

UNIVERSIDAD POLITECNICA DE MADRID



Proyecto de fin de grado

Grado en Ingeniería de Computadores

Detección de ataques DDoS mediante el uso de una red neuronal

Autor:

Adrián García Bartolomé

Tutores:

Borja Bordel Sánchez

Escuela técnica superior de ingeniería de sistemas informáticos

Agradecimientos

Ante todo, me gustaría agradecer a mi tutor Borja Bordel por ayudarme a lo largo de todos estos meses en la realización del proyecto de fin de grado, atendiendo siempre a cualquier duda que me ha surgido a lo largo de todo este tiempo en y aconsejándome en todo momento para poder llegar a terminarlo.

También le agradezco a mi familia, y especialmente a mi madre y a mi abuela, por todo lo que han hecho por mí para llegar hasta aquí.

Resumen

En la era de las tecnologías avanzadas, los ataques DDoS han ganado en accesibilidad y prevalencia. La capacidad de identificar estos ataques se ha vuelto vital, ya que, aunque muchos no logran causar daños graves o robar información, los que lo hacen pueden generar daños significativos en la infraestructura y poner en peligro la seguridad de las personas, especialmente si afectan a equipos críticos.

El aumento exponencial de dispositivos IoT en nuestra sociedad actual ha incrementado considerablemente el número de posibles víctimas de estos ataques. La conectividad extendida de estos dispositivos los hace vulnerables y, a su vez, incrementa el riesgo de que los sistemas sean deshabilitados o dañados.

A pesar de los avances en ciberseguridad, aún no estamos completamente equipados para detectar todos los ataques DDoS. Este trabajo se enfocará en analizar estos ataques y propondrá una solución para su detección mediante el uso de herramientas de aprendizaje automático, con el objetivo de mejorar la seguridad y la resiliencia de los sistemas conectados.

Acknowledges

With the increasing access to advanced technologies, it has become progressively easier to perpetrate a DDoS attack. Therefore, the task of detecting when an attack of this nature is being carried out against a system has become increasingly crucial.

Although this type of attack can be underestimated at first, as many do not succeed in damaging a machine or they do not aim to steal data or information, these attacks, if carried out under the right circumstances, can cause damage to the infrastructure. This can even put people at risk if certain critical equipment is damaged.

This is a serious problem considering the context of our current society, where the use of IoT devices has significantly increased. Consequently, there are many devices connected to the network and thus many potential victims of these attacks.

Unfortunately, we are still not ready to detect these attacks on all occasions, and therefore, as I have already mentioned, systems can be disabled or damaged by these attacks. This paper will address these attacks and attempt to provide a solution for their detection using machine learning tools.

Índice

Introducción	10
1.1. Contexto	10
1.2. Objetivos	11
1.2.1. Objetivo general.....	11
1.2.2. Objetivos específicos	11
1.3. Motivación y justificación	12
1.4. Estructura del documento	12
Marco teórico	14
2.1. Contexto	14
2.2. Términos de Machine Learning	15
2.3. Términos de ciberseguridad.....	16
Estado del arte	19
3.1. Introducción	19
3.2. Técnicas para el preprocesamiento	19
3.3. Técnicas de machine learning.....	20
3.3.1. Redes neuronales	20
3.3.2. Redes Neuronales Feedforward.....	21
3.3.3. Neuronas artificiales	22
3.3.4. Random forest.....	24
3.4. Técnicas de ciberseguridad	26
3.4.1. Ataques DDoS	26
3.4.2. Ataques TCP	28

3.4.3. Ataques UDP	29
Desarrollo de la aplicación.....	31
4.1. Arquitectura usada	31
4.2. Elección del dataset	32
4.3. Preprocesado	34
4.3.1. Modificación inicial.....	34
4.4.2. Limpieza de datos	37
4.5. Entrenamiento del modelo	39
4.5.1. Elección de la red neuronal	40
4.5.2. Creación de la red neuronal	40
4.5.3. Entrenamiento de la red neuronal	43
4.5.4. Conclusiones de la red neuronal.....	46
4.5.5. Random Forest.....	47
4.6. Análisis de paquetes en tiempo real.....	48
4.6.1. Adaptación de la red neuronal	48
4.6.2. Creación de un dataset en tiempo real	49
4.6.3. Notificación por web	55
Pruebas y resultados.....	58
5.1. Pruebas.....	58
5.1.1. Prueba del sistema de detección tiempo real	58
5.2. Resultados	61
Conclusiones y trabajos futuros	63
6.1. Conclusiones	63
6.1.1. Introducción.....	63

6.2. Impacto social y medioambiental	63
6.3. Trabajo futuro	64
Bibliografía	66

Índice de imágenes

Ilustración 1:Estructura de una red neuronal	22
Ilustración 2: La neurona artificial. (s. f.). https://www.ibiblio.org/pub/linux/docs/LuCaS/Presentaciones/200304curso-glisa/redes_neuronales/curso-glisa-redes_neuronales-html/x38.html	23
Ilustración 3: Función sigmoïdal	24
Ilustración 4: Función ReLU	24
Ilustración 5: Coreun. (2020, 10 junio). DDoS-Ataques denegación de servicio (parte I). COREUN. https://coreun.com/2020/06/10/ddos-ataques-denegacion-de-servicio-parte-i/	27
Ilustración 6: Maldonado, D. (2022, 16 abril). ¿Qué es una inundación TCP SYN Flood? Daniel Maldonado. https://danielmaldonado.com.ar/seguridad/que-es-una-inundacion-tcp-syn-flood/	29
Ilustración 7:Arquitectura del programa.....	32
Ilustración 8:Distribución del dataset	33
Ilustración 9: Unión del dataset	33
Ilustración 10:División de frame.protocols	35
Ilustración 11:Dataset imprimido	35
Ilustración 12:Escalada de las columnas numericas	36
Ilustración 13:Código para obtener la importancia de las columnas	38
Ilustración 14:Columnas más importantes.....	38
Ilustración 15:Columnas menos importantes.....	39
Ilustración 16:Limpieza de columnas	39
Ilustración 17:Función de activación ReLU	41
Ilustración 18: Función de activación sigmoïdal	42
Ilustración 19:Entrenamiento red neuronal.....	43
Ilustración 20:Evolución de la certeza del entrenamiento y validación	44

Ilustración 21:Resultados del entrenamiento de la red neuronal	45
Ilustración 22:Resultados del entrenamiento de Random Forest	47
Ilustración 23:Matriz de confusión de Random Forest.....	47
Ilustración 24:Tabla de los valores a modificar	48
Ilustración 25:Grafica de la evolución de la precisión del entrenamiento y la validación	49
Ilustración 26:Resultados del entrenamiento de la red neuronal modificada	49
Ilustración 27:Obtención de ip, paquete tcp y udp	50
Ilustración 28:Inicialización de valores	50
Ilustración 29:Obtención del tiempo tcp.....	50
Ilustración 30: Valores no relacionados con tcp ni udp.....	51
Ilustración 31:Obtencion de valores relacionados con udp	51
Ilustración 32: Añadir valores al dataframe.....	51
Ilustración 33:Exportación del csv.....	52
Ilustración 34:Resultados del dataset I	52
Ilustración 35:Resultados del dataset II	52
Ilustración 36:Guardar el modelo	53
Ilustración 37:Carga del modelo	54
Ilustración 38:Deteccción del ataque.....	54
Ilustración 39:Ejecucion del programa	55
Ilustración 40:Creación de página web.....	56
Ilustración 41:Threading para el programa y la web	56
Ilustración 42:Inicialización de proyecto node.js	59
Ilustración 43:Servidor node.js	59
Ilustración 44:Servidor arrancando.....	59
Ilustración 45:Monitorización del servidor.....	60
Ilustración 46:Ataque tumba el servidor.....	60

Capítulo 1

Introducción

1.1. Contexto

En nuestra era actual ha crecido exponencialmente el uso de dispositivos o equipos vinculados a la red, como son los dispositivos IoT, estos son dispositivos que cada vez un mayor número de personas utiliza a diario y estos tratan cantidades muy elevadas de información continuamente, información que puede ser privada o tener un carácter de alta importancia. Un problema clave es la protección de estos datos y para ello también hay que proteger estos dispositivos y los sistemas que reciben la información de estos dispositivos.

Además, este problema no es único del mundo de la informática si no que es aplicado a todos los sectores ya que en todos los sectores existe una creciente demanda relacionada con dispositivos informáticos o dispositivos IoT por lo tanto todos estos sectores necesitan protecciones para que sus datos no sean interceptados o sus sistemas informáticos dejados fuera de uso.

Por tanto, debido a la alta importancia que pueden tener estos datos los dispositivos y sistemas que los manejan se convierten en diana de ataques para inutilizarlos y gracias a ello obtener esta información o simplemente dañar los sistemas que los manejan, aquí es donde entran los ataques DDoS, que consisten en ataques en los que se intenta sobrecargar un sistema mediante su saturación haciendo que no pueda prestar servicio de forma normal o directamente tumbarlo.

El problema de los ataques DDoS no recae directamente en el hecho de que puedan tumbar o dañar a un equipo, si no que recae en la facilidad y la frecuencia de

sus ataques, ya que es un tipo de ataque que si se tienen los suficientes recursos se puede realizar de forma continua y dañina, además de que su capacidad de ignorar firewalls tradicionales está aumentando preocupantemente.

El método a través del cual se perpetran estos ataques es mediante el envío masivo de paquetes que parecen inofensivos en principio, pero al acumularse en un corto periodo de tiempo se produce una sobrecarga por parte del equipo, la cual lleva a la caída o el mal funcionamiento del equipo.

Aunque existen ataques capaces de incapacitar sistemas, existen herramientas que permiten su prevención y detección, gracias al auge de las inteligencias artificiales se han creado nuevas herramientas para identificar los patrones existentes en estos ataques para detectar cuando se realizan distintos ataques, actualmente estos campos de estudio están en auge por el interés de asegurar la seguridad de los datos.

En este trabajo de fin de grado desarrollaré una red neuronal que clasifique el tráfico de red en **ataque** y **no ataque** tras ello utilizando unas máquinas virtuales para hacer un ataque de una a otra comprobare si funciona o no esta red neuronal.

1.2. Objetivos

1.2.1. Objetivo general

El objetivo final de este trabajo es el de entrenar una red neuronal para que mediante la obtención de trafico de red se compruebe si esta red está o no recibiendo un ataque para ello se realizará el preprocesamiento de un dataset y este se modificará para que mantenga los datos esenciales y los datos se encuentren en perfecto estado para que la red neuronal pueda entrenarse correctamente, además del entrenamiento de una red neuronal.

1.2.2. Objetivos específicos

También hay una serie de objetivos más específicos que se buscan con este proyecto:

- Analizar los datos del dataset para conocer cuáles son los parámetros que más influyen en la detección de ataques DDoS y su optimización para su posterior uso en la creación de una red neuronal.
- Creación y optimización de una red neuronal que se encargue de predecir si la red es o no tráfico malicioso de tipo DDoS y ofrezca unos resultados notables.
- Realizar un estudio de los datos obtenidos de la red neuronal para poder detectar si esta ha sido entrenada de forma correcta.
- Detección de paquetes en tiempo real y el análisis de estos para saber si la red neuronal funciona en tiempo real.

1.3. Motivación y justificación

Una vez que mi tutor Borja me ofreció este tema para hacer el trabajo me pareció un tema bastante interesante ya que el uso de machine learning para abordar un problema real como son los ataques DDoS puede ser muy útil para crear una herramienta que tenga utilidad en el mundo actual, además una vez investigué sobre el tema mi interés sobre el creció ya que es un problema que se sigue abordando a día de hoy, donde aplicar conocimientos adquiridos a lo largo de la carrera en el que es posible que trabaje más adelante en mi carrera profesional.

1.4. Estructura del documento

En este apartado se explicará la distribución del documento, el cual est compuesto por los siguientes capítulos:

- Capítulo 1: Donde se explican los objetivos del proyecto y se da un breve resumen de lo que se hará.
- Capítulo 2: En este se explicarán una serie de conceptos importantes a la hora de comprender el proyecto.
- Capítulo 3: Aquí se entrará en detalle sobre las herramientas utilizadas para el desarrollo del trabajo.

- Capítulo 4: Este capítulo es el desarrollo del trabajo y sus resultados.
- Capítulo 5: Dentro de este capítulo se pueden observar los cambios posibles para la mejora del proyecto a futuro y conclusiones.

Capítulo 2

Marco teórico

2.1. Contexto

El avance tecnológico ha llevado a las empresas a adaptarse rápidamente implementando sistemas informáticos y haciendo uso de Internet para mantener su competitividad en el mercado. En la actualidad, los dispositivos IoT son fundamentales para estos sistemas, permitiendo la conexión de dispositivos y proporcionando una conexión rápida con capacidades de baja latencia y alta fiabilidad.

Esta integración acelerada de nuevas tecnologías ha creado brechas de seguridad aprovechadas por quienes tienen conocimientos informáticos para cometer actos fraudulentos, comprometiendo la privacidad de los usuarios y exponiéndolos a diversas amenazas.

Ante esta situación, ha surgido la necesidad de desarrollar medidas de ciberseguridad para proteger estos sistemas y garantizar la privacidad de los usuarios. La ciberseguridad se ha convertido en un campo crucial, enfocado en identificar y mitigar riesgos, así como en establecer políticas y procedimientos que protejan la información y los sistemas de las empresas frente a posibles ataques cibernéticos. Es fundamental invertir en la formación de profesionales capacitados en ciberseguridad y en el desarrollo de tecnologías y protocolos de seguridad robustos que puedan hacer frente a las constantes amenazas en un entorno digital en constante evolución.

En este contexto, el Proyecto de Fin de Grado abordará el problema de la detección y prevención de ataques que desestabilizan o deshabilitan uno o varios servicios de un sistema temporalmente, también llamados Ataques de Denegación de Servicio. La solución propuesta para este problema hace uso del Machine Learning.

En este apartado se introducirá conceptos surgidos en este campo de la inteligencia artificial que se requerirán para entender los siguientes apartados del trabajo.

2.2. Términos de Machine Learning

Dentro del campo de la Inteligencia Artificial (IA), se encuentra el Machine Learning. La definición formal de Machine Learning es la siguiente:

Definición 1: Machine Learning *Machine learning es una rama de la inteligencia artificial que se centra en el uso de datos y algoritmos para imitar la forma en la que funcionan los seres humanos, con el objetivo de crear una mejora en su precisión.*

La principal utilidad del Machine Learning, consiste en el uso de algoritmos capaces de aprender y mejorar a través del procesamiento de los datos que reciben, proporcionando una salida determinada en base a estos últimos.

Hay tres formas de aprendizaje, dos más comunes, el aprendizaje supervisado y no supervisado, y una menos común, el aprendizaje reforzado. El aprendizaje que se utilizará a lo largo de este trabajo es el supervisado

Definición 2: Aprendizaje Supervisado *El aprendizaje supervisado, también conocido como machine learning supervisado, es una subcategoría de machine learning y la inteligencia artificial. Se define por su uso de conjuntos de datos etiquetados para entrenar algoritmos que clasifican datos o prevén resultados con precisión. (¿Qué Es el Aprendizaje Supervisado? | IBM, s. f.)*

En el aprendizaje supervisado, el objetivo es predecir el resultado de una entrada. Esto se logra mediante el análisis de los datos proporcionados junto con datos

que han servido como entrenamiento. El aprendizaje supervisado consta de dos fases: entrenamiento o análisis, donde el algoritmo aprende patrones o métodos para la clasificación utilizando las etiquetas de los datos, y la segunda fase, pruebas o resultados, donde se verifica si el algoritmo realmente funciona y ha aprendido.

Dentro de los distintos tipos de aprendizaje hay una serie de herramientas que permiten realizar dicho aprendizaje, algunas concretas para un tipo de aprendizaje y otras como la que usaremos que valen para varios, siendo esta las redes neuronales.

Definición 3: Redes Neuronales *Una red neuronal es un método de la inteligencia artificial que enseña a las computadoras a procesar datos de forma inspirada en cómo lo hace el cerebro humano. (¿Qué Es una Red Neuronal? - Explicación de las Redes Neuronales Artificiales - AWS, s. f.*

Las redes neuronales son un tipo de algoritmo de Machine Learning que se inspira en la estructura y funcionamiento del cerebro humano. Se basa en la existencia de una serie de neuronas interconectadas entre ellas y trabajan entre ellas para procesar información. Estas neuronas están organizadas en capas, estas varían dependiendo del tipo de red neuronal que se implemente. Estas capas incluyen una capa de entrada, una o varias capas ocultas y una capa de salida. Cada neurona en el proceso de entrenamiento realiza una serie de operaciones matemáticas en los datos de entrada y transmitiendo el resultado a las neuronas de la siguiente capa. Las redes neuronales son efectivas en un amplio abanico de problemas, como el reconocimiento de patrones o la clasificación.

2.3. Términos de ciberseguridad

Dentro de la ciberseguridad en este proyecto el tema principal que se va a tratar es el de los ataques DDoS, además del tráfico de paquetes, concretamente UDP y TCP.

Definición 4: Ataque de Denegación de Servicio (DoS) *Un ataque de denegación de servicio es un intento malicioso de hacer que un servicio o recurso en línea sea inaccesible para los usuarios.*

Este tipo de ataques se logra abrumando el servicio objetivo con una gran cantidad de tráfico dirigido hacia él, esto intenta agotar los recursos disponibles, como ancho de banda, capacidad de procesamiento o memoria.

El resultado de este tipo de ataques es conseguir que el servicio se vuelva inaccesible para los usuarios o inhabilitar un equipo, ya que este no puede responder a sus solicitudes, aunque estos ataques pueden no llegar a perpetrar su objetivo.

Estos ataques pueden lanzarse desde una sola máquina o desde múltiples máquinas coordinadas, y afectar a la fuente que ofrece el servicio y a la infraestructura de red.

Definición 5: Tráfico de Red y Paquetes UDP/TCP *El tráfico de red se refiere al intercambio de datos entre dispositivos en una red de computadoras. Este intercambio se realiza mediante paquetes de información que utilizan protocolos.*

Los protocolos utilizados en este trabajo son los protocolos UDP y TCP.

- TCP (Protocolo de Control de Transmisión): Este protocolo garantiza la entrega de datos. Este establece una conexión entre emisor y receptor antes de enviar la información.
- UDP (Protocolo de Datagramas de Usuario): Es un protocolo que se usa también para enviar datos en redes, pero no tiene que hacer una conexión previa.

El análisis del tráfico de red y de los paquetes UDP/TCP es fundamental en ciberseguridad para detectar y prevenir ataques como los de denegación de servicio que se intentarán detectar en este trabajo, intrusiones y malware.

Existen herramientas de análisis de tráfico de red que ayudan a monitorear el tráfico, identificar comportamientos sospechosos y proteger la red y los sistemas contra posibles amenazas.

Capítulo 3

Estado del arte

3.1. Introducción

Dentro de este apartado se hablará del estado actual de tanto machine learning como el de la ciberseguridad, dentro de estos apartados se hablará de los modelos utilizados para la realización del trabajo y de modelos alternativos que se podrían haber usado o se ha planteado usarlos.

3.2. Técnicas para el preprocesamiento

A la hora de utilizar un dataset para entrenar una inteligencia artificial, lo primero que hay que plantearse es en el preprocesamiento de ese dataset para dejarlo más operable a la hora de entrenar la inteligencia artificial mediante el análisis de los datos y la realización de una serie de técnicas para lograrlo siendo estas las principales:

- **Normalización de datos:** Consiste en ajustar los datos para que tengan una escala uniforme. Este tratamiento ayuda a garantizar que todas las características tengan un peso igual a la hora de analizar.
- **Eliminación de ruido:** Esta técnica permite eliminar datos irrelevantes que puedan afectar negativamente la precisión del modelo. Esto puede incluir datos duplicados, datos innecesarios o datos que no son consistentes.
- **Detección y manejo de valores atípicos:** Se puede identificar y tratar los valores atípicos que pueden distorsionar los resultados del análisis gracias a esta técnica. Esto puede implicar su eliminación.

- **Reducción de dimensionalidad:** Gracias a esta herramienta se puede reducir el número de características en el conjunto de datos mientras se conserva la mayor cantidad posible de información relevante. Esto puede ayudar a mejorar la velocidad de procesamiento y reducir la complejidad del modelo.
- **Balanceo de datos:** Nos permite asegurar que el conjunto de datos esté equilibrado para evitar sesgos en el análisis.

3.3. Técnicas de machine learning

3.3.1. Redes neuronales

Dentro de las redes neuronales existen una serie de distintos tipos que se utilizan dependiendo de la naturaleza del problema que se quieran abordar, estas son las Redes Neuronales Artificiales, Redes Neuronales Convolucionales, Redes Neuronales Recurrentes, Redes Neuronales Profundas, Redes Neuronales Generativas Adversariales.

Redes Neuronales Feedforward (FNN): Son el tipo de redes neuronales más básicas y están compuestas por capas de neuronas interconectadas. Se utilizan para problemas de clasificación y regresión, este es el tipo de red neuronal que usaré para el entrenamiento de la red neuronal del trabajo, se hablará más de ellas en un apartado debajo.

Redes Neuronales Convolucionales (CNN): Son especialmente útiles para el procesamiento de datos estructurados, como por ejemplo imágenes. Las CNN son muy efectivas en tareas de reconocimiento de patrones por ello destaca su utilidad en datos estructurados.

Redes Neuronales Recurrentes (RNN): Diseñadas para procesar secuencias de datos. Se usan principalmente para procesar lenguaje natural y reconocer voz.

Redes Neuronales Generativas Adversariales (GAN): Utiliza dos redes neuronales, un generador y un discriminador, que compiten entre sí por cual es la más

optima y se dan retroalimentación entre ellas. Las GAN se usan en aplicaciones como la generación de imágenes y la síntesis de voz.

3.3.2. Redes Neuronales Feedforward

También son conocidas como Redes Neuronales Artificiales (ANN), son un tipo de red neuronal en la que la información fluye en una dirección, desde la capa de entrada hasta la capa de salida, sin ciclos ni retroalimentación. Las FNN están compuestas por tres tipos de capas distintas, siendo estas las capas de entrada que recibe los datos de entrada y pasa estos datos a la siguiente capa, además cada neurona de esta capa es una característica que se tiene para entrenar la red.

Las capas ocultas son las que calculan para transformar los datos de entrada de forma más útil para la capa de salida, además de las neuronas de estas capas conectadas a todas las neuronas de la capa anterior y la siguiente. El siguiente tipo de capa que existe son las capas ocultas, estas son capas que realizan cálculos entre las capas utilizando los pesos

Por último, la capa de salida genera una salida final de la red neuronal. La forma y función de esta capa dependen del problema que trate de resolver la red neuronal, en este proyecto será solo una neurona para decir si ha sido o no un ataque.

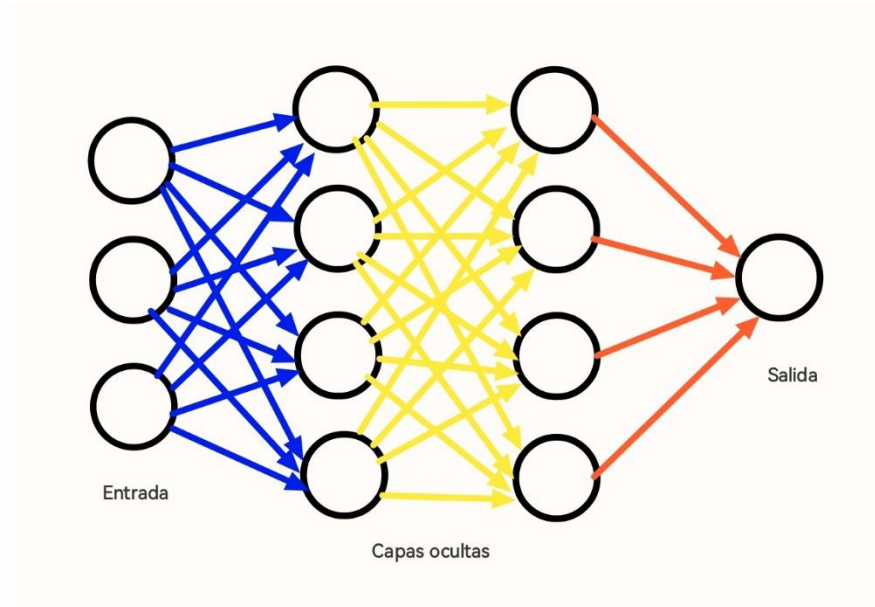


Ilustración 1: Estructura de una red neuronal

Dentro de este tipo de redes neuronales nos encontramos las redes neuronales multicapa o MLP, este tipo de red neuronal se caracteriza por tener una o más capas ocultas entre la capa de entrada y la capa de salida. Cada capa oculta en cada consta de neuronas interconectadas. La presencia de múltiples capas ocultas permite que la red multicapa aprenda representaciones más complejas de los datos de entrada. El objetivo de estas redes neuronales consiste en ir ajustando progresivamente los pesos sinápticos de las neuronas para conseguir que el error sea mínimo entre neuronas, para modificar estos pesos se usan herramientas como la retropropagación, la cual modifica los pesos entre conexiones de neuronas, para así minimizar el error.

3.3.3. Neuronas artificiales

Como ya se ha comentado, estas redes se componen de neuronas artificiales, elementos que detallaré a continuación.

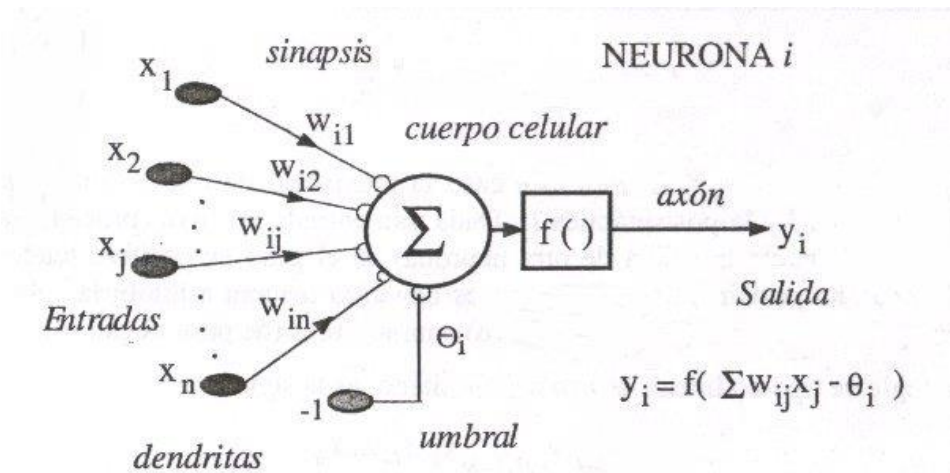


Ilustración 2: La neurona artificial. (s. f.). https://www.ibiblio.org/pub/linux/docs/LuCaS/Presentaciones/200304curso-glisa/redes_neuronales/curso-glisa-redes_neuronales-html/x38.html

- Primero se puede observar las entradas, que como ya se ha hablado antes tiene una entrada por cada neurona de la capa anterior.
- Cada entrada también tiene un peso asociado, este indica la importancia relativa para la neurona, cada peso se multiplica por el valor de la entrada para así ajustar su importancia.
- En una neurona hay un umbral que determina si esta neurona se activa o no, si el valor de la salida es mayor que el umbral se activa esta neurona.
- Por último, existe la función de activación que se transmite a través del axón, esta función de activación permite calcular el valor de la neurona, existen varios tipos de funciones de activación como la ReLU o la sigmoideal, las cuales serán las utilizadas en este proyecto.

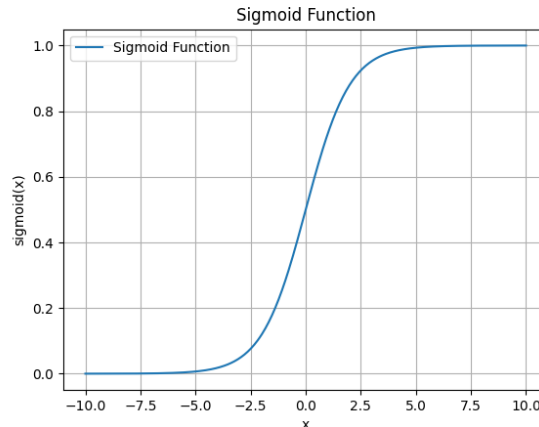


Ilustración 3: Función sigmoidal

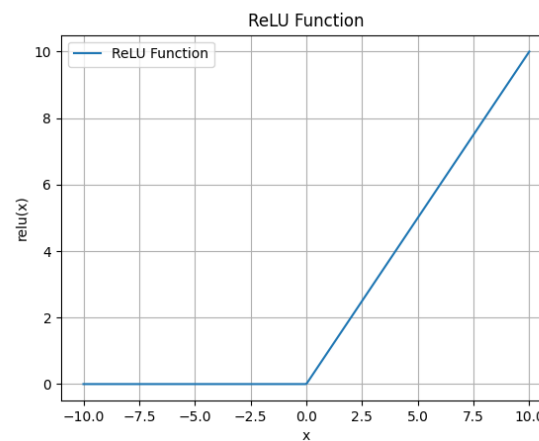


Ilustración 4: Función ReLU

3.3.4. Random forest

Alternativamente al uso de una red neuronal, también se puede utilizar la técnica de Random Forest. Random Forest es un potente algoritmo de aprendizaje automático basado en árboles de decisión. Es una técnica de conjunto de métodos, lo que significa que obtiene y combina resultados para unirlos en un resultado final, creando varios árboles de estado, puede haber de decenas a centenas y cada árbol utiliza un subconjunto de datos aleatorios sacados del conjunto de entrenamiento.

Esta técnica destaca en su utilización para entrenar un modelo capaz de distinguir entre distintas clases de datos, fundamental para detectar y clasificar intrusiones en el tráfico de red. Este algoritmo de clasificación puede desempeñar un

papel crucial en la eficacia y precisión del sistema de detección, ya que determina cómo se aprenderán y generalizarán los patrones presentes en los datos.

El algoritmo Random Forest se caracteriza por su versatilidad y eficacia comprobada en la detección de intrusiones en redes. Este algoritmo detecta comportamientos extraños de manera efectiva. En el caso de los comportamientos abusivos, Random Forest crea patrones de intrusiones automáticamente a partir de los datos de entrenamiento, lo que simplifica la identificación precisa de posibles intrusos al comparar los datos con estos patrones.

El estudio que he realizado se basa en la investigación de Nabila Farnaaz y M. A. Jabbar, titulada 'Random Forest Modeling for Network Intrusion Detection System', presentada en la Duodécima Conferencia Internacional Multi-Conferencia sobre Procesamiento de Información-2016 (IMCIP-2016). En este estudio, se exploró el uso del algoritmo Random Forest (RF) para la detección de varios tipos de ataques, incluyendo DOS, sonda, U2R y R2L. Se implementó una estrategia de validación cruzada de 10 iteraciones para la clasificación, y se aplicó selección de características para reducir la dimensionalidad y eliminar características redundantes e irrelevantes. El enfoque propuesto se evalúa con datos NSL KDD y se comparó con el clasificador J48 con métricas de rendimiento, tasa de detección (DR), tasa de falsos positivos (FAR) y coeficiente de correlación de Matthews (MCC). Los resultados experimentales demostraron mejoras significativas en la precisión, DR y MCC para los cuatro tipos de ataques con el método propuesto. Por lo tanto, la aplicación de Random Forest se presenta como una opción prometedora para la detección de ataques en entornos de seguridad informática, ofreciendo mejoras sustanciales en la capacidad de detección y precisión del modelo.

Esta es la tabla obtenida en dicho estudio para Random Forest:

Table 1. Performance Measure for Random Forest (No. of trees = 100).

SNO	Attack Type	Accuracy	DR	Far	MCC
1	DoS	99.67	99.84	0.00527	0.99
2	Probe	99.67	99.82	0.00502	0.99
3	R2L	99.67	99.82	0.00505	0.99
4	U2R	99.67	99.84	0.00552	0.99

A pesar de haber realizado este proyecto con redes neuronales también se realizó una versión utilizando Random Forest, tras analizar los resultados se concluyó que efectivamente esta herramienta es muy certera a la hora de detectar ataques a la red, en este caso DDoS.

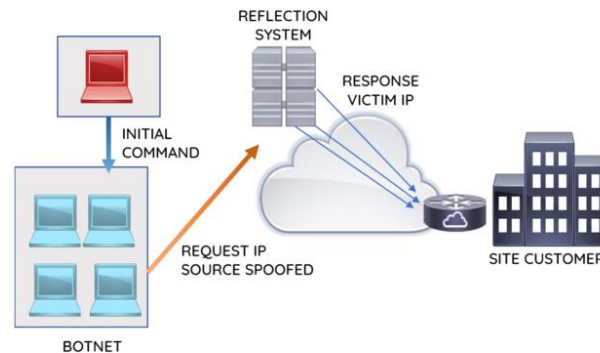
3.4. Técnicas de ciberseguridad

3.4.1. Ataques DDoS

La técnica de ataque de denegación de servicio existe desde finales de los 90, pero lejos de desaparecer o caer en desuso, esta técnica ha evolucionado y refinándose hasta hoy, donde es una de las formas más populares de ataque. Los primeros ataques DDoS eran relativamente simples, empleando un número limitado de máquinas comprometidas para inundar un servidor con tráfico. Sin embargo, con el tiempo, los atacantes han desarrollado métodos más sofisticados y potentes para realizar estos ataques, esto lleva ha llevado a su actual auge.

Mientras que en sus inicios los ataques se perpetraban desde un solo equipo o un grupo reducido hoy en día, los ataques DDoS pueden involucrar cientos de miles o incluso millones de dispositivos comprometidos, conocidos como botnets. Un ejemplo de estos ataques fue el que recibió Google en 2023 dirigido a un servicio de Google Cloud Plataform, en este ataque Google afirma haber recibido durante 7 minutos 398

millones de solicitudes por segundo, aunque desde Google fueron capaces de hacerle frente a este ataque debido a que su fuerte infraestructura global le permite mitigar los ataques gracias a que conecta todos sus centros de datos, permitiéndole una capacidad de cómputo de más de 300 terabytes por segundo.



*Ilustración 5: Coreun. (2020, 10 junio). DDoS-Ataques denegación de servicio (parte I). COREUN.
<https://coreun.com/2020/06/10/ddos-ataques-denegacion-de-servicio-parte-i/>*

Dentro de los ataques DDoS hay una serie de tipos, donde los principales actualmente son los volumétricos, los de protocolo y los de capa de aplicación:

- **Ataques Volumétricos:** Estos son los ataques que son asociados comúnmente como ataques DDoS debido a que son los más habituales, su objetivo es simple, abrumar el objetivo con solicitudes para ello intentan consumir el ancho de banda disponible entre el objetivo y el resto de la Internet. Estos ataques aprovechan dispositivos IoT para crear botnets y realizar ataques, esto es debido al creciente aumento de dispositivos IoT baratos con poca seguridad aprovechado por los perpetradores de estos ataques.
- **Ataques de Protocolo:** Este tipo de ataques consiste en la definición un conjunto específico de reglas para intercambiar información a través de la red. El protocolo más usado para el intercambio de solicitudes y datos es el protocolo TCP/IP. Aprovechando estas reglas, un actor malicioso puede causar graves interrupciones en un servicio en línea.

Los ataques de protocolo generalmente operan en las capas 3 y 4 del modelo OSI y afectan a dispositivos de red como routers. Debido a su naturaleza en la capa de red, estos ataques se miden en paquetes por segundo.

- **Ataques de Capa de Aplicación:** Estos ataques se caracterizan por su dificultad para detectarse porque ya que los atacantes suelen hacer solicitudes que parecen legítimas. Como resultado, estos ataques a menudo se manifiestan como pequeños picos de tráfico y no requieren la ayuda de una botnet. Los ataques a la capa de aplicación se miden en solicitudes por segundo, que es el número de solicitudes que realiza una aplicación. Este tipo de ataque se considera basado en recursos, por lo tanto, se necesitan menos solicitudes para derribar una aplicación porque el ataque se centra en abrumar la CPU y la memoria.

Un ataque a la capa de aplicación típicamente implica atacar el servidor web, ejecutar scripts PHP y acceder a la base de datos para cargar páginas web. Una única solicitud HTTP, que es fácil de realizar desde el lado del cliente, puede obligar al servidor a procesar múltiples solicitudes internas y cargar numerosos archivos, lo que provoca una ralentización del sistema.

Dentro de los ataques DDoS, los ataques pueden realizarse a través de diversos protocolos, en este trabajo debido al dataset elegido los protocolos para tener en cuenta son el TCP y el UDP, además estos son los más explotables debido a su uso extensivo, su facilidad de ser explotados debido a sus características más intrínsecas además de la facilidad que tienen para inundar recursos red. Además, TCP y UDP tienen tipos de ataque determinados.

3.4.2. Ataques TCP

El protocolo TCP establece una conexión antes de transferir datos. Esta característica lo hace vulnerable a diversos tipos de ataques DDoS. Entre ellos, el SYN Flood explota el proceso de establecimiento de conexión de TCP. En este tipo de ataque, un atacante envía una gran cantidad de solicitudes SYN para iniciar conexiones, pero no responde a las respuestas SYN-ACK del servidor. Como resultado, los recursos del servidor se agotan, volviéndolo incapaz de aceptar nuevas conexiones legítimas.

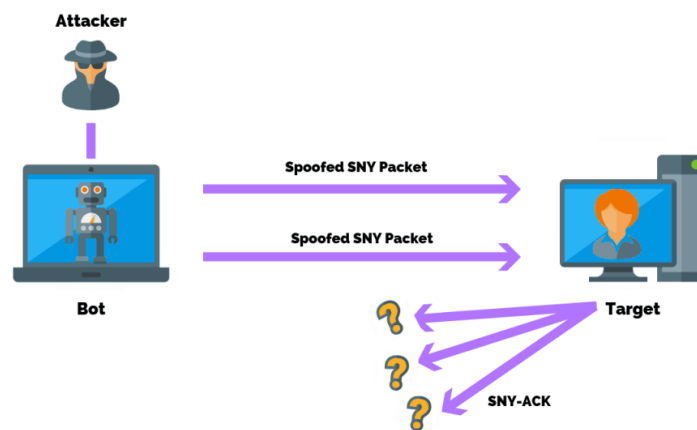


Ilustración 6: Maldonado, D. (2022, 16 abril). ¿Qué es una inundación TCP SYN Flood? Daniel Maldonado. <https://danielmaldonado.com.ar/seguridad/que-es-una-inundacion-tcp-syn-flood/>

Otro tipo de ataque que afecta al protocolo TCP es el ACK Flood. En este escenario, un atacante envía muchos paquetes ACK para sobrecargar la capacidad de red y los recursos del servidor. Al inundar el servidor con estos paquetes de reconocimiento, se interrumpen o ralentizan las operaciones normales del servidor, afectando su capacidad para manejar nuevas conexiones.

3.4.3. Ataques UDP

El protocolo UDP se caracteriza, como se ha comentado anteriormente, por ser un protocolo de transporte sin conexión, lo que implica que no establece una conexión previa antes de la transmisión de datos. Esta naturaleza lo convierte en un objetivo particularmente susceptible a una variedad de ataques DDoS. Uno de estos ataques comunes es el UDP Flood, en el cual el atacante envía una gran cantidad de paquetes

UDP a puertos aleatorios del servidor. Al no encontrar ninguna aplicación asociada con estos puertos, el servidor responde con paquetes ICMP de destino inalcanzable, lo que consume los recursos del servidor y de la red, resultando en una denegación de servicio.

Otro tipo de ataque que explota las vulnerabilidades inherentes al protocolo UDP es la Amplificación UDP. En este escenario, el atacante utiliza servidores intermedios para aumentar significativamente el volumen del tráfico dirigido a la víctima. El atacante envía consultas UDP de tamaño pequeño a servidores abiertos, como servidores DNS, falsificando la dirección IP de la víctima como la dirección de origen. Estos servidores, al responder a la solicitud, envían grandes cantidades de datos a la dirección IP falsificada, saturando así la red de la víctima con tráfico malicioso.

Capítulo 4

Desarrollo de la aplicación

4.1. Arquitectura usada

Antes de realizar el sistema de detección de ataques voy a hacer un esquema que sirva como objetivo a seguir a la hora de realizar el trabajo, siendo este el siguiente:

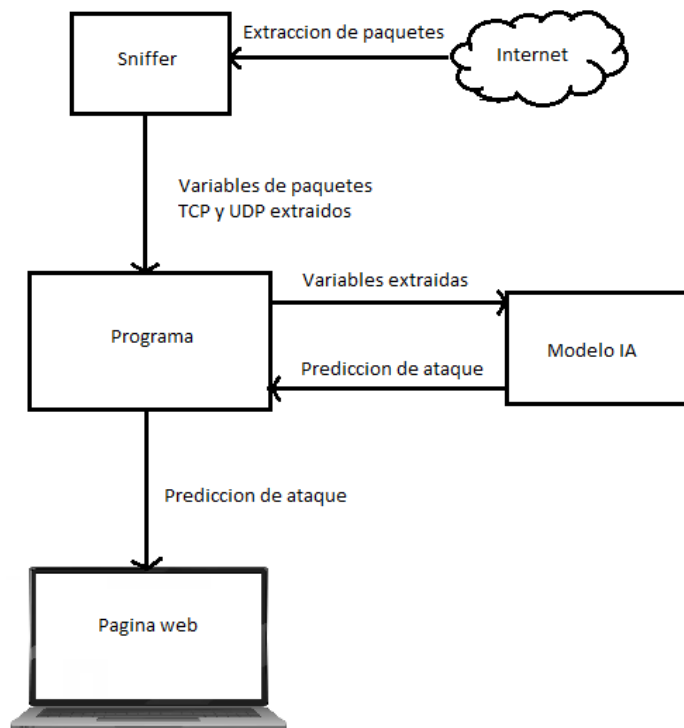


Ilustración 7:Arquitectura del programa

En este esquema se puede observar cómo se va a crear un “sniffer” cuyo trabajo consiste en recoger paquetes de internet y los enviará a un programa el cual utilizará un modelo de inteligencia artificial para determinar si se está realizando o no un ataque. Por último, se enviará esta predicción a una página web en la cual se mostrará si se está o no realizando un ataque.

4.2. Elección del dataset

A la hora de desarrollar una red neuronal para poder detectar si se está sufriendo un ataque DDoS, lo primero que hay que hacer es obtener un dataset en el que se obtenga tanto tráfico de ataque DDoS como tráfico normal, así poder enseñar a la red neuronal con casos balanceados. He utilizado dos datasets, uno con tráfico de red cuando se realiza un ataque y otro con tráfico normal de red, estos contienen ataques

mediante paquetes UDP y TCP, se puede observar la distribución de estos en la siguiente gráfica.

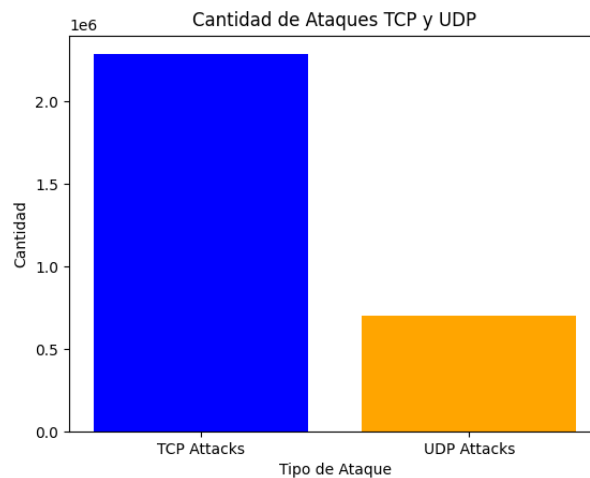


Ilustración 8: Distribución del dataset

Se puede observar que en el dataset hay una mayoría de ataques realizados a través de paquetes TCP, sin embargo, la cantidad de paquetes UDP es también muy elevada, por lo tanto, habrá que tener ambos en cuenta, pero a la hora de eliminar columnas es probable que las de los paquetes TCP tengan más importancia.

El primer paso una vez obtenido el dataset es realizar el preprocesamiento de los datos, para ello voy a utilizar la librería de python pandas.

Primero en vez de lidiar con ambos datasets voy a unir los dos en uno, pero debido a que 6 millones de filas de datos sumadas a la gran cantidad de columnas de los datasets elegidos, es demasiado para que mi ordenador pueda procesarlos correctamente, tan solo voy a coger la mitad de los datos de cada uno, dejando 3 millones de filas de datos en total, una cifra muy alta de datos y que mi ordenador puede manejar sin problema.

```
conjunto1 = pan.read_csv("C:/Users/drngb/Desktop/tfg/dataset/dataset_attack.csv")
conjunto2 = pan.read_csv("C:/Users/drngb/Desktop/tfg/dataset/dataset_normal.csv")

conjunto1 = conjunto1.iloc[1500000:]
conjunto2 = conjunto2.iloc[1500000:]

conjunto = pan.concat([conjunto1, conjunto2])
```

Ilustración 9: Unión del dataset

4.3. Preprocesado

4.3.1. Modificación inicial

Una vez listo el dataset se empezará con el preprocesamiento.

Primero se modifican los valores que no son numéricos, ya que las redes neuronales trabajan mejor con valores numéricos. Primero se designan los datos categóricos. Estos son datos clasificados en diferentes categorías, como "género", "color" o "tipo de ataque".

Esto lo hago para aplicar el one-hot encoding, el cual consiste en una técnica utilizada en el procesamiento de datos para convertir variables categóricas en una forma que pueda ser proporcionada a los algoritmos de aprendizaje automático para su mejor procesamiento.

Básicamente, el one-hot encoding convierte cada valor único en un vector binario donde un bit corresponde a cada posible valor único. Para una variable categórica con una serie de valores únicos, se crea esa cantidad de nuevos atributos binarios. Cada atributo binario representa un valor único de la variable categórica y toma el valor de 1 si la observación corresponde a ese valor y 0 en caso contrario.

En el caso de las direcciones IP (ip.src e ip.dst), aunque son de tipo “object”, no se consideran variables categóricas en este contexto. Esto se debe a que las direcciones IP son valores únicos que no representan categorías discretas, entonces estas variables no serán modificadas, además en el caso de se aplicara one-hot encoding en ellas se generaría una cantidad demasiado elevada de columnas.

Antes de realizar el one-hot encoding existe un problema dentro del dataset, este es debido a la columna de frame.protocols, dentro de esta columna se obtienen los protocolos encapsulados en un paquete de red de la siguiente forma:

`.eth:ethertype:ip:udp:data`, como se puede observar hay 5 datos distintos y para poder realizar un análisis correcto de la columna será oportuno dividirla en 5, para ello se desarrollará un script que realice tal acción dejando las columnas requeridas.

```

frame_protocol_split = conjunto["frame.protocols"].str.split(":", expand=True)
max_protocol_columns = 5
frame_protocol_split = frame_protocol_split.iloc[:, :max_protocol_columns]

frame_protocol_split.columns = [f"frame.protocols_{i}" for i in range(frame_protocol_split.shape[1])]
conjunto = pan.concat([conjunto, frame_protocol_split], axis=1)

```

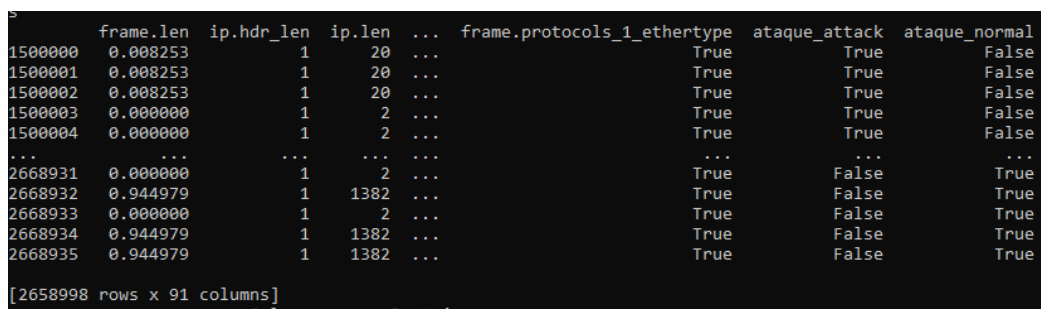
Ilustración 10: División de frame.protocols

Una vez dividida esta columna y con las nuevas columnas con valores categóricos asignados en su propia categoría, ya puedo realizar one-hot encoding, no sin antes eliminar la columna de frame.protocols original.

Para aplicar el one-hot encoding, en el primer y segundo protocolo puedo aplicarlo con una variable booleana, sin embargo, con el resto no debido a su elevado número de posibilidades.

Una vez ejecutado añade muchas columnas, porque 2 de estas solo tienen una posibilidad, pero 1 de ellas 3, 1 y 7 posibilidades, la última 61, por lo que hay que añadir muchas columnas para hacerlo binario.

Este problema lo abordare más adelante mediante la aplicación de métodos de eliminación de columnas.



	frame.len	ip.hdr.len	ip.len	...	frame.protocols_1_ethertype	ataque_attack	ataque_normal
1500000	0.008253	1	20	...	True	True	False
1500001	0.008253	1	20	...	True	True	False
1500002	0.008253	1	20	...	True	True	False
1500003	0.000000	1	2	...	True	True	False
1500004	0.000000	1	2	...	True	True	False
...
2668931	0.000000	1	2	...	True	False	True
2668932	0.944979	1	1382	...	True	False	True
2668933	0.000000	1	2	...	True	False	True
2668934	0.944979	1	1382	...	True	False	True
2668935	0.944979	1	1382	...	True	False	True

[2658998 rows x 91 columns]

Ilustración 11: Dataset imprimido

Una vez aplicado el one-hot encoding se busca una forma de hacer que los datos estén en una escala similar para mejorar el rendimiento y la confiabilidad del futuro entrenamiento de la red neuronal y ayudarlo a converger rápidamente, aplicamos el método de estandarización a los datos de entrenamiento y prueba

utilizando la función `StandardScaler` del módulo `sklearn.preprocessing`. Este método es un paso importante en el preprocesamiento de datos porque facilita la comparación de datos por parte de la red neuronal al entrenar y evita que haya números muy dispares y por lo tanto el entrenamiento de ralentice.

```
# Selecciona solo las columnas numéricas
numeric_cols = conjunto.select_dtypes(include=['float64', 'int64'])
scaler = StandardScaler()
conjunto[numeric_cols.columns] = scaler.fit_transform(numeric_cols)
```

Ilustración 12: Escalada de las columnas numéricas

Para su normalización se ha usado la función de `StandardScaler`, más concretamente utilizo la función de `fit_transform` que se utiliza para ajustar el `StandardScaler` al conjunto de datos y, al mismo tiempo, transformar los datos. En otras palabras, `fit_transform` calcula la media y la varianza de los datos y luego estandariza los datos aplicando la transformación.

Una vez normalizados los datos obtenemos datos más regulares que con los que contaba el dataset en un inicio, esto permitirá a la red neuronal poder trabajar de forma más óptima ya que en el dataset original los datos de algunas columnas iban desde 0 hasta valores muy altos.

4.4.2. Limpieza de datos

El siguiente paso en el preprocesamiento es el de descarte de columnas dentro del dataset, para ello voy a eliminar o mantener una serie de columnas de forma manual, tras ello haré uso de la herramienta de sklearn.

Primero de todo voy a eliminar la primera columna del dataset, esta es `frame.encap_type`, que consiste en el tipo de encapsulado utilizado en el marco de red, y siempre es 1, por lo tanto, es un dato que se puede omitir.

Tras ello mantendré las direcciones IP de origen y destino `ip.src` y `ip.dst` debido a que estas columnas pueden ser importantes, si bien es cierto que las direcciones IP y los puertos pueden ser variables dinámicas y no siempre proporcionarán información definitiva sobre un atacante, aun así, pueden ser útiles en el análisis de seguridad ya que por ejemplo puedo recibir un ataque DDoS de una IP concreta y si no tengo en cuenta las IPs podría ser más difícil la identificación de este.

Tras el análisis de estas variables voy a pasar con el análisis al uso de la herramienta de scikit-learn.

A la hora de usar esta herramienta se ha basado en un enfoque de selección de características utilizando el puntaje de chi-cuadrado (χ^2). Este enfoque evalúa la relación entre cada característica y la variable objetivo, en este caso, la presencia de un ataque (`ataque_attack`). El puntaje de chi-cuadrado proporciona una medida de la dependencia entre la característica y la variable objetivo, lo que ayuda a identificar qué características son más informativas para predecir el resultado deseado. Al eliminar las características con puntajes de chi-cuadrado más bajos, se simplifica el modelo y se reduce el riesgo de sobreajuste, lo que puede mejorar la capacidad predictiva del modelo y facilitar la interpretación de los resultados.

El código proporcionado realiza la preparación de datos y la selección de características utilizando la técnica de prueba estadística chi-cuadrado (χ^2) con el

método SelectKBest de scikit-learn. Primero, carga y preprocesa los conjuntos de datos de ataque y normal, realizando codificación one-hot para ciertas características categóricas y normalización de características numéricas. Luego, calcula los puntajes chi-cuadrado para cada característica y los clasifica de mayor a menor importancia, lo que permite identificar las características más relevantes para predecir ataques DDoS.

```
X = conjunto.drop(columns=['ataque_normal', 'ataque_attack', 'ip.src', 'ip.dst'])
y = conjunto['ataque_attack']

sel = SelectKBest(chi2, k=2)
sel.fit(X, y)

scores = sel.scores_

puntuaje = pan.DataFrame({'Columna': X.columns, 'Puntuaje': scores})

puntuaje_ordenado = puntuaje.sort_values(by='Puntuaje', ascending=False)
```

Ilustración 13: Código para obtener la importancia de las columnas

Las columnas que más efecto tienen en los ataques son las siguientes:

1	Columna, Puntuaje
2	tcp.ack, 85124693255.63806
3	tcp.time_delta, 16622016478.999758
4	tcp.srcport, 1847653212.6924758
5	tcp.dstport, 1344245886.5384603
6	tcp.window_size, 463739786.9408528
7	tcp.len, 2698267.8644755194
8	ip.len, 2115291.528470905
9	frame.protocols_4_ftp-data, 403684.7339986324
10	ip.ttl, 381357.69687704125
11	frame.protocols_4_http, 79715.97419656688
12	frame.protocols_4_data, 55421.988022926394
13	tcp.flags.push, 47951.181407655466
14	frame.protocols_3_udp, 21744.86978177214
15	tcp.flags.fin, 20561.04129797631
16	frame.protocols_4_ssl, 15004.633636289429
17	tcp.flags.syn, 10622.914712750706
18	frame.protocols_4_dns, 7739.0102888792735
19	tcp.flags.reset, 2283.2119139054284
20	frame.protocols_3_icmp, 2181.660127461988
21	frame.protocols_4_ip, 2181.660127461988
22	frame.protocols_4_bittorrent, 2053.005520575654
23	frame.len, 1437.5278328794907
24	frame.protocols_3_tcp, 917.3026983885379
25	frame.protocols_4_imap, 839.8795340754807
26	tcp.flags.ack, 449.6177578273865
27	frame.protocols_4_ftp, 338.15958285837365
28	ip.proto, 215.28463570000818
29	frame.protocols_4_smtp, 133.93866649699152
30	ip.flags.df, 109.9089625281535
31	frame.protocols_4_nbss, 92.99785059699758
32	frame.protocols_4_cattpp, 89.43165049508005
33	frame.protocols_4_pop, 58.684139360900225

Ilustración 14: Columnas más importantes

Las columnas que menos efecto tienen:

```

57 frame.protocols_4_udpencap,3.1379526489501774
58 frame.protocols_4_mih,3.1379526489501774
59 frame.protocols_4_opa.fe,3.1379526489501774
60 frame.protocols_4_ts2,2.353464486712633
61 frame.protocols_4_wow,2.353464486712633
62 frame.protocols_4_rsip,2.353464486712633
63 frame.protocols_4_openvpn,1.5689763244750887
64 frame.protocols_4_bundle,1.5689763244750887
65 frame.protocols_4_bootp,1.0441604391183863
66 frame.protocols_2_ipv6,0.9192241271751685
67 frame.protocols_4_memcache,0.7844881622375444
68 frame.protocols_4_ssh,0.7844881622375444
69 frame.protocols_4_zebra,0.7844881622375444
70 frame.protocols_4_vicp,0.7844881622375444
71 frame.protocols_4_asap,0.7844881622375444
72 frame.protocols_4_uudp,0.7844881622375444
73 frame.protocols_4_ax4000,0.7844881622375444
74 frame.protocols_4_tftp,0.7844881622375444
75 frame.protocols_4_mdns,0.7844881622375444
76 frame.protocols_4_dcerpc,0.7844881622375444
77 frame.protocols_4_diameter,0.7844881622375444
78 frame.protocols_4_socks,0.7844881622375444
79 frame.protocols_4_gift,0.7844881622375444
80 frame.protocols_4_rmi,0.7844881622375444
81 frame.protocols_4_pptp,0.7844881622375444
82 frame.protocols_4_h501,0.7844881622375444
83 ip.hdr_len,0.020344095915317904
84 frame.protocols_2_ip,0.01942474192022806
85 frame.protocols_0_eth,0.0
86 frame.protocols_1_ethertype,0.0
87 ip.flags.rb,
88 tcp.flags.urg,

```

Ilustración 15:Columnas menos importantes

Limpiaremos las que menos efecto tienen en relación con la variable de ataque. Se eliminarán todas las columnas con un puntaje menor a 10 que permitirá eliminar las de muy baja calidad a la hora de saber si se está realizando un ataque y ahorrar poder de cómputo y optimizar el sistema.

```

selected_columns = puntaje_ordenado[puntaje_ordenado['Puntaje'] >= 10]['Columna'].tolist()

selected_columns += ['ataque_normal', 'ataque_attack']
selected_columns += ['ip.src', 'ip.dst']

conjunto = conjunto[selected_columns]

```

Ilustración 16: Limpieza de columnas

Una vez se ha acabado esta modificación se puede concluir con el análisis y tratamiento del dataset y se puede seguir a el apartado de entrenamiento.

4.5. Entrenamiento del modelo

4.5.1. Elección de la red neuronal

Una vez completado el preprocesado del dataset, pasamos al siguiente punto que es la elección de un modelo de deep learning para el entrenamiento de un sistema capaz de diferenciar entre distintas clases de datos. Esto es fundamental para la detección y clasificación de intrusiones en el tráfico de red. El deep learning desempeña un papel crucial en la eficacia y precisión del sistema de detección, ya que determina cómo se aprenderán y generalizarán los patrones presentes en los datos. La elección de un modelo adecuado es esencial para garantizar la robustez y fiabilidad del modelo resultante, así como para maximizar su capacidad para identificar y responder oportunamente a posibles amenazas de seguridad en el entorno de la red.

Dentro del deep learning, existen muchas subclases y tipos de algoritmos. El principal criterio elegido para la selección es la elección de un modelo que puede tratar el problema con clases que no están representadas de forma equitativa.

Dentro de la detección de ataques las Redes Neuronales Artificiales (ANN) destacan por ser las más utilizadas, estas se basan en el uso de nodos interconectados que simulan las neuronas reales en el cerebro. Dentro de esta categoría, encontramos las feedforward (FNN), cuya característica principal es que las conexiones entre los nodos son lineales, van desde el principio hasta el final sin retroceder. En la FNN distinguimos dos modelos: el perceptrón de una sola capa y el que utilizaremos para este proyecto de perceptrón multicapa. He elegido este modelo ya que su implementación es sencilla y además ofrece alta precisión, baja tasa de falsos positivos, adaptabilidad y eficiencia computacional. También está bien documentado en la literatura científica.

4.5.2. Creación de la red neuronal

A la hora de hacer la red neuronal se ha utilizado Keras, que es una interfaz de alto nivel de Tensorflow, esto lo he hecho debido a que Keras simplifica mucho la

creación, el entrenamiento y la evaluación de modelos de redes neuronales y es compatible con redes feedforward como la que voy a usar para desarrollar el trabajo.

Primero voy a utilizar la clase Sequential para construir de forma sencilla un modelo en el que ir añadiendo capas de forma lineal.

```
model = Sequential()
```

Una vez creado el modelo voy a construir las capas, un modelo MLP se basa en 3 tipos de capas, estas son capa de entrada, capa oculta y capa de salida, primero añado la capa de entrada con los siguientes parámetros:

```
model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
```

Primero se van a añadir 64 unidades de entrada en base a los resultados con 64 ajustare el número de unidades, he escogido la capa de activación ReLU ya que aprovecho que no hay ningún valor negativo dentro de mi dataset, con los cuales la función ReLU no es capaz de trabajar, entonces no tengo ningún problema para usarla ya que esta es una muy buena función para redes neuronales y deep learning.

$$f(x) = \max(0, x)$$



Ilustración 17: Función de activación ReLU

```
model.add(Dense(units=64, activation='relu'))
```

La siguiente capa que se ha añadido es la capa oculta, con las mismas unidades y la misma función, 64 unidades y la función ReLU, que la capa de entrada, esta permite a la red neuronal aprender representaciones más complejas y abstractas de los datos de entrada, lo que puede ayudar a la red neuronal a aprender patrones más sutiles presentes en el tráfico de la red.

```
model.add(Dense(units=1, activation='sigmoid'))
```

Por último, se crea la capa de salida, en la que la red neuronal decidirá si es tráfico de ataque o es tráfico normal, esta tendrá únicamente una unidad y he utilizado la función de activación sigmoideal debido a que quiero obtener un resultado binario, si es un ataque DDoS o tráfico normal y debido a la naturaleza de la función sigmoideal que produce valores en un rango $[0,1]$ entonces es perfecta para este problema.

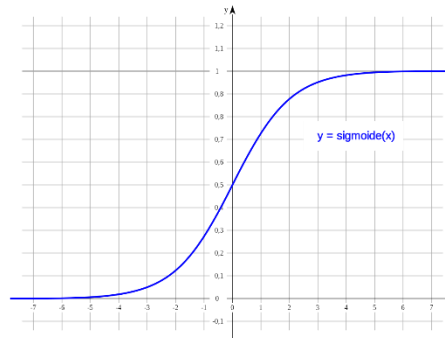


Ilustración 18: Función de activación sigmoideal

Una vez implementadas las capas entrenaré la red neuronal, para ello lo primero es compilar el modelo, que hago compilando.

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Dentro de esta función tengo que usar una primera función de pérdida, para ello usare binary crossentropy que es la más comúnmente usada en problemas de clasificación binaria como el de ataque DDoS en el que la estamos entrenando, tras ello tengo que especificar el algoritmo que usare para la optimización, que será el algoritmo Adam que es comúnmente utilizado para redes neuronales, este ajustará los pesos de la red neuronal durante el entrenamiento con el fin de minimizar la función de pérdida. Por último, he designado la métrica de “accuracy” para obtenerla durante el entrenamiento.

Una vez realizada la compilación voy a hacer el entrenamiento de la red neuronal, también voy a realizar la evaluación de los datos de los datos de validación, esto lo hago mediante la función de entrenamiento.

```
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
```

4.5.3. Entrenamiento de la red neuronal

Para entrenar el modelo utilizo los datos de entrenamiento que designé anteriormente, la red neuronal contará con 10 epochs, es decir que el modelo pasará 10 veces por todos los datos de entrenamiento, tras ello se define el “batch size” que consiste en el número de muestras que se utilizan en cada epoch para estimar el error del modelo, he definido este parámetro en 32, tras ello tengo que añadir los valores que he designado anteriormente como datos de validación.

Una vez añadido esto puedo realizar el entrenamiento de la red neuronal.

```
Epoch 1/10
+-----+-----+-----+-----+-----+
+1m49857/49857+0m+32m+-----+0m+37m+0m+1m29s+0m 572us/step - accuracy: 0.8442 - loss: 19.3211 - val_accuracy: 0.6880 - val_loss: 0.4928
Epoch 2/10
+-----+-----+-----+-----+-----+
+1m49857/49857+0m+32m+-----+0m+37m+0m+1m30s+0m 609us/step - accuracy: 0.8409 - loss: 0.2962 - val_accuracy: 0.8974 - val_loss: 0.2223
Epoch 3/10
+-----+-----+-----+-----+-----+
+1m49857/49857+0m+32m+-----+0m+37m+0m+1m31s+0m 615us/step - accuracy: 0.8935 - loss: 0.2270 - val_accuracy: 0.9027 - val_loss: 0.2628
Epoch 4/10
+-----+-----+-----+-----+-----+
+1m49857/49857+0m+32m+-----+0m+37m+0m+1m31s+0m 618us/step - accuracy: 0.9000 - loss: 0.2240 - val_accuracy: 0.9019 - val_loss: 0.2135
Epoch 5/10
+-----+-----+-----+-----+-----+
+1m49857/49857+0m+32m+-----+0m+37m+0m+1m30s+0m 610us/step - accuracy: 0.9016 - loss: 0.2211 - val_accuracy: 0.9045 - val_loss: 0.2083
Epoch 6/10
+-----+-----+-----+-----+-----+
+1m49857/49857+0m+32m+-----+0m+37m+0m+1m30s+0m 607us/step - accuracy: 0.9024 - loss: 0.2150 - val_accuracy: 0.9031 - val_loss: 0.2140
Epoch 7/10
+-----+-----+-----+-----+-----+
+1m49857/49857+0m+32m+-----+0m+37m+0m+1m30s+0m 596us/step - accuracy: 0.9029 - loss: 0.2130 - val_accuracy: 0.9049 - val_loss: 0.2111
Epoch 8/10
+-----+-----+-----+-----+-----+
+1m49857/49857+0m+32m+-----+0m+37m+0m+1m30s+0m 609us/step - accuracy: 0.9038 - loss: 0.2107 - val_accuracy: 0.9041 - val_loss: 0.2102
Epoch 9/10
+-----+-----+-----+-----+-----+
+1m49857/49857+0m+32m+-----+0m+37m+0m+1m23s+0m 454us/step - accuracy: 0.9035 - loss: 0.2094 - val_accuracy: 0.9015 - val_loss: 0.2495
Epoch 10/10
+-----+-----+-----+-----+-----+
+1m49857/49857+0m+32m+-----+0m+37m+0m+1m24s+0m 481us/step - accuracy: 0.9019 - loss: 0.2161 - val_accuracy: 0.9025 - val_loss: 0.2062
```

Ilustración 19:Entrenamiento red neuronal

Aquí podemos observar el entrenamiento de la red neuronal, obtenemos valores a la hora de ir progresivamente realizando el entrenamiento, obtenemos la accuracy , que es el valor de veces que la red neuronal ha acertado, podemos observar que va aumentando progresivamente hasta llegar al 90% donde se mantiene, estos valores son un signo positivo para saber que la red neuronal funciona, después obtenemos el valor de “loss”, que es el porcentaje de error a la hora de predecir las etiquetas, este va bajando hasta el 20%, también obtenemos estos valores para los datos de validación obteniendo una accuracy más alta y una perdida similar.

Ahora voy a utilizar esos valores para poder dar una idea de si se está dando overfitting o no, para ello hago una comparación entre los valores de los valores de entrenamiento y los de validación.

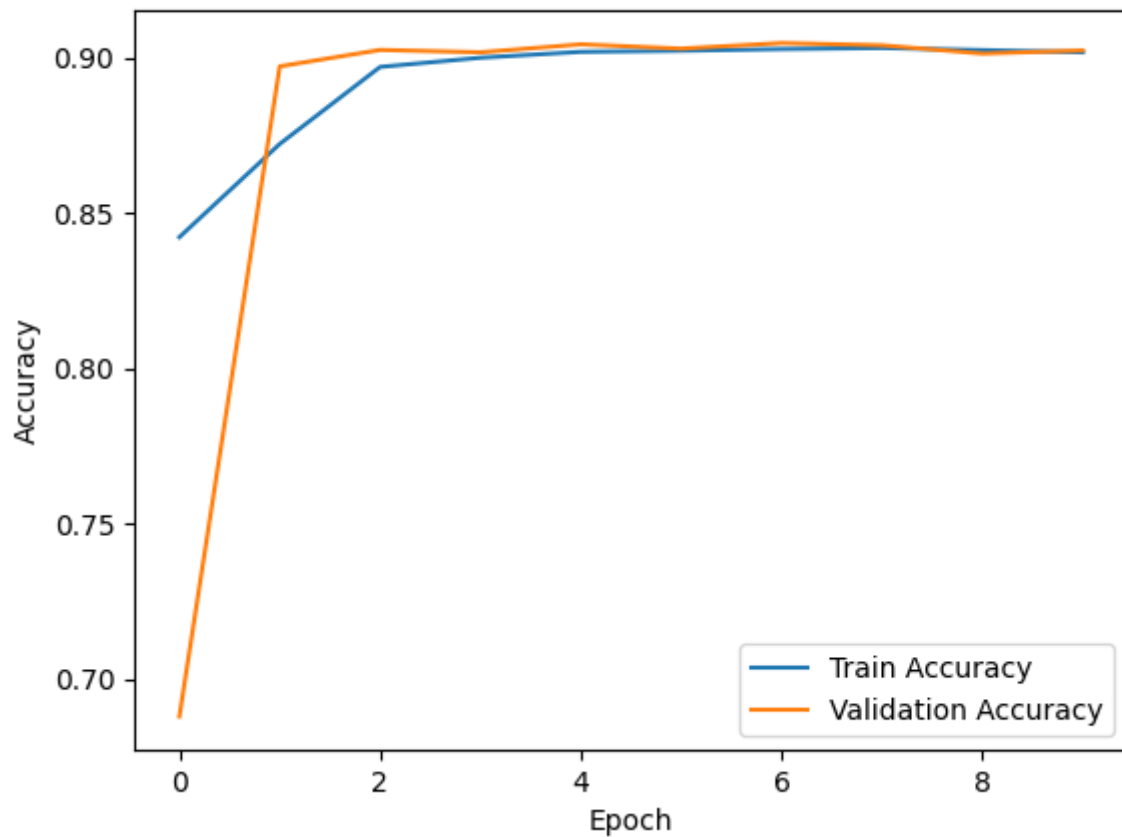


Ilustración 20: Evolución de la certeza del entrenamiento y validación

Observando el grafico no se puede ver ningún indicio de overfitting, este estaría presente si la “accuracy” del entrenamiento aumenta constantemente mientras que la “accuracy” de la validación se estanca o comienza a disminuir, también si la “accuracy” del entrenamiento es significativamente mayor a la de la validación, sin embargo, en nuestro caso se mantienen igualadas durante el entrenamiento de la red.

Ahora voy a obtener otras métricas más avanzadas una vez realizado este entrenamiento, estas serían, “accuracy”, “precision”, “recall”, “F1 score” y la “confusion matrix”.

```

Accuracy: 0.9024802557352388
Precision: 0.9825610793394455
Recall: 0.8409430961950067
F1 Score: 0.906252880960085
Confusion Matrix:
[[229269   4449]
 [ 47412 250670]]

```

Ilustración 21: Resultados del entrenamiento de la red neuronal

La “accuracy” la he ido observando según realizaba el entrenamiento, pero aquí la obtengo de forma global para el entrenamiento siendo esta un 90%.

La “precision” consiste en la capacidad que tiene la red neuronal de realizar predicciones correctas, se obtienen un 98% de predicciones correctas, un resultado muy bueno.

El “recall”, también conocido como la sensibilidad consiste en una métrica que evalúa la capacidad del modelo para identificar correctamente todos los ejemplos positivos de una clase en relación con todos los ejemplos que realmente son positivos en el conjunto de datos, se calcula mediante la siguiente formula:

$$\text{Recall} = \frac{\text{Verdaderos positivos}}{\text{Verdaderos positivos} + \text{Falsos negativos}}$$

Se obtiene un “recall” de 89%, una estadística bastante buena.

Por último, he tomado la medida de F1 score, que consiste en una medida que utiliza las funciones previamente obtenidas de “recall” y de “precision”, esta nos permite saber cuándo hay un desequilibrio entre las clases de los datos, esto es particularmente útil para problemas de clasificación binaria, la fórmula que usa esta métrica es la siguiente:

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Se obtiene un 90% por lo tanto sabemos que hay poco desequilibrio entre clases.

Dentro de la “confusion matrix” podemos obtener la siguiente información:

- El elemento [229269] representa los casos en los que el modelo predijo correctamente que no se está realizando un ataque.
- El elemento [4449] representa los casos en los que el modelo predijo incorrectamente que se está realizando un ataque, cuando es no lo hay.
- El elemento [47412] representa los casos en que el modelo predijo incorrectamente que no se está realizando un ataque, cuando realmente si se está produciendo.
- El elemento [250670] representa los casos en los que el modelo predijo correctamente que se está realizando un ataque.

Se pueden obtener conclusiones de estos datos, como que es raro que el modelo falle al detectar tráfico normal, pero es más frecuente que se detecten falsos positivos, que no deberían ser despreciables, ya que pueden hacer que se detecten falsos positivos que en un sistema alerten innecesariamente, esto se tratará más adelante en la elaboración del detector de ataques.

4.5.4. Conclusiones de la red neuronal

Una vez analizados todos estos datos podemos concluir que la red neuronal ha sido entrenada satisfactoriamente, ya que todos los datos parecen indicar que no ha habido problemas en su entrenamiento y en la gran mayoría de casos va a acertar a la hora de predecir si se está produciendo un ataque DDoS, sin embargo, es posible que se produzcan falsos positivos, este problema será tratado a la hora de construir el sistema de detección de ataques.

4.5.5. Random Forest

Alternativamente a esta versión del trabajo realizado con una red neuronal se ha realizado una versión con Random Forest, esto se hizo para analizar las diferencias entre el entrenamiento de una red neuronal y un entrenamiento con Random Forest, aquí se observan los datos obtenidos con esta versión finalmente desechada ya que resultó más interesante la opción de usar y ajustar una red neuronal.

```
C:\Users\drngb\Desktop\tfg>python division_rforest.py
Validation Accuracy: 0.9859778112072207
Test Accuracy: 0.9858819104926664
Test Precision: 0.9913158161712489
Test Recall: 0.9834273790433505
Test F1 Score: 0.9873558417762449
```

Ilustración 22: Resultados del entrenamiento de Random Forest

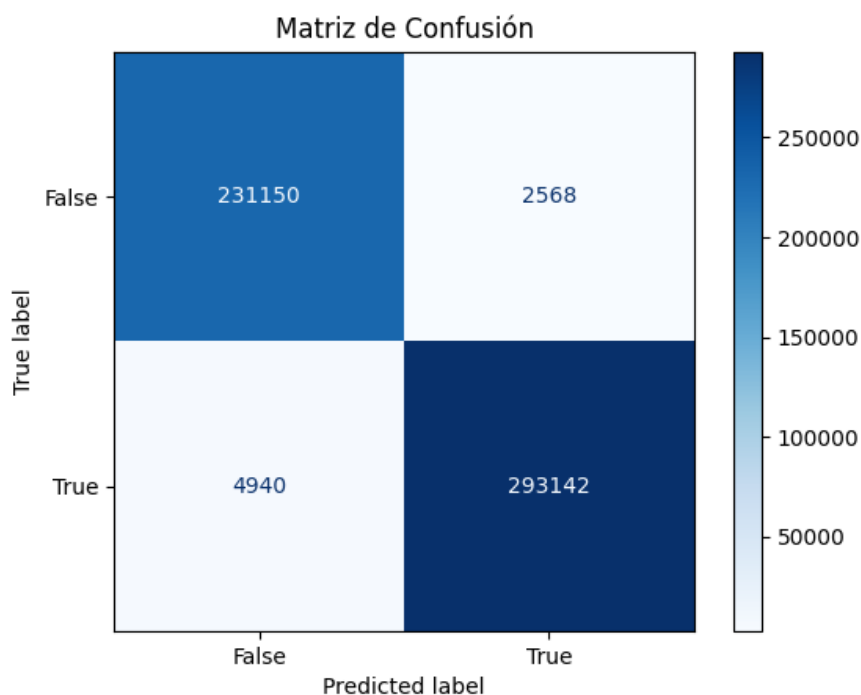


Ilustración 23: Matriz de confusión de Random Forest

A pesar de no usar esta versión me parece relevante observar que los datos obtenidos son mejores en todos los aspectos que los obtenidos con la red neuronal, así

que esta metodología también puede ser utilizada para realizar este proyecto y los resultados podrían ser mejores.

4.6. Análisis de paquetes en tiempo real

4.6.1. Adaptación de la red neuronal

El formato de análisis previamente realizado es correcto si buscamos entrenar una red neuronal de forma correcta, sin embargo, si lo que buscamos es analizar tráfico en tiempo real entonces el hecho de tener tantas variables supondrá un problema a la hora de probar la red neuronal, debido a esto voy a volver a analizar los datos obtenidos mediante la librería de sklearn.

```
Columna, Puntaje
tcp.ack, 85124693255.63806
tcp.time_delta, 16622016478.999758
tcp.srcport, 1847653212.6924758
tcp.dstport, 1344245886.5384603
tcp.window_size, 463739786.9408528
tcp.len, 2698267.8644755194
ip.len, 2115291.528470905
frame.protocols_4_ftp-data, 403684.7339986324
ip.ttl, 381357.69687704125
frame.protocols_4_http, 79715.97419656688
frame.protocols_4_data, 55421.988022926394
tcp.flags.push, 47951.181407655466
frame.protocols_3_udp, 21744.86978177214
tcp.flags.fin, 20561.04129797631
frame.protocols_4_ssl, 15004.633636289429
tcp.flags.syn, 10622.914712750706
frame.protocols_4_dns, 7739.0102888792735
tcp.flags.reset, 2283.2119139054284
frame.protocols_3_icmp, 2181.660127461988
frame.protocols_4_ip, 2181.660127461988
frame.protocols_4_bittorrent, 2053.005520575654
frame.len, 1437.5278328794907
frame.protocols_3_tcp, 917.3026983885379
_
```

Ilustración 24: Tabla de los valores a modificar

Voy a quedarme tan solo con las 14 primeras columnas, por lo tanto, en vez de hacer el corte en 10 lo voy a hacer en 20000, permitiéndome así obtener 14 columnas, añadiendo las columnas de ataque e IP desde donde se lanza el ataque y donde se recibe. Esto será más útil para abordar el problema de analizar datos reales.

Una vez limitamos el dataset ya podemos volver a entrenar la red neuronal.

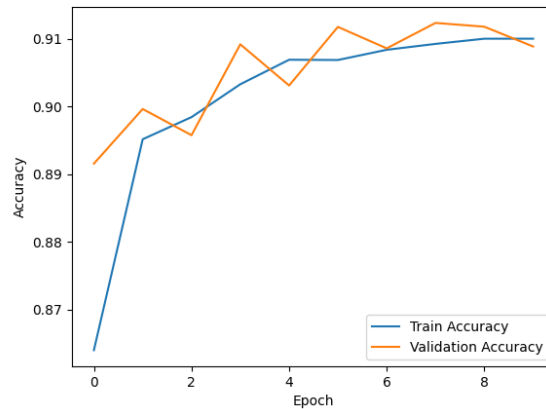


Ilustración 25: Gráfica de la evolución de la precisión del entrenamiento y la validación

```
Accuracy: 0.9094377585558481
Precision: 0.9762667937112911
Recall: 0.8593205896364088
F1 Score: 0.9140683406577133
Confusion Matrix:
[[227491  6227]
 [ 41934 256148]]
```

Ilustración 26: Resultados del entrenamiento de la red neuronal modificada

Una vez entrenada se observa que los valores siguen siendo buenos, podemos comprobar que durante el entrenamiento la precisión de la validación fluctúa, no manteniéndose tan regular como en el entrenamiento anterior, sin embargo, no son unos valores tan diferenciados como para considerarlo overfitting. Además, observamos que los valores finales tras el entrenamiento son bastante buenos.

Ahora se va a realizar el desarrollo de un programa para la obtención de un dataset en tiempo real de paquetes obteniendo datos del tipo del dataset, para ello haré uso de la librería de Python. Scapy es una poderosa herramienta de manipulación de paquetes de red escrita en Python. Permite a los usuarios capturar, enviar, decodificar y manipular paquetes de red en una red de forma sencilla y eficiente. Usaré las herramientas que proporciona para obtener datos detallados de los paquetes de red.

4.6.2. Creación de un dataset en tiempo real

Una vez se ha entrenado la red neuronal adaptada a el tráfico en tiempo real ya se puede comenzar el desarrollo de un código para almacenar información en base a paquetes reales.

```
src_ip = packet[IP].src
dst_ip = packet[IP].dst
tcp_layer = packet.getlayer(TCP)
udp_layer = packet.getlayer(UDP)
```

Ilustración 27: Obtención de ip, paquete tcp y udp

Primero se comprueba si el paquete tiene una capa IP. Si es así, extrae la dirección IP de origen y la dirección IP de destino del paquete, además si el paquete contiene una capa TCP o UDP asigna a las características “tcp_layer” o “udp_layer” el objeto que representa a la capa, si no hay nada asignara “None”.

```
ack = src_port = dst_port = window_size = tcp_len = ip_len = frame_protocols_4 ftp_data = ip_ttl = frame_protocols_4 http = frame_protocols_4 data = push_flag = fin_flag = syn_flag = normal_attack_flag = 0
```

Tras ello se inicializan los valores que se quieren obtener a 0.

```
if tcp_layer is not None:
    ack = tcp_layer.ack if hasattr(tcp_layer, 'ack') else 0
    src_port = tcp_layer.sport if hasattr(tcp_layer, 'sport') else 0
    dst_port = tcp_layer.dport if hasattr(tcp_layer, 'dport') else 0
    window_size = tcp_layer.window if hasattr(tcp_layer, 'window') else 0
    tcp_len = len(tcp_layer.payload) if hasattr(tcp_layer, 'payload') else 0
    push_flag = tcp_layer.flags.PSH if hasattr(tcp_layer, 'flags') and hasattr(tcp_layer.flags, 'PSH') else False
    fin_flag = tcp_layer.flags.F if hasattr(tcp_layer, 'flags') and hasattr(tcp_layer.flags, 'F') else False
    syn_flag = tcp_layer.flags.S if hasattr(tcp_layer, 'flags') and hasattr(tcp_layer.flags, 'S') else False
```

Ilustración 28: Inicialización de valores

Una vez inicializadas las variables, se obtienen los valores necesarios para el dataset relacionados con TCP. Si el paquete tiene una capa TCP, extrae las características de esa capa que se requieren para dejar un dataset igual que el anterior como el “ack” o el “src_port”. Al haber reducido el tamaño del dataset a simplemente los más importantes la mayoría de las características son de paquetes TCP. Como ya se adelantó al analizar el dataset la mayoría de los paquetes en el ataque son TCP y por ello estas características son las que tienen más importancia.

```
# Obtener el tiempo del paquete TCP
current_time = packet.time
time_delta = current_time - last_time if last_time else 0
last_time = current_time
```

Ilustración 29: Obtención del tiempo tcp

También hay que obtener la variable de “tcp.time_delta” en la que hay que tener en cuenta si es el primer paquete.

```
ip_len = packet[IP].len if hasattr(packet[IP], 'len') else 0
frame_protocols_4_ftp_data = 1 if 'FTP' in packet else 0
ip_ttl = packet[IP].ttl if hasattr(packet[IP], 'ttl') else 0
frame_protocols_4_http = 1 if 'HTTP' in packet else 0
frame_protocols_4_data = 1 if 'DATA' in packet else 0
normal_attack_flag = 1 if 'Attack' in packet else 0
```

Ilustración 30: Valores no relacionados con tcp ni udp

Tras ello, obtenemos los valores no relacionados con el protocolo TCP que son necesarios para el dataset basándose en el dataset ya existente con el que se ha entrenado la red neuronal en todo momento.

```
if udp_layer is not None:
    frame_protocols_3_udp = 1
```

Ilustración 31: Obtencion de valores relacionados con udp

Se puede observar que el único valor que ha quedado relacionado con UDP una vez reducido a solo los 15 más importantes es “frame_protocol_3_udp” que se obtiene siempre que exista el paquete UDP.

```
df = pd.DataFrame(packet_data, columns=['tcp.ack', 'tcp.time_delta', 'tcp.srcport', 'tcp.dstport', 'tcp.window_size',
    'tcp.len', 'ip.len', 'frame.protocols_4_ftp-data', 'ip.ttl',
    'frame.protocols_4_http', 'frame.protocols_4_data', 'tcp.flags.push',
    'frame.protocols_3_udp', 'tcp.flags.fin', 'tcp.flags.syn',
    'ip.src', 'ip.dst', 'protocol'])
```

Ilustración 32: Añadir valores al dataframe

Ahora se crea un dataframe con los datos del paquete que se van a meter dentro del dataset antes de añadirlos al .csv correspondiente, tras ello imprimo por pantalla el paquete y lo añado al .csv, así se podrá comprobar que los datos se están obteniendo correctamente.

```
sniff(prn=packet_handler, count=10)
```

Capturo tan solo los 10 primeros paquetes para poder hacer la prueba de su correcto funcionamiento y lo añade a el dataset creado como paquetes.csv

```
df = pd.DataFrame(packet_data, columns=['tcp.ack', 'tcp.time_delta', 'tcp.srcport', 'tcp.dstport', 'tcp.window_size',
                                       'tcp.len', 'ip.len', 'frame.protocols_4_ftp-data', 'ip.ttl',
                                       'frame.protocols_4_http', 'frame.protocols_4_data', 'tcp.flags.push',
                                       'frame.protocols_3_udp', 'tcp.flags.fin', 'frame.protocols_4_ssl',
                                       'tcp.flags.syn', 'ip.src', 'ip.dst', 'protocol'])

df.to_csv('paquetes.csv', index=False)
```

Ilustración 33:Exportación del csv

Ahora todo el código necesario está disponible para ejecutar y analizar 10 paquetes de red en esta prueba, en el código se ha limitado el tipo de paquetes que se puede obtener, solamente se puede obtener paquetes TCP (protocolo 6) y UDP (protocolo 17), esto es debido a que el modelo únicamente contempla paquetes TCP y UDP. Después de ejecutarlo, obtuvimos el siguiente resultado:

	tcp.ack	tcp.time_delta	tcp.srcport	tcp.dstport	tcp.window_size	tcp.len	ip.len	frame.protocols_4_ftp-data	ip.ttl
0	138786985	0.000000	443	55032	8	52	92	0	57
1	3128312153	0.053080	55032	443	1022	0	40	0	128
2	102924717	0.744683	55414	8095	1025	49	89	0	128
3	4252215752	0.098829	8095	55414	196	6	40	0	240
4	2885199518	3.617087	55638	443	1027	781	821	0	128
5	4077151196	0.003452	443	55638	424	453	493	0	58
6	3360248703	0.002168	55795	443	1021	1141	1181	0	128
7	2298759973	0.001130	443	55795	8	6	40	0	57
8	4077151196	0.004601	443	55638	424	453	493	0	58
9	2885199971	0.000040	55638	443	1025	0	52	0	128

Ilustración 34:Resultados del dataset I

frame.protocols_4_http	frame.protocols_4_data	tcp.flags.push	frame.protocols_3_udp	tcp.flags.fin	tcp.flags.syn	ip.src	ip.dst	protocol
0	0	False	0	False	False	162.159.130.234	192.168.1.215	6
0	0	False	0	False	False	192.168.1.215	162.159.130.234	6
0	0	False	0	False	False	192.168.1.215	54.156.108.240	6
0	0	False	0	False	False	54.156.108.240	192.168.1.215	6
0	0	False	0	False	False	192.168.1.215	162.247.243.29	6
0	0	False	0	False	False	162.247.243.29	192.168.1.215	6
0	0	False	0	False	False	192.168.1.215	162.247.241.14	6
0	0	False	0	False	False	162.247.241.14	192.168.1.215	6
0	0	False	0	False	False	162.247.243.29	192.168.1.215	6
0	0	False	0	False	False	192.168.1.215	162.247.243.29	6

Ilustración 35:Resultados del dataset II

Podemos observar que se obtienen 10 paquetes, en este caso son todos TCP, se puede observar en “protocol”, que es 6 (TCP), en este caso como los 10 paquetes analizados han sido TCP entonces se añaden al dataset todos los paquetes, sin embargo, si estos paquetes fueran de un protocolo diferente al TCP o al UDP no serían añadidos al dataset. Por ejemplo, si hay 10 paquetes y dos de ellos no son TCP o UDP solamente se añadirían 8 al dataset.

Una vez desarrollada la aplicación voy a modificarla para que se ejecute en bucle y así poder analizar en tiempo real, para ello lo primero que debo hacer es exportar la red neuronal entrenada anteriormente para así poder utilizarla dentro de este programa para detectar si es o no un ataque DDoS, para ello implemento la siguiente línea en el programa.

```
save_model(model, 'modelo_red_neuronal.keras')
```

Ilustración 36: Guardar el modelo

Tras ello, antes de implementarla dentro del programa de detección de paquetes debo hacer que se ejecute infinitamente, para ello modifiko el código ya existente para que llame infinitamente a la función definida para la detección de paquetes, en el siguiente video se puede observar el correcto funcionamiento:



Ahora si se añadirá la red neuronal entrenada previamente y una vez implementada esta ya se podrá abordar el problema de cómo hacer para poder detectar cuando realmente se está produciendo un ataque y evitar falsos positivos.

```
model = load_model('modelo_red_neuronal.keras')
```

Ilustración 37:Carga del modelo

Para evitar los falsos positivos voy a hacer que antes de que se analice el dataset creado en tiempo real se llene con 100 paquetes, para así tener un tamaño razonable de dataset y el modelo entrenado evite realizar valoraciones anticipadas y una vez se tiene esa base inicial se analiza en cada paquete un dataframe de los 100 paquetes más actuales, si supera un umbral que definiré como 0,85(asigno un valor elevado para evitar que se detecte un ataque cuando es tráfico normal) entonces se detectará que se está realizando un ataque y se imprimirá un mensaje de alerta.

```
if len(packet_data) >= 100:
    df_last_100 = df[-100:]
    y_pred_proba = model.predict(df_last_100)
    proba = np.mean(y_pred_proba)

    if proba > 0.85:
        print(";Ataque DDoS detectado!")
```

Ilustración 38:Detección del ataque

Una vez implementado el código ya se puede probar su ejecución, que efectivamente dentro de mi red no detecta ningún ataque ddos.

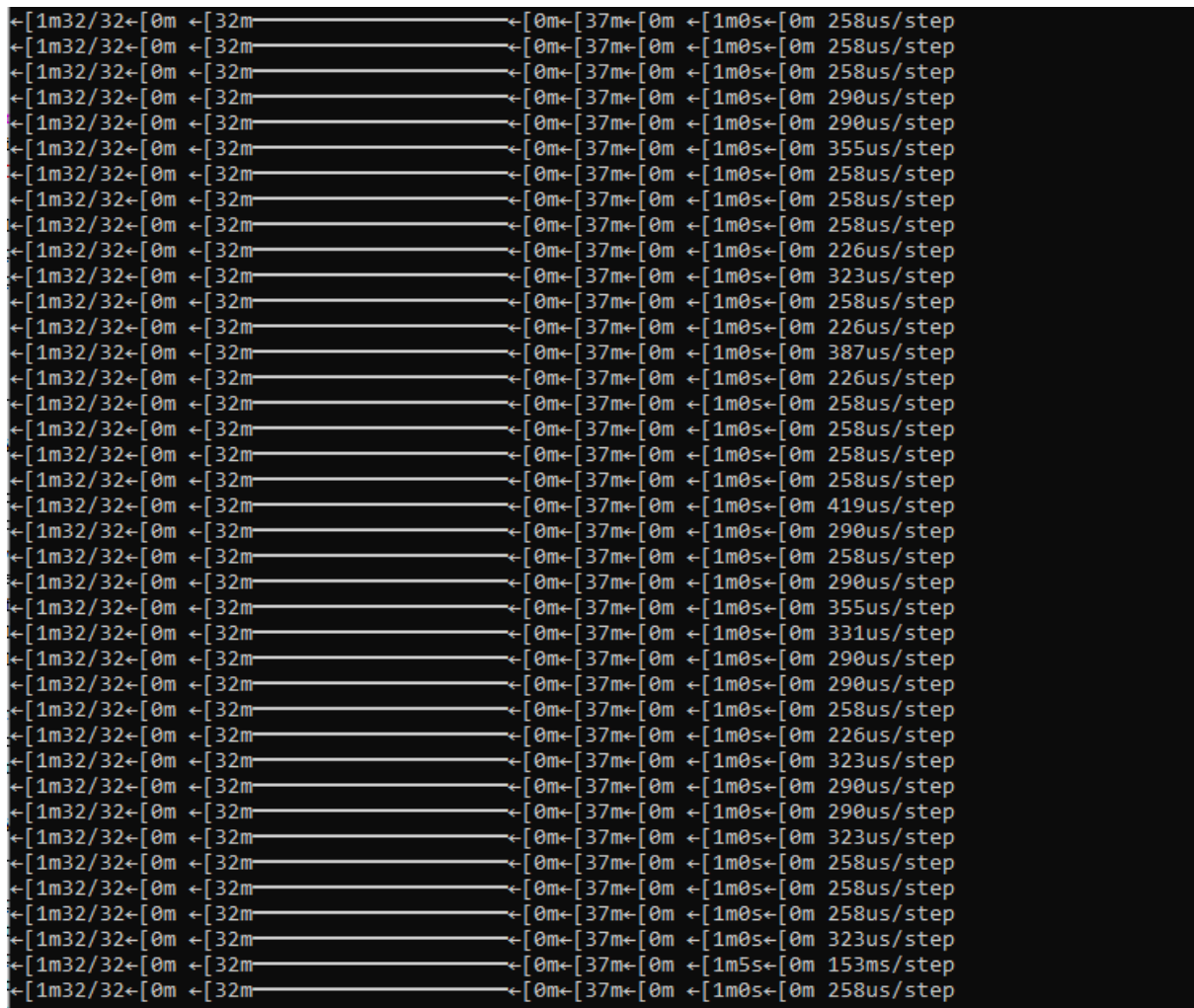


Ilustración 39:Ejecucion del programa

4.6.3. Notificación por web

Por último, se van a realizar una serie de modificaciones en el programa para que este al ejecutarse cree una página web sencilla en la que se indique si se está realizando un ataque o no en tiempo real, esto se hace mediante la herramienta de Flask.

Primero, se ha importado la librería de Flask al proyecto que permitirá crear una página web que muestre el estado del sistema en tiempo real. Para ello se ha utilizado “render_template_string” de Flask para poder añadir código HTML directamente en el programa.

```

@app.route('/')
def index():
    global attack_detected
    return render_template_string('''
        <!doctype html>
        <html>
        <head>
            <title>Detección de DDoS</title>
        </head>
        <body>
            <h1>Estado del Sistema</h1>
            {% if attack_detected %}
                <p style="color: red;">Ataque DDoS detectado!</p>
            {% else %}
                <p style="color: green;">No se han detectado ataques.</p>
            {% endif %}
        </body>
        </html>
    ''', attack_detected=attack_detected)

```

Ilustración 40: Creación de página web

La página mostrará un mensaje indicando si se ha detectado un ataque DDoS o no. Si se detecta un ataque, el mensaje se muestra en rojo; de lo contrario, se muestra en verde. Esta página web proporciona una interfaz visual simple y directa para monitorizar el estado del sistema.

Para poder ejecutar el código de la página web a la vez que el de detección de ataques DDoS necesitaremos crear un sistema multihilo, para ello se ha utilizado el módulo `threading` de Python. Se ha creado un hilo separado para ejecutar el proceso de sniffing de paquetes con Scapy, mientras que el servidor web de Flask se ejecuta en el hilo principal. Esto asegura que ambas tareas puedan correr al mismo tiempo sin bloquearse mutuamente y las notificaciones llegaran de forma correcta a la página web.

```

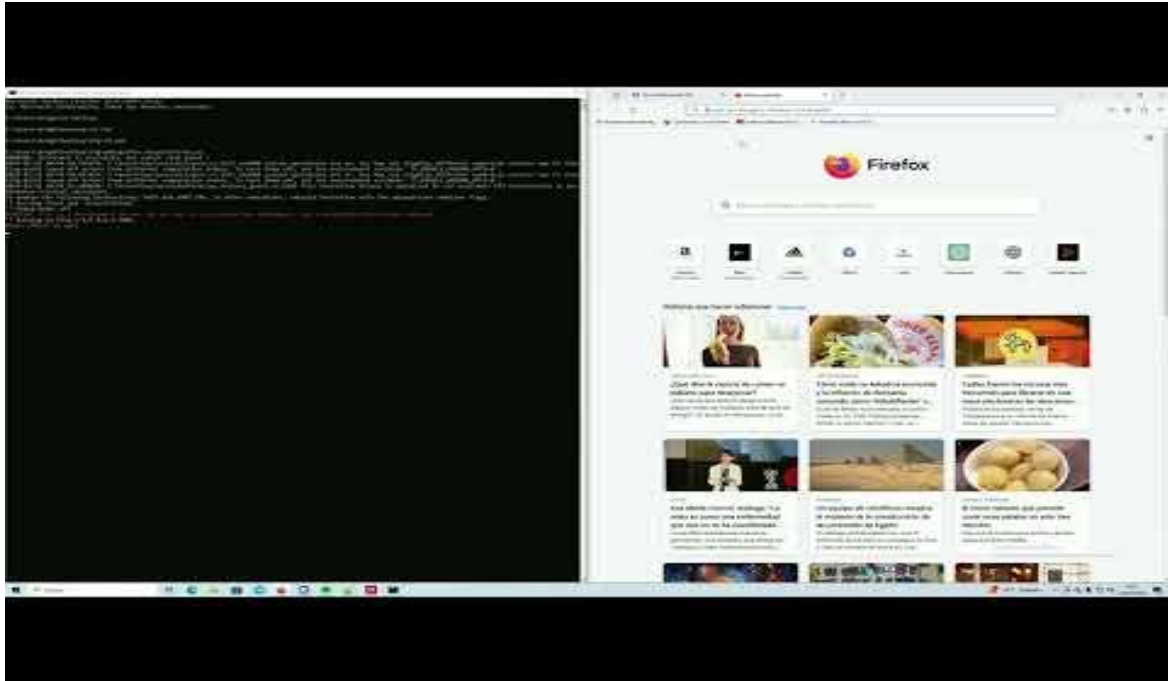
def start_sniffing():
    sniff(prn=packet_handler, store=False)

if __name__ == '__main__':
    import threading
    sniff_thread = threading.Thread(target=start_sniffing)
    sniff_thread.start()
    app.run(port=5000)

```

Ilustración 41: Threading para el programa y la web

En el siguiente vídeo se puede observar una ejecución del programa en un entorno, donde no hay tráfico de red malicioso, una vez añadida esta última parte del programa.



Capítulo 5

Pruebas y resultados

5.1. Pruebas

En este apartado se realizarán las pruebas para comprobar si la aplicación desarrollada funciona.

5.1.1. Prueba del sistema de detección tiempo real

Ahora para probar si realmente funciona con ataques DDoS se va a utilizar dos máquinas una de Linux para atacar de otra máquina de Windows para probar si se detecta el ataque, estas están configuradas para estar en la misma red, esto lo realizaré mediante el uso de una máquina virtual proporcionada por Ramón Alcarria equipadas para poder realizar ataques de negación de servicio, usaré la herramienta de ataque hulk. Hulk utiliza una gran cantidad de conexiones TCP o UDP (el tipo de paquetes que reconoce la aplicación) para abrumar el objetivo con un gran volumen de tráfico, lo que resulta en una pérdida de servicio para el objetivo.

Para realizar el ataque inicio la máquina virtual atacante y la víctima. Una vez iniciadas, en la victima iniciaré el servidor que recibirá el ataque, para ello haré uso de la herramienta node.js que me permitirá crear un servidor sencillo alojado en mi red para atacar sobre este servidor que alojaré en el puerto 3000.

Para crear el servidor primero inicializaré un proyecto de Node.js, mediante el comando de **npm init -y**, este creará el esqueleto del programa.

```
C:\Users\drngb\Desktop\tfg\mi-servidor>npm init -y
Wrote to C:\Users\drngb\Desktop\tfg\mi-servidor\package.json:

{
  "name": "mi-servidor",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

Ilustración 42: Inicialización de proyecto node.js

Una vez creado ahora crearemos un archivo .js llamado server.js donde añadiré el código para iniciar el servidor en el puerto 3000 y en este servidor se podrá realizar el ataque.

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('Servidor funcionando');
});

app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```

Ilustración 43: Servidor node.js

Una vez creado el código ya puedo ejecutar el código que alojará el servidor en el puerto en el que se podrá entrar para verificar su existencia mediante <http://localhost:3000/> donde se puede observar que funciona el código.



Ilustración 44: Servidor arrancando

Una vez se ha levantado el servidor se puede realizar el ataque mediante la herramienta de "hulk" desde la otra máquina virtual, antes de iniciar el ataque se ejecuta slow-http-test sobre el puerto alojado en la máquina víctima para poder

monitorizar el estado de la máquina donde se puede observar que efectivamente el servidor se encuentra funcionando correctamente.

```

syshield@syshield01: ~/Desktop/ataques/slowhttptest-master
File Edit View Search Terminal Help

slowhttptest version 1.6
- https://code.google.com/p/slowhttptest/ -
test type: SLOW HEADERS
number of connections: 4090
URL: http://192.168.1.130:3000/
verb: GET
Content-Length header value: 4096
follow up data max size: 52
interval between follow up data: 10 seconds
connections per seconds: 30
probe connection timeout: 3 seconds
test duration: 240 seconds
using proxy: no proxy

Tue May 28 18:52:52 2024:
slow HTTP test status on 10th second:

initializing: 0
pending: 2
connected: 251
error: 0
closed: 0
service available: YES

```

Ilustración 45: Monitorización del servidor

Una vez se está monitorizando el servidor se puede lanzar el ataque que realizará un envío masivo de paquetes al puerto lo que puede propiciar su caída, una vez ejecutado el ataque “hulk” en cierto momento puede llegarse a dar un colapso del servidor atacado, aunque tras un periodo de tiempo el servidor se recupera de la caída y vuelve a estar accesible.

```

syshield@syshield01: ~/Desktop/ataques/slowhttptest-master
File Edit View Search Terminal Help
- https://code.google.com/p/slowhttptest/ -
test type: SLOW HEADERS
number of connections: 4090
URL: http://192.168.1.130:3000/
verb: GET
Content-Length header value: 4096
follow up data max size: 52
interval between follow up data: 10 seconds
connections per seconds: 30
probe connection timeout: 3 seconds
test duration: 240 seconds
using proxy: no proxy

Tue May 28 18:56:22 2024:
slow HTTP test status on 220th second:

initializing: 0
pending: 0
connected: 85
error: 0
closed: 4005
service available: NO

syshield@syshield01: ~/Desktop/ataques/hulk-master
base) syshield@syshield01:~/Desktop/ataques/hulk-master$ sudo python hulk.py
/192.168.1.130:3000
[sudo] password for syshield:
python: can't open file 'hulk': [Errno 2] No such file or directory
base) syshield@syshield01:~/Desktop/ataques/hulk-master$ sudo python hulk.py
/192.168.1.130:3000
- HULK Attack Started --
160 Requests Sent
422 Requests Sent

```

Ilustración 46: Ataque tumba el servidor

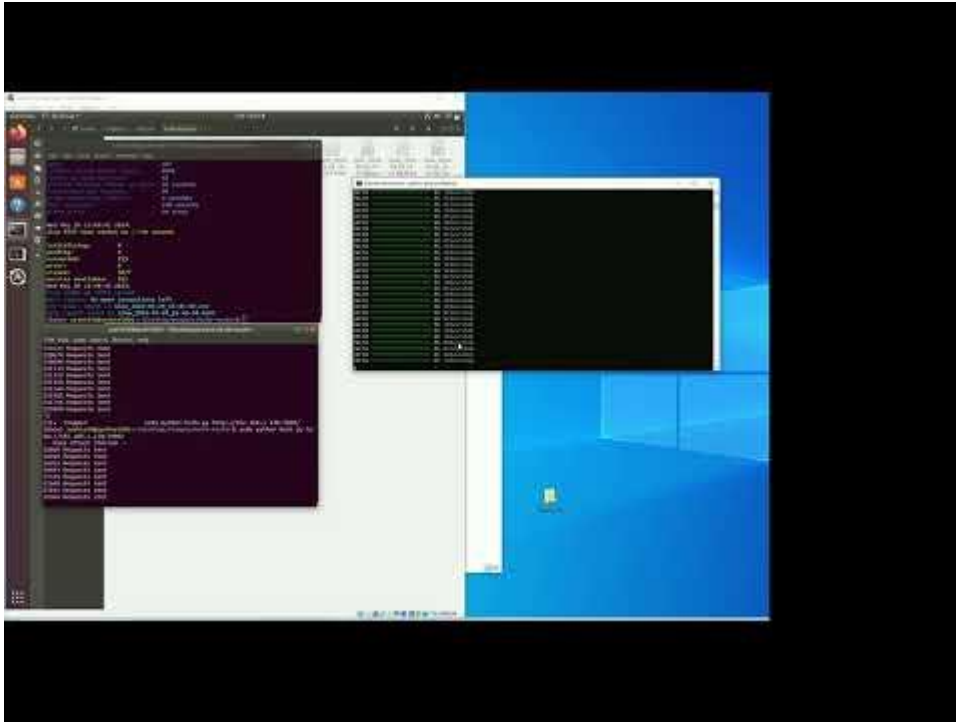
Una vez podemos realizar ataques a la red ya se podrá monitorizar el tráfico para comprobar si se detectan ataques iniciando la aplicación desarrollada previamente para analizar paquetes de red y descubrir si son maliciosos.

Primero se realizan pruebas con el proyecto tal y como estaba inicialmente, analizando 100 paquetes y haciendo la media de los valores obtenidos por la red neuronal previamente entrenada analizando estos paquetes, sin embargo, esta prueba resultó en un fracaso ya que a pesar de que detecte que se está realizando un ataque DDoS en inicio tras ello, aunque deje de realizarse el ataque al tener un dataset de 100 paquetes habría que esperar un amplio periodo de tiempo para que se genere un numero de paquetes lo suficientemente grande de tráfico normal para que la media de estos paquetes resulte en que detecte tráfico normal por lo tanto para observar si el ataque DDoS ha iniciado o ha acabado hay que esperar un momento a que se procesen nuevos paquetes.

Para minimizar la espera sin eliminar la fiabilidad del programa se han bajado los paquetes obtenidos para mediar ligeramente haciendo que sea un número suficiente de paquetes para esto critica, pero reduciendo la espera para evitar detecciones tardías.

5.2. Resultados

A continuación, se puede observar una ejecución del programa mientras se realizan los ataques previamente descritos en las pruebas.



Se comprueba que efectivamente la aplicación es capaz de reconocer cuando se están realizando ataques contra el equipo y cuando estos ataques dejan de hacerse, aunque no los detecta en el momento ya que como se ha dicho en el apartado anterior tardan en obtener los suficientes paquetes como para que la media de estos supere el umbral de detección.

Capítulo 6

Conclusiones y trabajos futuros

6.1. Conclusiones

6.1.1. Introducción

En este proyecto se propone una solución para un sistema capaz de monitorear el tráfico de red en tiempo real para identificar ataques de denegación de servicio. La detección de estos ataques se basa en un modelo de inteligencia artificial que realiza predicciones sobre la presencia de dichos ataques. Para construir el modelo, se realizaron múltiples experimentos, logrando una versión optimizada que alcanzó métricas satisfactorias, como una precisión del 90%.

6.2. Impacto social, medioambiental y económico

El impacto medioambiental de este proyecto es nulo debido a que el proyecto en sí no contribuye al medioambiente, aunque cabe destacar que detectar a tiempo un ataque DDoS si puede tener consecuencias medioambientales positivas ya que los servidores y dispositivos afectados por un ataque DDoS deben manejar un volumen de tráfico mucho mayor de lo normal, lo que incrementa su consumo de energía, además que si un equipo es dañado puede necesitar de una reparación o ser reemplazados lo cual generaría un despilfarro de recursos.

En cuanto a impactos sociales si se está contribuyendo, concretamente en la ODS 16 (Paz, justicia e instituciones) debido a que el uso de tecnologías avanzadas

como el machine learning para la detección y prevención de ciberataques ayuda a preservar la paz en el ciberespacio.

Por último, el ámbito donde más impacto tiene este proyecto es el económico ya que prevenir a tiempo un ataque DDoS permite actuar para evitar que este dañe máquinas que pueden contener información o ser necesarias para prestar servicios lo que generaría una pérdida en el momento y en gastos de reparación muy altos, pero si gracias a la prevención estos gastos pueden reducirse o eliminarse en el caso que se consiga evitar que el ataque dañe la infraestructura.

6.3. Trabajo futuro

A pesar de que se han obtenido los resultados que se buscaban todavía hay mejoras que se pueden realizar en la aplicación.

Unas de las cosas que se pueden mejorar respecto al trabajo es la limitación de tipos de ataques que puede detectar, esto es debido a que únicamente detecta paquetes TCP y UDP, así que en el futuro se podría entrenar una red neuronal con información de más tipos de ataques ya que el conjunto de datos utilizado no contaba con más información.

La aplicación también podría sufrir mejoras en el apartado estético como en la página web, donde se podría hacer un sistema de monitorización en tiempo real más claro donde se pudieran observar las alarmas de ataque generadas en el pasado.

Por último, se podría también intentar hacer que el tiempo que se tarda en la detección del ataque fuera más reducido para poder reaccionar con el menor retraso posible.

CÓDIGO

<https://github.com/PatoDDM/tfg>

Bibliografía

Arshi, M., Nasreen, & Madhavi, K. (2020). A Survey of DDOS Attacks Using Machine Learning Techniques. *E3S Web Of Conferences*, 184, 01052.

<https://doi.org/10.1051/e3sconf/202018401052>

ddos_attack_Prevention. (2020, 7 diciembre). Kaggle.

https://www.kaggle.com/datasets/preeti5607/ddos-attack-prevention?select=dataset_attack.csv

Arham, M. (s. f.). *Pandas: How to One-Hot Encode Data - KDnuggets*. KDnuggets.

<https://www.kdnuggets.com/2023/07/pandas-onehot-encode-data.html>

Machine Learning con Python y Scikitlearn. (s. f.).

https://cienciadedatos.net/documentos/py06_machine_learning_python_scikitlearn.html

DDoS Attacks Detection and Classification based on Deep Learning Model. (2023, junio).

https://www.researchgate.net/publication/371689645_DDoS_Attacks_Detection_and_Classification_based_on_Deep_Learning_Model.

Gamboa, J. A., Arroyave, J. C., & Unamuno, E. A. (2022). *Detección de Ataque de DDoS utilizando Machine Learning – algoritmo de Random Forest*. Dialnet.

<https://dialnet.unirioja.es/servlet/articulo?codigo=8590684>

Classification of large datasets using Random Forest Algorithm in various applications: Survey. (2014, septiembre).

https://faculty.ksu.edu.sa/sites/default/files/classification_of_large_datasets_using_random.pdf.

Prototipo de Red Neuronal Profunda Aplicada en la Ciberseguridad. (2023).

https://repositorio.ucaldas.edu.co/bitstream/handle/ucaldas/19449/FabianAlberto_RamirezRodriguez_2023.pdf?sequence=3&isAllowed=y.

Wang, M., Lu, Y., & Qin, J. (2020c). A dynamic MLP-based DDoS attack detection method using feature selection and feedback. *Computers & Security*, 88, 101645.

<https://doi.org/10.1016/j.cose.2019.101645>

Team, K. (s. f.). *Keras documentation: The Sequential model*.

https://keras.io/guides/sequential_model/

Team, K. (s. f.-a). *Keras documentation: Models API*. <https://keras.io/api/models/>

Sharma, S. (2022, 20 noviembre). Activation Functions in Neural Networks - Towards Data Science. *Medium*. <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

Krishnamurthy, B. (2024, 26 febrero). *An Introduction to the ReLU Activation Function*. Built In. <https://builtin.com/machine-learning/relu-activation-function>

Welcome to Scapy's documentation! — Scapy 2.6.0 documentation. (s. f.-b).

<https://scapy.readthedocs.io/en/latest/>

Scapy. (s. f.). <https://scapy.net/>

Daniel. (2023, 30 octubre). *Random Forest: Bosque aleatorio. Definición y funcionamiento.* Formación En Ciencia de Datos | DataScientest.com. <https://datascientest.com/es/random-forest-bosque-aleatorio-definicion-y-funcionamiento>

Akamai. (s. f.). *La rápida evolución y la creciente amenaza de los ataques DDoS.* <https://www.akamai.com/site/es/documents/white-paper/ddos-attacks-evolution-white-paper.pdf>.

CloudFlare. (s. f.). <https://www.cloudflare.com/es-es/learning/ddos/famous-ddos-attacks/>. Ataques DDoS Más Conocidos | los Mayores Ataques DDoS de la Historia.

Hasson, E. (2023, 20 diciembre). *DDOS attack types & mitigation Methods* | Imperva. Learning Center. <https://www.imperva.com/learn/ddos/ddos-attacks/>

Radware. (s. f.). *What is a UDP Flood DDoS Attack?* | Radware. <https://es.radware.com/security/ddos-knowledge-center/ddospedia/udp-flood/>

Welcome to Flask — Flask Documentation (3.0.x). (s. f.). <https://flask.palletsprojects.com/en/3.0.x/>