IceWall

Trabajo de Fin de Grado
Curso 2023–2024

Autor
Santiago Mourenza Rivero

Director
Jose Luis Vazquez-Poletti
Juan Carlos Fabero Jiménez

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

IceWall

Trabajo de Fin de Grado en Ingeniería Informática

Autor
Santiago Mourenza Rivero

Director
Jose Luis Vazquez-Poletti
Juan Carlos Fabero Jiménez

Convocatoria: *Junio 2024*

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

14 de Mayo de 2024

Dedicatoria

A mi amigo Miguel Ángel por estar siempre cuándo lo he necesitado. Y a Claudia por leerse este documento y acompañarme mientras lo redactaba.

Agradecimientos

A mis padres por apoyarme durante estos 5 años de carrera, incluso en los momentos dónde aguantarme fue una ardua tarea. A mi hermano por tener que soportar a un compañero de piso que pasa de cantante a organizador de eventos sociales en un segundo. A mis amigos por darme momentos en los que dejar de pensar en todo el trabajo que tenía que hacer y darme ánimos para continuar. Y por último a mí, que incluso con todo en contra y mil dudas he conseguido llegar hasta aquí que, al fin y al cabo, es sólo el inicio de lo que me queda por conseguir.

Resumen

IceWall

En este documento se van a abordar cuestiones relativas a la seguridad informática para conseguir crear un modelo basado en inteligencia artificial que ayude en la tarea de detectar y frenar ataques en redes. Para conseguir este objetivo, se va a utilizar un conjunto de datos para entrenar un modelo que dirá que tipo de ataque está ocurriendo para elegir un postproceso adecuado para frenar dicho ataque.

Palabras clave

Seguridad, Redes, *Sklearn*, PortScan, Botnet, DDoS, CICIDS2017

Abstract

This document will approach issues related to cybersecurity in order to create an artificial intelligence-based model that assists in detecting and stopping network attacks. To achieve this goal, a dataset will be used to train a model that identifies the type of the attack occurring, enabling the selection of an appropiate post-processing method to stop said attack.

Keywords

Security, Networks, *Sklearn*, *PortScan*, *Botnet*, *DDoS*, CICIDS2017

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

"No vive el que no vive seguro"
— Francisco de Quevedo

En el mundo de la seguridad y las redes, hay infinidad de procesos que los usuarios no comprenden pero asumen que funcionan correctamente y hacen uso de ellos sin pararse a pensar en la cantidad de cosas que podrían fallar o las vulnerabilidades que personas con intenciones maliciosas pueden explotar. Ataques como el DDoS *(Distributed Denial of Service)* se cobran víctimas a diario haciendo que diferentes empresas, como entidades bancarias, sufran una denegación de servicio. Ataques de este tipo constan de una fase previa dónde se infectan diferentes equipos *(botnet)* para hacer un ataque coordinado consiguiendo así privar de servicio a la entidad atacada. Generalmente el usuario del equipo infectado no es consciente de que su dispositivo forma parte de una *botnet*.

En este ejemplo, y en tantos otros, no deja de ser el desconocimiento el causante de que el ataque sea exitoso. Está claro que partimos de la premisa de que no van a dejar de existir estos ataques, por lo que la idea es conseguir una defensa lo suficientemente buena para evitar que sean llevados a cabo. Para ello existen herramientas como cortafuegos o *IDS* (*Intrusion Detection Systems*) que nos permiten filtrar ataques de red de manera que, por mucho que se intente, no dejan pasar los paquetes que conseguirían el objetivo del ataque.

El problema que tienen estas herramientas es que, hasta cierto punto, no son autónomas ya que tiene que haber expertos que revisen tanto que las reglas de filtrado son correctas como los registros de trazas. Esto puede ocasionar que se cometan algunos errores que podrían ser solucionados con algún tipo de automatización, ya sea por la eficiencia del método aplicado o por simplificar la detección de ataques para así ayudar al tratamiento a posteriori de los datos de un ataque. Estos errores pueden ser de múltiples tipos, desde que el experto pueda no darse cuenta de detalles cruciales que terminen generando vulnerabilidades en un sistema, hasta fallos en las herramientas difícilmente previsibles.

1.1. Objetivos

En este trabajo vamos a limitarnos a buscar la manera de mejorar la defensa frente a diferentes ataques de red. Para ello será importante conocer datos, como los ataques sus características, así como maneras de defenderse para evitar que resulten críticos. Esta materia es algo que se lleva estudiando desde hace mucho tiempo pero siempre es posible dar nuevos enfoques de cara a mejorar lo que ya existe. Es por ello por lo que como objetivos principales se va a buscar:

- Revisar y analizar los tipos de ataques y formas de mitigarlos.
- Entrenar un modelo de *machine learning* que permita identificar qué tipo de ataque se está produciendo.
- Crear códigos de postprocesado que permitan defender de uno o dos de los posibles ataques.
- Comprobar la utilidad o no de una herramienta de este estilo, en caso de ser posible.

1.2. Plan de trabajo

Para llegar a los objetivos propuestos se va a seguir un plan basado en diferentes fases de desarrollo. Estas fases son las siguientes:

- **Fase 1 - Datos y preproceso:** Durante esta etapa se escoge un conjunto de datos y se le hace un preproceso para que sea correcto de cara a poder entrenar un modelo con algoritmos de *MachineLearning* utilizando la libería *Sklearn*.
- **Fase 2 - Entrenamiento y decisiones de parámetros:** En esta segunda etapa se pretende hacer diversos entrenamientos mediante el conjunto de datos preprocesado para ver que parámetros son los que mejor modelo predictivo generan. Una vez con ello se guardará dicho entrenamiento para poder hacer predicciones de ataques basados en nuevos conjuntos de datos.
- **Fase 3 - Postproceso de PortScan y DDoS:** Una vez detectado el tipo de ataque se le hará un postproceso para mitigar el ataque. En este caso se van a implementar dos códigos de postproceso, uno para PortScan y otro para DDoS.

Se han elegido estos dos tipos de ataque por ser ejemplos sencillos que sirven para ejemplificar la idea que se pretende implementar de forma integral. En el caso de PortScan o exploración de puertos se espera detectar qué máquina intentando comprobar el estado de los puertos y bloquearle la posibilidad de acceso a esta información. Por otro lado en DDoS o denegación de servicio distribuída se espera frenar la llegada de múltiples paquetes para evitar que el servicio colapse.

Chapter 2

Introduction and Objectives

"One does not truly live if one does not live securely"
— Francisco de Quevedo

In the realm of security and networking, there are countless processes that users may not fully understand but often take for granted, assuming they will work correctly. Without considering the numerous potential failures or vulnerabilities, users unknowly expose themselves to risks that malicious actors can exploit. One such risk involves Distributed Denial of Service (DDoS) attacks, which occur daily and have severe impacts on various entities, including banks, by disrupting their services. These attacks usually start with a preliminary phase where multiple devices are infected to form a botnet. This botnet then carries out a coordinated attack, crippling the targeted service. Typically, the owners of these infected devices are unaware that their devices are being used in such a manner.

In this case, and many others, the success of an attack is often due to a lack of knowledge. We operate under the assumption that such attacks will not cease. Therefore, the strategy is to develop a defense robust enough to stop these attacks. Tools like firewalls and Intrusion Detection Systems (IDS) are crucial for this purpose. They help filter network attacks in such a way that, no matter the effort performed, they prevent the transmission of packets that could fulfill the attack's objective.

These tools have, to a certain extent, a lack of autonomy because they require experts to verify the accuracy of filtering rules and to review trace logs. Such reliance can lead to errors that might be mitigated through automation, which would enhance the method's efficiency or simplify the process of attack detection, thus improving post-attack data analysis. The types of errors encountered can vary widely, from experts missing critical details that introduce system vulnerabilities, to unpredictable tool failures.

2.1. Requirements

In this study, we will explore methods to improve defenses against various network attacks. It will be crucial to collect information about the types of attacks, their characteristics, and defensive methods to prevent these from escalating. While this topic has been extensively studied, there is always room for innovative perspectives to enhance current methodologies. The main objectives of this exploration will be:

- Review and analyze the various types of network attacks and explore methods to mitigate them.
- Train a machine learning model to accurately identify the type of attack as it occurs.
- Develop post-processing codes aimed at defending against one or two specific potential attacks.
- Evaluate the utility and effectiveness of such a tool, if possible.

2.2. Workplan

To achieve the proposed objectives, a plan based on different developement phases will be followed. These phases are as follows:

- **Phase 1 - Data and Preprocessing:** During this stage, a dataset is preprocessed to ensure it is suitable for training a model using Machine Learning algorithms with the Sklearn library.
- **Phase 2 - Training and Parameter Selection:** In this second stage, various training sessions are conducted using the preprocessed dataset to determine which parameters produce the best predictive model. Once determined, this training will be saved to make predictions on new datasets.
- **Phase 3 - Post-procesing for PortScan and DDoS:** After detecting the type of attack, post-processing will be performed to mitigate the attack. In this case, two post-processing codes will be implemented, one for PortScan and another for DDoS.

The two types of attacks selected, PortScan and DDoS, can be used as examples to illustrate the broader concept planned for implementation. For PortScan attacks, the objective is to identify which machine is attempting to inspect the status of ports and subsequently block its access to this information. In contrast, for Distributed Denial of Service (DDoS) attacks, the goal is to prevent the influx of multiple packets simultaneously to avoid service disruption.

Capítulo 3

Estado de la cuestión

En este capítulo se va a explicar en qué estado se encuentran acualmente las tecnologías asociadas a la defensa de ataques de red y por qué la idea que se presenta es necesaria.

3.1. Ataques

En esta sección vamos a comentar los diferentes ataques más importantes para así comprender en un futuro cómo pueden ser bloqueados o buscar contramedidas más eficientes.

3.1.1. PortScan o exploración de puertos

El ataque PortScan o exploración de puertos consiste en que el atacante envía paquetes a un conjunto de puertos de la víctima. La respuesta a dichos paquetes da información al atacante del estado del puerto. Con esto puede aprender sobre el sistema y detectar así vulnerabilidades. Hay diferentes tipos de exploración de puertos, dependiendo de cómo se efectúen (Moreno Vozmediano, 2023c) (Fyodor, 2005) (Kostas, 2018).

- **Método *connect***: Este método consiste en establecer una conexión TCP con la víctima para después ser cerrada con un RST. Las posibles respuestas de la víctima son las siguientes:
  - Acepta la conexión, es decir, devuelve SYN-ACK. Esto nos indica que el puerto está abierto.
  - Rechaza la conexión, es decir, devuelve RST. Esto nos indica que el puerto está cerrado.
  - No responde nada o responde con ICMP destino inalcanzable, es decir, el puerto está filtrado por un cortafuegos.

Este método permite, por lo tanto, ver si un puerto está abierto, cerrado o filtrado, quedando registrada en la víctima, lo que hace el ataque fácilmente detectable. Esto se puede ver en la **Figura 3.1**.

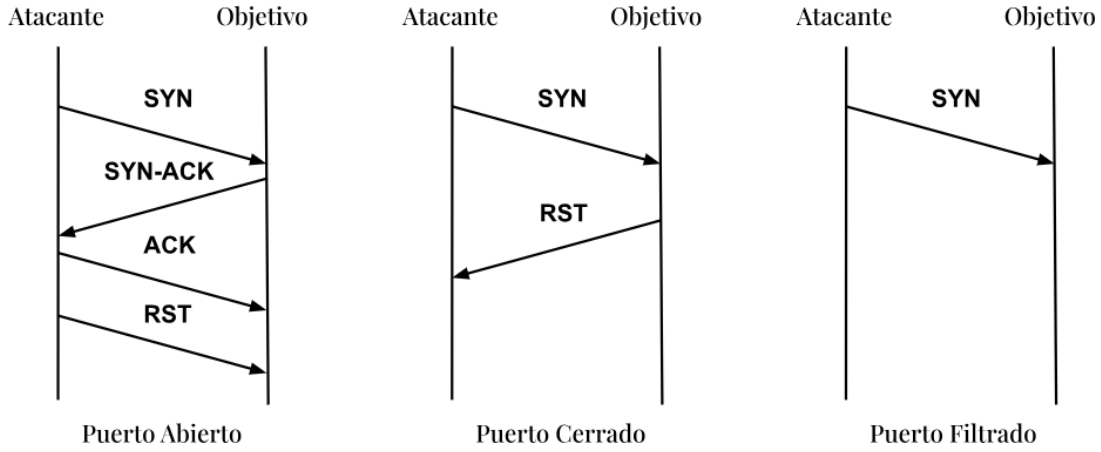

Figura 3.1: Ilustración del Método *Connect*

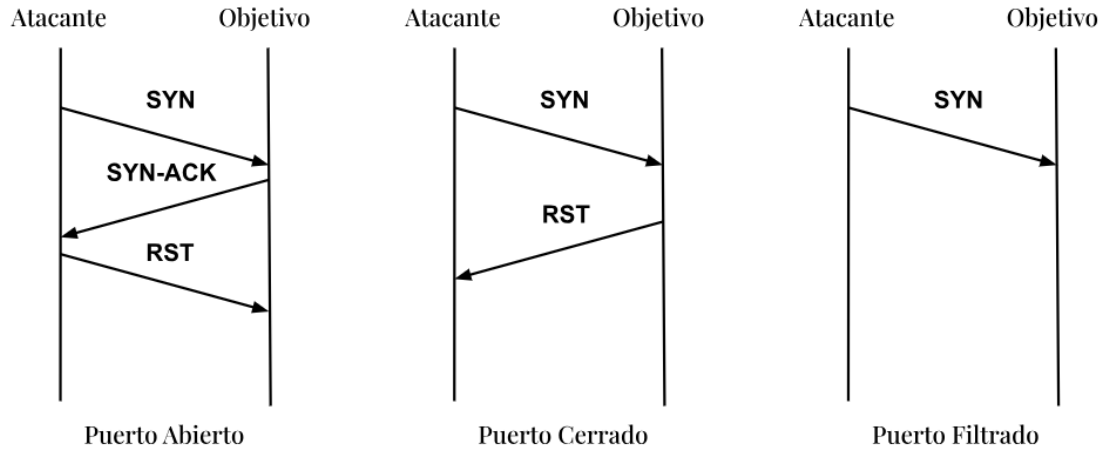

Figura 3.2: Ilustración del Método SYN *Stealth*

- **Método SYN *Stealth* o SYN Sigiloso**: Consiste en enviar un segmento con SYN = 1 a la víctima sin llegar a establecer la conexión TCP, para ello se aborta con RST antes de completar la conexión. Las posibles respuestas de la máquina objetivo son:
  - Acepta la conexión, devolviendo un SYN-ACK, como en el caso anterior, lo que nos dice que el puerto está abierto.
  - Rechaza la conexión, devolviendo un RST, lo que nos dice que el puerto está cerrado.
  - No responde o responde con ICMP destino inalcanzable, lo que indica que el puerto está filtrado por un cortafuegos.

Este método es equivalente al anterior en cuanto a la información que se obtiene, no obstante, no deja rastro de actividad en la víctima. Podemos ver este método en la **Figura 3.2**.

- **Método ACK**: Este método se efectúa enviando un segmento con ACK = 1 a la víctima, sin intentar establecer una conexión previamente. La víctima puede responder de las siguientes maneras:

  - Devuelve un RST, entonces el puerto está abierto o cerrado.
  - No responde o responde con ICMP destino inalcanzable, entonces el puerto está filtrado por un cortafuegos.

  Es útil para saber si un puerto está filtrado o no por un cortafuegos sin dejar rastro en la víctima. Podemos ver en detalle en la **Figura 3.3**.

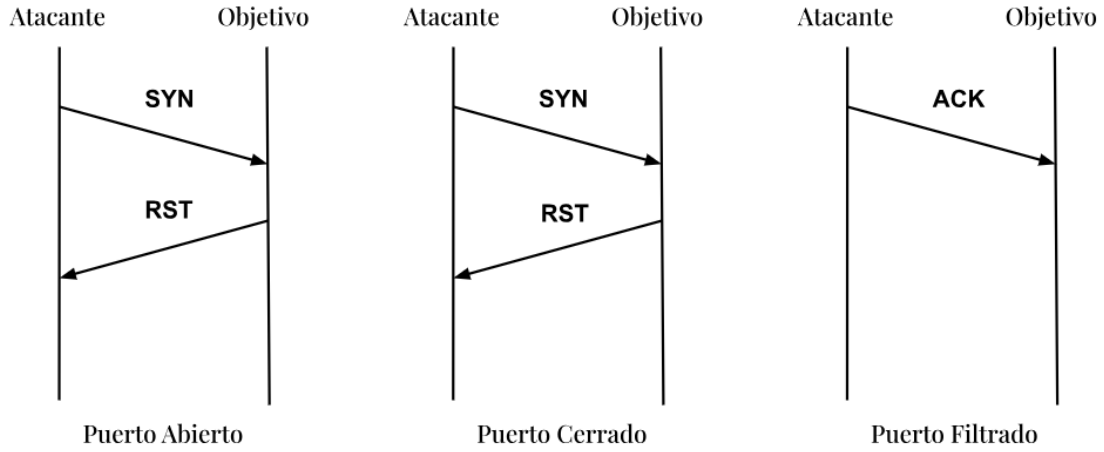

Figura 3.3: Ilustración del Método ACK

- **Método FIN**: Para este método se envía un segmento con FIN = 1 a la máquina objetivo, sin intentar establecer una conexión previamente. Las posibles respuestas son:

  - Devuelve RST, entonces el puerto está cerrado.
  - No responde, entonces el puerto está o bien abierto o bien filtrado por una cortafuegos.
  - Si devuelve un ICMP destino inalcanzable, entonces el puerto está filtrado por un cortafuegos.

  Este método resulta útil para comprobar si un puerto está cerrado. Combinado con el ya mencionado ACK para distinguir entre abierto y filtrado. Este tampoco deja rastro en la víctima. Información relevante del método puede verse en la **Figura 3.4**.

- **Método NULL**: Este método consiste en enviar un segmento con todos los *flags* desactivados a la víctima, véase la **Figura 3.5**. Guarda relación con el método FIN al funcionar de la misma manera. Más adelante veremos el porqué puede usarse como alternativa a FIN.
- **Método XMAS**: Para este método se envía un segmento con los flags FIN, PSH y URG a la víctima, véase la **Figura 3.6**. Este también guarda relación con los dos métodos anteriores por tener las mismas posibles respuestas.

Estos últimos tres métodos se utilizarán dependiendo de la máquina que se está atacando. Se utilizan para atravesar cortafuegos que no hagan inspección de estados o encaminadores que hagan filtrado de paquetes. Aunque respondan de la misma manera hay que tener en cuenta que podría ocurrir que una máquina envíe o no RST dependiendo de los *flags* que estén activados. De todos modos estos métodos generan muchos falsos positivos dado que es común que se envíen RST incluso si el puerto está abierto.

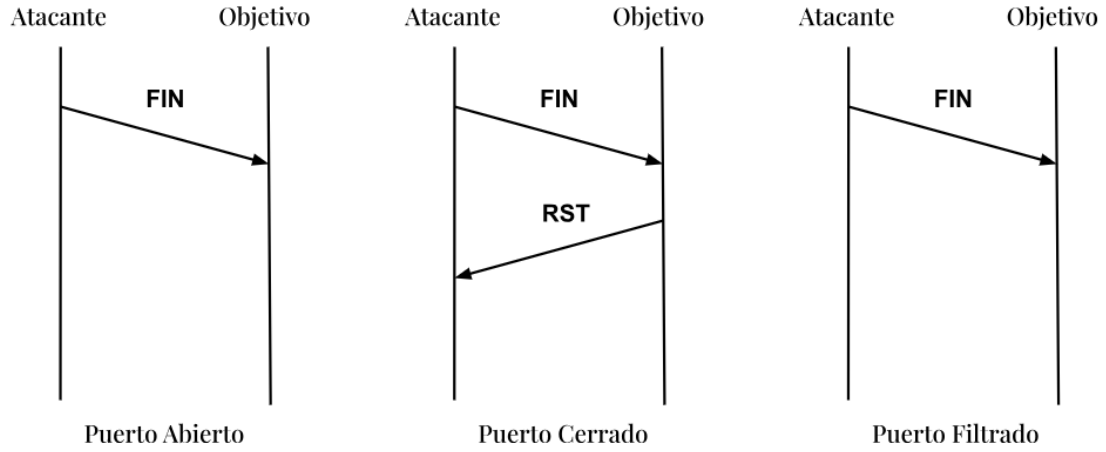

Figura 3.4: Ilustración del Método FIN

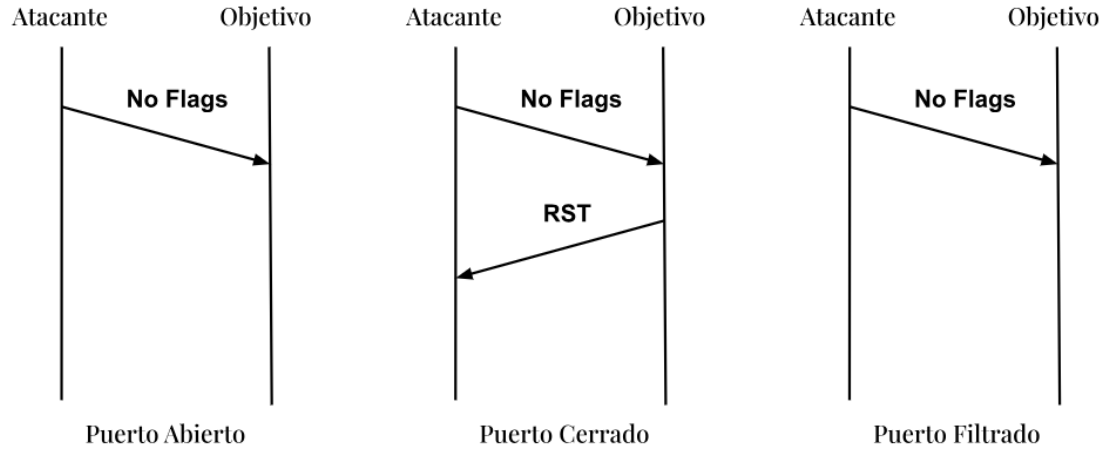

Figura 3.5: Ilustración del Método NULL

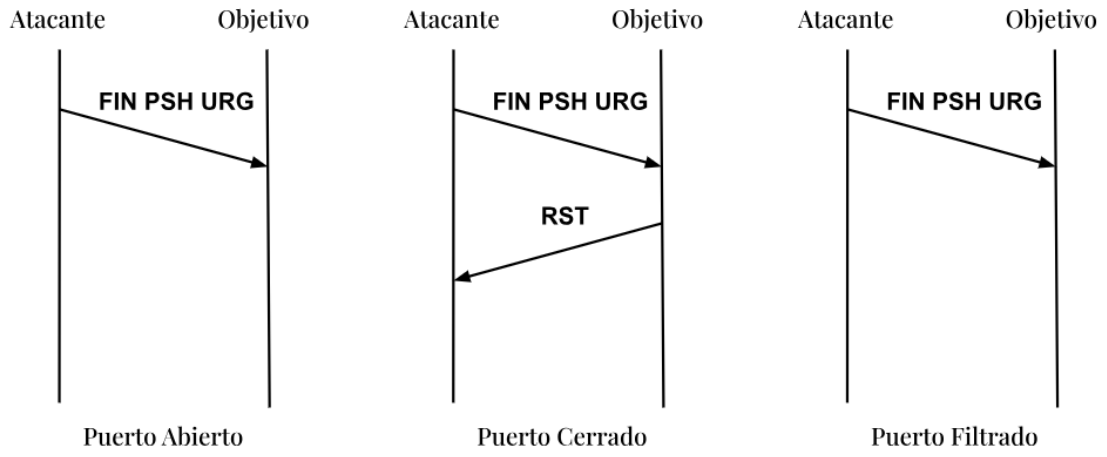

Figura 3.6: Ilustración del Método XMAS

Una de las herramientas más utilizadas para este tipo de ataque se conoce como nmap (Fyodor, 2005). Dependiendo de los parámetros que se introduzcan, estaríamos realizando uno de los métodos ya comentados. Por ejemplo, si se quiere hacer un ataque con el método ACK la regla a introducir sería la siguiente:

nmap -sA -p 80 192.168.0.10

Con esto se esta lanzando un ataque al puerto 80 de la máquina 192.168.0.10. Es importante también decir que se hacen este tipo de ataques a los puertos UDP, pero el funcionamiento es distinto. También con nmap se pueden hacer estos ataques si se usa el parámetro **-sU**. Este ataque es menos fiable que el TCP debido a la falta de confirmación de la conexión.

3.1.2. DDoS (Distributed Denial of Service)

Los ataques DDoS o de denegación de servicio distribuida suelen ser ataques de tipo SYN *flooding* o *Connection Flooding* que se realizan desde múltiples atacantes de forma simultánea y coordinada sobre una misma máquina víctima. Antes de comentar el funcionamiento vamos a definir estos ataques que se han nombrado.

- **TCP SYN *flooding***:

  Un atacante envía una gran cantidad de segmentos SYN a la víctima. Esto llena la cola de conexiones incompletas e impide nuevas conexiones. Para ello el atacante suele utilizar una IP origen falsa para evitar recibir los SYN-ACK de la víctima. Una contramedida para este ataque es utlizar el mecanismo SYN *cookies*. Este método es similar al portolocolo handshake de 3 vías pero sin completar la conexión, podemos verlo en la **Figura 3.7**. Es importante que la dirección IP falsa utilizada esté inactiva, de lo contrario se enviarían RST al servidor atacado (Haris et al., 2010).

- **TCP *connection flooding***:

  Es similar al ataque que acabamos de ver pero en este las conexiones TCP se completan. Consumen más recursos de la víctima y como contraparte también del atacante ya que para completar una conexión no se puede utilizar una IP falsa, esto hace que el atacante sea detectable. Para evitar esto es necesario utilizar diferentes máquinas para este tipo de ataque, enmascarando así su identidad.

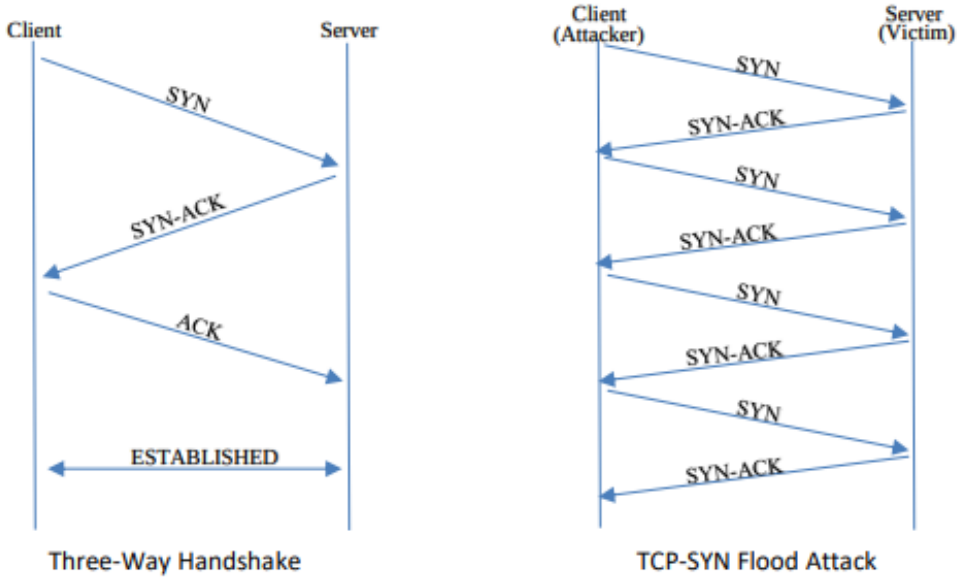

Figura 3.7: Comparación del Protocolo Handshake de 3 vías con el TCP SYN flooding. **Fuente**: Kostas (2018) página 9

.

Antes de continuar vamos a explicar de manera resumida el funcionamiento de SYN *cookies*. La forma en la que las SYN *cookies* frenan el ataque es utilizando una función que contiene cierta información del paquete SYN del cliente y del lado del servidor para calcular un número de secuencia inicial aleatorio. Se envía dicho número de secuencia en un SYN + ACK, en el caso de recibir un ACK con otro número de secuencia que, con datos y una función inversa se puede verificar que es válido, entonces se crea un TCB (Transmission Control Block) y se establece una conexión, se rechaza en caso contrario (Geeksforgeeks, 2021).

Una vez visto esto, lo que hace posible un ataque DDoS es un ataque o fase previa al mismo. Este ataque consiste en infectar múltiples máquinas en diferentes en localizaciones utilizando algún *malware* que permita el control de las mismas. Este ataque previo se conoce como *botnet*.

Una vez conformada se lanzan ataques de manera sincronizada desde estas máquinas a una misma víctima con la intención de bloquear el servicio que proporciona en base a saturarla con peticiones. Podemos hacernos una idea del funcionamiento con el diagrama de la **Figura 3.8**.

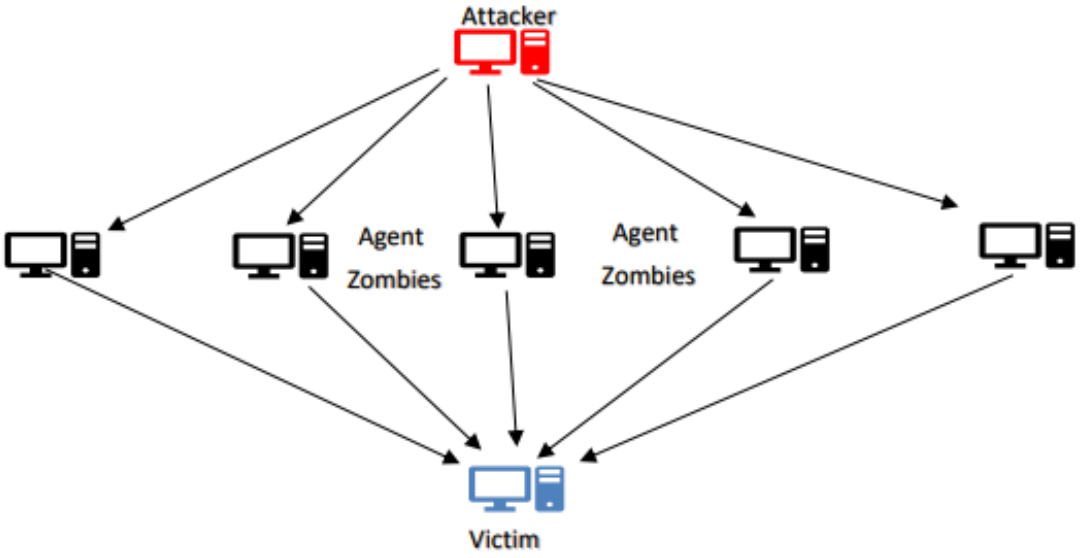

Figura 3.8: Diagrama del ataque DDoS. **Fuente**: Kostas (2018).

3.2. Contramedidas

Para los ataques que hemos visto hay contramedidas efectivas para evitar o bloquear los ataques. A continuación vamos a comentarlas poniendo especial atención en su posible uso posterior en la implementación de *IceWall*.

3.2.1. Cortafuegos

Un cortafuegos (Moreno Vozmediano, 2023a) es un sistema que separa una red del resto de internet. El objetivo es que únicamente permita pasar el trafico autorizado por diferentes reglas que han sido previamente configuradas. Vamos a ver a continuación las diferentes funcionalidades:

- **Filtrado de paquetes**: El cortafuegos protege la red controlando y limitando los paquetes de entrada y salida.
- **Registro de actividad**: Se registran las acciones realizadas con los paquetes que entran o salen de un equipo o de la propia red.

Desde el punto de vista de qué protege o a quién protege, tenemos dos opciones dentro de la implementación de un cortafuegos:

- **Personal o de servidor**: Esta implementación defiende únicamente a un equipo personal o a un servidor. Se suele implementar como una aplicación instalada en el equipo de modo que se pueda controlar el tráfico que entra o sale de él.

- **De red**: Esta implementación protege toda una red mediante un dispositivo ubicado entre la red interna e internet, que es el que controla el tráfico. La forma de implementarse es o bien un *software* instalado en un equipo con un sistema operativo convencional o con un aparato *hardware* dedicado con un sistema operativo básico.

Otra información importante a destacar son los tipos de cortafuegos, como son los de filtrado y *proxy*. Vamos a comentarlos dando datos sobre ellos, así como dar ejemplos para que sea más sencillo entender el concepto:

Filtrado

Filtran los paquetes basándose en las cabeceras de los protocolos de red (*stateless*) o en el estado de la conexión (*stateful*). Solamente permite entrar/salir paquetes de internet hacia/desde el servidor. No filtra comunicaciones con máquinas internas, dado que no ve el trafico de la intranet. En la **Figura 3.9**, podemos ver un diagrama.

Stateless

Este tipo de cortafuegos se basa en la configuración de una lista de reglas que se aplican a cada paquete que pasa por el mismo. El objetivo es comprobar los campos de las cabeceras de los protocolos para dejar pasar o bloquear los paquetes. En la configuración, las reglas van a estar en orden y se va a ir contrastando con los campos del paquete para ver si coinciden, en caso de no ser así, el cortafuegos tendrá una política por defecto que acepta o bloquea el paquete. Para la configuración de estas reglas utilizaremos *nftables* más adelante (Fabero Jiménez, 2023).

La **política por defecto** suele seguir una de dos estrategias. Una de ellas es denegar por defecto todo paquete que no esté permitido, es decir, al final de la cadena de reglas, de no haber coincidencia con ninguna, se descarta el paquete. La otra es permitir por defecto. Por lo general siempre se utiliza la de rechazar por defecto por ser más segura y restrictiva con el tráfico desconocido.

Stateful

Este tipo de cortafuegos comparte todas las características del anterior pero añade algo fundamental, tiene en cuenta el estado de las conexiones. Para ello se crea una tabla de entrada por cada conexión en estado *established*. Con esto se puede permitir el tráfico hacia puertos efímeros únicamente si hay una conexión establecida previamente con dichos puertos. A su vez también se identifican paquetes relacionados con conexiones establecidas previamente. Es importante decir que para UDP también funciona aunque no se establezca una conexión como tal. Esto se suele hacer mediante tablas de estado que mantienen información de las conexiones realizadas recientemente, así aunque no esté la conexión establecida se considera como tal.

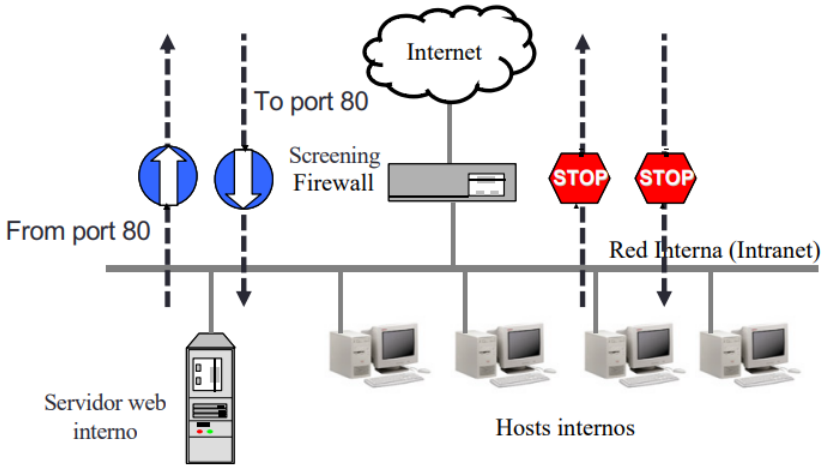

Figura 3.9: Uso del cortafuegos de filtrado. **Fuente**: Moreno Vozmediano (2023a) página 10.

Proxy

Actúan como intermediarios entre origen y destino, creando conexiones de transporte distintas (interna y externa). Intercepta paquetes y los inspecciona antes de enviarlos al destino.

Complementa a un cortafuegos de filtrado como podemos ver en la **Figura 3.10**. Las máquinas internas pueden conectarse a servidores externos de forma indirecta mediante el servidor *proxy*. Los hay de dos tipos, de aplicación y de circuito.

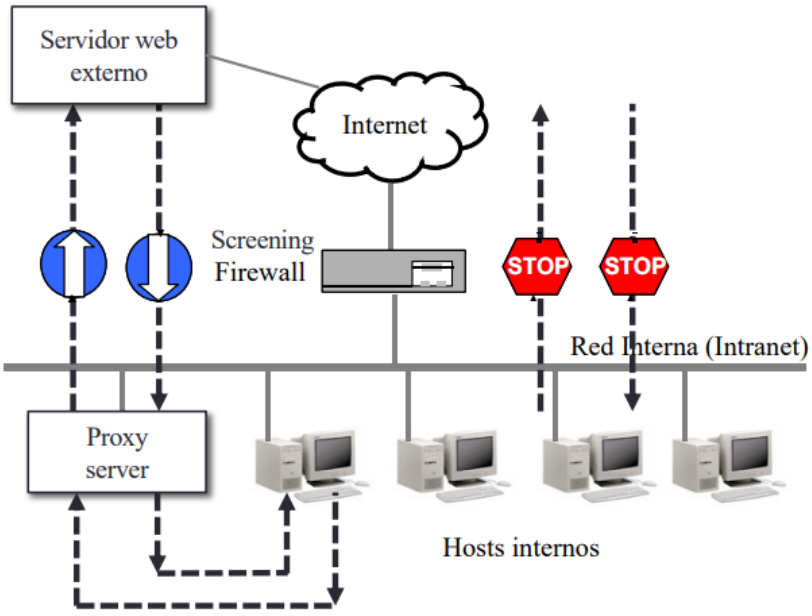

Figura 3.10: Uso del cortafuegos *proxy* complementario. **Fuente**: Moreno Vozmediano (2023a) página 10.

Aplicación

Es un intermediario relativo a aplicación, por ejemplo para solicitar el acceso a una web. En este ejemplo el cliente solicia al *proxy* el acceso a la web, este redirige la petición al servidor web, el cual envía de vuelta la infomación solicitada. Es entonces cuando, finalmente, el *proxy* envía al cliente los datos de la web. Con un diagrama como el de la **Figura 3.11** se entiende mejor el funcionamiento.

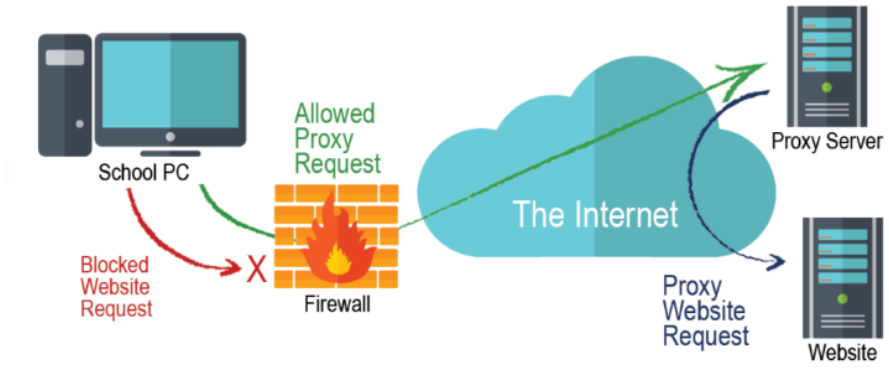

Figura 3.11: Ejemplo de solicitud a servidor *proxy* web. **Fuente**: Dhavaji (2021).

Las características principales de este cortafuegos son:

- El filtrado del contenido no deseado o comandos peligrosos.
- El permiso o no del acceso a servidores externos.
- La posibilidad de implementación de autenticación de usuarios.
- Se permite el uso de caché para acelerar las respuestas.
- Es más seguro que el filtrado de paquetes, pero introduce sobrecarga en cada conexión.
- Específico de cada portocolo de aplicación.

Relativo a circuito

Tiene la función de ser un intermediario en la capa de transporte. No permite una conexión TCP de extremo a extremo, la comunicación entre el cliente y el servidor reales se hace a través del *proxy* de forma indirecta. Es conocido a su vez como cortafuegos *proxy* transparente[1] ya que no modifica la solicitud o respuesta más allá de lo necesario para la autenticación e identificación de *proxy*. Un ejemplo conocido de este tipo de cortafuegos es SOCKS como podemos ver en la **Figura 3.12**.

[1]Este término se define en la página de IBM poniendo de ejemplo **SOCKS**

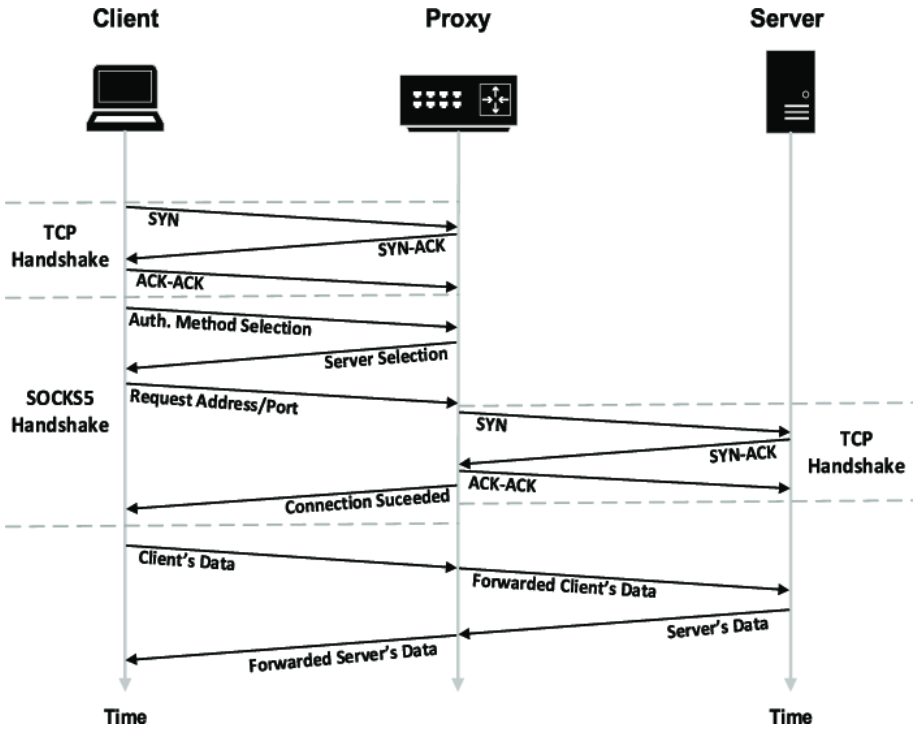

Figura 3.12: Ejemplo de establecimiento de conexión TCP con *proxy* relativo a circuito SOCKS. **Fuente**: Stephen Huang y Cao (2020) página 4.

3.2.2. Sistemas de detección y prevención de intrusiones

Definimos una intrusión como cualquier acceso no autorizado a un sistema o a la utilización maliciosa de sus recursos de información. El objetivo de una intrusión es conseguir el acceso al sistema para causar daños, obtener información o utilizar el sistema para realizar otros ataques. Para ello vamos a ver sistemas que nos ayuden a deterner o prevenir estas intrusiones. Podemos clasificarlos como *Intruder Detection System* (IDS) o *Intruder Prevention System* (IPS) (Moreno Vozmediano, 2023b).

Sistemas de detección de intrusos : IDS pasivo

Es un dispositivo o aplicación software que se encarga de monitorizar un sistema o una red para detectar cualquier tipo de actividad maliciosa y de alertar o notificar cuando se detecten estas actividades. Hay dos tipos de IDS:

- HIDS (*Host-based IDS*): Monitoriza la actividad de un único sistema.
- NIDS (*Network-based IDS*): Monitoriza la actividad de una red.

3.2.3. Sistemas de prevención de intrusos: IPS o IDS activo

Con el tiempo, además de monitorizar y enviar alertas, los IDS empezaron a efectuar acciones cuando se detecta una intrusión. Vamos a hablar de dos tipos:

- NIPS (*Network-based IPS*): protege una red utilizando detección basada en firmas. La detección basada en firmas busca patrones almacenados en una base de datos para comprobar de qué tipo de ataque está ocurriendo. Por ejemplo se busca si la combinación de flags TCP coincide con los usados durante una exploración de puertos.

- NBA (*Network Behaviouts Analysis*): portege una red utilizando detección de anomalías. La detección de anomalías busca desviaciones del comportamiento normal, clasificando como anómala o normal la actividad de los diferentes usuarios. Un ejemplo de actividad anómala es el consumo excesivo de ancho de banda.

Una vez definidos estos sistemas vamos a ver qué acciones realizan los NIDS/NIPS. Para poder detectar o prevenir ataques es necesario que estén continuamente analizando el tráfico de red, en particular en NIDS funciona como un rastreador de red (*sniffer*) para hacer esto. Con la idea de no sobrecargar la herramienta se utilizan filtros para no tener que analizar en bruto todo el tráfico. Una vez hecho esto se aplican reglas para detectar anomalías en el tráfico y por último generar alertas cuando algún paquete o secuencia de paquetes coincide con las reglas. En particular, NIPS puede descartar el tráfico en vez de notificarlo.

3.2.4. NIDS/NIPS y Cortafuegos : ¿Diferentes o compatibles?

Firewall	**NIDS/NIPS**
Limita el acceso entre redes, previniendo intrusiones.	Identifica cuándo una intrusión está teniendo lugar, generando una alerta (NIPS puede descartar).
No identifica cuándo un ataque está ocurriendo en la red.	Es capaz de detectar ataques en tiempo real.
Se basa en reglas simples; filtrado por IP, puerto origen, etc.	Se basa en reglas complejas como la detección de anomalías o la detección basada en firmas.
Limitada a las reglas configuradas.	Puede detectar ataques desconocidos mediante análisis de comportamiento.
Puede operar en modo de estado y sin estado.	Operación en tiempo real para detectar intrusiones.

Tabla 3.1: Comparación entre un Firewall y un NIDS/NIPS

3.3. Alternativas sin inteligencia artificial

En esta sección vamos a ver algunas herramientas similares a *Icewall* que existen actualmente en el mercado y que no utilizan inteligencia artificial para llevar a cabo su objetivo.

3.3.1. Snort

Descripción

Snort (Cisco, 2024) es una herramienta de código abierto para análisis de tráfico. Puede funcionar como rastreador o registrador de red, NIDS o NIPS. La similitud que tiene con el sistema que planteamos es que dado un flujo de paquetes genera alertas sobre posibles ataques de red.

Características

Snort utiliza un motor de detección basado en reglas que se actualiza de manera regular para poder reconocer las últimas vulnerabilidades y técnicas de ataque.

3.3.2. Suricata

Descripción

Al igual que *Snort*, *Suricata* (OISF, 2024) es otra solución que permite la detección de intrusiones y la prevención de amenazas de red. Es capaz de monitorizar el tráfico de red a tiempo real y clasificarlo en benigno y maligno basándose en reglas predefinidas.

Características

Destaca por su capacidad de realizar análisis con *multithreading*, lo que permite manejar mucho tráfico de forma simultánea con técnicas de detección basadas en anomalías y firmas.

3.3.3. Zeek

Descripción

Zeek (The Zeek Project, 2024) (anteriormente llamado *Bro*) es un *framework* de análisis de seguridad en redes que se centra en el análisis de comportamientos y el registro de actividades en red, en vez de las soluciones basadas en firmas (*Snort* y *Suricata*).

Características

Es ideal para entornos de red avanzados ya que permite una gran personalización para detectar anomalías específicas.

3.3.4. Alternativas con inteligencia artificial

Para completar la sección anterior vamos a ver unas herramientas que existen en el mercado pero que en este caso sí utilizan inteligencia artificial para su objetivo, es por ello que se aproxima aún más a la idea que se busca en *Icewall*.

3.3.5. Cisco Firepower

Descripción

Cisco Firepower (Cisco Firepower, 2018) es un sistema de firewall de próxima generación que integra múltiples capas de defensa de seguridad en una sola plataforma. Utiliza un enfoque basado en amenazas, proporcionando visibilidad y control de aplicaciones, usuarios y *url* a través de sus capacidades de prevención de intrusiones y filtrado de contenido. Los dispositivos *Firepower* operan con el software *Cisco Firepower Threat Defense*, que incluye protección avanzada contra malware (AMP), prevención de intrusiones de siguiente generación (NGIPS) y capacidades de filtrado de *url*.

Características

Consta de un rendimiento excepcional incluso con funciones avanzadas de defensa de amenazas activas, una gestión eficiente y la capacidad de escalar para cumplir con las demandas de diferentes entornos desde pequeñas oficinas hasta grandes centros de datos. Además, su arquitectura modular permite adaptarse a necesidades específicas de rendimiento y seguridad. Utiliza la inteligencia artificial en AMP y NGIPS, esto ayuda en el análisis y la detección automática de amenazas, permitiendo una respuesta más rápida y efectiva a incidentes de seguridad.

3.3.6. Palo Alto Networks Next-Generation Firewall

Descripción

Al igual que la alternativa de *Cisco*, *Palo Alto Networks* (Palo Alto Networks, 2024) combina cortafuegos, prevención de intrusiones además de funciones de seguridad basadas en la nube.

Características

El funcionamiento de estos NGFW se centra en la prevención de amenazas mediante el uso integrado de inteligencia de seguridad generada a partir de una gran cantidad de información de clientes. Esto permite detectar amenazas conocidas y desconocidas, incluso en tráfico encriptado. Además, con el soporte de aprendizaje automático integrado, los firewalls pueden adaptar y mejorar continuamente sus capacidades de detección y respuesta ante amenazas emergentes.

3.3.7. Darktrace

Descripción

Dentro de las ya vistas, *Darktrace* (DarkTrace, 2023) es la solución más avanzada ya que utiliza la inteligencia artificial para detectar ataques a tiempo real. La plataforma de seguridad *ActiveAI* se especializa en la visibilidad continua de todo el entorno digital de una organización, permitiendo una respuesta adaptada a las necesidades propias del enterno.

Características

El sistema de *Darktrace* aprende de manera continua en base a los patrones de cada dispositivo y usuario dentro de una red para así detectar comportamientos anómalos. Utiliza aprendizaje no supervisado, lo que permite detectar ataques sin necesidad de datos de entrenamiento previos o modelos específicos.

3.4. Comparativa con *IceWall*

El objetivo que tiene *IceWall* es utilizar inteligencia artificial para detectar ataques que están ocurriendo en una red para poder aplicar técnicas concretas para el ataque detectado y así frenar la actividad maliciosa de una forma personalizada y eficiente.

Hemos podido ver que en el mercado encontramos soluciones en las que se utiliza inteligencia artificial para llevar esta tarea a cabo, no obstante al ser desarrolladas por empresas que comercializan su uso, nos impide tener acceso a las implementaciones y por ello a no comprender su funcionamiento interno. Vamos a intentar comprender de qué manera funcionan los modelos predictivos de detección de ataques mediante el reconocimiento de patrones dependiendo el tipo de ataque. En los capítulos posteriores se irá explicando más en profundidad el funcionamiento de la implementación, pero es necesario una comparación con alternativas ya existentes.

Las fases de funcionamiento de *Icewall* serán: **preprocesado de datos, entrenamiento para encontrar patrones, detección de ataques, postprocesado y mitigación.**

Capítulo 4

Proceso de entrenamiento

En este capítulo vamos a abordar toda la parte relativa al entrenamiento del modelo. Comenzaremos con un preprocesado de los datos, elección de los algoritmos concretos, análisis de los resultados y entrenamiento del modelo final. Con esto obtendremos un modelo con una configuración óptima para detectar qué ataque está ocurriendo en un tráfico específico.

4.1. Introducción al conjunto de datos CICIDS2017

El conjunto de datos CICIDS2017 (*Canadian Institute for Cibersecurity Intrusion Detection Systems*) fue creado por el Instituto de Cyberseguridad de la Universidad de Nuevo Brunswick en Canadá. Este conjunto de datos consiste en el tráfico generado entre los días 3 y 7 de Julio de 2017. Toda la información contenida en los datos se resume en el contenido de la **Tabla 4.1**.

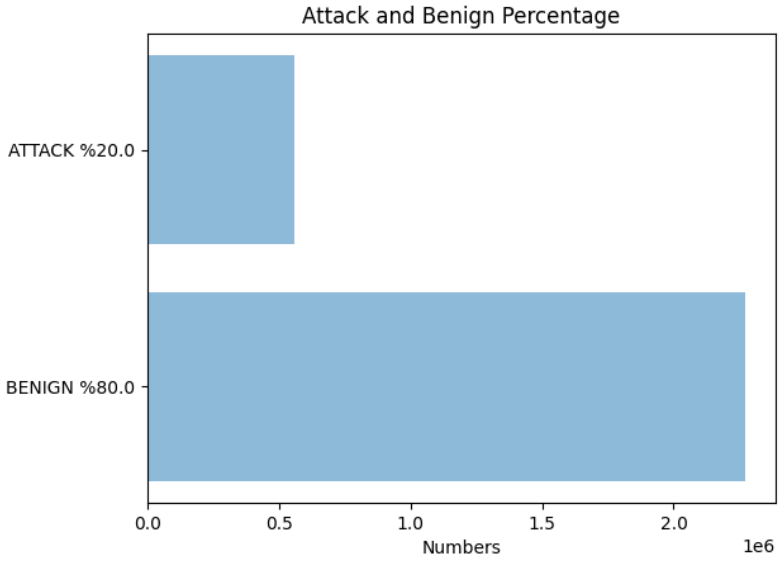

Figura 4.1: Porcentaje de paquetes malignos y benignos.

Día de ocurrencia (Working Hours)	**Tamaño Pcap**	**Duración**	**Tamaño CSV**	**Nombre Ataque**
Lunes	10 GB	Todo el día	257 MB	Benigno
Martes	10 GB	Todo el día	166 MB	FTP-Patator, SSH-Patator
Miércoles	12 GB	Todo el día	272 MB	DoS Hulk, DoS GoldenEye, DoS slowloris, DoS Slowhttptest, Heartbleed
Jueves	7.7GB	Mañana	87.7 MB	Web Attacks (Brute Force, XSS, Sql Injection)
		Tarde	103 MB	Infiltration
viernes	8.2GB	Mañana	71.8 MB	Bot
		Tarde	92.7 MB	DDoS
		Tarde	97.1 MB	PortScan

Tabla 4.1: Detalles del conjunto de datos CICIDS2017

Estos datos han sido obtenidos mediante tráfico real recogido de diferentes ordenadores con sistemas operativos actualizados (Mac, Windows y Linux en diferentes versiones). Es importante destacar que el 80 % del tráfico recogido es benigno y el 20 % restante son el contenido de los ataques, como podemos observar en la **Figura 4.1**. Para el entrenamiento utilizaremos todo el contenido de los ataques pero nos hemos centrado en poder diferenciar PortScan y el DDoS para representar la utilidad de la herramienta ejemplificando con un caso particular.

4.2. Preprocesamiento

Al intentar entrenar el modelo nos damos cuenta de que hay ciertos errores en el conjunto de datos. Son errores pequeños fácilmente solucionables con un preprocesado correcto. Es por ello por lo que en esta sección vamos a comentar de qué manera hacemos esta tarea para que los algoritmos de *machine learning* puedan entenderlos.

4.2.1. Limpieza inicial de los datos

Empezamos leyendo cada archivo por líneas para encontrar y reemplazar caracteres que pudiesen dar problemas a *Pandas* (McKinney, 2024) para abrir los archivos de entrada como un *dataframe*[1].

El conjunto de datos contiene 3119345 paquetes en total. Examinando los datos se detectan 288602 catalogados como erróneos o incompletos. El primer paso va a ser eliminar estos datos para evitar errores posteriores en el proceso de entrenamiento. Para detectarlos se observa que a la hora de crear un *dataframe*. Estos datos constan de 86 columnas definiendo diferentes propiedades de los paquetes de red como podemos ver en **Apéndice A**.

Una de las propiedades, *Fwd Header Length* (define la dirección del flujo de datos para el total de bytes usados), está duplicada en las columnas 41 y 62 por lo que se soluciona eliminando la columna 62.

Otro cambio necesario es convertir las propiedades categóricas y *string* (Flow ID, Source IP, Destination IP, Timestamp, External IP) en datos numéricos para usarse en algoritmos de *machine learning*. Esto puede hacerse con LabelEncoder() de *Sklearn*[2] (Cournapeau, 2024) para convertir estas propiedades en numéricas.

Aunque el campo "Label" no es numérico, no se hacen cambios en el dado que durante el entrenamiento es necesario para clasificar los ataques. Por último se hacen unas pequeñas modificaciones necesarias para evitar errores al utilizar los datos, estas son:

- Se han encontrado varios guiones con código unicode 8211 que provocan confusión a los algoritmos de aprendizaje. Para evitar este problema se sustituirán por el guion con código unicode 45.
- En los campos "Flow Byte/s" y "Flow Packet/s" podemos ver que, en algunos casos, incluyen los valores "Infinity" o "NaN" que serán substituídas por -1 y 0 respectivamente ya que los algoritmos de entrenamiento funcionan correctamente con estos valores. Podría probarse con el valor máximo de los enteros y con -1 para indicar un elemento muy grande y uno que no tiene valor, no obstante funciona correctamente con los valores previamente mencionados.

[1] Son arrays bidimensionales, de tamaño variable utilizados por *Pandas*.

[2] Es un conjunto de rutinas escritas en Python para hacer análisis predictivo, que incluyen clasificadores, algoritmos de clusterización, entre otros.

4.3. Estadística de los datos

Esta sección nos sirve para arrojar un poco de comprensión hacia los datos que acabamos de preprocesar. Ver las gráficas nos ayuda a saber que las etiquetas son las más y menos frecuentes para así saber el orden de magnitud aproximado de los datos con los que vamos a entrenar el modelo.

4.3.1. Grupo de datos con número reducido de paquetes

Este grupo es pequeño dado que los ataques tienen un número muy limitado de paquetes, en este caso menos de 600. Por lo que para el entrenamiento pueden resultar limitados y entonces podría ocurrir que no se reconociesen estos ataques de manera precisa. De todos modos como para el entrenamiento vamos a centrarnos en PortScan y DDoS no van a tener gran impacto en el resultado final. Podemos ver los datos en la **Figura 4.2**.

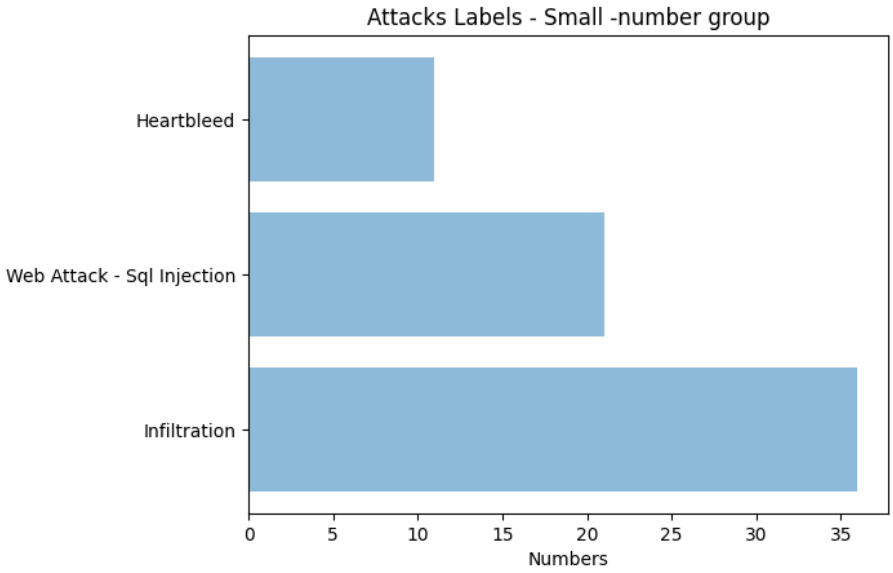

Figura 4.2: Grupo de ataques con un número pequeño de paquetes.

4.3.2. Grupo de datos con número medio de paquetes

Hemos catalogado este grupo como mediano por tener ataques con un número de paquetes comprendido entre los 600 y 11000. Esto resulta mucho más esperanzador que el anterior ya que el número de datos parece suficiente para que se detecten correctamente los tipos de ataque, podemos corroborar esta información en la **Figura 4.3**. Del mismo modo que el anterior conjunto de paquetes, estos ataques no vamos a estudiarlos en profundidad.

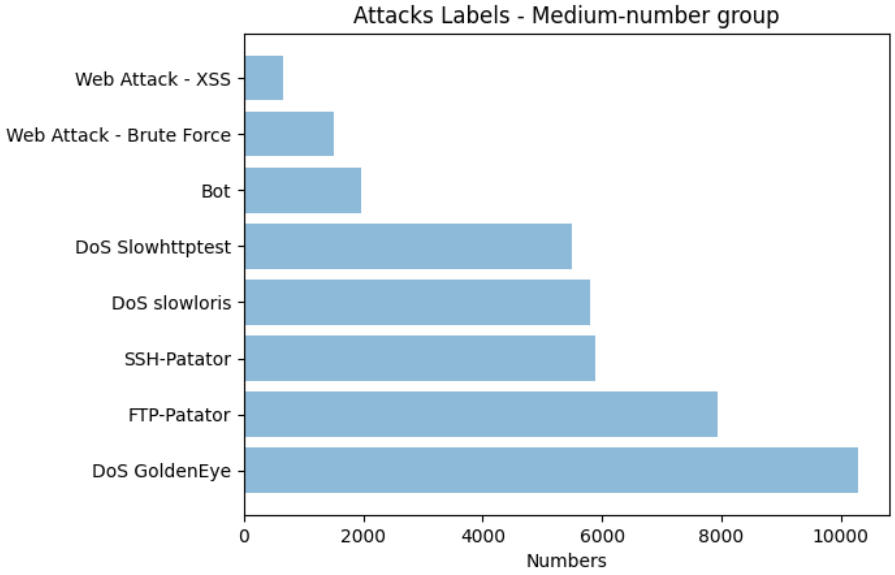

Figura 4.3: Grupo de ataques con un número intermedio de paquetes.

4.3.3. Grupo de datos con número elevado de paquetes

Este último grupo tiene un número muy elevado de paquetes. Por lo que va a resultar útil para clasificar correctamente los ataques con un entrenamiento que los utilice. Podemos observar en la **Figura 4.4** que este grupo contiene los ataques PortScan y DDoS que por su naturaleza es lógico pensar que el número de paquetes va a ser mayor que el de otros. Como también tenemos el flujo benigno de la red, podemos concluir que este grupo es el más relevante para la tarea que vamos a realizar a lo largo de este proceso.

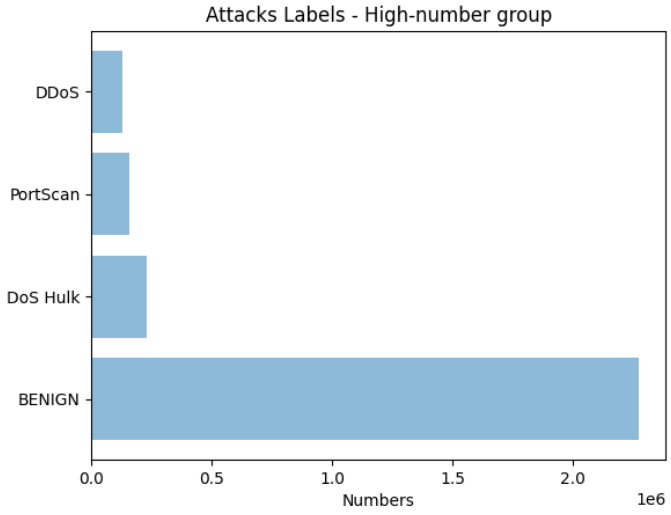

Figura 4.4: Grupo de ataques con un número elevado de paquetes.

4.4. Pasos previos al entrenamiento

Antes de entrenar un modelo que nos arroje resultados para la detección de los ataques es necesario crear subconjuntos de datos para cada tipo de ataque y encontrar que características son más representativas para cada uno de los ataques. Dado que hay un número elevado de paquetes y columnas, hacer esto facilitará el entrenamiento y reducirá la confusión más adelante cuando entrenemos el modelo final.

4.4.1. Creación de una función de entrada por tipo de ataque

Para ser usado en los siguientes pasos de entrenamiento vamos a crear un archivo con datos por cada ataque. El objetivo es tomar todo el flujo de ataques y luego rellenar con flujo benigno para tener en cada documento un aproximado de 30 % de ataques y 70 % de flujo benigno. Es importante destacar que los ataques de tipo *Web* van a unirse en el mismo archivo.

Para crear estos documentos comenzamos guardando el número de paquetes que son benignos y un diccionario de ataques donde las claves serán los nombres de los ataques y los valores el número de paquetes asociados a cada uno.

Se va a ir recorriendo el archivo *all_ data.csv* y cada vez que obtengamos una línea benigna vamos a elegir de manera aleatoria si se añade o no al documento del ataque asociado, de esta manera podemos asegurarnos que el tráfico benigno no es el mismo siempre en todos los ataques. En caso de que la etiqueta sea del ataque que estemos agrupando siempre se añadirá al conjunto de manera que se respete los porcentajes anunciados anteriormente. Una vez hecho esto tenemos todas las funciones de entrada necesarias para continuar con el proceso de entrenamiento.

4.4.2. Características más importantes para cada ataque y para conjunto de todos los datos

En esta sección vamos a entrenar un modelo para comprobar que características son más importantes para comprender mejor como identificar cada tipo de ataque. Se va a utilizar durante todo el proceso de entrenamiento la librería *Sklearn* (Cournapeau, 2024) de dónde, en este caso, usaremos el algoritmo *RandomForestRegressor* [3].

En primer lugar vamos a necesitar hacer unas pequeñas modificaciones para que el algoritmo pueda funcionar correctamente. La columna "Label" la vamos a ir reemplazando por un vector en el que benigno se va a representar con un 1 y los ataques se van a representar con un 0.

Una vez hecho esto vamos a guardar en una variable los datos de test, que van a ser las etiquetas en representación numérica y en otra variable vamos a guardar el resto de los datos que van a ser los de entrenamiento. Ahora entrenaremos el modelo

[3]La información sobre este algoritmo se encuentra en la documentación de **scikit-learn**.

con estos datos para luego sacar las características deseadas.

Una realizado entrenamiento podemos graficar los resultados de las predicciones para ver en cada ataque cuales son las características más importantes. La disposición de las gráficas es tener en el eje vertical la importancia de las características y en el eje horizontal las etiquetas de las columnas del *dataframe*, ver en el **Apéndice B**.

4.5. Entrenamiento con Machine Learning

En esta sección vamos a entrenar utilizando los diferentes conjuntos de datos y características obtenidas previamente para poder encontrar un algoritmo óptimo que será con el que entrenemos el modelo final para predecir los ataques. Para cada entrenamiento vamos a utilizar diferentes algoritmos y características que se van a ir indicando a continuación, para obtener resultados más fiables se va a repetir el proceso 10 veces para descartar casos patológicos, es decir, casos que por motivos particulares entrenen un modelo erróneo. Al entrenar tantas veces podemos promediar los resultados evitando este efecto.

4.5.1. Entrenamiento sobre funciones de entrada de ataques

Vamos a utilizar los subconjuntos de datos de cada ataque para los entrenamientos de esta sección. A continuación se va a dar la información necesaria para comprender como se ha realizado la tarea de aprendizaje del modelo.

- **Algoritmos**: Naive Bayes, QDA, Random Forest, ID3, AdaBoost, MLP y Nearest Neighbors.
- **Características**: Para cada ataque vamos a utilizar las primeras cuatro características de las que hemos obtenido previamente. Por ejemplo en el caso particular de PortScan usaremos Flow Byte/s, Total Lenght of Fwd Packets, Fwd IAT Total y Flow Duration. Hay que recordar que "Label" siempre se incluirá por ser la característica que permite la clasificación.

Una vez hecho este proceso vamos a observar unos parámetros que nos permiten ver la calidad del entrenamiento. Estos son los siguientes:

- **Precisión**: Lo definimos como la fracción

$$\text{Precisión} = \frac{TP}{TP + FP}$$

  donde TP representa los verdaderos positivos y FP los falsos positivos. El mejor valor de precisión es 1 y el peor es 0.[4]

[4]La información sobre la **precisión** está en la documentación de scikit-learn.

- ***Recall***: Lo definimos como la fracción

$$\text{Recall} = \frac{TP}{TP + FN}$$

  donde TP representa los verdaderos positivos y FN los falsos negativos. El mejor valor de recall es el 1 y el peor es 0.[5]

- ***F1 Score***: Esta medida se puede interpretar como una media armónica de la precisión y el *recall*, dónde alcanza su mejor valor en el 1 y el peor en el 0. La fórmula es

$$F1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

  donde TP representa los verdaderos positivos, FP los falsos positivos y FN los falsos negativos.[6]

- ***Accuracy***: La definimos como la fracción

$$\text{Accuracy} = \frac{TN + TP}{FP + TN + TP + FN}$$

  donde TP representa los verdaderos positivos, FP los falsos positivos, TN los verdaderos negativos y FN los falsos negativos.[7]

Se van a guardar estos datos en un archivo llamado **results_1.csv** que por columnas tiene el ataque, algoritmo, *accuracy*, precisión, *recall*, *F1-Score* y tiempo. Esto será utilizado después de haber completado todos los entrenamientos para comprobar que algoritmo deberemos utilizar para el entrenamiento final. Vamos a crear un archivo de las mismas características para los siguientes entrenamientos.

4.5.2. Entrenamiento sobre la función de entrada de todos los datos

En esta parte vamos a entrenar de diversas maneras el archivo *all_ data.csv* para extraer datos dependiendo de características y algoritmos (Kostas, 2018).

- **Primer Entrenamiento**: Este entrenamiento va a utilizar los mismos algoritmos que el apartado anterior, en cambio va a utilizar las características resultantes de combinar las cuatro primeras de cada ataque y como son 12 ataques tenemos un total de 48 que, eliminando repeticiones, resulta en 18 más el "Label". Los resultados se guardarán en el archivo **results_2.csv**.

- **Segundo Entrenamiento**: Este entrenamiento va a seguir combinando los mismos algoritmos pero con otras características. Ahora vamos a tomar las características sobre el fichero de todos los ataques. Se van a escoger siete características más relevantes de ente entrenamiento junto con el campo "Label". Los datos resultantes serán guardados en el archivo **results_3.csv**.

[5]La información sobre el ***recall*** está en la documentación de scikit-learn.
[6]La información sobre el ***F1 Score*** está en la documentación de scikit-learn.
[7]La información sobre el ***accuracy*** está en la documentación de scikit-learn.

- **Tercer Entrenamiento**: En esta versión se van a tomar para los algoritmos Naive Bayes, QDA y MLP unas características determinadas atendiendo a los datos previamente calculados que tienen un peso relevante para el entrenamiento. En cambio, para Random Forest, ID3, AdaBoost y Nearest Neighbors se considera que han de tener una misma lista de características de 7 elementos extraídos de las características del conjunto de todos los datos. Los resultados se guardarán en un archivo llamado **results_Final.csv**.

4.5.3. Análisis de los resultados del entrenamiento

Una vez terminadas las fases de entrenamiento vamos a analizar los archivos resultantes para poder ver la eficiencia de cada algoritmo para elegir el mejor de entre todos para entrenamiento final. Creamos para esta tarea diferentes gráficas para poder observar los valores de las métricas y así comparar dichos algoritmos. Para crear estas gráficas, haremos un promedio de las características por ataque y algoritmo, podemos ver el resultado en el **Anexo B**.

Una vez creadas vamos a ver por cada métrica qué algoritmos resultan mejores para cada ataque. Tras esta clasificación, vamos a hacer un resumen general dónde extraeremos el algoritmo más eficiente para los datos y entrenamientos dados.

Bot	AdaBoost
DDoS	ID3
DoS GoldenEye	Random Forest
DoS Hulk	AdaBoost / ID3
DoS Slowhttptest	AdaBoost
DoS slowloris	AdaBoost
FTP-Patator	Random Forest
Heartbleed	AdaBoost/ID3/Naive Bayes/Nearest Neighbors/ QDA/Random Forest
Infiltration	Random Forest
PortScan	Nearest Neighbors
SSH-Patator	AdaBoost
Web Attack	Random Forest

Tabla 4.2: Mejores algoritmos por cada ataque para la métrica *accuracy*

Bot	AdaBoost
DDoS	ID3
DoS GoldenEye	Random Forest
DoS Hulk	ID3
DoS Slowhttptest	AdaBoost
DoS slowloris	AdaBoost
FTP-Patator	AdaBoost/ID3/MLP/Nearest Neighbors/ Random Forest
Heartbleed	AdaBoost/ID3/Naive Bayes/Nearest Neighbors/ QDA/Random Forest
Infiltration	Random Forest
PortScan	ID3/Nearest Neighbors/Random Forest
SSH-Patator	AdaBoost/ID3/Random Forest
Web Attack	Ada Boost

Tabla 4.3: Mejores algoritmos por cada ataque para la métrica *F1-Score*

Bot	AdaBoost
DDoS	ID3
DoS GoldenEye	Random Forest
DoS Hulk	AdaBoost
DoS Slowhttptest	AdaBoost
DoS slowloris	AdaBoost/ID3
FTP-Patator	AdaBoost/ID3/MLP/Nearest Neighbors/ Random Forest
Heartbleed	AdaBoost/ID3/Naive Bayes/Nearest Neighbors/ QDA/Random Forest
Infiltration	Random Forest
PortScan	ID3/Nearest Neighbors/Random Forest
SSH-Patator	AdaBoost/ID3/Random Forest
Web Attack	Random Forest

Tabla 4.4: Mejores algoritmos por cada ataque para la métrica *precision*

Bot	AdaBoost
DDoS	ID3
DoS GoldenEye	Random Forest
DoS Hulk	ID3/Nearest Neighbors
DoS Slowhttptest	AdaBoost
DoS slowloris	AdaBoost
FTP-Patator	AdaBoost/ID3/MLP/Nearest Neighbors/ Random Forest
Heartbleed	AdaBoost/ID3/Naive Bayes/Nearest Neighbors/ QDA/Random Forest
Infiltration	Random Forest
PortScan	ID3/Nearest Neighbors
SSH-Patator	ID3
Web Attack	AdaBoost

Tabla 4.5: Mejores algoritmos por cada ataque para la métrica *recall*

Ahora vamos a hacer una tabla que sirva de resumen de las que acabamos de ver como son la **Tabla 4.2**, **Tabla 4.3**, **Tabla 4.4** y **Tabla 4.5**. En este resumen que recogemos en la **Tabla 4.6** podemos ver que el algoritmo que más veces ha resultado ser el mejor es AdaBoost, por lo que este va a ser con el que entrenemos el modelo con el que nos servirá para predecir los ataques. Nos fijamos en que, por ejemplo, DDoS y PortScan tienen diferentes algoritmos óptimos, no obstante más adelante observaremos que el comportamiento de AdaBoost es bueno para predecir estos ataques.

Bot	AdaBoost
DDoS	ID3
DoS GoldenEye	Random Forest
DoS Hulk	ID3
DoS Slowhttptest	AdaBoost
DoS slowloris	AdaBoost
FTP-Patator	Random Forest
Heartbleed	AdaBoost/ID3/Random Forest
Infiltration	Random Forest
PortScan	Nearest Neighbors
SSH-Patator	ID3
Web Attack	AdaBoost

Tabla 4.6: Mejores algoritmos entre todas las métricas.

4.5.4. Entrenamiento del modelo predictor

Vamos a entrenar con AdaBoost el archivo *all_ data.csv* con las mismas características que hemos obtenido de los entrenamientos anteriores. En este caso vamos a centrarnos en distinguir los ataques y analizar la matriz de confusión resultante. De cara a poder hacer un código de postprocesado adecuado nos vamos a centrar en distinguir: **Benigno**, **DDoS**, **PortScan** y **Otros Ataques**.

Esto es muy probable que genere confusión al distinguir la categoría **Otros Ataques**. Esto es normal porque tienen peculiaridades que estamos agrupando en una misma clase, de cara a mejorar la clasificación tendríamos que poder elegir entre los diferentes ataques utilizando un diccionario más amplio, no obstante esto se haría posteriormente cuando se quisiesen tratar más ataques y no únicamente los dos que hemos elegido como ejemplo. Hemos ejecutado este código varias veces para obtener el mejor resultado posible para luego guardar el estado de entrenamiento para poder hacer predicciones sobre otros conjuntos de datos. Este entrenamiento tiene las métricas que se observan en la **Tabla 4.7**.

Accuracy	**F1-Score**	**Precisión**	**Recall**
0.84	0.66	0.75	0.73

Tabla 4.7: Datos del modelo final con AdaBoost.

Vamos a analizar la matriz de confusión de la **Figura 4.5** para ver si se han detectado correctamente los ataques:

- **Benigno**: Observamos que lo mejor clasificado en este modelo son los paquetes benignos. Es fácil que esto ocurra debido a la alta densidad de paquetes benignos dentro del dataset, incluso habiendo controlado la cantidad de ellos de cara al entenamiento. Podría decirse que se confunde un poco con PortScan pero la diferencia entre los datos muy grande por lo que la confusión no es relevante.
- **DDoS**: Del mismo modo este ataque se clasifica bastante bien. Casi todos los paquetes DDoS se han identificado como tal y hay una parte que se confunde con paquetes benignos. No obstante, se clasifica lo suficientemente bien.
- **PortScan**: Este ataque prácticamente no tiene confusión, es decir, el modelo que hemos entrenado es capaz de detectar de manera muy precisa cuando un ataque se trata de un PortScan.
- **Otros Ataques**: Vemos que debido a las diversas características de los ataques restantes la confusión es total. Se prevé que si identificamos correctamente los ataques la confusión se reduce.

Ahora para comprobar un caso particular vamos a recuperar el modelo entrenando que dio lugar a la anterior matriz de confusión. Vamos a alimentar este modelo con datos que representaran un ataque de exploración de puertos. Una vez hecho esto analizamos la nueva matriz de confusión (**Figura 4.6**) para ver que ha funcionado correctamente el modelo para un caso particular.

Del mismo modo vamos a alimentar el modelo con datos de un ataque de tipo DDoS. El resultado podemos observarlo en la **Figura 4.7**. Dónde vemos que la gran mayoría de los paquetes detectados son correctos aunque hay una pequeña confusión con paquetes benignos.

Ahora viendo que clasifica correctamente podemos pasar a la implementación de un código de postprocesado para bloquear, en casa caso, el ataque pertinente.

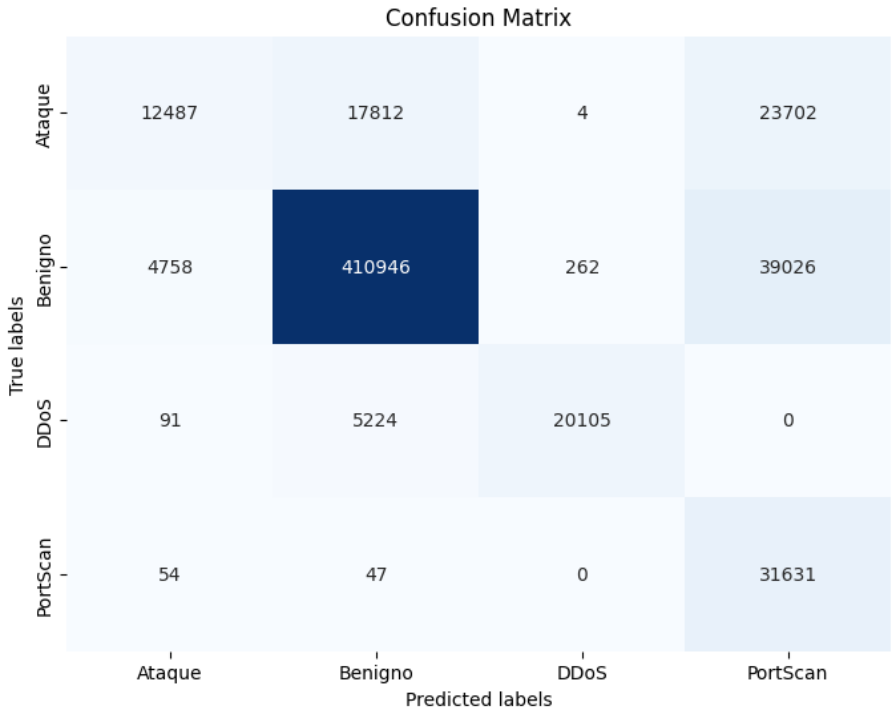

Figura 4.5: Matriz de confusión después del entrenamiento.

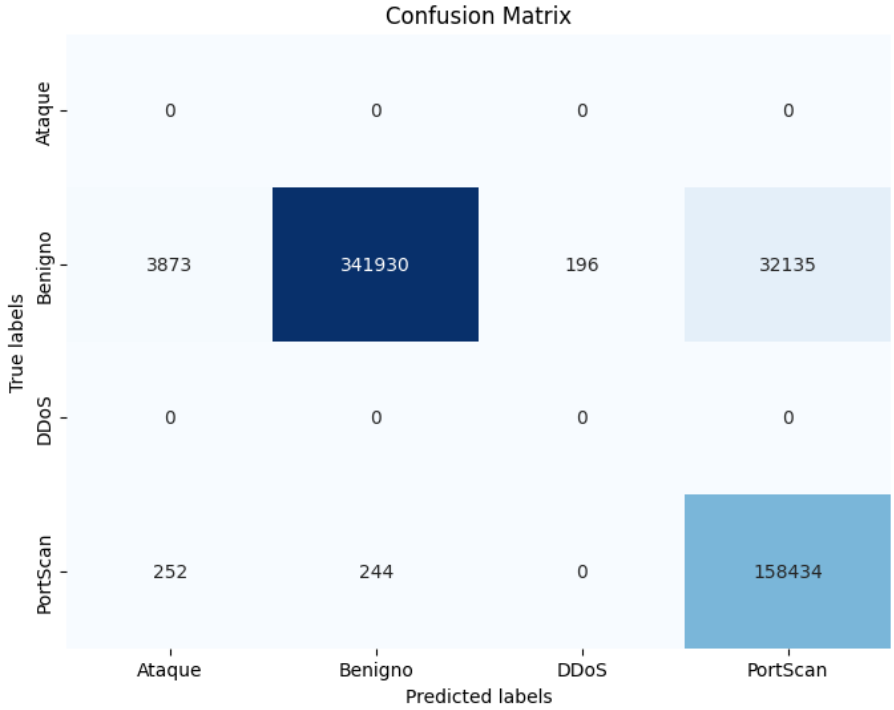

Figura 4.6: Matriz de confusión para un ataque PortScan.

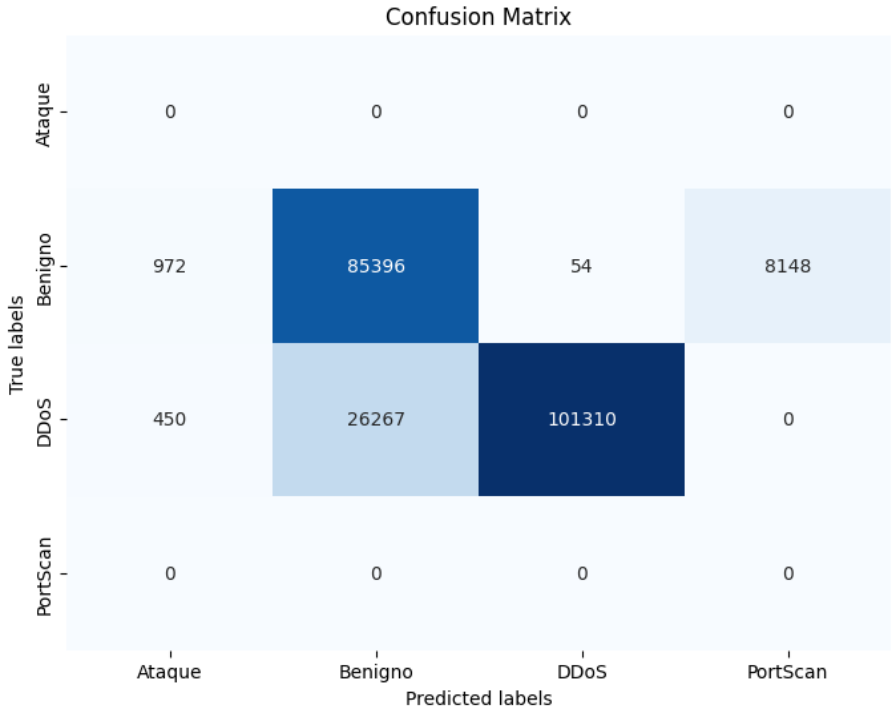

Figura 4.7: Matriz de confusión para un ataque DDoS.

4.6. Postprocesado y funcionamiento final

Con todo lo que hemos hecho hasta ahora sabemos que un conjunto de datos está siendo atacado o no y con que tipo de ataque en particular. Ahora tenemos que procesar los datos de manera que se frene el ataque en que se ha detectado. Este proceso tiene que ser personalizado para cada ataque debido a sus peculiaridades, es por ello por lo que vamos a centrarnos en PortScan y DDoS.

4.6.1. Bloqueo de PortScan

El código que bloquea el PortScan utilizará el archivo que representa al viernes por la tarde dónde sabemos que ha habido un ataque de este tipo, cosa que ha sido confirmada con el modelo predictor obtenido en el capítulo 4.

Lo primero que vamos a hacer es filtrar por protocolo ya que en caso de no ser **UDP** o **TCP** sabemos que esos paquetes no están relacionados con un ataque de exploración de puertos. Una vez hemos hecho esto vamos a quedarnos con los campos *Source IP*, *Destination Port* y *TimeStamp* por ser los datos relevantes para las reglas pertinentes.

Ahora vamos a bloquear diferentes ip en base a dos parámetros:

- **N** := Número de puertos.
- **K** := Intervalo de tiempo en minutos.

Tenemor que ir contando en intervalos de tiempo de **K** minutos. En el caso de que en esas ventanas de tiempo detectemos más de **N** puertos distintos entonces vamos a asumir que las ip cumpliendo estas características son sospechosas. Una vez terminado ese proceso se va a bloquear con una regla *nftables* dichas ip. La regla es la siguiente:

nft add rule ip filter input ip saddr **{ip}** drop

Para entender bien lo que hace esta regla expliquemos brevemente en que consiste. Como indica **add rule ip** estamos añadiendo una regla que se aplicará en la capa ip. Ahora es necesario indicar que la regla se añadirá a la tabla de filtrado de paquetes para los que sean detectados como entrantes, **filter input**. Ahora queda por elegir el criterio de coincidencia de la regla que, en este caso, es la ip origen; **saddr** (source address). En caso de que se cumpla entonces vamos a descartar dicho paquete, de ahí el **drop** (Fabero Jiménez, 2023).

4.6.2. Bloqueo de DDoS

El código para bloquear el DDoS utilizará el archivo que representa al viernes por la tarde, en este caso, al que registra un ataque de este tipo confirmado además por el predictor.

En primer lugar vamos a filtrar de nuevo por protocolo del mismo modo que en exploración de puertos. En este caso es importante destacar que el ataque se hará sobre un único puerto que es al que se pretende denegar el servicio. Nos vamos a quedar con los datos que representan a la cantidad de veces que una ip envía un datagrama a un puerto en específico. Con estos datos vamos a elegir cual de estos puertos ha sido el más atacado. Una vez encontrado este puerto vamos a elegir las ip que se contabilizan por encima de un umbral, en ese caso se considerará sospechosa y se bloqueará por cierto tiempo.

Estos parámetros de bloqueo serán el tiempo que representamos como **T** y el umbral de número de paquetes enviados **N**. Por lo que generamos la siguiente regla:

nft add rule ip filter input ip saddr **{ip}** drop ip saddr **{ip}**

time after **{T}**h counter drop

La única diferencia a la regla generada para escaneo de puertos es que en este caso se pretende bloquear la ip durante un tiempo **T** determinado.

4.6.3. Funcionamiento de *IceWall*

Ahora que hemos implementado el modelo predictor y los códigos de postprocesado, ya es posible detectar y prevenir ataques, siempre y cuando los datos de entrada estén preprocesados como se espera.

1 Preprocesado de los datos

Del mismo modo que ya hemos hecho, en un nuevo conjunto de datos se necesita un preprocesado para que el entrenamiento pueda realizarse. Es importante para esta tarea que los datos de entrada tengan la misma forma que los que ya se han utilizado, es decir, las mismas cabeceras. No es un problema si cambiase ya que lo único que habría que hacer es adaptar el código de preproceso al ámbito en el que se esté trabajando. Ahora con un archivo de entrada que entienda el modelo podemos continuar.

2 Predicción del ataque

Ahora que tenemos los datos de entrada y un modelo entrenado tenemos que utilizarlo para que nos indique que ataque se está produciendo. Para ello hay que volver a cargar el archivo que contiene al modelo e introducirle los datos para observar las predicciones. Hemos realizado previamente con PortScan y DDoS esta tarea,

donde hemos confirmado que el modelo predice correctamente sobre conjuntos de datos dónde se están produciendo estos ataques. Ahora sabiendo que ataque está ocurriendo pasamos a elegir el código de postprocesado adecuando.

3 Mitigación del ataque

Cuando el modelo predictor nos devuelve el ataque que está ocurriendo, entonces se utiliza el código de postprocesado de dicho ataque para que lo bloquee de manera personalizada. Con esto ya tendríamos todo el proceso de *IceWall* completado y ahora volveríamos a repetir todo para los nuevos datos que se vayan leyendo.

En resumen, el proceso de *IceWall* va a ser un bucle que lea datos de una red y vaya efectuando los pasos previamente comentados para mitigar los ataques que van ocurriendo. Es importante destacar que es necesario aportar los archivos de datos que se pretenden preprocesar para su funcionamiento, es decir, no funciona a tiempo real aunque la previsión futura es que así sea.

Capítulo 5

Conclusiones y trabajo futuro

A lo largo de este documento hemos ido paso a paso construyendo una manera de detectar y bloquear ataques de red utilizando inteligencia artificial. Hemos llegado a la conclusión de que para los ataques PortScan y DDoS esta técnica funciona para el objetivo buscado. Recordemos cuales son estos objetivos y veamos que, en efecto, se han llegado a conseguir:

- **Revisar y analizar los tipos de ataques y formas de mitigarlos.**

  Este objetivo ha sido cubierto en el capítulo 3, en el cual hemos visto los diferentes ataques y contra medidas de los mismos.

- **Entrenar un modelo de *machine learning* que permita identificar qué tipo de ataque se está produciendo.**

  También se ha logrado este objetivo a lo largo del capítulo 4 donde se ha ahondado en lo relativo al entrenamiento, así como en los resultados del mismo. El modelo predictivo resultante cumple lo esperado.

- **Crear códigos de post procesado que permitan defender de uno o dos de los posibles ataques.**

  Del mismo modo que el entrenamiento, hemos explicado el funcionamiento de estos códigos de post procesado en el capítulo 4. En combinación con el modelo predictivo son capaces de mitigar los ataques como se pretendía.

- **Comprobar la utilidad o no de una herramienta de este estilo, en caso de ser posible.**

  Después de todo lo explicado previamente, observamos que *IceWall* funciona de la manera que se esperaba. Es por ello por lo que podemos concluir que este tipo de herramienta es útil.

En resumen, podemos concluir que los objetivos han sido cumplidos con éxito, no obstante siempre hay margen de mejora, sobre todo si la herramienta es tan modular como lo es *IceWall*. Por lo tanto vamos a ver unas cuantas posibles mejoras que podrían hacerse a la herramienta en un futuro:

- **Ampliar el modelo para más ataques**

  Hemos ejemplificado con dos ataques comunes pero en el conjunto de datos hay muchos más ataques que interesaría detectar y frenar. También es necesario añadir más conjuntos de datos con otros ataques para así mejorar cada vez más el predictor de ataques.

- **Crear más códigos de postprocesado y mejorar los existentes**

  Aunque es cierto que los códigos de postproceso bloquean los ataques que esperamos, convendría probar con más datos de otros ámbitos para ver si bloquean los ataques correctamente. También habría que crear códigos de postproceso derivados de aumentar el alcance del modelo predictor.

- **Agrupar todo el trabajo para tener una herramienta consistente**

  Tenemos por módulos el modelo predictor y los códigos de postprocesado. Pero el objetivo final de *IceWall* es tener una estructura robusta que recoja datos a tiempo real, los preprocese, prediga los ataques y, tras ello, elija el postproceso correcto y bloquee así el ataque.

Chapter 6

Conclusions and Future Work

Throughout this document, we have step-by-step constructed a method to detect and block network attacks using artificial intelligence. We have concluded that for PortScan and DDoS attacks, this technique achieves the intended goal. Let's remember what these goals are and see that, indeed, they have been achieved:

- **Review and analyze types of attacks and ways to mitigate them.** This goal was covered in Chapter 3, where we looked at different attacks and countermeasures.
- **Train a machine learning model that can identify what type of attack is occurring.**

  This goal has also been achieved throughout Chapter 4, which delved into training and its results. The resulting predictive model meets expectations.
- **Create post-processing codes that can defend against one or two possible attacks.**

  Just as with the training, we have explained the operation of these post-processing codes in Chapter 4. In combination with the predictive model, they are capable of mitigating the attacks as intended.
- **Verify the utility of a tool of this kind, if possible.**

  After all the explanations provided, we observe that *IceWall* functions as expected. Therefore, we can conclude that this type of tool is useful.

In summary, we can conclude that the goals have been successfully met, although there is always room for improvement, especially since the tool is as modular as *IceWall*. Therefore, let's look at a few possible improvements that could be made to the tool in the future:

- **Expand the model to include more attacks** We have exemplified with two common attacks, but there are many more attacks in the dataset that would be interesting to detect and stop. It is also necessary to add more datasets with other attacks to continually improve the attack predictor.

- **Create more post-processing codes and improve the existing ones**

  Although the current post-process codes block the attacks we expect, it would be advisable to test with more data from other fields to see if they correctly block attacks. We should also create post-process codes derived from expanding the scope of the predictive model.

- **Aggregate all the work to have a consistent tool**

  We currently have the predictive model and the post-processing codes as modules. However, the ultimate goal of *IceWall* is to have a robust structure that collects real-time data, preprocesses it, predicts attacks, and then selects the correct post-process to block the attack.

Bibliografía

Cisco. Snort - Network Intrusion Detection & Prevention System. https://snort.org/, 2024.

Cisco Firepower. Cisco firepower next-generation firewall data sheet, 2018.

Cournapeau, D. Scikit-learn. https://scikit-learn.org/stable/, 2024.

DarkTrace. Darktrace | ciberseguridad que aprende. https://es.darktrace.com/, 2023.

Dhavaji, C. How Proxy Server Works. https://chethana-arachchi.medium.com/firewall-mechanisms-ee1424c9d799, 2021.

Fabero Jiménez, J. C. Ampliación de Redes, 2023.

Fyodor. *Técnicas de sondeo de puertos*. Nmap, 2005.

Geeksforgeeks. How SYN cookies are used to preventing SYN Flood attack. https://www.geeksforgeeks.org, 2021.

Haris, S., Ahmad, R. y Ghani, M. Detecting TCP SYN Flood Attack based on Anomaly Detection. 2010.

Kostas, K. Anomaly Detection in Networks Using Machine Learning. 2018.

McKinney, W. Pandas - python data analysis library. https://pandas.pydata.org/, 2024.

Moreno Vozmediano, R. Tema 4 Ataques y Mecanismos de Seguridad en Redes. Cortafuegos, 2023a.

Moreno Vozmediano, R. Tema 4 Ataques y Mecanismos de Seguridad en Redes. Detección de intrusos, 2023b.

Moreno Vozmediano, R. Tema 4 Ataques y Mecanismos de Seguridad en Redes. Vulnerabilidades en protocolos de red y ataques, 2023c.

OISF. Suricata ids/ips.
https://suricata.io/features/, 2024.

Palo Alto Networks. Palo alto networks next-generation firewall.
https://www.paloaltonetworks.com/network-security/next-generation-firewall, 2024.

Stephen Huang, S.-H. y Cao, Z. Detecting malicious users behind circuit-based anonymity networks. 2020.

The Zeek Project. The zeek network security monitoring tool.
https://zeek.org/, 2024.

Apéndice A

Lista de características y explicaciones

№	Feature Name	Feature Description
1	Flow ID	Flow ID
2	Source IP	Source IP
3	Source Port	Source Port
4	Destination IP	Destination IP
5	Destination Port	Destination Port
6	Protocol	Protocol
7	Timestamp	Timestamp
8	Flow Duration	Duration of the flow in Microsecond
9	Total Fwd Packets	Total packets in the forward direction
10	Total Backward Packets	Total packets in the backward direction
11	Total Length of Fwd Packets	Total size of packet in forward direction
12	Total Length of Bwd Packets	Total size of packet in backward direction
13	Fwd Packet Length Max	Maximum size of packet in forward direction
14	Fwd Packet Length Min	Minimum size of packet in forward direction
15	Fwd Packet Length Mean	Mean size of packet in forward direction
16	Fwd Packet Length Std	Standard deviation size of packet in forward direction
17	Bwd Packet Length Max	Maximum size of packet in backward direction
18	Bwd Packet Length Min	Minimum size of packet in backward direction
19	Bwd Packet Length Mean	Mean size of packet in backward direction
20	Bwd Packet Length Std	Standard deviation size of packet in backward direction
21	Flow Bytes/s	Number of flow bytes per second
22	Flow Packets/s	Number of flow packets per second
23	Flow IAT Mean	Mean length of a flow
24	Flow IAT Std	Standard deviation length of a flow
25	Flow IAT Max	Maximum length of a flow
26	Flow IAT Min	Minimum length of a flow
27	Fwd IAT Total	Total time between two packets sent in the forward direction
28	Fwd IAT Mean	Mean time between two packets sent in the forward direction
29	Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
30	Fwd IAT Max	Maximum time between two packets sent in the forward direction
31	Fwd IAT Min	Minimum time between two packets sent in the forward direction
32	Bwd IAT Total	Total time between two packets sent in the backward direction
33	Bwd IAT Mean	Mean time between two packets sent in the backward direction
34	Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
35	Bwd IAT Max	Maximum time between two packets sent in the backward direction
36	Bwd IAT Min	Minimum time between two packets sent in the backward direction
37	Fwd PSH Flags	Number of packets with PUSH
38	Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
39	Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
40	Bwd URG Flags	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
41	Fwd Header Length	Total bytes used for headers in the forward direction
42	Bwd Header Length	Total bytes used for headers in the backrward direction
43	Fwd Packets/s	Number of forward packets per second
44	Bwd Packets/s	Number of backward packets per second

Figura A.1: Lista de características y explicaciones del conjunto de datos CICIDS2017 - Primera Parte.

№	Feature Name	Feature Description
45	Min Packet Length	Minimum inter-arrival time of packet
46	Max Packet Length	Maximum inter-arrival time of packet
47	Packet Length Mean	Mean inter-arrival time of packet
48	Packet Length Std	Standard deviation inter-arrival time of packet
49	Packet Length Variance	Packet Length Variance
50	FIN Flag Count	Number of packets with FIN
51	SYN Flag Count	Number of packets with SYN
52	RST Flag Count	Number of packets with RST
53	PSH Flag Count	Number of packets with PUSH
54	ACK Flag Count	Number of packets with ACK
55	URG Flag Count	Number of packets with URG
56	CWE Flag Count	Number of packets with CWE
57	ECE Flag Count	Number of packets with ECE
58	Down/Up Ratio	Download and upload ratio
59	Average Packet Size	Average size of packet
60	Avg Fwd Segment Size	Average size observed in the forward direction
61	Avg Bwd Segment Size	Average size observed in the backward direction
62	Fwd Avg Bytes/Bulk	Average number of bytes bulk rate in the forward direction
63	Fwd Avg Packets/Bulk	Average number of packets bulk rate in the forward direction
64	Bwd Avg Bulk Rate	Average number of bulk rate in the backward direction
65	Bwd Avg Bytes/Bulk	Average number of bytes bulk rate in the backward direction
66	Bwd Avg Packets/Bulk	Average number of packets bulk rate in the backward direction
67	Bwd Avg Bulk Rate	Average number of bulk rate in the backward direction
68	Subflow Fwd Packets	The average number of packets in a sub flow in the forward direction
69	Subflow Fwd Bytes	The average number of bytes in a sub flow in the forward direction
70	Subflow Bwd Packets	The average number of packets in a sub flow in the backward direction
71	Subflow Bwd Bytes	The average number of bytes in a sub flow in the backward direction
72	Init_Win_bytes_forward	The total number of bytes sent in initial window in the forward direction
73	Init_Win_bytes_backward	The total number of bytes sent in initial window in the backward direction
74	act_data_pkt_fwd	Count of packets with at least 1 byte of TCP data payload in the forward direction
75	min_seg_size_forward	Minimum segment size observed in the forward direction
76	Active Mean	Mean time a flow was active before becoming idle
77	Active Std	Standard deviation time a flow was active before becoming idle
78	Active Max	Maximum time a flow was active before becoming idle
79	Active Min	Minimum time a flow was active before becoming idle
80	Idle Mean	Mean time a flow was idle before becoming active
81	Idle Std	Standard deviation time a flow was idle before becoming active
82	Idle Max	Maximum time a flow was idle before becoming active
83	Idle Min	Minimum time a flow was idle before becoming active
84	Label	Label
85	External IP	External IP

Figura A.2: Lista de características y explicaciones del conjunto de datos CI-CIDS2017 - Segunda Parte.

Apéndice B

Importancia de características por ataque

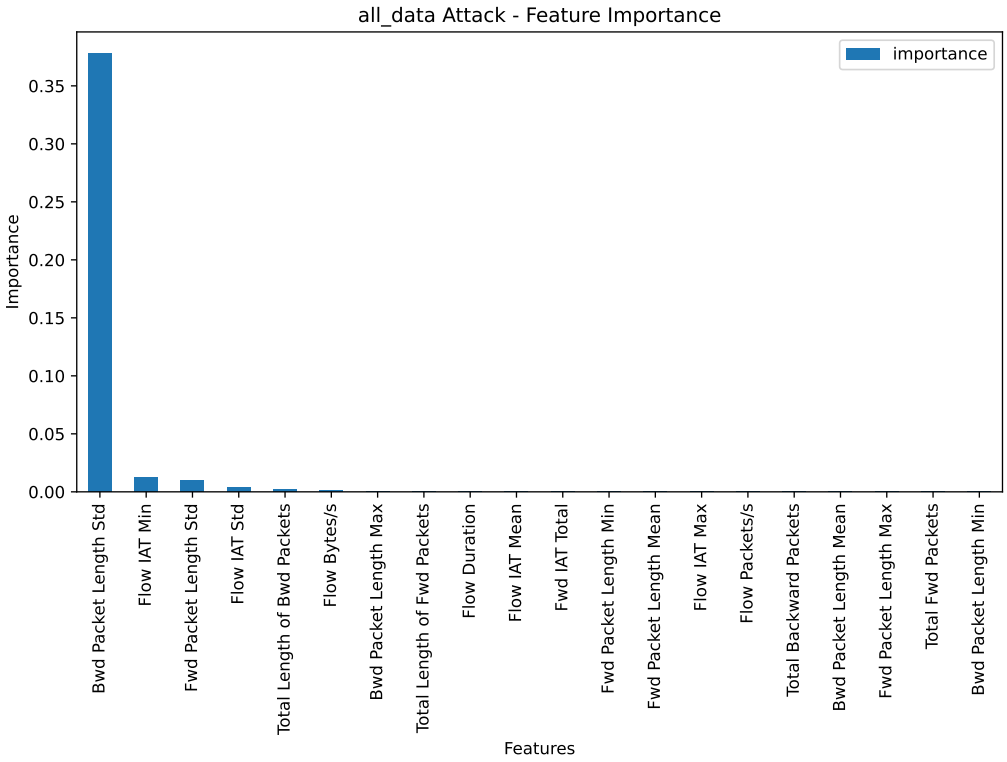

Figura B.1: Características de *all_ data*

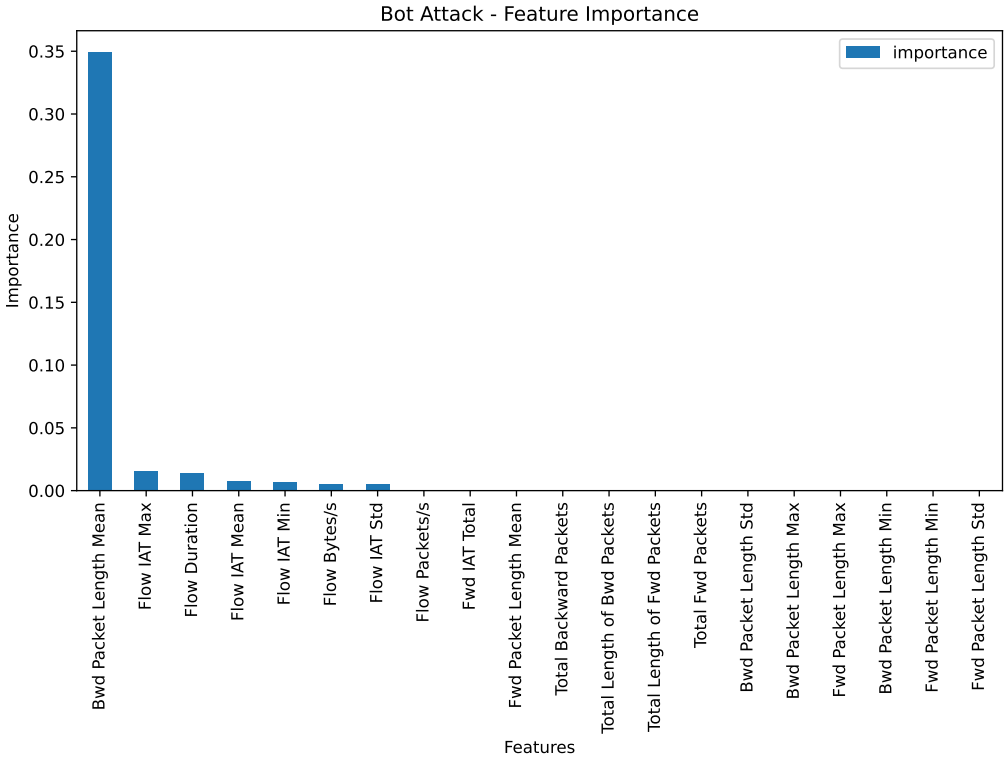

Figura B.2: Características de *bot*

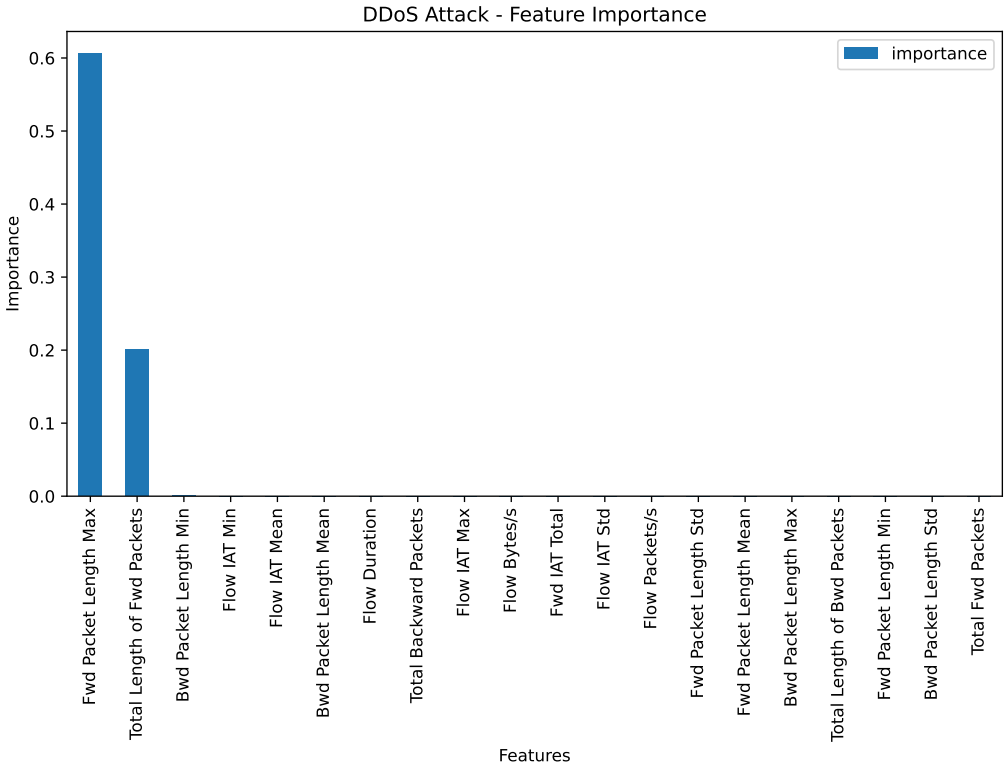

Figura B.3: Características de *DDoS*

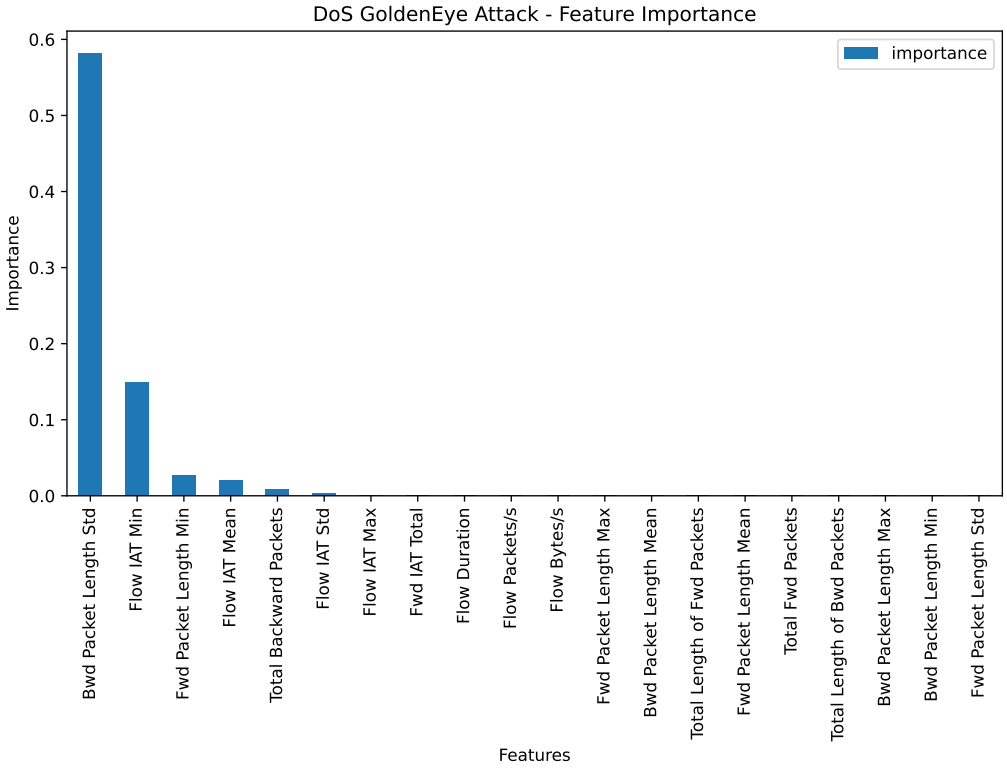

Figura B.4: Características de *DoS GoldenEye*

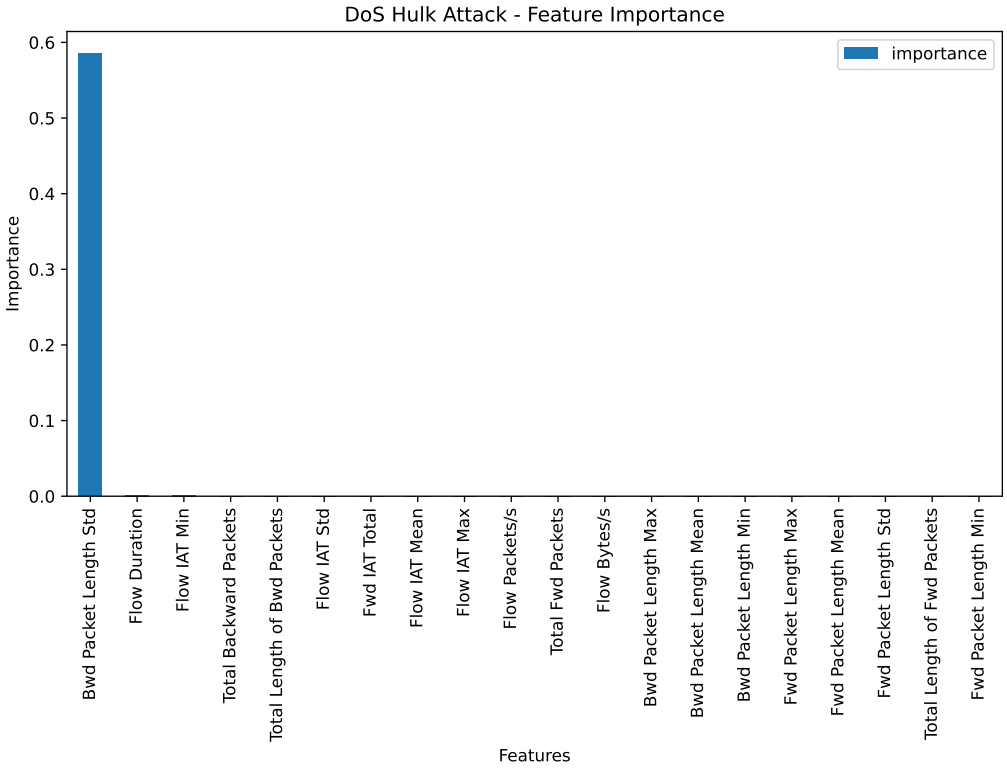

Figura B.5: Características de *DoS Hulk*

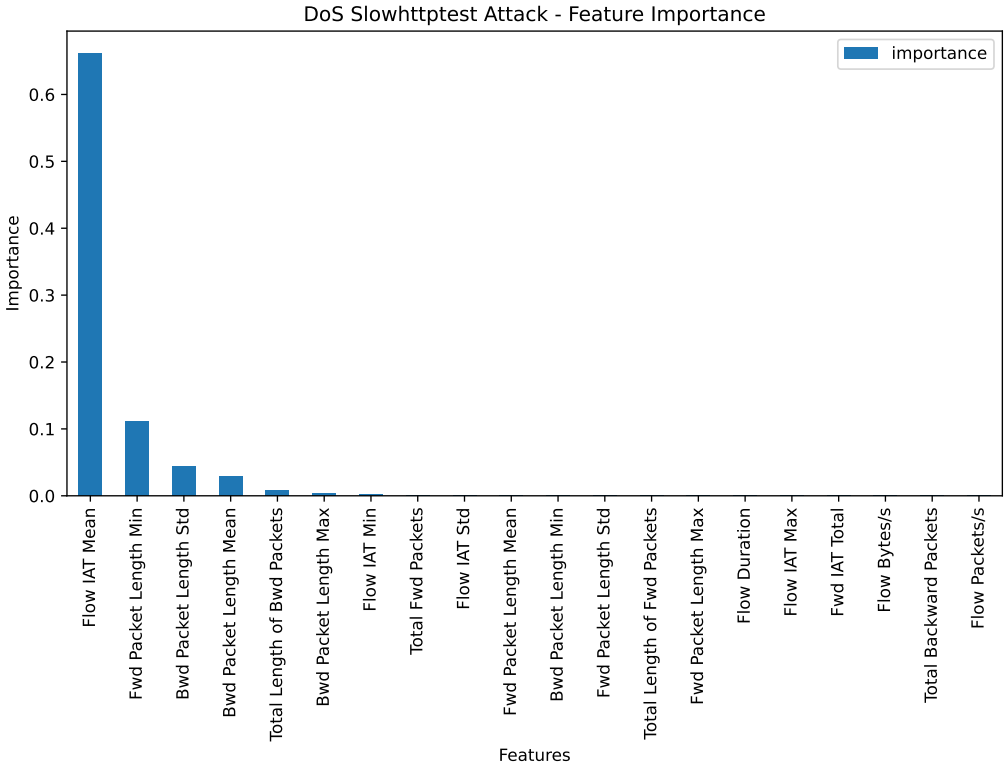

Figura B.6: Características de *DoS Slowhttptest*

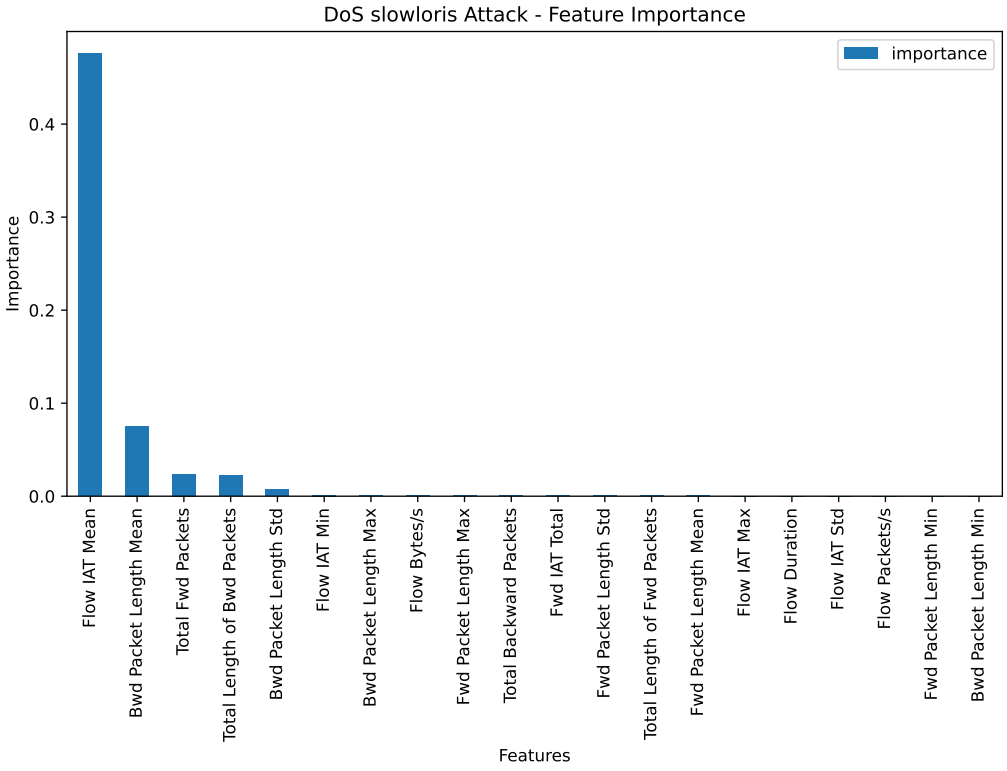

Figura B.7: Características de *DoS slowloris*

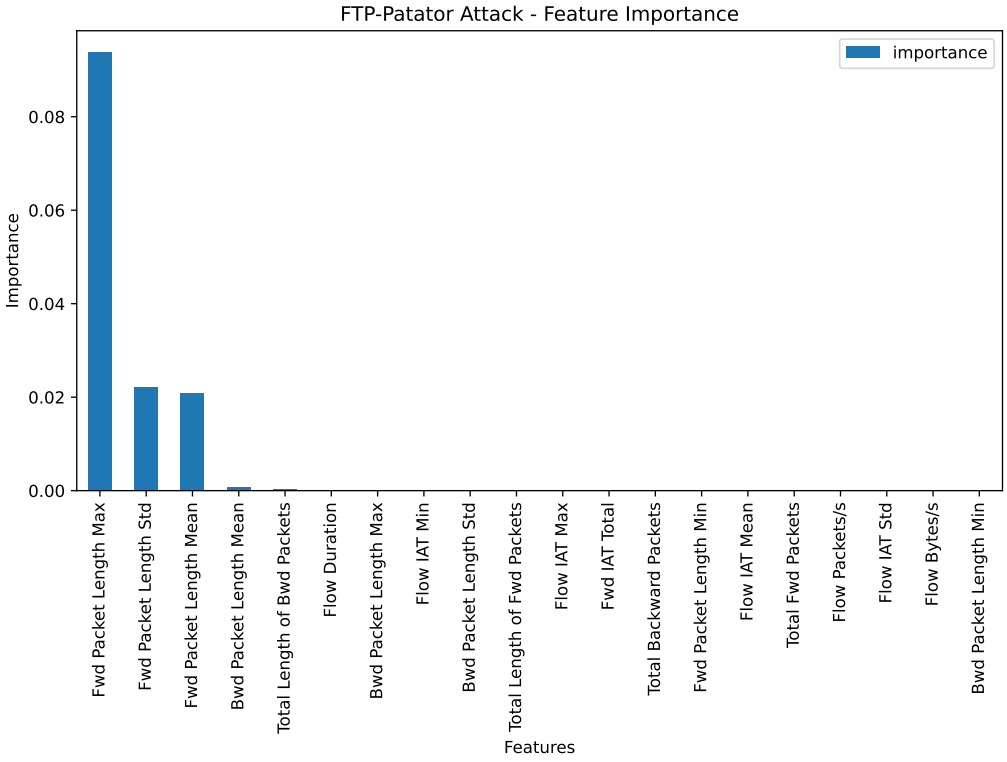

Figura B.8: Características de *FTP-Patator*

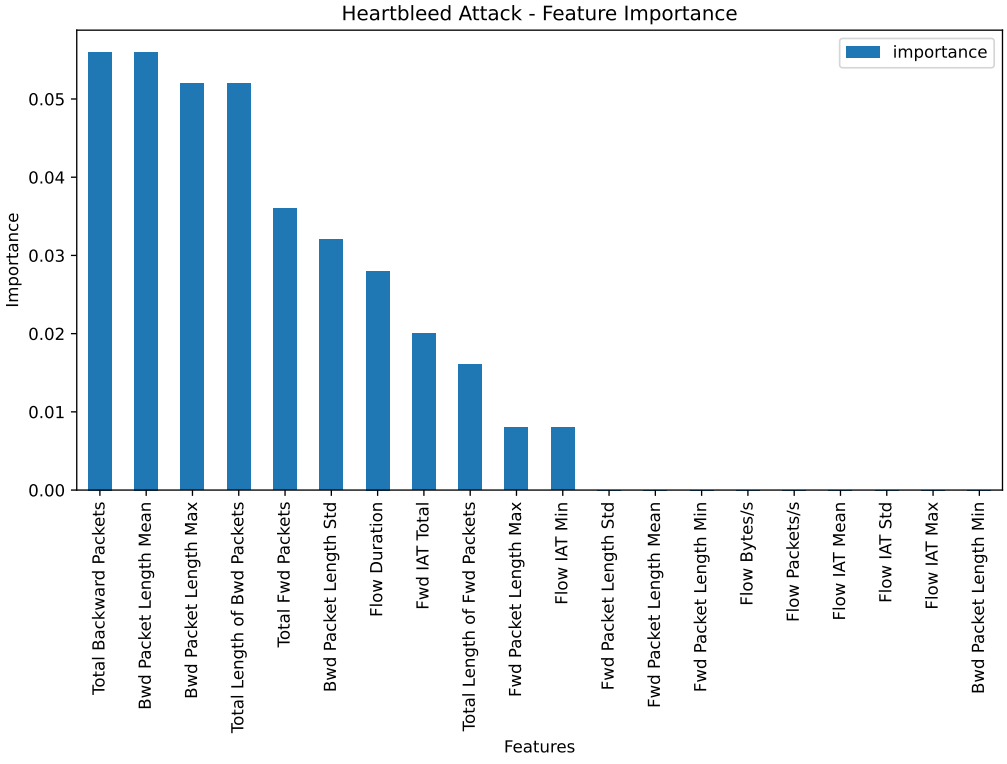

Figura B.9: Características de *Heartbleed*

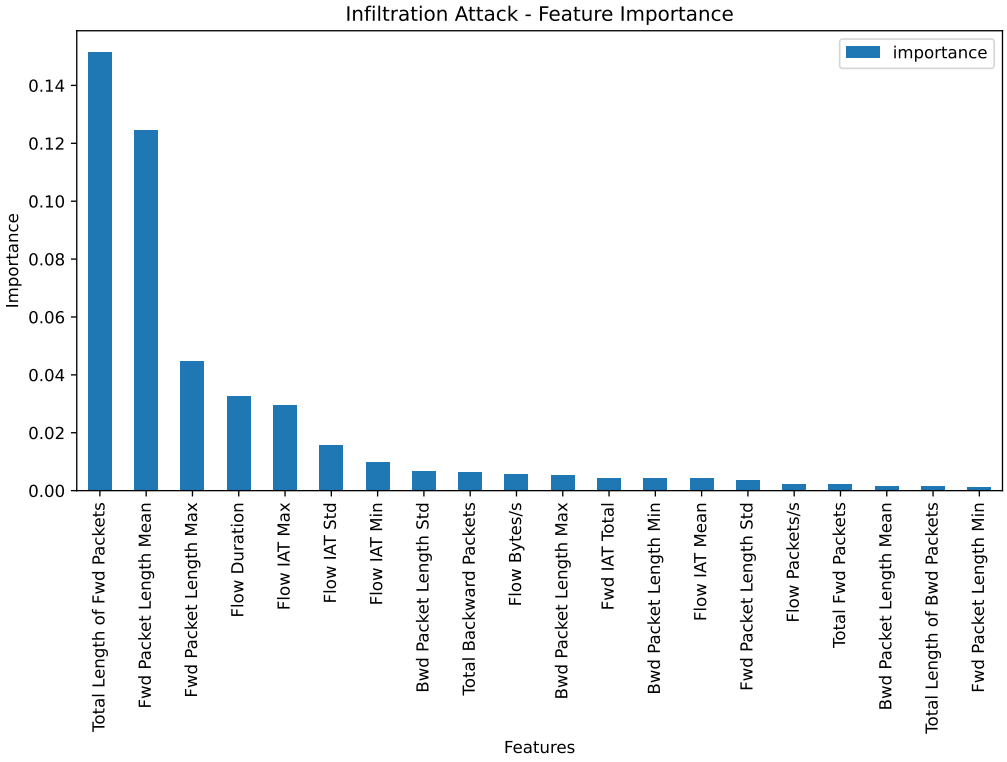

Figura B.10: Características de *Infiltration*

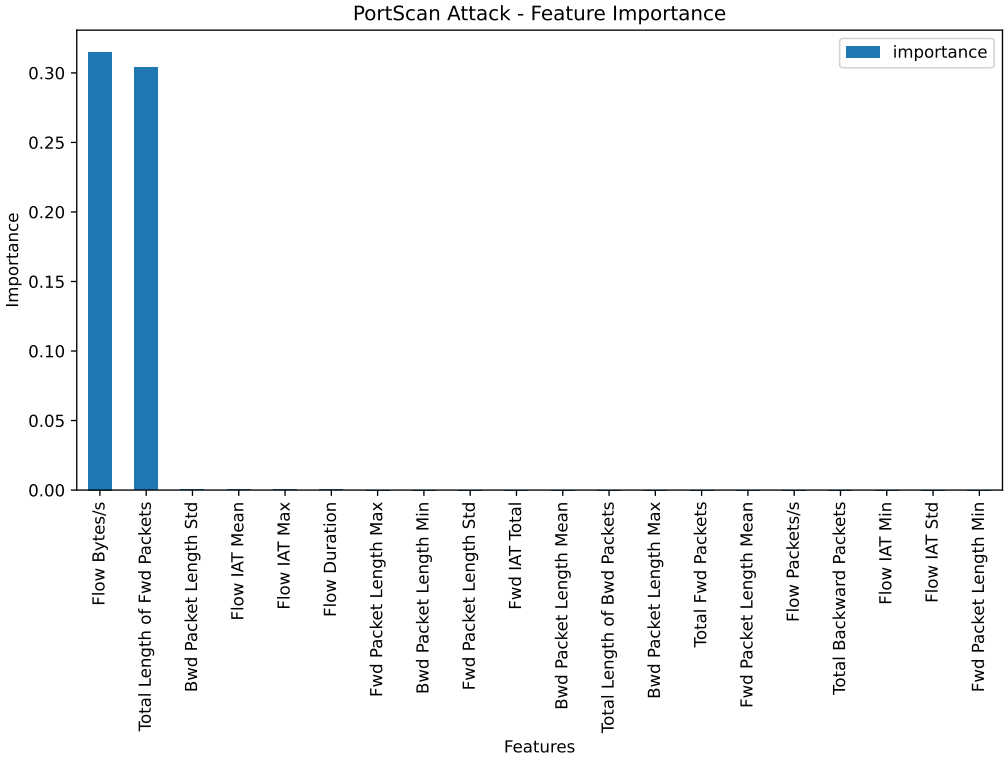

Figura B.11: Características de *PortScan*

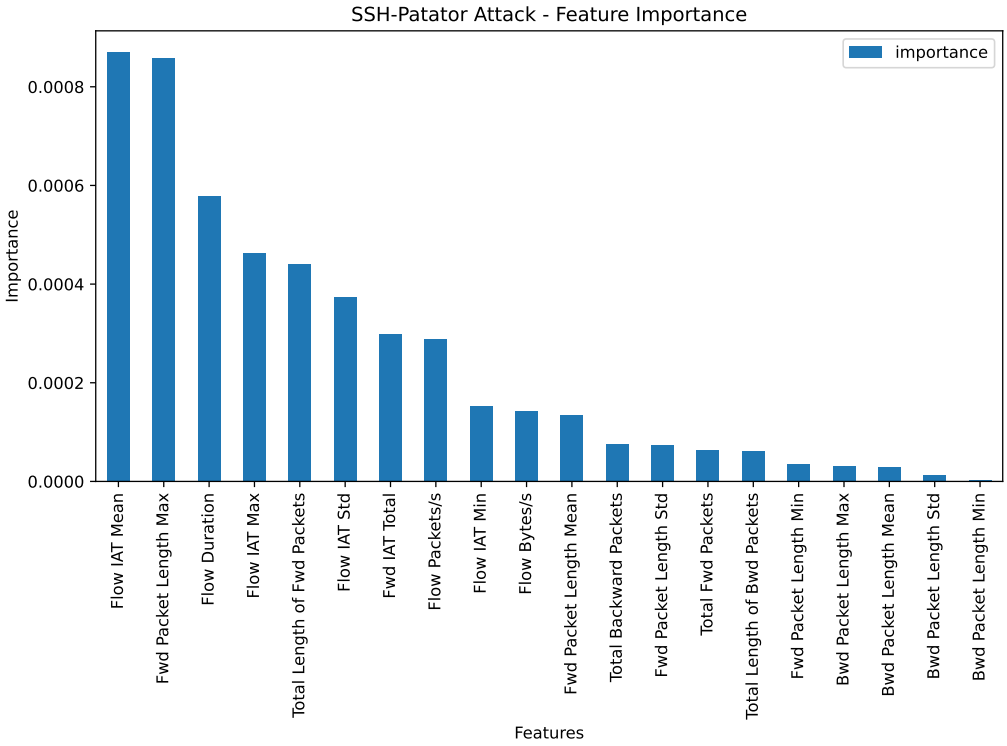

Figura B.12: Características de *SSH-Patator*

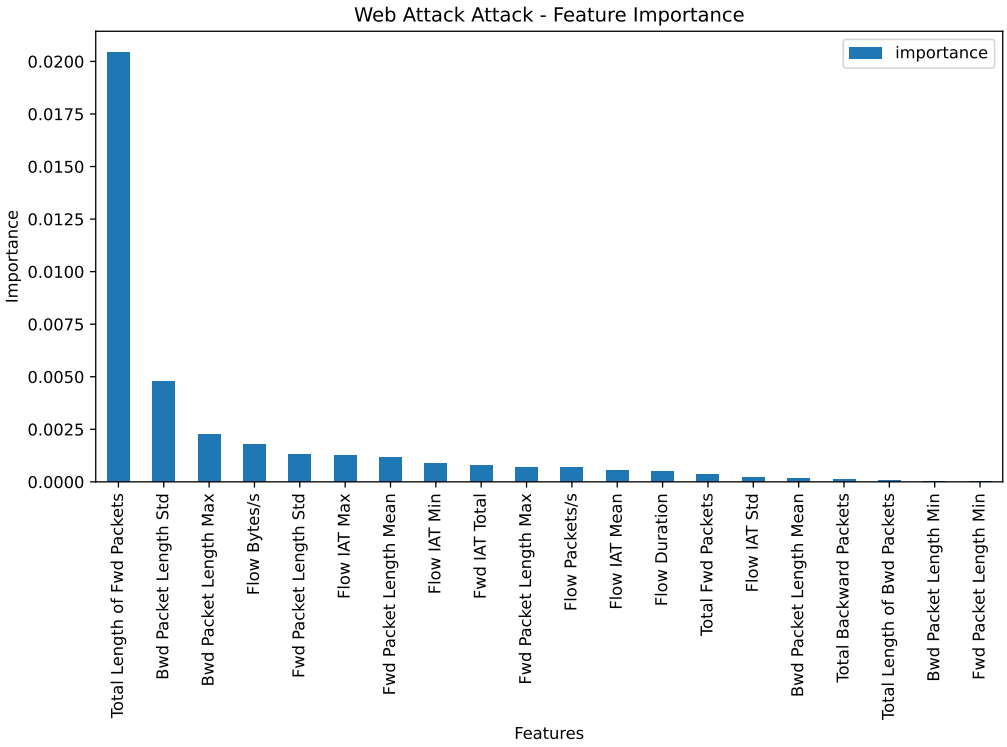

Figura B.13: Características de *Web Attack*

Apéndice C

Resultados de entrenamientos

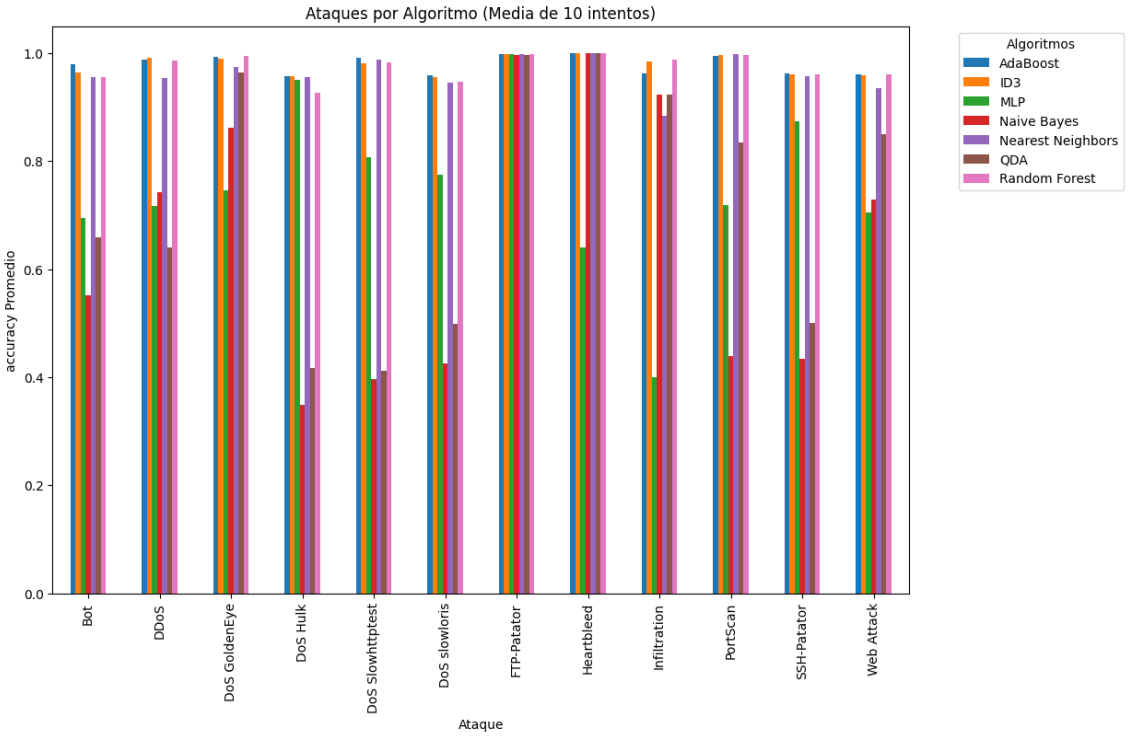

Figura C.1: Accuracy promedio por Ataque

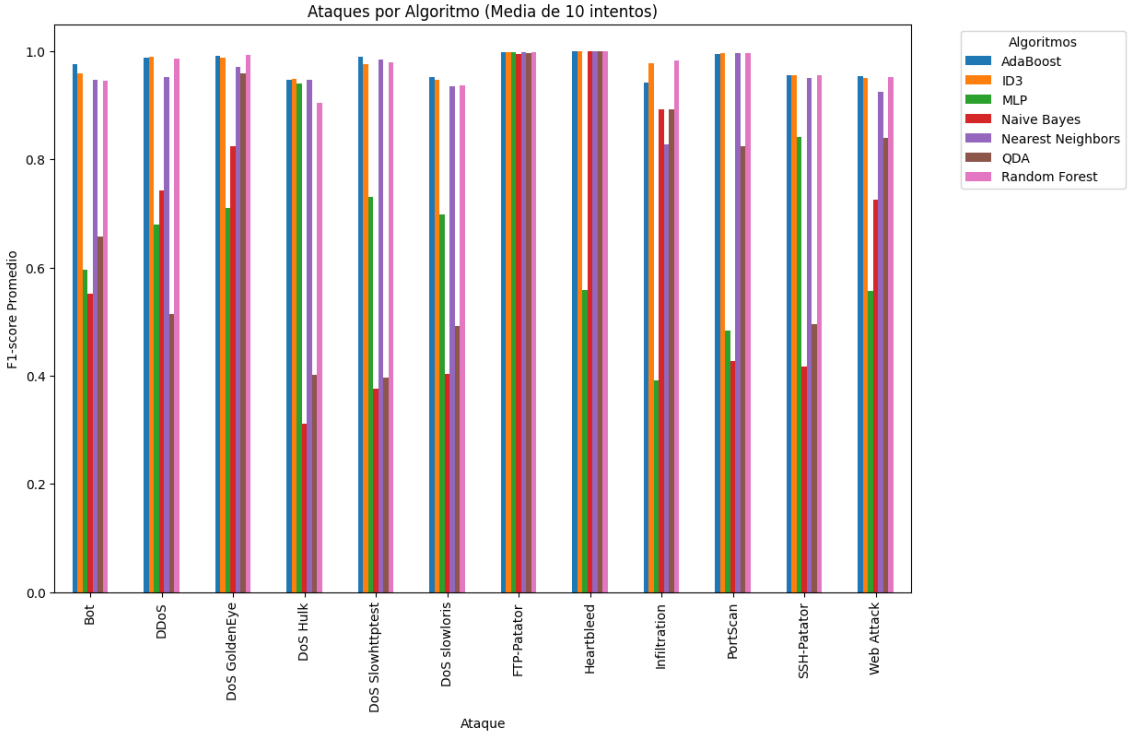

Figura C.2: F1-score promedio por Ataque

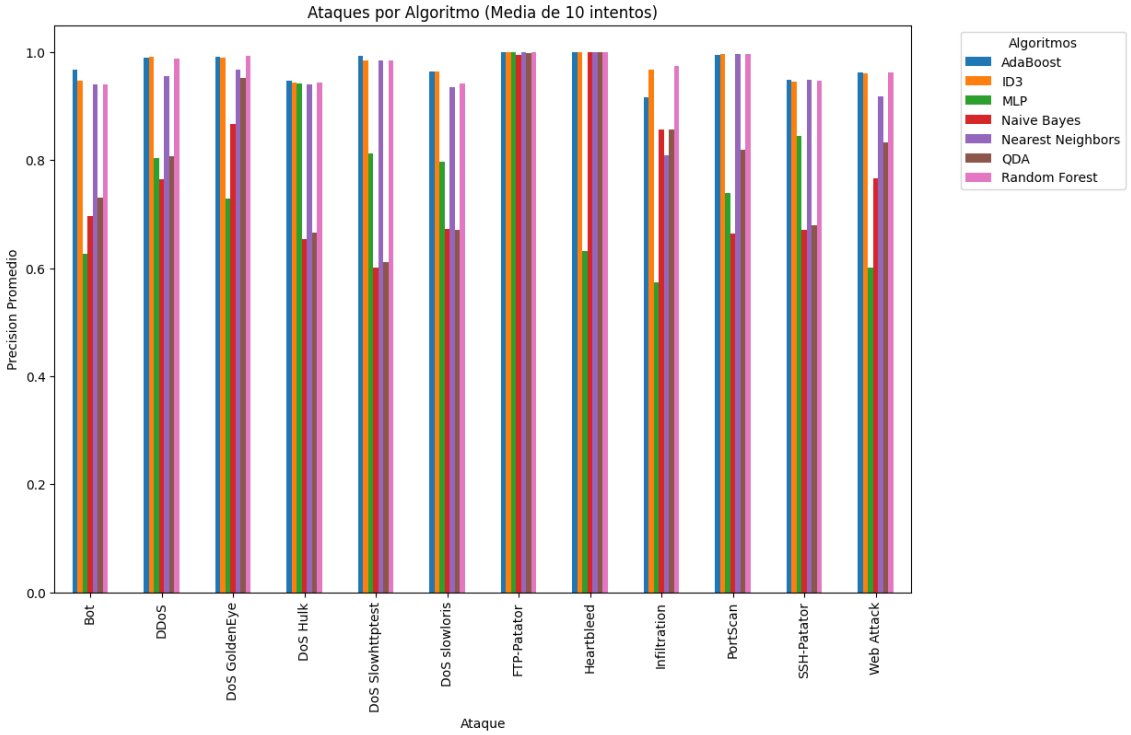

Figura C.3: Precisión promedio por Ataque

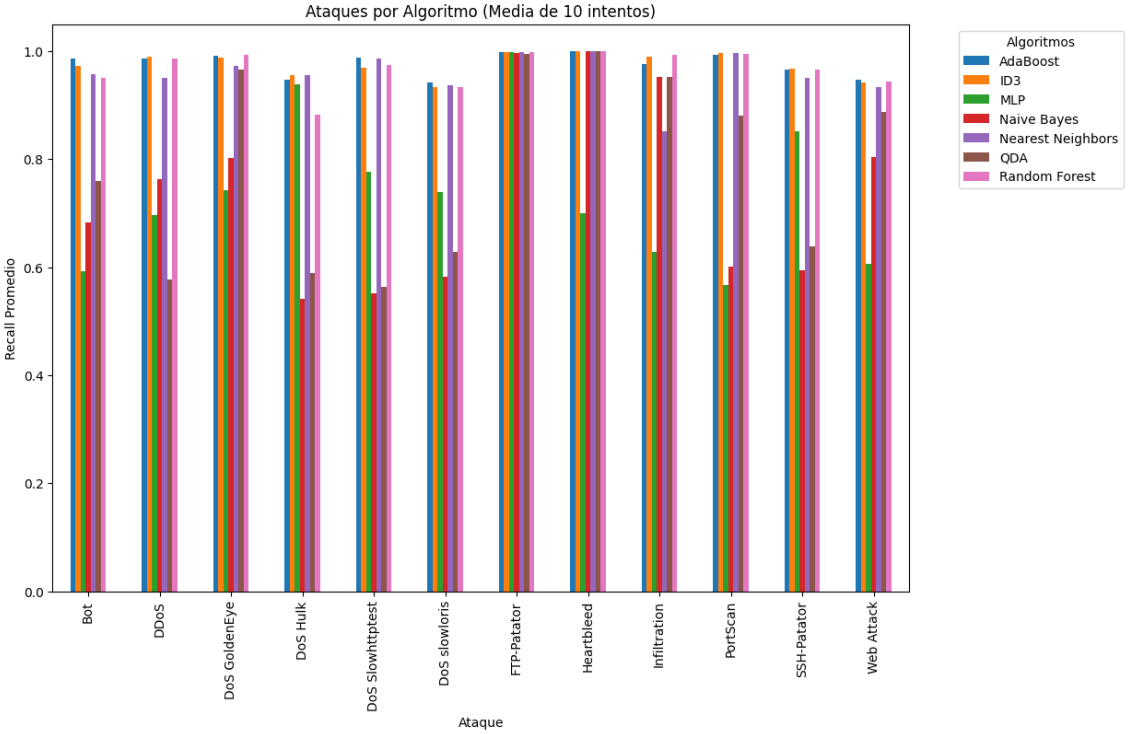

Figura C.4: Recall promedio por Ataque

C.1. Resultados en todos los datos

C.1.1. Results_2

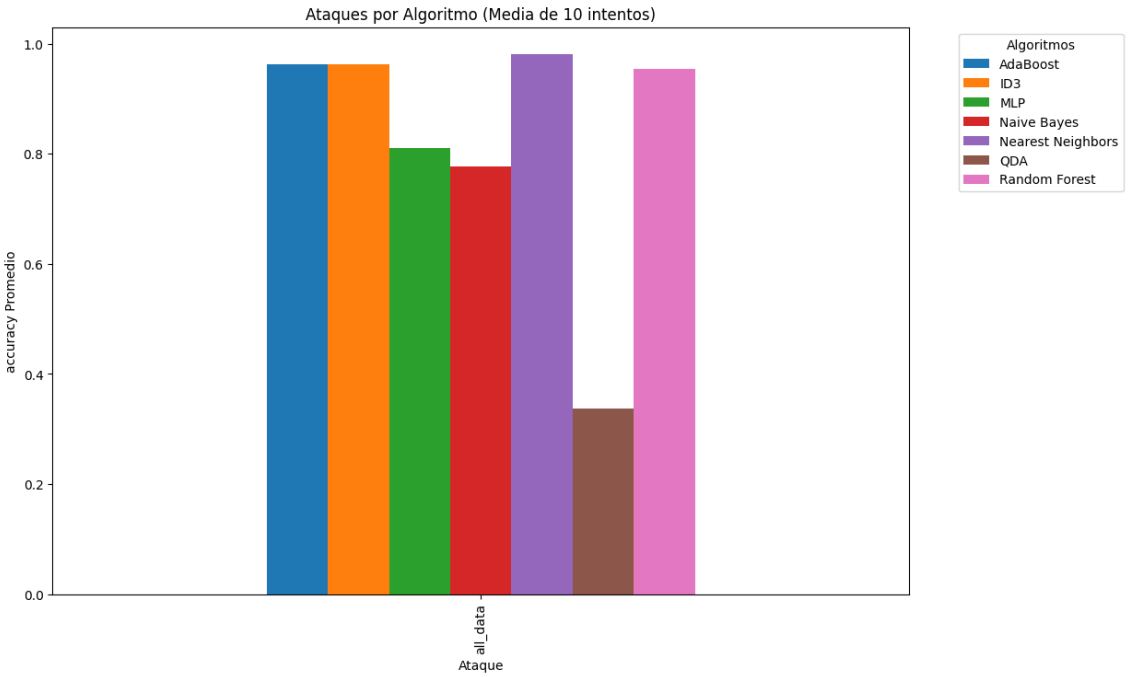

Figura C.5: Accuracy promedio en *all_data*

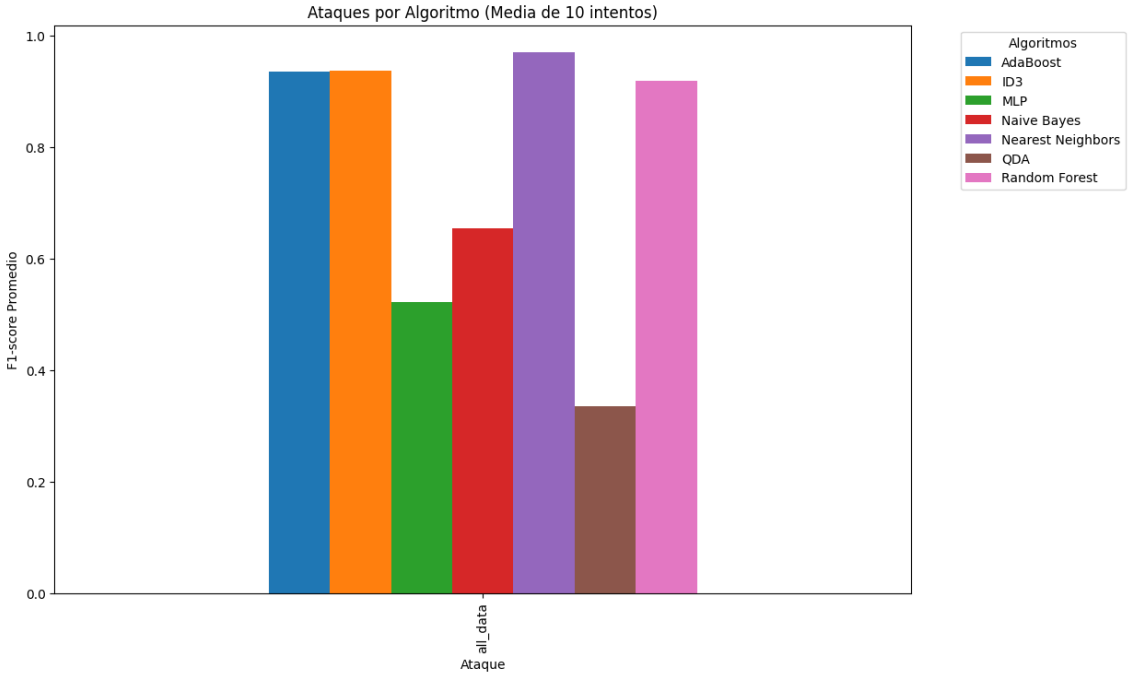

Figura C.6: F1-score promedio en *all_ data*

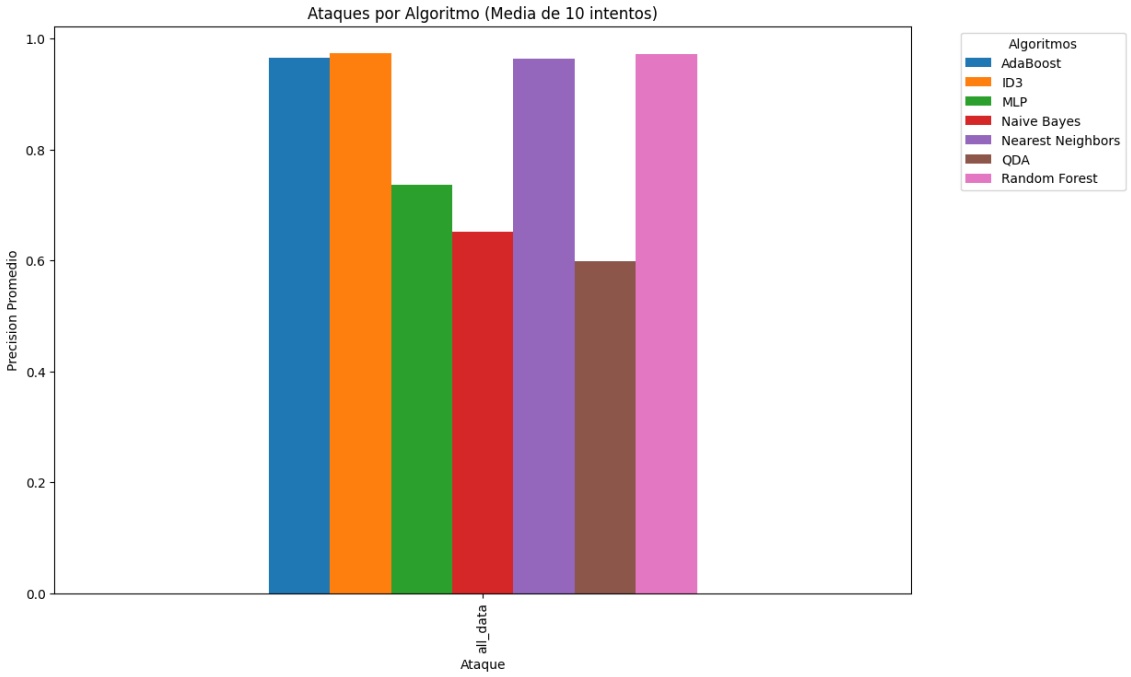

Figura C.7: Precisión promedio en *all_ data*

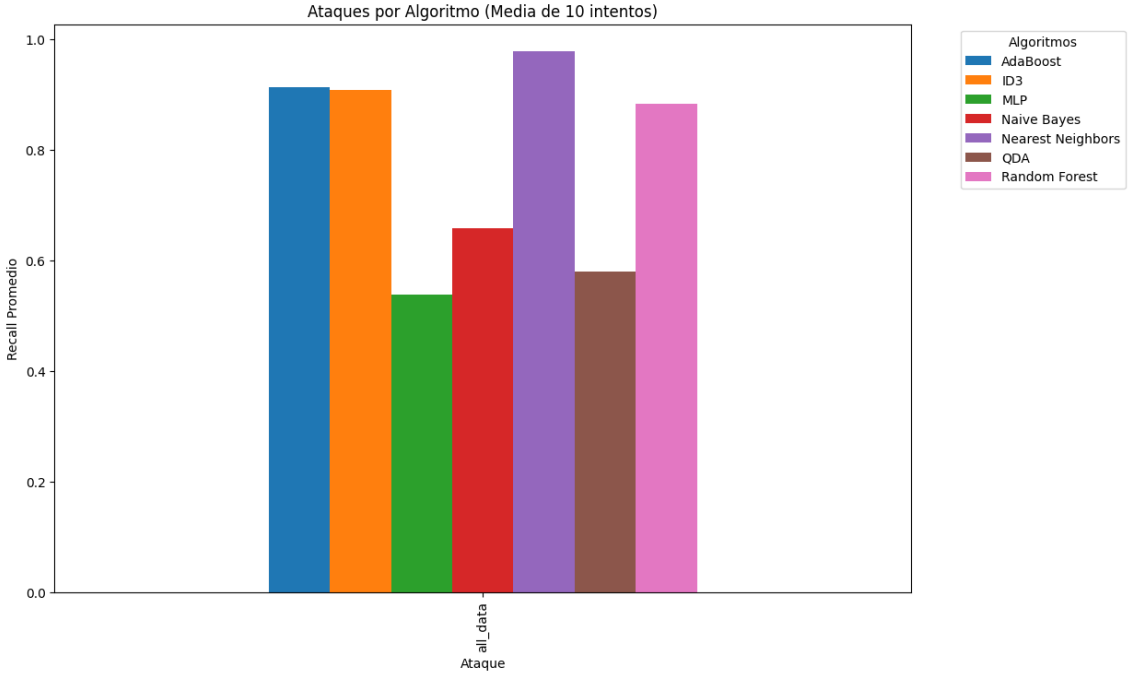

Figura C.8: Recall promedio en *all_data*

C.1.2. Results_3

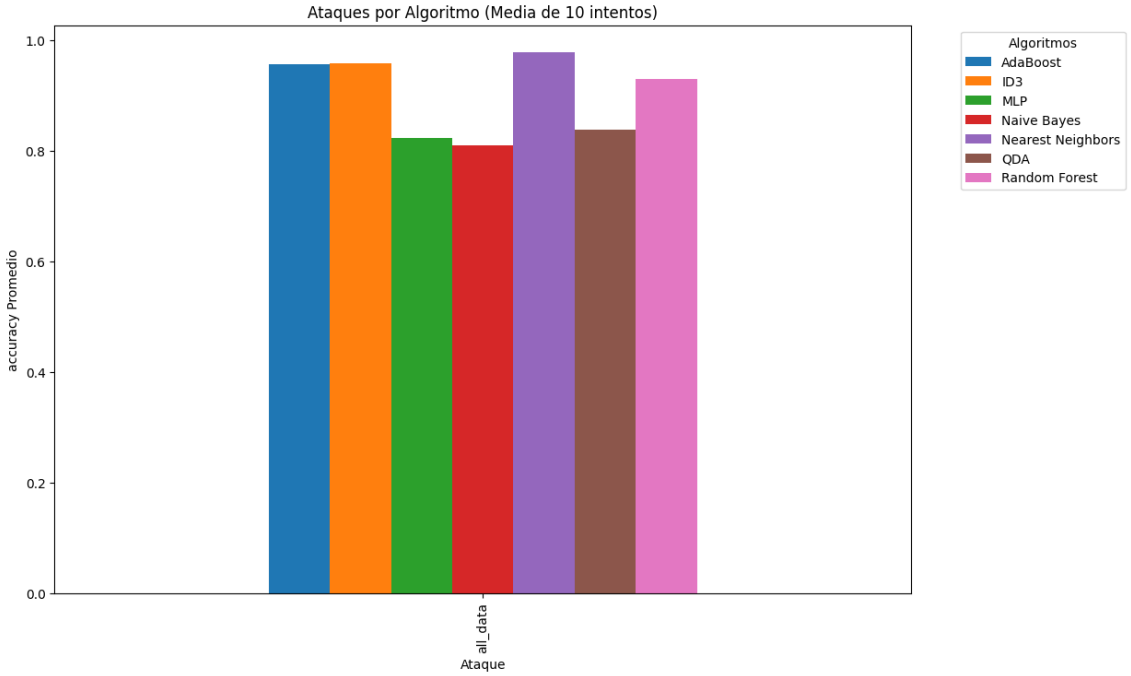

Figura C.9: Accuracy promedio en *all_data*

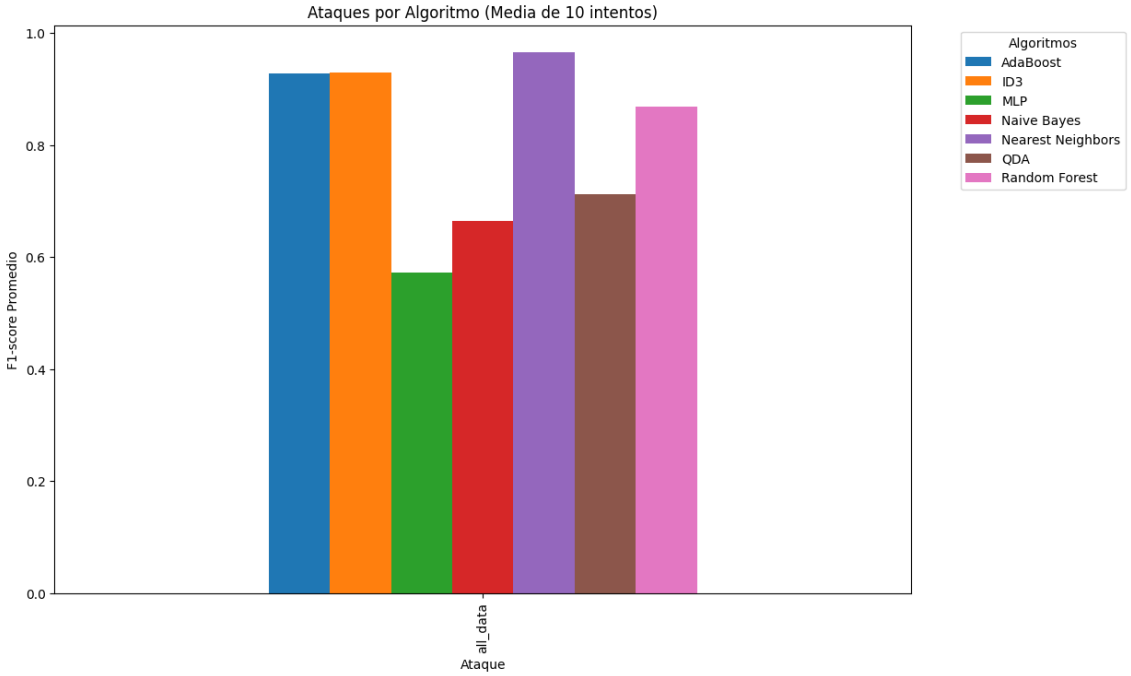

Figura C.10: F1-score promedio en *all_ data*

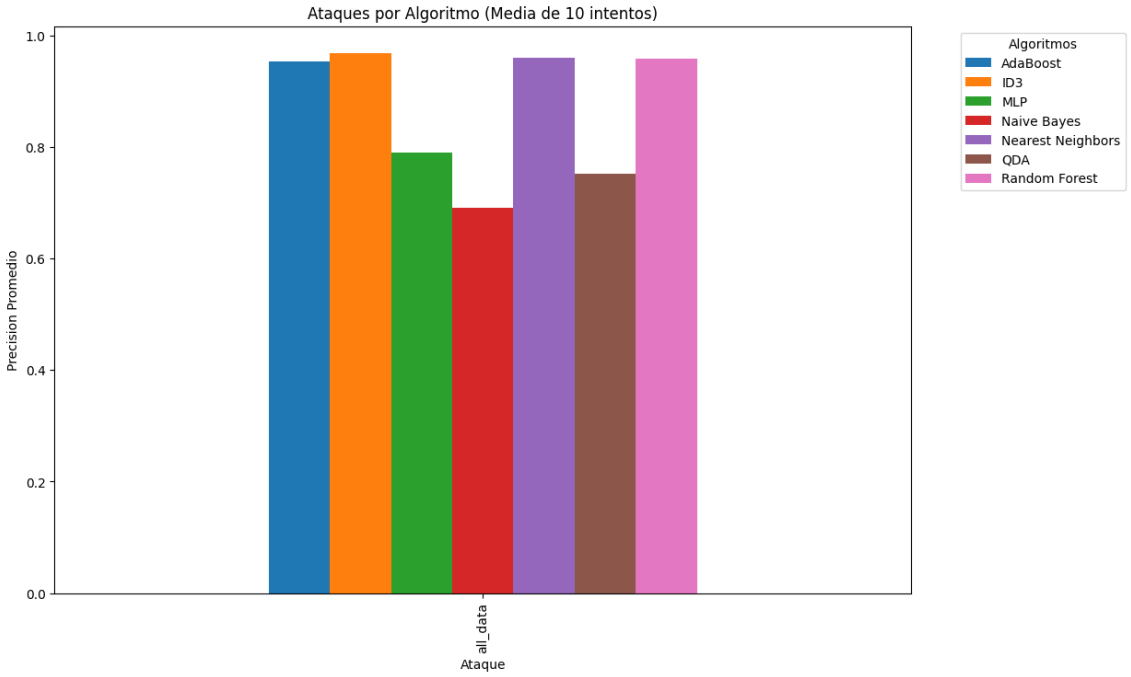

Figura C.11: Precisión promedio en *all_ data*

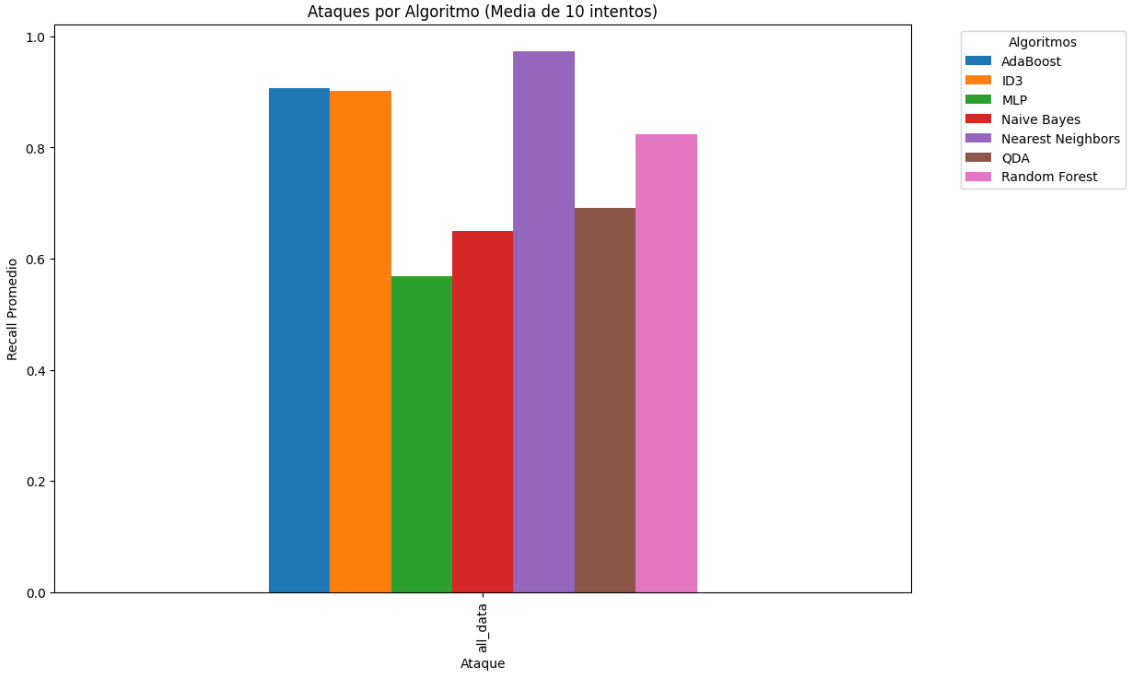

Figura C.12: Recall promedio en *all_data*

C.1.3. Results_Final

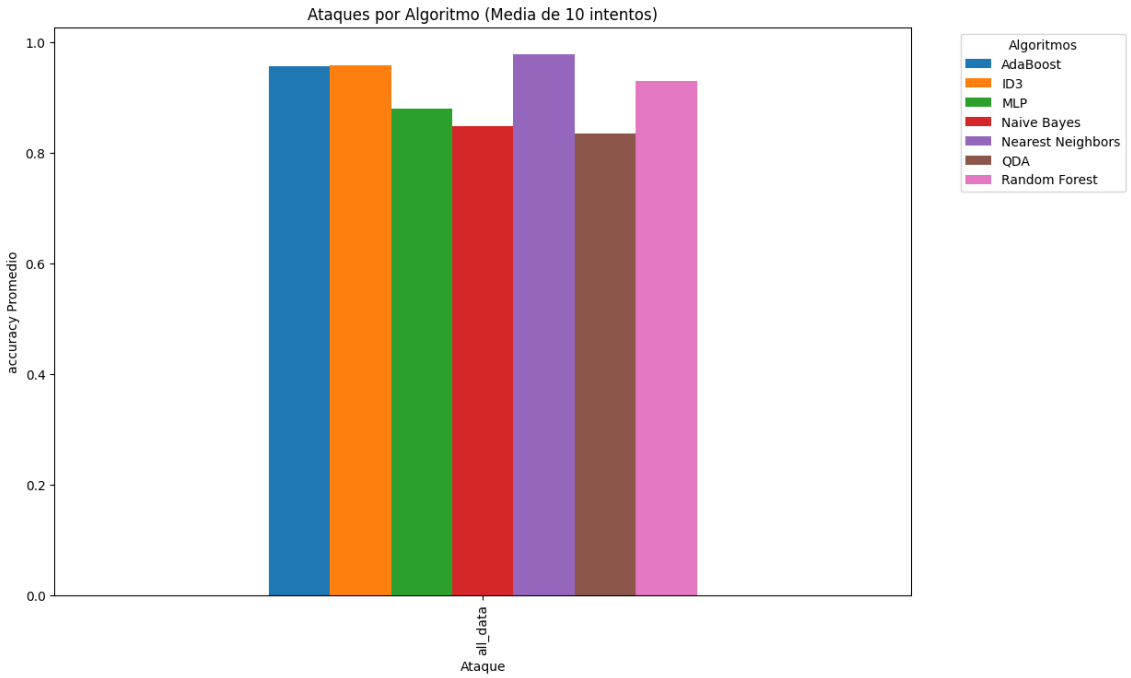

Figura C.13: Accuracy promedio en *all_data*

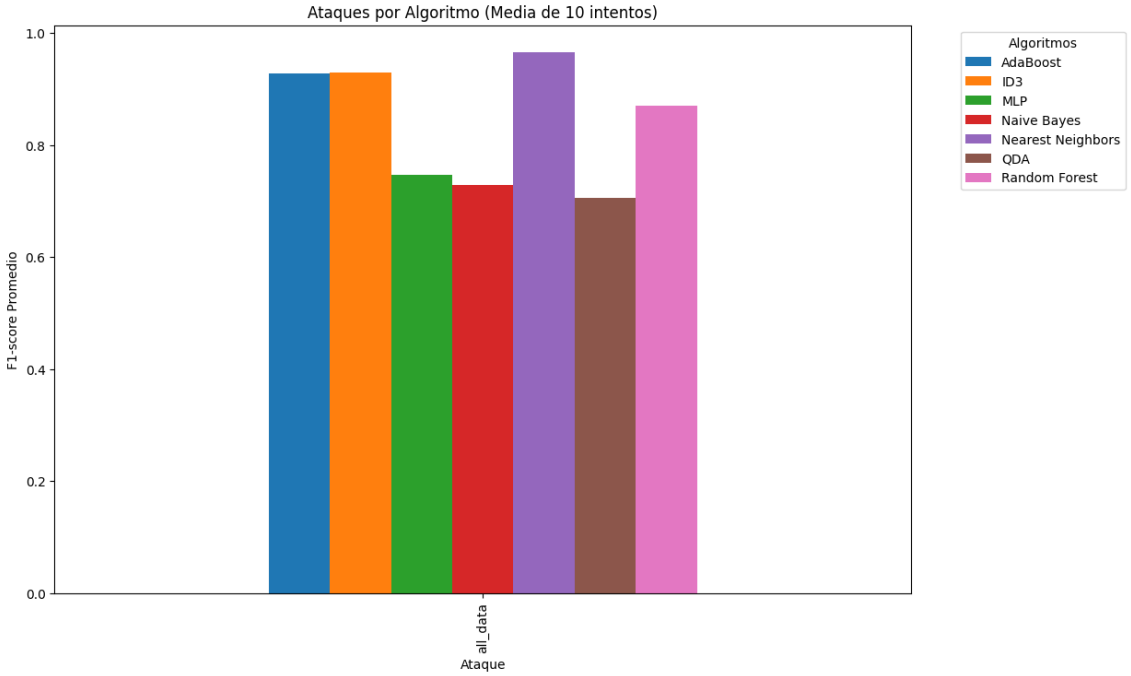

Figura C.14: F1-score promedio en *all_ data*

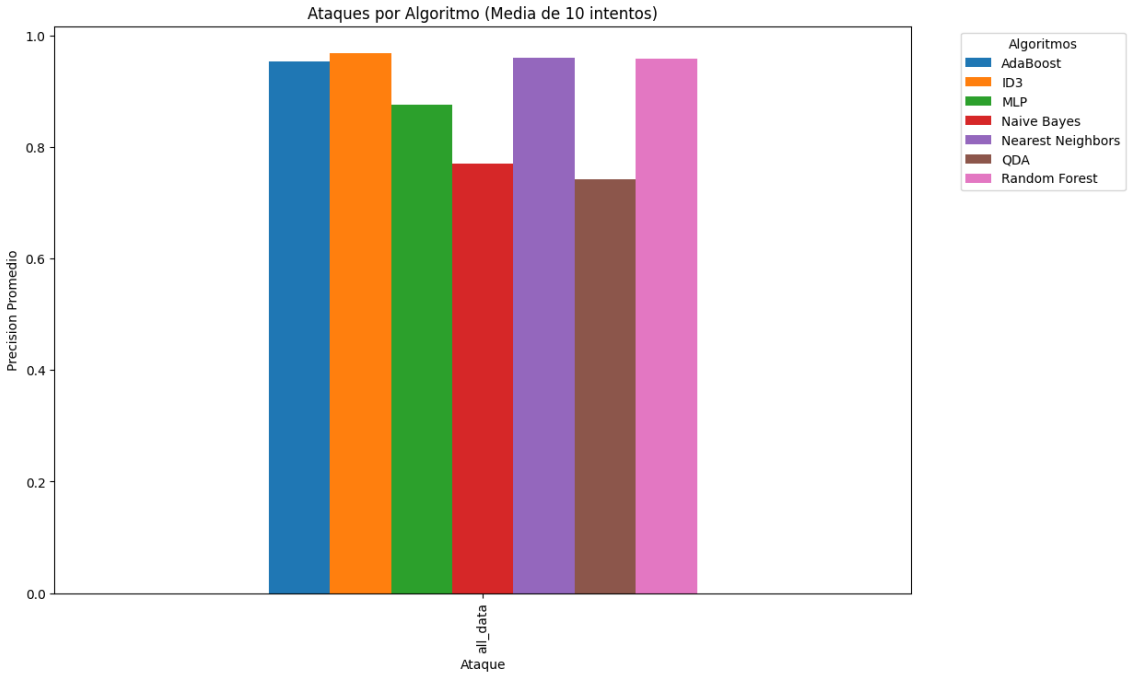

Figura C.15: Precisión promedio en *all_ data*

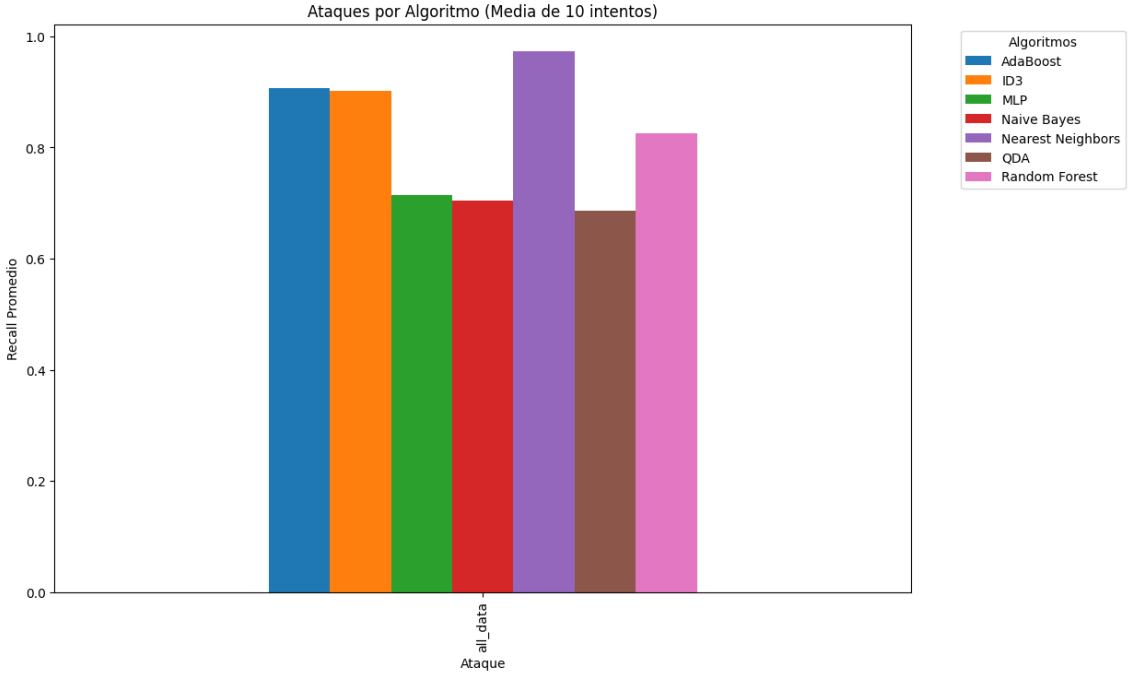

Figura C.16: Recall promedio en *all_data*