Cybersecurity4All: software orientado al aprendizaje de la seguridad informática

Ainhoa Blasco Muñoz

Resumen — Cybersecurity4All es un proyecto creado con el objetivo de educar sobre ciberseguridad tanto a usuarios comunes como a programadores de C y C++. El proyecto se divide en dos enfoques principales: la seguridad informática para usuarios no necesariamente técnicos, y la seguridad en el código para desarrolladores de C/C++. El primer enfoque se centra en proporcionar una educación accesible y práctica con el propósito de mejorar la capacidad de defensa del usuario frente a amenazas corrientes como podrán verse más adelante. El segundo enfoque introduce la inteligencia artificial, la cual analiza el código y detecta amenazas potenciales dentro de éste, basándose en vulnerabilidades comunes. La unión de ambos componentes yace en una aplicación web que busca mejorar la comprensión de la ciberseguridad tanto en el ámbito diario como en el técnico.

Palabras clave — Django, Vue.js, Ciberseguridad, Análisis de código estático, Inteligencia Artificial, Educación, C, C++, Python, Phishing, Ingeniería social

Abstract — Cybersecurity4All is a project created with the goal of educating both ordinary users and C and C++ programmers about cybersecurity. The project is divided into two main approaches: network security for non-technical users, and code security for C/C++ developers. The first approach focuses on providing accessible and practical education with the aim of improving the user's ability to defend against common threats, which we will discuss below. The second approach introduces artificial intelligence, which analyzes code and detects potential threats within it based on common vulnerabilities. The union of the two components lies in a web application that seeks to improve the understanding of cybersecurity in both the daily and technical domains.

Index Terms — Django, Vue.js, Cybersecurity, Static code analysis, Artificial Intelligence, Education, C, C++, Python, Phishing, Social engineering

1 Introducción

Actualmente las cifras de ciberataques han incrementado llegando a posicionar a España como el quinto país más amenazado por ciberataques en el 2024 [1], muchos de estos ataques roban los datos del usuario mediante troyanos [1, 2, 3] capaces de interactuar con otras aplicaciones del dispositivo como pueden ser bancos. Es bien sabido que este tipo de *malware* está diseñado para operar sin ser detectado, por lo que puede llegar a ser complicado saber si nuestro dispositivo está afectado y más considerando que muchos usuarios no son conocedores del funcionamiento de éste. La motivación de Cybersecurity4All recae en enseñar al usuario sobre la prevención de este tipo de amenazas así como la actuación frente a otros tipos como puede ser el *phishing* [4] o el *scareware* [5].

Dejando a un lado a los usuarios comunes, Cybersecurity4All quiere ir más allá intentando prevenir amenazas que puede ocasionar un miembro de la comunidad de IT como pueden ser desarrolladores. Existen infinidad de vulnerabilidades relacionadas con el código que son explotadas cada día, por ello se implementa una rama dentro de la aplicación para la detección de vulnerabilidades mediante inteligencia artificial. Esta parte pretende analizar el código escrito en C o C++ estáticamente y advertir al desarrollador sobre si su programa es potencialmente peligroso, así como enseñarle la vulnerabilidad relacionada.

2 Objetivos del proyecto

El objetivo general del proyecto es lograr educar a la población, en el ámbito tanto tecnológico como común, sobre la importancia de la ciberseguridad. Para lograr este objetivo se plantean los siguientes objetivos específicos.

- Estudio de las amenazas sobre la ingeniería social que más afectan a la población a fin de que el aplicativo actúe en el contexto más afectado en los siguientes tres días.
- Investigación sobre el desarrollo web para encontrar la estrategia de implementación que más se ajuste al proyecto, con una duración de tres días.
- Desarrollo de la parte didáctica de la aplicación sobre las vulnerabilidades de la ingeniería social en las siguientes cuatro semanas.
- Investigación sobre las vulnerabilidades comunes en el ámbito de la programación, con el fin de obtener resultados reales de seguridad. Se estiman tres días de trabajo.
- Estudio de las tecnologías utilizadas en la detección de vulnerabilidades en el código durante los siguientes tres días a la primera investigación.
- Aprendizaje sobre detección de anomalías con inteligencia artificial con el fin de crear un modelo

en base a un *dataset* en el que se trabajará durante nueve semanas.
- Desarrollo de la aplicación para la integración del modelo en las siguientes dos semanas.
- Durante los tres días siguientes a la finalización del programa, se encuentra la fase de pruebas con usuarios reales de diferentes categorías para asegurar unos buenos resultados de la aplicación en la etapa de prelanzamiento.

3 METODOLOGÍA Y PLANIFICACIÓN

El proyecto se ha estructurado utilizando la metodología Kanban [6] debido a que permite ver visualmente la carga de trabajo en cada fase, además de ser flexible y escalable lo cual es idóneo para este proyecto ya que permite focalizar en trabajo en diversas tareas pudiendo priorizarlas. Esta metodología se ha implementado utilizando la herramienta YouTrack [7] la cual permite mucha flexibilidad pudiéndose integrar con otros recursos. Mediante este aplicativo se ha dividido el proyecto en tres grandes fases. Estas fases se alinean con los objetivos específicos del trabajo que se han detallado en el anterior punto, es posible visualizar el cronograma del proyecto creado como Diagrama de Gantt [8] en el anexo A1.

4 ESTADO DEL ARTE

A lo largo de este apartado se detalla el contexto del trabajo, donde se incluyen las discusiones entre las opciones planteadas para llevar a cabo el proyecto.

4.1. Amenazas en la Ingeniería Social

En el contexto de la ciberseguridad, la ingeniería social es una de las mayores amenazas por su capacidad de explotar las vulnerabilidades propias del comportamiento humano. Se realizó un estudio de amenazas que afectan directamente a la sociedad y que causan un gran impacto en la seguridad de ésta. Entre las amenazas podemos encontrar de muchos tipos, en este apartado mencionamos unos cuantos que afectan a la sociedad hoy en día por lo que suponen un riesgo en la seguridad:

- *Phishing* [22]. Mensajes digitales o de voz que intentan manipular a los destinatarios haciéndose pasar por una entidad o persona creíble para que compartan información confidencial, descarguen software malicioso, etc.
- *Baiting* [22]. Mediante un señuelo se atrae a las víctimas para que faciliten información confidencial o descarguen código malicioso, tentándolas con una oferta o un objeto de valor.
- *Quid Pro Quo* [22]. Ofrecen un bien o servicio deseable como ganar concursos falsos o recompensas de fidelidad a cambio de la información confidencial de la víctima.
- *Watering Hole Attack* [22]. Inyectan código malicioso en una página web legítima frecuentada por sus objetivos.
- *Deepfake Engineering* [23]. Falsificaciones generadas por IA (imágenes, audio o video falsos) que son convincentes.
- *Typosquatting* [24]. Técnica para engañar a los usuarios y hacerles creer que un dominio falsificado es legítimo. El dominio falso se puede utilizar para engañar a los usuarios y hacerlos visitar un sitio malicioso o confiar en un correo electrónico que han recibido.
- *Social Media Reconnaissance* [25]. Proceso de recopilar información sobre un objetivo a partir de fuentes disponibles públicamente.
- *Credential Stuffing* [26]. Inyección automática de credenciales en los formularios de inicio de sesión de sitios web, con el fin de obtener acceso fraudulento a las cuentas de usuario.
- *Evil Twin* [27]. Se establecen puntos de acceso maliciosos en lugares donde los consumidores esperan encontrar wifi público.

4.2. Tecnologías para la creación de aplicaciones web

Lograr conseguir una aplicación que se adapte a las ideas del proyecto puede ser una tarea difícil sin estudio previo, es por eso por lo que se comparan las diferentes tecnologías en el siguiente apartado. La investigación concluye en la comparación de dos *backends* y dos *frontends*.

***Backend*: Node.js vs Django**

Node.js [28] es un *framework* de JavaScript que se usa para manejar aplicaciones en tiempo real y dinámicas, como sistemas de mensajería o aplicaciones con actualizaciones instantáneas. Está adecuado a las aplicaciones modernas como APIs REST y aplicaciones con alta concurrencia, además está pensada para webs de todo tipo al ser flexible y escalable.

Django [29] es un *framework* de Python enfocado en el desarrollo rápido, teniendo seguridad integrada y capacidad de manejar aplicaciones web completas desde el *backend*. Además, es ideal para plataformas educativas y portales de contenido, así como sistemas de gestión por ser muy adecuado para las operaciones básicas CRUD[2]. Al ser más monolítico, concentra toda su estructura en un único proyecto, lo que lo orienta a proyectos pequeños o medianos, aunque su escalabilidad queda demostrada en aplicaciones como Instagram [30] y Pinterest [31].

***Frontend*: Vue.js vs React**

Vue.js [32] es un *framework* de JavaScript usado por su simplicidad y facilidad de integración. Es ideal para construir interfaces de usuario dinámicas e interactivas, además es adecuado para aplicaciones SPAs[3] y *dashboards* interactivos. Empresas como Alibaba [33] y Xiaomi [34] lo usan, pero es recomendable para aplicaciones pequeñas o medianas porque requiere menos configuración inicial y su curva de aprendizaje es suave.

React [35] es una librería de JavaScript usada por su flexibilidad además de ser usado en aplicaciones interactivas y escalables teniendo a empresas como Facebook [36] y Airbnb [37] utilizándola, siendo recomendable el uso para aplicaciones más grandes por su flexibilidad.

4.3. Vulnerabilidades en Código Existentes

Las vulnerabilidades en código representan un gran problema en el ámbito de la seguridad del software y por este motivo es necesario encontrar una solución para la detección de estas fallas en la seguridad. Tras un previo estudio, se identificó la importancia del análisis de las amenazas de los lenguajes de bajo nivel como C y C++ dado que estos lenguajes siguen siendo utilizados en software crítico y sistemas operativos [9], donde su rendimiento y control sobre los recursos del sistema son esenciales. No obstante, estas características los hacen propensos a fallos de seguridad sobre todo en aplicativos que puedan contener código heredado ya que puede que estos no hayan seguido los estándares modernos de seguridad incrementando así el riesgo a incluir vulnerabilidades.

Estos fallos son típicos debido al manejo manual de memoria y la falta de verificaciones automáticas de C y C++. Por ejemplo, al tener control explícito de memoria, las vulnerabilidades relacionadas con el *buffer overflow* pueden permitir a un atacante escribir datos fuera de los límites de un búfer, lo que puede causar corrupción de memoria o la ejecución de código arbitrario [10]. Asimismo, el *use after free* permite acceder a memoria ya liberada lo que podría explotarse para cambiar el flujo de ejecución del programa [11]. De este modo, sería posible ejecutar código malicioso ya que si el objeto se libera, pero se sigue teniendo una referencia a él cualquier acceso tras su liberación podría resultar en explotable.

4.4. Tecnologías Actuales para Análisis de Código Estático

En el contexto del análisis del código estático, se han desarrollado tecnologías que resuelven el problema de la detección de vulnerabilidades. En el estudio sobre las herramientas existentes para el análisis de código estático en lenguajes de bajo nivel encontramos Codesonar [12].

Codesonar es una herramienta comercializada para encontrar y corregir errores y vulnerabilidades de seguridad en código fuente y binario. Realiza análisis interprocedimentales de todo el programa con interpretación abstracta en C, C++ , C# , Java , así como en bibliotecas y ejecutables binarios x86 y ARM de esta manera su análisis no se limita a funciones individuales sino que abarca toda la interacción con el programa sin tener que ejecutarlo simulando cómo se manejarían las estructuras.

Otra herramienta que podemos encontrar es PVS-Studio [13]. Este software, también comercializado, es un analizador de código estático que protege la calidad, la seguridad y la protección del código siendo compatible con C, C++, C++11, C++/CLI, C++/CX, C# y Java. Utiliza técnicas avanzadas de análisis estático, como análisis de flujo de datos, análisis interprocedimental e intermodular, y concordancia de patrones.

4.5. Modelos de Inteligencia Artificial para el Análisis de Código Estático

La inteligencia artificial ofrece una visión innovadora al problema de la detección de las vulnerabilidades mejorando por ellas mismas y adaptándose a la complejidad del código.

Para el análisis se pueden utilizar diferentes tipos de modelos los cuales se deben adaptar a las características del código fuente:

- Basados en Procesamiento de Lenguaje Natural (NLP): Como el código se puede tratar como un texto, podría ser útil para capturar la sintaxis y la semántica del código. Por ejemplo, se pueden emplear métodos como TF-IDF[1], que extraen características relevantes al medir la importancia de los tokens en el contexto del conjunto de datos. También se podrían utilizar modelos preentrenados como los Transformers [14] para la detección o la clasificación de las vulnerabilidades al aprender representaciones profundas del código.
- Basados en Redes Neuronales: Son capaces de identificar patrones específicos o analizar funciones permitiendo detectar vulnerabilidades recurrentes o errores en el flujo del programa.
- Basados en Métodos de Ensamble: Métodos como Random Forest clasifican vulnerabilidades usando características numéricas del código, como los vectores generados mediante TF-IDF o embeddings de modelos preentrenados. Son robustos frente a ruido y efectivos para identificar patrones complejos. Random Forest, en particular, aprovecha la combinación de múltiples árboles de decisión para lograr un rendimiento sólido, haciéndolo ideal para tareas de clasificación de vulnerabilidades en código estático.

Tras la creación del modelo se emplean las métricas de precisión, *recall* y *F1-score,* así como la matriz de confusión para evaluar el rendimiento.

La precisión mide la proporción de predicciones positivas que son correctas, es decir, de todas las predicciones que detectó como positivas cuantas veces acertó. Se calcula como el cociente entre los verdaderos positivos y el total de predicciones positivas.

En cuanto al *recall* o tasa de verdaderos positivos, mide la proporción de casos positivos correctamente, es decir, de todos los casos que realmente eran positivos cuantos acertó. Evalúa qué tan bien el modelo captura las instancias positivas reales.

El *F1-score* es la media armónica entre precisión y *recall*, proporcionando un equilibrio entre ambas métricas. Es especialmente útil cuando las clases están desbalanceadas, ya que combina la exactitud y la exhaustividad del modelo en un único valor.

Por último, la matriz de confusión es una tabla n x n, donde n es el número de clases, que muestra la cantidad de predicciones que el modelo ha realizado. Proporciona un resumen de cómo el modelo predice en comparación con los valores reales, ayudando a entender su rendimiento y los errores que comete.

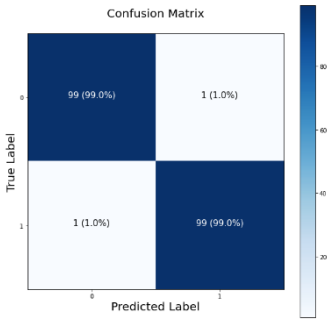

Figura 1. Ejemplo de matriz de confusión binaria (2 clases)

4.6. *Datasets* sobre Vulnerabilidades

Seleccionar un *dataset* adecuado es una parte muy importante para entrenar modelos de inteligencia artificial. A lo largo de la búsqueda se encontraron diversos *datasets* que podrían servir para el proyecto. Uno de ellos es el Big-Vul Dataset [15] el cual está constituido por 348 proyectos de código abierto en C y C++. Este conjunto de datos recopila 3754 vulnerabilidades abarcando 91 tipos de ellas diferentes y estructura el archivo clasificando cada proyecto en columnas por el ID de la vulnerabilidad, la puntuación de la seguridad y el impacto entre otras. Además, también incluyen los ID de los *commits* antes del arreglo y tras este.

Al seguir con la búsqueda, encontramos el MegaVul Dataset [16], este conjunto contiene 17380 vulnerabilidades de código en C y C++ recopiladas de 992 repositorios de código abierto, abarcando 169 tipos diferentes de vulnerabilidades. El *dataset* es más extenso que el Big-Vul Dataset y hoy en día se sigue actualizando. Los datos están estructurados también en campos como el nombre del repositorio de donde proviene, la función, el ID de la vulnerabilidad, etc.

Se encontró otro *dataset* proveniente del estudio *Automated Vulnerability Detection in Source Code Using Deep Representation Learning* [17]. En el trabajo, se centran en vulnerabilidades de los lenguajes C y C++. Recoge 104439 funciones extraídas de repositorios de código abierto y ya viene clasificado por vulnerabilidades como CWE-119 [18], CWE-120 [19], CWE-469 [20] y CWE-476 [21] donde estas son las columnas que contienen el valor '*True*' o '*False*' dependiendo si dicha función contiene dicha vulnerabilidad.

5 Desarrollo

Durante el siguiente apartado se muestran las decisiones tomadas justificadas tras el estudio realizado en la sección anterior.

5.1. Diseño y Elección de Herramientas y Métodos

Amenazas en la ingeniería social

Día a día la población sufre ataques cada vez más innovadores y de difícil detección, por lo que es muy importante mantenerse actualizado de las nuevas amenazas. Tras una búsqueda se concluyó que hay ciertos tipos de ataques que siguen afectando a los usuarios. Estos tipos están mencionados en el apartado 4.5. y se tratan de una manera educativa, concienciando sobre su importancia con ejemplos prácticos y laboratorios de detección. Cybersecurity4All busca que la tasa de ataques efectivos que dependen del factor humano baje exponencialmente con su interfaz *user-friendly* la cual, se pretende, sea usada por todos aquellos que no tengan conocimientos previos sobre informática y/o seguridad y aun así se logre un aprendizaje igualmente efectivo. Por esto se requiere centrar la atención en los ataques que más afectan a la población como los *phishings*, los cuales representaron el 36% de todas las violaciones de datos en EE. UU. en 2023 [40].

También, otro ataque frecuente debido a la reutilización de contraseñas por parte de los usuarios es el *Credential Stuffing*. Este ataque supuso que el 24,3 % de los intentos de inicio de sesión en Customer Identity Cloud de OKTA[4] cumplieran con los criterios de robo de credenciales [41] ya que los atacantes aprovechan bases de datos de credenciales filtradas.

Y, por último, un ataque que cada vez se convierte en más popular es el *Typosquatting* el cual se ha demostrado que aumenta con eventos como el Black Friday [42].

Estos suponen el top de ataques actuales por ello, Cybersecurity4All pretende enfocarse en la educación sobre la identificación y actuación del usuario.

Tecnología web

Existen muchos tipos de tecnologías válidas para el propósito de la aplicación, pero una vez finalizado el estudio se concluyó que Django sería la indicada para el *backend* por los requerimientos de Cybersecurity4All, tales como la necesidad de una integración fácil con el modelo de inteligencia artificial. Ya que el modelo está desarrollado en Python usar un *framework* de este lenguaje hace que la integración sea más rápida.

Además, Django ofrece un rápido desarrollo y escalabilidad al ser un *framework* con herramientas integradas como autenticación, gestión de usuarios, bases de datos, formularios y seguridad.

Por otro lado, dado que el proyecto se basa en la seguridad informática, Django es una buena elección porque tiene medidas ya integradas como la protección contra SQL Injection, XSS, y CSRF. Lo cual sigue la filosofía de Cybersecurity4All y revisa estos fallos en primera instancia.

Para el *frontend* se decidió usar Vue.js ya que la integración con Django es fluida por su diseño que facilita su inclusión en las plantillas de Django sin necesidad de una separación completa entre *frontend* y *backend*, permitiendo la evolución progresiva de la aplicación según las necesidades del proyecto.

Además, Vue.js tiene una curva de aprendizaje suave, lo que permitió comenzar rápidamente sin una configuración inicial compleja sin dejar de lado la interactividad ya que permite crear componentes dinámicos que mejoran la experiencia del usuario en la aplicación, como formularios interactivos, *dashboards* de seguridad informática o visualización de la información de las diferentes amenazas.

Por último, otro aspecto que se tuvo en cuenta fue la comunicación con el *backend*. Gracias a bibliotecas como axios, Vue.js puede utilizar fácilmente las APIs REST servidas por Django, simplificando la transferencia de datos entre el *frontend* y el *backend*.

Dataset Utilizado para el Entrenamiento del Modelo

Tras el previo estudio de los *datasets* existentes que pueden ser utilizados; se decidió que el conjunto de datos que se usaría sería el del estudio *Automated Vulnerability Detection in Source Code Using Deep Representation Learning* [17].

La elección se basó en los criterios como el foco en vulnerabilidades concretas. Debido a que el proyecto está en su primera versión, se prefirió centrarse en unas vulnerabilidades específicas en vez de abarcar más variedad, de esta manera se puede mejorar el análisis y hacerlo más preciso para estas explotaciones. En este caso se utilizaron CWE-119, CWE-120, CWE-469 y CWE-476 y se eliminó la clasificación de "Otras vulnerabilidades" del *dataset* ya que se pretende focalizarse en la detección más precisa y añadirlo podría ocasionar peores resultados.

Otro criterio que condicionó la elección fue que el conjunto de datos ya estaba categorizado y preprocesado en tres archivos: entrenamiento, test y validación, lo cual daba más tiempo al desarrollo del modelo.

Además, brinda la posibilidad de comparación dado que el *dataset* ha sido utilizado en el estudio [17], facilita la comparación de resultados y mejora la validación del modelo.

Vulnerabilidades que Analizará el Modelo

Como ya se menciona en el anterior punto, el modelo se enfoca en las siguientes vulnerabilidades:

- CWE-119: Fallo en la gestión de límites de memoria.
- CWE-120: Desbordamiento de búfer.
- CWE-469: Uso incorrecto de punteros en memoria.
- CWE-476: Acceso a punteros nulos.

Estas explotaciones, como ya comentamos en el apartado 4.1., son de vital importancia su identificación ya que pueden perjudicar gravemente la seguridad de sistemas críticos donde pueden comprometer la integridad, disponibilidad y confidencialidad de la información. Esto es debido a que C y C++ tienen acceso a la administración de la memoria manual. La selección de estas vulnerabilidades no solo responde a su relevancia en la seguridad, sino que también al hecho de que representan una buena base para un análisis inicial robusto, permitiendo así un enfoque más preciso en la identificación de las mencionadas amenazas.

Modelos de Inteligencia Artificial

Con tal de detectar vulnerabilidades en el código, se propone utilizar una combinación de Procesamiento de Lenguaje Natural (NLP) con métodos tradicionales. Este enfoque fue elegido debido a la problemática de las funciones multietiqueta y a los recursos computacionales disponibles. Para comenzar, el TF-IDF se describe como un extractor de características. Es una técnica muy utilizada en el procesamiento de lenguaje natural que consiste en la transformación de textos a representaciones numéricas, a partir de la importancia de cada término en un texto.

El algoritmo TF-IDF captura los patrones sintácticos básicos y las relaciones de frecuencia en el código. Su incorporación como extractor de características ayuda a gestionar la carga computacional y a codificar el código fuente como representaciones vectoriales eficientes que capturan la información estructural del texto. Esto no sólo facilita el entrenamiento del modelo de aprendizaje automático, sino que también ayuda a detectar patrones dentro de grandes cantidades de datos con mayor precisión.

Una vez extraídas las características, se procede al desarrollo del modelo para la clasificación multietiqueta. La decisión final dejó como seleccionado el modelo clasificador basado en Random Forest [38] con la estrategia de MultiOutputClassifier de la librería scikit-learn [39]. Este enfoque permite entrenar un modelo para cada vulnerabilidad de manera eficiente y es adecuado para conjuntos de datos donde las relaciones entre características no son demasiado complejas. Se decidió este modelo porque permite establecer una línea base rápida y robusta con un bajo coste computacional.

5.2. Creación del *frontend* de la aplicación

La implementación de la aplicación se estructuró con un diseño modular que garantiza escalabilidad, claridad y facilidad de mantenimiento. A lo largo de este apartado se describirá su estructura.

Estructura de Componentes

Se desarrollaron cinco componentes utilizando Vue.js. El componente principal (App.vue) el cual actúa como contenedor general, integrando y gestionando la lógica de navegación entre los distintos componentes de manera dinámica para cargarlos según las acciones del usuario. Tras este, el componente de inicio (start.vue) el cual introduce al usuario en la aplicación. Este contiene dos botones que llevan a la parte del análisis del código o a la parte de educación en ingeniería social.

Figura 2. start.vue inicio de la aplicación

Una vez clicado en "START" de la parte "I'm a developer", se abre la pestaña del componente ai-tab.vue. Éste permite subir un archivo .txt y se hace una petición al *backend* mediante la función handleFileUpload() que gestiona la lógica principal para el análisis de archivo. Primero comprueba si se seleccionó un archivo y muestra un error si no es así. Luego crea un FormData con el archivo seleccionado y lo envía al *backend* a través de una solicitud HTTP POST a la API. Entonces si la respuesta es exitosa, analiza los datos devueltos y emite un evento con los resultados de las vulnerabilidades detectadas que activará el componente code-w-vulnerability.vue en caso de que detecte una vulnerabilidad o code-wt-vulnerability.vue en caso de que no detecte ninguna. Además, en caso de error, muestra mensajes específicos para orientar al usuario.

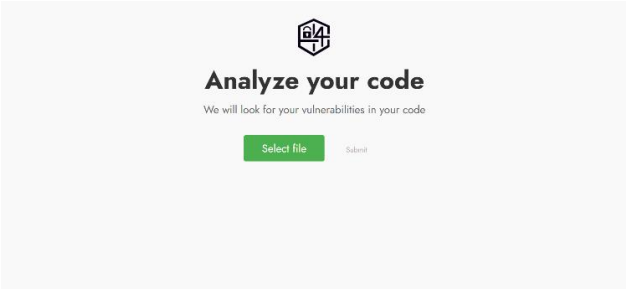

Figura 3. Inicio del componente ai-tab.vue

Una vez clicado "Submit" dependiendo de la respuesta del servidor manejada mediante la función handleFileUpload() activa los componentes mencionados anteriormente. En el caso de que haya alta probabilidad de haber vulnerabilidad, se activa el componente code-w-vulnerability.vue que recibe las vulnerabilidades detectadas en un *array* y las muestra por pantalla junto con explicaciones de éstas y como arreglar el error.

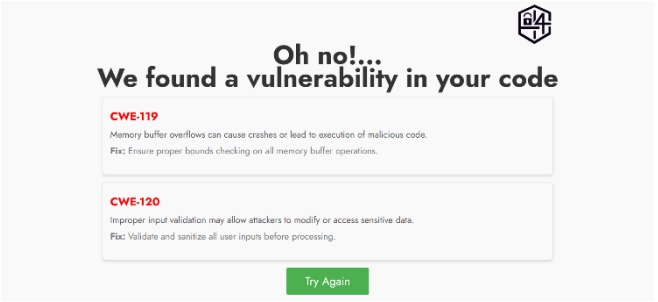

Figura 4. Vulnerabilidades detectadas con el componente code-w-vulnerability.vue

En el caso de que el modelo no detecte ninguna vulnerabilidad, se emitirá el *array* vacío y se activará el componente code-wt-vulnerability.vue que muestra que no se han detectado vulnerabilidades.

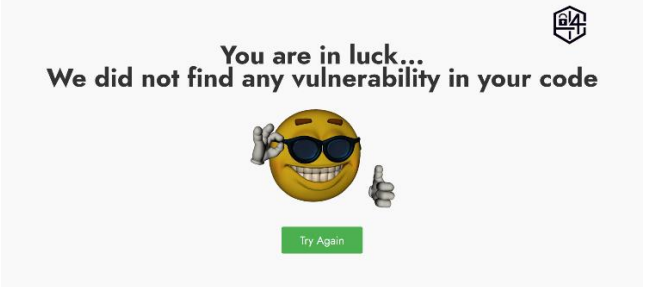

Figura 5. Vulnerabilidades no detectadas con el componente code-wt-vulnerability.vue

Tras clicar en "START" del apartado "I want to be safer", el usuario es redirigido al componente de aprendizaje, learning-tab.vue, cuya función principal es facilitar el aprendizaje interactivo sobre amenazas de ingeniería social mediante ejemplos prácticos y tarjetas educativas. Este componente contiene una barra de progreso diseñada para mostrar el avance del usuario en tiempo real y una barra de aprendizaje, que crece o decrece dependiendo de los aciertos del usuario en los apartados de cuestiones. Además, utiliza un archivo JSON como fuente de datos para generar dinámicamente el contenido de aprendizaje. Este archivo contiene elementos estructurados categorizados como "info", "learningCard" y "options" que permiten definir títulos, descripciones, imágenes y opciones diferentes según lo que se quiera mostrar, haciéndolo muy escalable.

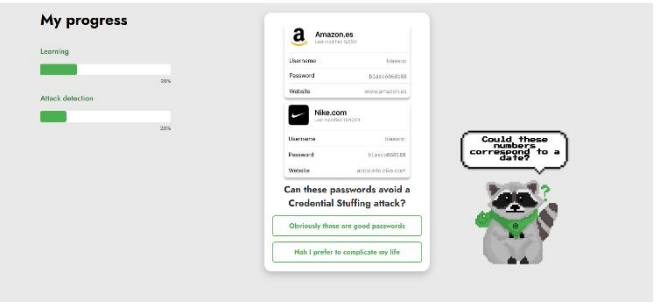

Figura 6. Sección de pregunta en learning-tab.vue

Estilo y Diseño Visual

Para el diseño visual de la aplicación, se utilizaron cuatro hojas de estilo, styles-start.css, styles-learning.css, styles-codes.css y styles-ai.css, que implementan un diseño moderno y accesible. La separación de estilos por componentes facilita el mantenimiento y escalabilidad del diseño.

5.3. Creación del Modelo

Extracción de Datos desde HDF5 y Primera Fase de Preprocesamiento

Para comenzar con el desarrollo del modelo, fue necesario extraer los datos los cuales estaban almacenados en formato HDF5[5]. Para esto fue necesario crear un *script* utilizando la biblioteca h5py [43] para acceder a las estructuras de datos dentro de los archivos.

El *dataset* completo se divide en 3 archivos HDF5: *train*, *test* y *validate*. Dentro de cada uno de ellos hay un diccionario cuya estructura consiste en la *key* como la vulnerabilidad o las funciones y el *value* como array. Los elementos del diccionario son las vulnerabilidades CWE-119, CWE-120, CWE-469, CWE-476 y CWE-other (*keys*) y cada una con un array (*value*) con '*true*' o '*false*' dependiendo de si en esa posición en el elemento functionSource, cuyo *value* es un array de funciones, existe esa vulnerabilidad concreta. Es decir, si en la función que se encuentra en la posición 2 del array functionSource existe la vulnerabilidad CWE-120, en este array habrá un '*true*'.

Con este diccionario se procedió a crear un *csv* para cada *dataset* (*train*, *test* y *validate*) en el cual la primera columna consistía en la función y las siguientes en las cuatro vulnerabilidades (cambiando '*true*´ por 1 y '*false*' por 0), descartando el objeto CWE-other ya que para esta primera versión se pretende focalizarse en la detección más precisa y añadirlo podría ocasionar peores resultados.

Una vez obtenidos los archivos se continuó con la segunda fase de preprocesamiento.

Preprocesamiento y Vectorización con TF-IDF

En esta segunda fase se *tokenizaron* y vectorizaron la primera columna del *dataset train,* ya que el modelo no puede trabajar directamente con datos en formato de texto bruto. Este proceso se realizó mediante un enfoque basado en TF-IDF donde se procesaron los datos en lotes para optimizar

la gestión de grandes volúmenes de código. Cada lote del conjunto de datos fue *tokenizado* utilizando el TfidfVectorizer [44] de la biblioteca scikit-learn, configurado para extraer un máximo de 20,000 características representativas del código fuente. Esta técnica permitió transformar fragmentos de código en vectores numéricos que capturan la relevancia de cada token el contexto del conjunto de datos. Los vectores resultantes se almacenaron en formato *npz* para ser utilizados en fases posteriores.

Generación y Validación de la Representación TF-IDF

Para que el modelo se pueda probar y validar fue necesario procesar estos *datasets* reutilizando el modelo de vectorización TF-IDF previamente entrenado. De esta manera los fragmentos de código fueron transformados bajo el mismo espacio vectorial, manteniendo la consistencia entre los datos de entrenamiento y los de evaluación. Una vez procesados estos datos, se procedió con el entrenamiento del modelo.

Entrenamiento del Modelo Random Forest

Una vez obtenidas las matrices TF-IDF generadas y las etiquetas extraídas se entrenó el modelo Random Forest configurado con 100 árboles y optimizado para su ejecución en paralelo (n_jobs=-1) de esta manera utiliza todos los núcleos disponibles para paralelizar el entrenamiento de los árboles. Ya habiendo entrenado el modelo se continuó con la validación de este.

Evaluación del Modelo

El modelo entrenado se evaluó utilizando el *dataset validate*, aplicando métricas como precisión, recall y F1-score, así como la matriz de confusión mencionadas en el punto 4.3. y calculadas con funciones de scikit-learn, para validar su desempeño. De esta manera se creó el modelo y estaba listo para su utilización.

5.4. Creación del *backend*

Tras el despliegue del entorno en Django se tuvieron que hacer cambios en diferentes partes del proyecto configurando *middlewares*[6] que añaden capas adicionales de seguridad, como la protección contra *clickjacking* en la que el usuario es engañado para interactuar con un sitio web cargado dentro de un *iframe*[7] malicioso. Y la habilitación de HTTP Strict Transport Security (HSTS) que es una política que obliga a los navegadores a realizar todas las solicitudes a través de HTTPS.

Sin embargo, para permitir que el entorno funcione correctamente con el *frontend* ejecutándose en un origen diferente, se incluyó soporte para CORS[8] en el archivo de configuración del proyecto, añadiendo 'corsheaders.middleware.CorsMiddleware' en la lista de *middlewares*. Esto permite añadir los encabezados HTTP necesarios para gestionar solicitudes entre dominios o puertos diferentes como es el caso, habilitando la comunicación entre el *backend* Django (http://localhost:8000) y el *frontend* Vue.js (http://localhost:8080) ya que la política de mismo origen de los navegadores bloquea cualquier intento de comunicación entre orígenes distintos.

Además, también se necesitó definir una lista de orígenes confiables en CORS_ALLOWED_ORIGINS, como http://localhost:8080, que corresponde al dominio donde se ejecuta el *frontend* para asegurar que solo las solicitudes con este origen sean aceptadas por el *backend*.

Una vez configurado el proyecto para permitir la comunicación, se establece el *endpoint* ''api/file-processing/' de manera que cualquier petición con un archivo adjunto será procesada en la función 'file_processing' en view.py. Tras recibir el archivo, la función lo procesa aplicándole el tokenizador cargado y realizando predicciones con el modelo también cargado tras sus creaciones. Ya con las predicciones hechas, la función crea un JSON y lo manda como respuesta al *frontend* que la procesa como se menciona en el apartado 5.2.

6 Resultados

Una vez creada la aplicación y el modelo se detallan los resultados siguientes tras las ejecuciones de éstos. Los resultados se muestran según las pruebas hechas para el modelo y según el *feedback* de los usuarios reales que probaron la aplicación.

Tras la Ejecución del Modelo

El modelo de detección de vulnerabilidades basado en Random Forest presentó los siguientes resultados al ser evaluado en el conjunto de validación:

1. **Primera ejecución con batch_size = 30000, max_features = 20000, n_estimators=100, random_state=42 y n_jobs=-1**
   El modelo alcanzó un 96% de precisión general por lo que clasificó correctamente la mayoría de las muestras del conjunto. Sin embargo, si analizamos los resultados, se muestran F1-scores especialmente bajos en todas las clases, indicando problemas para capturar correctamente las instancias reales como se aprecia en la métrica recall. La clase CWE-469 tiene el peor desempeño debido a su bajo soporte (252 ejemplos), mientras que CWE-476 es la mejor clase relativa con un recall de 0.21, pero sigue siendo insuficiente. Este bajo rendimiento sugiere desbalance en las clases lo que afecta al modelo.

Results in validate:

	precision	recall	f1-score	support
CWE119	0.71	0.09	0.16	2419
CWE120	0.72	0.09	0.17	4750
CWE469	0.42	0.03	0.06	252
CWE476	0.83	0.21	0.33	1208

Figura 6. Resultados de la primera ejecución para las métricas *precisión, recall* y *F1-Score*

Si se utiliza la métrica de la matriz de confusión para evaluar el modelo se obtiene que para la clase CWE-469 hay muy pocos verdaderos positivos (8) frente a 244 falsos negativos, lo que refleja un modelo casi incapaz de detectar esta clase. Para la CWE-120 no cambia el resultado, aunque detecta 449 verdaderos positivos, no logra identificar la mayoría (4301 falsos negativos) y tiene 175 falsos

positivos, lo que explica el bajo *F1-score*. Al igual que para la clase CWE-119 que solo detecta 214 verdaderos positivos frente a 2205 falsos negativos y por último la clase CWE-476 que tiene un mejor desempeño relativo con 249 verdaderos positivos y solo 50 falsos positivos, pero aún tiene 959 falsos negativos, limitando el *recall*.

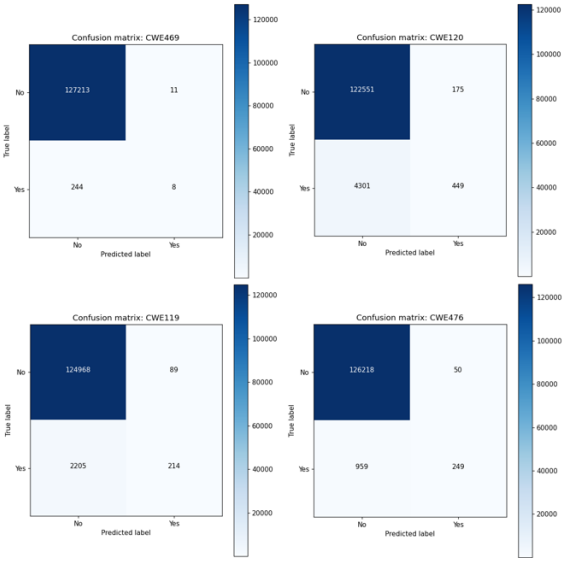

Figura 7. Resultados de la primera ejecución utilizando la matriz de confusión

2. **Segunda ejecución con batch_size = 5000, max_features = 10000, n_estimators=200, random_state=42 y n_jobs=-1**
Se hizo un balance para los datasets eliminando un 70% de ceros y los resultados muestran un desempeño significativamente mejorado, con F1-scores altos en CWE-119 y CWE-120, reflejando un buen balance entre precisión y recall en ambas clases. Sin embargo, las clases minoritarias (CWE469 y CWE476) aún presentan problemas, con F1-scores bajos debido a su bajo recall y soporte limitado. A pesar de estos retos, el promedio ponderado del modelo muestra un desempeño sólido, aunque se puede observar el desequilibrio en el rendimiento entre clases. Para seguir mejorando, sería útil aumentar datos en clases minoritarias.

Results in validate:

	precision	recall	f1-score	support
CWE119	0.84	0.82	0.83	2419
CWE120	0.84	0.82	0.83	2511
CWE469	0.67	0.04	0.08	99
CWE476	0.69	0.09	0.17	96

Figura 8. Resultados de la segunda ejecución para las métricas *precisión, recall* y *F1-Score*

Con la matriz de confusión para evaluar el modelo, se obtiene que para la clase CWE-469 hay muy pocos verdaderos positivos (4) frente a 95 falsos negativos, lo que refleja un modelo que aún tiene dificultades para detectar esta clase, a pesar de los ajustes realizados. Para la clase CWE-120, el modelo detecta 2070 verdaderos positivos, pero no logra identificar 441 instancias reales (falsos negativos) y comete 380 falsos positivos, lo que contribuye al desempeño mejorado, pero no óptimo. De manera similar, para la clase CWE-119, el modelo logra identificar 1995 verdaderos positivos, aunque todavía tiene 424 falsos negativos y 379 falsos positivos. Finalmente, la clase CWE-476 muestra un desempeño moderado con 9 verdaderos positivos, pero aún tiene 87 falsos negativos y solo 4 falsos positivos, lo que mejora ligeramente su precisión, pero limita el *recall* debido a las muchas instancias perdidas.

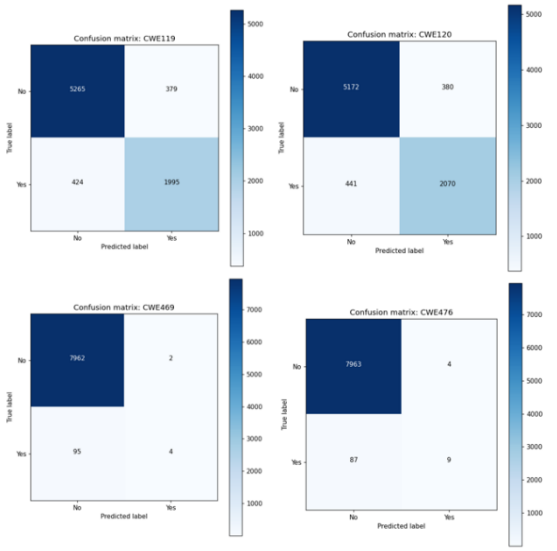

Figura 9. Resultados de la segunda ejecución utilizando la matriz de confusión

En comparación con el estudio *Automated Vulnerability Detection in Source Code Using Deep Representation Learning* [17], ambos trabajos utilizan Random Forest para la detección de vulnerabilidades en código C y C++, pero difieren en su enfoque y resultados. El primero combina características aprendidas mediante CNN con Random Forest, logrando métricas robustas. Por otro lado, Cybersecurity4All utiliza Random Forest basado en TF-IDF, obteniendo una precisión global del 96%, pero con problemas de *recall* en clases minoritarias debido al desbalance del *dataset*. Mientras que el enfoque del estudio es más sofisticado y generalista, Cybersecurity4All se centra en un modelo más sencillo y específico para vulnerabilidades concretas, con margen de mejora en la representación y balance de datos.

Tras las Pruebas con Usuarios Finales

Se dividieron los usuarios de testeo en dos grupos, el primer grupo, compuesto por 10 personas de edad avanzada (media de 50 años), representó a usuarios habituales de tecnología. El segundo grupo incluyó 7 programadores, en su mayoría en etapas iniciales de su carrera. Los resultados permitirán evaluar la usabilidad para ambos perfiles. En general, ambos grupos creen que la aplicación es una herramienta interesante y útil, especialmente en un

contexto donde las ciberamenazas están cada vez más presentes. Lo que más les llamó la atención fue la interfaz llamativa y poder ver con ejemplos prácticos qué tan vulnerable eran ante ataques como el *phishing*, *credential stuffing* y *typosquatting*. Esto les hizo recapacitar sobre ciertos comportamientos de su día a día.

Por otro lado, las cartas explicativas le parecieron un detalle atractivo. Pensaron que era una forma amigable y accesible de aprender sobre conceptos de manera no técnica. Aunque, los usuarios de edad avanzada creen que se debería mejorar la legibilidad con un tamaño mayor ya que la aplicación está orientada a usuarios de edades medias más que jóvenes.

Como puntos de mejora, los usuarios creyeron que sería conveniente ajustar el contenido según el nivel de conocimiento del usuario ya que algunos están acostumbrados a campañas de ciberseguridad. Además, creen que sería divertido que hubiera una comunidad en la que se puedan crear retos para detectar las vulnerabilidades en conjunto y añaden que la mascota les parece muy llamativa y que estaría bien personalizarla en un futuro.

7 Conclusiones

A lo largo de este apartado se detalla la resolución del trabajo tras realizar el desarrollo completo de las diferentes partes.

7.1. Conclusiones Generales

Como conclusión general, el modelo debería mejorarse todavía más para alcanzar el objetivo general de esta parte del proyecto. No ha alcanzado la suficiente precisión para ser usado en producción, pero es capaz de detectar vulnerabilidades en el código para las dos primeras clases.

En cuanto a la parte de ingeniería social, ha alcanzado el objetivo general según los usuarios. Aunque tiene muchos puntos de mejora y teniendo en cuenta que está en su primera versión, se podría decir que ha sido un éxito en su fase de lanzamiento.

Sobre las decisiones de las tecnologías, la elección de Vue.js fue todo un acierto que permitió desarrollar la aplicación rápidamente cumpliendo con los tiempos y sobre Django al principio fue complicado entender la estructura, pero la implementación fue bastante rápida.

Por último, se cree que el modelo podría haber dado mejores resultados con una combinación de algoritmos o con redes neuronales pero debido a la capacidad de cómputo que precisan y a los que hubo accesibilidad, los resultados son aceptables.

7.2. Tras el Estudio de las Métricas del Modelo

Primera Ejecución del Modelo

Con 100 árboles y optimizado para su ejecución en paralelo (n_jobs=-1) los resultados evidencian que el modelo tiene dificultades para detectar fragmentos vulnerables, lo que podría implicar que los hiperparámetros no son los correctos o que hay un desbalance de clases por lo que se procede a editar la cantidad de elementos de la clase 0 o No Vulnerables para entrenar el modelo. Si esto no fuera suficiente se cambiarían los hiperparámetros hasta llegar a la optimización del modelo para la detección de la clase 1 o Vulnerable, objetivo principal de esta parte del proyecto.

Segunda Ejecución del Modelo

Con 200 árboles los resultados muestran una mejora en la detección, aunque el modelo sigue teniendo dificultades para identificar fragmentos vulnerables en las clases minoritarias. Esto indica que persisten retos relacionados con el desbalance de clases. Por ello, se plantea continuar ajustando la cantidad de elementos en las clases menos representadas para equilibrar el conjunto de datos y seguir mejorando con los hiperparámetros.

7.3. Tras las Pruebas con Usuarios Finales

En general, el usuario piensa que la combinación de teoría con práctica hace que la aplicación sea completa y efectiva, aunque les gustaría que se reflejase un resumen final con lo que se ha aprendido y con la nota final.

Sobre el *feedback* de los usuarios sobre el nivel de la aplicación, esta opinión se da porque los grupos de prueba fueron usuarios que trabajan con ordenadores en empresas multinacionales y usuarios que no utilizan tecnologías en sus trabajos. Por lo que estos usuarios que trabajan con ordenadores son los que recomiendan esta mejora, al ya haber visto ciertos ejemplos en campañas de sus empresas. Por ejemplo, los usuarios con menos avance en la plataforma podrían recibir explicaciones más básicas, mientras que los usuarios más avanzados podrían aprender sobre amenazas más complicadas.

Agradecimientos

Agradezco a la Universidad Autónoma de Barcelona y a todo su profesorado por brindarme la oportunidad de aprender en un entorno cómodo y que me permitiese compaginar mis estudios con mi situación laboral desde el inicio del grado. En especial a mi tutora del TFG, Ana Oropesa, por confiar en mi desde la asignatura de TDIW y por brindarme apoyo en estos meses de trabajo. Por último, también agradezco a mis compañeros de clase y amigos por el tiempo invertido en ayudarme a entender conceptos del proyecto que no había visto antes y a mi hermana Candela por diseñar la mascota de la aplicación.

Bibliografía

[1] S21sec, "*Threat Landscape Report 2024: Primer Semestre*," S21sec, 2024.

[2] KNF, "*Hookbot: A new mobile malware*," disponible en: https://cebrf.knf.gov.pl/komunikaty/artykuly-csirt-knf/362-ostrzezenia/858-hookbot-a-new-mobile-malware [Accedido: 20-sept-2024]

[3] Waverley Software, "*Top software vulnerabilities*," disponible en: https://waverleysoftware.com/blog/top-software-vulnerabilities/, [Accedido: 20-sept-2024]

[4] Wikipedia, "*Phishing*," disponible en: https://es.wikipedia.org/wiki/Phishing, 2024. [Accedido: 20-sept-2024]

- *E-mail de contacto: 1601314@uab.cat*
- *Mención realizada: Tecnologías de la Información*
- *Trabajo tutorizado por: Ana Oropesa (Departamento de Ingeniería de la Información y de las Comunicaciones)*
- *Curso 2024/25*

[5] Wikipedia, "*Scareware*," disponible en: https://es.wikipedia.org/wiki/Scareware, 2024. [Accedido: 20-sept-2024]

[6] Wikipedia, "*Kanban (desarrollo)*," disponible en: https://es.wikipedia.org/wiki/Kanban_(desarrollo), 2024. [Accedido: 20-sept-2024]

[7] JetBrains, "*YouTrack*," disponible en: https://www.jetbrains.com/youtrack/, [Accedido: 20-sept-2024]

[8] Asana, "*Fundamentos de los diagramas de Gantt*," disponible en: https://asana.com/es/resources/gantt-chart-basics, [Accedido: 20-sept-2024]

[9] S. Peta, "*C Programming Language - Still Ruling the World*," *International Journal of Science and Research*, vol. 11, no. 4, 2022, DOI: 10.21275/SR22403142926

[10] OWASP Foundation, "*Buffer Overflow Attack*," OWASP, 2023, disponible en: https://owasp.org/www-community/attacks/Buffer_overflow_attack, [Accedido: 4-nov-2024]

[11] MITRE, "*CWE-416: Use After Free*," disponible en: https://cwe.mitre.org/data/definitions/416.html, [Accedido: 4-nov-2024]

[12] Doymus, "*CodeSonar*," disponible en: https://doymus.com/products/codesonar/, [Accedido: 4-nov-2024]

[13] PVS-Studio, "*PVS-Studio*," disponible en: https://pvs-studio.com/en/pvs-studio/, [Accedido: 4-nov-2024]

[14] Hugging Face, "*Transformers Documentation*," disponible en: https://huggingface.co/docs/transformers/index, [Accedido: 4-nov-2024]

[15] Y. Zhou, S. Liu, Y. Liu, and S. Wang, "*A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries*," in *Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, Melbourne, Australia, 2023, pp. 1-2. doi: 10.1109/ICSE-Companion58688.2023.00001

[16] C. Ni, L. Shen, X. Yang, Y. Zhu, and S. Wang, "*MegaVul: A C/C++ Vulnerability Dataset with Comprehensive Code Representation*," *arXiv preprint arXiv:2406.12415*, 2024. Disponible en: https://arxiv.org/abs/2406.12415

[17] R. L. Russell, L. Kim, L. H. Hamilton, T. Lazovich, J. A. Harer, O. Ozdemir, P. M. Ellingwood, and M. W. McConley, "*Automated Vulnerability Detection in Source Code Using Deep Representation Learning*," *arXiv preprint*, arXiv:1807.04320v2 [cs.LG], 2018. DOI: 10.48550/arXiv.1807.04320

[18] CWE, "*CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (4.15)*," disponible en: https://cwe.mitre.org/data/definitions/119.html, [Accedido: 4-nov-2024]

[19] CWE, "*CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (4.15)*," disponible en: https://cwe.mitre.org/data/definitions/120.html, [Accedido: 4-nov-2024]

[20] CWE, "*CWE-476: Null Pointer Dereference*," disponible en: https://cwe.mitre.org/data/definitions/476.html, [Accedido: 4-nov-2024]

[21] CWE, "*CWE-469: Use of Pointer Subtraction to Determine Size*," disponible en: https://cwe.mitre.org/data/definitions/469.html, [Accedido: 4-nov-2024]

[22] IBM, "*¿Qué es la Ingeniería Social?*", disponible en: https://www.ibm.com/es-es/topics/social-engineering, [Accedido: 5-nov-2024]

[23] CrowdStrike, "*¿Qué es un ataque de deepfake?*", disponible en: https://www.crowdstrike.com/es-es/cybersecurity-101/social-engineering/deepfake-attack/, [Accedido: 5-nov-2024]

[24] CrowdStrike, "*Inteligencia de adversarios: Tiposquatting*", disponible en: https://www.crowdstrike.com/tech-hub/counter-adversary-operations/monitor-for-malicious-domain-impersonations/, [Accedido: 5-nov-2024]

[25] TechLatest, "*Reconocimiento en redes sociales*", disponible en: https://medium.com/@techlatest.net/social-media-reconnaissance-906d104c9b90, [Accedido: 5-nov-2024]

[26] OWASP, "*Credential stuffing*", disponible en: https://owasp.org/www-community/attacks/Credential_stuffing, [Accedido: 5-nov-2024]

[27] Ciberseguridad.com, "*Ataque Evil Twin: qué es y cómo evitarlo*", disponible en: https://ciberseguridad.com/amenzas/ataque-evil-twin/, [Accedido: 5-nov-2024]

[28] Node.js, "*Node.js*," disponible en: https://nodejs.org/en/, [Accedido: 5-nov-2024]

[29] Django Software Foundation, "*Overview: Get started with Django*," disponible en: https://www.djangoproject.com/start/overview/, [Accedido: 5-nov-2024]

[30] Instagram, "*Instagram – Social Media Platform*," disponible en: https://www.instagram.com/, [Accedido: 5-nov-2024]

[31] Pinterest, *"Pinterest – Plataforma de descubrimiento visual,"* disponible en: https://es.pinterest.com/, [Accedido: 5-nov-2024].

[32] Vue.js, "*Vue.js – The Progressive JavaScript Framework*," disponible en: https://vuejs.org/, [Accedido: 5-nov-2024]

[33] Alibaba, *"Alibaba – Global Online Marketplace,"* disponible en: https://www.alibaba.com/, [Accedido: 5-nov-2024]

[34] Xiaomi, *"Mi – Sitio web oficial de Xiaomi,"* disponible en: https://www.mi.com/, [Accedido: 5-nov-2024].

[35] React, "*React – Una biblioteca de JavaScript para construir interfaces de usuario*," disponible en: https://es.react.dev/, [Accedido: 5-nov-2024]

[36] Facebook, *"Facebook – Plataforma de redes sociales,"* disponible en: https://www.facebook.com/, [Accedido: 5-nov-2024]

[37] Airbnb, *"Airbnb – Alquileres vacacionales, alojamientos únicos y experiencias,"* disponible en: https://www.airbnb.es/, [Accedido: 5-nov-2024]

[38] Scikit-learn, "*sklearn.ensemble.RandomForestClassifier*," disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, [Accedido: 5-nov-2024]

[39] Scikit-learn, *"sklearn.multioutput.MultiOutputClassifier,"* disponible en: https://scikit-learn.org/dev/modules/generated/sklearn.multioutput.MultiOutputClassifier.html, [Accedido: 5-nov-2024]

[40] Techopedia, *"Estadísticas sobre Phishing,"* disponible en: https://www.techopedia.com/es/estadisticas-sobre-phishing, [Accedido: 14-nov-2024]

[41] Okta, *"Key Findings from Our 2023 State of Secure Identity Report,"* disponible en: https://www.okta.com/blog/2024/02/key-findings-from-our-2023-state-of-secure-identity-report/, [Accedido: 14-nov-2024]

[42] Cinco Días, *"Typosquatting: la estafa que debes evitar en este Black Friday,"* disponible en: https://cincodias.elpais.com/smartlife/lifestyle/2024-11-12/typosquatting-la-estafa-que-debes-evitar-en-este-black-friday.html, [Accedido: 14-nov-2024]

[43] h5py, *"The h5py Package,"* disponible en: https://www.h5py.org/, [Accedido: 14-nov-2024]

[44] Scikit-learn, *"sklearn.feature_extraction.text.TfidfVectorizer,"* disponible en: https://scikit-learn.org/1.5/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html, [Accedido: 14-nov-2024]

ANEXO

A1. DIAGRAMA DE GANTT

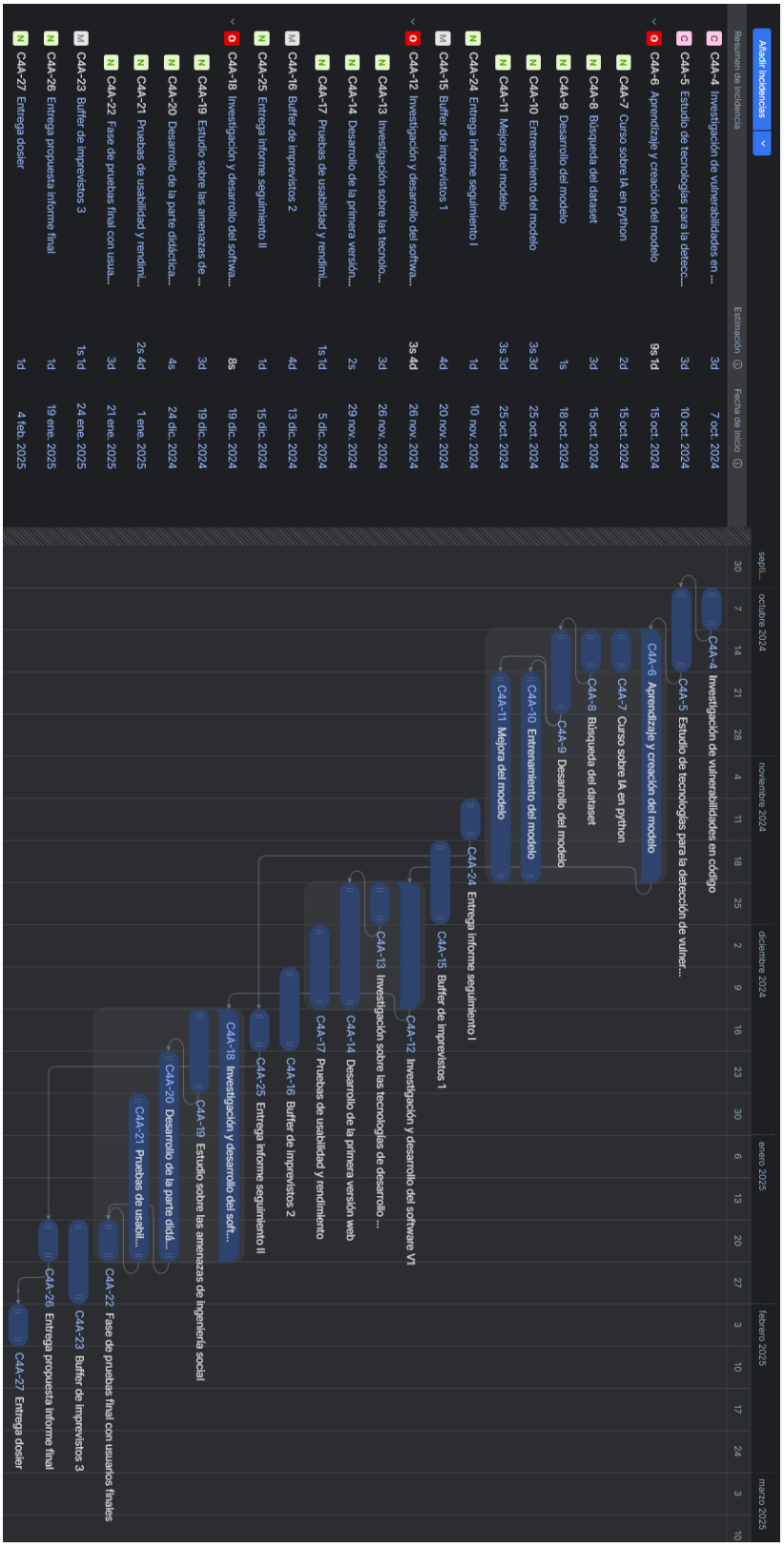

Fig. 1. Cronograma del proyecto

A2. GLOSARIO

1. TF-IDF (Term Frequency-Inverse Document Frequency) técnica de ponderación utilizada en minería de textos y recuperación de información para evaluar la relevancia de una palabra en un texto dentro de un conjunto de datos, considerando su frecuencia local (TF) y su rareza global (IDF).
2. CRUD (Create Read Update Delete) representa las operaciones básicas de persistencia en bases de datos o sistemas de almacenamiento. Estas acciones permiten gestionar datos en cualquier sistema, siendo fundamentales en el desarrollo de aplicaciones y la interacción con bases de datos.
3. SPA (Single Page Application) tipo de aplicación web que carga una única página HTML y actualiza dinámicamente su contenido a medida que el usuario interactúa, sin recargar toda la página.
4. Customer Identity Cloud de Okta (antes Auth0) plataforma de gestión de identidad diseñada para ofrecer autenticación segura, autorización y personalización de accesos para aplicaciones, APIs y dispositivos, enfocada en mejorar la experiencia del usuario final y garantizar la seguridad.
5. HDF5 (Hierarchical Data Format version 5) formato de archivo diseñado para almacenar y organizar grandes volúmenes de datos complejos, estructurados en un formato jerárquico similar a un sistema de archivos, ideal para datos científicos y de alto rendimiento.
6. Middlewares componentes intermedios en aplicaciones o sistemas que gestionan la comunicación y procesamiento de datos entre distintas partes, como servidores, bases de datos o aplicaciones cliente, encargándose de tareas como autenticación, registro de eventos, validación de datos o transformación de información para facilitar la interacción y modularidad del sistema.
7. iFrame (Inline Frame) elemento HTML que permite incrustar otro documento HTML dentro de una página web, utilizado comúnmente para mostrar contenido externo, como videos, mapas o widgets, manteniendo la separación entre el contenido incrustado y la página principal, lo que facilita la integración y modularidad del diseño web.
8. CORS (Cross-Origin Resource Sharing) mecanismo de seguridad en navegadores que permite controlar cómo un servidor define qué recursos pueden ser solicitados por dominios externos, utilizando encabezados HTTP específicos para restringir o autorizar solicitudes, asegurando una interacción segura entre aplicaciones web y APIs alojadas en diferentes orígenes.