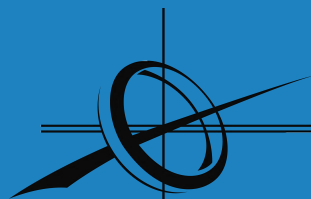




POLITÉCNICA



Universidad
Politécnica
de Madrid

**ETSI SISTEMAS
INFORMÁTICOS**

Diseño e implementación de un sistema de detección y gestión de ataques de Denegación de Servicio en redes aplicando técnicas de inteligencia artificial

Proyecto Fin de Grado

Grado en Tecnologías para la Sociedad de la
Información

Autor:

David Ramos Archilla

Tutor:

Borja Bordel Sánchez

Febrero 2024

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
SISTEMAS INFORMÁTICOS



**Diseño e implementación de un sistema
de detección y gestión de ataques de
Denegación de Servicio en redes
aplicando técnicas de inteligencia
artificial**

Proyecto Fin de Grado

Grado en Tecnologías para la Sociedad de la
Información

Curso académico 2023-2024

Autor:

David Ramos Archilla

Tutor:

Borja Bordel Sánchez

Quisiera aprovechar este espacio para expresar mi más sincero agradecimiento a las personas que han sido fundamentales en la realización de mi trabajo de fin de grado. Este momento no habría sido posible sin el apoyo y el aliento de aquellos que me han acompañado en este viaje académico. En primer lugar, quiero agradecer profundamente mi familia, gracias a ellos he llegado hasta donde estoy hoy. Por último, quiero agradecer el apoyo que me ha dado mi tutor, que me ha guiado y ayudado durante toda la realización del proyecto.

Resumen

Este trabajo consiste en la construcción de un sistema que actúe de firewall para la detección de ataques de denegación de servicio. Para realizar dichas detecciones, se han empleado técnicas de Machine Learning y Deep Learning para entrenar modelos con estas capacidades. Además, se ha implementado una interfaz gráfica a través de la cual se puede observar el tráfico recibido por el sistema en tiempo real, así como las predicciones que hace este sobre si dicho tráfico se trata de un ataque o no.

En primer lugar, para entrenar un modelo de inteligencia artificial se necesitan unos datos de entrenamiento. Se ha escogido un dataset que contiene diferentes características sobre una gran cantidad de flujos de red y una etiqueta que indica si es un ataque, y en caso de serlo, el tipo de ataque que es. Tras escoger el dataset, es necesario realizar una etapa de preprocesamiento, que consiste en preparar los datos con el objetivo de que la fase de entrenamiento sea buena. Una vez están los datos listos, se han entrenado dos modelos. El primero una red neuronal, en concreto, un perceptrón multicapa, y el segundo un algoritmo de gradient boosting basado en árboles de decisión. Tras entrenarlos, evaluarlos y compararlos, se obtenido que el modelo de gradient boosting supera a la red neuronal, logrando una precisión del 99 % sobre el conjunto de datos de validación, por lo que ha sido este modelo el usado al poner en marcha el sistema.

Por otro lado, ha sido necesario implementar una herramienta para capturar el tráfico de red, y que además, sea capaz de obtener las mismas características sobre los flujos de red que se emplean para el entrenamiento de los modelos. Para ello, se ha usado como base una herramienta existente, desarrollada por la misma universidad que creó el dataset utilizado. Sin embargo, esta herramienta no tiene todos los requisitos necesarios para ser incluida en este trabajo, por lo que se ha implementado una versión de ella que sí los tiene.

Por último, se ha implementado la interfaz gráfica que permite visualizar el tráfico y se ha integrado todo ello bajo el mismo sistema. Tras la realización de varios experimentos, se obtienen unos resultados bastante prometedores, ya que se observa como el sistema ha sido capaz de detectar ataques de denegación de servicio.

Abstract

This work consists of building a system that acts as a firewall for detecting denial-of-service attacks. To perform these detections, Machine Learning and Deep Learning techniques have been employed to train models with these capabilities. In addition, a graphical interface has been implemented through which the traffic received by the system can be observed in real-time, as well as the predictions it makes about whether or not the traffic is an attack.

Firstly, to train an artificial intelligence model, training data is required. A dataset has been chosen that contains different features about a large number of network flows and a label indicating whether it is an attack, and if so, what type of attack it is. After choosing the dataset, a preprocessing stage is necessary, which consists of preparing the data so that the training phase is good. Once the data is ready, two models have been trained. The first is a neural network, specifically, a multilayer perceptron, and the second is a gradient boosting algorithm based on decision trees. After training, evaluating, and comparing them, it has been found that the gradient boosting model outperforms the neural network, achieving 99 % accuracy on the validation dataset, so this model has been used when launching the system.

On the other hand, it has been necessary to implement a tool to capture network traffic, which is also capable of obtaining the same features about network flows that are used for model training. To do this, an existing tool developed by the same university that created the dataset used has been used as a base. However, this tool does not meet all the requirements necessary to be included in this work, so a version of it that does meet the requirements has been implemented.

Finally, a graphical interface has been implemented that allows traffic to be visualized and all of this has been integrated under the same system. After conducting several experiments, quite promising results have been obtained, as the system has been able to detect denial-of-service attacks.

Índice

| | |
|--|-----------|
| Agradecimientos | I |
| Resumen | II |
| Abstract | III |
| 1. Introducción | 1 |
| 1.1. Contexto | 1 |
| 1.2. Objetivos | 1 |
| 1.3. Estructura del documento | 2 |
| 2. Estado del arte | 3 |
| 2.1. Ataques de denegación de servicio distribuidos (DDoS) | 3 |
| 2.1.1. Ataques TCP | 4 |
| 2.1.2. Ataques UDP | 5 |
| 2.1.3. Ataques con reflexión MS SQL | 5 |
| 2.1.4. Ataques con reflexión LDAP | 5 |
| 2.2. Inteligencia artificial | 6 |
| 2.2.1. Machine Learning | 6 |
| 2.2.2. Deep Learning | 8 |
| 2.3. Herramientas y librerías | 11 |
| 2.4. Trabajos similares | 12 |
| 3. Desarrollo del proyecto | 13 |
| 3.1. Dataset | 13 |
| 3.1.1. Procesado del dataset | 13 |
| 3.1.2. Análisis del dataset | 14 |
| 3.2. Entrenamiento de modelos | 16 |
| 3.2.1. Cross-validation | 17 |
| 3.2.2. Red neuronal | 17 |
| 3.2.3. Modelo de gradient boosting | 23 |
| 3.3. Herramienta de captura de tráfico | 27 |
| 3.3.1. PyFlowmeter | 27 |
| 3.4. Arquitectura del sistema | 28 |
| 3.4.1. Servidor | 29 |
| 3.4.2. Modelo de inteligencia artificial | 30 |
| 3.4.3. Interfaz web | 30 |
| 4. Resultados | 33 |
| 4.1. Sistema en funcionamiento | 33 |
| 4.1.1. Tráfico en tiempo real | 34 |
| 4.1.2. Ataque LDAP | 35 |
| 4.1.3. Ataque UDP null | 36 |
| 4.1.4. Ataque TCP SYN flood | 37 |
| 5. Conclusiones y trabajos futuros | 38 |
| 5.1. Conclusiones | 38 |
| 5.2. Impacto social y medioambiental | 38 |
| 5.3. Lineas futuras | 39 |

| | |
|--|-----------|
| <i>ÍNDICE</i> | V |
| Bibliografía | 40 |
| Anexos | 43 |
| A. Código fuente del proyecto | 44 |
| A.1. preprocesado_dataset.py | 44 |
| A.2. red_neuronal.py | 45 |
| A.3. busqueda_hiperparametros.py | 46 |
| A.4. pcap_to_csv_pyflowmeter.py | 47 |
| A.5. enviar_trafico_real_a_servidor.py | 47 |
| A.6. simular_trafico_pyflowmeter.py | 47 |
| A.7. endpoint_iniciar_sniffer.py | 48 |
| A.8. send_traffic_y_get_data.py | 49 |
| A.9. router.js | 49 |
| A.10.firewall_model.py | 50 |

Índice de tablas

| | |
|--|----|
| 2.1. Principales funciones de activación para redes neuronales | 10 |
| 3.1. Informe clasificación red neuronal durante cross-validation | 20 |
| 3.2. Informe de clasificación red neuronal final | 21 |
| 3.3. Informe de clasificación del modelo de gradient boosting | 25 |

Índice de figuras

| | |
|---|----|
| 2.1. Esquema de un ataque DDoS | 3 |
| 2.2. Esquema de un ataque de inundación SYN | 4 |
| 2.3. Esquema de un ataque de con reflexión LDAP | 5 |
| 2.4. Mapa conceptual de la inteligencia artificial | 6 |
| 2.5. Fases del aprendizaje automático | 7 |
| 2.6. Funcionamiento básico de los algoritmos de gradient boosting | 8 |
| 2.7. Comparación entre Machine Learning y Deep Learning | 9 |
| 2.8. Imagen de una red neuronal | 10 |
| 3.1. Visualización del dataset | 14 |
| 3.2. Distribución de los tipos de ataques | 15 |
| 3.3. Distribución de los tipos de ataques tras el balanceado | 16 |
| 3.4. Esquema cross-validation | 17 |
| 3.5. Gráfica de error durante cross-validation en la red neuronal | 21 |
| 3.6. Gráfica de error durante el entrenamiento de la red neuronal final | 22 |
| 3.7. Matriz de confusión de la red neuronal final | 22 |
| 3.8. Importancia de los hiperparámetros | 24 |
| 3.9. Histórico de la precisión durante la optimización | 24 |
| 3.10. Distribución de los parámetros probados durante la optimización | 24 |
| 3.11. Matriz de confusión del modelo de gradient boosting | 26 |
| 3.12. Importancia de las variables de entrada | 26 |
| 3.13. Diagrama sobre el sistema | 28 |
| 3.14. Vista dashboard | 31 |
| 3.15. Vista traffic-analysis | 32 |
| 4.1. Tabla del análisis del tráfico recibido en tiempo real | 34 |
| 4.2. Gráficas del análisis del tráfico recibido en tiempo real | 34 |
| 4.3. Tabla del análisis de un ataque LDAP | 35 |
| 4.4. Gráficas del análisis de un ataque LDAP | 35 |
| 4.5. Tabla del análisis de un ataque UDP null | 36 |
| 4.6. Gráficas del análisis de un ataque UDP null | 36 |
| 4.7. Tabla del análisis de un ataque TCP SYN flood | 37 |
| 4.8. Gráficas del análisis de un ataque TCP SYN flood | 37 |

Capítulo 1

Introducción

1.1. Contexto

En la era digital actual, la creciente dependencia de los servicios en línea ha generado una mayor vulnerabilidad frente a diversas modalidades de ciberataques. Entre estas amenazas, los ataques de denegación de servicio distribuidos (DDoS) destacan como una estrategia con la finalidad de interrumpir el acceso a servicios en línea, tales como sitios web. Este fenómeno se manifiesta mediante la saturación de tráfico malintencionado proveniente de múltiples fuentes, con el propósito de superar la capacidad de los recursos del servidor objetivo.

Según un informe de Cloudflare [1], los ataques DDoS hipervolumétricos han aumentado en los últimos años. En el primer trimestre de 2023, se observó un aumento significativo en la cantidad de ataques DDoS hipervolumétricos, el mayor de los cuales alcanzó un pico superior a 71 millones de solicitudes por segundo, un 55 % por encima del anterior récord mundial de 46 millones de solicitudes por segundo de Google. Además, según el informe anual de internet de Cisco [2], el número de ataques de denegación de servicio distribuidos ha aumentado gradualmente en los últimos años. En 2018 se registraron 7,9 millones de ataques, en 2019 se incrementó a 9,5 millones y en 2020 se elevó a 10,8 millones.

Por otro lado, los avances en el campo de la inteligencia artificial estos últimos años han sido enormes, y han surgido una gran cantidad de nuevas aplicaciones que se apoyan en estos avances. El núcleo de estos avances ha sido el Machine Learning, una rama de la inteligencia artificial que se centra en el desarrollo de algoritmos y modelos que permiten a las máquinas aprender patrones y tomar decisiones sin una programación explícita, mejorando su rendimiento con la experiencia y datos. Dentro del Machine Learning se encuentra el Deep Learning, una disciplina que utiliza redes neuronales profundas como modelos de aprendizaje para comprender y procesar datos y, en ocasiones, realizar otro tipo de tareas más complejas. Un ejemplo de aplicación de estas novedosas técnicas de inteligencia artificial ha sido en la detección de ataques de denegación de servicio.

Es por ello por lo que se plantea en este trabajo el uso de técnicas de Machine Learning y Deep Learning para tratar de detectar ataques de denegación de servicio distribuidos a partir del tráfico recibido.

1.2. Objetivos

En este trabajo se propone crear un sistema que capture los paquetes de red, los analice usando un modelo de inteligencia artificial para detectar posibles ataques de denegación de servicio y tras ello que se muestren en una interfaz

gráfica que contenga información adicional sobre el tráfico. Teniendo esto en cuenta, se definen los siguientes objetivos:

- Entrenar un modelo de Machine Learning que pueda detectar ataques de denegación de servicio y clasificarlos según el tipo que sean. El modelo debe proporcionar la confianza con la que hace cada predicción.
- Diseñar e implementar un sistema que capture el tráfico de red en tiempo real y sea capaz de extraer variables sobre el tráfico.
- Diseñar e implementar una interfaz gráfica a través de la cual se pueda observar el tráfico de red y las predicciones que realiza el modelo de Machine Learning sobre el mismo.
- Integrar el modelo de Machine Learning, la herramienta de captura de tráfico y la interfaz en un servidor.

1.3. Estructura del documento

A continuación, se realiza una descripción sobre la estructura de este documento:

- El capítulo 1 sirve como breve introducción al trabajo, explicando el contexto en el que se desarrolla y los objetivos establecidos.
- El capítulo 2 ofrece una perspectiva del estado del arte y tecnologías que han sido usadas durante el desarrollo del trabajo.
- El capítulo 3 proporciona una explicación detallada del proceso llevado a cabo en el desarrollo del trabajo.
- El capítulo 4 muestra resultados que se han obtenido tras la realización del trabajo.
- El capítulo 5 aborda las conclusiones obtenidas tras la realización del trabajo, así como posibles líneas futuras de investigación.

Capítulo 2

Estado del arte

2.1. Ataques de denegación de servicio distribuidos (DDoS)

Los ataques distribuidos de denegación de servicio (DDoS) son un tipo de ciberataque cuyo objetivo es interrumpir el funcionamiento normal del servicio o red objetivo consumiendo su ancho de banda disponible, agotando sus recursos computacionales o provocando otros tipos de fallos. Como resultado, los usuarios legítimos no pueden acceder al recurso atacado, lo que provoca una denegación de servicio para ellos.

Normalmente, esto se lleva a cabo empleando botnets, que son equipos informáticos que han sido infectados con malware y que pueden ser controlados remotamente sin que sus propietarios se den cuenta. Estos ordenadores “zombis” envían peticiones continuas y masivas a la víctima, lo que provoca un aumento significativo del tráfico de red y puede causar que el sistema de la víctima se colapse o se vuelva lento y difícil de acceder. El hecho de que los ataques sean distribuidos, es decir, se realizan desde una gran cantidad de máquinas; los hacen muy difíciles de detectar y de mitigar, ya que el tráfico del ataque es apenas indistinguible del tráfico legítimo.

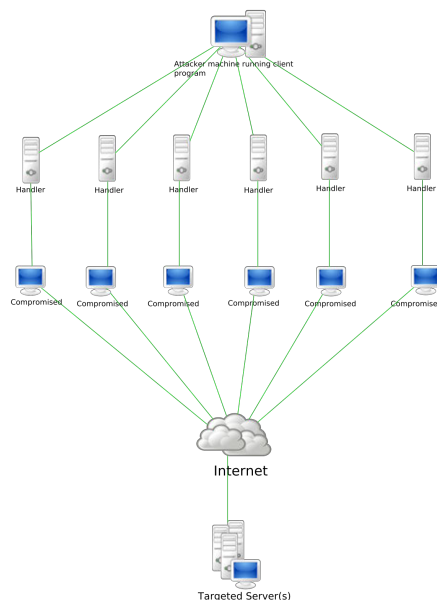


Figura 2.1: Esquema de un ataque DDoS

2.1. ATAQUES DE DENEGACIÓN DE SERVICIO DISTRIBUIDOS (DDOS)⁴

Uno de los mayores ataques DDoS lo sufrió la empresa Github en 2018, que le llevó algo más de 10 minutos detectarlo. El ataque envió a los servidores de GitHub 1,3 terabits por segundo de tráfico, enviado a una velocidad de 126,9 millones de paquetes por segundo. En 2020, Amazon Web Services (AWS) informó que logró mitigar exitosamente un ataque de denegación de servicio distribuido de 2.3 terabits por segundo, marcando un hito como el mayor enfrentamiento en la historia de los ataques DDoS hasta la fecha. Según el informe de AWS [3], este incidente involucró un ataque de reflexión DDoS utilizando CLDAP. Por último, el 1 de junio de 2022, un cliente de Google Cloud Armor fue atacado con un ataque DDoS mediante HTTPS que alcanzó un máximo de 46 millones de peticiones por segundo. Para ponerlo en perspectiva, esto equivale a recibir todas las peticiones que recibe en un día Wikipedia pero en tan solo 10 segundos [4].

Existen una gran cantidad de tipos de ataques DDoS, aunque la mayoría comparten algunas características comunes. Una de las clasificaciones más frecuentes se basa en el nivel de protocolo contra el cual actúan, como, por ejemplo, la capa de transporte, la capa de red o la capa de aplicación. En lo que resta de sección, se va a realizar una descripción de los tipos de ataques con los que se han entrenado los modelos.

2.1.1. Ataques TCP

Dentro de los ataques a través del protocolo TCP (Transmission Control Protocol), la forma de ataque más común se la conoce como inundación SYN (SYN flood). Estos ataques se aprovechan de cómo se establece una nueva conexión en el protocolo TCP. En primer lugar, para iniciar la conexión, el cliente envía un paquete SYN al servidor. Luego, el servidor le responde con un paquete SYN/ACK. Por último, el cliente devuelve un paquete ACK para confirmar la recepción del paquete del servidor.

Teniendo esto en cuenta, para realizar un ataque de denegación de servicios, un atacante envía un alto volumen de paquetes SYN a un servidor. El servidor entonces responderá a cada uno de estos paquetes, dejando puertos abiertos a la espera de recibir los paquetes ACK. La parte clave del ataque es que los clientes nunca enviarán los paquetes ACK, dejando al servidor en espera.

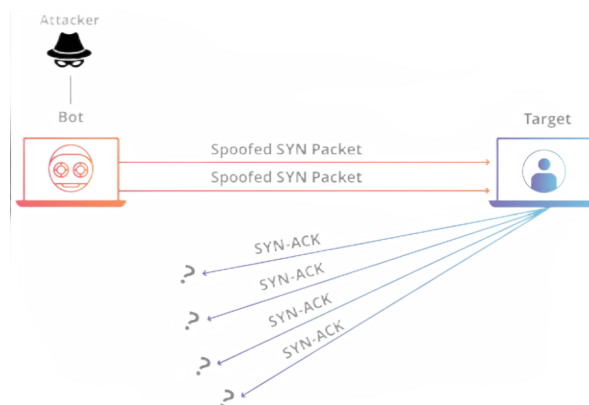


Figura 2.2: Esquema de un ataque de inundación SYN

2.1.2. Ataques UDP

Este tipo de ataque funciona de manera similar a los ataques de inundación SYN sobre TCP, pero en este caso se realiza sobre el protocolo UDP (User Datagram Protocol). En esta situación, cuando un servidor recibe un paquete UDP en un puerto determinado, comprueba primero si hay algún programa en ejecución que esté escuchando solicitudes por ese puerto. En caso de haberlo, el servidor responde con un paquete ICMP (ping) para informar de que no se podrá alcanzar el destino.

Para saturar el servidor, el atacante enviará un número elevado de paquetes UDP, de tal forma que el servidor deba dedicar sus recursos en comprobar y responder cada uno de los paquetes UDP recibidos. Así, los recursos del servidor pueden agotarse muy rápido si el ataque es lo suficientemente volumétrico.

2.1.3. Ataques con reflexión MS SQL

Este tipo de ataques aprovecha vulnerabilidades en servidores de bases de datos Microsoft SQL Server para amplificar y aumentar la magnitud de los ataques. El ataque ocurre usando el protocolo de privilegios de Microsoft SQL Server (MC-SQLR), que se utiliza siempre que un cliente necesita obtener información de MS SQL Server. Al conectarse a un servidor de bases de datos, el cliente recibe como respuesta una lista de instancias de bases de datos. Es aquí donde los atacantes envían solicitudes falsificadas que aparentan provenir de la víctima, con el fin de que el servidor vulnerable responda al objetivo con paquetes mucho más grandes que los originales, generando una amplificación del tráfico dirigida a la víctima.

2.1.4. Ataques con reflexión LDAP

Los ataques con reflexión LDAP (Protocolo Ligero de Acceso a Directorios) funcionan de una manera muy similar a los ataques con reflexión MSSQL. La diferencia es que en este caso las peticiones con la dirección IP falsificada se realizan a servidores LDAP, que del mismo modo que los servidores SQL, amplificarán el tráfico al ser sus respuestas mucho mayores que las peticiones realizadas.

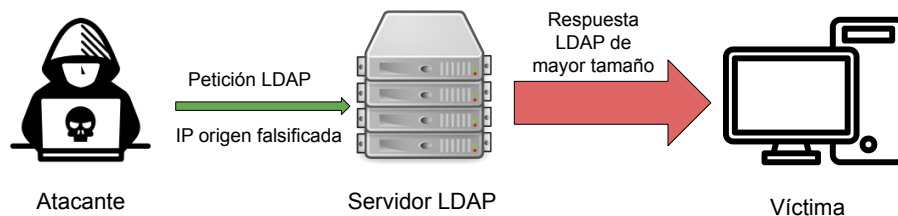


Figura 2.3: Esquema de un ataque de con reflexión LDAP

2.2. Inteligencia artificial

La ciencia de la inteligencia artificial (IA) se dedica al estudio y desarrollo de sistemas y programas capaces de imitar, simular o, en ocasiones, incluso superar la inteligencia humana en tareas específicas. Esto se logra mediante el uso de algoritmos y modelos matemáticos complejos que procesan grandes cantidades de datos y aprenden de ellos, lo que permite a las máquinas realizar tareas que normalmente requerirían de la intervención humana.

Dentro de la amplia variedad de tareas, destacan algunas como la traducción automática, detección de objetos, reconocimiento de voz, conducción autónoma, diagnóstico médico, generación de nuevos datos, y muchas más.

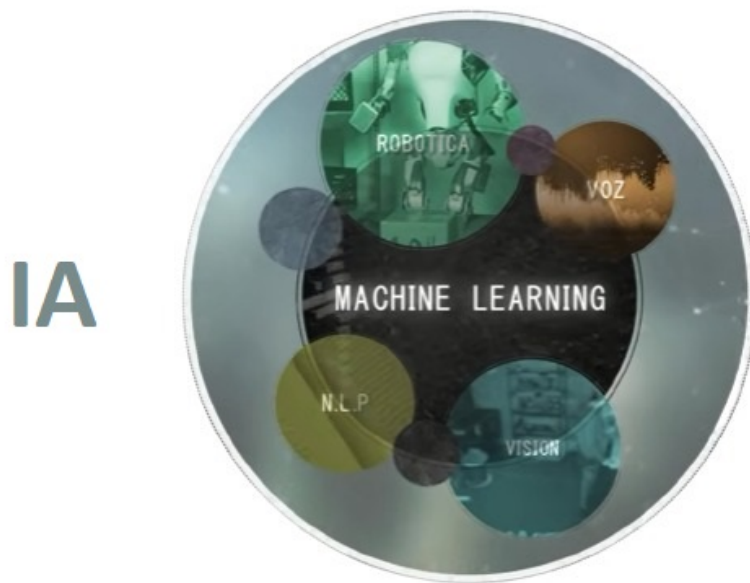


Figura 2.4: Mapa conceptual de la inteligencia artificial

Dentro de la inteligencia artificial, existen diversas técnicas y enfoques, entre los que se incluyen el aprendizaje automático (también conocido como Machine Learning), el procesamiento del lenguaje natural, la lógica borrosa, la visión por ordenador, algoritmos evolutivos y los sistemas expertos. Cada técnica se ajusta en base al problema particular que se necesita resolver y los datos disponibles.

Sin embargo, la inteligencia artificial también plantea algunos problemas éticos. Algunos de ellos son la toma de decisiones automatizadas por agentes inteligentes y la privacidad y la seguridad de los datos. Esto último, se ha agravado sobre todo con el avance de las inteligencias artificiales generativas, tanto en texto como en imágenes, llegando en algunos casos a ocasionar problemas de derechos de imagen.

2.2.1. Machine Learning

El Machine Learning [5] es una rama de la inteligencia artificial cuyo principal objetivo es desarrollar algoritmos y técnicas que permiten a las

máquinas aprender de los datos y realizar predicciones sin haber sido explícitamente programadas para ello. Para lograrlo, se emplean conjuntos de datos que incluyen elementos de entrada junto con las correspondientes salidas deseadas (etiquetas o resultados). El objetivo central es entrenar un modelo con esos datos para que logre generalizar con otros datos no vistos previamente y genere resultados útiles.

El procedimiento de aprendizaje automático normalmente tiene diversas etapas. En primer lugar, se lleva a cabo una fase de preprocesamiento de datos que incluye actividades como la depuración de datos, la elección de características relevantes y la normalización de los datos. Posteriormente, se elige un algoritmo de aprendizaje automático adecuado, que puede ser de tipo supervisado, donde el modelo se entrena con pares de entrada-salida, o no supervisado, donde el modelo busca patrones sin contar con salidas etiquetadas. Una vez se ha elegido el algoritmo, se pasa a la fase de entrenamiento, momento en el cual se le proporcionan los datos al modelo para que este aprenda de ellos y se ajuste a los patrones presentes. Después del entrenamiento, se valida el modelo con el conjunto de datos de validación, un subconjunto del conjunto inicial que se divide antes del entrenamiento. Con estos datos se evalúa el rendimiento del modelo empleando diferentes métricas. En caso de no obtener buenos resultados, se puede ajustar los hiperparámetros del modelo y hacer un nuevo entrenamiento. Por último, cuando el modelo esté listo para ser usado, se puede poner en producción para que realice inferencia con datos nuevos.



Figura 2.5: Fases del aprendizaje automático

2.2.1.1. Gradient boosting

Los algoritmos de gradient boosting son un tipo de modelos de ensamble, es decir, modelos que combinan múltiples modelos más simples para formar uno más robusto y preciso. Normalmente, los modelos simples que se usan son árboles de decisión [6] y este tipo de técnica suele dar mejores resultados que random forest [7].

La idea subyacente está en la construcción iterativa de árboles de decisión débiles, donde cada nuevo árbol se ajusta para corregir los errores de los modelos anteriores. En cada iteración, el algoritmo ajusta un modelo débil a los errores de los modelos anteriores, es decir, a la diferencia entre los valores reales y las predicciones de los modelos anteriores. De esta manera, el nuevo modelo débil se especializa en predecir los errores cometidos por los modelos anteriores, lo que lleva a una mejora gradual en el desempeño del modelo global.

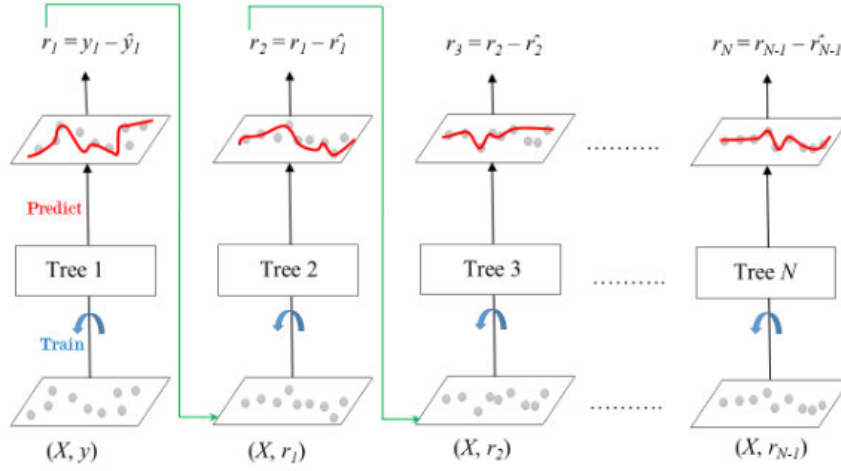


Figura 2.6: Funcionamiento básico de los algoritmos de gradient boosting

2.2.2. Deep Learning

El Deep Learning [8] (aprendizaje profundo) es una subárea del Machine Learning que se enfoca en el desarrollo de algoritmos de aprendizaje basados en redes neuronales artificiales con múltiples capas ocultas. A diferencia de los modelos de Machine Learning tradicionales, los modelos de Deep Learning son capaces de aprender características relevantes directamente de los datos de forma automática. Esos modelos son capaces de enfrentarse a tareas mucho más complejas, como pueden ser, el procesamiento de lenguaje natural, el reconocimiento de voz y de imágenes, la conducción autónoma, y la generación de imágenes y texto.

Estos modelos son capaces de realizar tales tareas gracias a su capacidad de aprender de forma abstracta y de procesar grandes cantidades de datos. Un ejemplo en el que se puede observar esta diferencia puede ser una situación en la que un modelo tenga que reconocer si una imagen contiene un coche o no. Para un algoritmo de Machine Learning tradicional, primero habría que extraer variables sobre la imagen, como, por ejemplo, la cantidad de ruedas que hay, la forma de los objetos o su textura y color. En cambio, un modelo de Deep Learning simplemente necesitaría las imágenes, y sería el propio modelo quien aprendería las características más importantes de los datos.

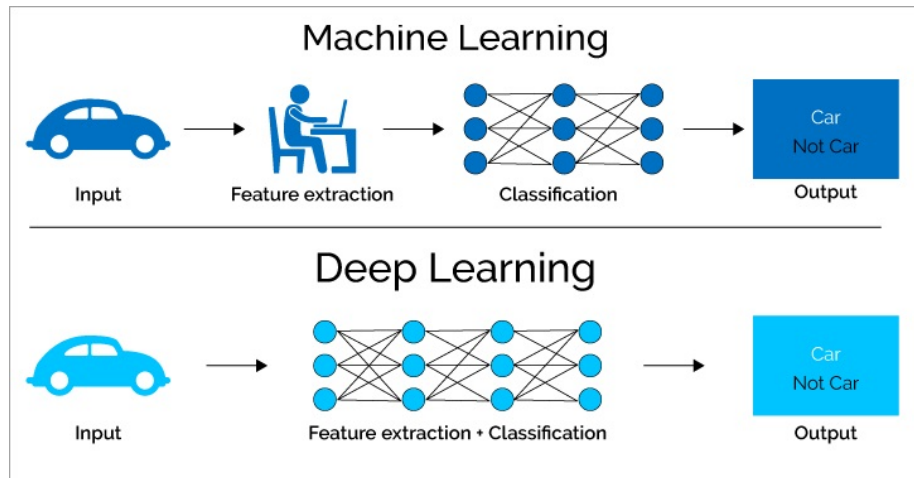


Figura 2.7: Comparación entre Machine Learning y Deep Learning

Los avances en la capacidad de cómputo y la posibilidad de crear conjuntos de datos masivos, han dado lugar a que modelos muy potentes puedan ser desarrollados. Estos modelos tienen del orden de miles de millones de parámetros, como pueden ser Stable Diffusion [9] o los LLM [10] (Large Language Models - Modelos Grandes del Lenguaje) que han dado lugar a aplicaciones como ChatGPT, y que están trayendo consigo una gran revolución para la humanidad.

2.2.2.1. Redes neuronales

Las redes neuronales son un modelo computacional que se basa en el funcionamiento del cerebro. Están formadas por neuronas, que se organizan en capas, y se conectan unas con otras.

Una de las arquitecturas más utilizadas en redes neuronales es el perceptrón multicapa, que consta de una capa de entrada, una o varias capas ocultas y una capa de salida. Cada neurona en una capa oculta está conectada a todas las neuronas de la capa anterior y a todas las neuronas de la capa siguiente. De esta forma, la red neuronal recibe los datos por su capa de entrada y, tras realizar sus respectivos cálculos, se los transmite a la siguiente capa. Este proceso se repite hasta que los datos llegan a la capa de salida, cuyos resultados se consideran la salida de la red neuronal. A este proceso se le conoce como “forward propagation”.

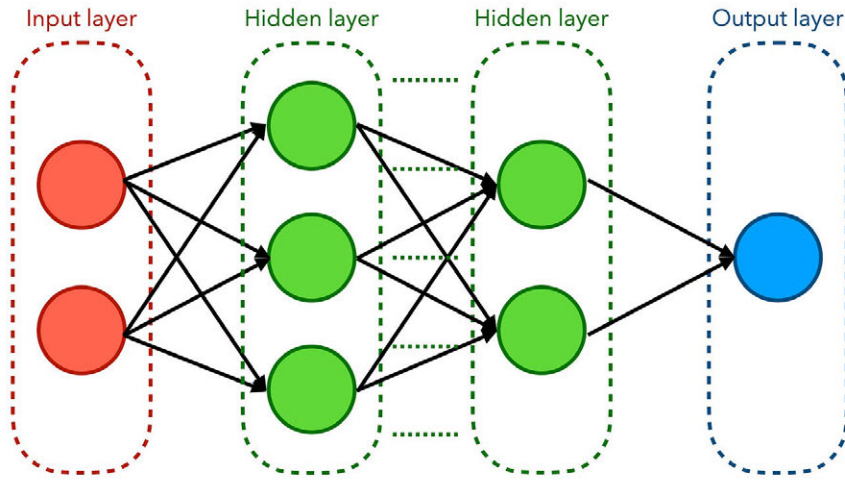


Figura 2.8: Imagen de una red neuronal

Los cálculos que realiza cada neurona se trata de una combinación lineal con sus coeficientes, conocidos como pesos, y los datos que recibe. Tras realizar este cálculo, se aplica una función no lineal, conocida como función de activación. Las funciones de activación mas habituales son las siguientes:

| Función de Activación | Fórmula | Rango |
|------------------------------|---|---------------------|
| Lineal | $f(x) = x$ | $(-\infty, \infty)$ |
| Sigmoide | $f(x) = \frac{1}{1+e^{-x}}$ | $(0, 1)$ |
| Tangente hiperbólica | $f(x) = \tanh(x)$ | $(-1, 1)$ |
| ReLU (Rectified Linear Unit) | $f(x) = \max(0, x)$ | $(0, \infty)$ |
| Leaky ReLU | $f(x) = \max(0.01x, x)$ | $(-\infty, \infty)$ |
| Softmax | $f(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$ | $(0, 1)$, suma a 1 |

Tabla 2.1: Principales funciones de activación para redes neuronales

Hay que tener en cuenta que hay arquitecturas muy diversas y complejas de redes neuronales. Algunas de ellas pueden ser, las redes neuronales convolucionales (CNN) [11], las redes neuronales recurrentes (RNN) [12], las redes neuronales generativas adversarias (GAN) [13] y las redes Transformers [14]. Cada una de estas arquitecturas está diseñada para resolver una tareas específica como son el análisis de imágenes, el procesamiento de secuencias, la generación de datos sintéticos y el procesamiento de lenguaje natural.

2.3. Herramientas y librerías

Para la realización de este trabajo se ha hecho uso de las siguientes herramientas:

- NumPy [15]: NumPy es una librería para Python que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas eficientemente.
- Pandas [16]: Pandas es una librería de Python especializada en la manipulación y el análisis de datos. Ofrece estructuras de datos y operaciones para manipular tablas numéricas, es como el Excel de Python. Ha sido usada para leer los ficheros CSV del dataset y modificar los datos para dejarlos listos para el entrenamiento.
- Matplotlib [17]: Matplotlib es una librería para la generación de gráficos en dos dimensiones, a partir de datos contenidos en listas o arrays en Python. Ha sido usada para crear las gráficas sobre los datos del entrenamiento de la red neuronal.
- Tensorflow [18] y Keras [19]: TensorFlow es una plataforma de código abierto para Machine Learning desarrollado por Google. Keras se ejecuta sobre Tensorflow, y proporciona una interfaz de alto nivel para la creación de redes neuronales.
- XGBoost [20]: XGBoost es una librería optimizada que implementa algoritmos de gradient boosting, diseñada para ser altamente eficiente, flexible y portable.
- Optuna [21]: Optuna es un framework de optimización automática de hiperparámetros, especialmente diseñado para el Machine Learning. Se ha usado con el modelo de gradient boosting.
- Holoviews [22] y Hvplot [23]: HvPlot es una librería construida sobre HoloViews, y del mismo modo que matplotlib, han sido usadas para la creación de gráficos.
- Vue.js [24]: Vue.js es un framework de JavaScript para la creación de interfaces web con HTML, CSS y JavaScript. Proporciona un modelo de programación declarativo y basado en componentes que facilita el desarrollo de interfaces de usuario. Ha sido usada para la creación de la interfaz web en la que se mostrará la información del tráfico.
- Flask [25]: Flask es un framework web ligero y flexible para el desarrollo de aplicaciones web en Python, que facilita la creación de sitios y servicios web de manera rápida y sencilla. Con ayuda de este framework, se ha implementado el servidor web.
- Pickle [26]: Es una librería que permite serializar y deserializar objetos Python en una secuencia de bytes, permitiendo guardar dichos objetos en ficheros. Ha sido usado para almacenar en ficheros los modelos entrenados, el label encoder y el standard scaler.

- Scikit-learn [27]: Scikit-learn es una librería de Machine Learning para Python que proporciona herramientas simples y eficientes. De entre todas sus funcionalidades, se ha usado para calcular las métricas.
- Google Colab [28]: Google Colab es un producto de Google Research. Permite a cualquier usuario escribir y ejecutar código arbitrario de Python a través del navegador. Es especialmente adecuado para tareas de Machine Learning y análisis de datos. Ha sido usado para procesar los datos, entrenar los modelos y crear un notebook para facilitar que otras personas puedan probar este trabajo.
- Ngrok [29]: Ngrok es un servicio que permite crear un servidor local en un subdominio para poder visualizarlo fuera de la LAN (Red de área local), a través de internet. Esto ha permitido que se pueda acceder al servidor creado en la máquina que se ejecuta el notebook de Google Colab.

2.4. Trabajos similares

A continuación, se muestra una lista de trabajos actuales que tienen objetivos similares a los de este:

- Enhanced Security Against Volumetric DDoS Attacks Using Adversarial Machine Learning [30]: Este trabajo aborda el reto de detectar ataques DDoS empleando una red generativa adversaria (GAN) para desarrollar un detector de ataques robusto. El modelo propuesto puede generar y clasificar elementos de tráfico sintéticos benignos y malignos, que son muy similares a las correspondientes reales.
- Detecting DDoS Attacks Using Machine Learning Techniques and Contemporary Intrusion Detection Dataset [31]: Este trabajo presenta un conjunto de datos fiable que contiene flujos de red benignos y de ataques comunes, y explora el uso de técnicas de aprendizaje automático para detectar y rastrear ataques DDoS.
- Machine Learning Techniques for Detecting DDOS Attacks [32]: Este artículo analiza el uso de técnicas de aprendizaje automático, en concreto Random Forest, Árboles de decisión, SVM (Support vector machine), Naive Bayes y XGBoost, para detectar ataques DDoS con un alto nivel de precisión. También utiliza el conjunto de datos CICDDoS2019 para el análisis.
- Impact of Feature Selection Methods on Machine Learning-based for Detecting DDoS Attacks : Literature Review [33]: Este estudio se centra en el impacto de los métodos de selección de características en la detección de ataques DDoS basada en aprendizaje automático. Destaca la importancia de la ingeniería de características (feature engineering) y su influencia en la precisión de las soluciones de aprendizaje automático para ataques DDoS.

La novedad que trae este trabajo que no incluye ninguno de los mencionados en la lista, es que se incluye una herramienta de captura de tráfico y una interfaz gráfica en la que se puede observar el funcionamiento del sistema en tiempo real.

Capítulo 3

Desarrollo del proyecto

3.1. Dataset

Para entrenar un modelo capaz de detectar ataques, es necesaria una gran cantidad de datos de tráfico. Con este objetivo, se ha elegido el dataset CICDDoS2019 [34], que contiene tráfico benigno y ataques de denegación de servicio comunes y actualizados, que se asemejan a datos verdaderos del mundo real (PCAPs). Este dataset consiste en capturas de Wireshark, sin embargo, también incluye ficheros CSV que contienen resultados de un análisis del tráfico con flujos etiquetados en función de las IP de origen y destino, los puertos de origen y destino, los protocolos y el tipo ataque. Dicho análisis se ha obtenido con CICFlowMeter [35], una herramienta para analizar el tráfico capaz de proporcionar más de 80 variables. Estos ficheros CSV serán usados para entrenar a los modelos posteriormente y la descripción de cada una de las variables presentes se puede ver en <https://github.com/ahlashkari/CICFlowMeter/blob/master/ReadMe.txt>.

Para implementar el procesamiento de los datos y el entrenamiento de los modelos se ha empleado un notebook. Este puede ser accedido a través de https://colab.research.google.com/github/DavidRamosArchilla/Firewall-AI/blob/main/notebooks/Data_processing_and_model_training.ipynb.

3.1.1. Procesado del dataset

El tamaño de los ficheros que forman el dataset es bastante elevado, del orden de GigaBytes, y esto puede ser un factor limitante a la hora de cargarlo en memoria para entrenar el modelo. Es por ello por lo que se ha decidido hacer un preprocesado con la intención de reducir su tamaño. Para ello, se ha empleado el código A.1.

Este preprocesado consiste en lo siguiente: primero, se eliminan columnas que no serán útiles para entrenar al modelo, como pueden ser las IPs origen y destino y los puertos origen y destino, segundo, se eliminan los valores nulos (o nan) y los elementos repetidos en el dataset, luego se transforman los tipos de datos de las columnas en otros equivalentes que ocupen menos (por ejemplo convertir de float64 a float32) y por último se guardan los ficheros en disco en formato Parquet. Parquet es un formato de archivo de almacenamiento en columnas optimizado para su uso con frameworks de procesamiento de big data. Está diseñado para proporcionar un almacenamiento y procesamiento eficientes de grandes conjuntos de datos. Los archivos Parquet almacenan los datos de forma altamente comprimida y divisible, lo cual es ideal para reducir el tamaño del dataset en disco.

3.1.2. Análisis del dataset

Para tener una vista general sobre como se distribuyen los datos dentro del dataset, se ha aplicado un algoritmo de reducción dimensional para poder crear una imagen en 2 dimensiones del dataset. El algoritmo que se ha usado es t-sne [36] y la imagen que se obtiene es la siguiente:

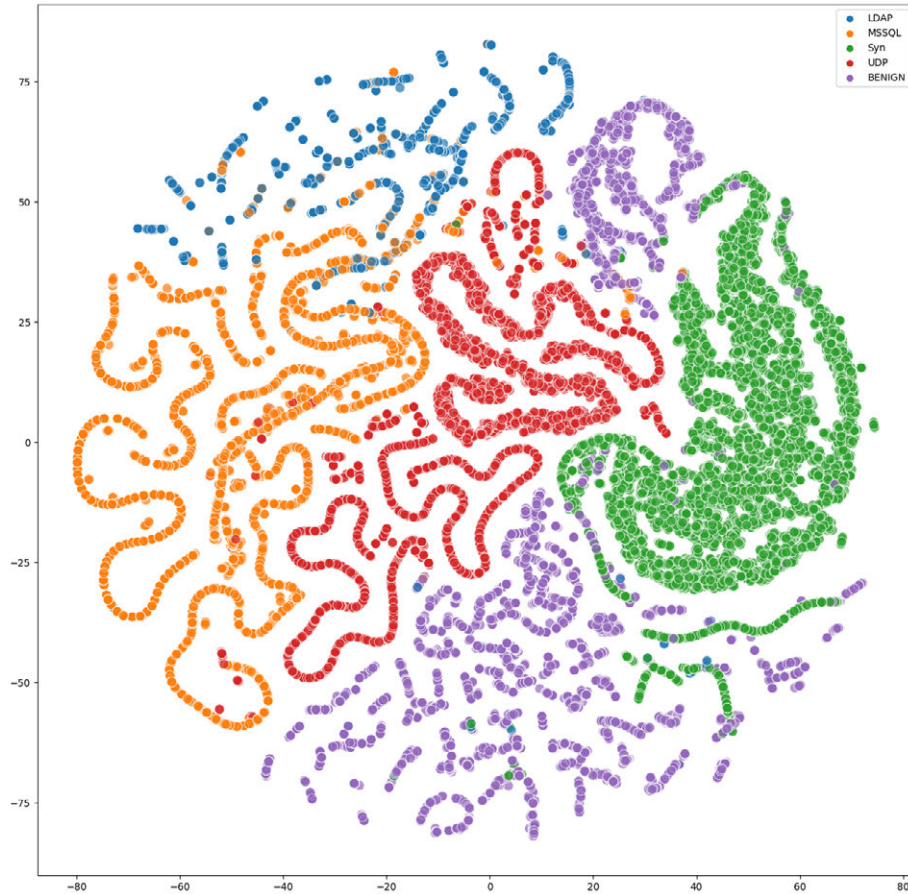


Figura 3.1: Visualización del dataset

Por otro lado, el dataset contiene un campo llamado Label, que indica el tipo de ataque. En el siguiente diagrama se muestra la cantidad de datos para cada tipo de ataque:

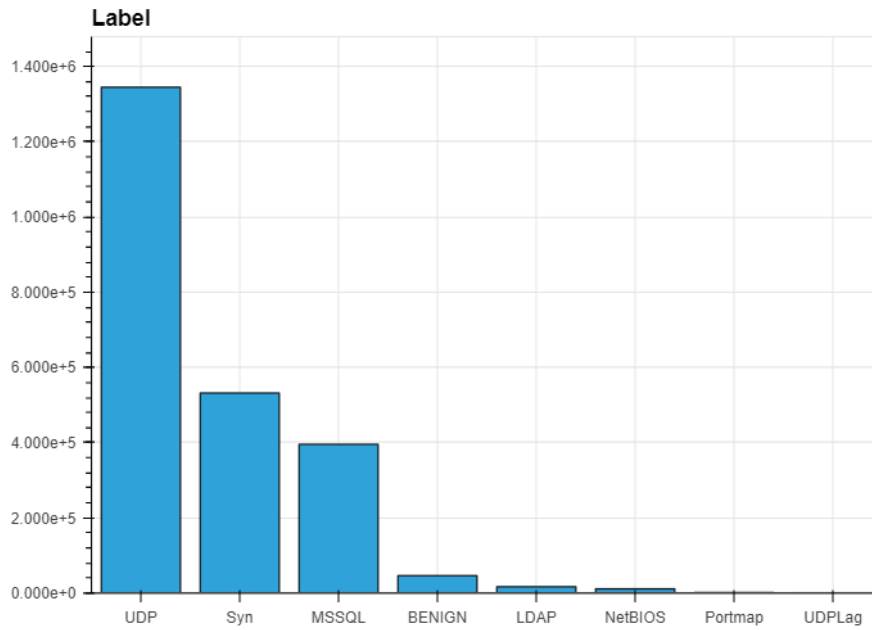


Figura 3.2: Distribución de los tipos de ataques

Como se puede ver, el dataset está muy desbalanceado y hay clases que tiene demasiados pocos datos como para que un modelo pueda aprender sobre ellos. Por tanto, se han eliminado las 3 clases con menos ocurrencias y se han balanceado el resto de clases, quedando el nuevo dataset de la siguiente forma:

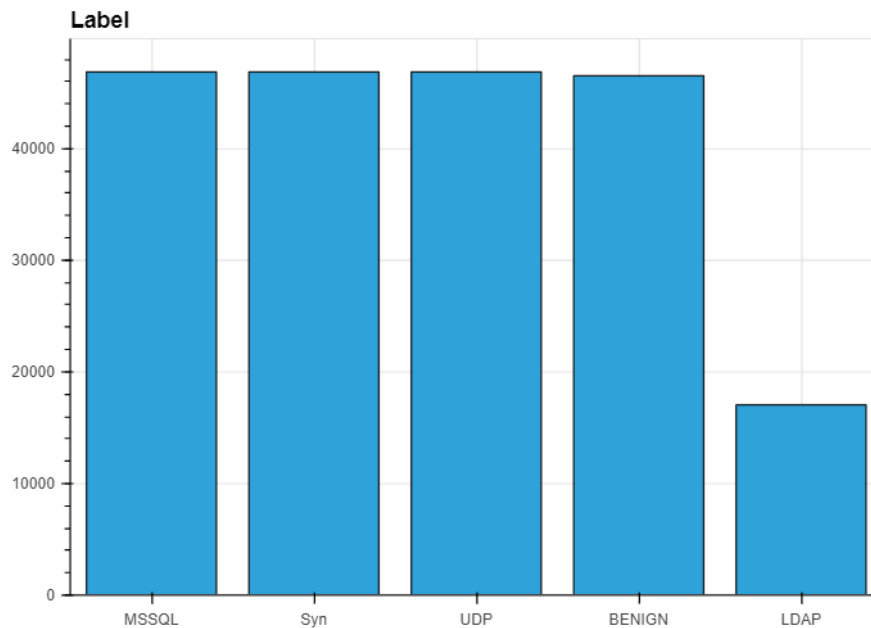


Figura 3.3: Distribución de los tipos de ataques tras el balanceado

3.2. Entrenamiento de modelos

Una vez se tiene el dataset procesado, ya estaría casi todo listo para el entrenamiento. Uno de los últimos pasos que faltan es la normalización de las variables de entrada. Para normalizar una variable, se le resta su media y se divide por su desviación típica. La normalización del dataset es un requisito común para muchos estimadores de aprendizaje automático: podrían comportarse mal si las características individuales no se parecen más o menos a datos estándar distribuidos normalmente (por ejemplo, gaussianos con media 0 y varianza unitaria).

Por ejemplo, muchos elementos utilizados en la función objetivo de un algoritmo de aprendizaje suponen que todas las características están centradas en torno a 0 y tienen una varianza del mismo orden. Si una característica tiene una varianza que es varios órdenes de magnitud mayor que otras, puede dominar la función objetivo y hacer que el estimador no pueda aprender de otras características correctamente como se espera, lo que traería una reducción en la precisión del modelo. Para realizar la normalización, se ha usado la clase `StandardScaler` de la librería `scikit-learn`.

Por último, las etiquetas que indican el tipo de ataque vienen como cadenas de texto, por lo que hay que convertirlas a valores numéricos para que los modelos puedan trabajar con ellas. Para ello, se ha hecho uso de la clase `LabelEncoder` de la librería `scikit-learn`.

Tras realizar estas transformaciones, ya se puede pasar a la fase de entrenamiento de los modelos.

3.2.1. Cross-validation

Durante el entrenamiento de los modelos se emplea la técnica de cross-validation (o validación cruzada), que consiste en dividir el dataset en múltiples subconjuntos llamados folds (o pliegues). Luego, se entrena y evalúa el modelo en diferentes combinaciones de estos pliegues para ver como se comporta con cada uno de ellos.

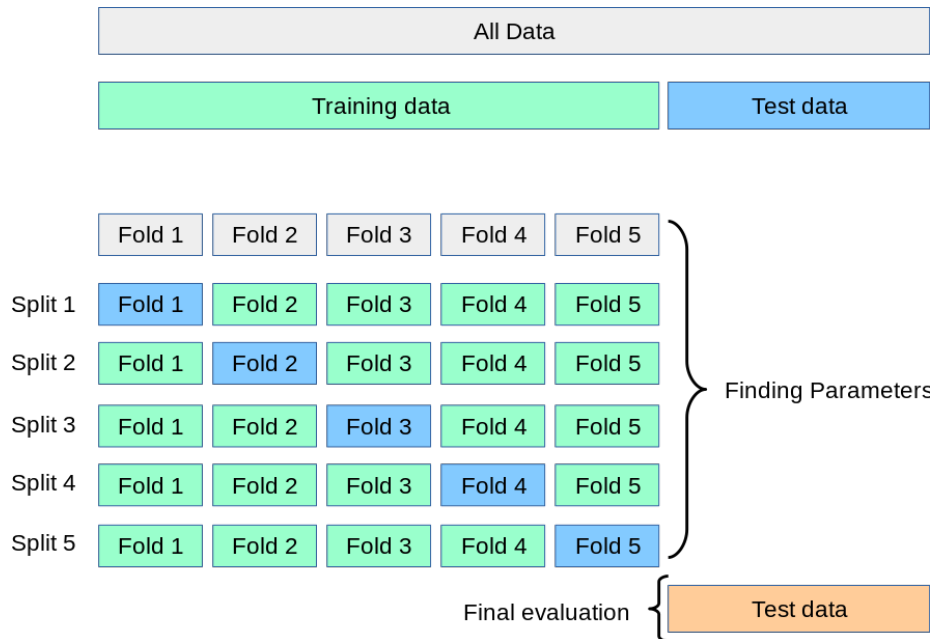


Figura 3.4: Esquema cross-validation

Al evaluar el modelo en diferentes subconjuntos de datos, se obtiene una medida más robusta del rendimiento del modelo en comparación con una única división entrenamiento/prueba. Además, se reduce la posibilidad de que se produzcan sesgos debidos a una división específica que favorezca accidentalmente al modelo.

3.2.2. Red neuronal

Para construir la red neuronal se han empleado librerías especializadas. En concreto, se ha usado Keras, una librería de Deep Learning de alto nivel escrita sobre TensorFlow. El tipo de red neuronal a construir es un perceptrón multicapa.

El modelo consta de una capa de entrada, que tiene un tamaño igual al número de variables de entrada que recibirá la red neuronal; dos capas ocultas, ya que menos capas puede llevar a que el modelo no aprenda lo suficiente y más capas aumenta el coste computacional y puede producir overfitting; y una capa de salida con tantas neuronas como clases hay para predecir, en este caso, 5.

Tras varias pruebas, se ha establecido a 64 el número de neuronas de cada una de las capas ocultas se obtienen unos buenos resultados, por lo que el modelo queda definido de la siguiente forma:

Model: "model_1"

| Layer (type) | Output Shape | Param # |
|-------------------------------------|--------------|---------|
| input_2 (InputLayer) | [(None, 75)] | 0 |
| dense_3 (Dense) | (None, 64) | 4864 |
| dense_4 (Dense) | (None, 64) | 4160 |
| dense_5 (Dense) | (None, 5) | 325 |
| Total params: 9349 (36.52 KB) | | |
| Trainable params: 9349 (36.52 KB) | | |
| Non-trainable params: 0 (0.00 Byte) | | |

Como se puede ver, el número de parámetros de la red neuronal es 9349 y el tamaño que ocupa es 36.52KB, por lo que será bastante eficiente a la hora de realizar inferencias incluso en CPUs.

En cuanto a las funciones de activación de las capas de la red neuronal tenemos las siguientes:

- ReLU: La función de activación ReLU (Rectified Linear Unit) es una función no lineal ampliamente utilizada en redes neuronales y modelos de aprendizaje profundo para las capas ocultas. La función ReLU se define matemáticamente como: $Relu(z) = \max(0, z)$.

Esta función de activación la tendrán las 2 capas ocultas.

- Softmax: La función de activación softmax es comúnmente utilizada en problemas de clasificación en aprendizaje automático y redes neuronales. Su función principal es convertir un vector de números reales en un vector de probabilidades. Es particularmente útil cuando se trata de problemas de clasificación multiclase, lo cual la hace esencial para este caso. Se define de la siguiente forma:

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

siendo K el número de clases.

Esta es la función de activación que tendrá la capa de salida.

Una cosa a destacar, es el hecho de que la función softmax devuelva un vector de probabilidades. Cuando se ponga el funcionamiento el sistema, esto será útil para saber la confianza con la que el modelo realiza las predicciones.

Debido a que el dataset no esta completamente balanceado (la clase LDAP tiene menos elementos) se van a establecer unos pesos para cada una de las

clases de tal forma que el modelo prestará más atención a las clases con mayor peso. Esta medida deberá reducir el impacto que pueda tener el hecho de que el dataset esté ligeramente desbalanceado.

Por último, para que el error del modelo pueda ser calculado correctamente, hay que aplicar un one-hot encoding a la variable objetivo, en este caso, al tipo de ataque. Esto consiste en convertir la variable en un vector del mismo tamaño que el número de elementos diferentes que haya en la variable, en este caso 5. Los vectores tendrán un 0 en todos sus elementos salvo en el índice de la clase que representen.

La función de coste que se emplea es la entropía cruzada categórica, y se calcula usando la clase `CategoricalCrossentropy` de TensorFlow. Su formulación matemática es la siguiente

$$L(y, \hat{y}) = - \sum_i y_i \cdot \log(\hat{y}_i)$$

Donde:

- $L(y, \hat{y})$ representa el error de entropía cruzada (el nombre de la función).
- y representa los datos reales a predecir (etiquetado con one-hot encoding).
- \hat{y} representa las predicciones.
- i itera sobre todas las clases.

Para el entrenamiento se emplea el código de A.2. Se puede observar que se aplica validación cruzada con un total de 5 folds. Otros hiperparámetros importantes son el batch size (o tamaño de batch), establecido a 32, y el learning rate (o tasa de aprendizaje), con un valor de 0.0001. Estos valores se han escogido para tratar de prevenir que el modelo tenga overfitting. Por último, el número de epochs se ha establecido a 15, suficientes para que el modelo tenga una buena precisión.

Además, en cada iteración se crea una gráfica con los errores en entrenamiento y validación para ver la evolución del modelo, así como un informe de clasificación, que contiene métricas sobre las predicciones del conjunto de validación. Por ejemplo, se puede ver la precisión para cada una de las clases, lo cual es un indicador muy bueno para ver como está funcionando el modelo.

A continuación se muestra dicho informe de clasificación y una gráfica con los errores en entrenamiento y validación a lo largo de los epochs para uno de los folds:

| Class | Precision | Recall | F1-Score | Support |
|---------------------|-----------|--------|----------|---------|
| BENIGN | 0.99 | 1.00 | 1.00 | 7443 |
| LDAP | 0.84 | 0.97 | 0.90 | 2739 |
| MSSQL | 0.99 | 0.93 | 0.96 | 7495 |
| Syn | 1.00 | 1.00 | 1.00 | 7501 |
| UDP | 1.00 | 1.00 | 1.00 | 7487 |
| Accuracy | | | 0.98 | 32665 |
| Macro Avg | 0.96 | 0.98 | 0.97 | 32665 |
| Weighted Avg | 0.98 | 0.98 | 0.98 | 32665 |

Tabla 3.1: Informe clasificación red neuronal durante cross-validation

Descripciones de las Columnas:

- **Clase:** Representa los diferentes tipos de ataques (o BENIGN en caso de no serlo).
- **Precision:** Proporción de verdaderos positivos respecto a la suma de verdaderos positivos y falsos positivos para cada clase. Se calcula con la fórmula:

$$\text{Precision} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Positivos}}$$

- **Recall (Sensibilidad):** Proporción de verdaderos positivos respecto a la suma de verdaderos positivos y falsos negativos para cada clase. Se calcula con la fórmula:

$$\text{Recall} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Negativos}}$$

- **F1-Score:** Media armónica de precisión y recall para cada clase. Proporciona una medida balanceada ya que considera falsos y verdaderos positivos. Se calcula con la fórmula:

$$F1 - Score = 2 \cdot \frac{\text{Precisión} \cdot \text{Recall}}{\text{Precisión} + \text{Recall}}$$

- **Support:** Número de ocurrencias reales de cada clase.
- **Accuracy (Exactitud):** Corrección general del modelo de clasificación. Se calcula con la fórmula:

$$\text{Accuracy} = \frac{\text{Predicciones Correctas}}{\text{Total de Predicciones}}$$

- **Macro Avg:** Promedio de precisión, recall y F1-Score para todas las clases.

- **Weighted Avg:** Promedio ponderado de precisión, recall y F1-Score considerando el desbalance de clases.

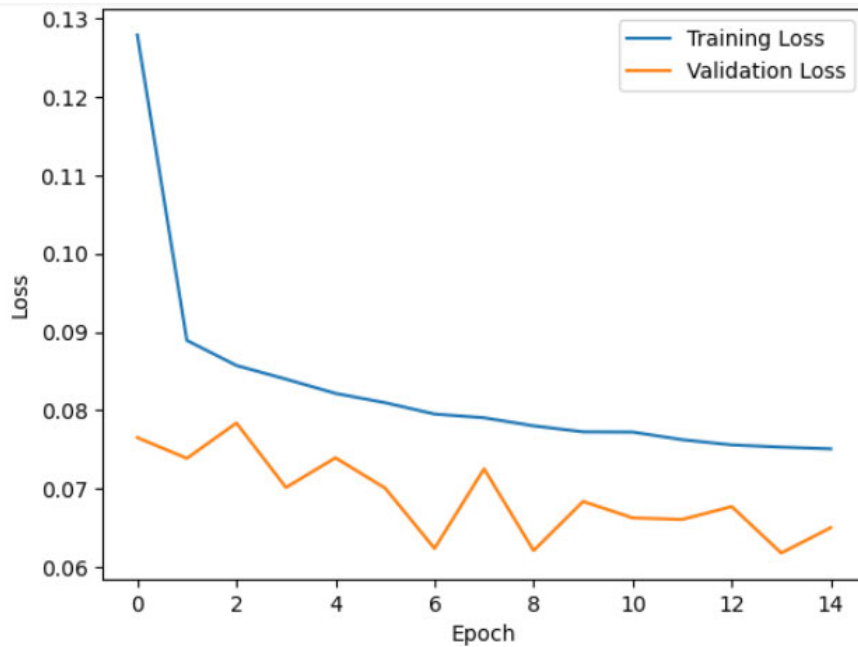


Figura 3.5: Gráfica de error durante cross-validation en la red neuronal

Tras aplicar cross-validation, se observa que el modelo tiene una precisión media del 98 %, por lo que se procede a entrenar el modelo con todo el conjunto de entrenamiento. Los resultados obtenidos son los siguientes:

| Class | Precision | Recall | F1-Score | Support |
|---------------------|-----------|--------|----------|---------|
| BENIGN | 1.00 | 0.99 | 0.99 | 9308 |
| LDAP | 0.82 | 0.96 | 0.89 | 3345 |
| MSSQL | 0.98 | 0.93 | 0.96 | 9388 |
| Syn | 1.00 | 1.00 | 1.00 | 9360 |
| UDP | 1.00 | 0.99 | 1.00 | 9431 |
| Accuracy | | | 0.98 | 40832 |
| Macro Avg | 0.96 | 0.98 | 0.97 | 40832 |
| Weighted Avg | 0.98 | 0.98 | 0.98 | 40832 |

Tabla 3.2: Informe de clasificación red neuronal final

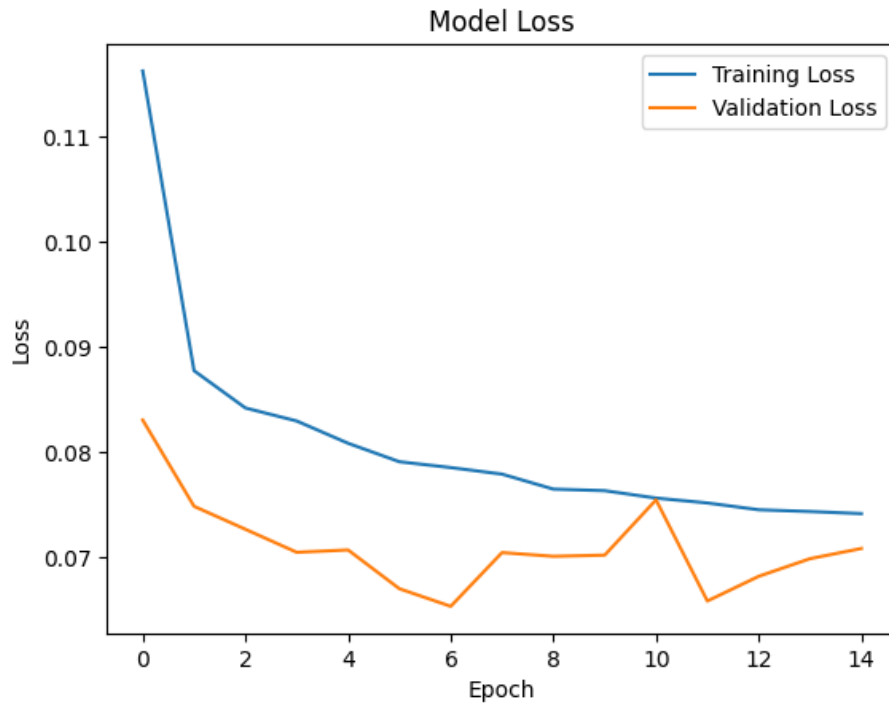


Figura 3.6: Gráfica de error durante el entrenamiento de la red neuronal final

Matriz de confusión

| | | | | | |
|-------------|-----------------|---------------|----------------|--------------|--------------|
| Real BENIGN | 9232 | 53 | 0 | 23 | 0 |
| Real LDAP | 7 | 3216 | 102 | 20 | 0 |
| Real MSSQL | 6 | 627 | 8754 | 1 | 0 |
| Real Syn | 25 | 5 | 3 | 9324 | 3 |
| Real UDP | 0 | 1 | 53 | 1 | 9376 |
| | Predicho BENIGN | Predicho LDAP | Predicho MSSQL | Predicho Syn | Predicho UDP |

Figura 3.7: Matriz de confusión de la red neuronal final

Esta última gráfica es una matriz de confusión. Cada fila indica para cada valor real, cuantas predicciones ha hecho el modelo de cada una de las clases, siendo el eje X las predicciones y el eje Y lo datos reales. Por ejemplo, el modelo predijo 102 veces que un flujo era un ataque MSSQL cuando en realidad era LDAP, y predijo 627 veces que un flujo era un ataque LDAP cuando en realidad se trataba de un ataque MSSQL.

3.2.3. Modelo de gradient boosting

El otro modelo que se ha entrenado se basa en otras técnicas de Machine Learning y se trata de un modelo basado en algoritmos de gradient boosting [37]. Básicamente, esta técnica consiste en un conjunto de árboles de decisión débiles que se entrenan de manera secuencial para corregir los errores del modelo anterior. Cada árbol se enfoca en los errores restantes del conjunto, y la salida final es la suma ponderada de las predicciones de todos los árboles, mejorando gradualmente la precisión del modelo. Para crear el modelo, se ha empleado la clase `XGBClassifier` de la librería `XGBoost`.

Para escoger los hiperparámetros se ha empleado la librería `Optuna`, una librería creada para este tipo de propósitos. Simplemente hay que definir una función objetivo a optimizar, proporcionar unos rangos de búsqueda para los hiperparámetros y elegir la cantidad de iteraciones que durará la optimización.

Durante la búsqueda de hiperparámetros, será necesario entrenar el modelo, que se hará empleando `cross-validation` al igual que se hizo con la red neuronal. El código correspondiente es A.3. Con el objeto `trial`, se sugiere un rango del que se escogerá un valor para el hiperparámetro en cuestión.

Tras dejar ejecutando el estudio, se obtienen los siguientes hiperparámetros:

```
{
    'lambda': 3.242851557666658e-06,
    'alpha': 1.8440002794817977e-06,
    'max_depth': 9,
    'eta': 0.1294208168667795,
    'gamma': 8.192698273313952e-07,
    'colsample_bytree': 0.14470448735955996,
    'min_child_weight': 6,
    'n_estimators': 1023
}
```

Con el módulo `optuna.visualization` se pueden obtener gráficas sobre la optimización. Uno de ellos es el siguiente, que muestra la importancia que ha tenido cada hiperparámetro durante la optimización:

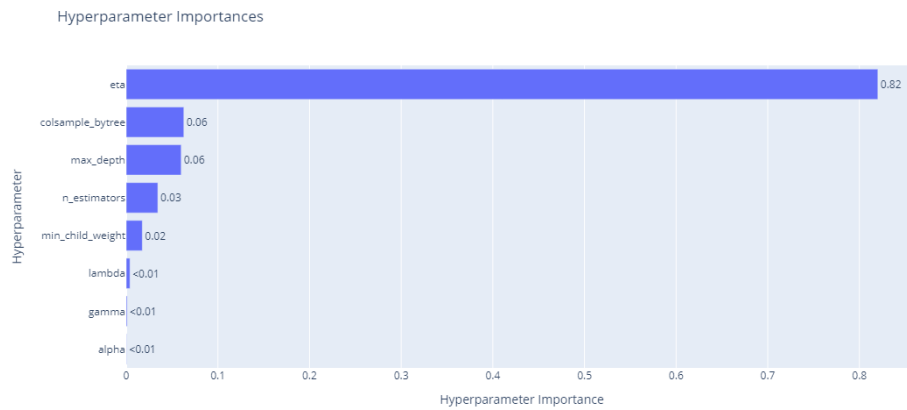


Figura 3.8: Importancia de los hiperparámetros

También se pueden crear otras gráficas que aportan información de valor sobre el proceso de optimización como:

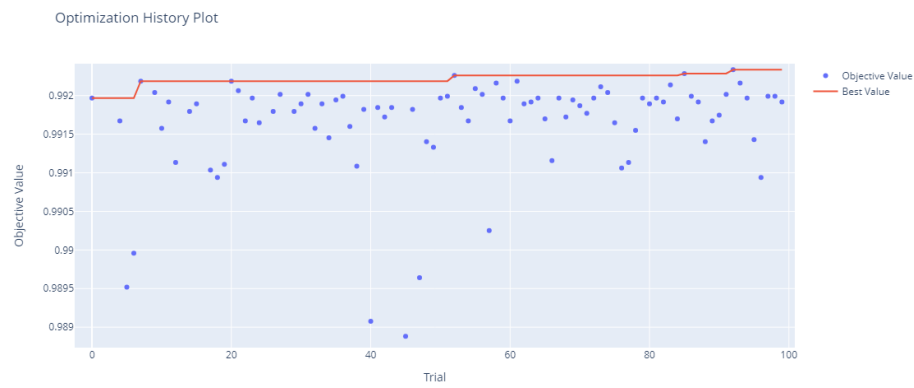


Figura 3.9: Histórico de la precisión durante la optimización

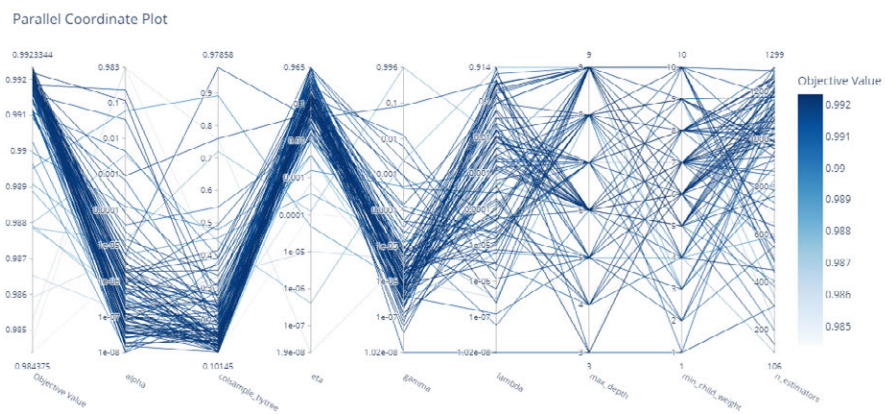


Figura 3.10: Distribución de los parámetros probados durante la optimización

La primera, muestra la evolución de la precisión del modelo durante el experimento. La segunda, muestra como se han distribuido las combinaciones de los diferentes hiperparámetros y qué precisión se ha alcanzado con ellos. Esto puede ser útil para modificar los rangos sobre los que se realiza la búsqueda de los hiperparámetros, ya que si los valores se agrupan mucho sobre un extremo puede indicar que hay que aumentar el rango por dicho extremo, y al contrario ocurre si en un extremo se hay pocos valores escogidos. Por ejemplo, se puede observar que esto ocurre con la variable eta, que hace referencia a la tasa de aprendizaje.

Tras obtener unos hiperparámetros con los que iniciar el modelo, se entrena el modelo empleando el conjunto de entrenamiento. Los resultados tras el entrenamiento son los siguientes:

| Class | Precision | Recall | F1-Score | Support |
|---------------------|------------------|---------------|-----------------|----------------|
| BENIGN | 1.00 | 1.00 | 1.00 | 9308 |
| LDAP | 0.95 | 0.97 | 0.96 | 3345 |
| MSSQL | 0.99 | 0.98 | 0.98 | 9388 |
| Syn | 1.00 | 1.00 | 1.00 | 9360 |
| UDP | 1.00 | 1.00 | 1.00 | 9431 |
| Accuracy | | | 0.99 | 40832 |
| Macro Avg | 0.99 | 0.99 | 0.99 | 40832 |
| Weighted Avg | 0.99 | 0.99 | 0.99 | 40832 |

Tabla 3.3: Informe de clasificación del modelo de gradient boosting

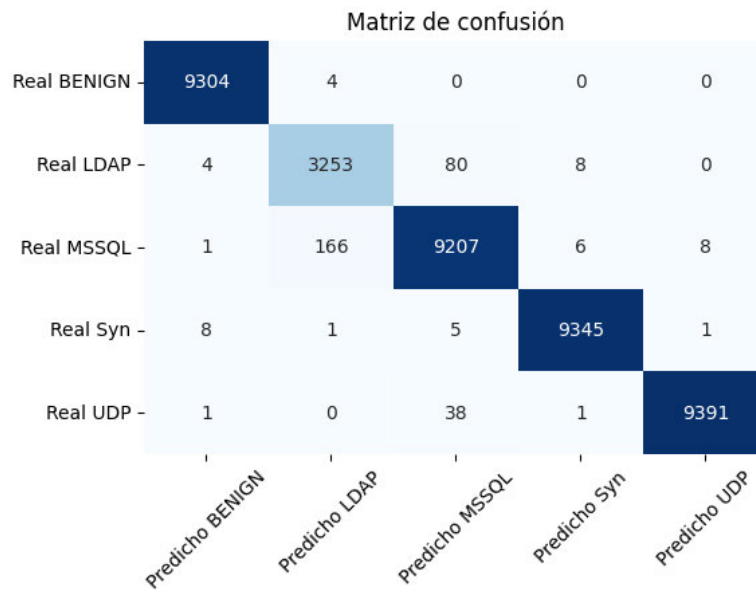


Figura 3.11: Matriz de confusión del modelo de gradient boosting

Uno de los aspectos positivos que tiene este modelo, es que al estar basado en árboles de decisión es posible ver en qué variables se apoya para realizar las predicciones. Es por esto por lo que se puede ver cómo de importante es cada una de las variables del dataset, algo que puede ser muy útil a la hora de seleccionar las variables de entrada para un modelo, en caso de ser necesario. A continuación, se muestra una gráfica con las 25 variables más importantes según este modelo.

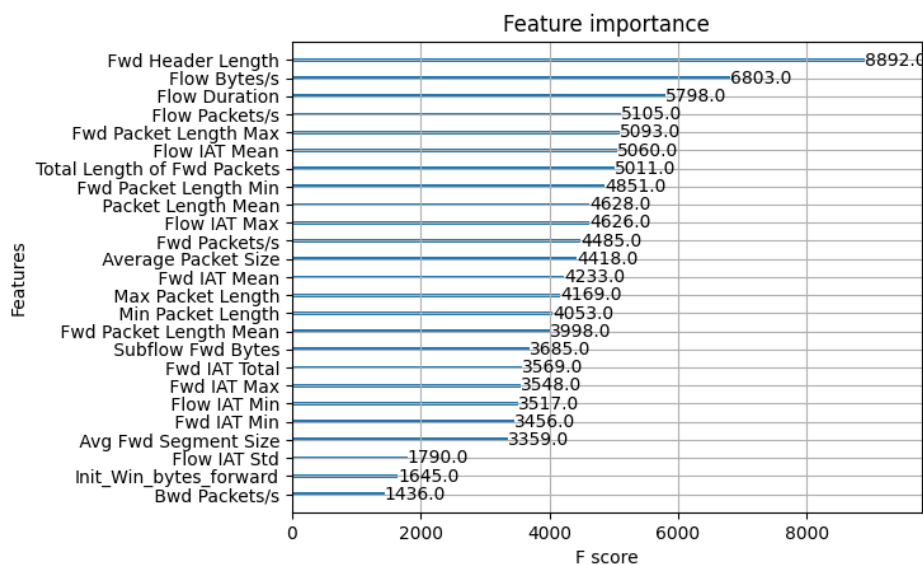


Figura 3.12: Importancia de las variables de entrada

3.3. Herramienta de captura de tráfico

Para poner en funcionamiento los modelos entrenados, es necesario de alguna forma capturar el tráfico de red para que luego sea analizado. La misma institución que creó el dataset empleado en este trabajo, el CiC (Canadian Institute for Cybersecurity), tiene también una herramienta de captura de tráfico, a la que han llamado CicFlowmeter, ya mencionada anteriormente. Esta herramienta, además de tener la capacidad de producir ficheros CSV que contiene variables sobre el tráfico a partir de ficheros pcap (es un formato habitual para almacenar capturas de paquetes) o similares, también puede leer los paquetes que recibe una interfaz de red en tiempo real. Esto lo convierte en la herramienta ideal para el sistema a desarrollar, ya que permitirá capturar el tráfico y, a su vez, calcular las variables de entrada para el modelo, todo ello en tiempo real.

Sin embargo, hay un pequeño inconveniente, la herramienta está escrita en java y complica la integración con el resto del sistema. Afortunadamente, la comunidad ha creado una versión similar para Python [38]. Como punto de inicio está muy bien, pero no es suficiente para lo que se busca, por lo que, a partir de un fork, se ha creado una librería en Python para satisfacer las necesidades de este trabajo.

3.3.1. PyFlowmeter

PyFlowmeter [39] es el nombre de la librería que se ha creado. Para instalarla, se puede hacer de manera muy sencilla ejecutando: `pip install pyflowmeter`. Como ya se ha mencionado, se han hecho modificaciones al proyecto original para adaptarlo a este trabajo. Algunas de sus funcionalidades son:

- Captura de paquetes en tiempo real. Esta herramienta será capaz de capturar paquetes en tiempo real con tan solo especificar una o varias interfaces por las que escuchar. Para obtener solamente los paquetes necesarios, se le ha añadido un filtro, que es “ip and (tcp or udp)”.
- Simulación de tráfico de red a partir de ficheros pcap. A través de ficheros que contienen datos de tráfico de red, la herramienta puede simular dicho tráfico, como si lo estuviera recibiendo. Esto es muy útil la hora de realizar pruebas sobre el sistema.
- Envío de paquetes capturados a un servidor. Periódicamente, la herramienta podrá enviar los datos de tráfico que tenga almacenados a un servidor a especificar. El envío de estos datos se realiza en segundo plano y puede ser tanto de tráfico capturado en tiempo real por las interfaces de red o de tráfico simulado.
- Generación de ficheros CSV sobre el tráfico de red. La herramienta es capaz de generar un fichero CSV con más de 70 variables sobre el tráfico de red, ya sea este en tiempo real o simulado desde un fichero. En caso de tratarse de un fichero de simulación, esta funcionalidad puede verse como la conversión de ficheros de paquetes de tráfico a ficheros CSV con variables sobre dicho tráfico.

Hay que destacar que los datos que recoge PyFlowmeter los almacena en forma de flujos. Los flujos de red son secuencias de datos que representan la comunicación entre dispositivos en una red. Cada flujo está compuesto por información como direcciones IP de origen y destino, puertos, protocolos y la cantidad de datos transmitidos. De esta forma, cada línea de los ficheros CSV corresponderá con un flujo y los datos que se enviarán al servidor serán una lista de flujos.

Por último, los siguientes fragmentos de código muestran ejemplos de cómo puede ser usada esta librería:

- Obtener análisis CSV de un archivo pcap: A.4
- Escanear paquetes en tiempo real desde la interfaz de red y enviar los datos a un servidor cada 5 segundos: A.5
- Simular tráfico desde un fichero pcap: A.6

Todo ello y más puede ser visto en detalle en <https://pypi.org/project/pyflowmeter/>

3.4. Arquitectura del sistema

El sistema está formado por diferentes componentes o subsistemas tal y como se muestra en el siguiente diagrama:

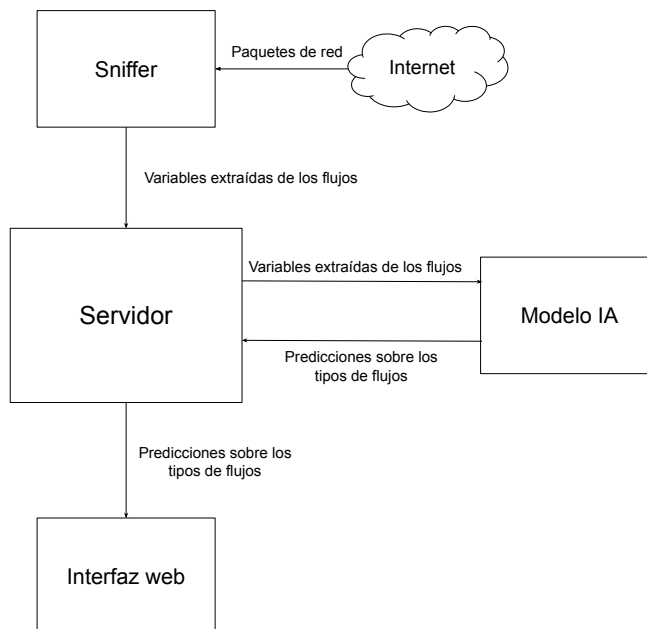


Figura 3.13: Diagrama sobre el sistema

Primero, el sniffer (o analizador de paquetes) capturará todos los paquetes que se reciban por las interfaces de red seleccionadas y enviará al servidor las variables sobre los datos de los flujos de red. Luego, el servidor hará uso de los modelos de inteligencia artificial para analizar los flujos y determinar si se tratan de un ataque o no. Esto se hace a través del componente modelo, que estará integrado en el servidor. Por último, el servidor mostrará, a través de un servicio http, una aplicación web en la que se puede ver el tráfico recibido y las predicciones que hecho el modelo sobre él, entre otras cosas.

El sistema ha sido desarrollado en WSL (Windows Subsystem for Linux), con una distribución Debian. Todo el código está disponible en github (<https://github.com/DavidRamosArchilla/Firewall-AI>) para que pueda ser replicado con facilidad.

3.4.1. Servidor

EL servidor es el componente principal del sistema y será el encargado de enviar y recibir los datos. Para su implementación se ha usado la librería Flask de python.

Por simplicidad, el sniffer se ha incluido en el servidor. La alternativa a esto sería crear una máquina que ejecutara el sniffer y que se comuniqué con otra máquina que estaría ejecutando el servidor. El servidor enviaría de vuelta las predicciones al sniffer y sería este el que actuaría como firewall. Este sniffer se crea haciendo uso de la herramienta explicada anteriormente, PyFlowmeter. Su forma de funcionamiento es la siguiente: el sniffer captura todos los paquetes que recibe por la interfaz especificada y los agrupa en flujos. Para cada uno de estos flujos, el sniffer es capaz de obtener variables sobre ellos, que son las mismas variables que se usaron para el entrenamiento. Por último, el sniffer tiene que enviar estos datos al servidor. Para ello, hay un hilo que cada ciertos segundos envía los datos sobre los flujos al servidor. La frecuencia con la que se envían los datos se puede configurar, y por defecto se envían cada segundo.

Al estar incluido en el servidor, el sniffer podrá ser iniciado desde la interfaz web. Para ello se ha creado un endpoint, `/start_sniffer` A.7, que podrá iniciar el sniffer para leer el tráfico de red en tiempo real o simular tráfico a partir de unos ficheros a elegir (esto último será útil para hacer las pruebas).

Además, el servidor tendrá otros endpoints como `/send_traffic` y `/get_data` A.8. El endpoint `/send_traffic` será de tipo POST, y por ahí es por donde el servidor recibirá los flujos obtenidos por el sniffer. Tras recibir los flujos, lo primero que hace el servidor es convertir los datos a un diccionario de Python. Luego, pasa estos datos al modelo para que este los procese y realice las predicciones. Tras obtener estas predicciones, el servidor las almacena en una lista. Por otro lado, `/get_data`, que es de tipo GET, devolverá una lista con todos los flujos y la predicción realizada por el modelo de inteligencia artificial, lo cual será de importancia a la hora de elaborar la interfaz web.

Por último, para arrancar el servidor hay que ejecutar el comando `'sudo venv/bin/python app.py'`. Sobre este comando, cabe destacar que es necesario tener permisos de superusuario para que el sniffer pueda capturar los paquetes de red y que la ruta que se indica de Python es de un entorno virtual. Un entorno virtual de Python es un directorio aislado que permite hacer la instalación de las dependencias de un proyecto, sin necesitar instalarlas en todo el sistema.

Para replicar el servidor, se pueden seguir los siguientes pasos: Primero, hay que instalar las dependencias y clonar el repositorio del proyecto:

```
sudo apt install libpcap0.8
sudo apt install tcpdump
git clone https://github.com/DavidRamosArchilla/Firewall-AI.git
cd Firewall-AI
```

Luego, es recomendable crear un entorno virtual de Python

```
python -m venv venv
source venv/bin/activate
```

e instalar las librerías:

```
pip install -q --upgrade pip
pip install -q pyflowmeter
pip install -q -r Firewall-AI/requirements.txt
```

Por último, se ejecuta el comando mencionado anteriormente:

```
sudo venv/bin/python app.py
```

3.4.2. Modelo de inteligencia artificial

El módulo del modelo de inteligencia artificial es el encargado de realizar las predicciones sobre si los flujos de tráfico son ataques o no. Para ello, primero hay que normalizar los datos de entrada empleando el mismo “normalizador” que durante el entrenamiento. Luego, se realiza la inferencia con el modelo, que produce un vector de enteros entre 0 y el número de clases menos 1 (4 en este caso) con tantos elementos como flujos se pasen como entrada. Por último, se convierten estos enteros, que representan la clase del flujo, a sus etiquetas correspondientes usando el mismo objeto `LabelEncoder` que se usó durante el entrenamiento.

Todo ello está encapsulado en una clase, y es el servidor quien crea un objeto de dicha clase para realizar las predicciones.

El código correspondiente es A.10

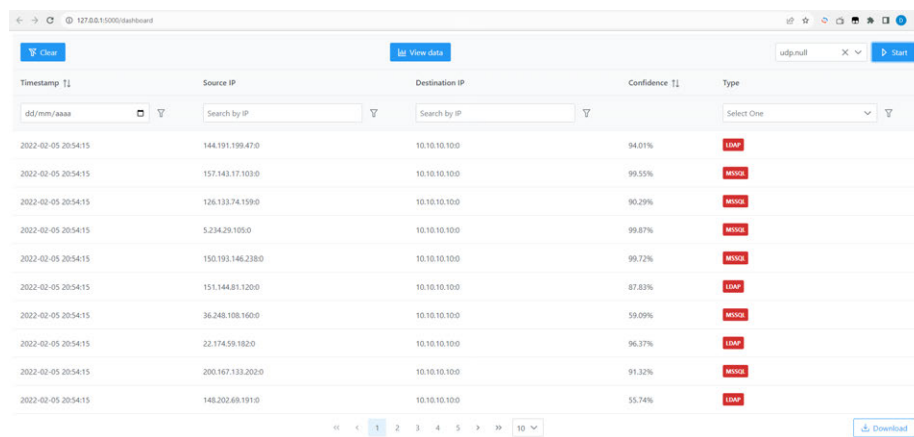
3.4.3. Interfaz web

En la interfaz web se puede ver información sobre el tráfico de red y lo que predice el modelo sobre este. Para su implementación, se ha usado `Vue.js` un framework de JavaScript diseñado para la creación de interfaces gráficas. Se ha optado por hacer una aplicación de una sola página (single page application o SPA), que son un tipo de aplicaciones web que carga una única página inicial desde el servidor y luego actualiza dinámicamente su contenido. Para la creación de algunos de los componentes se ha hecho uso de `PrimeVue`, una biblioteca de componentes de interfaz de usuario para `Vue.js`, que proporciona una variedad de componentes predefinidos y estilos listos para usar.

La interfaz tiene 2 vistas, que serán controladas por un router A.9, que son `dashboard` y `traffic-analysis`. Ambas vistas obtienen los datos necesarios a través de una petición al endpoint `get_data` del servidor, ya explicado anteriormente.

La primera, es una tabla en la que aparece el tráfico de red. En la tabla se incluye las direcciones IP tanto de origen como de destino, así como los puertos, la fecha o timestamp, la confianza con la que el modelo ha realizado la predicción y el tipo de flujo, que aparecerá de un color u otro en función de si se trata de un ataque o no. La tabla se puede ordenar por timestamp o por la confianza de la predicción, y además, se pueden filtrar los elementos por las direcciones IP y puertos, el timestamp y por el tipo de flujo.

Por otro lado, hay varios botones: abajo a la derecha hay un botón que permite descargar la tabla en formato CSV, arriba a la izquierda hay un botón que elimina los filtros que se hayan establecido, arriba a la derecha hay un desplegable y un botón que permiten iniciar el sniffer para tráfico en tiempo real o con ficheros pcap preestablecidos, y por último, arriba en el centro hay un botón para abrir la otra vista. A continuación se muestra una captura de pantalla de la vista dashboard:



The screenshot shows a web dashboard with a table of network traffic data. The table has columns for Timestamp, Source IP, Destination IP, Confidence, and Type. There are filters for each column and a 'View data' button. A 'Download' button is at the bottom right.

| Timestamp | Source IP | Destination IP | Confidence | Type |
|---------------------|-------------------|----------------|------------|------|
| 2022-02-05 20:54:15 | 144.191.199.47:0 | 10.10.10.10:0 | 94.01% | LOAF |
| 2022-02-05 20:54:15 | 157.143.17.103:0 | 10.10.10.10:0 | 99.55% | MOIS |
| 2022-02-05 20:54:15 | 126.133.74.159:0 | 10.10.10.10:0 | 90.29% | MOIS |
| 2022-02-05 20:54:15 | 5.234.29.105:0 | 10.10.10.10:0 | 99.87% | MOIS |
| 2022-02-05 20:54:15 | 150.193.146.238:0 | 10.10.10.10:0 | 99.72% | MOIS |
| 2022-02-05 20:54:15 | 151.144.81.120:0 | 10.10.10.10:0 | 87.83% | LOAF |
| 2022-02-05 20:54:15 | 36.248.108.160:0 | 10.10.10.10:0 | 59.09% | MOIS |
| 2022-02-05 20:54:15 | 22.174.59.182:0 | 10.10.10.10:0 | 96.37% | LOAF |
| 2022-02-05 20:54:15 | 200.167.133.202:0 | 10.10.10.10:0 | 91.32% | MOIS |
| 2022-02-05 20:54:15 | 148.202.69.191:0 | 10.10.10.10:0 | 55.74% | LOAF |

Figura 3.14: Vista dashboard

La otra vista es traffic-analysis. Esta vista contiene 2 gráficas que muestran la información que aparece en la tabla de la vista dashboard, pero de una forma más visual. Para la creación de las gráficas se emplea la librería Chart.js, que se integra con PrimeVue. La gráfica de a izquierda muestra la cantidad de cada tipo de flujo que se ha detectado y la gráfica de la derecha, muestra los tipos de flujo que se han detectado a lo largo del tiempo. A continuación se muestra una captura de pantalla de esta vista:

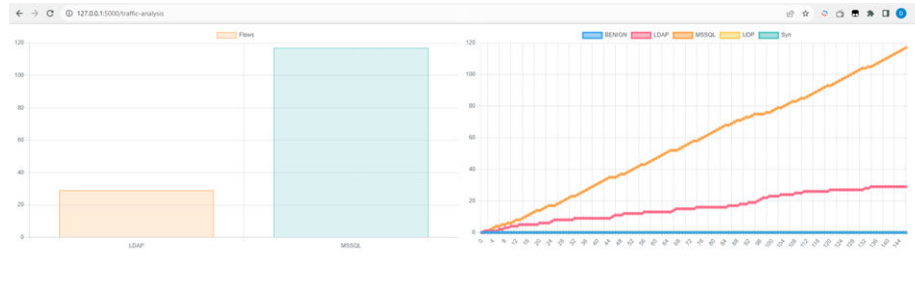


Figura 3.15: Vista traffic-analysis

Capítulo 4

Resultados

Tras terminar la implementación del sistema, es el momento de comprobar que este funciona correctamente. Para que pueda ser probado sin necesidad de instalar el servidor y sus dependencias en una máquina, se ha creado un notebook en Google Colab que despliega el servidor. De esta forma, el servidor se ejecuta en la máquina que ofrece Google Colab, y con ayuda de Ngrok, se puede acceder a la interfaz web a través de una URL. Ngrok es un servicio que crea un túnel seguro a un servidor que se ejecuta localmente y proporciona una URL de acceso público que permite a los usuarios externos acceder a su servidor. Sin esta característica, no sería posible acceder a la interfaz web del servidor que se ejecuta en el notebook. El enlace para acceder a dicho notebook es https://colab.research.google.com/github/DavidRamosArchilla/Firewall-AI/blob/main/notebooks/server_firewall_ai.ipynb

Para usar los servicios de Ngrok es necesario tener un token de autenticación. Puede ser obtenido de manera gratuita tras crear una cuenta en Ngrok en el siguiente enlace <https://dashboard.ngrok.com/get-started/your-auth-token>.

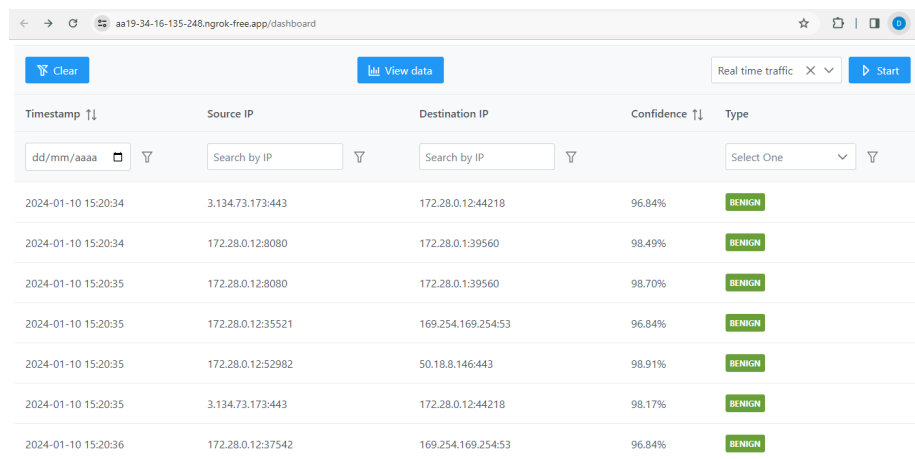
4.1. Sistema en funcionamiento

Para evaluar el rendimiento del sistema, se van a simular ataques empleando la misma herramienta que se usa para capturar el tráfico, Pyflowmeter. Esta funcionalidad es muy interesante, ya que a partir de ficheros que contienen datos sobre los paquetes de una red, se puede simular el mismo tráfico. Los ficheros con los que se van a realizar los experimentos provienen del proyecto StopDDoS (<https://github.com/StopDDoS/packet-captures>).

Por otro lado, para la ejecución del sistema se va a emplear el modelo entrenado con XGBoost, ya que, como se ha visto en secciones anteriores, obtiene mejores métricas para el conjunto de validación que la red neuronal. Para almacenar el modelo entrenado en un fichero y posteriormente cargarlo en el servidor y dejarlo listo para realizar inferencias, se ha usado Pickle, una librería para serializar y deserializar objetos de Python.

4.1.1. Tráfico en tiempo real

Para el primer experimento realizado se a ejecutado el sistema para analizar el tráfico en tiempo real. Para esta situación, viene bien crear el servidor con el notebook de Google Colab, ya que la máquina en la que se ejecuta está constantemente recibiendo y enviando paquetes (aunque en realidad Google usa contenedores). Tras dejar ejecutando el servidor un periodo de tiempo se obtienen los siguientes resultados:



| Timestamp ↑↓ | Source IP | Destination IP | Confidence ↑↓ | Type |
|---------------------|-------------------|--------------------|---------------|------------|
| dd/mm/aaaa | Search by IP | Search by IP | | Select One |
| 2024-01-10 15:20:34 | 3.134.73.173:443 | 172.28.0.12:44218 | 96.84% | BENIGN |
| 2024-01-10 15:20:34 | 172.28.0.12:8080 | 172.28.0.1:39560 | 98.49% | BENIGN |
| 2024-01-10 15:20:35 | 172.28.0.12:8080 | 172.28.0.1:39560 | 98.70% | BENIGN |
| 2024-01-10 15:20:35 | 172.28.0.12:35521 | 169.254.169.254:53 | 96.84% | BENIGN |
| 2024-01-10 15:20:35 | 172.28.0.12:52982 | 50.18.8.146:443 | 98.91% | BENIGN |
| 2024-01-10 15:20:35 | 3.134.73.173:443 | 172.28.0.12:44218 | 98.17% | BENIGN |
| 2024-01-10 15:20:36 | 172.28.0.12:37542 | 169.254.169.254:53 | 96.84% | BENIGN |

Figura 4.1: Tabla del análisis del tráfico recibido en tiempo real

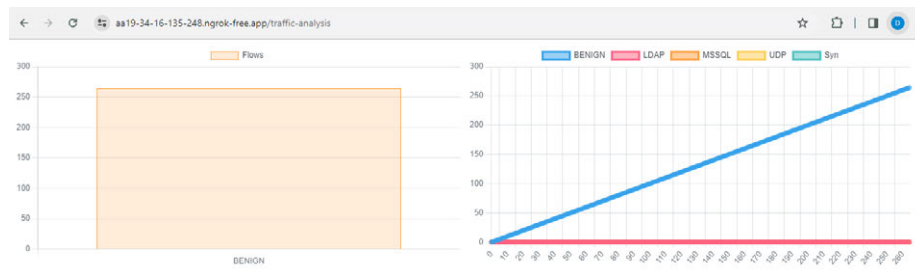
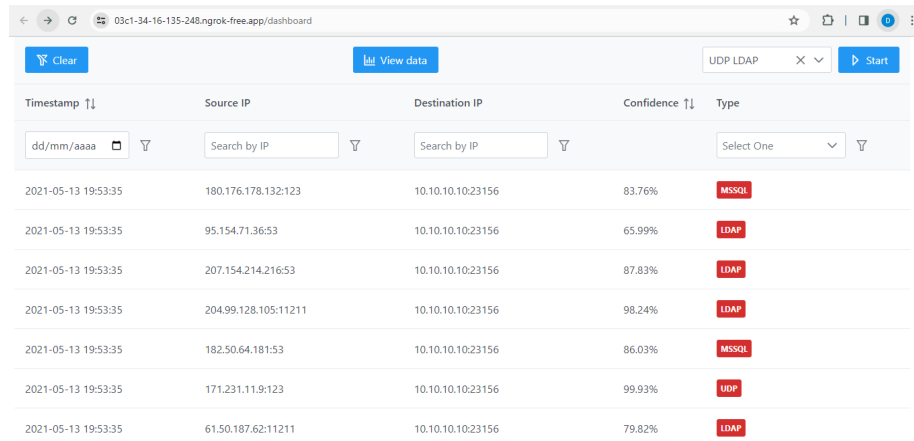


Figura 4.2: Gráficas del análisis del tráfico recibido en tiempo real

Como se puede ver, se han recibido más de 200 flujos de red y el sistema ha predicho para todos ellos que se eran benignos, es decir, que no eran ataques. Esto indica que el sistema está realizando las predicciones correctamente.

4.1.2. Ataque LDAP

Como primer ataque simulado se ha escogido un ataque con reflexión LDAP. A continuación se muestran los resultados que el sistema ofrece tras el análisis:



| Timestamp ↑↓ | Source IP | Destination IP | Confidence ↑↓ | Type |
|---------------------|----------------------|-------------------|---------------|-------|
| 2021-05-13 19:53:35 | 180.176.178.132:123 | 10.10.10.10:23156 | 83.76% | MSSQL |
| 2021-05-13 19:53:35 | 95.154.71.36:53 | 10.10.10.10:23156 | 65.99% | LDAP |
| 2021-05-13 19:53:35 | 207.154.214.216:53 | 10.10.10.10:23156 | 87.83% | LDAP |
| 2021-05-13 19:53:35 | 204.99.128.105:11211 | 10.10.10.10:23156 | 98.24% | LDAP |
| 2021-05-13 19:53:35 | 182.50.64.181:53 | 10.10.10.10:23156 | 86.03% | MSSQL |
| 2021-05-13 19:53:35 | 171.231.11.9:123 | 10.10.10.10:23156 | 99.93% | UDP |
| 2021-05-13 19:53:35 | 61.50.187.62:11211 | 10.10.10.10:23156 | 79.82% | LDAP |

Figura 4.3: Tabla del análisis de un ataque LDAP

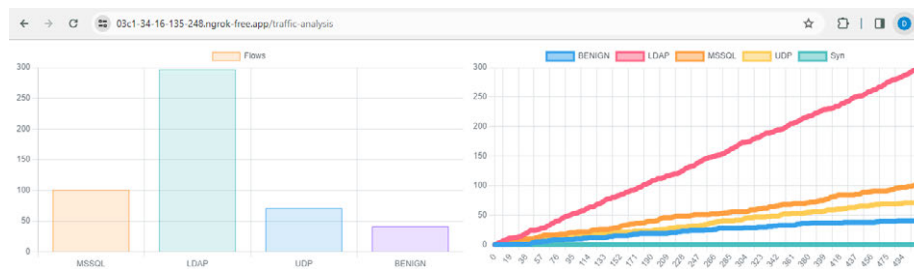
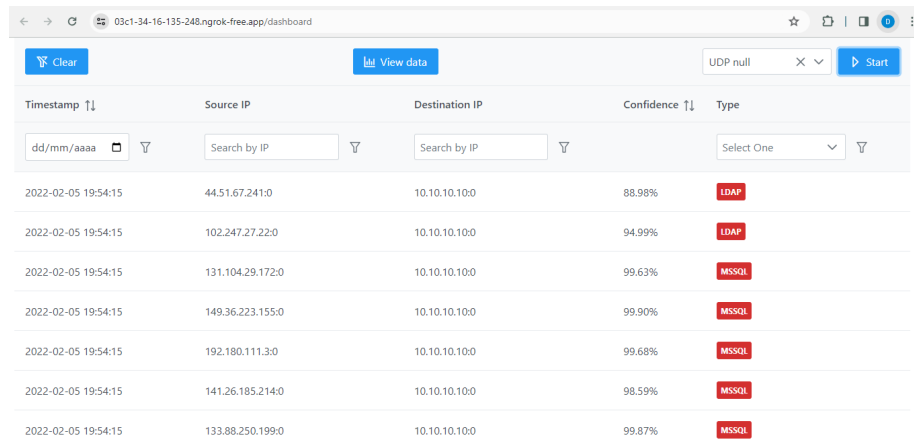


Figura 4.4: Gráficas del análisis de un ataque LDAP

Tras realizar la simulación, el sistema detecta que la gran mayoría de los flujos son de un ataque de tipo LDAP, lo que es correcto. También detecta una menor parte de los flujos como ataques MS SQL y UDP. Esto se debe a la similitud que hay entre estos tipos de ataques. Por último, hay algunos flujos que se detectan como benignos, pero esto no supone un problema ya que sobre el total de los flujos recibidos se ve claramente que el sistema es capaz de detectar el ataque.

4.1.3. Ataque UDP null

Otro ataque que se ha simulado ha sido un ataque UDP null. UDP null es un tipo de ataque que se basa en el envío de paquetes UDP vacíos a un servidor o dispositivo de red, con el objetivo de sobrecargarlo y hacer que deje de funcionar. Estos son los resultados tras la simulación de este ataque:



| Timestamp ↑↓ | Source IP | Destination IP | Confidence ↑↓ | Type |
|---------------------|------------------|----------------|---------------|-------|
| 2022-02-05 19:54:15 | 44.51.67.241:0 | 10.10.10.10:0 | 88.98% | LDAP |
| 2022-02-05 19:54:15 | 102.247.27.22:0 | 10.10.10.10:0 | 94.99% | LDAP |
| 2022-02-05 19:54:15 | 131.104.29.172:0 | 10.10.10.10:0 | 99.63% | MSSQL |
| 2022-02-05 19:54:15 | 149.36.223.155:0 | 10.10.10.10:0 | 99.90% | MSSQL |
| 2022-02-05 19:54:15 | 192.180.111.3:0 | 10.10.10.10:0 | 99.68% | MSSQL |
| 2022-02-05 19:54:15 | 141.26.185.214:0 | 10.10.10.10:0 | 98.59% | MSSQL |
| 2022-02-05 19:54:15 | 133.88.250.199:0 | 10.10.10.10:0 | 99.87% | MSSQL |

Figura 4.5: Tabla del análisis de un ataque UDP null

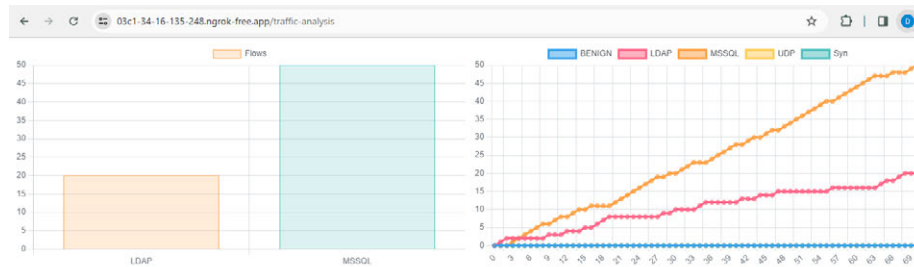
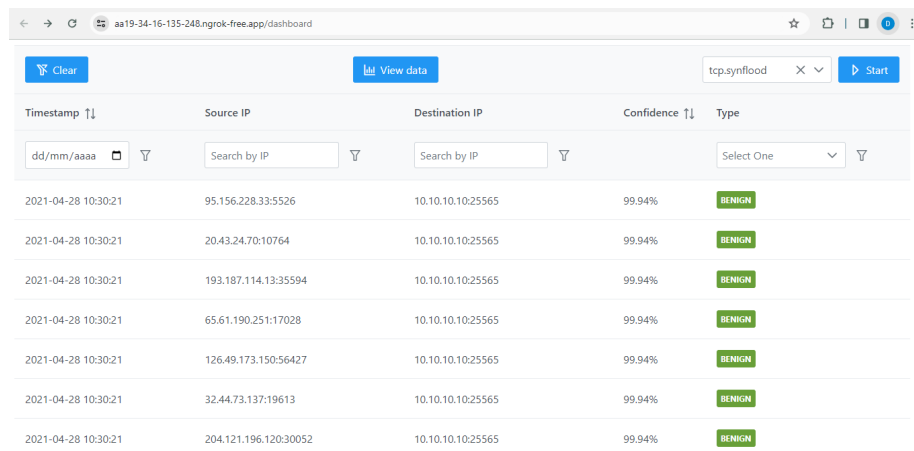


Figura 4.6: Gráficas del análisis de un ataque UDP null

En este ataque se puede observar que el sistema ha detectado que todos los flujos correspondían a un ataque. Sin embargo, las predicciones que ha realizado han sido que los flujos eran ataques de tipo LDAP y MS SQL. Esto puede deberse a que el tipo de ataque simulado no se asemeje lo suficiente a ninguno de los tipos que había en el conjunto de datos de entrenamiento y por ello el modelo se esté comportando así. De todos modos, el sistema ha sido capaz de detectar que el tráfico recibido se trataba de un ataque.

4.1.4. Ataque TCP SYN flood

A pesar de que el sistema ha sido capaz de detectar los ataques anteriores, hay algunos ataques que se le escapan por su dificultad para ser detectados. Un buen ejemplo, son los ataques sobre TCP de SYN flood, ya que los paquetes que se envían en estos ataques parecen que son legítimos. A continuación, se muestran los resultados obtenidos tras realizar este ataque:



| Timestamp ↑↓ | Source IP | Destination IP | Confidence ↑↓ | Type |
|---------------------|-----------------------|-------------------|---------------|--------|
| 2021-04-28 10:30:21 | 95.156.228.33:5526 | 10.10.10.10:25565 | 99.94% | BENIGN |
| 2021-04-28 10:30:21 | 20.43.24.70:10764 | 10.10.10.10:25565 | 99.94% | BENIGN |
| 2021-04-28 10:30:21 | 193.187.114.13:35594 | 10.10.10.10:25565 | 99.94% | BENIGN |
| 2021-04-28 10:30:21 | 65.61.190.251:17028 | 10.10.10.10:25565 | 99.94% | BENIGN |
| 2021-04-28 10:30:21 | 126.49.173.150:56427 | 10.10.10.10:25565 | 99.94% | BENIGN |
| 2021-04-28 10:30:21 | 32.44.73.137:19613 | 10.10.10.10:25565 | 99.94% | BENIGN |
| 2021-04-28 10:30:21 | 204.121.196.120:30052 | 10.10.10.10:25565 | 99.94% | BENIGN |

Figura 4.7: Tabla del análisis de un ataque TCP SYN flood

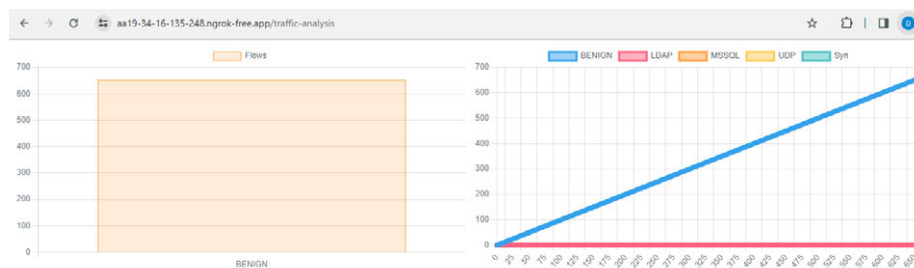


Figura 4.8: Gráficas del análisis de un ataque TCP SYN flood

Como se puede observar, el sistema no es capaz de detectar el ataque y detecta los flujos de red como benignos.

Capítulo 5

Conclusiones y trabajos futuros

5.1. Conclusiones

Este trabajo propone una solución para tener un sistema que pueda analizar en tiempo real el tráfico de red, con el fin de detectar ataques de denegación de servicio. Para lograrlo, un modelo de inteligencia artificial se encarga de realizar predicciones sobre si se está produciendo un ataque o no. Para crear el modelo, se han llevado a cabo una serie de experimentos para conseguir la mejor versión posible, llegando a obtener métricas más que satisfactorias, como una precisión del 99 %.

Tras poner el sistema a prueba llevando a cabo simulaciones de diferentes tipos de ataque, se observa que este es capaz de detectarlos en la mayoría de los casos. Sin embargo, hay algunas situaciones en las que el tráfico del ataque se asemeja a tráfico lícito de tal forma que parece indistinguible, y el sistema no es capaz de detectar el ataque. Esto indica que, a pesar de cumplir con los objetivos, aún hay margen de mejora y posiblemente sea necesario emplear otro tipo de técnicas.

Por último, hay que tener en cuenta que el dataset que se ha usado para entrenar a los modelos no es perfecto. Esto quiere decir, que ha sido generado artificialmente para este propósito, y durante su creación, se ha podido introducir algún sesgo. Evitar esto es una tarea muy compleja, y esta puede ser una explicación de por qué el sistema no detecta algunos ataques a pesar de tener una precisión tan elevada sobre el conjunto de datos de validación.

5.2. Impacto social y medioambiental

El impacto medioambiental de este trabajo podría considerarse como nulo, ya que no tiene relación con este tema. En cuanto al impacto social, el hecho de poder reducir el número de ataques DDoS es algo positivo, ya que son considerados como ataques maliciosos. Del mismo modo, quienes realizan estos ataques dejarían de beneficiarse de ello.

Por otro lado, este trabajo podría alinearse con los Objetivos de Desarrollo Sostenible (ODS) de la ONU. Para empezar, el uso de tecnologías avanzadas como el Machine Learning y la inteligencia artificial para detectar ciberataques se alinea con el ODS 9 (Industria, innovación e infraestructura), ya que fomenta la innovación en infraestructuras tecnológicas y promueve el desarrollo de sistemas de información seguros. En relación a esto, la ciberseguridad es fundamental para la estabilidad y la paz en el ciberespacio. Al tratarse este trabajo sobre la detección de ataques DDoS, se contribuye a promover la paz en el entorno digital, lo que se relaciona con el ODS 16 (Paz, justicia e instituciones

sólidas).

5.3. Lineas futuras

Aunque los objetivos establecidos para este trabajo se han cumplido, a continuación se presentan unas posibles líneas futuras de investigación:

- Aumentar la cantidad de tipos de ataques a detectar para crear un sistema más completo. A pesar que en conjunto de datos se incluían otros tipos de ataques, la cantidad de datos que hay sobre ellos no son suficientes como para poder entrenar un modelo.
- Emplear herramientas que simulen ataques reales para comprobar el correcto funcionamiento del sistema.
- Mejorar la apariencia de la aplicación web y añadir otras páginas, como por ejemplo, una página de inicio.
- Crear un sistema de usuarios para acceder a los datos de la página web. De esta forma solo los usuarios registrados en el sistema tendrían acceso a ellos. Esto no se ha llevado a cabo dado que en la arquitectura planteada el servidor se encuentra dentro de la red local, y por tanto se podría controlar con más facilidad quiénes pueden acceder a él.
- Modificar la arquitectura para que el sniffer (analizador de paquetes) y el servidor se ejecuten en máquinas diferentes. De esta forma, la máquina que ejecuta el sniffer actuaría de firewall y el servidor simplemente realizaría los cálculos para predecir si el tráfico analizado se trata de un ataque o no.

Bibliografía

- [1] O. Yoachimik and J. Pacheco, “Ddos threat report 2023 q1 (es),” Tech. Rep., Marzo 2023. [Online]. Available: <https://blog.cloudflare.com/es-es/ddos-threat-report-2023-q1-es-es/>
- [2] C. Systems, “Reporte anual cisco 2018,” Tech. Rep., Febrero 2018. [Online]. Available: https://www.cisco.com/c/dam/global/es_mx/solutions/pdf/reporte-anual-cisco-2018-espan.pdf
- [3] A. W. Services, “Threat landscape report – q1 2020,” Tech. Rep., Mayo 2020. [Online]. Available: <https://aws-shield-tlr.s3.amazonaws.com/2020-Q1-AWS-Shield-TLR.pdf>
- [4] G. Cloud, “How google cloud blocked the largest layer 7 ddos attack at 46 million rps,” Tech. Rep., Junio 2022. [Online]. Available: <https://cloud.google.com/blog/products/identity-security/how-google-cloud-blocked-largest-layer-7-ddos-attack-at-46-million-rps>
- [5] Z.-H. Zhou, *Machine learning*. Springer Nature, 2021.
- [6] C. Kingsford and S. L. Salzberg, “What are decision trees?” *Nature biotechnology*, vol. 26, no. 9, pp. 1011–1013, 2008.
- [7] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.
- [8] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [9] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 684–10 695.
- [10] Wikipedia contributors, “Llm (modelo grande de lenguaje),” 2023, [Online; accessed 2-June-2023]. [Online]. Available: [https://es.wikipedia.org/wiki/LLM_\(modelo_grande_de_lenguaje\)](https://es.wikipedia.org/wiki/LLM_(modelo_grande_de_lenguaje))
- [11] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv preprint arXiv:1511.08458*, 2015.
- [12] A. Sherstinsky, “Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network,” *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.
- [13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.
- [14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

- [15] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, p. 357–362, 2020.
- [16] W. McKinney *et al.*, “Data structures for statistical computing in python,” in *Proceedings of the 9th Python in Science Conference*, vol. 445. Austin, TX, 2010, pp. 51–56.
- [17] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.
- [18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: <https://www.tensorflow.org/>
- [19] F. Chollet *et al.*, “Keras,” <https://keras.io>, 2015.
- [20] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’16. ACM, Aug. 2016. [Online]. Available: <http://dx.doi.org/10.1145/2939672.2939785>
- [21] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” 2019.
- [22] HoloViz, “HoloViews,” <https://holoviews.org/>.
- [23] —, “HvPlot,” <https://hvplot.holoviz.org/>.
- [24] “Vue.js,” 2014. [Online]. Available: <https://vuejs.org/>
- [25] M. Grinberg, *Flask web development: developing web applications with python*. O’Reilly Media, Inc., 2018.
- [26] G. Van Rossum, *The Python Library Reference, release 3.8.2*. Python Software Foundation, 2020.
- [27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, “Scikit-learn: Machine learning in python,” *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.
- [28] Google, “Google Colab,” <https://colab.research.google.com/>, 2018.
- [29] A. Shreve, “Ngrok,” <https://ngrok.com/>, 2012.

- [30] J. Shroff, R. Walambe, S. K. Singh, and K. Kotecha, “Enhanced security against volumetric ddos attacks using adversarial machine learning,” *Wireless Communications and Mobile Computing*, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:247408815>
- [31] N. Bindra and M. Sood, “Detecting ddos attacks using machine learning techniques and contemporary intrusion detection dataset,” *Automatic Control and Computer Sciences*, vol. 53, pp. 419 – 428, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:208086076>
- [32] M. M. Saeed, H. N. R. Mohammed, O. A. H. Gazem, R. A. Saeed, H. M. A. Morei, A. E. T. Eidah, A. S. A. Gaid, A. S. A. S. Al-Uosfi, and M. G. Q. Al-Madhagi, “Machine learning techniques for detecting ddos attacks,” *2023 3rd International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, pp. 1–6, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:264881735>
- [33] M. N. Faiz, O. Somantri, A. R. Supriyono, and A. W. Muhammad, “Impact of feature selection methods on machine learning-based for detecting ddos attacks : Literature review,” *JOURNAL OF INFORMATICS AND TELECOMMUNICATION ENGINEERING*, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:246802887>
- [34] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, “Developing realistic distributed denial of service (ddos) attack dataset and taxonomy,” in *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2019, pp. 1–8.
- [35] A. H. Lashkari, Y. Zang, G. Owhuo, M. Mamun, and G. Gil, “Cicflowmeter,” *GitHub*. [vid. 2021-08-10]. Dostupné z: <https://github.com/ahlashkari/CICFlowMeter/blob/master/ReadMe.txt>, 2017.
- [36] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne.” *Journal of machine learning research*, vol. 9, no. 11, 2008.
- [37] J. H. Friedman, “Greedy function approximation: a gradient boosting machine,” *Annals of statistics*, pp. 1189–1232, 2001.
- [38] H. Le, “CICFlowMeter,” <https://gitlab.com/hieulw/cicflowmeter>, 2021.
- [39] D. Ramos Archilla, “PyFlowmeter,” <https://pypi.org/project/pyflowmeter/>, 2023.

Anexos

Apéndice A

Código fuente del proyecto

A.1. preprocesado_dataset.py

```
import pandas as pd
import numpy as np

import os
from pathlib import Path

WORKING_DIR = Path.cwd()
DATA_UNPROCESSED = WORKING_DIR / "03-11"
DATA_PROCESSED = WORKING_DIR / "processed_files"

def transform_column_types(df):
    # Convert any float columns to float32 format
    df = df.apply(lambda x: x.astype("float32") if np.issubdtype(x.dtype, np.floating) else x)

    # Convert any int columns to int32 format
    df = df.apply(lambda x: x.astype("int32") if np.issubdtype(x.dtype, np.integer) else x)

    return df

def prerprocess_dataframe(df):
    start_mem_usg = df.memory_usage().sum() / 1024**2
    print("Memory usage of properties dataframe is :",start_mem_usg," MB")
    df.dropna(inplace=True)
    df.drop_duplicates(inplace=True)
    df = transform_column_types(df)
    final_mem_usg = df.memory_usage().sum() / 1024**2
    print("Memory usage of properties dataframe is :",final_mem_usg," MB")
    return df

def load_file(file_path):
    df = pd.read_csv(file_path,
                     usecols=lambda column: column not in columns_to_exclude)
    return prerprocess_dataframe(df)

columns_to_exclude = ['Unnamed: 0', 'Flow ID', ' Source IP', ' Source Port',
                      ' Destination IP', ' Destination Port', ' Timestamp', 'SimillarHTTP']

for file in os.listdir(DATA_UNPROCESSED):
    if file != 'MSSQL.csv' and file.endswith('.csv'):
        df = load_file(DATA_UNPROCESSED / file)
        basename = file.split('.')[0]
        df.to_parquet(DATA_PROCESSED / f"{basename}.parquet")
        print(file, '\n', df[' Label'].value_counts())
```

A.2. red_neuronal.py

```

from sklearn.metrics import classification_report
from sklearn.model_selection import StratifiedKFold
from sklearn.utils.class_weight import compute_class_weight

import tensorflow as tf
from tensorflow.keras import layers, Model
from tensorflow.keras.optimizers import AdamW
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras import metrics

import numpy as np

N_FEATURES = X_train.shape[1]
N_CLASSES = len(label_encoder.classes_)

def create_model(n_features, n_classes):
    input = layers.Input(shape=(n_features, ))
    x = layers.Dense(64, activation='relu')(input)
    x = layers.Dense(64, activation='relu')(x)
    out = layers.Dense(n_classes, activation='softmax')(x)

    model = Model(inputs=input, outputs=out)

    learning_rate = 5e-4
    training_metrics = [
        metrics.Precision(name='Precision'),
        metrics.Recall(name='Recall'),
        metrics.F1Score(name='F1Score')
    ]

    model.compile(
        optimizer=AdamW(learning_rate),
        loss=CategoricalCrossentropy(),
        metrics=training_metrics
    )
    return model

bs = 32
epochs = 15

skf = StratifiedKFold(n_splits=5)
fold_idx = 0
for train_index, test_index in skf.split(X_train, y_train):
    print(f"Fold {fold_idx + 1}:")
    X_train_fold, X_test_fold = X_train[train_index, :], X_train[test_index, :]
    y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]

    class_weights = compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
    class_weights = dict(enumerate(class_weights))
    model = create_model(N_FEATURES, N_CLASSES)
    history = model.fit(
        X_train_fold,
        tf.keras.utils.to_categorical(y_train_fold),
        validation_data=(X_test_fold, tf.keras.utils.to_categorical(y_test_fold)),
        batch_size=bs,
        class_weight=class_weights,
        epochs=epochs,
        verbose=0
    )

```

```

model.save(f"model_{fold_idx}.keras")
plt.figure()
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title(f'Fold {fold_idx + 1} Loss')
plt.legend()
plt.show()

y_pred = model.predict(X_test_fold)
report = classification_report(y_test_fold,
                              np.argmax(y_pred, axis=1),
                              target_names=label_encoder.classes_)

print(report)
fold_idx += 1

```

A.3. busqueda_hiperparametros.py

```

import optuna
import xgboost as xgb
from sklearn.model_selection import cross_val_score

def objective(trial):
    params = {
        'objective': 'multi:softmax', # Multiclass classification
        'eval_metric': 'mlogloss',
        'device': 'gpu',
        'lambda': trial.suggest_float('lambda', 1e-8, 1.0, log=True),
        'alpha': trial.suggest_float('alpha', 1e-8, 1.0, log=True),
        'max_depth': trial.suggest_int('max_depth', 3, 9),
        'eta': trial.suggest_float('eta', 1e-8, 1.0, log=True),
        'gamma': trial.suggest_float('gamma', 1e-8, 1.0, log=True),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.1, 1.0),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 10),
        'n_estimators': trial.suggest_int('n_estimators', 100, 1300),
        'num_class': len(set(Y)), # Number of classes
    }

    model = xgb.XGBClassifier(**params)
    model.fit(X_train, y_train)

    scores = cross_val_score(model, X_val, y_val, cv=5, scoring='accuracy')

    return scores.mean()

study = optuna.create_study(direction='maximize', study_name='Classifier')
study.optimize(objective, n_trials=100, show_progress_bar=True)

best_params = study.best_params
best_accuracy = study.best_value
print(f"Best Hyperparameters: {best_params}")
print(f"Best Accuracy: {best_accuracy}")

```

A.4. pcap_to_csv_pyflowmeter.py

```
from pyflowmeter.sniffer import create_sniffer

sniffer = create_sniffer(
    input_file='path_to_the_file.pcap',
    to_csv=True,
    output_file='./flows_test.csv',
)

sniffer.start()
try:
    sniffer.join()
except KeyboardInterrupt:
    print('Stopping the sniffer')
    sniffer.stop()
finally:
    sniffer.join()
```

A.5. enviar_trafico_real_a_servidor.py

```
from pyflowmeter.sniffer import create_sniffer

sniffer = create_sniffer(
    server_endpoint='http://127.0.0.1:5000/send_traffic',
    verbose=True,
    sending_interval=5
)

sniffer.start()
try:
    sniffer.join()
except KeyboardInterrupt:
    print('Stopping the sniffer')
    sniffer.stop()
finally:
    sniffer.join()
```

A.6. simular_trafico_pyflowmeter.py

```
from pyflowmeter.sniffer import create_sniffer

sniffer = create_sniffer(
    input_file='path_to_the_file.pcap',
    server_endpoint='http://127.0.0.1:5000/send_traffic',
)

sniffer.start()
try:
    sniffer.join()
except KeyboardInterrupt:
    print('Stopping the sniffer')
    sniffer.stop()
finally:
    sniffer.join()
```


A.7. endpoint_iniciar_sniffer.py

```

@app.route('/start_sniffer', methods=['POST'])
def start_sniffer():
    if request.is_json:
        data = request.get_json()
        test_type = data['file']
        test_file = TYPES_DICT[test_type]
        reload_sniffer(test_file)
        return jsonify({"message": "Data received successfully"}), 200
    else:
        return jsonify({"error": "Invalid JSON data in the request"}), 400

def reload_sniffer(test_file):
    global traffic_sniffer
    global sniffer_created
    global predicted_data
    if sniffer_created:
        try:
            traffic_sniffer.stop()
            traffic_sniffer.join()
        except:
            pass
    else:
        sniffer_created = True

    predicted_data = []
    if test_file == 'Real time traffic':
        traffic_sniffer = sniffer.create_sniffer(
            input_interface='eth0',
            server_endpoint='http://127.0.0.1:5000/send_traffic',
        )
    else:
        traffic_sniffer = sniffer.create_sniffer(
            input_file=test_file,
            server_endpoint='http://127.0.0.1:5000/send_traffic',
        )
    traffic_sniffer.start()

```

A.8. send_traffic_y_get_data.py

```

@app.route("/send_traffic", methods=["POST"])
def post_data():
    if request.is_json:
        data = request.get_json()
        confidences, predicted_classes = model.predict(data["flows"])
        for (flow, confidence, predicted_class) in zip(
            data["flows"], confidences, predicted_classes
        ):

            predicted_data.append(
                {
                    "type": predicted_class,
                    "src_ip": f'{flow["src_ip"]}:{flow["src_port"]}',
                    "dst_ip": f'{flow["dst_ip"]}:{flow["dst_port"]}',
                    "confidence": f'{confidence:.2%}',
                    "timestamp": flow["timestamp"],
                }
            )

        print(model.predict(data["flows"]))
        return jsonify({"message": "Data received successfully"}), 200
    else:
        return jsonify({"error": "Invalid JSON data in the request"}), 400

@app.route("/get_data", methods=["GET"])
def get_data():
    return jsonify(predicted_data)

```

A.9. router.js

```

import { createRouter, createWebHistory } from 'vue-router'
import Dashboard from '../components/Dashboard.vue'
import TrafficAnalysis from '../components/TrafficAnalysis.vue'

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/dashboard',
      name: 'dashboard',
      component: Dashboard
    },
    {
      path: '/traffic-analysis',
      name: 'traffic-analysis',
      component: TrafficAnalysis
    }
  ]
})

export default router

```

A.10. firewall_model.py

```

from joblib import load
import numpy as np
import pickle
import tensorflow as tf

class FirewallModel:

    TF_MODEL_FILE_PATH = "./ML_models/model.keras"
    XGB_MODEL_FILE_PATH = "./ML_models/xgboost_model.pkl"
    SCALER_PATH = "./ML_models/std_scaler.bin"
    LABEL_ENCODER_PATH = "./ML_models/label_encoder.bin"
    COLUMNS_ORDER = [
        "protocol",
        "flow_duration",
        "tot_fwd_pkts",
        "tot_bwd_pkts",
        ... # Omitido para la memoria
        "idle_min",
    ]

    def __init__(self, use_xgboost=True):
        self.use_xgboost = use_xgboost
        self.model = self.load_model()
        self.scaler = self.load_scaler()
        self.label_encoder = self.load_label_encoder()

    def load_model(self):
        if self.use_xgboost:
            with open(FirewallModel.XGB_MODEL_FILE_PATH, "rb") as model_file:
                return pickle.load(model_file)
        else:
            return tf.keras.models.load_model(FirewallModel.TF_MODEL_FILE_PATH)

    def load_scaler(self):
        return load(FirewallModel.SCALER_PATH)

    def load_label_encoder(self):
        return load(FirewallModel.LABEL_ENCODER_PATH)

    # devuelve la clase y la probabilidad (confianza en la prediccion)
    def predict(self, data):
        prepared_data = self.prepare_data(data)
        print(prepared_data.shape)
        if self.use_xgboost:
            preds = self.model.predict_proba(prepared_data)
        else:
            preds = self.model.predict(prepared_data)
        return preds[
            np.arange(preds.shape[0]), predicted_classes
        ], self.label_encoder.inverse_transform(predicted_classes)

    def prepare_data(self, data):
        final_data = []
        for flow in data:
            features_list = [flow[feature] for feature in FirewallModel.COLUMNS_ORDER]
            final_data.append(features_list)
        return self.scaler.transform(np.array(final_data))

```