XSS, CSRF…).	Frases concisas describiendo cada vulnerabilidad.
CISA KEV	Catálogo JSON de CVE explotados activamente.	Campo exploited=true; prioriza patrones de riesgo.
Wordlists extra *(SecLists, XSS-Radar, PayloadsXSS, SQLI-Payloads, LFIhub)*	Repos de payload específicos para una sola familia.	Amplían la variedad léxica en clases minoritarias.
Logs etiquetados propios (tbl_prueba)	Peticiones reales generadas en Juice-Shop / DVWA con el resultado conocido.	Ejemplos completos “como los ve el servidor”.
Síntesis controlada	Combina un path real + payload → request sintético.	Multiplica las muestras y balancea clases.

Cuadro 21.1: Fuentes de datos empleadas

21.2 Formato y estructura esperada

Nota: todos los ejemplos se almacenan finalmente con el **mismo esquema Mongo** (source, category, pattern, synthetic, exploited, kind)

Fuente principal	Formato original	Ficheros	Campos que extraemos	Extracción de datos
FuzzDB **SecLists-SQLi** **XSS-Radar** **PayloadsXSS** **SQLI-Payloads** **LFIhub**	Texto plano	*.txt, *.payload	Línea completa del archivo	• Se lee **línea a línea**. • classify() detecta la categoría por keywords. • Se guarda tal cual en pattern, kind="payload".
Nuclei-templates (ProjectDiscovery)	YAML	*.yaml, *.yml	id, info.description, cadenas de matchers	• yaml.safe_load() → dict en memoria. • Concatenamos id, info.description y los matcher-strings. • Cada línea extraída pasa por classify(); casi siempre acaba como kind="payload" (si contiene un exploit) o kind="description".

GitHub Advisory DB (GHSA)
OSS-Fuzz-Vulns
CISA KEV
	JSON	*.json	summary, details, cveID, notes	• json.loads() → accedemos a:
				  ○ summary, details (GHSA)
				  ○ details (OSS-Fuzz)
				  ○ cveID, notes (KEV).
				• Normalmente son frases narrativas ⇒ kind="description".
				• Si en notes vemos ' ' OR 1=1, <script>, etc. cambiamos a payload.

NVD recent / modified	JSON comprimido	nvdcve-*.json.gz	CVE_Items[].cve.description…	• Descomprimir **.json.gz** y recorrer CVE_Items. • Tomar la descripción primaria. • Registrar source="NVD", kind="description" (sin payload).
MITRE CVE *allitems*	XML (comprimido)	allitems.xml.gz	<entry><description>	• Descomprimir (si procede), parsear con ElementTree. • Extraer <description> → mismos pasos que NVD.
Exploit-DB índice	CSV	files_exploits.csv	Columna description	• csv.DictReader() → columna description. • Suele contener el nombre + tipo + CVE → kind="description".
PacketStorm Security **OWASP Labs (PortSwigger)**	HTML scrapeado	Listados web	Texto de los enlaces o títulos (<a>, <h3>)	• BeautifulSoup para visitor las etiquetas <a> o <h3>. • Texto resultante ≈ título corto → kind="description".
Logs reales etiquetados (tbl_prueba)	Texto HTTP completo	N/A (en Mongo)	Campo request (línea de petición) + label	• *Ya vienen con request completo* → no pasan por tbl_resultados_patterns; se usan tal cual en el dataset de entrenamiento.

Cuadro 21.2: Formato y estructura de los datos

22. Cosas a tener en cuenta

A lo largo del desarrollo del sistema se han tomado una serie de decisiones técnicas y operativas que, si bien no forman parte explícita de la estructura de diseño documentada en los apartados anteriores, son relevantes para comprender el funcionamiento real del sistema y su lógica interna. A continuación, se detallan algunos aspectos clave que deben tenerse en cuenta:

- **Generación de mitigaciones mediante LLM (Ollama)**: el sistema no genera de forma automática una mitigación por cada vulnerabilidad detectada. En su lugar, se lanza una consulta al modelo Ollama desde la celda de entrenamiento del modelo de Machine Learning que tenemos en el cuaderno de Jupyter, en la cual se le proporciona como *prompt* una pregunta contextualizada con los datos previamente almacenados en la base de datos. Esto implica que la calidad de la mitigación generada dependerá directamente de cómo se formule dicha consulta y de la información recuperada para componerla. El sistema no aprende mitigaciones, sino que **las solicita activamente mediante técnicas de NLP (procesamiento del lenguaje natural)**.
- **Los patrones sintéticos no se almacenan siempre en la base de datos**: durante la construcción del dataset de entrenamiento, se generan patrones sintéticos para enriquecer el aprendizaje del modelo. Sin embargo, en muchos casos estos no son almacenados de forma explícita en la colección tbl_prueba, ya que su objetivo es mejorar la precisión del clasificador y no necesariamente dejar constancia de su origen.
- **El crawling de rutas no tiene un límite fijo, pero sí práctico**: aunque MongoDB no impone restricciones sobre el tamaño de los arrays rutas[], en la implementación actual se establece un máximo de 50 subrutas por dominio (rutas[0] a rutas[49]) para simplificar la exportación y visualización de los resultados.
- **El número máximo de vulnerabilidades que se almacenan por dominio es 7**: esto se debe a que, para mantener un formato homogéneo en la colección tbl_resultados_escaneo, se estructuraron los campos como vulnerabilidades[0] a vulnerabilidades[6], siendo este límite más que suficiente en el entorno de pruebas definido.
- **La puntuación (score) que se utiliza como umbral de decisión no es fija para todos los dominios**: el campo umbral_confianza se define por registro, lo que permite ajustar dinámicamente la sensibilidad del modelo en función del tipo de dominio o los patrones detectados en el codigo en donde decidimos lanzar un escaneo del dominio dentro de nuestro cuaderno de Jupyter.

- **El modelo de Machine Learning no opera en tiempo real en esta versión**: el entrenamiento se realiza offline en Jupyter Notebook y el archivo generado (.joblib) se carga manualmente en el backend para ser utilizado en predicciones. Esto se diseñó así para separar claramente la fase de aprendizaje de la de inferencia. (Esto se modificará con la implementación de la página web)
- **El sistema podrá permitir la introducción de cualquier dominio**: aunque el sistema no incorpora en su código una validación explícita que impida escanear dominios no autorizados, en el desarrollo del proyecto se ha respetado estrictamente la legislación vigente limitando el uso del sistema a entornos de prueba locales como DVWA, Juice Shop, etc. Para un despliegue futuro, se podría implementar una validación automática mediante listas blancas o comprobación de IPs privadas.

23. Ampliaciones y posibles mejoras

1. Una de las posibles ampliaciones de este sistema consiste en integrar capacidades de aprendizaje continuo, de forma que el modelo pueda ir actualizándose de manera automática con nuevos ejemplos o patrones que se vayan encontrando. Esto permitiría que la herramienta no solo mantenga su precisión a lo largo del tiempo, sino que también pueda adaptarse a nuevas formas de ataque que vayan surgiendo.
2. Mejorar la interacción entre el usuario y el sistema añadiendo una interfaz más completa en la parte del dashboard o sección de resultados, donde se puedan consultar estadísticas generales, evolución de seguridad por dominio, o vulnerabilidades frecuentes. Esto facilitaría mucho la visualización y comprensión del estado de seguridad de una aplicación concreta.
3. Añadir al código fuente una “White list” para que solo deje escanear aquellos dominios que estén permitidos ser escaneados de tal manera que así evitaríamos cualquier compromiso legal.
4. Implementar una interfaz de personalización de vulnerabilidades, donde el usuario pueda configurar qué tipos de vulnerabilidades quiere priorizar o qué nivel de score considera crítico, adaptando así la herramienta al entorno concreto en el que se vaya a utilizar.
5. Almacenar los patrones sintéticos en una tabla/colección dentro de la base de datos, añadiendo etiquetas y metadatos que permitan su reutilización.

24. Conclusiones

Este proyecto ha demostrado que es posible construir una herramienta práctica, efectiva y automatizada para detectar vulnerabilidades en aplicaciones web, combinando técnicas clásicas de pentesting con enfoques modernos basados en inteligencia artificial. A través de un sistema modular y escalable, se ha logrado analizar dominios completos y ofrecer recomendaciones concretas para poder llegar a mitigar esos fallos de seguridad encontrados.

Además de lo técnico, el proyecto ha servido como una experiencia de aprendizaje integral en la que se han trabajado múltiples tecnologías punteras, desde FastAPI y Docker hasta el uso de modelos de lenguaje como LLaMA. Todo ello bajo un enfoque ético, legal y didáctico, respetando los límites de uso responsable que debe tener una herramienta de este tipo.

Aunque el sistema aún tiene margen de mejora, los resultados obtenidos y las pruebas realizadas pueden resultar convincentes para su viabilidad como solución real a la hora de reforzar la seguridad en entornos web. Sin duda, este trabajo deja abierta la puerta a futuras ampliaciones que podrían convertirlo en una herramienta de referencia dentro del ámbito de la ciberseguridad ofensiva.

25. Anexos

25.1 Acrónimos y abreviaturas

A continuación, mostraremos una tabla con los diferentes abreviaturas y acrónimos que se han usado durante todo el proyecto

Acrónimo	Significado
LLM	Large Language Model (Modelo de Lenguaje Extenso)
ML	Machine Learning
OWASP	Open Web Application Security Project
ZAP	Zed Attack Proxy (herramienta de escaneo de OWASP)
URL	Uniform Resource Locator
XSS	Cross-Site Scripting
IA	Inteligencia Artificial
API	Application Programming Interface
UI	User Interface (Interfaz de Usuario)
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
WSL	Windows Subsystem for Linux
NFR	Non-Functional Requirements (Requisitos No Funcionales)
OSINT	Open Source Intelligence
ML	Machine Learning (Aprendizaje Automático)
TF	Term Frequency
IDF	Inverse Document Frequency
CVE	Common Vulnerabilities and Exposures
DB	Database (Base de Datos)
SQLI	SQL Injection
LFI	Local File Inclusion
JS	JavaScript
SQL	Structured Query Language
DVWA	Damn Vulnerable Web Application
KEV	Known Exploited Vulnerabilities
CSV	Comma-Separated Values
PDF	Portable Document Format

Cuadro 25.1: Acrónimos y abreviaturas

26. Bibliografía

26.1 Repositorios y fuentes de payloads:

https://github.com/swisskyrepo/PayloadsAllTheThings.git

https://github.com/payloadbox/sql-injection-payload-list.git

https://github.com/payloadbox/xss-payload-list.git

https://github.com/projectdiscovery/nuclei-templates.git

https://github.com/fuzzdb-project/fuzzdb.git

https://github.com/danielmiessler/SecLists.git

https://github.com/0xInfection/XSS-Radar.git

https://github.com/danielmiessler/LFIHub.git

https://github.com/tennc/awesome-sql-injection.git

26.2 Datasets y tráfico real para entrenar:

https://gitlab.fing.edu.uy/gsi/web-application-attacks-datasets/-/raw/master/csic_2010/anomalousTrafficTest.txt

https://gitlab.fing.edu.uy/gsi/web-application-attacks-datasets/-/raw/master/csic_2010/normalTrafficTest.txt

https://gitlab.fing.edu.uy/gsi/web-application-attacks-datasets/-/raw/master/ecml_pkdd/ecml_csv.csv

https://archive.org/download/CICIDS2018/01-03-2018.zip

26.3 Bases de datos de vulnerabilidades (CVE):

https://nvd.nist.gov/feeds/json/cve/1.1/nvdcve-1.1-recent.json.gz

https://nvd.nist.gov/feeds/json/cve/1.1/nvdcve-1.1-modified.json.gz

https://cve.mitre.org/data/downloads/allitems.xml.gz

https://www.cve.org/CVERecord?id={cve_id

https://www.cisa.gov/sites/default/files/feeds/

26.4 Sitios usados como prueba de vulnerabilidades:

https://testphp.vulnweb.com/

https://portswigger.net/web-security/all-labs

https://packetstormsecurity.com/files/

https://demo.owasp-juice.shop

http://zero.webappsecurity.com

26.5 Otras fuentes de exploits y bases de datos:

https://gitlab.com/exploit-database/exploitdb/-/raw/main/files_exploits.csv

https://github.com/github/advisory-database.git

https://github.com/google/oss-fuzz-vulns.git

https://www.securityfocus.com/archive/1