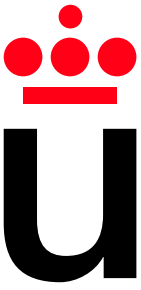

Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Ingeniería de la Ciberseguridad

Curso 2022-2023

Trabajo Fin de Grado

TÉCNICAS DE APRENDIZAJE AUTOMÁTICO PARA LA DETECCIÓN DE ATAQUES DE DENEGACIÓN DE SERVICIO

Autor: Clara Contreras Nevares
Tutor: Marta Beltrán Pardo
Cotutor: Raúl Martín Santamaría

Agradecimientos

En primer lugar, me gustaría agradecer a mis tutores, Marta Beltrán y Raúl Martín, sin los que hacer este TFG hubiese mucho más complicado, por darme la oportunidad de desarrollar este tema, apoyarme y guiarme en el desarrollo del mismo.

Además, me gustaría agradecer a todas las personas que me han ayudado en este trabajo, en especial a Carlos Rabazo, mostrándose siempre dispuesto a ayudar, compartiendo sus conocimientos y apostando por este trabajo y por mí. También a todos los profesores que han despertado en mí interés por el mundo de la ciberseguridad, la inteligencia artificial y las redes y comunicaciones.

Agradecer también a todas las personas que, de una u otra forma, han contribuido con opiniones o comentarios para hacer posible este trabajo.

Por último, mi más sincero agradecimiento a mi familia y amigos, por apoyarme durante todo el proceso de elaboración de este trabajo con amor y paciencia.

Resumen

La pérdida de disponibilidad de los servicios que se ofrecen en Internet o en redes corporativas es una de las principales amenazas a la seguridad en la actualidad. En este trabajo se explora el principal tipo de ataque que puede ocasionar dicha pérdida de disponibilidad, el ataque de Denegación de Servicio (DoS). Y en concreto el que se basa en técnicas volumétricas que consisten en inundar al servidor víctima con tráfico o peticiones en un volumen superior al que es capaz de procesar.

Este trabajo pretende comprobar si es posible clasificar el tráfico de red que llega a un servidor como legítimo o malicioso empleando para ello diferentes técnicas de aprendizaje automático. El objetivo es comprobar además si existe alguna que presente un rendimiento mejor para el tipo de ataque mencionado. Para ello se ha desarrollado una solución que procesa y trata un conjunto de datos abierto relativo al dominio de aplicación escogido, que permite seleccionar el modelo que se desea utilizar con unos parámetros específicos, que lo entrena y mide el rendimiento obtenido. Gracias a esta solución se ha podido comparar el rendimiento de técnicas como la regresión logística, K-Nearest Neighbors (KNN), Support Vector Machine (SVM) y Random Forest. Esta última es la que ha obtenido unos mejores resultados para el conjunto datos empleado en este trabajo.

Palabras clave:

- Ciberseguridad
- Aprendizaje automático
- Python
- Denegación de servicio
- Detección de ataques

Índice de contenidos

Índice de tablas

Índice de figuras

1 Introducción y objetivos

Las denegaciones de servicio (DoS) son una de las amenazas para la seguridad que más preocupan en la actualidad. Este tipo de amenaza afecta a la disponibilidad de los activos y hace que éstos no puedan dar servicio a sus usuarios o clientes legítimos con una calidad adecuada cuando se necesita.

En los últimos años, los adversarios han empleado diferentes tipos de patrones para materializar estas amenazas, desde el *malware* hasta la saturación de redes o servidores con tráfico o peticiones masivas. Este trabajo se centra en estos últimos, denominados habitualmente ataques volumétricos. Ninguna de estas técnicas suele ser especialmente sofisticada, pero si algo las distingue, es que suelen ser difíciles de detectar. ¿Cómo distinguir una situación de sobrecarga de una en la que es un adversario quien provoca la denegación de servicio? ¿Qué diferencia a las peticiones de usuarios legítimos de las de un atacante malicioso? ¿Cómo dejar pasar unas y bloquear las otras?

Los métodos tradicionales para la prevención de ataques DoS, como la configuración de *firewalls* o la limitación del ancho de banda por IP origen, por ejemplo, no resultan muy efectivos en muchos de los escenarios actuales. Los ataques más complejos, por ejemplo, los distribuidos, resultan además difíciles de detectar. Por este motivo, en la actualidad, se investiga en técnicas de detección basadas en aprendizaje automático, que puedan aprender a distinguir entre peticiones legítimas y peticiones maliciosas y, gracias a ello, realizar una detección temprana que permita a los equipos de seguridad y de respuesta ante incidentes a reaccionar de manera adecuada.

Los objetivos de este trabajo se pueden resumir en:

1. Identificar uno o varios conjuntos de datos abiertos que incluyan información etiquetada sobre el tráfico de red en entornos reales y que permitan proponer modelos de aprendizaje automático para la detección de ataques DoS a un servidor.

2. Construir y validar una solución que se base en un clasificador binario tráfico legítimo y tráfico malicioso que permita detectar de forma temprana ataques DoS.

3. Seleccionar diferentes técnicas de aprendizaje automático que puedan emplearse en esta solución.

4. Evaluar el rendimiento obtenido por las técnicas empleadas y comparar sus resultados, para poder así extraer conclusiones acerca de su idoneidad en el escenario estudiado.

Para conseguir estos objetivos se ha seguido la planificación mostrada en la figura 1.1, para interpretar dicha figura, la leyenda será la siguiente, siendo T las tareas y ST las subtareas:

T1: Búsqueda de investigaciones previas
T2: Lectura de investigaciones previas
T3: Estructuración del trabajo
T4: Búsqueda del conjunto de datos
T5: Comprensión del conjunto de datos
T6: Planteamiento esqueleto del código
T7: Declaración de funciones
T8: Código principal
T9: Reducción del conjunto de datos
T10: Tratamiento de datos
ST1: Estudio columnas poco relevantes
ST2: Supresión columnas poco relevantes
ST3: Codificación variables categóricas
ST4: Normalización de los datos
ST5: Estudio PCA
ST6: División de los datos
T11: Selección de los algoritmos
T12: Selección de métodos de medición
T13: Programación algoritmo aleatorio
T14: Programación Regresión Logística
T15: Programación KNN
T16: Programación SVM
T17: Programación Random Forest
T18: Programación métodos de medición, entrenamiento y experimentos con los modelos

T19: Introducción
T20: Estado del Arte
T21: Metodología
T22: Implementación
T23: Conclusiones

El resto del documento se estructura de la siguiente manera: En el capítulo 2 se presentan conceptos básicos acerca de los ataques de denegación de servicio, y se resume el estado de arte en técnicas que permiten prevenir y detectar este tipo de ataques. En el capítulo 3 se presenta la metodología que se ha seguido para la realización de este trabajo, tradicional en proyectos de aprendizaje automático. En el capítulo 4 se muestra la implementación de la solución construida y se resumen los resultados de los experimentos realizados para su validación y para la evaluación de las diferentes técnicas comparadas. Por último, el capítulo 5 resume las conclusiones más importantes de este trabajo y propone algunas líneas de trabajo futuro.

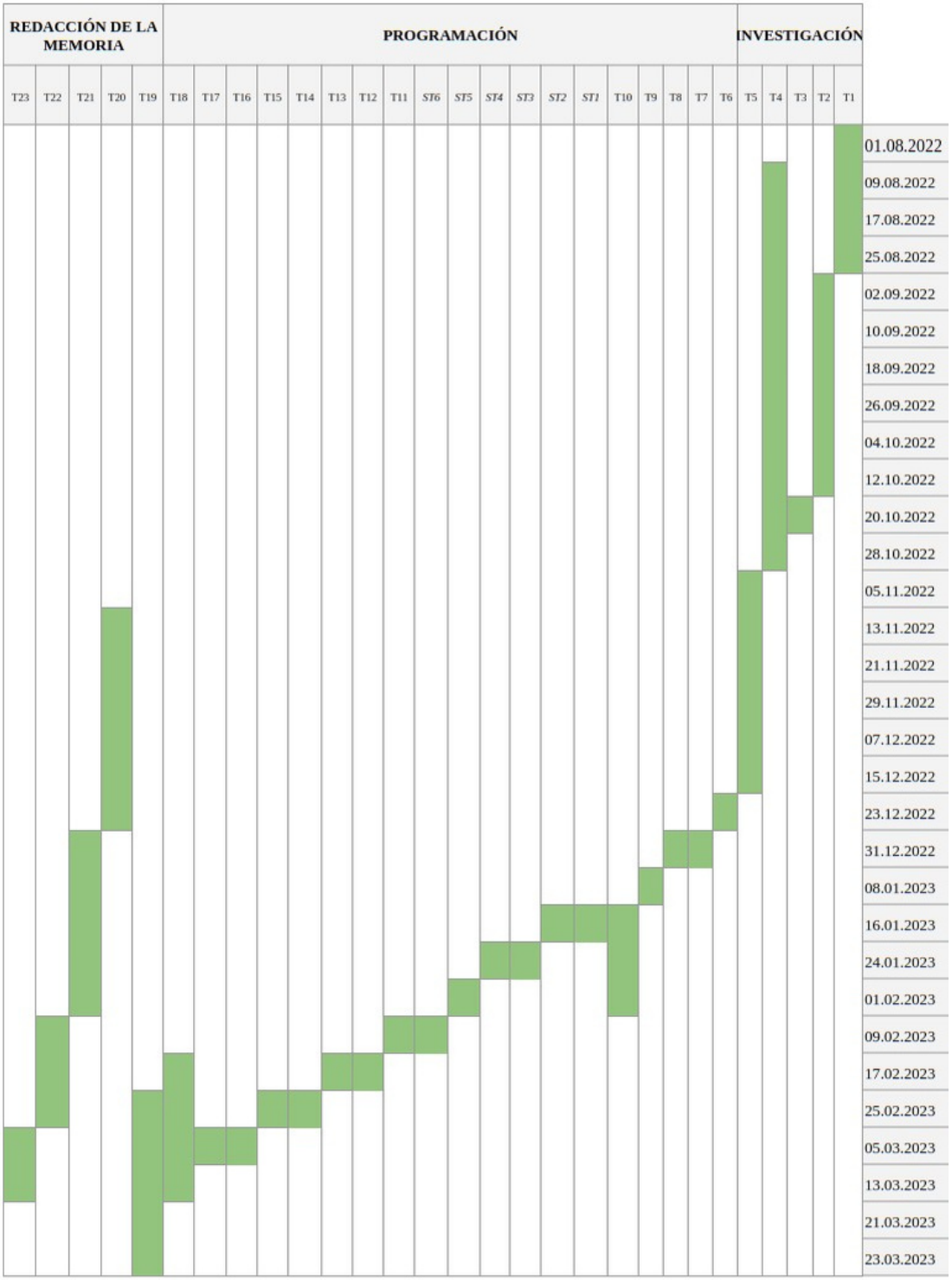

Figura 1.1: Diagrama de Gantt

2

Estado del arte

En la actualidad, la disponibilidad de los sistemas informáticos es esencial para garantizar el funcionamiento de la mayoría de las organizaciones y empresas. Del mismo modo, la dependencia de la tecnología ha llevado a un aumento en las amenazas a la disponibilidad. En este capítulo, profundizaremos en los distintos tipos de ataques DoS, los métodos clásicos utilizados para detectarlos y protegerse ante ellos. También exploraremos las últimas tendencias en el uso de aprendizaje automático para mejorar la detección y protección contra estos ataques.

2.1. Amenazas a la disponibilidad

Según el NIST SP 800-53 Rev. 5 [1], la disponibilidad es "la propiedad de un sistema o recurso para estar accesible y utilizarse en el momento en que se requiere por una entidad autorizada". Es un problema crucial hoy en día porque interrumpir este servicio puede tener impacto negativo en el flujo de trabajo y en la prestación de servicios.

Entre las amenazas y ataques actuales a la disponibilidad se encuentran:

1. Ransomware: *malware* que cifra los datos y archivos de una empresa y pide un rescate a cambio de la clave de descifrado.
2. Explotación de vulnerabilidades del sistema: mediante el uso de *exploits* y la inyección de código malicioso.

3. Bombas lógicas: *malware* que se activa al cumplirse una condición, como una hora específica. Esto lo hace difícil de detectar. Puede llevar a cabo diversas acciones, como borrado de archivos.

4. Ataques DoS (*Denial of Service*) y DDoS (*Distributed Denial of Service*): inundación de un sistema con peticiones para hacerlo inaccesible.

Los ataques de denegación de servicio (DoS) y denegación de servicio distribuido (DDoS) constituyen un problema y riesgo crítico en la seguridad de la información. Este tipo de ataque consiste en la sobrecarga o saturación de un servicio mediante el envío de un alto número de peticiones, lo que dificulta el acceso de los usuarios legítimos. Con el aumento de la dependencia de los servicios en línea y la complejidad creciente de las redes, detectar estos ataques se ha convertido en un desafío importante.

La diferencia entre estos dos tipos de ataques radica en la cantidad de sistemas utilizados para llevarlos a cabo. En el caso de ataques DoS el atacante hace uso de un único sistema, mientras que en ataques DDoS, en los que se utilizan múltiples dispositivos comprometidos, conocidos como "bots" o "zombis", para enviar un gran volumen de tráfico o solicitudes al objetivo. Los ataques DDoS son realizados por un gran grupo de dispositivos comprometidos controlados por un solo atacante, conocido como *botmaster*, estos ataques son conocidos como ataques por inundación. Por otra parte, el atacante puede lograr la colaboración de las máquinas víctima suplantando el servidor o sistema víctima, haciendo creer a dichas máquinas víctima que se les está solicitando una respuesta, este ataque es conocido como ataque reflejado. Los ataques DDoS son más difíciles de defender debido al gran volumen de tráfico que se origina desde múltiples direcciones y que puede ser difícil de rastrear.

Dentro de estos ataques existen distintas formas de lograr el objetivo de pérdida de la disponibilidad del sistema víctima.

2.1.1. Patrones basados en protocolos de la capa de infraestructura

Este tipo de ataque DoS explota debilidades en los protocolos de la capa de infraestructura de la red, como el protocolo TCP, UDP o ICMP.

Para saturar el servidor o sistema se hace uso de dos técnicas:

- Amplificación por tamaño de paquete: aumento del tamaño de los paquetes

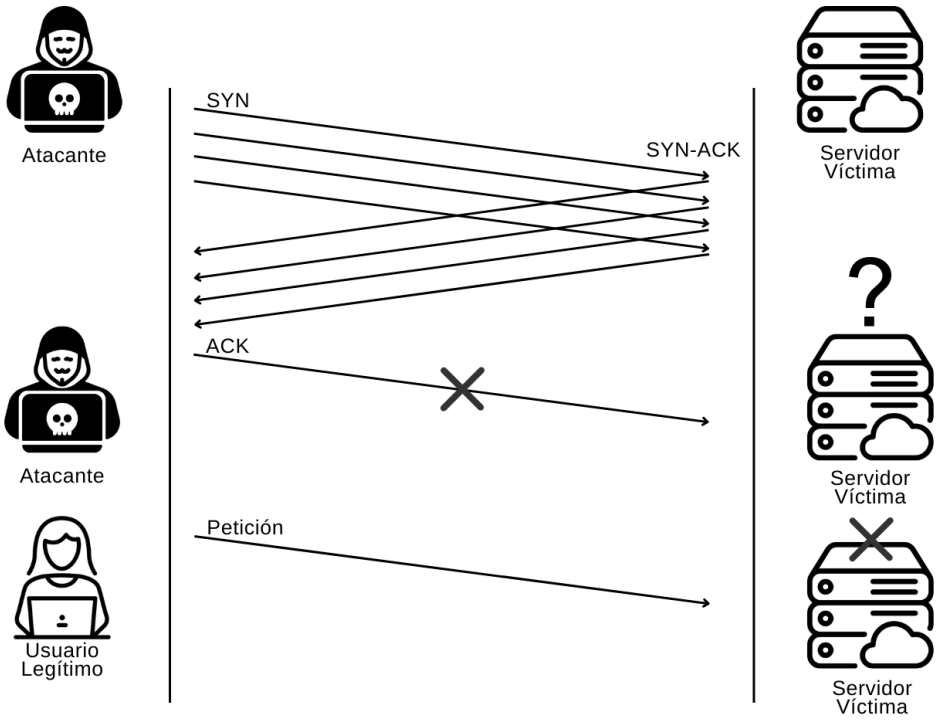

Figura 2.1: Representación de ataque SYN Flood

que se consigue que lleguen al servidor o sistema víctima, con el objetivo de aumentar el ancho de banda de red consumido.

- Amplificación por distribución: llevando a cabo el ataque desde distintos dispositivos y convirtiéndolo en un ataque DDoS.

Del mismo modo podemos encontrarnos con diferentes tipos de ataques en función del protocolo cuyas debilidades son explotadas.

Ataque SYN Flood [2]

Como podemos ver en la figura 2.1, este tipo de ataque explota el protocolo TCP y tiene como objetivo consumir todo el ancho de banda disponibles en un sistema o servidor. El atacante envía una gran cantidad de paquetes SYN al objetivo, lo que hace que el objetivo responda con un paquete SYN-ACK. Sin embargo, el atacante nunca envía el paquete ACK correspondiente, lo que hace que el objetivo espere una respuesta que nunca llega.

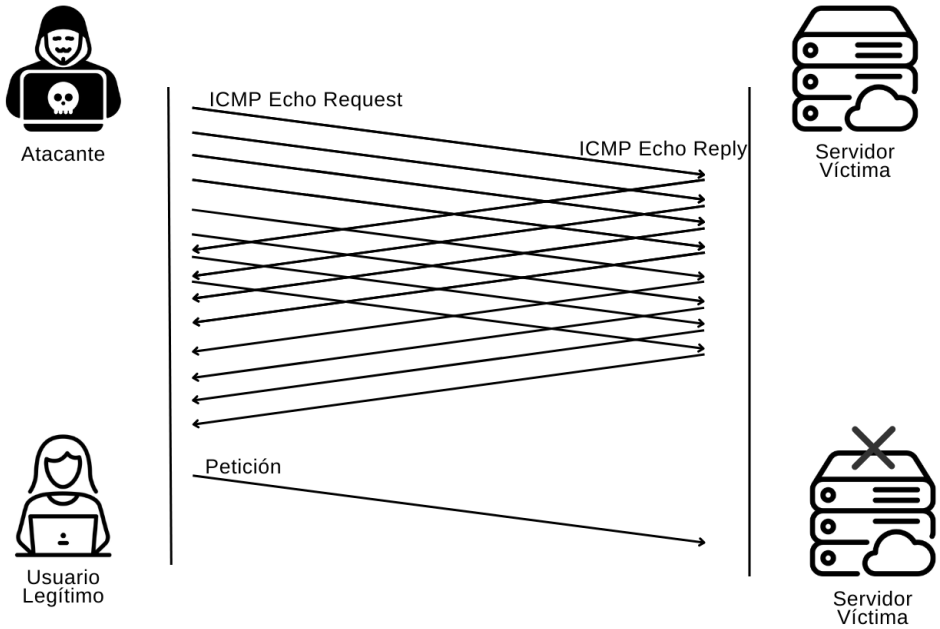

Figura 2.2: Representación de ataque Ping Flood

Ataque Ping Flood [2]

Como se puede observar en la figura 2.2, este ataque explota el protocolo ICMP y tiene como objetivo sobrecargar el sistema o servidor víctima con una gran cantidad de paquetes ICMP Echo Request, la víctima responde a este paquete con otro del tipo ICMP Echo Reply, lo que provoca que la cantidad de tráfico en la red aumente a gran velocidad y el servidor o sistema deje de proporcionar servicio.

Ataque UDP Flood [2]

Como se puede ver en la figura 2.3, este tipo de ataque explota el protocolo UDP y tiene como objetivo sobrecargar el sistema o servidor víctima con una gran cantidad de paquetes UDP. Como estos paquetes no requieren una respuesta por parte del servidor, este no puede descartarlos.

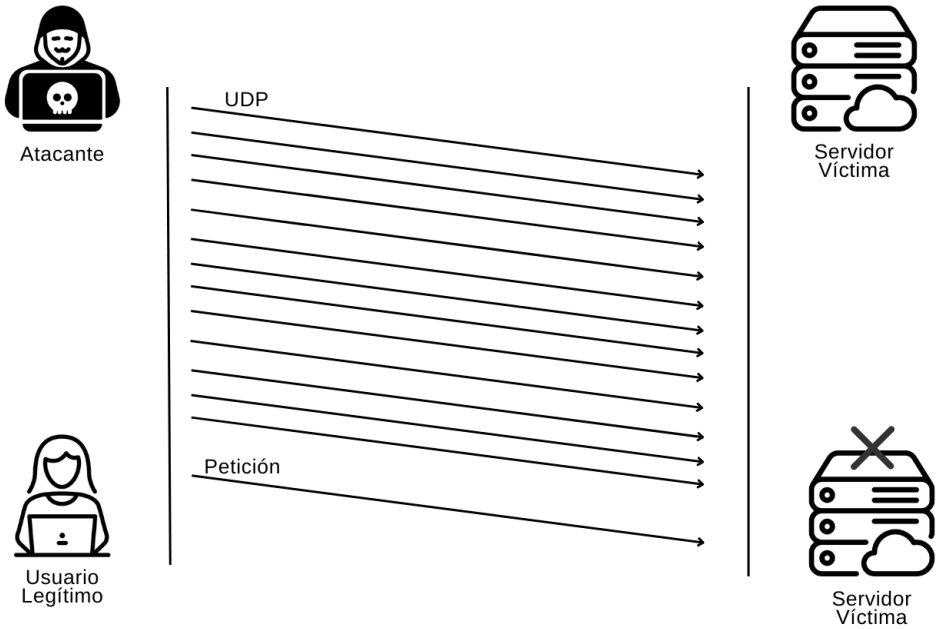

Figura 2.3: Representación de ataque UDP Flood

Ataque Smurf [3]

Como se indica en la figura 2.4, este ataque combina el ataque Ping Flood con la técnica de IP Spoofing, dicha técnica consiste en la falsificación de la IP del remitente en un paquete de red, el objetivo es engañar al servidor simulando ser una fuente confiable y legítima, esta técnica puede utilizarse también para llevar a cabo accesos no autorizados.

En este ataque, se envía una gran cantidad de paquetes ICMP Echo Request, suplantando a la máquina víctima, a diversos dispositivos de la red, provocando que estos envíen un paquete ICMP Echo Reply al servidor o sistema víctima y logrando un ataque DDoS reflejado.

Ataque de Nuke [4]

Como podemos ver en la figura 2.5, este ataque consiste en el envío de paquetes maliciosos o mal formados con el objetivo de consumir recursos del servidor o sistema víctima y saturarlo. Estos paquetes pueden contener grandes canti-

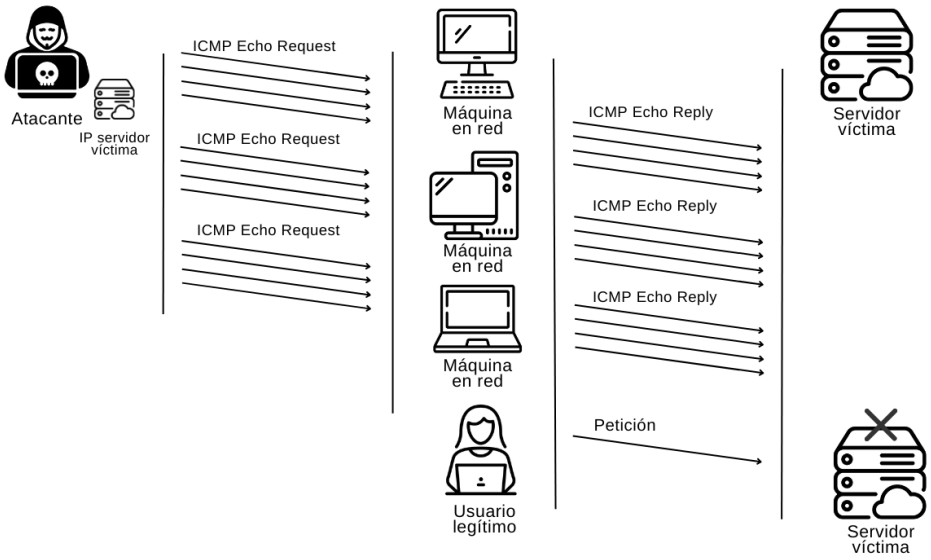

Figura 2.4: Representación de ataque Smurf

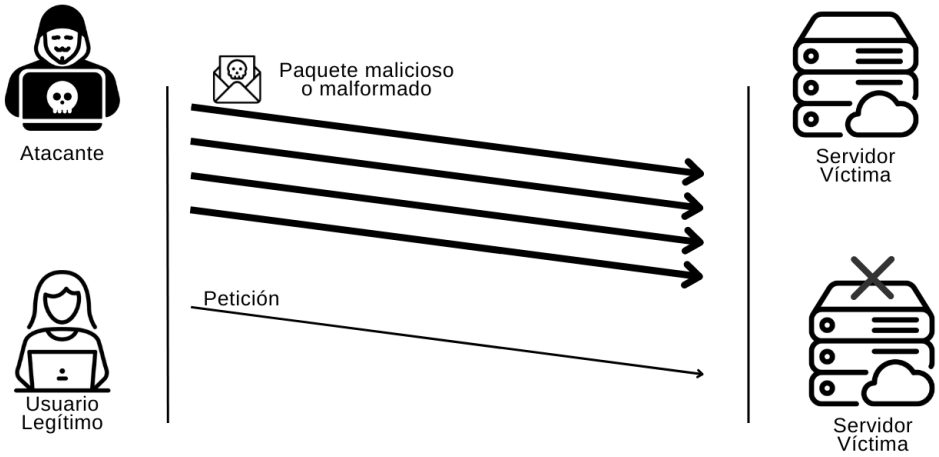

Figura 2.5: Representación de ataque Nuke

dades de datos mal formados o código malicioso diseñado para explotar alguna vulnerabilidad en el sistema víctima.

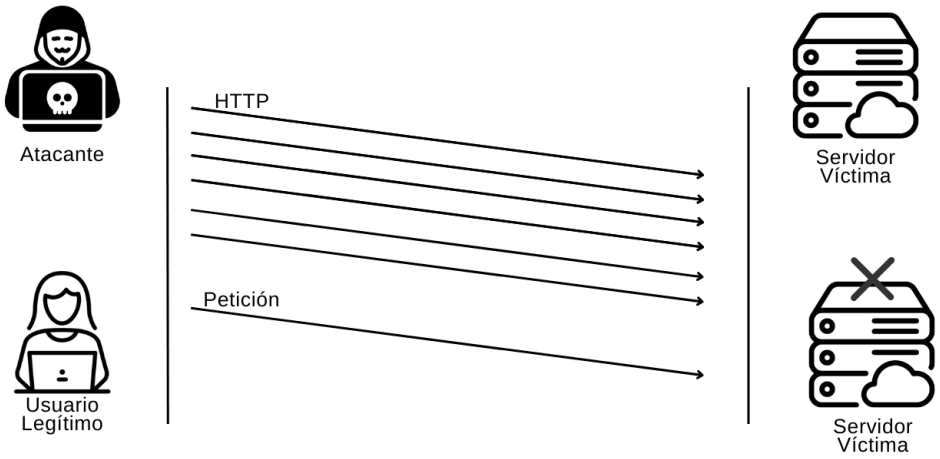

Figura 2.6: Representación de ataque por HTTP

2.1.2. Patrones basados en protocolos de la capa de aplicación

Este tipo de ataque DoS explota debilidades en los protocolos de la capa de aplicación y suelen ser más efectivos, ya que los protocolos de capa de aplicación están pensados para ser los que interactúan con el usuario final y son más manipulables de forma maliciosa. A pesar de esto, son menos comunes que los ataques DoS a protocolos de capa de infraestructura.

Se detallan a continuación algunos tipos de estos ataques.

Ataque de petición HTTP [2]

Como se observa en la figura 2.6, consiste en el envío de un número de peticiones HTTP, mayor al que puede gestionar, al servidor víctima con el objetivo de consumir sus recursos. Se hace uso de las peticiones HTTP GET y HTTP POST.

- **HTTP GET:** Con este tipo de petición se puede conseguir el ataque DoS amplificando por distribución.
- **HTTP POST:** Con este tipo de petición se obliga al servidor o sistema víctima a llevar a cabo tareas de alto cómputo, acelerando de esta manera

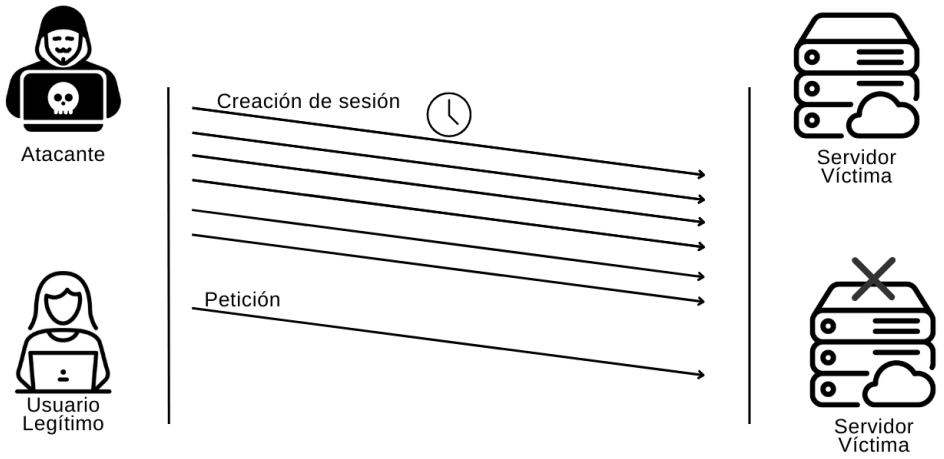

Figura 2.7: Representación de ataque de parálisis de sesión

el ataque DoS. Una posible aplicación de este ataque podría ser un ataque de sobrecarga de formularios, en el que se envía una gran cantidad de formularios maliciosos especialmente construidos para consumir los recursos del sistema.

Ataque de parálisis de sesión [5]

Como se indica en la figura 2.7, dicho ataque consiste en la creación de un gran número de sesiones que se mantengan abiertas al mismo tiempo y consuman recursos en el sistema víctima, causando la interrupción del servicio.

Ataque de buffer overflow [6]

Como se observa en la figura 2.8, este ataque consiste en el envío de una gran cantidad de datos al sistema, saturándolo y provocando la pérdida de disponibilidad.

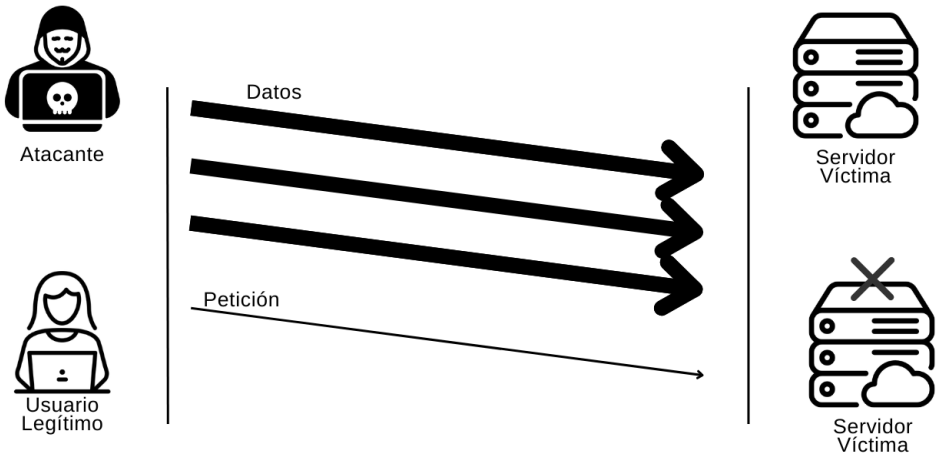

Figura 2.8: Representación de ataque de buffer overflow

2.2. Mecanismos de protección y detección anti-DoS clásicos

2.2.1. Mecanismos de protección anti-DoS clásicos

Aunque los patrones de ataque presentados en la sección anterior son ampliamente conocidos y no son especialmente sofisticados, la dificultad de protegerse ante un ataque DoS radica en la imposibilidad de distinguir, en la mayor parte de las ocasiones, entre tráfico o peticiones legítimas y maliciosas. Trabajos como el publicado por Yarochkin en 2015 [7] o el publicado por Chowdhury en 2018 [8], han propuesto clasificaciones de las soluciones propuestas hasta el momento, principalmente:

Filtros de dirección IP

Consistentes en el bloqueo de las solicitudes.

Límites de velocidad

Se limita la cantidad de peticiones por unidad de tiempo que puede realizar cada dirección IP.

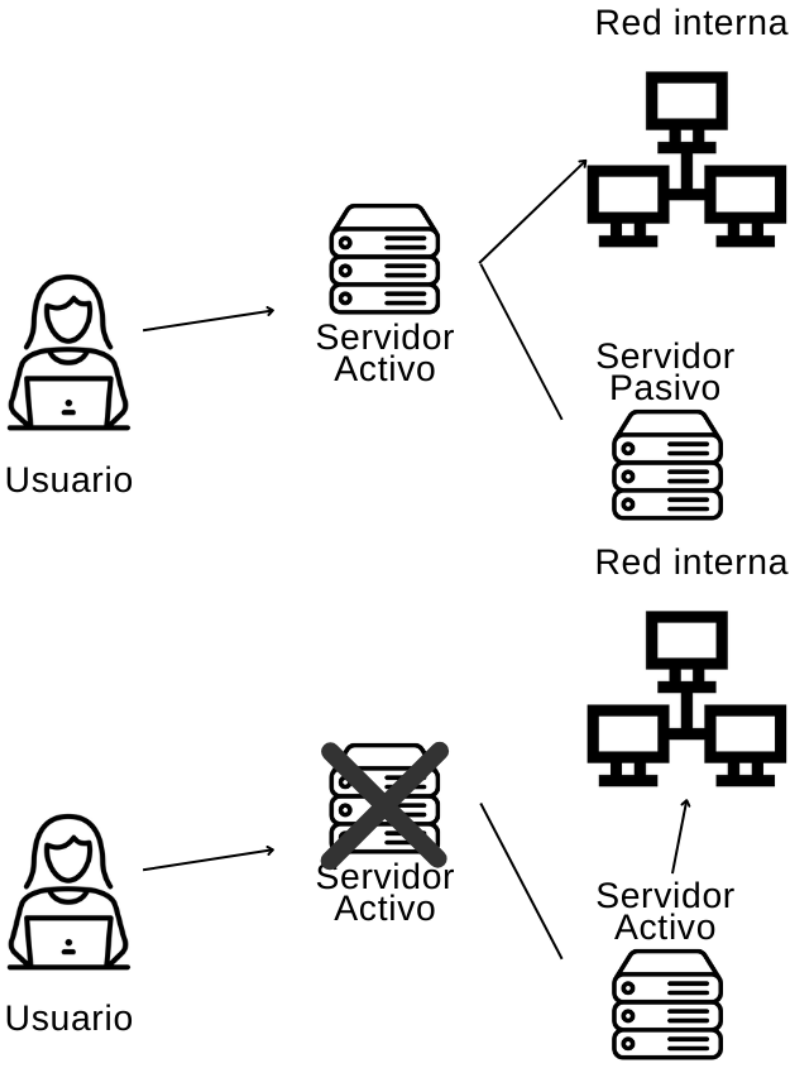

Figura 2.9: Representación de arquitectura Activo-Pasivo para garantizar alta disponibilidad

Balanceo de carga

Se distribuye la carga de solicitudes entre diferentes servidores para disminuir la posibilidad de sufrir un ataque DoS.

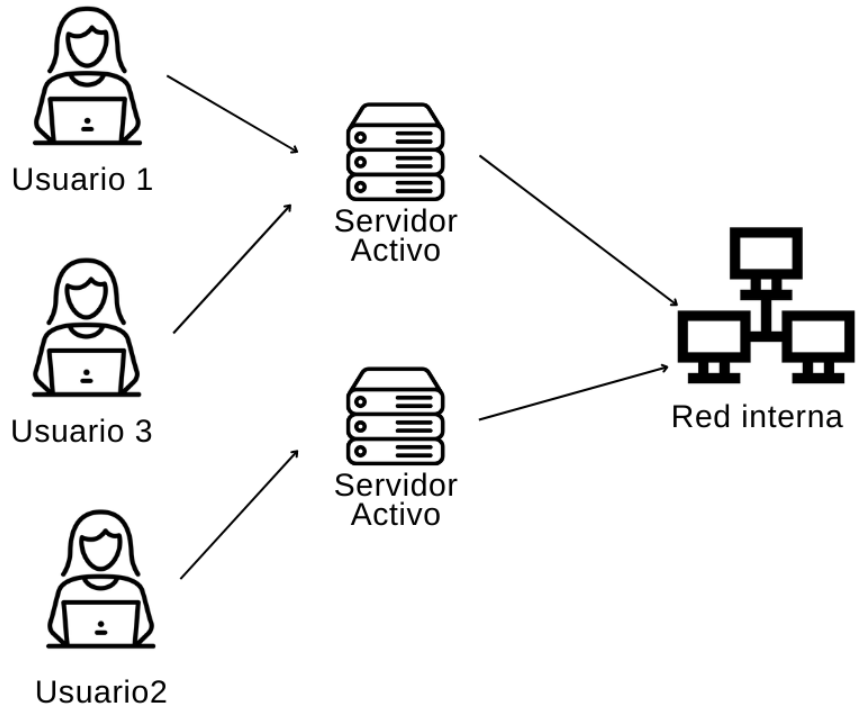

Figura 2.10: Representación de arquitectura Activo-Activo para garantizar alta disponibilidad

Arquitectura de alta disponibilidad

Este tipo de arquitectura consiste en la implementación de sistemas que estén activos sin interrupciones, a través de la redundancia y la tolerancia a fallos en componentes como servidores, dispositivos, alimentación eléctrica, etc.

La arquitectura de alta disponibilidad suele combinarse con mecanismos de monitorización y detección de incidentes en tiempo real.

Dentro de la arquitectura de alta disponibilidad, existen distintos tipos:

- Activo-Pasivo: Como se representa en la figura 2.9, el servidor activo proporciona servicio y el servidor pasivo queda en espera para tomar el control

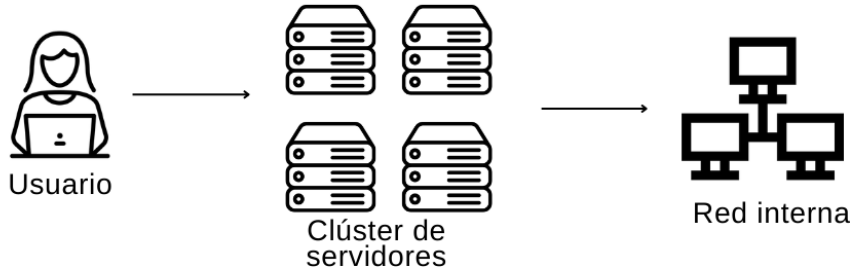

Figura 2.11: Representación de un clúster de alta disponibilidad

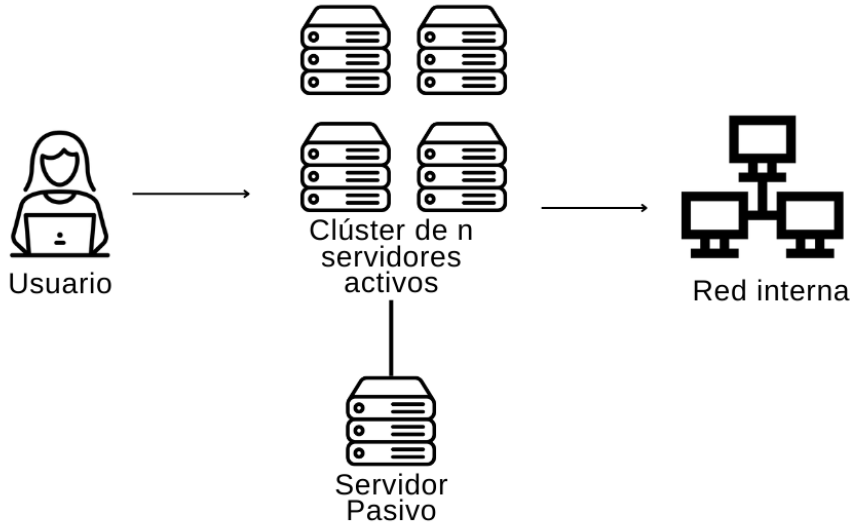

Figura 2.12: Representación de Arquitectura N + 1 para garantizar alta disponibilidad

en caso de incidencia.

- Activo-Activo: Como se indica en la figura 2.10, ambos servidores se mantienen activos trabajando en paralelo y repartiendo la carga de trabajo.

- Clúster de alta disponibilidad: Conjunto de máquinas que actúa como una única, existe un software de gestión del clúster que monitoriza y controla el rendimiento, como puede verse en la figura 2.11.

- Arquitectura N + 1: Como se puede ver en la figura 2.12, en este tipo de arquitectura se crean n servidores en estado activo, proporcionando servicio, y se mantiene un servidor adicional de respaldo (pasivo).

2.2.2. Mecanismos de detección clásicos

Hasta el momento, artículos como el publicado en 2003 por Fung [9], han propuesto diversos mecanismos de detección de ataques DoS, entre los que se encuentran los detallados a continuación.

Análisis de tráfico

Este método consiste en la monitorización de distintos aspectos del tráfico y relacionados con él. A continuación se detallan algunos de ellos.

- **Velocidad de transmisión:** Se monitoriza la velocidad de transmisión con el fin de detectar aumentos súbitos del tráfico que indiquen un ataque DoS.

- **Recursos:** Monitorización de recursos del sistema utilizados, como la RAM o la CPU, analizando y buscando patrones anómalos o gran consumo de los mismos.

- **Dirección IP:** Se monitoriza la dirección IP de origen de los paquetes para identificar patrones como un gran volumen de datos enviados por la misma dirección IP.

- **Duración de sesiones:** Análisis de la duración de las sesiones, identificando patrones anómalos como sesiones de larga duración o muy cortas. También puede analizarse la creación de un gran número de sesiones en un corto periodo de tiempo.

- **Intensidad de tráfico:** Con el fin de identificar un crecimiento súbito en la cantidad de paquetes recibidos en un periodo corto de tiempo, lo cual puede indicar un ataque DoS.

Detección de anomalías

En este método se compara un perfil de tráfico definido como normal con el tráfico de red en ese momento, buscando anomalías que puedan indicar un ataque DoS.

Uso de reglas

Consiste en la definición de reglas de tráfico según determinados patrones para detectar tráfico en elementos como IDS (Intrusion Detection System).

Uso de firmas

Búsqueda en bases de datos de firmas conocidas de ataques DoS.

2.2.3. Limitaciones de los mecanismos de protección y detección clásicos

Los sistemas clásicos de detección de ataques DoS tienen muchas limitaciones que reducen su efectividad para identificar y prevenir ataques de denegación de servicio. Estos sistemas dependen de firmas conocidas, tienen dificultades para identificar ataques distribuidos, producen altos niveles de falsos positivos y requieren una gran cantidad de recursos computacionales. Además, estos sistemas solo pueden identificar ataques después de que estos hayan comenzado, lo que los hace incapaces de detener completamente los ataques DoS antes de que sucedan.

Es por ello que se han desarrollado nuevas tendencias en relación a los sistemas de protección y detección de este tipo de ataques que abordan y mejoran los resultados en relación a estas limitaciones, para proporcionar una protección y defensa más efectivas contra dichos ataques.

2.3. Nuevas tendencias en detección de ataques DoS

En los últimos años, se ha estudiado ampliamente la detección de ataques DoS mediante el uso de técnicas de aprendizaje automático. Este tipo de técnicas, se utilizan para analizar grandes cantidades de datos de tráfico de red y detectar

patrones anómalos que indican un posible ataque. Esto permite a los administradores de sistemas responder rápidamente a los ataques y mitigar sus efectos. Sin embargo, también existen desafíos en el uso de estas técnicas para la detección de DoS, como la necesidad de un gran volumen de datos de entrenamiento y la posibilidad de falsos positivos. A medida que los atacantes se vuelven más sofisticados, es importante que las técnicas de detección de DoS también evolucionen para mantenerse un paso adelante.

Actualmente, la investigación en la detección de ataques de denegación de servicio mediante técnicas de aprendizaje automático está enfocada en mejorar la precisión y eficacia de los sistemas de detección, así como en desarrollar nuevos enfoques y algoritmos.

En el artículo publicado por Parvinder Singh en 2020 [10], los autores presentan un sistema de detección de ataques de denegación de servicio distribuido (DDoS) utilizando algoritmos de aprendizaje automático. El sistema propuesto utiliza una combinación de técnicas de aprendizaje automático, incluyendo Random Forest, SVM y Naive Bayes para analizar datos de tráfico de red y detectar patrones anómalos que indican un posible ataque DDoS. Los autores realizaron pruebas en una plataforma experimental utilizando datos de tráfico real y compararon el rendimiento de su sistema con un enfoque basado en reglas y otro basado en aprendizaje automático. Los resultados mostraron que el sistema propuesto logró detectar con éxito los ataques DDoS con una tasa de falsas alarmas baja y una tasa de detección alta.

Por otra parte, en el artículo publicado por S. Malliga [11], los autores realizan una revisión de técnicas de aprendizaje profundo utilizadas para detectar ataques de denegación de servicio (DoS) y ataques de denegación de servicio distribuido (DDoS). En particular, se centran en las técnicas utilizadas para detectar estos ataques en redes de internet de las cosas (IoT). Los autores discuten las ventajas y desventajas de las diferentes técnicas de aprendizaje automático utilizadas, incluyendo redes neuronales, sistemas de aprendizaje automático basados en reglas y algoritmos de aprendizaje supervisado y no supervisado.

El artículo también proporciona un análisis detallado de las diferentes métricas utilizadas para evaluar el rendimiento de las técnicas de detección de ataques DDoS y una discusión de los desafíos y limitaciones actuales en la detección de ataques DDoS utilizando técnicas de aprendizaje automático. Los autores concluyen que, a pesar de los avances en la detección de ataques DDoS utilizando técnicas de aprendizaje automático, todavía hay desafíos importantes que deben ser abordados y se necesita una mayor investigación en este campo.

Otro artículo que estudia el impacto de los ataques DDoS en las redes IoT es el publicado por Dutta Sai Eswari en 2021 [12], en el que se presentan las diferentes técnicas de aprendizaje automático utilizadas para detectar ataques DDoS, incluyendo redes neuronales, sistemas de aprendizaje automático basados en reglas, algoritmos de aprendizaje supervisado y no supervisado. Los autores proporcionan un análisis detallado de las ventajas y desventajas de cada técnica y discuten cómo se utilizan en la detección de ataques DDoS en las redes IoT. También proporcionan un análisis detallado de las ventajas y desventajas de cada técnica y discuten cómo se utilizan en la detección de ataques DDoS en las redes IoT.

Debido a su capacidad para analizar grandes cantidades de datos en tiempo real y reconocer patrones inusuales de tráfico de red que apuntan a la presencia de un ataque DoS, las técnicas basadas en aprendizaje automático son más efectivas para identificar ataques de denegación de servicio (DoS).) ataques. Además, estos métodos pueden adaptarse y desarrollarse para detectar nuevas formas de ataques que los sistemas convencionales de detección de intrusos no pueden reconocer.

Según un estudio de M. A. Butun en 2018 [13], en comparación con los sistemas convencionales basados en reglas y firmas, los sistemas de detección de intrusos basados en aprendizaje automático, han demostrado un alto rendimiento en la detección de ataques DoS en 2018. Dado que los ataques DDoS son más complejos y variables que otros tipos DoS, los autores enfatizan que el aprendizaje automático puede ser útil para identificar y mitigar estos ataques.

Otro estudio de D.C. Nguyen en 2020 [14] muestra cómo se puede usar el aprendizaje automático para detectar patrones de tráfico inusuales en tiempo real y notificar a los administradores de red para que tomen las medidas preventivas necesarias. Los autores señalan que los sistemas de detección basados en reglas y firmas son menos efectivos cuando se trata de detectar ataques DoS que utilizan tácticas sigilosas, como el enmascaramiento de direcciones IP.

Como se ha podido observar, las técnicas de aprendizaje automático son una herramienta prometedora para la detección temprana de este tipo de ataques. Por lo tanto, en este trabajo se utilizarán este tipo de técnicas para la detección de ataques DoS. Para lograr este objetivo, se realizará un estudio exhaustivo y comparativo de diferentes técnicas, con el fin de determinar cuál de ellas es la más eficaz en la resolución del problema descrito. Esto incluirá la evaluación de distintos algoritmos de aprendizaje automático para identificar patrones de comportamiento malicioso y predecir la ocurrencia de ataques DoS.

3
Metodología

En el presente capítulo se profundiza en el problema para la disponibilidad identificado. También se explicará la metodología seguida para la resolución del mismo, se comienza explicando el conjunto de datos que hemos escogido y el porqué de esa elección. Después, se explicará cómo se realizó el tratamiento y análisis de los datos para garantizar que estén limpios y preparados para su uso en modelos de estadísticos o de aprendizaje automático. Además, se presentarán los modelos que se han seleccionado y cómo se han entrenado para obtener los resultados deseados. Finalmente, se discutirán los métodos de evaluación utilizados para medir la eficacia de los modelos y los resultados obtenidos.

3.1. Fase I: Identificación del problema

Tal y como se ha detallado previamente, los ataques DoS son una de las principales amenazas a la disponibilidad de los servicios de una empresa u organización. Es por ello que resulta de vital importancia detectarlos y detenerlos antes de que cumplan su objetivo (la pérdida de la disponibilidad), sobre todo en el escenario actual.

El problema que aborda este trabajo es el de la detección de ataques DoS utilizando técnicas de aprendizaje automático. La detección de estos ataques es un desafío complejo debido a su diversidad y complejidad, lo que hace que sea difícil distinguir entre tráfico legítimo y malicioso. Por lo tanto, el objetivo de

este trabajo es comparar diferentes técnicas para la identificación de tráfico de red malicioso como ataque de DoS.

Para desarrollar un sistema eficaz de detección de ataques DoS mediante el uso de técnicas de aprendizaje automático, es necesario cumplir varios requisitos. En primer lugar, el sistema debe ser capaz de procesar grandes volúmenes de datos en tiempo real para detectar ataques de forma temprana. En segundo lugar, el sistema debe tener la capacidad de diferenciar el tráfico legítimo del malicioso, lo que requiere el uso de algoritmos adecuados. Además, el sistema debe poder adaptarse a diferentes entornos de red y a diversos tipos de ataques. Por último, el sistema debe ser fácil de utilizar y mantener para garantizar su efectividad a largo plazo.

3.2. Fase II: Obtención, estudio y comprensión de los datos

Para llevar a cabo la comparativa de técnicas de aprendizaje automático, se utiliza un conjunto de datos representativo de tráfico de red que incluye diferentes tipos de ataques DoS, así como tráfico legítimo.

El conjunto de datos utilizado para este trabajo ha consistido en una extracción de datos de los siguientes conjunto de datos: CSE-CIC-IDS2018-AWS, CICIDS2017, CIC DoS conjunto de datos(2016). Dicho proceso de extracción se encuentra detallado en el paper "Machine Learning DDoS Detection Using Stochastic Gradient Boosting"[15].

Dichos datos consisten en un conjunto de tráfico de red capturado por un IDS, el tráfico se clasifica como *Benign* (Legítimo) o *ddos*. Aspecto importante para hacer uso de los algoritmos de aprendizaje automático de clasificación que explicaremos más adelante.

El conjunto de datos escogido fue creado extrayendo el de ataques DDoS así como tráfico normal o legítimo de tres conjuntos de datos procedentes de estudios realizados durante distintos años, los cuales se explicarán a continuación. El proceso de extracción puede consultarse en el paper "Machine Learning DDoS Detection Using Stochastic Gradient Boosting" [16].

3.2.1. CSE-CIC-IDS2018-AWS

Se trata de un conjunto de datos de tráfico de red capturado en un entorno de nube de *Amazon Web Services* (AWS), que contiene una variedad de ataques cibernéticos y tráfico normal de red. Fue creado por el Centro de Seguridad Cibernética e Investigación de la Información de Canadá (CSE) en colaboración con la Universidad de *New Brunswick* (UNB) [17].

El tráfico fue capturado durante cinco días en el año 2018 y contiene un total de 16.7 millones de registros. Los ataques empleados durante la captura de tráfico incluyen: denegación de servicio (DoS), inyección SQL, escaneo de puertos, fuerza bruta, *phishing*, manipulación de paquetes y secuestro de sesiones, entre otros. Por otra parte, los registros de tráfico normal de red incluyen navegación web, correo electrónico y transferencia de archivos.

Los registros contienen información sobre las características de los paquetes de red capturados, como el tamaño de los paquetes, el protocolo utilizado, el número de puerto, el tipo de servicio y la duración de la comunicación.

3.2.2. CICIDS2017

El conjunto de datos CICIDS2017, presentado en el artículo publicado por M. Sharafaldin en 2018 [18], al igual que el anterior, contiene tanto datos de tráfico malicioso (ataques cibernéticos) como de tráfico normal. Fue creado y recopilado por el *Canadian Institute for Cybersecurity* (CIC).

Los datos fueron capturados durante un período de varios meses del año 2017 y contiene más de 2.8 millones de registros de tráfico de red capturado. Para la recogida se utilizaron técnicas de captura de paquetes y registro de eventos para capturar tanto tráfico normal como tráfico malicioso.

Se llevaron a cabo siete tipos de ataques para construir el tráfico malicioso: *Fuzzers*, análisis de Shell, *Backdoors*, DoS, *Exploits*, *Worms* y reconocimiento de red (escaneo de puertos, análisis de configuración de servicios de red, búsqueda de vulnerabilidades, etc.). Por otra parte, el tráfico legítimo contiene actividades como la navegación web, la descarga y carga de archivos, la comunicación de correo electrónico, la transmisión de vídeo y audio, entre otras actividades propias de una red empresarial.

Los registros más relevantes incluyen información sobre los protocolos de red,

los tamaños de paquetes, los números de puerto, los servicios de red y las direcciones IP de origen y destino.

3.2.3. CIC DoS (2016)

Este último conjunto de datos fue también creado y recogido por el CIC, siendo presentado en el paper publicado por H. Hadian en 2017 "Detecting HTTP-based Application Layer DoS attacks on Web Servers in the presence of sampling" [19]. Para recoger dichos datos, se simuló un ataque DDoS en una red en la que también se capturó tráfico normal.

La recopilación de los datos se realizó durante un período de 72 horas en un entorno controlado de laboratorio en la Universidad de New Brunswick, capturándose más de 12 millones de paquetes de red.

El conjunto de datos contiene un total de 15 tipos diferentes de ataques DDoS, que incluyen ataque SYN *flood*, UDP *flood*, HTTP con petición GET y POST, HTTP *slow-rate*, TCP, consulta DNS, consulta LDAP, consulta SNMP, consulta NTP, consulta SSDP, Chargen por amplificación, consulta NetBIOS, UDP por amplificación, SNMP por amplificación. Los ataques fueron realizados por una variedad de herramientas y programas maliciosos, como *Low Orbit Ion Cannon* (LOIC), *HULK*, *RUDY*, entre otros. Por otra parte, el tráfico normal consiste en actividades típicas de una red en funcionamiento normal, como la navegación web, la descarga y carga de archivos, la comunicación de correo electrónico y otros tipos de tráfico comunes.

Entre los registros proporcionados encontramos algunos relevantes como las direcciones IP de origen y destino, los números de puerto, los protocolos utilizados y los tiempos de llegada y otros relacionados con estadísticas de red como el número de paquetes y bytes recibidos, útiles para la identificación de los ataques.

3.3. Fase III: Análisis y preparación de los datos

El conjunto de datos escogido cuenta con 84 columnas, las cuales son características del tráfico de red enviado en cada paquete.

Para las pruebas iniciales se ha reducido el conjunto de datos a 1000 líneas extraídas aleatoriamente del original.

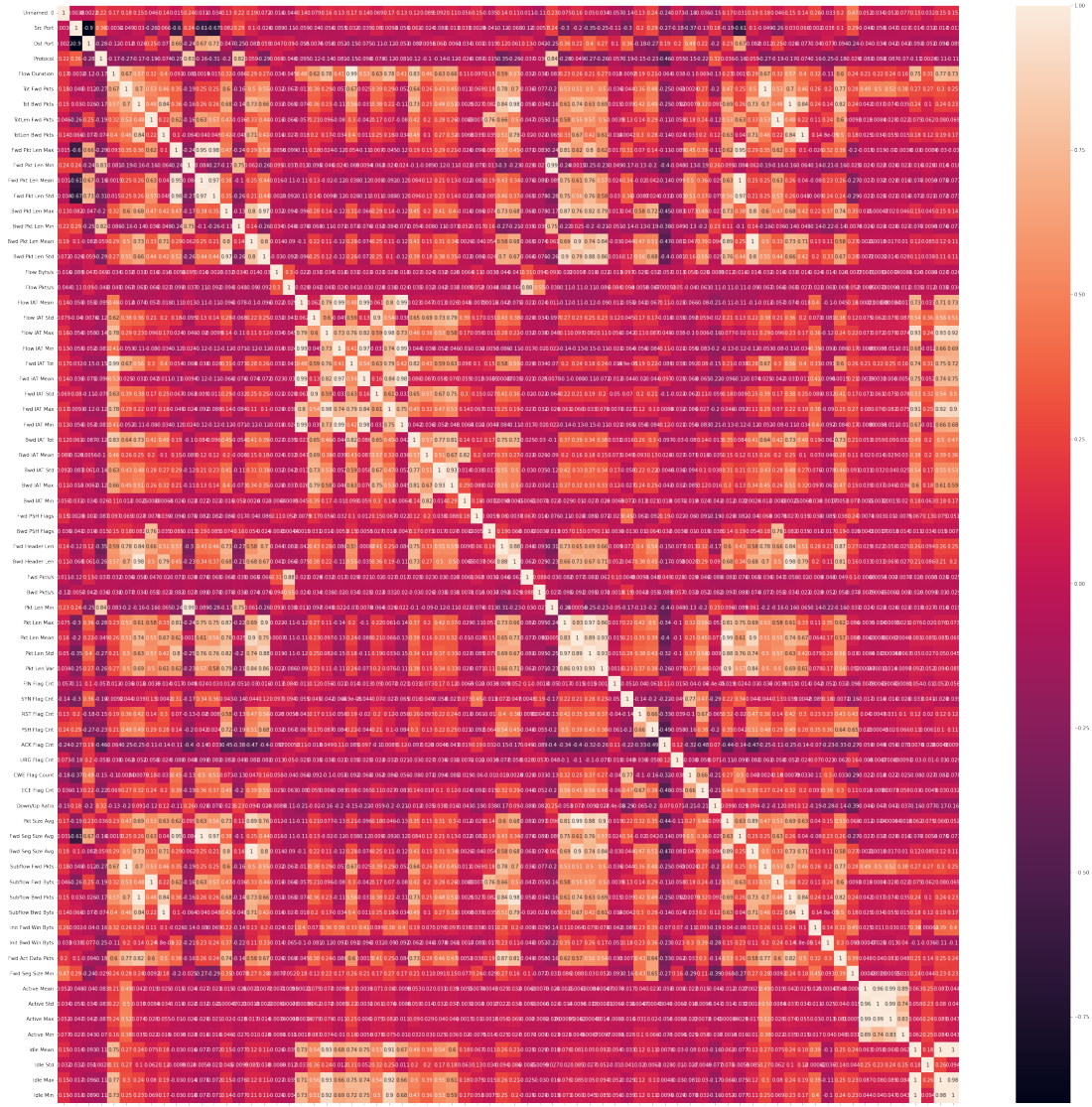

Figura 3.1: Matriz de correlación del conjunto de datos empleado

Para llevar a cabo el análisis y la preparación de los datos se han llevado a cabo los pasos detallados a continuación.

3.3.1. Limpiar columnas poco relevantes

El primer paso del tratamiento será eliminar la primera columna, que se corresponde al índice de cada paquete dentro de cada conjunto de datos, así como las columnas de *Flow ID* y *Timestamp*, para evitar la clasificación directa de los algoritmos con estas características. Por otra parte, se eliminan las columnas que

no contienen ningún valor y se rellenan los valores vacíos de aquellas columnas que contienen una mayor parte de valores presentes. Por último, se calcula la matriz de correlación con el fin de eliminar aquellas columnas con una correlación mayor a 0.80, ya que con ello se consigue reducir la cantidad de características sin perder información relevantes para el funcionamiento de los algoritmos.

Como se puede ver en la imagen 3.1, las columnas con color más claro y más oscuro (extremos), tienen una mayor correlación, lo que nos indica que podemos eliminar una de ellas. Gracias a esto pasamos a tener 38 columnas en lugar de 84, eliminándose las siguientes: *Src IP*, *Src Port*, *Dst IP*, *Protocol*, *Flow Duration*, *Tot Fwd Pkts*, *Tot Bwd Pkts*, *Fwd Pkt Len Max*, *Fwd Pkt Len Min*, *Bwd Pkt Len Max*, *Flow Byts/s*, *Flow Pkts/s*, *Flow IAT Mean*, *Flow IAT Std*, *Flow IAT Max*, *Fwd IAT Std*, *Bwd IAT Mean*, *Bwd IAT Std*, *Fwd PSH Flags*, *Bwd PSH Flags*, *Bwd Pkts/s*, *Pkt Len Var*, *FIN Flag Cnt*, *SYN Flag Cnt*, *RST Flag Cnt*, *PSH Flag Cnt*, *ACK Flag Cnt*, *URG Flag Cnt*, *CWE Flag Count*, *ECE Flag Cnt*, *Down/Up Ratio*, *Init Fwd Win Byts*, *Init Bwd Win Byts*, *Fwd Seg Size Min*, *Active Mean*, *Active Std*, *Idle Std*, *Label*.

Por otra parte, se mantienen las siguientes columnas o características: *Src IP*, *Src Port*, *Dst IP*, *Protocol*, *Flow Duration*, *Tot Fwd Pkts*, *Tot Bwd Pkts*, *Fwd Pkt Len Max*, *Fwd Pkt Len Min*, *Bwd Pkt Len Max*, *Flow Byts/s*, *Flow Pkts/s*, *Flow IAT Mean*, *Flow IAT Std*, *Flow IAT Max*, *Fwd IAT Std*, *Bwd IAT Mean*, *Bwd IAT Std*, *Fwd PSH Flags*, *Bwd PSH Flags*, *Bwd Pkts/s*, *Pkt Len Var*, *FIN Flag Cnt*, *SYN Flag Cnt*, *RST Flag Cnt*, *PSH Flag Cnt*, *ACK Flag Cnt*, *URG Flag Cnt*, *CWE Flag Count*, *ECE Flag Cnt*, *Down/Up Ratio*, *Init Fwd Win Byts*, *Init Bwd Win Byts*, *Fwd Seg Size Min*, *Active Mean*, *Active Std*, *Idle Std*, *Label*.

Las características que se mantienen están relacionadas con la información de flujo (*flow*) y paquetes (*packets*) de la red, mientras que las características que se eliminaron están relacionadas con la información de las banderas (*flags*) de los paquetes. En general, las características eliminadas están más enfocadas en el análisis de patrones de tráfico y la detección de ataques específicos, mientras que las características que se mantienen están más relacionadas con la información general de los flujos de red.

3.3.2. Codificar variables categóricas

Las variables categóricas son variables que toman valores discretos y no continuos, como colores, tamaños, etiquetas, categorías, etc. Los algoritmos de aprendizaje automático a menudo se basan en cálculos matemáticos y estadísticos, que requieren que los datos de entrada sean numéricos y continuos. Por lo tanto, es

necesario convertir las variables categóricas en variables numéricas antes de aplicar los algoritmos. En nuestro caso, se convertirán a tipo *float*.

En el conjunto de datos seleccionado, la mayoría de columnas son del tipo *int* o *float*, sin embargo, existen columnas del tipo *object* como *Flow ID*, *Src IP*, *Dst IP*, *Timestamp* y *Label*. Dichas columnas se convertirán a tipo *float* asignando a cada elemento un número.

3.3.3. Tratar valores extremos

Los valores extremos, también conocidos como valores atípicos o *outliers*, pueden tener un impacto significativo en el rendimiento de los algoritmos de aprendizaje automático, especialmente en aquellos que dependen de la media y la varianza de los datos.

Los valores extremos pueden sesgar la media y la varianza de los datos, lo que puede afectar negativamente el rendimiento de los algoritmos. Por ejemplo, si un conjunto de datos contiene valores extremos muy grandes, la media podría ser significativamente mayor que el valor medio de la mayoría de los datos. Esto podría hacer que un algoritmo de IA que depende de la media (como la regresión lineal) sea menos preciso.

Para evitar esto, una técnica común es reemplazar los valores extremos por la media o la mediana del conjunto de datos. De esta manera, se elimina el impacto de los valores extremos en la media y la varianza, lo que puede ayudar a mejorar el rendimiento de los algoritmos. En este caso, sustituiremos estos valores por la media.

En el caso del conjunto de datos seleccionado, se reemplazan los valores infinito y menos infinito por la media de la columna.

3.3.4. Normalizar datos

La normalización de datos es un proceso que consisten en el escalado de datos a un rango común para estandarizar los datos con el fin de que tengan una distribución de probabilidad común. Esto es importante, ya que facilita el aprendizaje de los algoritmos, ya que al normalizar los datos, se puede mejorar la estabilidad y la eficiencia de determinados algoritmos más sensibles a las escalas de datos, ayudando a reducir la variabilidad y mejorar la convergencia. Por otra parte, puede mejorar la precisión de algunos algoritmos de clasificación que se basan en

cálculos de distancia entre puntos de datos y ayudar a asegurar que las distancias se calculen de manera precisa y justa.

Para llevar a cabo este proceso, se identifica el valor mínimo y el máximo de cada columna, asignándose el primero a 0 y el segundo a 1. Posteriormente, se aplica la siguiente fórmula al valor de cada dato de cada columna:

$$\frac{valor - valor_{min}}{valor_{max} - valor_{min}}$$

3.3.5. Análisis de componentes principales (PCA)

El Análisis de Componentes Principales (PCA) es una técnica de reducción de dimensionalidad que se utiliza habitualmente en el tratamiento de datos antes de entrenar un algoritmo de aprendizaje automático como los que se utilizarán más adelante. PCA reduce la complejidad de los datos al transformar las variables originales en un conjunto más pequeño de variables no correlacionadas conocidas como componentes principales, al reducirse el número de variables, también se reduce la complejidad del modelo. Del mismo modo, se eliminan las variables que representen características redundantes, ya que se seleccionan solamente los componentes más informativos. Además de la reducción de la complejidad ya mencionada, PCA tiene como ventajas la mejora en la interpretación y en la precisión del modelo.

Para llevar a cabo el PCA del conjunto de datos descrito es necesario estandarizar los datos y definir un umbral para la varianza, el cual será 0.9 en este estudio, para evitar perder demasiada información. A continuación se calcula la proporción acumulada de varianza explicada por cada componente principal, lo que ayuda en el cálculo de componentes principales. Por último, se transforman los datos en el nuevo espacio de características generado por las componentes principales.

3.4. Fase IV: Modelado

Para resolver el problema van a probarse tres *(si hago 4 se cambia esto)* modelos diferentes: Regresión Logística, KNN, Subflow Vector Machine y Random Forest. Uno de los objetivos de este trabajo es comparar dentro de estos enfoques (estadística tradicional y aprendizaje automático) el rendimiento y la eficacia de dichos algoritmos para llegar a una conclusión sobre el más óptimo. Como base para comparar, se utilizará un algoritmo de clasificación aleatoria.

3.4.1. Clasificación aleatoria

Es importante usar un clasificador aleatorio como una línea base (*baseline*) con la que comparar los resultados de otros algoritmos para fijar una referencia que nos permita evaluar y optimizar los resultados de otros algoritmos. La metodología a seguir consistirá en la decisión aleatoria de la clase de cada paquete.

3.4.2. Regresión Logística

La regresión logística es un algoritmo de aprendizaje automático que se utiliza para resolver problemas de clasificación binaria. En este algoritmo, el objetivo es predecir si una muestra pertenece a una clase u otra en función de sus características. La salida representa la probabilidad de que la muestra pertenezca a una de las dos clases.

Este algoritmo puede ser beneficioso para la detección de ataques DDoS, ya que puede manejar conjuntos de datos grandes y complejos, lo que lo hace adecuado para el análisis de ataques DDoS. Además, no requiere muchos recursos computacionales, por lo que puede manejar más fácilmente estos grandes volúmenes de datos que otras técnicas.

Por otra parte, la regresión logística permite la selección de características, lo que significa que solo se utilizan las características más relevantes para la clasificación, reduciendo la complejidad del modelo y mejora su precisión. Además de ser capaz de identificar patrones en los datos y aprender de ellos.

3.4.3. *K-Nearest Neighbors* (KNN)

K-NN es un algoritmo de aprendizaje automático supervisado utilizado para predecir la clase de una muestra en función de las clases de los n vecinos más cercanos pertenecientes al conjunto de datos etiquetados, para ello calcula la distancia entre la muestra y cada elemento del conjunto de datos etiquetado, seleccionando los n vecinos más cercanos y determinando la clase más común entre ellos.

Dicho algoritmo puede ser útil de aplicar en el problema descrito en relación a los ataques DDoS, ya que si un conjunto de características se correlaciona con un ataque DDoS, el modelo podría identificar una nueva muestra sospechosa como DDoS de forma eficiente. Además, k-NN puede manejar grandes conjuntos de datos, lo que lo hace adecuado para el análisis de tráfico en línea en tiempo real.

Una limitación de K-NN puede ser el ser sensible a valores atípicos o al ruido, lo que puede perjudicar al rendimiento del modelo.

3.4.4. *Support Vector Machine* (SVM)

SVM es un algoritmo de aprendizaje automático supervisado utilizado para la resolución de problemas de clasificación y regresión cuyo objetivo es separar los datos en dos clases, encontrando el plano o recta que mejor separe dichos datos en las clases definidas, dicha recta o plano se denomina hiperplano.

Esto puede resultar beneficioso para la detección de ataques DDoS, ya que se puede utilizar para identificar los patrones que diferencian el tráfico normal del tráfico malicioso y predecir si una nueva muestra es legítima o maliciosa. Además, SVM puede manejar grandes conjuntos de datos y tiene la capacidad de manejar datos no lineales mediante el uso de una función de kernel, lo que permite separar clases en espacios de alta dimensión mediante la transformación de características, lo que hace que sea útil para el procesado de datos complejo como el de tráfico de red que puede ser un ataque DDoS.

3.4.5. *Random Forest*

Random Forest es un algoritmo de aprendizaje automático de ensamble ampliamente utilizado para resolver problemas de clasificación y regresión. Su objetivo es predecir la etiqueta de clase de un objeto basándose en un conjunto de variables de entrada, es decir, características. Esta técnica de ensamble se compone de múltiples árboles de decisión y combina sus salidas para obtener un resultado más preciso.

En el contexto de la detección de ataques DDoS, *Random Forest* puede ser útil, ya que puede manejar grandes volúmenes de datos. Un ataque DDoS genera grandes cantidades de datos y puede ser difícil analizarlos de manera efectiva. *Random Forest* puede manejar conjuntos de datos grandes y complejos, lo que lo hace adecuado para el análisis de ataques DDoS. Por otra parte, los ataques DDoS pueden involucrar patrones sutiles y cambiantes que pueden ser difíciles de detectar para los sistemas de detección tradicionales. *Random Forest* puede identificar patrones complejos en los datos y detectar incluso los ataques más sutiles.

Además de esto, *Random Forest* es tolerante al ruido y a los datos faltantes,

	Predicción positiva	**Predicción negativa**
Clase positiva	Verdaderos positivos	Falsos negativos
Clase negativa	Falsos positivos	Verdaderos negativos

Tabla 3.1: Matriz de confusión genérica para un problema de clasificación binaria

lo que significa que puede manejar conjuntos de datos incompletos o con datos innecesarios sin afectar negativamente la precisión del modelo, este tipo de datos es habitual en el tráfico de red. Esta característica es positiva, aunque en este trabajo, el preprocesado y tratamiento de los datos elimina ese tipo de situaciones.

3.5. Fase V: Evaluación

En este apartado se describen las métricas y los experimentos que se van a utilizar para medir el rendimiento de los algoritmos seleccionados para abordar el problema descrito. Las métricas que se van a emplear son la precisión, el *recall*, *F1-Score* y la matriz de confusión.

3.5.1. Precisión

Es una métrica de evaluación que se utiliza en problemas de clasificación para medir la proporción de verdaderos positivos en relación con el total de predicciones positivas realizadas, midiendo la capacidad del modelo para predecir los verdaderos positivos, minimizando las falsas alertas.

3.5.2. *Recall*

El *recall* se utiliza para medir la proporción de verdaderos positivos en relación con el total de casos positivos reales. Limitando la cantidad de falsos negativos, evitando tráfico malicioso no detectado.

3.5.3. F1-Score

Dicha métrica de evaluación proporciona un equilibrio entre la precisión y el *recall*, combinando ambas y proporcionando una medida más completa de su rendimiento, por lo que puede evaluar el rendimiento del algoritmo y optimizar sus parámetros para lograr una alta precisión y recall al mismo tiempo.

3.5.4. Matriz de confusión

La matriz de confusión permite visualizar la cantidad de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos obtenidos en el conjunto de datos evaluado.

Como podemos ver en la tabla 3.1 en una matriz de confusión, cada fila representa las instancias en una clase real, mientras que cada columna representa las instancias en una clase predicha por el modelo. Por lo tanto, la diagonal principal de la matriz de confusión corresponde a los casos en los que la clase predicha es igual a la clase real, es decir, los verdaderos positivos y verdaderos negativos. Las celdas fuera de la diagonal principal corresponden a los casos en los que la clase predicha es diferente de la clase real, es decir, los falsos positivos y falsos negativos.

3.5.5. Rendimiento del sistema

Se lleva a cabo un análisis del tiempo y recursos consumidos por la función de selección del algoritmo, entrenamiento y prueba del mismo sobre el conjunto de datos. Se trata de una información importante en el problema planteado, ya el tiempo y el consumo de recursos son los factores atacados por un ataque DDoS.

4

Implementación y evaluación

En este capítulo se describen los *frameworks* utilizados para el desarrollo del software solución al problema planteado, así como la arquitectura general del mismo. Para dicho análisis se presentan las tecnologías y herramientas disponibles en el marco del proyecto, explicando su funcionalidad y utilidad para el mismo. Por otra parte, se proporciona una visión general del flujo de funcionamiento del programa solución, explicando los componentes que lo forman. Por último, se explican y comparan los resultados obtenidos con la selección de algoritmos realizada.

4.1. Selección de *frameworks*, herramientas y librerías

A continuación se desarrolla la selección de frameworks, herramientas y librerías seleccionadas.

4.1.1. Framework

En primer lugar, se ha elegido la herramienta de programación en línea Google Colab, ya que permite la ejecución de código en la nube sin consumir recursos locales, proporcionando el acceso a recursos de hardware como CPU, GPU y TPU de alta calidad, un aspecto positivo en el procesado de conjuntos de datos de gran

tamaño, para los cuales los recursos locales pueden resultar limitados.

4.1.2. Lenguaje de programación

Por otra parte, se ha seleccionado Python como lenguaje de programación debido su simplicidad, facilidad de uso y a la gran cantidad de bibliotecas y herramientas disponibles para el procesamiento de datos y el aprendizaje automático, así como para la visualización de datos, como Numpy, Pandas, Matplotlib, Scikit-learn, TensorFlow, etc. Además de esto, Python es un lenguaje multiplataforma, lo que facilita su ejecución en distintos sistemas operativos y plataformas, permitiendo el despliegue del modelo en diferentes entornos.

4.1.3. Librerías

Pandas [20]

Se utiliza la librería Pandas para la carga, procesado y tratamiento de los datos, ya que gracias a esta librería podemos convertir el conjunto de datos a formato de *dataframe*, con las ventajas de su uso y modificación sencillas.

Sklearn [21]

Sklearn es una librería de aprendizaje automático diseñada para ser eficiente en términos de procesamiento y recursos, algo positivo para la resolución del problema planteado, en el que es necesario el análisis de tráfico. Por otra parte, dispone de una gran cantidad de algoritmos de aprendizaje automático ya implementados, entre los que se encuentran los seleccionados en este trabajo, lo cual permite llevar a cabo la comparativa de distintos tipos.

NumPy [22]

También se utiliza la librería NumPy, en este caso durante el tratamiento de datos. Se ha seleccionado esta librería debido a la implementación de funciones matemáticas avanzadas que facilitan el tratamiento de los datos. También es importante resaltar su integración con otras librerías como Sklearn, también utilizada en este trabajo.

Matplotlib [23]

Por otra parte, Matplotlib se utiliza en esta aproximación, ya que ofrece ventajas como la personalización de gráficos y la visualización de datos. Además, también permite la integración con otras librerías como NumPy y Sklearn, descritas previamente. Esto significa que se puede utilizar Matplotlib junto con estas bibliotecas para visualizar los datos de manera efectiva.

Seaborn [24]

La librería Seaborn permite la visualización estadística avanzada y la personalización de gráficos. En el caso de este trabajo, se hace uso de la misma para la visualización de la matriz de correlación de los componentes del conjunto de datos.

Time [25]

Mediante la librería Time se puede medir el rendimiento de un fragmento de código o función en relación al tiempo consumido por el mismo, se resta el tiempo anterior al código que contiene el entrenamiento y prueba del algoritmo seleccionado y el posterior a su ejecución, restando ambos.

4.1.4. Resources

Resources es una librería que permite conocer la cantidad de recursos del sistema consumidos por un proceso, lo que permite conocer el rendimiento de los algoritmos seleccionados.

4.2. Arquitectura software de la solución

La arquitectura del software de la solución ofrecida se basa en funciones modulares que se combinan para realizar tareas específicas de limpieza y preparación de datos, y aplicar algoritmos de aprendizaje automático para su análisis y visualización.

En primer lugar, se presentan funciones relacionadas con la carga de datos desde un archivo CSV, convirtiéndolos en un *dataframe* que se pueda tratar mediante la librería Pandas de Python, así como el tratamiento del mismo, filtrando

```
Uso de memoria: 2707308 bytes
Precision: 0.4921285831449661
Recall: 0.4991444899063335
F1-score: 0.4956117083783802
Confusion Matrix:
+---------------+---------------------+---------------------+
|               | Predicción negativa | Predicción positiva |
+---------------+---------------------+---------------------+
| Real negativa |        75968        |        76167        |
| Real positiva |        74059        |        73806        |
+---------------+---------------------+---------------------+
Tiempo de ejecución:  1.0177481174468994  segundos
```

Figura 4.1: Resultados obtenidos con el algoritmo aleatorio

los datos para mantener tan solo los necesarios y en un formato que pueda ser utilizado por los algoritmos seleccionados. Para ello se utilizan funciones que lleven a cabo el análisis de los datos e identifiquen aquellas características que resulten poco relevantes o no aporten suficiente información.

También se implementan funciones para cada uno de los algoritmos seleccionados y para la medición del rendimiento, la eficacia y eficiencia de los mismos.

4.3. Evaluación experimental y comparativa

4.3.1. Algoritmo Aleatorio

El primer algoritmo que se prueba consiste en un clasificador aleatorio como base para comparar otros algoritmos con el fin de comprobar que estos mejoran los resultados de un algoritmo que clasifica las entradas del conjunto de datos mediante el azar y sin tener en cuenta las características del mismo. Por otra parte, se trata de una primera prueba para comprobar el correcto funcionamiento de la arquitectura desarrollada. Como se puede ver en la imagen 4.1, se obtienen resultados cercanos al 50 % de precisión en relación a las medidas de rendimiento seleccionadas.

4.3.2. Regresión Logística

Para el estudio de la efectividad de este algoritmo con el conjunto de datos seleccionado, se han escogido los siguientes valores para los parámetros:

```
Uso de memoria: 2707308 bytes
Precision: 0.9979641444299945
Recall: 0.9912284854428025
F1-score: 0.9945849110378241
Confusion Matrix:
+---------------+---------------------+---------------------+
|               | Predicción negativa | Predicción positiva |
+---------------+---------------------+---------------------+
| Real negativa |        151836       |         299         |
| Real positiva |         1297        |        146568       |
+---------------+---------------------+---------------------+
Tiempo de ejecución:  257.80441522598267  segundos
```

Figura 4.2: Resultados obtenidos con el algoritmo Regresión Logística

- **solver:** el valor que ofrece mejores resultados es "saga", dicho valor suele funcionar mejor para grandes conjuntos de datos y soporta el valor l1 para el argumento penalty (el utilizado por defecto es "lbfgs").
- **C:** El valor óptimo escogido es 0.83, por defecto se utiliza 1, con un número mayor o menor, el resultado empeora.
- **penalty:** se elige "l1", ya que es el soportado por el valor "saga" del argumento "solver", además, sirve para reducir la dimensionalidad de los datos, algo positivo en el caso del conjunto de datos escogido, ya que tenemos muchas características.
- **multi_class:** se escoge el valor "multinomial", para tener en cuenta las correlaciones entre clases
- **max_iter:** Con el valor "200" se mejora la precisión, por defecto se utiliza 100.

Con los argumentos explicados y los valores escogidos, se obtienen los resultados de la figura 4.2. Como puede observarse, se obtiene una precisión del 99 % para los parámetros de medición seleccionados.

4.3.3. KNN

En el caso de KNN se fijan los siguientes valores para los parámetros:

- **n_estimators:** Por defecto tiene el valor "100", al aumentarlo 110 se mantiene la precisión, al disminuirlo, se empeora.

```
Uso de memoria: 2707308 bytes
Precision: 0.9993162694033942
Recall: 0.9983295573665167
F1-score: 0.9988226696979537
Confusion Matrix:
+---------------+---------------------+---------------------+
|               | Predicción negativa | Predicción positiva |
+---------------+---------------------+---------------------+
| Real negativa |        152034       |         101         |
| Real positiva |         247         |        147618       |
+---------------+---------------------+---------------------+
Tiempo de ejecución:  6898.2984738349915  segundos
```

Figura 4.3: Resultados obtenidos con el algoritmo KNN

- **criterion:** Se escoge el valor "entropy", por defecto es "gini", al cambiarlo a entropy, se mejora la precisión.
- **max_depth:** Con e conjunto de datos el valor que mejor funciona es "13", de manera predeterminada se utiliza "None", al aumentarlo, mejora la precisión.
- **max_features:** El valor estándar para este argumento es "todas", al disminuirlo a 10 se aumenta la precisión, si se aumenta, la precisión empeora.

Con los argumentos explicados y los valores escogidos, se obtienen los resultados de la figura 4.3. Como puede observarse, se obtiene una precisión cercana al 100 % para los parámetros de medición seleccionados.

4.3.4. SVM

Para el algoritmo SVM se utilizan los siguientes parámetros:

- **C:** Al aumentar C, se permite un mayor grado de sobreajuste para maximizar la precisión, mientras que un valor más bajo de C se consigue un más peso a la generalización para obtener una mejor precisión en los datos no vistos. Tras distintas pruebas, se consigue la máxima precisión con el valor "2".
- **kernel:** Puede adquirir los valores de "lineal", "poly" y "rbf". Se obtienen los mejores resultados con el valor "linear".
- **decision_function_shape:** Puede tomar los valores de "ovr" (por defecto) o ovo. Con "ovr" se asigna un clasificador SVM para cada clase y se selecciona la clase con mayor valor obtenido en la función de decisión, mientras

```
Uso de memoria: 2707308 bytes
Precision: 0.9975967463643086
Recall: 0.9853650289115071
F1-score: 0.9914431625254749
Confusion Matrix:
+---------------+---------------------+---------------------+
|               | Predicción negativa | Predicción positiva |
+---------------+---------------------+---------------------+
| Real negativa |        151784       |         351         |
| Real positiva |         2164        |        145701       |
+---------------+---------------------+---------------------+
Tiempo de ejecución:  3637.393208503723  segundos
```

Figura 4.4: Resultados obtenidos con el algoritmo SVM

que con "ovo" se asigna un clasificador SVM para cada par de clases. Se elige "ovo" al obtener mejores resultados con dicho valor.

- **tol:** Mide la precisión del algoritmo, con valores más bajos se mejora la precisión del algoritmo, pero se aumenta el tiempo que tarda en ejecutarse. Por defecto adquiere el valor de "1e-3", para este trabajo se ha seleccionado "1e-7".

Con los valores escogidos, se obtienen los resultados de la figura 4.4. Como puede observarse, se obtiene una precisión del 99 % para los parámetros de medición seleccionados.

4.3.5. Random Forest

Se han elegido los siguientes valores para los parámetros con el fin de estudiar el rendimiento de este algoritmo cuando se usa con el conjunto de datos elegido:

- **criterion:** Por defecto toma el valor de "gini", al establecer el valor "entropy" se mejora la precisión.
- **max_depth:** Este parámetro se utiliza para la poda de los árboles de decisión y se establece en "None" por defecto. Al cambiarse por "10", aumenta la precisión.
- **max_features:** Cantidad máxima de características que se contemplan en un árbol de decisión, por defecto se contemplan todas. El valor óptimo para este trabajo es "10".

Con dicha selección de valores y parámetros, se obtienen los resultados en relación al rendimiento y a la precisión indicados en la figura 4.5, con un 99 % de precisión en las métricas seleccionadas.

```
Uso de memoria: 2707308 bytes
Precision: 0.9999864738742882
Recall: 0.9999661853717918
F1-score: 0.9999763295201318
Confusion Matrix:
+---------------+---------------------+---------------------+
|               | Predicción negativa | Predicción positiva |
+---------------+---------------------+---------------------+
| Real negativa |        152133       |          2          |
| Real positiva |          5          |        147860       |
+---------------+---------------------+---------------------+
Tiempo de ejecución:  231.52248001098633  segundos
```

Figura 4.5: Resultados obtenidos con el algoritmo Random Forest

4.3.6. Resumen de resultados obtenidos

Cabe destacar, en primer lugar, el hecho de que no se ha trabajado con el conjunto de datos completo, sino que se han llevado experimentos con una reducción del mismo, un millón de líneas escogidas de forma aleatoria con el fin de reducir la complejidad, el tiempo y recursos consumidos para poder ejecutar la arquitectura desarrollada en la plataforma seleccionada (Google Colab).

Como podemos observar en la figura 4.1, los algoritmos que mejores resultados ofrece en términos de precisión son KNN y Random Forest, lo que significa que con dicho algoritmo se minimizan los falsos positivos. Por otra parte, en relación al recall, el mejor algoritmo es Random Forest, por lo que con este algoritmo se minimiza la cantidad de tráfico malicioso no detectado. Teniendo en cuenta la métrica de F1-Score, los algoritmos más óptimos también son KNN y Random Forest, siendo este también último el que menos falsos positivos y negativos identifica.

Otra de las medidas relevantes es el tiempo de ejecución, en el caso del experimento realizado, se mide el tiempo que tarda en ejecutarse, la creación del modelo con el algoritmo seleccionado, el entrenamiento con los datos separados para ello y la fase de pruebas con los datos de validación. Con respecto a este tiempo de ejecución, se observa una diferencia notable entre los diferentes algoritmos, siendo KNN el que más tiempo de ejecución consume y Random Forest

Métricas	**Aleatorio**	**Regresión Logística**	**KNN**	**SVM**	**Random Forest**
Uso de memoria (MB)	2.581	2.581	2.581	2.707	2.581
Precisión	0.429	0.998	0.999	0.998	0.999
Recall	0.499	0.991	0.998	0.985	0.999
F1-Score	0.496	0.994	0.999	0.991	0.999
Falsos Positivos	74059	1297	247	2164	5
Falsos Negativos	73806	299	101	351	2
Tiempo de ejecución	1 s	4 min	1.9 h	1 h	4 min

Tabla 4.1: Resumen de resultados obtenidos en el estudio

el que menos.

Teniendo en cuenta los resultados obtenidos, el algoritmo de uso recomendable es Random Forest, ya que es con el que mejores resultados se obtiene en todas las métricas seleccionadas y el que menos tiempo consume, con un consumo de memoria comparable al del resto de técnicas.

5

Conclusiones y trabajos futuros

En este capítulo se presentan las conclusiones más importantes derivadas de la realización de este trabajo, y se proponen algunas líneas interesantes de trabajo futuro.

5.1. Conclusiones

Gracias al desarrollo de este trabajo se ha podido comprender qué son los ataques de Denegación de Servicio (DoS), así como los distintos tipos de patrones existentes para llevarlos a cabo y el impacto que tienen sobre la disponibilidad de los servicios en red. También se ha podido comprobar que las técnicas tradicionales para prevenir y detectar este tipo de ataques encuentran cada vez más limitaciones.

Por este motivo se están evaluando diferentes técnicas y mecanismos basados en aprendizaje automático e inteligencia artificial que ayuden a realizar esta prevención y detección de manera más sofisticada. Para entrenar los modelos en los que se basan estas soluciones es necesario disponer de conjuntos de datos relevantes para el dominio de aplicación. Se ha comprobado en este trabajo que existen algunos disponibles que ayudan a realizar una primera aproximación al problema.

Gracias a ellos se ha podido seguir una metodología habitual en proyectos de ciencia de datos para implementar diferentes modelos de aprendizaje automático con el fin de clasificar el tráfico entrante en un servidor en dos categorías diferentes: legítimo o malicioso. En concreto, se han implementado los algoritmos de regresión logística, KNN, SVM y Random Forest.

Una vez entrenados se han realizado experimentos para evaluar su rendimiento en términos de precisión en cuanto a la correcta clasificación del tráfico, de consumo de recursos del sistema (principalmente, memoria) y de tiempo de ejecución. Todo el código desarrollado puede consultarse en el siguiente repositorio de Github: Detecting-DDoS-with-ML-techniques. De esta forma, se concluye que Random Forest se trata del modelo con el que se consiguen mejores resultados para este problema en concreto.

5.2. Trabajo futuro

Tras la realización del trabajo surgen varias líneas en las que sería interesante continuar investigando.

La primera línea de trabajo que se propone consiste en realizar los experimentos descritos en este TFG con el conjunto de datos completo en un entorno de altas prestaciones. La segunda tiene que ver con la selección de los valores de los diferentes parámetros de los modelos entrenados, en lugar de realizarse como en este TFG, buscando por prueba-error, se podría investigar acerca de los métodos que permiten optimizar sus valores para un problema o conjunto de datos concreto.

Por último, se propone continuar este trabajo con la implementación de Redes Neuronales Recurrentes que analicen datos de series temporales, ya que se ha comprobado que la información temporal es especialmente relevante para la correcta detección de los ataques de denegación de servicio.

Bibliografía

[1] National Institute of Standards and Technology, "Security and privacy controls for information systems and organizations," https://doi.org/10.6028/NIST.SP.800-53r5, 2020, special Publication 800-53, Revision 5.

[2] S. Baset and H. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 5, pp. 961–971, 2006.

[3] L. Garber, S. Jha, and S. Banerjee, "An analysis of smurf attacks and their defenses," in *Proceedings of the 22nd Annual Computer Security Applications Conference*. ACM, 2006, pp. 191–200.

[4] H. Chen and V. Paxson, "Nuke attacks: Bypassing source address filtering and countermeasures," in *Proceedings of the 8th USENIX Security Symposium*, 1999, pp. 129–140.

[5] O. Bahiga, I. Al-Anbagi, and L. Qabajeh, "Session paralysis attack in sip-based voip networks: A comprehensive review," *Journal of Network and Systems Management*, vol. 27, no. 2, pp. 291–316, 2019.

[6] B. H. Han, D. H. Lee, J. W. Kim, and H. Lee, "Ddos attack detection using the entropy of a buffer overflow in a wireless sensor network," *IEEE Access*, vol. 6, pp. 23 610–23 619, 2018.

[7] F. Yarochkin and A. Nikitin, "Classical dos attack and defense mechanisms," in *2015 9th International Conference on Application of Information and Communication Technologies (AICT)*. IEEE, 2015, pp. 1–6.

[8] A. Chowdhury and S. Abdulla, "A survey of ddos attack and defence mechanisms," *Journal of Network and Computer Applications*, vol. 110, pp. 70–85, 2018.

[9] C. C. Fung and K.-Y. Lam, "A distributed intrusion detection system for dos attacks," in *Proceedings of the 2003 Symposium on Applications and the Internet*, ser. SAINT '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 407–413.

[10] S. B. Parvinder Singh Saini, Sunny Behal, "Detection of ddos attacks using machine learning algorithms," 2020.

[11] S. V. K. S. Malliga, P. S. Nandhini, "A comprehensive review of deep learning techniques for the detection of (distributed) denial of service attacks," *Information Technology and Control*, vol. 51, no. 1, 2022.

[12] P. Dutta Sai Eswari, "A survey on detection of ddos attacks using machine learning approaches," *Turkish Journal of Computer and Mathematics Education*, vol. 12, no. 11, pp. 4923–4931, 2021.

[13] M. A. Butun, İ. Korkmaz, and A. Çelik, "A survey of intrusion detection systems using machine learning techniques," *Journal of Network and Computer Applications*, vol. 103, pp. 1–25, 2018.

[14] D. C. Nguyen, D. T. Dao, and M. T. Thai, "Machine learning-based ddos attack detection: A comprehensive review," *Computer Networks*, vol. 170, p. 107143, 2020.

[15] C. A. M Devendra Prasad, Prasanta Babu V, "Machine learning ddos detection using stochastic gradient boosting," *International Journal of Computer Sciences and Engineering*, vol. 7, pp. 157–166, 4 2019. [Online]. Available: https://www.ijcseonline.org/full_paper_view.php?paper_id=4011

[16] ——, "Machine learning ddos detection using stochastic gradient boosting," *International Journal of Computer Sciences and Engineering*, vol. 7, pp. 157–166, 4 2019. [Online]. Available: https://www.ijcseonline.org/full_paper_view.php?paper_id=4011

[17] C. I. for Cybersecurity, "Cse-cic-ids2018: Amazon web services (aws) dataset," https://www.unb.ca/cic/datasets/ids-2018.html, 2018, accessed:06/03/2023.

[18] M. Sharafaldin, A. Ghorbani, and M. Karimzadeh, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," 2018.

[19] H. Hadian Jazi, H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Detecting http-based application layer dos attacks on web servers in the presence of sampling," *Computer Networks*, vol. 117, pp. 109–126, 2017.

[20] *pandas documentation*, 1st ed., 2023. [Online]. Available: https://pandas.pydata.org/docs/

[21] *scikit-learn user guide*, 1st ed., 2023. [Online]. Available: https://scikit-learn.org/stable/user_guide.html

[22] *NumPy documentation*, 1st ed., 2022. [Online]. Available: https://numpy.org/doc/stable/

[23] *Matplotlib 3.7.1 documentation*, 3rd ed., 2023. [Online]. Available: https://matplotlib.org/stable/index.html

[24] *seaborn API reference*, 0th ed., 2022. [Online]. Available: https://seaborn.pydata.org/api.html

[25] *Time access and conversions*, 3rd ed., 2023. [Online]. Available: https://docs.python.org/3/library/time.html