UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA

FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

TÍTULO DEL TRABAJO DE TITULACIÓN

Sistema basado en Deep Learning para la detección y clasificación de Malware en el tráfico de Red.

AUTOR

Pozo Quirumbay, Olmedo Josue

PROYECTO DE UNIDAD DE INTEGRACIÓN CURRICULAR

Previo a la obtención del grado académico en
INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

TUTOR

Ing. Coronel Suárez Iván Alberto, Msia.

Santa Elena, Ecuador

Año 2025

UPSE

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

TRIBUNAL DE SUSTENTACIÓN

Ing. José Sánchez Aquimo. Msc.
DIRECTOR DE LA CARRERA

Ing. Iván Coronel Suárez. Msia.
TUTOR

Lsi. Daniel Quirumbay Yagual. Msia
DOCENTE ESPECIALISTA

Ing. Marjorie Coronel Suárez. Mgti.
DOCENTE GUÍA UIC

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

CERTIFICACIÓN

Certifico que luego de haber dirigido científica y técnicamente el desarrollo y estructura final del trabajo, este cumple y se ajusta a los estándares académicos, razón por el cual apruebo en todas sus partes el presente trabajo de titulación que fue realizado en su totalidad por **Olmedo Josue Pozo Quirumbay**, como requerimiento para la obtención del título de Ingeniero en Tecnologías de la Información.

La Libertad, a los 17 días del mes de noviembre del año 2025

TUTOR

Firmado electrónicamente por:
IVAN ALBERTO CORONEL SUAREZ
Validar únicamente con FirmaEC

Ing. Iván Coronel Suárez, Msia

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES
DECLARACIÓN DE RESPONSABILIDAD

Yo, **Olmedo Josue Pozo Quirumbay**

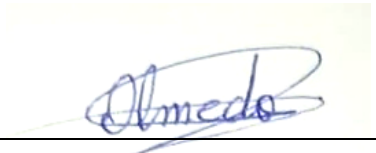
DECLARO QUE:

El trabajo de Titulación, Sistema basado en Deep Learning para la detección y clasificación de malware en el tráfico de red previo a la obtención del título en Ingeniero en Tecnologías de la Información, ha sido desarrollado respetando derechos intelectuales de terceros conforme las citas que constan en el documento, cuyas fuentes se incorporan en las referencias o bibliografías. Consecuentemente este trabajo es de mi total autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance del Trabajo de Titulación referido.

La Libertad, a los 17 días del mes de noviembre del año 2025

EL AUTOR



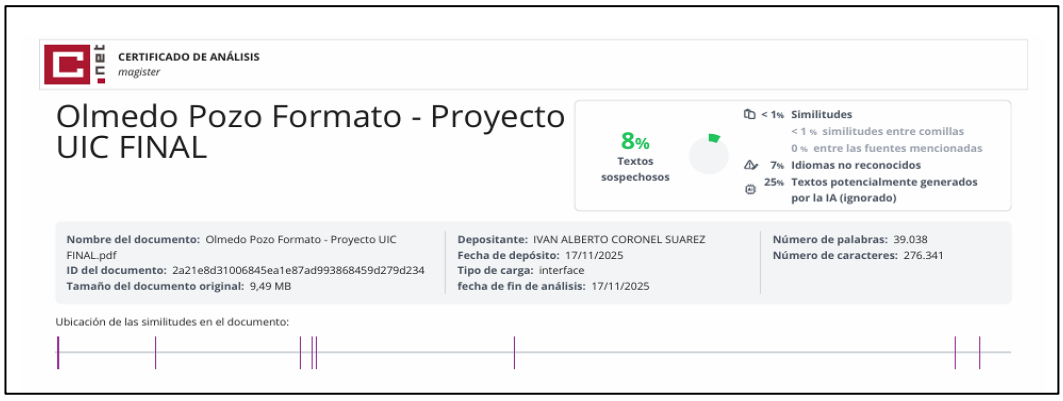
Olmedo Josue Pozo Quirumbay

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA

FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

CERTIFICACIÓN DE ANTIPLAGIO

Certifico que después de revisar el documento final del trabajo de titulación denominado **Sistema basado en Deep Learning para la detección y clasificación de malware en el tráfico de red**, presentado por el estudiante, Olmedo Josue Pozo Quirumbay fue enviado al Sistema Antiplagio, presentando un porcentaje de similitud correspondiente al 8%, por lo que se aprueba el trabajo para que continúe con el proceso de titulación.



CERTIFICADO DE ANÁLISIS
magister

Olmedo Pozo Formato - Proyecto UIC FINAL

8% Textos sospechosos

< 1% Similitudes
< 1 % similitudes entre comillas
0 % entre las fuentes mencionadas
7% Idiomas no reconocidos
25% Textos potencialmente generados por la IA (ignorado)

Nombre del documento: Olmedo Pozo Formato - Proyecto UIC FINAL.pdf
ID del documento: 2a21e8d31006845ea1e87ad993868459d279d234
Tamaño del documento original: 9,49 MB

Depositante: IVAN ALBERTO CORONEL SUAREZ
Fecha de depósito: 17/11/2025
Tipo de carga: interface
fecha de fin de análisis: 17/11/2025

Número de palabras: 39.038
Número de caracteres: 276.341

Ubicación de las similitudes en el documento:

TUTOR

Firmado electrónicamente por:
IVAN ALBERTO CORONEL SUAREZ
Validar únicamente con FirmaEC

Ing. Iván Coronel Suárez, Msia

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

AUTORIZACIÓN

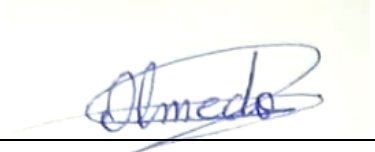
Yo, **Olmedo Josue Pozo Quirumbay**

Autorizo a la Universidad Estatal Península de Santa Elena, para que haga de este trabajo de titulación o parte de él, un documento disponible para su lectura consulta y procesos de investigación, según las normas de la Institución.

Cedo los derechos en línea patrimoniales del presente trabajo de titulación con fines de difusión pública, además apruebo la reproducción dentro de las regulaciones de la Universidad, siempre y cuando esta reproducción no suponga una ganancia económica y se realice respetando mis derechos de autor.

Santa Elena, a los 17 días del mes de noviembre del año 2025

EL AUTOR



Olmedo Josue Pozo Quirumbay

AGRADECIMIENTO

En primer lugar, agradezco a Dios por darme la sabiduría, fortaleza y perseverancia necesarias para superar cada obstáculo durante este proceso académico y nunca dejarme solo en los momentos difíciles.

Agradezco profundamente a mis padres, Olmedo Pozo y Narcisa Quirumbay, por su amor incondicional y por ser mi mayor fuente de inspiración. Gracias por su apoyo constante, por sus consejos y por enseñarme el valor del esfuerzo y la perseverancia. Cada paso que he dado en este camino ha sido posible gracias a los valores que me inculcaron y al sacrificio que realizaron para que pudiera alcanzar mis metas.

También extiendo mi agradecimiento a mis hermanos y a mi familia en general, por sus palabras de ánimo, por creer en mí y por acompañarme durante este proceso.

Olmedo Josue, Pozo Quirumbay

DEDICATORIA

Dedico este trabajo, con profundo cariño y gratitud, a mis padres, quienes han sido la base de cada uno de mis logros. Su esfuerzo, sacrificio y amor incondicional me han ayudado en todo momento. Todo lo que he alcanzado es fruto de los valores que me enseñaron y del apoyo que nunca me ha faltado.

Dedico también este logro a mi familia, quienes con sus palabras de aliento y su compañía me brindaron la fortaleza necesaria para continuar avanzando. Su confianza en mí ha sido un impulso constante para llegar hasta aquí.

Olmedo Josue, Pozo Quirumbay

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El presente trabajo desarrolla un sistema basado en Deep Learning para la detección y clasificación de malware en tráfico de red, empleando arquitecturas de redes neuronales profundas (DNN/MLP). Utilizando el dataset Improved CSE-CIC-IDS2018, se entrena un modelo para identificar tres categorías de malware: Botnet, Infiltration y Web Attacks, además de tráfico benigno. El proceso incluye preprocesamiento de datos mediante selección de 40 características, técnicas de balanceo como SMOTE y undersampling, y evaluación mediante validación cruzada estratificada. Los resultados demuestran una precisión global del 99.73%, con F1-Score de 0.9969 y tasas de recall superiores al 99.56% para todas las clases. El sistema se implementa como herramienta de línea de comandos en Python, validándose en escenarios controlados de tráfico real, alcanzando detectabilidad superior al 95% de amenazas de malware. Esta solución proporciona un enfoque automatizado para mejorar la ciberseguridad en entornos de red.

Palabras claves: Deep Learning, Detección de Malware, Tráfico de Red, Sistema de Detección de Intrusiones (IDS), Ciberseguridad

ABSTRACT

This study develops a Deep Learning-based system for malware detection and classification in network traffic, employing deep neural network architectures (DNN/MLP). Using the Improved CSE-CIC-IDS2018 dataset, a model is trained to identify three malware categories: Botnet, Infiltration, and Web Attacks, in addition to benign traffic. The process includes data preprocessing through selection of 40 features, balancing techniques such as SMOTE and undersampling, and evaluation via stratified cross-validation. Results demonstrate global accuracy of 99.73%, with F1-Score of 0.9969 and recall rates exceeding 99.56% across all classes. The system is implemented as a Python command-line tool, validated in controlled real-traffic scenarios, achieving malware threat detectability exceeding 95%. This solution provides an automated approach to enhance cybersecurity in network environments.

Keywords: Deep Learning, Malware Detection, Network Traffic, Intrusion Detection System (IDS), Cybersecurity.

INTRODUCCIÓN

El presente proyecto, titulado "Sistema basado en Deep Learning para la detección y clasificación de malware en el tráfico de red", se centra en el desarrollo de un sistema que utiliza aprendizaje profundo para identificar y clasificar distintas amenazas. El enfoque permite reconocer patrones asociados a actividades maliciosas dentro del tráfico de red y generar resultados de clasificación a partir del análisis de los datos capturados.

El tráfico se clasifica como benigno o malicioso—identificando tipos específicos de amenazas como Botnet, Web Attacks e Infiltration—, integrando herramientas de inteligencia artificial que superan las limitaciones de los métodos tradicionales de seguridad informática basados en firmas o reglas predefinidas.

En el capítulo 1 se expone el contexto que da origen a la necesidad de mejorar los mecanismos de identificación de amenazas en el tráfico de red. Asimismo, se presenta el enfoque metodológico adoptado para el desarrollo del modelo propuesto, junto con una descripción de las arquitecturas de aprendizaje profundo empleadas, MLP y LSTM, los procedimientos de tratamiento y preparación de los datos, y los criterios utilizados para medir el rendimiento del sistema de clasificación.

El capítulo 2 profundiza en el contexto técnico, la configuración del entorno simulado mediante máquinas virtuales y los experimentos realizados para validar el sistema. Se presentan los resultados obtenidos tras el entrenamiento del modelo, resaltando su capacidad para distinguir de manera confiable las categorías de tráfico malicioso evaluadas y su adecuado desempeño en la clasificación de cada una de ellas.

Este trabajo aporta al desarrollo de estrategias de protección en redes, al evidenciar que las técnicas de aprendizaje profundo permiten mejorar significativamente los procesos de interpretación y clasificación del tráfico de red. Del mismo modo, ofrece lineamientos y experiencias que pueden ser aprovechados en distintos escenarios donde la integridad de la información es prioritaria, favoreciendo la incorporación de soluciones tecnológicas avanzadas para enfrentar de manera más efectiva las amenazas digitales actuales.

CAPÍTULO 1. FUNDAMENTACIÓN

1.1 Antecedentes.

Hoy en día nuestra sociedad depende cada vez más de los servicios digitales para actividades cotidianas y comerciales, lo que expone tanto a usuarios como a organizaciones a diversos ciberataques que pueden comprometer su reputación y recursos financieros [1]. El malware es un software informático de naturaleza maliciosa diseñado para comprometer, dañar o infiltrar computadoras, servidores o redes sin el conocimiento o consentimiento del usuario, a menudo con fines de lucro. Reconocido como una amenaza desde los inicios de la computación personal, el malware es actualmente el vector de ataque cibernético más costoso. Los creadores de malware explotan las vulnerabilidades en los sistemas operativos y priorizan plataformas con mayor cuota de mercado, ya que la concentración de usuarios facilita la propagación y el escalamiento de los ataques. Este enfoque en la cuota de mercado es clave para su estrategia [2].

El tráfico de red continúa siendo un canal esencial para la propagación de malware, con más de 560,000 nuevas variantes detectadas cada día y un aumento del 87% en infecciones en la última década. En 2022 se registraron 5.5 mil millones de ataques de malware, lo que evidencia una persistente actividad maliciosa en los flujos de red. Cada 39 segundos ocurre un intento de intrusión, mientras que diariamente se hackean 30,000 sitios web. Además, emergen 5.33 nuevas vulnerabilidades por minuto, ampliando la superficie de ataque dentro del tráfico digital global [3]. Estos hallazgos resaltan la urgente necesidad de implementar soluciones de ciberseguridad efectivas para proteger tanto a los usuarios como a las organizaciones ante un panorama de amenazas que se encuentra en constante evolución.

El comportamiento de red del malware es un aspecto fundamental que debe ser exhaustivamente considerado en su análisis y clasificación automatizada. A pesar de que la investigación existente y la mayoría de los enfoques basados en el comportamiento se han centrado tradicionalmente en el monitoreo y procesamiento de rastros de llamadas al sistema —una tarea intensiva en recursos—, la actividad de red es un componente crucial y subutilizado para la caracterización y

clasificación efectiva de muestras maliciosas. Dado que casi todo el malware moderno se comunica de alguna manera con hosts externos, el tráfico de red proporciona una fuente de información invaluable sobre las actividades de comunicación del malware, permitiendo distinguir y clasificar muestras basándose *únicamente* en su comportamiento de red [4].

Las soluciones tradicionales se han basado en firmas o reglas predefinidas para identificar patrones de ataque, pero estos enfoques fallan frente a ataques nuevos o variantes de malware. Sin embargo, investigaciones como la de Nari y Ghorbani (2013) sugieren que cuando surge una nueva variante de malware, ésta tiende a mostrar un comportamiento de red similar al de su predecesor, a pesar de las técnicas de ofuscación, polimorfismo o metamorfismo utilizadas para crearla [4]. Por ello, se ha recurrido al uso del Deep Learning, una subrama del Machine Learning, para detectar comportamientos anómalos mediante el análisis de características en la red.

En el trabajo de Omopintemi et al. (2023), se aborda la detección de malware en el tráfico de red con el fin de proteger activos digitales, proponiendo un enfoque basado en aprendizaje automático. Los autores utilizaron los algoritmos Random Forest, Support Vector Machine (SVM), Adaboost y K-Nearest Neighbor (KNN). El estudio empleó un conjunto de datos benignos de Kaggle con 10,593 muestras (4238 normales y 6355 benignas), divididas 80% para entrenamiento y 20% para pruebas. El rendimiento de los modelos se evaluó con métricas como precisión, exactitud, recall, F1 Score, Tasa de Verdaderos Positivos (TPR) y Tasa de Falsos Positivos (FPR). Los resultados revelaron que el TPR para todos los clasificadores superó el 97% y el FPR fue inferior al 4%. El algoritmo Adaboost demostró el mejor rendimiento, con un TPR de casi el 100% y un FPR de casi el 0%. Esto indica una identificación altamente precisa de las amenazas [5].

El trabajo de Shilpa P. Khedkar y Aroul Canessane Ramalingam se enfoca en la clasificación del tráfico de red usando un modelo de Perceptrón Multicapa (MLP) con dos capas ocultas. Este modelo fue entrenado con 14,000 registros de un conjunto de datos real de la Universidad de California, Irvine, con 18 características. Gracias al aprendizaje profundo, el modelo extrajo automáticamente las características relevantes, logrando una precisión del 99.28% sin necesidad de

ingeniería de características y superando algoritmos como SVM, AdaBoost y XGBoost en un 13%. La clasificación abarca tráfico periódico, de eventos, de consulta y, especialmente, malicioso, lo que permite bloquearlo tempranamente y mejorar el rendimiento de la red [6].

También El proyecto de R.K. Rahul, T. Anjali, Vijay Krishna Menon y K.P. Soman emplea redes neuronales convolucionales (CNN) y autoencoders para la clasificación de protocolos de red, aplicaciones y malware. Para el análisis del tráfico de red, los investigadores utilizaron sus propios conjuntos de datos. Para la clasificación de malware, se basaron en el conjunto de datos de Microsoft Kaggle, que contiene 9 familias de malware representadas por su contenido binario hexadecimal. Los resultados obtenidos demuestran la efectividad de su enfoque, logrando una precisión del 100% en la clasificación de aplicaciones de navegador y chat, un 98.90% en la clasificación de torrent (incluso con cifrado), y una notable precisión general del 99.63% en la clasificación de aplicaciones. Además, la clasificación de malware alcanzó una precisión promedio del 94.91% [7]. El uso de técnicas avanzadas de Deep Learning para la detección de malware en el tráfico de red es un campo en constante evolución. Estudios recientes han demostrado la efectividad de estos enfoques para mejorar la precisión en la clasificación de diferentes tipos de amenazas en las redes, lo que abre nuevas oportunidades para optimizar los sistemas de ciberseguridad.

1.2 Descripción de proyecto

Este proyecto tiene como objetivo desarrollar un sistema de detección y clasificación de malware en el tráfico de red mediante Deep Learning, abordando la creciente amenaza de ciberataques. Para este proyecto se eligió la metodología CRISP-DM debido a su enfoque estructurado y orientado al procesamiento y análisis de datos, lo cual es esencial en este caso para el desarrollo en Deep Learning.

CRISP-DM establece un proceso claro desde la comprensión inicial de los objetivos hasta la implementación final del modelo, asegurando un enfoque sistemático en el manejo y transformación de datos, que son la base de este sistema de detección, su

flexibilidad permite adaptar el modelo a los requisitos específicos del proyecto, facilitando el ajuste y evaluación de resultados para garantizar la efectividad en la clasificación de tráfico malicioso [8].

Es importante señalar que, dado que CRISP-DM está orientada principalmente a proyectos de minería de datos, en este trabajo no se aplicarán de manera estricta todos los pasos definidos en cada fase. En su lugar, las fases se adaptarán para ajustarse a un enfoque de Inteligencia Artificial y Deep Learning, manteniendo la estructura metodológica, pero enfocándola en los procesos que resultan más relevantes para el sistema propuesto.

A continuación, se presentan las siguientes fases del proyecto [9]:

1. Obtención de Datos

   Se recolectarán los datos de tráfico de red mediante el dataset Improved CSE-CIC-IDS 2018, una versión corregida del dataset original, que mejora la calidad del etiquetado, especialmente en la clase Infiltration, previamente identificada con errores de clasificación como tráfico ARP mal etiquetado. Esta mejora permite trabajar con datos más confiables para el entrenamiento del modelo. Se seleccionarán las categorías Botnet —por ejemplo, el troyano Zeus—, Web Attacks —vulnerabilidades automatizadas mediante DVWA—, e Infiltration —intrusiones y escaneos de puertos—, además de la clase Benign, que representa el tráfico de red normal sin actividad maliciosa. Esta etapa permitirá obtener una base de datos representativa de los patrones de tráfico de red para el análisis.

2. Preprocesamiento de Datos

   Se eliminarán registros duplicados, nulos o que contengan ruido que podría afectar la calidad del modelo. Se identificará y seleccionará las características, es este caso las columnas que mejor representen los patrones de malware, como el Protocolo o duración de flujo, entre otras, descartando aquellas menos relevantes. Se ajustarán los valores de las variables para estandarizar el dataset, asegurando que el modelo de deeplearning procese los datos de manera uniforme.

3. Entrenamiento del Modelo

   Se utilizará una arquitectura como una red neuronal profunda (DNN) para clasificar el tráfico de red en categorías benignas y maliciosas, ajustando

hiperparámetros como el número de capas y tasa de aprendizaje. Se validará el modelo utilizando un conjunto de datos de prueba, midiendo su precisión, recall, F1-Score, y tasa de falsos positivos para asegurar un rendimiento efectivo.

4. Clasificación del Modelo

   El sistema clasificará el tráfico de red en "Clase Benigna" o "Clase Maliciosa", asignando etiquetas específicas para tipos de malware identificados, como Botnet, Web Attacks o Infiltración, según corresponda. Si los resultados del modelo no cumplen con los parámetros establecidos, se realizará una optimización.

5. Implementación.

   El modelo se integrará en un programa de línea de comandos (CLI) desarrollado en python, el cual permitirá ejecutar de forma modular la carga del modelo entrenado, la captura y análisis de tráfico, entre otras. Este programa operará en Windows y estará preparado para interactuar con herramientas como scapy para la captura de tráfico y CICFlowMeter para la extracción de características de tráfico y la conversión a flujos que soporta el modelo entrenado, también empleará scripts en Python para la visualización y registro de resultados.

   Esto permitirá recrear escenarios realistas, evaluando el rendimiento del sistema frente a ataques simulados. Para evaluar el sistema, se considerarán tres categorías de ataques; Botnet, caracterizado por malware que establece comunicación con servidores de control remoto y permite acciones como ejecución de comandos o transferencia de archivos, para lo cual se utilizará herramientas como Ares; Web Attacks, enfocados en vulnerabilidades en aplicaciones web como ataques de fuerza bruta, que pueden ser simuladas mediante entornos vulnerables como DVWA y frameworks de automatización como Selenium; e Infiltración, que incluye accesos no autorizados y técnicas de reconocimiento dentro de la red mediante herramientas como Nmap, mientras que para establecer la conexión inicial se pueden emplear vectores de intrusión como shells reversas generadas con Metasploit. Se supervisará el rendimiento del sistema para asegurar un nivel de desempeño confiable.

1.3 Objetivos de proyecto

1.3.1 Objetivo general

Desarrollar un sistema de detección y clasificación de malware en el tráfico de red utilizando técnicas de Deep Learning que pueda identificar tipos específicos de malware presentes en la red con el fin de mejorar la seguridad y protección de las redes.

1.3.2 Objetivos específicos

- Entrenar un modelo de Deep Learning para la detección y clasificación de malware utilizando redes neuronales profundas.
- Evaluar el rendimiento del sistema utilizando métricas como precisión, recall, F1-Score, y tasa de falsos positivos, con el fin de garantizar su efectividad en la detección de amenazas.
- Probar el sistema en un entorno controlado que simule condiciones de tráfico de red real para analizar su efectividad en la detección de malware.

1.4 Justificación del proyecto

El creciente volumen de ciberataques y la complejidad de las técnicas empleadas por los ciberdelincuentes exigen soluciones avanzadas de ciberseguridad. En 2022, América Latina experimentó más de 360 mil millones de intentos de ataques. Según Derek Manky, de FortiGuard Labs, los atacantes recurren a métodos cada vez más sofisticados para evadir la detección, lo que hace imprescindible que las organizaciones implementen inteligencia de amenazas basada en aprendizaje automático en tiempo real. Estas soluciones permiten detectar y mitigar acciones sospechosas de manera coordinada, fortaleciendo la seguridad en toda la superficie de ataque de una organización y mejorando su capacidad de respuesta frente a amenazas avanzadas [10].

La inteligencia artificial ha impulsado avances significativos en el control y gestión del tráfico de redes de datos, mejorando aspectos clave como la clasificación y predicción de tráfico, la seguridad en redes, y la optimización de rutas. Estos logros son posibles gracias a las ventajas de los algoritmos de aprendizaje automático y

profundo, que ofrecen rapidez, precisión y capacidades de autoaprendizaje. Estas tecnologías permiten una gestión de redes más autónoma y eficiente, optimizando el rendimiento y la calidad del servicio a los usuarios. En este contexto, el aprendizaje profundo se destaca frente a enfoques tradicionales, ya que facilita la clasificación de datos con menor preparación y resulta ideal para optimizar rutas e identificar tráfico malicioso [11].

De acuerdo con MarketsandMarkets, el mercado de inteligencia artificial aplicado a ciberseguridad podría alcanzar un valor de $38.200 millones para 2026, con un crecimiento anual estimado del 23,3%. Por otra parte, se proyecta que la inteligencia artificial en general crecerá a una tasa compuesta anual del 26,4% entre 2022 y 2027, alcanzando aproximadamente $390.900 millones para el final de este periodo Estas cifras resaltan el papel fundamental que tendrá la inteligencia artificial sobre la ciberseguridad en un futuro [12].

El sistema propuesto ayuda a mejorar significativamente la seguridad en entornos que demandan protección continua, como las redes de organizaciones. Al detectar y clasificar malware, este sistema puede reducir la cantidad de incidentes de seguridad y optimizar la respuesta ante amenazas, proporcionando una solución efectiva para mitigar riesgos y proteger datos sensibles. De esta manera, se asegura la continuidad de las operaciones y eficiencia en la gestión de la ciberseguridad dentro de la organización.

Este proyecto está orientado al Plan Nacional de Desarrollo 2025-2029 “Ecuador No Se Detiene”, descrito a continuación.

Eje social

Objetivo 3.- “Garantizar un estado soberano, seguro, y justo promoviendo la convivencia pacífica y el respeto a los derechos humanos” [13].

Política 3.3.- “Potenciar las capacidades de inteligencia y contrainteligencia del Estado que permita identificar, prevenir y neutralizar amenazas que puedan comprometer la seguridad y la estabilidad nacional.” [13].

Literal C.- Incorporar tecnologías emergentes en la ciberinteligencia para identificar, monitorear y analizar amenazas, tendencias y oportunidades asociadas con innovaciones tecnológicas.

1.5 Alcance del proyecto

El presente proyecto se desarrollará en un entorno de laboratorio simulado, utilizando software especializado para la emulación de equipos en el mismo entorno. No se ejecutará en una institución o empresa específica, sino que se implementará en un entorno simulado mediante herramientas como CICFlowMeter, Python y simuladores como VirtualBox. El objetivo es diseñar, entrenar y evaluar un sistema de detección y clasificación de malware en tráfico de red, aclarando que el procesamiento se realizará específicamente sobre flujos de red, debido a que esta es la naturaleza del dataset utilizado. No obstante, también se realizará la captura de tráfico de red, generando archivos en formato PCAP que representen el tráfico capturado. Dichos archivos serán procesados con CICFlowMeter para convertirlos en flujos válidos que puedan ser analizados por el modelo de Deep Learning. Se realizarán las siguientes fases:

En la primera etapa, se contempla la obtención de datos a partir del conjunto Improved CSE-CIC-IDS2018, el cual se selecciona por su confiabilidad en el etiquetado y por corregir errores presentes en la versión original. Se utilizarán únicamente las categorías Botnet, Web Attacks, Infiltration y Benign, asegurando que el sistema se entrene y evalúe en un rango de escenarios de tráfico representativo pero controlado.

La segunda etapa se enfocará en el preprocesamiento del dataset, garantizando que los datos cumplan con los requisitos de calidad y consistencia necesarios. Este proceso incluirá la depuración de registros inválidos, la selección de características clave y la normalización de los datos, con el objetivo de optimizar el rendimiento del sistema bajo un entorno de laboratorio.

En la tercera etapa, se llevará a cabo el entrenamiento del modelo de detección y clasificación, empleando una arquitectura de red neuronal profunda. Esta fase estará orientada a alcanzar métricas de rendimiento aceptables en términos de precisión y recall, priorizando la reducción de falsos negativos en la detección de ataques ya que en ciberseguridad es algo fundamental.

La cuarta etapa comprenderá la clasificación del tráfico de red según las categorías establecidas. El sistema evaluará flujos de red y asignará etiquetas que permitan identificar comportamientos maliciosos o benignos. Se establecerán parámetros

mínimos de calidad para validar el funcionamiento, y en caso de no alcanzarlos, se aplicarán ajustes al modelo.

Finalmente, la quinta etapa abarcará la implementación y pruebas del sistema en un entorno simulado, sin conexión a infraestructura real. Se ejecutarán escenarios controlados con tráfico generado por herramientas específicas para cada tipo de ataque, validando la capacidad del sistema para operar bajo condiciones seguras y reproducibles, y generando reportes con los resultados obtenidos.

Quedan fuera del alcance: la implementación del sistema en redes reales de producción, el análisis de malware ejecutable fuera del flujo de red, la detección de ataques no presentes en las 4 clases seleccionadas del dataset, la inspección de tráfico cifrado punto a punto y la integración directa con otros sistemas de ciberseguridad como firewalls o SIEM. Tampoco se incluirá el desarrollo de interfaces gráficas ni la adaptación del sistema a dispositivos de red embebidos.

1.6 Metodología del proyecto

1.6.1 Metodología de investigación

Esta investigación se enmarca en la creciente necesidad de proteger redes de datos en organizaciones, donde la actividad diaria depende en gran medida de una infraestructura de red segura. En estos ambientes, la presencia de malware puede poner en riesgo tanto la integridad de los datos como la continuidad de las operaciones [14]. Aunque muchas organizaciones implementan soluciones de seguridad, como firewalls comerciales y software antivirus, estas tecnologías a menudo no son suficientes para identificar y clasificar nuevas amenazas de malware que constantemente evolucionan.

1.6.1.1 Enfoque de la investigación

El enfoque cuantitativo, según Ñaupas et al., se fundamenta en la recolección y análisis de datos numéricos, permitiendo la medición precisa de variables y la verificación de hipótesis. De acuerdo con los principios positivistas establecidos por figuras como Augusto Comte y Emilio Durkheim, el enfoque cuantitativo busca descubrir verdades científicas mediante un método riguroso y estructurado [15]. Esto lo hace especialmente relevante en investigaciones de seguridad cibernética que requieren un alto nivel de precisión.

En este contexto, el enfoque cuantitativo es particularmente adecuado, ya que se busca medir el desempeño del modelo de Deep Learning en términos de métricas objetivas como precisión, puntuación F1, entre otras. Estas métricas se utilizarán para evaluar la efectividad del sistema en la detección de amenazas de malware, posibilitando una validación estadística rigurosa de los resultados obtenidos.

1.6.1.2 Alcance de la investigación

El alcance descriptivo en investigación se enfoca en identificar y especificar atributos, rasgos y características de diversos sujetos de estudio. Este enfoque se distingue por recopilar y cuantificar información sobre variables específicas sin establecer vínculos causales o correlacionales entre ellas [16]. Hernández, Fernández y Baptista señalan que su utilidad principal radica en revelar con precisión las múltiples facetas y dimensiones de un fenómeno o contexto determinado. El investigador debe definir claramente qué elementos medirá y de qué población obtendrá los datos. A diferencia de estudios exploratorios o correlacionales, el enfoque descriptivo se limita a caracterizar y documentar la realidad observada [16].

Para el proyecto de detección y clasificación de malware, se optó por el alcance descriptivo, ya que la meta principal es observar y evaluar el rendimiento del modelo de Deep Learning en la clasificación de tráfico de red. En lugar de proponer relaciones o hipótesis que expliquen el comportamiento del tráfico malicioso, el proyecto se enfoca en medir el desempeño del modelo en términos de precisión o en tasa de falsos positivos, proporcionando una visión clara la capacidad de detección bajo distintas condiciones de tráfico.

1.6.1.3 Tipo de investigación

La investigación experimental es un tipo de investigación científica que utiliza el experimento como su principal técnica para verificar hipótesis. Es considerada una de las metodologías más rigurosas debido a su capacidad para manipular y controlar variables independientes y observar los efectos en las variables dependientes. Este enfoque se caracteriza por el uso de herramientas estadísticas, matemáticas y lógicas para medir y controlar las diferencias entre los resultados, lo que permite obtener conclusiones precisas y confiables sobre las relaciones causales entre las variables involucradas [15].

Se ha elegido la investigación experimental para este proyecto debido a la necesidad de validar el sistema propuesto para la detección de malware en el tráfico de red. El análisis experimental permite controlar las variables y simular situaciones de tráfico de red específicas donde se pueden observar los efectos directos de las manipulaciones del modelo de Deep Learning. Este tipo de investigación es ideal para probar el rendimiento del modelo bajo condiciones controladas y para verificar si realmente es capaz de clasificar correctamente el tráfico malicioso y normal, sin la interferencia de variables externas.

1.6.1.4 Método

El método hipotético-deductivo, propuesto por Karl Popper, es una estrategia de investigación que parte de una hipótesis para deducir consecuencias observables y medibles, con el objetivo de confirmar o refutar dichas hipótesis mediante la experimentación. Este enfoque se basa en el principio de falsación, que sostiene que una hipótesis científica debe ser susceptible de ser refutada a través de la observación de los experimentos [15].

Se ha elegido el método hipotético-deductivo debido a que permite validar de manera sistemática las hipótesis formuladas. Dado que el objetivo principal es validar la efectividad de un sistema propuesto para clasificar y detectar tráfico malicioso, se parte de una hipótesis inicial: "El modelo basado en Deep Learning puede detectar y clasificar correctamente el malware en el tráfico de red". De acuerdo con el método hipotético-deductivo, los resultados obtenidos de estos experimentos serán utilizados para confirmar o refutar la hipótesis [15].

1.6.1.5 Población y muestra

La población se define como el conjunto de elementos con características comunes sobre los cuales se busca extender las conclusiones del estudio, abarcando todos los casos posibles de interés en relación con el problema de investigación [17]. En este proyecto, la población comprende todos los registros del conjunto de datos Improved CSE-CIC-IDS2018, ya que contiene una variedad de eventos de tráfico de red que incluyen tanto tráfico legítimo como diversas amenazas de ciberseguridad.

La muestra, por otro lado, es un subconjunto representativo de la población que se extrae con el fin de simplificar el análisis manteniendo la capacidad de generalizar los resultados [17]. En este proyecto, la muestra seleccionada consiste en los registros de categorías específicas del dataset que están directamente relacionadas con el malware, tales como Botnet, ataques web, e infiltración de red desde el interior. Esta selección asegura que el modelo de detección se enfoque únicamente en las amenazas de malware.

1.6.2 Beneficiarios del proyecto

El desarrollo de un sistema inteligente capaz de detectar y clasificar tráfico malicioso en redes ofrece beneficios significativos a distintos actores del entorno digital. En primer lugar, profesionales en ciberseguridad y administradores de red se benefician directamente al contar con una herramienta de apoyo para la detección temprana de intrusiones y amenazas avanzadas, permitiendo una reacción oportuna y precisa. Asimismo, instituciones educativas y centros de investigación pueden aprovechar el sistema como base para estudios aplicados en ciberseguridad y aprendizaje automático.

1.6.3 Variables

La presente investigación se centra en evaluar el efecto de implementar un modelo de Deep Learning para la detección de malware en el tráfico de red. La principal variable cuantificable en este proyecto es la cantidad de instancias de tráfico malicioso detectadas antes y después de aplicar el modelo, midiendo así el impacto del sistema de clasificación. Antes de la implementación del modelo de Deep Learning, se espera que el número de detecciones de tráfico malicioso sea nulo, mientras que, tras la aplicación del modelo, debería observarse un aumento en las detecciones efectivas de malware.

1.6.4 Técnicas de recolección de información

1.6.4.1 Recolección de datos primarios

La recolección de datos primarios implica la obtención de información original y fresca por primera vez. En la investigación experimental, estos datos se recopilan al realizar experimentos, mientras que en la investigación descriptiva y las

encuestas (incluidas las encuestas de muestra o censales), se obtienen mediante observación o comunicación directa con los encuestados. Algunos métodos para recopilar datos primarios incluyen la observación, entrevistas y cuestionarios [18].

1.6.4.2 Recolección de datos secundarios

La recolección de datos secundarios se refiere a la utilización de información que ya ha sido recopilada previamente por otra persona y que ya ha pasado por un proceso estadístico. Las fuentes de datos secundarios incluyen libros, revistas y periódicos previamente publicados, correos electrónicos y datos brutos de estudios no publicados. Las bases de datos también son una fuente donde la información ya está organizada y es de fácil acceso [18].

1.6.4.3 Revisión bibliográfica

La revisión bibliográfica consiste en recopilar y analizar investigaciones previas, artículos académicos y documentación técnica relacionada con la detección y clasificación de malware en tráfico de red. Este proceso facilita la identificación de enfoques y técnicas empleadas por otros investigadores, lo que permite definir metodologías adecuadas y establecer una base teórica sólida para el proyecto [19].

1.6.5 Análisis de recolección de datos

1.6.5.1 Recolección de datos primarios

La recolección de datos primarios en este proyecto corresponde a la captura de tráfico de red generado directamente en un entorno controlado. Esta técnica permite obtener información original y actualizada, reflejando el comportamiento de una red y de posibles amenazas en ejecución para eso se evaluaron las siguientes herramientas:

Característica	Pcapy	Scapy	PyShark
Captura de paquetes en la red	✓	✓	✓
Lista dispositivos disponibles para "sniffer"	✓	✓	✗
Analiza cabeceras Ethernet, IP, TCP, UDP, ICMP	✓	✓	✓
Analiza otros protocolos IP (no solo TCP/UDP/ICMP)	✗	✓	✓
Soporta opciones de TCP como timestamp	✗	✓	✓
Manejo de TCP fragmentado	✗	✓	✓

Característica	Pcapy	Scapy	PyShark
Fácil acceso a tramas y paquetes específicos	✗	✓	✓
Función de callback para cada paquete	✓	✓	✓
Filtrado de tráfico fácil de realizar	✓	✓	✓
Integración con Wireshark	✗	✓	✓
Captura en vivo desde diferentes interfaces	✓	✓	✓
Problemas de instalación reportados	✓	✓	✗
Dependencia de tshark (Wireshark)	✗	✗	✓
Devuelve atributos como cadenas de texto	✗	✗	✓
Creación y envío de paquetes	✗	✓	✗

Tabla 1 Comparativa de Herramientas Python para Captura de Tráfico de Red [20] [21] [22]

En base a la tabla comparativa, se eligió Scapy por su capacidad de personalización avanzada y su soporte para análisis de protocolos complejos, creación y envío de paquetes, y filtrado de tráfico, orientado a capturas desde línea de comandos. También se usara CICFlowMeter, que convierte los paquetes capturados en flujos de red para extraer características estadísticas de estos flujos.

La captura primaria no será utilizada para el entrenamiento del modelo, ya que este proceso requiere grandes volúmenes de datos y etiquetado manual. La recolección primaria se aplicará en una fase final de validación del modelo. Para ello se diseñará una red simulada con máquinas virtuales, en la cual se generará tráfico benigno y malicioso controlado. Dicho tráfico será capturado y posteriormente analizado mediante el modelo previamente entrenado con el dataset seleccionado, con el fin de observar su desempeño frente a un entorno más cercano a una red real.

1.6.4.2 Recolección de datos secundarios

La recolección secundaria se fundamenta en el uso de datasets ya existentes y ampliamente utilizados en la comunidad científica. Este enfoque resulta ventajoso porque permite trabajar con grandes volúmenes de datos previamente etiquetados, lo cual garantiza un entrenamiento adecuado de los modelos de detección sin necesidad de realizar la costosa y compleja tarea de captura y clasificación manual de tráfico. Entre los datasets más relevantes en ciberseguridad se encuentran los siguientes:

Dataset	UNSW-NB15	CIC-IDS2017	CSE-CIC-IDS2018	Improved CSE-CIC-IDS2018
Autores	Intelligent Security Group UNSW Canberra, Australia, Dr Nour Moustafa	Canadian Institute for Cybersecurity (UNB).	Canadian Institute for Cybersecurity (UNB), 2018.	Lisa Liu, Gints Engelen, Timothy Lynar, Daryl Essam, Wouter Joosen.
Año de publicación	2015	2017.	2018.	2022; última actualización 2023 para correcciones
Número de características (features)	~49 features	~79 features.	~80 features	~91 features
Tipos de ataques cubiertos algunos ejemplos	Fuzzer, Backdoor, DoS, Exploits, Recon, Shellcode, Worms, Generic, Analysis.	DoS, DDoS, Brute-force, Web Attacks, Infiltration, Botnet, etc. (varios escenarios).	Brute Force, Heartbleed, Botnet, DoS, DDoS, Web Attacks, Infiltration.	Mismos ataques que CSE-CIC-IDS2018; se corrigieron etiquetas/mapeos de algunos flujos problemáticos.
Fortalezas	Diseñado para combinar tráfico real + tráfico sintético; buena variedad de tipos de ataque.	Relativamente realista; bien usado en literatura; incluye PCAPs y flujos.	Experimento a escala, mucho host, múltiples servicios, buena diversidad de escenarios	Corregido y documentado: arregla artefactos de etiquetado y problemas detectados en algunas versiones; incluye código de etiquetado para reproducibilidad.
Problemas / limitaciones conocidas	Problemas de desbalance entre clases y solapamiento de clases;	Investigaciones han señalado problemas con construcción de flows, etiquetas y artefactos.	Problemas de desbalance entre clases y solapamiento sobre todo con la clase Infiltration	Reduce esos problemas porque corrige etiquetas y documenta errores; aun así, conviene validar y muestrear antes de entrenar.

Dataset	UNSW-NB15	CIC-IDS2017	CSE-CIC-IDS2018	Improved CSE-CIC-IDS2018
Recomendado para	Modelos que necesiten variedad de tipos de ataque / testing de multiclass; estudios de balanceo/clasificación.	Evaluaciones comparativas históricas; investigación sobre ataques clásicos.	Experimentos que busquen realismo de red a escala (multi-host, servicios variados).	Cuando se quiere una versión más fiable y etiquetada— ideal para una reproducibilidad y un menor ruido en etiquetas.

Tabla 2 Comparación entre diversos dataset [23] [24] [25] [26] [27] [28]

En esta investigación se selecciona el Improved CSE-CIC-IDS2018, una versión corregida del conjunto original. Esta mejora corrige inconsistencias de etiquetado detectadas en el dataset base y proporciona un marco más confiable para el entrenamiento de modelos de Deep Learning. Además, el dataset incluye una amplia variedad de escenarios de ataque, pero para este proyecto se seleccionaron; Bening, Botnet, Web Attacks e Infiltration. Su uso garantiza que el modelo cuente con una base sólida para aprender patrones de tráfico tanto benignos como maliciosos.

1.6.4.3 Revisión bibliográfica

La revisión bibliográfica se emplea como técnica complementaria para sustentar teóricamente el proyecto y guiar la toma de decisiones metodológicas. Este proceso consiste en la consulta y análisis de literatura académica especializada. La revisión se orienta a identificar las arquitecturas de aprendizaje profundo más utilizadas, las técnicas de selección de características aplicadas en datasets de seguridad y las métricas de evaluación comúnmente empleadas en el área.

Para garantizar la calidad y actualidad de la información recopilada, la búsqueda se realizará en bases de datos científicas como IEEE Xplore, ScienceDirect, SpringerLink, entre otros, priorizando recientes. Esta técnica permitirá establecer un marco teórico robusto y detectar limitaciones en estudios previos, lo que orientará la propuesta hacia un enfoque más original y contextualizado.

1.7 Metodología de desarrollo

Esta investigación sigue la metodología Cross Industry Standard Process for Data Mining (CRISP-DM) [29]. Las primeras tres fases de la metodología (Comprensión del negocio, Comprensión de los datos, Preparación de los datos) abarcan la preparación inicial general y la base para el modelado y la evaluación. Las últimas tres fases de modelado, evaluación e implementación se producen como parte del enfoque general para el desarrollo y la prueba y evaluación experimental de clasificadores de malware basados en IA [30].

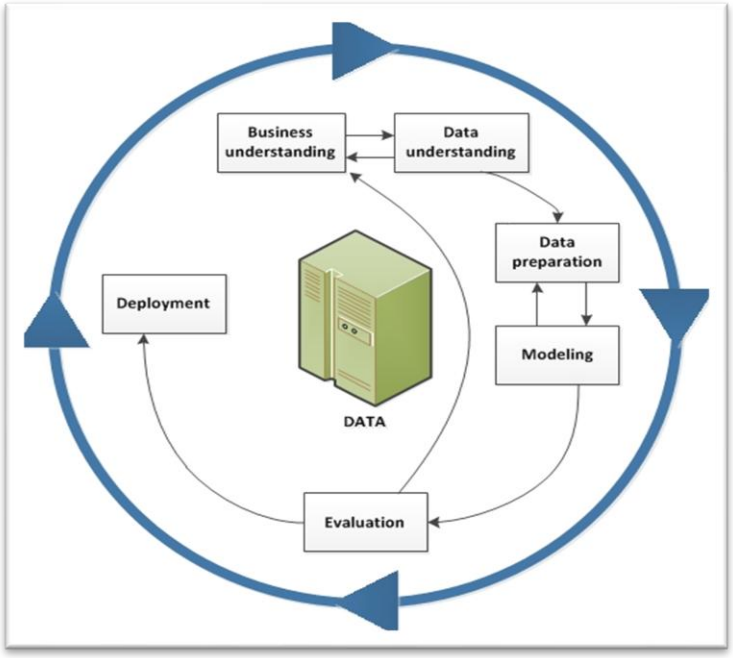


Ilustración 1. El proceso de la Metodología CRISP-DM [31]

A continuación, se describen las siguientes fases:

1. Comprensión del proyecto

   Desarrollar un sistema avanzado de detección y clasificación de malware en el tráfico de red, utilizando técnicas de Deep Learning. La solución abordará la necesidad de mejorar la seguridad de las redes y protegerlas contra ciberataques mediante un análisis detallado de los patrones de tráfico malicioso, permitiendo una detección eficiente de amenazas sin depender de métodos tradicionales basados en firmas o reglas predefinidas.

2. Comprensión de los datos

   Aquí se estudia el dataset Improved CSE-CIC-IDS2018, que contiene tráfico de red simulado con una variedad de actividades, tanto benignas como maliciosas. Se analizan las columnas del dataset, tales como Protocol, Flow Duration, Total Fwd Packets, entre otras, que podrían contener información relevante para detectar la presencia de malware. Se evalúa si el dataset es adecuado, asegurando que contiene muestras relevantes para entrenar el modelo de clasificación de malware.

3. Preparación de los datos

   Esta etapa consiste en limpiar y preprocesar el dataset Improved CSE-CIC-IDS2018. Se eliminan valores nulos o duplicados y se transforman las características relevantes del tráfico en un formato adecuado para el entrenamiento de un modelo de Deep Learning. Se realiza la normalización o escalado de los datos y se dividen en conjuntos de entrenamiento y prueba.

4. Modelado

   En esta fase, se construye y entrena un modelo como el de Redes Neuronales Profundas (DNN) utilizando frameworks como TensorFlow. Se ajustan los hiperparámetros del modelo, como el número de capas y neuronas, las funciones de activación, y la tasa de aprendizaje, para mejorar su precisión en la clasificación del tráfico de red como benigno o malicioso en su clasificación adecuada. Se prueba el modelo en conjunto con datos de prueba para medir su rendimiento inicial.

5. Evaluación

   Tras el entrenamiento del modelo, se procede a una evaluación rigurosa del desempeño utilizando métricas clave como precisión, recall, F1-Score, y tasa de falsos positivos. Estas métricas permiten determinar la capacidad del sistema para identificar y clasificar correctamente el malware en el tráfico de red. La evaluación incluye también un análisis comparativo con estudios previos y

benchmarks establecidos, con el fin de asegurar que el modelo desarrollado ofrezca un rendimiento competitivo y esté alineado con las mejores prácticas.

6. Implementación

   Finalmente, el sistema se implementa en un entorno controlado para la emulación de máquinas y su conectividad como Virtualbox. Las pruebas incluyen escenarios delimitados de malware: para Botnet se utilizará Ares, con simulación de sus comportamientos maliciosos. Para Web Attacks se utilizará ataques de fuerza bruta en un entorno controlado con DVWA y Selenium. Para Infiltración se realizará mediante la explotación de aplicaciones vulnerables o con un reverse Shell en Metasploit, seguido de un análisis de red con herramientas y Nmap. El despliegue se realiza a través de una interfaz de línea de comandos realizada con Python y ejecuta en Windows, la cual facilita el análisis del tráfico de red simulado, permitiendo verificar el rendimiento del modelo.

CAPÍTULO 2. PROPUESTA

2.1 Marco contextual

2.1.1 Contexto global de las amenazas cibernéticas

El malware, definido como software diseñado para ejecutar procesos no autorizados y comprometer sistemas, representó el 5,11% de los incidentes cibernéticos globales entre julio 2023 y junio 2024, con un 2,45% en la Unión Europea. Este software malicioso se utiliza para ejecutar campañas cibernéticas y mantener persistencia en sistemas comprometidos, con variantes comunes como virus, gusanos y troyanos. En América Latina, se reportaron campañas de malware que aprovechaban servicios como Google Cloud Run, lo que indica su impacto en la región. Además, las botnets, redes de dispositivos comprometidos, se utilizan para ataques DDoS, minería de criptomonedas y propagación de malware, representando aproximadamente la mitad de los ataques DDoS observados. Ejemplos como Mirai, que alcanzó un pico de 1,9 Tbps en 2023, y la resurgencia de Qakbot, destacan la persistencia y el impacto de estas amenazas [32].

Las técnicas de infiltración como los backdoors y el escaneo de redes siguen siendo fundamentales en los ciberataques. En 2024, el proyecto XZ Utils sufrió una infiltración mediante ingeniería social, lo que permitió el control remoto a través de conexiones SSH. Además, se observó un 76,29% de vulnerabilidades explotables remotamente a través de la red. En el ámbito web, amenazas como el Cross-site Scripting (XSS), que representó el 22,68% de las debilidades críticas, y la inyección SQL, responsable del 26,63% de las vulnerabilidades más críticas, continúan siendo explotadas. Los ataques de fuerza bruta, incluyendo técnicas como password spraying y credential stuffing, siguen siendo utilizados por cibercriminales y actores estatales, como los vinculados a China, Irán y Corea del Norte, para obtener acceso a cuentas protegidas [32].

2.1.2 ¿Por qué elegir Deep Learning sobre machine Learning?

Los sistemas de detección de intrusiones (IDS) tradicionales, incluidos los basados en machine learning, enfrentan desafíos como las altas tasas de falsos positivos, la complejidad de la ingeniería de características manual y el desequilibrio de clases

en los datos. El deep learning supera estas limitaciones al eliminar la necesidad de ingeniería de características, aprendiendo representaciones directamente de la información fundamental del tráfico de red. Esto permite a los modelos de deep learning identificar patrones de ataque conocidos y novedosos con mayor precisión y adaptabilidad, incluso al analizar la evolución de los patrones de ataque en el monitoreo de red en tiempo real [33].

Frente a la constante evolución del malware, los modelos de deep learning superan las limitaciones de los métodos tradicionales al aprender automáticamente características intrincadas de grandes conjuntos de datos, lo que les confiere una notable capacidad de adaptación a nuevas amenazas. Destacan en la extracción de detalles sutiles en muestras de malware, una tarea difícil para sistemas basados en reglas o firmas, lo que resulta en mayor precisión y menores tasas de falsos positivos. Además, ofrecen velocidad, eficiencia, escalabilidad y mejora continua, herramientas esenciales para la detección en tiempo real de amenazas emergentes en el ciberespacio [34].

2.1.3 Entorno de desarrollo del proyecto

El proyecto se desarrolla en un laboratorio virtual aislado, instalado en un equipo local propio, sin depender de infraestructura empresarial. Esto se debe a que la reproducción de ataques de red en entornos reales podría implicar riesgos significativos, como exposición de datos o interrupción de servicios críticos, afectando la seguridad de una organización [35].

Para la emulación de la topología de red se empleará VirtualBox, herramienta que permite crear y administrar entornos virtualizados con diferentes sistemas operativos, facilitando la simulación de escenarios de red de forma controlada y segura [35]. La captura del tráfico se realizará directamente desde las interfaces de red de las máquinas virtuales, generando archivos en formato PCAP, los cuales posteriormente serán procesados con CICFlowMeter para obtener métricas detalladas de los flujos de red [36]. Se utilizará una versión mejorada de CICFlowMeter que incorpora soporte para ICMP, precisión temporal en microsegundos y mejoras en la gestión de flujos TCP [37].

El conjunto de datos empleado será Improved CSE-CIC-IDS2018, una reconstrucción del dataset original que corrige errores de etiquetado y añade detalles sobre cada tipo de ataque [28]. Este dataset es de acceso público y contiene tráfico simulado, lo que asegura que no se manipulen datos reales de usuarios, garantizando así un entorno de prueba seguro.

El modelo de detección se implementará en Python utilizando algoritmos de Deep Learning como MLP y CNN. Los modelos entrenados se almacenarán en formato H5, junto con parámetros y preprocesadores en formato PKL para garantizar su reutilización e integración. Todo el sistema se implementará como una aplicación de línea de comandos, con opciones para cargar modelos, analizar capturas, procesar flujos y realizar detección en tiempo real [38].

2.2 Marco conceptual

2.2.1 Tráfico de red

El tráfico de red se refiere al conjunto de datos que se transmiten entre dispositivos conectados, e incluye actividades como la navegación web, la transferencia de archivos y la transmisión de contenido multimedia. Este flujo de información permite la comunicación efectiva dentro de entornos digitales y constituye un elemento esencial en la infraestructura de las redes actuales. En este contexto, la ciberseguridad desempeña un papel fundamental al proteger dicho tráfico frente a amenazas como accesos no autorizados, ataques maliciosos y violaciones a la confidencialidad e integridad de los datos. Para ello, se requiere de mecanismos robustos de seguridad capaces de detectar, prevenir y mitigar ataques cibernéticos, especialmente en un entorno cada vez más expuesto por el crecimiento del internet de las cosas y la nube [39]. Este fundamento es clave en nuestro proyecto, ya que se analiza el tráfico de red para detectar y clasificar malware mediante técnicas de aprendizaje profundo.

El monitoreo del tráfico es también una herramienta crítica para la identificación de malware y otras actividades maliciosas. Los ataques informáticos suelen alterar los patrones normales de comunicación, generando anomalías que pueden ser detectadas mediante el análisis de protocolos como TCP, UDP o ICMP, y el rastreo

de direcciones IP asociadas a comportamientos sospechosos. El uso de visualizaciones de datos y modelos avanzados de inteligencia artificial ha permitido automatizar este proceso, mejorando la capacidad para reconocer amenazas en tiempo real. Esto ha dado lugar a soluciones más eficientes y adaptativas que permiten una defensa más rápida ante eventos de seguridad [39]. Esta capacidad de análisis y monitoreo automatizado es central en nuestra propuesta, donde se aplica un modelo de deep learning para identificar comportamientos maliciosos en el tráfico de red.

2.2.1.1 Flujo de red

El RFC 3917 define un flujo de tráfico IP se entiende como la agrupación de paquetes IP observados en un punto específico de la red dentro de un intervalo temporal determinado. Los paquetes agrupados en un mismo flujo comparten un conjunto de propiedades comunes, donde cada propiedad es el resultado de aplicar una función a los valores de uno o más campos del encabezado del paquete, por ejemplo, dirección IP de destino, del encabezado de transporte, por ejemplo, número de puerto de destino, o del encabezado de aplicación (por ejemplo, campos del encabezado RTP). También pueden incluirse características del paquete en sí por ejemplo, número de etiquetas MPLS o campos derivados del tratamiento del paquete, por ejemplo, dirección IP del próximo salto, interfaz de salida [40].

Esta definición es muy amplia, abarcando desde un flujo que contiene todos los paquetes observados en una interfaz de red hasta un flujo que consta de un solo paquete entre dos aplicaciones con un número de secuencia específico. Para distinguir los flujos, el proceso de medición (Metering Process) debe ser capaz de separar los flujos por la interfaz de entrada o salida, la dirección IP de origen y destino (incluyendo prefijos), y el tipo de protocolo IP. También debe ser capaz de separar flujos por los números de puerto de origen y destino TCP/UDP [40].

Los flujos de red son vitales para la detección de malware y anomalías al permitir distinguir con éxito el tráfico normal del malicioso. Ante la creciente sofisticación de los ciberataques, la aplicación de algoritmos de aprendizaje automático sobre datos de tráfico como NetFlow es crucial para clasificar con alta precisión las

anomalías, logrando resultados de hasta 97.68% en la detección. Esta capacidad predictiva es fundamental para identificar comportamientos desconocidos y tomar medidas oportunas para detener el tráfico no deseado, mejorando significativamente la seguridad de los sistemas de TI [41].

2.2.2 Malware

El malware es un tipo de software malicioso diseñado para alterar el funcionamiento normal de los sistemas, comprometer la integridad de los datos o acceder sin autorización a recursos informáticos. En el ámbito del tráfico de red, este tipo de amenaza se manifiesta a través de actividades anómalas que alteran los patrones habituales de comunicación, utilizando brechas de seguridad para propagarse entre dispositivos. Estas acciones pueden generar un aumento inusual en el volumen de tráfico, conexiones no autorizadas o la transmisión de información sensible sin el consentimiento del usuario, elementos que pueden ser detectados mediante técnicas de monitoreo y análisis del comportamiento de red [42]. Este conocimiento es fundamental para nuestro proyecto, que busca detectar estas amenazas mediante el análisis del tráfico generado por distintos tipos de malware.

La identificación de malware en entornos de red exige el uso de mecanismos especializados que permiten observar y evaluar el tráfico en tiempo real. Herramientas como sistemas de detección de intrusos, firewalls y soluciones de seguridad integradas son capaces de reconocer patrones sospechosos, bloquear accesos no autorizados y mitigar intentos de propagación. A su vez, la capacitación de los usuarios y administradores de red resulta fundamental para reducir los riesgos asociados al error humano, que continúa siendo una de las principales vías de infección [42]. En nuestro enfoque, se emplean modelos de deep learning para identificar tráfico asociado a estas amenazas, simulando escenarios realistas en entornos controlados.

2.2.2.1 Botnet

Las botnets son redes de dispositivos comprometidos y conectados a Internet que actúan de forma coordinada bajo el control de un atacante, a menudo sin el conocimiento de sus propietarios. Estos dispositivos, conocidos comúnmente como

"zombis", pueden ser utilizados para diversas actividades maliciosas que incluyen el envío masivo de mensajes no deseados, la distribución de software malicioso, la sustracción de datos sensibles o la ejecución de ataques que afectan la disponibilidad de servicios en línea. Uno de los elementos clave en el funcionamiento de una botnet es el canal de comando y control (C&C), que permite al operador enviar instrucciones remotas y coordinar el comportamiento de toda la red de bots. La sofisticación de estas redes ha aumentado significativamente, pasando de arquitecturas centralizadas fácilmente detectables a modelos descentralizados y resilientes que imitan el tráfico legítimo, dificultando su identificación [43]. Este conocimiento es relevante para nuestro proyecto, ya que el modelo propuesto debe ser capaz de detectar comportamientos asociados al control y coordinación del tráfico de red por parte de botnets.

En los últimos años, los mecanismos de comunicación utilizados por las botnets también han evolucionado. Se han incorporado técnicas que mezclan el tráfico malicioso con flujos normales de red, empleando protocolos cifrados o retrasos intencionales en las comunicaciones para evitar su detección. Estas características plantean desafíos importantes para las herramientas tradicionales de monitoreo y requieren enfoques más avanzados para diferenciar entre actividad legítima y tráfico malicioso disfrazado. Además, la variedad de estructuras de las botnets, junto con su capacidad de adaptación, hace que su análisis deba considerar patrones de comportamiento más amplios en lugar de depender exclusivamente de firmas conocidas o reglas estáticas [43]. Nuestro sistema de detección se orienta al reconocimiento de este tipo de tráfico mediante técnicas de aprendizaje profundo, buscando identificar patrones ocultos en la comunicación entre dispositivos comprometidos.

2.2.2.2 Infiltration

La infiltración en redes informáticas ha aumentado con el crecimiento del uso de computadoras e internet, lo que ha generado un entorno más vulnerable a ataques maliciosos. Los atacantes emplean métodos cada vez más sofisticados para acceder sin autorización a datos sensibles, aprovechando las debilidades en el diseño y la configuración de las redes. Estas acciones incluyen técnicas como la sobrecarga de

tráfico, el sondeo no autorizado y la propagación de software malicioso, lo cual representa una amenaza constante para empresas e instituciones. Frente a este panorama, los sistemas de detección de intrusiones (IDS) se vuelven esenciales para anticipar y mitigar actividades no deseadas, ya que analizan patrones anómalos y permiten identificar posibles intentos de acceso indebido, incluso antes de que se materialicen [44]. Este enfoque es fundamental para nuestro sistema de detección, que busca reconocer comportamientos maliciosos en el tráfico antes de que se comprometa la red.

El uso de herramientas de escaneo, como las que permiten mapear redes y detectar servicios activos, representa una técnica habitual tanto para administradores legítimos como para actores maliciosos. Estas herramientas envían paquetes a dispositivos conectados con el fin de identificar hosts, puertos abiertos, versiones de servicios y sistemas operativos, lo que ayuda a determinar posibles vulnerabilidades. Sin embargo, este tipo de escaneo también puede ser utilizado con fines ofensivos, ya que brinda a los atacantes una visión detallada de la infraestructura de red. Por ello, su uso debe limitarse a contextos autorizados, ya que realizar análisis en redes ajenas sin permiso constituye una violación legal [45]. Nuestro modelo considera la detección de patrones similares a los generados por estas técnicas de reconocimiento, integrando la identificación temprana de posibles infiltraciones.

2.2.2.3 Webattacks

Los ataques basados en la web son un método atractivo que permite a los atacantes engañar a las víctimas a través de sistemas y servicios web. Cuando estos ataques se dirigen directamente a un sitio web para causar daño, a menudo implican la inyección de código malintencionado en sitios legítimos, pero comprometidos. El propósito de estos ataques varía desde la obtención de ganancias financieras, el robo de datos o el chantaje con ransomware, hasta el daño a la reputación. En última instancia, pueden comprometer la confidencialidad y la integridad de los datos, así como afectar la disponibilidad de sitios web, aplicaciones e interfaces de programación de aplicaciones (API) [46].

2.2.2.3.1 Dvwa

DVWA (Damn Vulnerable Web App) es una aplicación web vulnerable desarrollada en PHP/MySQL, diseñada para ser intencionalmente insegura. Su objetivo principal es servir como una herramienta de asistencia para profesionales de la seguridad para probar sus habilidades y herramientas en un entorno legal y seguro. Además, busca ayudar a los desarrolladores web a comprender mejor los procesos de protección de aplicaciones web y es útil para la enseñanza y el aprendizaje de la seguridad de aplicaciones web en el ámbito educativo [47].

2.2.2.3.2 Fuerza bruta

Los ataques por fuerza bruta es un método de ciberataque que consiste en intentar, de manera sistemática y automatizada, todas las combinaciones posibles de contraseñas, claves o credenciales con el objetivo de obtener acceso no autorizado a sistemas, servicios o aplicaciones web. Este tipo de ataque se basa en el uso intensivo de recursos computacionales y herramientas especializadas que permiten realizar múltiples intentos en poco tiempo, aprovechando contraseñas débiles o prácticas de seguridad deficientes. En el contexto de las páginas web, los ataques de fuerza bruta suelen dirigirse a formularios de autenticación para vulnerar cuentas de usuario o paneles administrativos, pudiendo derivar en el robo de información sensible o la alteración de los sistemas comprometidos. [48].

2.2.3 Inteligencia artificial

2.2.3.1 Machine learning

Machine Learning (ML) es un subconjunto de la inteligencia artificial (IA) que dota a los sistemas de una gama de habilidades perceptuales y la capacidad de aprender y adaptarse a lo largo del tiempo, permite a las computadoras realizar tareas sin una programación explícita, confiando en patrones e instrucciones. Su objetivo es que las máquinas adquieran la capacidad de percibir, analizar datos complejos, e identificar patrones. Con esta información, mejoran su rendimiento, toman decisiones y generan predicciones.[49].

2.2.3.1 Deep learning

Deep Learning (DL) es un campo emergente de investigación dentro del Machine Learning (ML). Se basa en múltiples capas ocultas de redes neuronales artificiales. Esta metodología aplica transformaciones no lineales y abstracciones de alto nivel

sobre grandes bases de datos. Conocido inicialmente como aprendizaje jerárquico, considera clave el procesamiento no lineal en múltiples capas y el aprendizaje supervisado o no supervisado. Su objetivo es optimizar aplicaciones de ML, mejorando resultados y tiempos de procesamiento. Es altamente efectivo en reconocimiento de voz y procesamiento de imágenes digitales [50].

2.2.3.1.1 MLP

El Multilayer Perceptron (MLP) es el modelo más utilizado en aplicaciones de redes neuronales, empleando comúnmente el algoritmo de entrenamiento de retropropagación. Se trata de una variante del modelo Perceptron original propuesto por Rosenblatt en la década de 1950. Un MLP está constituido por una capa de entrada, una capa de salida y una o más capas ocultas intermedias, donde las neuronas se organizan en capas y las conexiones siempre se dirigen de capas inferiores a superiores, sin interconexión entre neuronas de la misma capa. El entrenamiento de un MLP implica el ajuste de los pesos de las conexiones para minimizar la diferencia entre la salida de la red y la salida deseada, un proceso que puede verse como la minimización de una función de error [51].

La importancia del MLP para la clasificación multiclase se evidencia en su amplia aplicación en reconocimiento de patrones y otros problemas de clasificación. Para clasificar patrones, el MLP toma las mediciones como entrada y proporciona la clasificación correcta como salida. La capa de salida del MLP contiene tantas neuronas como clases distintas posea el conjunto de datos, y la neurona que produce la señal de salida más alta determina la clase asignada al patrón. Optimizar la arquitectura del MLP, incluyendo el número de capas y neuronas, es fundamental para asegurar una buena generalización y precisión de clasificación [51].

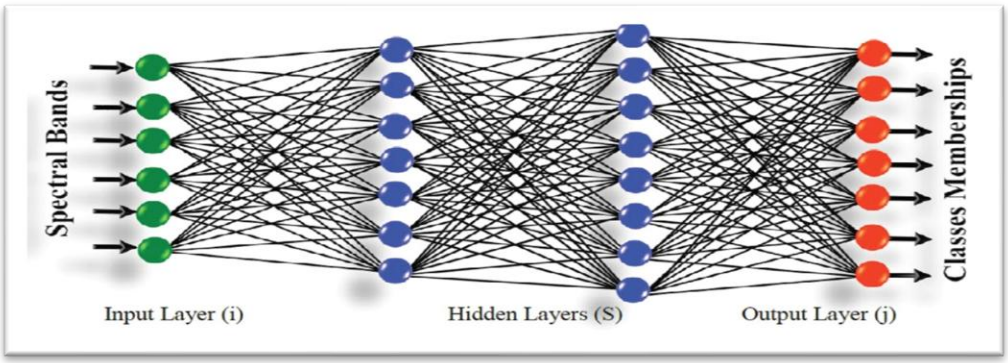


Ilustración 2 Arquitectura típica de MLP con DeepLearning [52]

2.2.3.1.2 LSTM

El Long Short-Term Memory (LSTM) es un tipo especializado de red neuronal recurrente (RNN), diseñado para superar las limitaciones de las RNN convencionales, como el problema de la desaparición o explosión de gradientes. Las RNN estándar tienen una capacidad limitada para establecer puentes entre un número determinado de pasos, ya que la información de los pasos anteriores decae, lo que provoca que las predicciones capturen solo dependencias a corto plazo. Para abordar esto, las redes LSTM introducen una unidad de memoria (estado de celda) y un mecanismo de compuertas, lo que les permite aprender dependencias a largo plazo [53].

La importancia del LSTM para la clasificación multiclase y otros problemas de datos secuenciales radica en su arquitectura única. El bloque LSTM contiene cuatro capas principales: el estado de la celda (memoria), una compuerta de entrada, una compuerta de salida y una compuerta de olvido. Estas compuertas permiten a la red recordar o descartar selectivamente información en una secuencia, asegurando que la información importante de pasos de tiempo anteriores no se pierda. Esta capacidad es fundamental para analizar datos secuenciales complejos [53].

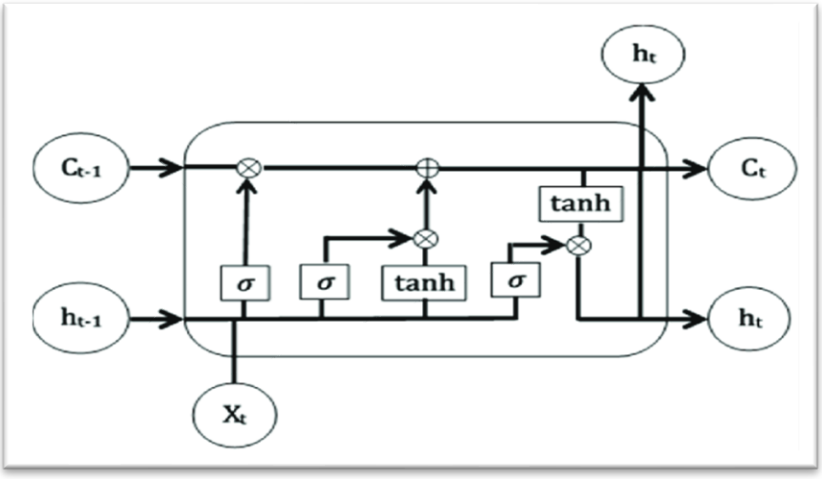


Ilustración 3.Arquitectura LSTM

Esta arquitectura LSTM controla el flujo de información mediante puertas (sigmoide y tanh) que deciden qué recordar, olvidar o actualizar del estado interno C_t, combinando la entrada X_t y el estado previo h_{t-1}. Así, permite aprender dependencias a largo plazo en secuencias sin que el gradiente se desvanezca [54]

2.2.4 Herramientas de análisis de tráfico

Las herramientas de Análisis de Tráfico de Red (NTA) son aplicaciones o plataformas diseñadas para monitorear, capturar, analizar y visualizar el tráfico de red, ya sea en tiempo real o históricamente. Son fundamentales para la seguridad de la red, la optimización del rendimiento y la resolución de problemas, permitiendo identificar amenazas, anomalías y patrones de tráfico. Su importancia para el aprendizaje automático (ML) reside en que facilitan la recolección, filtrado y extracción de características cruciales de los datos. Esto permite que los algoritmos de ML automaticen la clasificación de tráfico y la detección de actividades maliciosas o anómalas, mejorando la precisión, reduciendo falsos positivos y aumentando la escalabilidad del análisis [55].

2.2.4.1 Cicflowmeter

Es una herramienta de código abierto desarrollada por el Canadian Institute for Cybersecurity (CIC) para la generación de flujos de tráfico de red y la extracción de hasta 84 características. A partir de archivos PCAP, permite generar informes gráficos y exportar los resultados en formato CSV, facilitando el análisis posterior. Está desarrollada en Java y ofrece flexibilidad para integrarse en otros proyectos, permitiendo personalizar las métricas calculadas, agregar nuevas características y controlar el tiempo de expiración de los flujos [56].

Para este proyecto, se utilizará una versión mejorada de CICFlowMeter, disponible públicamente en un repositorio oficial. Esta versión corrige y amplía hasta 91 características clave del extractor original, como la correcta finalización de flujos TCP según el intercambio mutuo de paquetes FIN, el soporte para paquetes RST y la incorporación de nuevas métricas como duración total de la conexión, soporte para ICMP y mejoras en la precisión temporal de los registros. Estas actualizaciones permiten una caracterización más precisa del tráfico de red, especialmente útil en contextos donde se analizan múltiples tipos de ataques [57].

2.2.4.2 Scapy

Scapy es un programa Python interactivo y potente diseñado para la manipulación de paquetes de red. Su función principal es enviar, capturar, diseccionar y forjar

paquetes de una amplia gama de protocolos. Esto permite construir herramientas para sondeos, escaneos, rastreos, pruebas de unidad, ataques o descubrimiento de redes. Sus ventajas clave incluyen una flexibilidad inigualable para construir exactamente los paquetes deseados y la capacidad de decodificar información completa sin interpretaciones sesgadas, proporcionando datos crudos para un análisis profundo y sin restricciones. Scapy simplifica el diseño de paquetes y la reimplementación de herramientas de red en pocas líneas de código Python [58].

2.2.5 Dataset

Los Dataset son el pilar fundamental en el entrenamiento y desarrollo de modelos de inteligencia artificial (IA). Son los depósitos de información que alimentan a los algoritmos, permitiéndoles aprender, adaptarse y tomar decisiones inteligentes. Esencialmente, son la materia prima de la IA, y su calidad y diversidad son cruciales para el éxito o fracaso de los sistemas. Es imperativo que estos datos sean precisos, imparciales y reflejen la diversidad del mundo real, ya que un conjunto de datos sesgado puede generar resultados igualmente sesgados y decisiones injustas, socavando la confianza en estas tecnologías [59].

2.2.5.1 Improved cse-cic-ids2018

Proveniente del dataset CSE-CIC-IDS2018 el cual es un proyecto colaborativo entre el Communications Security Establishment (CSE) y el Canadian Institute for Cybersecurity (CIC). Su objetivo principal es generar una base de datos de referencia diversa y exhaustiva para analizar, probar y evaluar sistemas de detección de intrusiones, especialmente los detectores de anomalías basados en red. Este dataset aborda la escasez de conjuntos de datos reales y actualizados, siendo dinámico y reproducible para reflejar las tendencias actuales de tráfico y las intrusiones. Contiene siete escenarios de ataque distintos (como fuerza bruta, Heartbleed y ataques web), incluyendo tráfico de red capturado y logs del sistema de 470 máquinas, con más de 80 características extraídas usando CICFlowMeter-V3 [27].

El Improved CSE-CIC-IDS 2018 es una versión corregida y actualizada del conjunto de datos original, que aborda la prevalencia de errores en el etiquetado. Se considera mejorado porque resuelve problemas de contaminación de etiquetas y flujos mal clasificados presentes en la versión inicial. Específicamente, en Botnet

Ares se modificaron flujos y se solucionaron errores de segmentación TCP. Para Infiltración, se rectificó la significativa corrupción de etiquetas, separando sus componentes (descarga de Dropbox, escaneo NMAP, comunicación víctima-atacante) y corrigiendo clasificaciones erróneas. En los Ataques Web, se afinó la lógica de etiquetado para distinguir el tráfico malicioso real de los artefactos de inicio o flujos sin carga útil [60].

2.2.6 Ambiente simulado

Un ambiente simulado para seguridad de la información es un entorno controlado diseñado para que las personas adquieran, refuercen e incrementen sus conocimientos en seguridad informática. Su propósito es permitir la ejecución de pruebas, como tests de intrusión o el análisis del impacto de malware, sin riesgo de afectar servicios críticos o causar pérdidas de datos en sistemas reales. Esto ofrece una experiencia práctica sin limitaciones ni consecuencias indeseables [61].

2.2.6.2 Virtualbox

Oracle VirtualBox es una aplicación de virtualización multiplataforma que permite a tu computadora existente ejecutar múltiples sistemas operativos (SO) al mismo tiempo dentro de máquinas virtuales (VMs). Es una herramienta potente para pruebas y recuperación de desastres, ya que una VM puede congelarse, copiarse o respaldarse fácilmente. Esto posibilita experimentar libremente con un entorno informático; si algo sale mal, como una infección de malware, puedes revertir a un estado anterior guardado mediante instantáneas. Es ideal para correr software de un SO en otro sin reiniciar y para instalar configuraciones complejas pre-empaquetadas como "appliances" [62].

2.2.7 Python

Python, creado por Guido Van Rossum a finales de la década de 1980, es un lenguaje de programación potente, procedural, orientado a objetos y funcional. Se utiliza en una variedad de dominios de aplicación, incluyendo el desarrollo de software, desarrollo web, desarrollo de GUI de escritorio, educación y aplicaciones científicas. Es conocido por su simplicidad y robustez, lo que facilita su aprendizaje y comprensión. El código de Python es conciso, eficiente, comprensible y manejable, y el hecho de que no use corchetes ayuda a mantener el código corto [63].

Python es ampliamente utilizado y se ha convertido en uno de los lenguajes de programación preferidos en el campo de la Inteligencia Artificial (IA). Las razones de su popularidad en IA incluyen su sintaxis sencilla y versatilidad, que requiere muy poca codificación en comparación con otros lenguajes. Esto facilita las pruebas y permite a los desarrolladores concentrarse más en la lógica de la programación. Una de sus principales ventajas es que viene con bibliotecas integradas para casi todos los tipos de proyectos de IA. Ejemplos de estas bibliotecas incluyen NumPy, SciPy, Matplotlib, NLTK, SimpleAI, así como TensorFlow, Scikit-learn, Theano y Keras, que son cruciales para el aprendizaje automático y el aprendizaje profundo [63].

2.2.8 Librerías para modelos de ia

Las librerías de Python para entrenar modelos de IA son colecciones de código preescrito que los desarrolladores utilizan para realizar tareas estándar en la creación y análisis de modelos, evitando la necesidad de escribir todo desde cero. Python es un lenguaje muy popular en ciencia de datos y aprendizaje automático, ya que cuenta con numerosas bibliotecas para la manipulación de datos y la implementación de algoritmos. Entre estas se encuentran Scikit-learn, que provee algoritmos para aprendizaje supervisado y no supervisado como clasificación y regresión, y Keras, una API de alto nivel para construir y entrenar redes neuronales profundas. También se usan NumPy para operaciones numéricas y Pandas para manipulación de datos [64].

2.2.8.1 Tensorflow

TensorFlow es una librería de código abierto desarrollada por Google para la computación numérica y el aprendizaje automático. Ofrece un marco flexible para construir y desplegar modelos de aprendizaje profundo, con soporte para computación distribuida y despliegue en varias plataformas. Es crucial para el aprendizaje automático y profundo, destacándose por su escalabilidad y un extenso ecosistema de modelos preentrenados. Además, proporciona soporte multi-lenguaje, APIs avanzadas para el desarrollo de modelos de alto nivel y diferenciación automática [64].

2.2.8.2 Scikit-learn

Scikit-learn es una librería de aprendizaje automático escrita en Python que es cada vez más popular. Está diseñada para ser simple, eficiente y reutilizable, sirviendo tanto a expertos como a no expertos. Su propósito principal es proporcionar una colección de implementaciones eficientes de algoritmos de machine learning clásicos y consolidados. Esto incluye algoritmos de aprendizaje supervisado y no supervisado, además de herramientas para la evaluación, selección y preprocesamiento de datos. Se integra con el ecosistema científico de Python, utilizando herramientas como NumPy y SciPy [65].

2.3 Marco teórico.

2.3.1 Ciberseguridad moderna y detección automatizada de amenazas en redes.

La pericia forense en redes (NF) es fundamental para identificar evidencias de actividades ilegales. Sin embargo, las herramientas convencionales son costosas, requieren mucho tiempo y dependen de experiencia humana cara, siendo ineficientes en redes modernas. Ante ataques sofisticados como DDoS y ransomware exacerbados por vulnerabilidades del IoT, se propone un enfoque innovador utilizando inteligencia artificial (IA) y aprendizaje automático (ML) para acelerar investigaciones y potenciar el monitoreo automatizado sin intervención humana [66].

La IA es tecnología facilitadora clave en respuesta a incidentes, procesando datos para reconocer patrones que amplifican eficiencia forense. El ML permite predicción de ataques de día cero, detección de malware e identificación de anomalías, transformando la defensa de reactiva a proactiva. El marco de detección de amenazas cibernéticas (CTDF) basado en AI/DL demostró rendimiento prometedor con precisión del 88.2%, recall del 87.4% y F1-Score del 87.8%, destacando robustez en diferenciación de comportamientos normales y maliciosos. [66].

2.3.2 Deep learning como herramienta clave en la defensa cibernética.

El crecimiento exponencial de las comunicaciones y tecnologías como el Internet de las Cosas (IoT) ha incrementado la complejidad de los riesgos de seguridad, haciendo que las soluciones de ciberseguridad convencionales sean a menudo

ineficaces. El cibercrimen representa una amenaza significativa, con costos globales proyectados para alcanzar los 10.5 billones de dólares para 2025. En este contexto, la ciencia de datos, la inteligencia artificial (IA) y, específicamente, el aprendizaje profundo (Deep Learning - DL), emergen como opciones cruciales para mejorar los mecanismos de análisis y la protección de sistemas cibernéticos. Este estudio ofrece una visión general de la ciberseguridad desde la perspectiva de las redes neuronales y las técnicas de aprendizaje profundo [67].

El aprendizaje profundo es una subcategoría de la IA y el aprendizaje automático, con un rendimiento superior, especialmente al procesar grandes volúmenes de datos de seguridad. El documento analiza cuatro técnicas populares de redes neuronales y aprendizaje profundo aplicables a la ciberseguridad: el Perceptrón Multicapa (MLP) para tareas como la detección de intrusiones y análisis de malware; las Redes Neuronales Convolucionales (CNN), ideales para detección automática de características importantes en intrusiones o malware; las Redes Neuronales Recurrentes de Memoria a Corto Plazo (LSTM-RNN), adecuadas para el análisis de datos secuenciales y predicciones en series temporales; y el Aprendizaje Profundo por Transferencia (DTL o Deep TL), que resuelve problemas de insuficiencia de datos de entrenamiento y mejora la precisión [67].

Estas técnicas son altamente aplicables en diversos problemas de ciberseguridad, incluyendo la detección de intrusiones, identificación de malware o botnets, phishing, predicción de ciberataques y detección de anomalías. Sin embargo, la efectividad de estas soluciones basadas en DL depende críticamente de la calidad y las características de los datos de seguridad, así como de la selección adecuada del algoritmo. La investigación subraya la importancia de la recopilación y preprocesamiento de datos de alta calidad para el entrenamiento de modelos, sirviendo como guía para profesionales y académicos en el desarrollo de soluciones inteligentes y automatizadas en seguridad cibernética [67].

2.3.3 Deepmal: modelo de deep learning para detección y clasificación multiclase de malware en red

El artículo presenta DeepMAL, un modelo de aprendizaje profundo diseñado para la detección y clasificación de tráfico de malware utilizando datos de red en bruto. A través de un enfoque innovador que evita la necesidad de características

elaboradas por expertos, DeepMAL logra una alta precisión en la detección de malware, superando el 97% de tasa de detección con menos del 1% de falsos positivos. Los experimentos se realizaron utilizando conjuntos de datos de tráfico de red, lo que demuestra la capacidad del modelo para aprender patrones complejos directamente de los datos sin procesar [68].

Además, el estudio amplía el modelo de clasificación binaria a una versión multiclase que no solo detecta la presencia de malware, sino que también clasifica distintos tipos de amenazas. Los resultados obtenidos indican que los modelos de DeepMAL superan en rendimiento a otros enfoques tradicionales, como los basados en bosques aleatorios (RF), así como a otros métodos de aprendizaje automático superficial comúnmente empleados en la literatura de seguridad de redes. Esta capacidad de clasificar diferentes tipos de malware refuerza la versatilidad del modelo en entornos más complejos y dinámicos [68].

A pesar de la efectividad de los modelos tradicionales de aprendizaje automático, DeepMAL destaca por su capacidad para capturar las estadísticas subyacentes del tráfico malicioso sin la intervención de características manuales. Este avance subraya la importancia creciente de los modelos de aprendizaje profundo en el campo de la ciberseguridad, ya que ofrecen una solución mucho más flexible y adaptativa frente a las amenazas emergentes. La investigación sugiere que DeepMAL no solo representa una opción viable, sino un paso significativo hacia la mejora de la detección de malware en redes dinámicas, ofreciendo un enfoque que evoluciona de manera eficiente en respuesta a nuevas amenazas [68].

2.3.4 Modelos de deep learning para detección de anomalías en tráfico de red

Este artículo presenta una revisión exhaustiva de metodologías de aprendizaje profundo aplicadas a la detección de anomalías en redes, analizando arquitecturas como CNN, RNN, MLP, Autoencoders, GANs, LSTM y otras. Se abordan diversos tipos de ataques cibernéticos, desde Denegación de Servicio (DoS) hasta ransomware y botnets, utilizando datasets variados como KDD CUP'99, CICIDS-2017, NSL-KDD y CICDDoS2019, que ofrecen diversidad en dominios y tipos de ataques. La investigación también examina el contexto de aplicación de estas técnicas, evaluando su rendimiento y eficiencia en escenarios reales y simulados, proporcionando un marco integral para futuros desarrollos en ciberseguridad [69].

Para evaluar estos modelos, se emplean métricas como Precisión, Recall, F1-Score y coeficientes de correlación. Entre los principales desafíos se destacan el desequilibrio de datos, la alta dimensionalidad, la necesidad de escalabilidad y la falta de interpretabilidad. Las tendencias futuras sugieren el desarrollo de modelos híbridos, mejor interpretación, supervisión débil y estandarización de datasets para mejorar la eficacia y adaptabilidad en la detección de anomalías. Además, se enfatiza la importancia de optimizar el hardware y las técnicas de entrenamiento para enfrentar el creciente volumen de datos en redes modernas y garantizar una respuesta rápida ante nuevas amenazas [69].

2.4 Marco legal

2.4.1 Interceptación ilegal de datos

El Artículo 230 del Código Orgánico Integral Penal (COIP) del Ecuador establece que será sancionada con pena de tres a cinco años la persona que, sin orden judicial, intercepte, desvíe, grabe u observe datos informáticos, diseñe o distribuya programas maliciosos que induzcan a acceder a sitios distintos a los deseados, o copie y comercialice información de dispositivos electrónicos [70]. Este proyecto no realiza ninguna interceptación real, trabajando únicamente con tráfico y datos simulados en entornos controlados, garantizando el cumplimiento de la normativa vigente.

2.4.2 Replicación de malware

El artículo 232 del Código Orgánico Integral Penal (COIP) del Ecuador sanciona con pena de tres a cinco años a quien "diseñe, desarrolle, programe, adquiera, envíe, introduzca, ejecute, venda o distribuya programas informáticos maliciosos" o altere sistemas informáticos sin autorización [70]. Esta investigación no crea malware real, sino los utiliza para fines académicos, además se utilizará en un entorno controlado. Cualquier uso malicioso derivado de esta metodología contraviene lo establecido en el COIP y es rechazado categóricamente por los autores.

2.4.3 Uso de ambientes controlados

El artículo 234 del COIP establece que acceder sin autorización, en todo o en parte, a sistemas informáticos o telemáticos es un delito sancionado con prisión de tres a cinco años [70]. Este proyecto se ejecuta en entornos virtuales cerrados, utilizando

herramientas como Virtualbox para máquinas virtuales que no están conectadas a redes reales. Las direcciones IP utilizadas pertenecen exclusivamente al rango privado definido en la RFC 1918 [71], garantizando que no se afecte ninguna red de producción.

2.4.4 Manejo de direcciones ip y metadatos

Según la Ley Orgánica de Protección de Datos Personales (LOPDP), los datos personales son toda información relativa a una persona identificada o identificable (Art. 5), y su tratamiento debe ser limitado a lo estrictamente necesario (Art. 6) [72]. Este proyecto solo manipula información generada artificialmente, incluyendo direcciones IP privadas, puertos, protocolos, y metadatos temporales. Ninguno de estos datos corresponde a usuarios reales ni permite identificarlos.

2.4.5 Advertencia para replicación en entornos reales

Si se replica esta metodología en redes reales, se deben cumplir las obligaciones de la LOPDP: consentimiento informado (Art. 9), minimización de datos (Art. 6), finalidad clara (Art. 6), medidas de seguridad (Art. 44) y períodos de retención definidos (Art. 6). La falta de cumplimiento puede acarrear sanciones de hasta el 2 % de los ingresos anuales o USD 20,000, según el Art. 69 [72].

2.4.6 Conformidad ética e institucional

Este trabajo responde a fines legítimos de investigación científica. Se aplican principios de proporcionalidad, transparencia y responsabilidad ética. Se buscará la aprobación del comité institucional correspondiente. Además, la Constitución del Ecuador reconoce el derecho a la protección de datos personales y la privacidad como garantías fundamentales (Art. 66, numeral 19) [73].

2.5 Requerimientos

2.5.1 Requerimientos funcionales

Carga de modelos y archivos	
Código	Descripción
RF-1	El sistema debe cargar modelos de Deep Learning en formato .h5 desde la carpeta definida (recursosMLP/).
RF-2	Junto con el modelo, se deben cargar los archivos asociados: scaler.pkl, label_encoder.pkl y feature_columns.pkl.

Carga de modelos y archivos	
RF-3	El sistema debe aceptar archivos de tráfico en formato PCAP y CSV (CICFlowMeter) como entrada para el análisis.
RF-4	Se debe validar que los archivos proporcionados cumplan con la estructura y columnas necesarias antes de ser procesados.
RF-5	En caso de error de carga o formato inválido, el sistema debe mostrar un mensaje claro en la consola sin interrumpir la ejecución.

Tabla 3. Requerimientos Funcionales - Cargar modelos y archivos

Preprocesamiento de datos	
Código	Descripción
RF-6	El sistema debe detectar si un archivo CSV es demasiado grande para la memoria disponible y, en ese caso, procesarlo en chunks (porciones) para evitar errores por falta de memoria.
RF-7	Los archivos PCAP deben ser ordenados previamente, ya sea con Scapy (modo lento) o Wireshark (modo rápido).
RF-8	Una vez ordenados, los archivos PCAP deben convertirse a formato CSV mediante CICFlowMeter.
RF-9	El sistema debe rechazar archivos CSV que no provengan de CICFlowMeter o que no contengan las columnas esperadas.
RF-10	El sistema debe validar que los valores de las características numéricas no contengan datos no válidos (NaN, infinitos o strings) antes de la predicción.
RF-11	El sistema debe aplicar automáticamente el escalado definido en scaler.pkl.
RF-12	El sistema debe verificar que las columnas procesadas coincidan exactamente con las esperadas por feature_columns.pkl.

Tabla 4 Requerimientos Funcionales - Preprocesamiento de datos

Análisis de tráfico	
Código	Descripción
RF-13	El sistema debe permitir realizar análisis de tráfico con archivos PCAP o con CSV en el formato requerido.
RF-14	El sistema permitía al usuario seleccionar la interfaz de red que se usará para capturar tráfico.
RF-15	El sistema debe permitir realizar análisis después de tráfico ya sea con la herramienta Scapy o Cicflowmeter proporcionadas por el usuario
RF-16	El análisis debe incluir la clasificación del tráfico en las clases reconocidas por el modelo (benign, bot, infiltration, webattacks).
RF-17	El sistema debe mostrar en consola un top 10 de registros analizados, incluyendo IPs, protocolos, puertos y la clase de captura de flujo.

Tabla 5. Requerimientos Funcionales - análisis de trafico

Generación y almacenamiento de resultados	
Código	Descripción
RF-18	El sistema debe permitir guardar un archivo CSV con las etiquetas de predicción añadidas.
RF-19	El sistema debe generar un informe en formato PDF con estadísticas y gráficos del análisis realizado.
RF-20	Todos los resultados (CSV y PDF) deben almacenarse en la carpeta resultados/, organizada por subcarpetas con el nombre del archivo analizado.
RF-21	El sistema debe preguntar al usuario si desea generar y guardar cada resultado, evitando crear archivos innecesarios.
RF-22	El sistema debe almacenar automáticamente las capturas procesadas por CICFlowMeter en la carpeta captura_cicflowmeter_flujos/, generando subcarpetas con nombre de fecha y hora, y guardando los archivos en formato .csv.

Generación y almacenamiento de resultados	
RF-23	El sistema debe almacenar automáticamente las capturas obtenidas con Scapy en la carpeta capturas_scapy_pcap/, generando subcarpetas con nombre de fecha y hora, que contengan tanto el archivo .pcap como su respectivo .csv convertido por CICFlowMeter.

Tabla 6 Requerimientos Funcionales - Generación y almacenamiento de resultados

Interacción con el usuario	
Código	Descripción
RF-24	El sistema debe proporcionar un menú principal en modo consola con opciones numeradas para facilitar la navegación.
RF-25	El sistema debe mostrar un menú de ayuda que describa el propósito de cada opción y su uso correcto.
RF-26	El sistema debe incluir una opción para que el usuario concluya su sesión de forma controlada, liberando los recursos utilizados.

Tabla 7. Requerimientos Funcionales - interacción con el usuario

Operativos	
Código	Descripción
RF-27	El sistema debe ejecutarse completamente en modo consola (CLI).
RF-28	Debe funcionar tanto en Windows 10/11
RF-29	Debe ser robusto ante errores: entradas inválidas deben generar mensajes claros y no colapsar la ejecución.
RF-30	El sistema debe operar en cualquier interfaz de red seleccionada por el usuario

Tabla 8 Requerimientos Operativos

2.5.2 Requerimientos técnicos

Configuración mínima	
Código	Descripción
RTM-1	Procesador: Intel Core i7 (8va generación) o equivalente
RTM-2	Memoria RAM: 8 GB
RTM-3	Almacenamiento: -5 GB libres

Configuración mínima	
RTM-4	Sistema Operativo: Windows 10
RTM-5	El sistema se distribuye como un archivo ejecutable (.exe) independiente, que incluye todas las dependencias necesarias

Tabla 9. Requerimientos Técnicos - Configuración Mínima

Configuración recomendada	
Código	Descripción
RTR-1	Procesador: Intel Core i7 (11va generación) o Ryzen 5 (7ma generación)
RTR-2	GPU: NVIDIA RTX 2050 o superior con soporte CUDA versión mínima 12.7 y TensorRT 8.2.2.1
RTR-3	Memoria RAM: 16 GB
RTR-4	Almacenamiento: 50 GB SSD
RTR-5	Sistema Operativo: Windows 11
RTR-6	Dependencias: mismas que en RTM-5

Tabla 10. Requerimientos Técnicos - Configuración Recomendada

2.6 Componente de la propuesta tecnológica

2.6.1 Fase 1: Comprensión del proyecto

La fase de comprensión del proyecto, correspondiente a la primera etapa de la metodología CRISP-DM, busca entender en profundidad los objetivos, contexto y factores clave que influyen en el éxito de la investigación. En este sentido, los antecedentes, expuestos en el apartado 1.1 Antecedentes, abordan la problemática del malware en redes y la necesidad de un sistema de detección y clasificación mediante Deep Learning; los objetivos del proyecto, definidos en el apartado 1.3

Objetivos del proyecto, establecen como meta principal el desarrollo de dicho sistema; la justificación y beneficios, descritos en el apartado 1.4 Justificación, resaltan su relevancia en el campo de la ciberseguridad; y los requerimientos y restricciones, detallados en el apartado 2.2 Requerimientos, precisan los recursos, limitaciones y condiciones técnicas que orientan su desarrollo.

2.6.1.1 Criterios de éxito del proyecto

Las métricas de evaluación de modelos de clasificación, como Precisión, Exactitud, Recall, F1-Score, y la Tasa de Falsos Positivos, son fundamentales para comprender y analizar el rendimiento de un modelo. Estas métricas proporcionan una visión integral de la capacidad del modelo para hacer predicciones correctas, considerando tanto los verdaderos positivos como los errores cometidos. La precisión, al centrarse en la proporción de verdaderos positivos entre los casos predichos como positivos, es crucial para evaluar la calidad de las predicciones del modelo, especialmente en situaciones donde el costo de los falsos positivos es significativo. De manera similar, la exactitud ofrece una medida global del desempeño del modelo, evaluando qué tan bien clasifica las instancias en todas las categorías posibles [74].

Por otro lado, el recall es particularmente importante cuando se busca maximizar la detección de casos positivos, incluso a costa de cometer algunos falsos positivos. Esta métrica resulta esencial en contextos donde es crítico identificar todos los casos positivos. El F1-Score, al combinar precisión y recall, proporciona un equilibrio entre la capacidad del modelo para hacer predicciones correctas y su habilidad para encontrar todos los casos positivos, lo que lo convierte en una métrica ideal cuando se busca una evaluación más completa del rendimiento del modelo [74].

2.6.1.2 Evaluar la situación

El objetivo de este punto es realizar un análisis detallado de la situación actual del proyecto, considerando todos los recursos disponibles, restricciones, suposiciones y otros factores que influirán en el desarrollo del plan de análisis de datos y el alcance del proyecto. En esta fase, se deben profundizar los aspectos clave que impactarán la ejecución efectiva del proyecto, para poder definir un plan claro y viable [29].

2.6.1.2.1 Inventario de Recursos

Recursos de Software	
Recurso	Descripción
Anaconda	Herramienta para gestionar entornos de Python.
TensorFlow	Framework utilizados para entrenar modelos de Deep Learning.
VirtualBox	Plataforma de virtualización utilizada para crear y administrar las máquinas virtuales del entorno de pruebas.
Google Colab	Plataforma en la nube para entrenamiento con mayor capacidad de cómputo.
Principales Librerías Python	TensorFlow 2.10, Scikit-learn 1.6, numpy 1.26.4, Pandas 2.2.3, Joblib 1.4.2, ReportLab 4.4.3, CICFlowMeter v4.0, Scapy 2.6.1

Tabla 11 Recursos de software iniciales

Recursos de Hardware	
Recurso	Especificación
Marca	HP
Modelo	Victus 15
Procesador	AMD Ryzen™ 5 7535HS
Memoria RAM	16 GB DDR5
Almacenamiento	512 GB SSD M.2
GPU	NVIDIA RTX 2050
Sistema Operativo	Windows 11

Tabla 12 Recursos de Hardware

2.6.1.2.2 Requisitos, Suposiciones y Restricciones

Requisitos del proyecto	
Requisito	Descripción
Cronograma	El proyecto debe completarse en los tiempos establecidos.
Precisión global	El modelo debe alcanzar ≥95% en detección de tráfico malicioso.
Clasificación	Precisión ≥90% en la identificación del tipo de malware.
Validación	El modelo se probará en un entorno simulado para garantizar seguridad.

Tabla 13 Requisitos del proyecto

Supuestos del proyecto	
Supuesto	Descripción
Representatividad del dataset	Se asume que el dataset Improved CSE-CIC-IDS2018 refleja patrones reales de malware en una red controlada.
Recursos locales	Se considera que GPU y RAM disponibles serán suficientes para la mayoría de los entrenamientos.
Alternativa en la nube	En caso de limitaciones, se utilizará Google Colab para complementar el procesamiento.

Tabla 14 Supuestos del proyecto

Restricciones del proyecto	
Restricción	Descripción
Hardware limitado	La capacidad de procesamiento de la laptop puede afectar la velocidad de entrenamiento.
Dependencia de la nube	En caso de requerir más rendimiento, se deberá recurrir a Google Colab.
Entorno controlado	Los resultados no reflejarán completamente un despliegue en producción, al probarse solo en un entorno simulado.

Tabla 15 Restricciones del proyecto

2.6.2 Fase 2: Comprensión de los Datos

2.6.2.1 Recopilación de Datos Iniciales

El presente trabajo utiliza como base el conjunto de datos Improved CSE-CIC-IDS2018, una versión corregida y reetiquetada del dataset original desarrollada a partir del estudio de Liu et al. (2022), en el cual se identificaron y solucionaron múltiples inconsistencias en la fase de etiquetado y generación de características. Esta versión mejorada mantiene la estructura del dataset original —capturas de tráfico en formato PCAP transformadas en flujos mediante la herramienta CICFlowMeter—, pero garantiza mayor coherencia en las etiquetas y precisión en las características extraídas. Los archivos se encuentran en formato CSV y contienen un amplio conjunto de atributos relacionados con el comportamiento de los paquetes.

En el marco de este proyecto se seleccionó un subconjunto específico del dataset, correspondiente a las clases vinculadas a la presencia de malware en la red. Concretamente, se incluyeron las etiquetas Botnet, Web Attacks e Infiltration, además de la clase Benign, que permite establecer una referencia del tráfico normal frente al tráfico malicioso. Esta selección responde al objetivo de centrar el análisis únicamente en escenarios que representen amenazas directas derivadas de software malicioso, descartando otros tipos de ataques de diferente naturaleza.

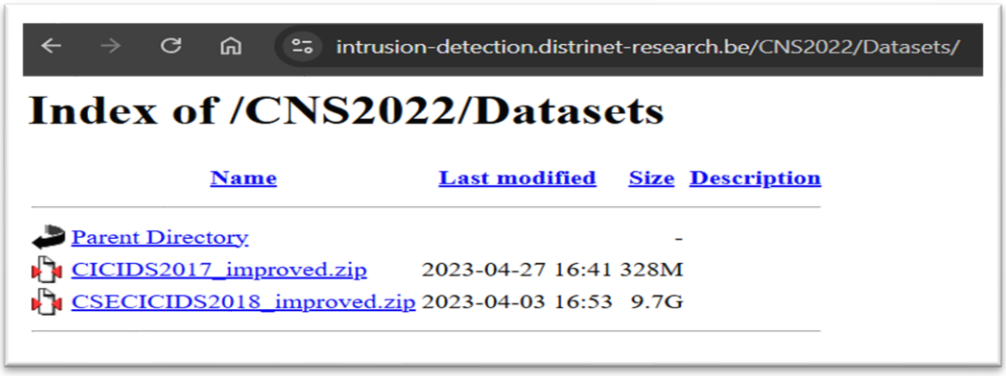


Ilustración 4 Página para descargar el dataset proporcionada por los autores.

El dataset fue descargado desde la página oficial de los autores, tal como se muestra en la Ilustración 4, en formato comprimido (.zip) y posteriormente almacenado en directorios locales para su procesamiento. A continuación, se seleccionaron los

archivos en formato CSV correspondientes a las etiquetas previamente mencionadas, aprovechando que dichos archivos se encuentran organizados de manera separada según los diferentes tipos de ataques. Los archivos seleccionados se muestran a continuación:

Nombre	Tamaño
Friday-02-03-2018.csv	3.39 GB
Friday-23-02-2018.csv	3.14 GB
Thursday-01-03-2018.csv	3.54 GB
Thursday-22-02-2018.csv	3.27 GB
Wednesday-28-02-2018.csv	3.57 GB

Tabla 16 Nombre de los archivos originales y su tamaño que componen el dataset

Uno de los principales inconvenientes destaca el elevado consumo de memoria RAM al cargar varios archivos de forma simultánea, la marcada desproporción entre el número de instancias benignas y maliciosas.

Para mitigar estos problemas se plantearon varias estrategias. En primer lugar, se optó por procesar los datos de manera secuencial, cargando archivo por archivo en lugar de realizar cargas masivas que excedan la capacidad de la máquina disponible, equipada con 16 GB de memoria RAM. En segundo lugar, se empleó la librería Polars como alternativa a pandas, dado que ofrece un manejo más eficiente de grandes volúmenes de datos en memoria. Asimismo, se aplicó un muestreo controlado de la clase Benign, limitándola a un número representativo de instancias con el fin de reducir el desbalance.

2.6.2.2 Descripción de los Datos

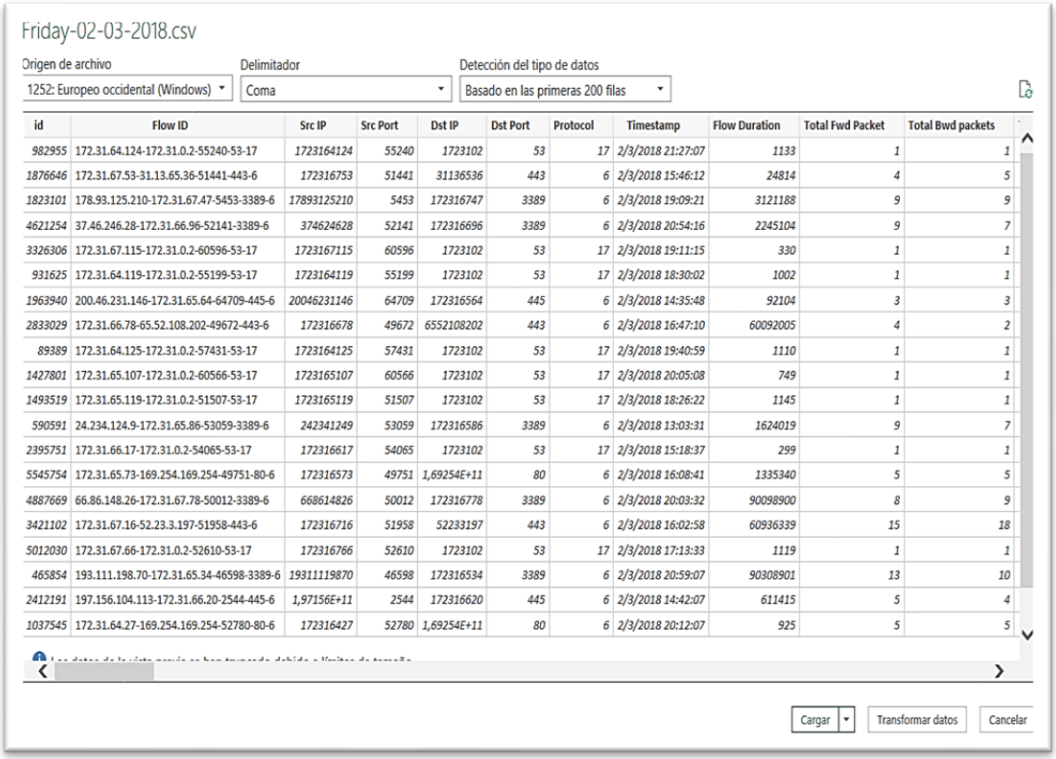
El subconjunto seleccionado del improved CSE-CIC-IDS2018 está conformado por archivos en formato CSV que representan flujos de red generados a partir de capturas PCAP mediante la herramienta CICFlowMeter. En total, el volumen de datos supera los seis millones de registros, aunque la distribución por clase es altamente desigual, con una clara predominancia del tráfico benigno frente a las instancias de ataques. La siguiente tabla muestra la distribución de registros por etiqueta:

Etiqueta	Conteo
BENIGN	31,245,820
Botnet Ares	142,921
Botnet Ares - Attempted	262
Infiltration - Communication Victim Attacker	204
Infiltration - Dropbox Download	85
Infiltration - Dropbox Download - Attempted	28
Infiltration - NMAP Portscan	89,374
Web Attack - Brute Force	131
Web Attack - Brute Force - Attempted	137
Web Attack - SQL	39
Web Attack - SQL - Attempted	14
Web Attack - XSS	113
Web Attack - XSS - Attempted	4
Total, de Registros	31,479,412

Tabla 17 Distribución del dataset con las clases seleccionadas

Cada registro del dataset está compuesto por un total de 91 columnas. La tabla muestra la distribución de etiquetas en el subconjunto del Improved CSE-CIC-IDS2018 utilizado en este estudio. Se observa un predominio marcado del tráfico benigno, con más de 31.2 millones de instancias, en contraste con las clases de ataque, cuyo número de registros es considerablemente menor. Dentro de estas, la categoría Botnet Ares concentra aproximadamente 142 mil registros, seguida por Infiltration - NMAP Portscan con más de 89 mil instancias. Otras variantes de infiltración, como *Communication Victim-Attacker* y *Dropbox Download*, presentan cantidades reducidas que no superan los 300 registros. En el caso de los ataques web, como Brute Force, SQL Injection y XSS, su representación es mínima, con apenas unas 280 instancias. En conjunto, el volumen total asciende a

31,479,412 registros, lo que confirma un fuerte desbalance entre el tráfico normal y las diferentes categorías de ataque.



id	Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestamp	Flow Duration	Total Fwd Packet	Total Bwd packets
982955	172.31.64.124-172.31.0.2-55240-53-17	1723164124	55240	1723102	53	17	2/3/2018 21:27:07	1133	1	1
1876646	172.31.67.53-31.13.65.36-51441-443-6	172316753	51441	31136536	443	6	2/3/2018 15:46:12	24814	4	5
1823101	178.93.125.210-172.31.67.47-5453-3389-6	17893125210	5453	172316747	3389	6	2/3/2018 19:09:21	3121188	9	9
4621254	37.46.246.28-172.31.66.96-52141-3389-6	374624628	52141	172316696	3389	6	2/3/2018 20:54:16	2245104	9	7
3326306	172.31.67.115-172.31.0.2-60596-53-17	1723167115	60596	1723102	53	17	2/3/2018 19:11:15	330	1	1
931625	172.31.64.119-172.31.0.2-55199-53-17	1723164119	55199	1723102	53	17	2/3/2018 18:30:02	1002	1	1
1963940	200.46.231.146-172.31.65.64-64709-445-6	20046231146	64709	172316564	445	6	2/3/2018 14:35:48	92104	3	3
2833029	172.31.66.78-65.52.108.202-49672-443-6	172316678	49672	6552108202	443	6	2/3/2018 16:47:10	60092005	4	2
89389	172.31.64.125-172.31.0.2-57431-53-17	1723164125	57431	1723102	53	17	2/3/2018 19:40:59	1110	1	1
1427801	172.31.65.107-172.31.0.2-60566-53-17	1723165107	60566	1723102	53	17	2/3/2018 20:05:08	749	1	1
1493519	172.31.65.119-172.31.0.2-51507-53-17	1723165119	51507	1723102	53	17	2/3/2018 18:26:22	1145	1	1
590591	24.234.124.9-172.31.65.86-53059-3389-6	242341249	53059	172316586	3389	6	2/3/2018 13:03:31	1624019	9	7
2395751	172.31.66.17-172.31.0.2-54065-53-17	172316617	54065	1723102	53	17	2/3/2018 15:18:37	299	1	1
5545754	172.31.65.73-169.254.169.254-49751-80-6	172316573	49751	1,69254E+11	80	6	2/3/2018 16:08:41	1335340	5	5
4887669	66.86.148.26-172.31.67.78-50012-3389-6	668614826	50012	172316778	3389	6	2/3/2018 20:03:32	90098900	8	9
3421102	172.31.67.16-52.23.3.197-51958-443-6	172316716	51958	52233197	443	6	2/3/2018 16:02:58	60936339	15	18
5012030	172.31.67.66-172.31.0.2-52610-53-17	172316766	52610	1723102	53	17	2/3/2018 17:13:33	1119	1	1
465854	193.111.198.70-172.31.65.34-46598-3389-6	19311119870	46598	172316534	3389	6	2/3/2018 20:59:07	90308901	13	10
2412191	197.156.104.113-172.31.66.20-2544-445-6	1,97156E+11	2544	172316620	445	6	2/3/2018 14:42:07	611415	5	4
1037545	172.31.64.27-169.254.169.254-52780-80-6	172316427	52780	1,69254E+11	80	6	2/3/2018 20:12:07	925	5	5

Ilustración 5 Ejemplo representativo de las columnas y sus tipos de datos

La ilustración corresponde a una vista previa del archivo Friday-02-03-2018.csv, parte del dataset CSE-CIC-IDS2018, en donde se muestran los primeros registros de flujos de red capturados en formato tabular. Cada fila representa un flujo identificado por un Flow ID único e incluye atributos clave como las direcciones IP de origen y destino, los puertos asociados, el protocolo empleado (predominantemente TCP con valor 6, y en menor medida UDP con valor 17), la marca temporal (Timestamp) y métricas de tráfico tales como la duración del flujo, el número de paquetes enviados en dirección directa (Fwd) y en dirección inversa (Bwd).

Por otra parte, se identificaron atributos que, si bien son necesarios para la reconstrucción de los flujos en un entorno operativo real, no aportan valor en el proceso de clasificación automática; entre ellos destacan, Flow ID, *Timestamp y Dst Port, entre otros*, los cuales serán descartados en la fase de preprocesamiento con el fin de evitar la introducción de sesgos y reducir la dimensionalidad de los datos.

2.6.2.3 Verificar la calidad de datos

El dataset presenta un desbalance extremo entre clases, donde la categoría Benign concentra más de 31 millones de registros, mientras que las clases de ataque cuentan desde unos pocos cientos hasta poco más de 140,000 instancias El submuestreo o undersampling es una técnica que reduce el número de muestras de la clase mayoritaria en un conjunto de datos desequilibrado para equilibrarlo con la clase minoritaria. Se utiliza para mejorar el rendimiento del modelo al evitar que la clase mayoritaria sesgue los resultados. Aunque tiene la ventaja de ser más eficiente en términos de almacenamiento y tiempo de entrenamiento, puede generar pérdidas de información clave de la clase mayoritaria. [75].

El submuestreo aleatorio es una técnica que selecciona y elimina ejemplos de forma aleatoria de la clase mayoritaria, reduciendo su número en el conjunto de datos de entrenamiento. El objetivo es lograr una distribución de clases más equilibrada. Se utiliza para abordar el problema de los conjuntos de datos desequilibrados, donde el sesgo de la clase mayoritaria puede llevar a los a ignorar la clase minoritaria [76].

```python
import os
import polars as pl

# Carpetas
input_folder = r"C:/U Octavo Semestre/tesis/Preprocesamiento de datos dataset actualizado/DATASET ACTUALIZADO V2/"
output_folder = r"C:/U Octavo Semestre/tesis/Preprocesamiento de datos dataset actualizado/dataset_undersampling/"

os.makedirs(output_folder, exist_ok=True)

# Parámetros
limite_benign_total = 500_000
benign_por_archivo = 100_000
benign_acumulado = 0

archivos_csv = [f for f in os.listdir(input_folder) if f.endswith('.csv')]

for archivo in archivos_csv:
    print(f"Procesando {archivo}...")

    # Leer archivo completo (Eager)
    df = pl.read_csv(os.path.join(input_folder, archivo))

    # Separar BENIGN y resto
    benign_df = df.filter(pl.col("Label") == "BENIGN")
    resto_df = df.filter(pl.col("Label") != "BENIGN")

    # Limitar BENIGN a 60k por archivo, respetando límite global
    if benign_acumulado < limite_benign_total:
        remaining_global = limite_benign_total - benign_acumulado
        take = min(benign_por_archivo, remaining_global, benign_df.shape[0])
        # Selección aleatoria de registros de la clase BENIGN
        benign_keep = benign_df.sample(n=take, shuffle=True, seed=42)
        benign_acumulado += benign_keep.shape[0]
    else:
        benign_keep = pl.DataFrame()

    # Combinar BENIGN limitado + resto
    df_final = pl.concat([benign_keep, resto_df], how="diagonal_relaxed")

    # Guardar archivo procesado en la nueva carpeta
    output_path = os.path.join(output_folder, archivo)
    df_final.write_csv(output_path)
    print(f"Guardado en: {output_path}")
    print(f"BENIGN acumulado hasta ahora: {benign_acumulado}")
    print(f"Filas procesadas de este archivo: {df_final.shape[0]}")

    # Liberar memoria antes de procesar el siguiente archivo
    del df, benign_df, resto_df, benign_keep, df_final

    # Si se alcanzó el límite global de BENIGN, avisar
    if benign_acumulado >= limite_benign_total:
        print(" Límite global de BENIGN alcanzado.")
```

Ilustración 6 Código empleado para el submuestreo aleatorio

Se aplico submuestreo aleatorio y se redujo el número de registros benignos a un máximo de 500,000 en total y fueron 100,000 por archivo. Esta reducción se realizó para mitigar el desbalance extremo de clases y permitir que los ataques tengan mayor peso en el entrenamiento. Por tanto, no se requieren procedimientos de limpieza adicionales en este aspecto

Archivo	Total, de filas	NaN	Inf/-Inf	Duplicados
Friday-02-03-2018.csv	243,183	0	0.0	0
Friday-23-02-2018.csv	100,230	0	0.0	0
Thursday-01-03-2018.csv	139,847	0	0.0	0
Thursday-22-02-2018.csv	100,208	0	0.0	0
Wednesday-28-02-2018.csv	149,844	0	0.0	0

Tabla 18 Verificación de calidad de datos en archivos seleccionados del dataset CSE-CIC-IDS2018

La tabla presenta los resultados del análisis de calidad realizado sobre cinco archivos representativos del dataset CSE-CIC-IDS2018. Se observa que no existen valores nulos (NaN), infinitos (Inf/-Inf) ni filas duplicadas en los registros procesados, lo cual confirma que los datos se encuentran consistentes y completos en estas dimensiones. Cabe destacar que el dataset utilizado corresponde a una versión balanceada mediante submuestreo o undersampling aleatorio de la clase mayoritaria Benign.

Adicionalmente, se aplicarán técnicas de sobremuestreo en la siguiente fase de preprocesamiento como SMOTE (Synthetic Minority Over-sampling Technique) es una técnica de sobremuestreo utilizada para corregir el desbalance de clases en conjuntos de datos. En lugar de duplicar las muestras minoritarias, SMOTE genera nuevas muestras sintéticas mediante la interpolación de puntos cercanos en el espacio de características. Esta estrategia es útil cuando las clases minoritarias están infrarrepresentadas, lo que podría sesgar el modelo hacia la clase mayoritaria [77]. Esto incrementa la representatividad de ataques como los Web Attacks sin introducir duplicados exactos, ayudando al modelo a aprender patrones más diversos y generalizables.

Con el fin de reducir la complejidad y garantizar la coherencia en el análisis, se realizó un proceso de normalización de etiquetas. Las instancias con la denominación Attempted fueron reclasificadas dentro de la categoría Benign, dado que representan intentos sin ejecución efectiva del ataque. Esta decisión se respalda en la recomendación de los autores del dataset, quienes señalan que se traten todos los flujos 'Attempted' como Benign [78].

Asimismo, se unificaron las variantes de etiquetas relacionadas con Botnet Ares, Infiltration y Web Attacks en tres categorías principales: Bot, Infiltration y WebAttacks, respectivamente, simplificando así la representación de los distintos escenarios maliciosos. Finalmente, debido a su escasa representación con solo 39 registros, la clase Web Attack – SQL fue eliminada, al no aportar un volumen estadísticamente significativo para el entrenamiento del modelo.

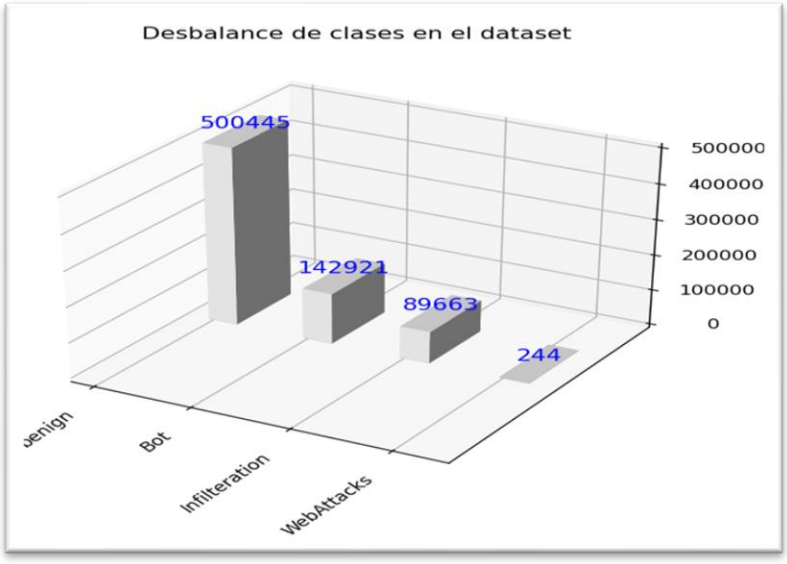


Ilustración 7 desbalance de clases en el dataset después de aplicar submuestreo

Para evaluar la disponibilidad de las métricas generadas por la herramienta CICFlowMeter mejorado, se realizó una comparación entre las columnas del dataset original mejorado de CSE-CIC-IDS2018 y los archivos CSV obtenidos directamente de CICFlowMeter al capturar los mismos flujos. El procedimiento consistió en cargar ambos conjuntos de datos y obtener las diferencias entre sus columnas, con el objetivo de identificar cuáles métricas presentes en el dataset original no eran reproducibles a partir de los flujos generados por la herramienta.

Columna del dataset	Presente en CICFlowMeter mejorado	Observaciones
Total TCP Flow Time	No	Métrica calculada por los autores, no reproducible directamente ya que no se proporciona información sobre esta.
Attempted Category	No	Columna auxiliar del dataset original para clasificación de intentos de ataque.
id	No	Columna de identificación interna del dataset original, no necesaria para análisis de flujos.

Tabla 19 Columnas presentes en el dataset original y ausentes en CICFlowMeter mejorado

El análisis mostró que algunas columnas del dataset original, incluyendo "Total TCP Flow Time", "id" y "Attempted Category", no aportan información directamente útil para el entrenamiento del modelo. Su inclusión podría introducir inconsistencias o sesgos, por lo que se decidió excluirlas en favor de métricas de flujo que representan características observables y medibles. En particular, la columna "id" es un identificador interno sin valor analítico, mientras que "Attempted Category" indica si un flujo fue solo un intento de ataque sin comportamiento malicioso efectivo; por ello, estos flujos deben ser tratados como benignos y no como una clase separada en el modelo

2.6.2.3.1 Eliminación de la métrica Total TCP Flow Time

La métrica "Total TCP Flow Time" fue eliminada del análisis, será eliminada de la selección de características y del entrenamiento del modelo debido a que, aunque aparece en el *improved CSE-CIC-IDS2018*, no es generada por la herramienta CICFlowMeter, ni siquiera en su versión mejorada. Esto indica que los valores fueron incorporados directamente por los autores del dataset a partir de los archivos PCAP originales, sin documentación oficial ni fórmula pública que explique su cálculo. Esta ausencia de definición clara convierte a la métrica en un valor dependiente de una implementación específica y no reproducible en otros escenarios, lo que limita su utilidad práctica y podría introducir inconsistencias o sesgos si se intentara replicarla manualmente.

La hipótesis planteada en este caso fue que *Total TCP Flow Time* no constituía una métrica independiente, sino una variable derivada de otras ya presentes en el flujo principalmente con *Flow Duration*, lo que la convertía en redundante y de bajo poder discriminante. Los análisis exploratorios realizados sobre los cinco archivos principales del dataset de entrenamiento confirmaron esta hipótesis.

Se verificó que, en la gran mayoría de los casos, la métrica "Total TCP Flow Time" cumple con la regla definida: ser igual a *Flow Duration* en conexiones TCP o igual a cero en conexiones no TCP. Sin embargo, un pequeño porcentaje de filas presenta valores que no cumplen esta regla, evidenciando inconsistencias que reflejan la lógica interna no documentada aplicada por los autores. A continuación, se presentan los resultados resumidos de la verificación sobre todos los archivos:

Archivo	Filas Totales	Cumplen la Regla	No Cumplen	% Cumplimiento
Friday-02-03-2018.csv	243,183	237,892	5,291	97.82 %
Friday-23-02-2018.csv	100,230	94,142	6,088	93.93 %
Thursday-01-03-2018.csv	139,847	134,193	5,654	95.96 %
Thursday-22-02-2018.csv	100,208	94,194	6,014	94.00 %
Wednesday-28-02-2018.csv	149,844	143,946	5,898	96.06 %

Tabla 20 Verificación de la métrica Total TCP Flow Time en archivos del dataset

Estos resultados muestran que la métrica es redundante en más del 93 % de los casos y ambigua en el resto, coincidiendo con otras métricas de duración de flujo ya disponibles como, Flow Duration. Dado que CICFlowMeter, la herramienta estándar para generación de flujos no produce esta columna, su inclusión en el modelo habría comprometido la aplicabilidad práctica del sistema. Por estas razones, se decidió eliminar la métrica Total TCP Flow Time y priorizar únicamente aquellas métricas consistentes, reproducibles y presentes en cualquier flujo de red, asegurando así la validez y confiabilidad del sistema de detección de malware en escenarios reales.

2.6.2.4 Explorar los datos

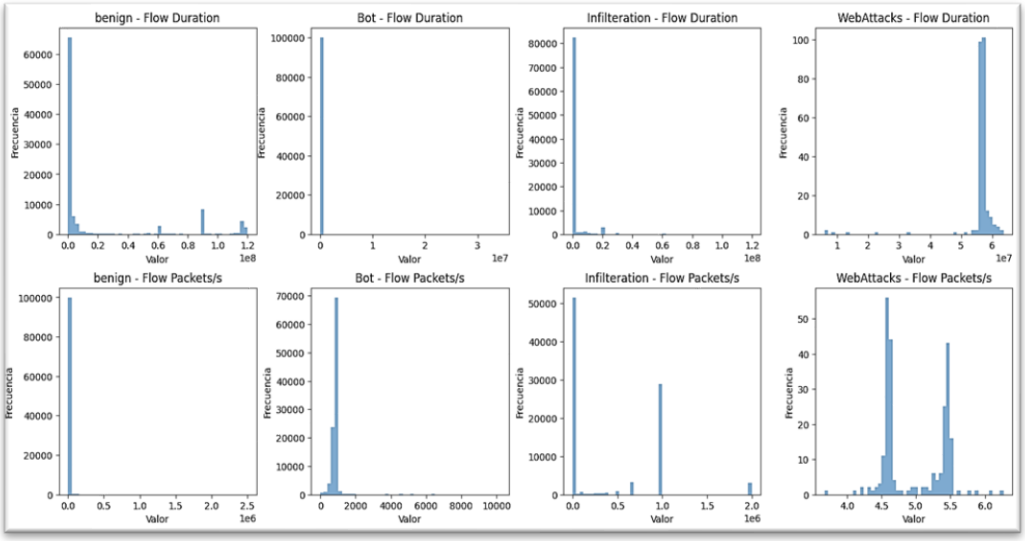


Ilustración 8 Histogramas de duración y paquetes de flujos de red por tipo de tráfico

En la ilustración 8 se observan los histogramas de las variables Flow Duration y Flow Packets/s, distribuidos por tipo de tráfico: benigno, Bot, Infiltration y Web Attacks. Se aprecia claramente que el tráfico benigno y los ataques de tipo Bot se concentran en valores bajos de duración y cantidad de paquetes por segundo. En contraste, los Web Attacks muestran una distribución más dispersa, con valores que se agrupan en rangos más altos y específicos. Por otro lado, la distribución del tráfico de Infiltration presenta un gran solapamiento con el tráfico benigno, lo que sugiere que podría ser más difícil de diferenciar utilizando estas variables.

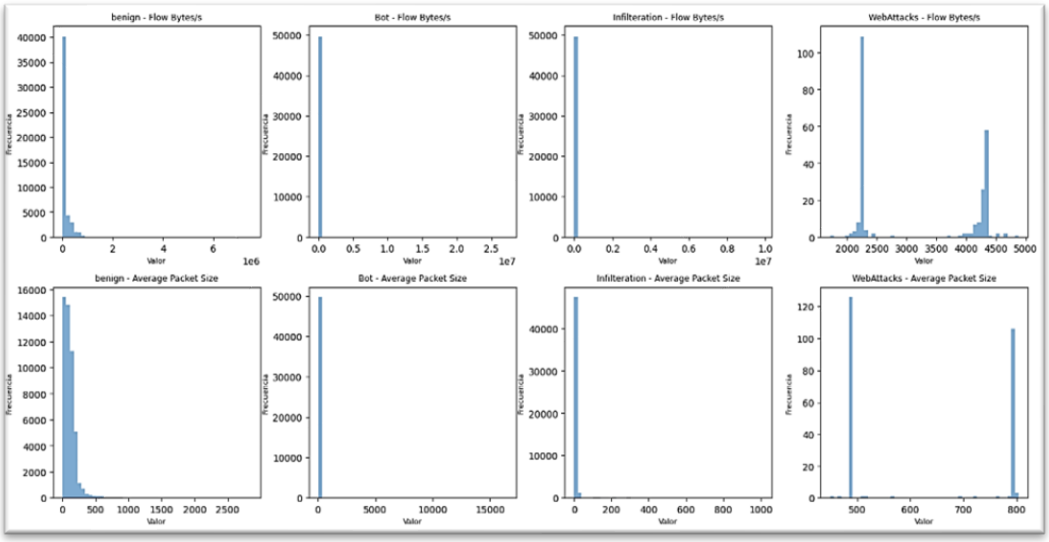


Ilustración 9 Histogramas de Tráfico de Red: Bytes y Tamaño de Paquetes por Tipo de Tráfico

En la ilustración 9 se presenta los histogramas de las variables Flow Bytes/s y Average Packet Size, desglosados por cuatro categorías de tráfico: benigno, Bot, Infiltration y Web Attacks. Se observa que tanto el tráfico benigno como los ataques de tipo Bot muestran una alta concentración en valores muy bajos para ambas variables. En contraste, los ataques Web se distinguen por valores más altos y claramente agrupados en rangos específicos. Aunque el tráfico de Infiltration se asemeja al tráfico benigno con una mayoría de valores bajos, presenta picos adicionales que indican un comportamiento anómalo.

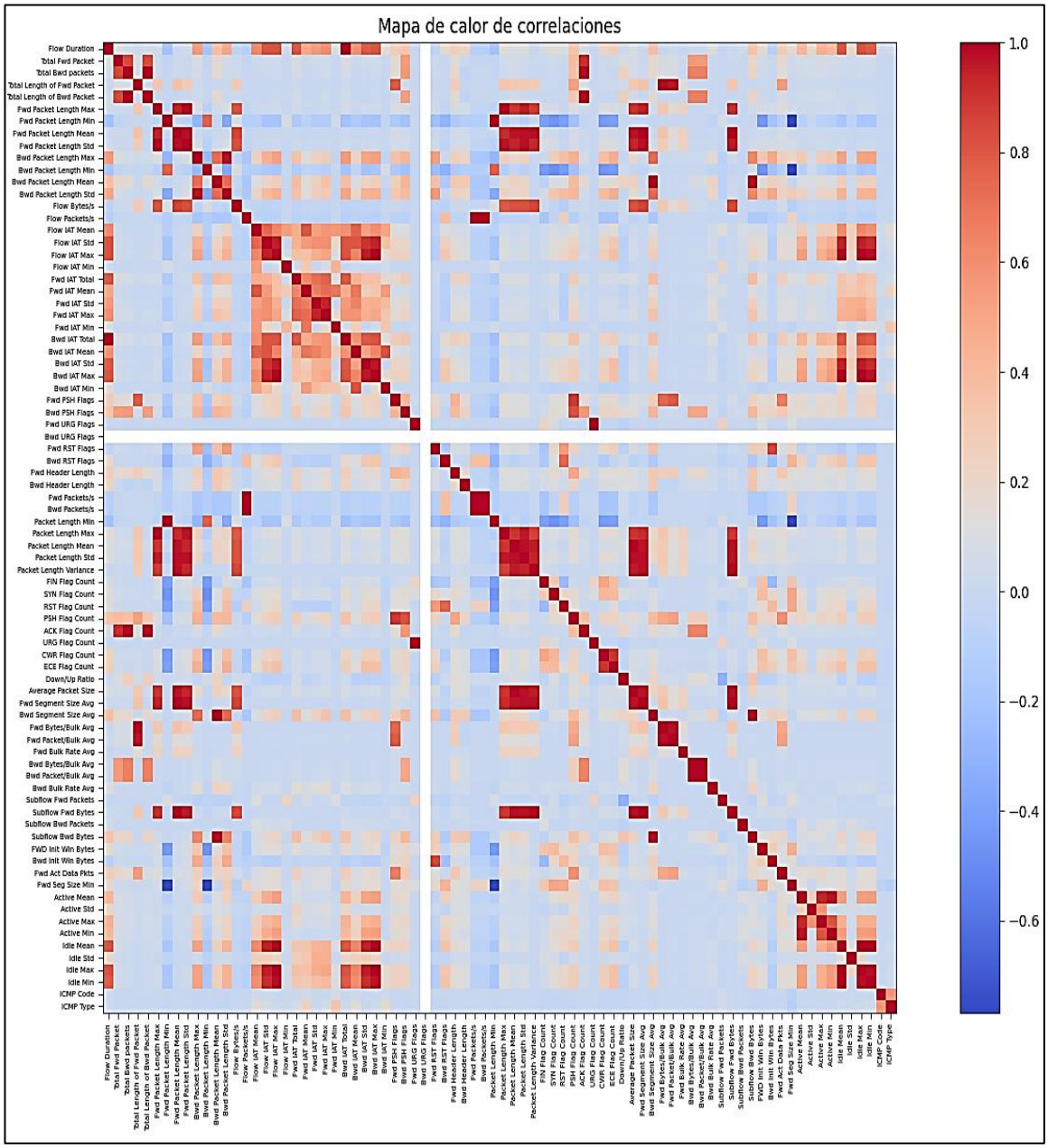


Ilustración 10 Mapa de calor de correlaciones de características de flujos de red

En la ilustración mapa de calor de correlaciones muestra la relación entre las diferentes características de los flujos de red. Las celdas rojas indican una correlación positiva fuerte, lo que significa que cuando una variable aumenta, la otra también lo hace. Las celdas azules indican una correlación negativa fuerte, donde el aumento de una variable se asocia con la disminución de la otra. Las celdas blancas o de color claro representan una correlación baja o nula. La diagonal, compuesta por celdas rojas, muestra que cada variable está perfectamente correlacionada consigo misma. La matriz revela bloques de características altamente correlacionadas, como las relativas a la duración de los flujos o al tamaño de los paquetes, lo cual es útil para comprender la redundancia en los datos y para la selección de características en modelos de aprendizaje automático.

A partir de estas exploraciones se pueden plantear suposiciones preliminares. Por ejemplo, que los ataques *Botnet* se caracterizan por la alta frecuencia de paquetes, mientras que los *Web Attacks* muestran un volumen elevado de bytes en poco tiempo. Estos resultados sugieren que, aunque en algunos casos como *Infiltration* existe solapamiento con el tráfico benigno, las visualizaciones iniciales respaldan la idea de que es posible identificar patrones diferenciadores entre las distintas clases de tráfico. Cabe destacar que en esta etapa únicamente se han mostrado algunos ejemplos con un subconjunto de variables (por ejemplo, *Flow Duration*, *Flow Packets/s*, *Flow Bytes/s* y *Average Packet Size*), mientras que el conjunto de datos completo contiene 91 atributos. Esto implica que podrían existir columnas más discriminantes para la clasificación, las cuales serán detectadas y aprovechadas en la fase posterior de procesamiento de datos. Por lo tanto, lo presentado en esta sección constituye un primer acercamiento ilustrativo y no exhaustivo al análisis exploratorio

2.6.3 Fase 3: Preparación de los datos

2.6.3.1 Selección de características

La selección de características es un proceso fundamental en la construcción de modelos, ya que permite reducir la cantidad de atributos a aquellos que realmente aportan valor al análisis. Al eliminar columnas irrelevantes, se reduce la complejidad y evita información redundante, mejorando su rendimiento y

reduciendo la complejidad. Esto no solo optimiza el tiempo y los recursos durante el entrenamiento, sino que también previene problemas como el sobreajuste, donde el modelo se adapta excesivamente a datos irrelevantes [79].

Además, este proceso mejora la eficiencia del modelo al disminuir la carga computacional y el espacio de almacenamiento necesarios. El uso de características relevantes asegura que el modelo se enfoque en patrones significativos y reduzca la influencia de datos ruidosos o redundantes. De esta manera, la selección de características es esencial para crear modelos más precisos y eficientes, adaptados a los datos más importantes para el análisis [79].

En primer lugar, se descartaron manualmente atributos que no aportaban valor predictivo, como Flow ID, Timestamp, Source IP, Destination IP y Port, los cuales representan identificadores o información contextual que podría introducir sesgos sin mejorar la detección. Posteriormente, se aplicaron tres métodos complementarios de selección automática de características:

2.6.3.1.1 Método univariado con ANOVA

El ANOVA de una vía (ANalysis Of VAriance) es una técnica estadística que permite analizar la variación en una variable de respuesta continua aleatoria, siendo esencial para comparar simultáneamente las medias de más de dos grupos. Fue desarrollado por Ronald Aylmer Fisher y es especialmente útil en estudios donde se desea comparar grupos sin incurrir en el error de realizar múltiples comparaciones pareadas, como ocurre al utilizar repetidamente el test t de Student. Este enfoque evitaría aumentar la probabilidad de encontrar diferencias significativas por azar, ya que el diseño experimental exige un análisis global de los datos. El ANOVA, al contrario, ofrece una prueba global que evalúa si existen diferencias significativas entre los grupos, basándose en la comparación de la variabilidad dentro de los grupos y entre los grupos [80].

La hipótesis nula en un ANOVA establece que no existen diferencias entre los niveles de la variable dependiente, lo que implicaría que las muestras provienen de una misma población. Para evaluar esta hipótesis, el ANOVA calcula la razón F, que compara la varianza entre los grupos con la varianza dentro de los grupos. Si no hay diferencias significativas, la razón F será cercana a 1. Si la varianza entre

los grupos es mayor que dentro de los grupos, la razón F aumentará y el valor de p disminuirá, lo que indica que hay diferencias significativas. El ANOVA de una vía es más robusto que los métodos univariados como el test t, y es esencial en estudios con más de dos grupos. Para aplicaciones más complejas, se recomienda la asesoría de un estadístico profesional para garantizar un diseño y análisis adecuado [80].

Se emplea ANOVA como filtro inicial para seleccionar características con mayor relevancia estadística respecto a la variable objetivo, mediante Python. Para ello se utilizó la clase SelectKBest con la función f_classif, que evalúa la relación entre cada atributo y la etiqueta de salida. El modelo calcula los valores F y selecciona automáticamente las variables con mayor poder discriminativo.

```python
# Método univariado
print(f"\n Selección Univariada (k={n_features})...")
selector_f = SelectKBest(score_func=f_classif, k=n_features)
selector_f.fit(X, y_encoded)
selected_f = X.columns[selector_f.get_support()].tolist()
```

Ilustración 11 Método univariado Anova en python

2.6.3.1.2 Método wrapper utilizando RFE

Recursive Feature Elimination (RFE) es un algoritmo de selección de características que elimina gradualmente las menos importantes para mejorar el rendimiento del modelo. Como método "wrapper", RFE depende de un algoritmo de aprendizaje, como Random Forest o SVM, para evaluar la importancia de los subconjuntos de características. Esta estrategia implica modificar el proceso de entrenamiento y utilizar la clasificación como mecanismo de evaluación, lo que le permite proporcionar resultados más precisos que los métodos de filtro. RFE es una técnica robusta y eficaz, ya que ajusta continuamente el modelo para identificar las variables más relevantes para la predicción [81].

Cuando se utiliza Random Forest como modelo de aprendizaje subyacente (RF-RFE), RFE se convierte en uno de los métodos más populares para la selección de características. Random Forest es ampliamente reconocido por su fiabilidad y eficiencia, y el método RF-RFE ha mostrado ser superior a otros enfoques, como el SVM-RFE, en algunos casos. Esta técnica es esencial en análisis de datos con miles

de variables, ya que reduce la dimensionalidad y mejora la calidad de los atributos, lo que se traduce en una mayor precisión en las tareas de clasificación y predicción. Además, la reducción de características no solo mejora el rendimiento del modelo, sino que también facilita su interpretación, ayudando a entender mejor los fenómenos estudiados [81].

Se aplicó Recursive Feature Elimination (RFE), una técnica wrapper que elimina de forma iterativa las características menos relevantes. En este caso se implementó con un estimador de Random Forest (RFE(estimator=rf_estimator)), lo que permitió identificar las variables que optimizan la capacidad predictiva del modelo. Su uso garantiza un modelo optimizado al conservar progresivamente las características más significativas para la clasificación

```python
# Método RFE
print(f" Recursive Feature Elimination (k={n_features})...")
rf_estimator = RandomForestClassifier(n_estimators=50, random_state=42, n_jobs=-1)
selector_rfe = RFE(estimator=rf_estimator, n_features_to_select=n_features)
selector_rfe.fit(X, y_encoded)
selected_rfe = X.columns[selector_rfe.get_support()].tolist()
```

Ilustración 12 Método RFE en python

2.6.3.1.3 Método ensemble utilizando Feature Importance con Random Forest.

La importancia de las características con Random Forest se refiere a cómo este clasificador de ensamble, compuesto por numerosos árboles de decisión, asigna un valor a cada variable para medir su contribución a la predicción. Este valor se calcula a través de la disminución promedio en la precisión de los árboles al eliminar un atributo. Dado que cada árbol utiliza un subconjunto aleatorio de datos y características, Random Forest es sensible incluso a atributos débilmente relevantes. Esto permite que el modelo capture relaciones complejas, incluso si ciertos atributos solo aparecen en un pequeño número de árboles, manteniendo la visibilidad de características marginales pero significativas [82].

Elegir las características más relevantes con Random Forest es fundamental para optimizar el análisis de datos, especialmente en conjuntos con miles de variables. Al emplear este modelo de ensamble, se facilita la selección de un subconjunto de atributos clave, lo que mejora la precisión de las predicciones, y también reduce la

dimensionalidad de los datos. Este enfoque ayuda a simplificar el procesamiento y mejora la calidad de los resultados. Además, Random Forest es computacionalmente eficiente, con pocos parámetros ajustables, y la evaluación de la importancia de las características no supone una sobrecarga significativa, convirtiéndolo en una herramienta potente para tareas como la identificación de genes asociados a enfermedades o problemas complejos [82].

Se aplicó la evaluación de importancia de características con Random Forest, donde se entrenó un clasificador con 100 árboles de decisión y se calcularon los pesos relativos de cada atributo en la predicción. Este enfoque permite captar relaciones no lineales y dependencias complejas en el conjunto de datos.

```python
# Importancia Random Forest
print(f" Feature Importance (k={n_features})...")
rf = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
rf.fit(X, y_encoded)
feature_importance = list(zip(X.columns, rf.feature_importances_))
feature_importance.sort(key=lambda x: x[1], reverse=True)
selected_importance = [feat for feat, imp in feature_importance[:n_features]]
```

Ilustración 13 Método importancia random forest en python

2.6.3.1.4 Métodos híbridos de selección

En el artículo publicado en 2023 por, Salah Yassen, Abdulrazzq y Mohammed propusieron un enfoque híbrido de selección de características denominado ANOVA-Recursive Feature Elimination (ANOVA-RFE), aplicado en la detección y clasificación de ataques en entornos IoT e IIoT [83]. Este método combina la capacidad de ANOVA para seleccionar características con mayor correlación con la variable objetivo y la estrategia iterativa de RFE, que elimina progresivamente atributos menos relevantes, reduciendo el riesgo de sobreajuste y mejorando la precisión de los modelos. Los resultados experimentales obtenidos en dicho estudio demostraron altos niveles de exactitud, alcanzando 100% y 99.96% de precisión al emplear únicamente las cinco características más relevantes en dos conjuntos de datos distintos, lo que valida la eficiencia del enfoque híbrido [83].

Siguiendo la idea del enfoque híbrido, en este trabajo se combinan los resultados de tres métodos de selección automática, ANOVA, RFE y Feature Importance con Random Forest. Para ello, se desarrolló un procedimiento en Python que contabiliza

la frecuencia con que cada característica es elegida y conserva únicamente aquellas presentes en al menos dos de los tres métodos, lo que garantiza un consenso en su relevancia. De esta forma, se obtiene un conjunto de atributos más compacto, robusto y confiable.

```python
selected_features_dict = {
    'univariate': selected_f,
    'rfe': selected_rfe,
    'importance': selected_importance
}

if method == 'hybrid':
    # Combinar métodos - características que aparecen en al menos 2 métodos
    all_features = set()
    for features in selected_features_dict.values():
        all_features.update(features)

    feature_counts = {}
    for features in selected_features_dict.values():
        for feat in features:
            feature_counts[feat] = feature_counts.get(feat, 0) + 1

    # Seleccionar características más consensuadas
    final_features = [feat for feat, count in feature_counts.items() if count >= 2]

    # Si no hay suficientes, tomar las más frecuentes
    if len(final_features) < n_features // 2:
        sorted_features = sorted(feature_counts.items(), key=lambda x: x[1], reverse=True)
        final_features = [feat for feat, count in sorted_features[:n_features]]

    self.selected_features = final_features
else:
    self.selected_features = selected_features_dict[method]

print(f"\n Características seleccionadas: {len(self.selected_features)}")
print(" Top 10 características:")
for i, feat in enumerate(self.selected_features[:10], 1):
    print(f"  {i:2d}. {feat}")

```

Ilustración 14 selección de características combinando ANOVA + RFE + Importancia RF en python

2.6.3.1.4.1 Características seleccionadas

Las características seleccionadas por el método fueron las siguientes:

N°	Característica	Descripción
1	Fwd Packet Length Std	Desviación estándar del tamaño de paquetes en dirección forward (fw_pkt_l_std) .
2	Fwd Packet Length Max	Tamaño máximo de paquete en dirección forward (fw_pkt_l_max) .
3	Total Length of Fwd Packet	Tamaño total acumulado de paquetes en dirección forward (tot_l_fw_pkt) .
4	Bwd Init Win Bytes	Número de bytes enviados en la ventana inicial TCP en dirección backward (bw_win_byt) .
5	Subflow Bwd Bytes	Bytes promedio en subflujo en dirección backward (subfl_bw_byt) .
6	Flow Packets/s	Número de paquetes por segundo en el flujo completo (fl_pkt_s) .

N°	Característica	Descripción
7	Bwd Packets/s	Número de paquetes por segundo en dirección backward (bw_pkt_s) .
8	Flow Bytes/s	Número de bytes por segundo en el flujo completo (fl_byt_s) .
9	Flow IAT Max	Intervalo de tiempo máximo entre paquetes consecutivos del flujo (fl_iat_max) .
10	Fwd IAT Mean	Intervalo de tiempo promedio entre paquetes forward consecutivos (fw_iat_avg) .
11	Average Packet Size	Tamaño promedio de paquete en el flujo completo (pkt_size_avg) .
12	Bwd Packet Length Mean	Tamaño promedio de paquetes en dirección backward (bw_pkt_l_avg) .
13	Flow IAT Mean	Intervalo de tiempo promedio entre paquetes consecutivos del flujo (fl_iat_avg) .
14	Packet Length Std	Desviación estándar del tamaño de paquetes en el flujo (pkt_len_std) .
15	Fwd Packets/s	Número de paquetes por segundo en dirección forward (fw_pkt_s) .
16	Packet Length Max	Tamaño máximo de paquete en el flujo completo (pkt_len_max) .
17	Bwd Segment Size Avg	Tamaño promedio de los segmentos TCP en dirección backward (bw_seg_avg) .
18	Total Length of Bwd Packet	Tamaño total acumulado de paquetes en dirección backward (tot_l_bw_pkt) .
19	Bwd IAT Total	Tiempo total acumulado entre paquetes enviados en dirección backward (bw_iat_tot) .
20	Fwd Seg Size Min	Tamaño mínimo de segmento TCP en dirección forward (fw_seg_min) .
21	Flow Duration	Tiempo total de duración del flujo de red (fl_dur) .
22	Fwd IAT Max	Intervalo de tiempo máximo entre paquetes forward consecutivos (fw_iat_max) .
23	Flow IAT Std	Desviación estándar del intervalo entre paquetes del flujo (fl_iat_std) .
24	Fwd IAT Std	Desviación estándar del intervalo entre paquetes forward (fw_iat_std) .
25	Bwd Packet Length Max	Tamaño máximo de paquete en dirección backward (bw_pkt_l_max) .

N°	Característica	Descripción
26	Fwd Packet Length Mean	Tamaño promedio de paquetes en dirección forward (fw_pkt_l_avg) .
27	FWD Init Win Bytes	Número de bytes enviados en la ventana inicial TCP en dirección forward (fw_win_byt) .
28	PSH Flag Count	Número de paquetes TCP con el flag PSH activado (psh_cnt) .
29	Bwd IAT Mean	Intervalo de tiempo promedio entre paquetes backward consecutivos (bw_iat_avg) .
30	ECE Flag Count	Número de paquetes TCP con el flag ECE (ECN-Echo) activado (ece_cnt) .
31	Fwd IAT Total	Tiempo total acumulado entre paquetes enviados en dirección forward (fw_iat_tot) .
32	Fwd Header Length	Bytes totales usados en las cabeceras de paquetes forward (fw_hdr_len) .
33	Packet Length Variance	Varianza del tamaño de paquetes en el flujo (pkt_len_va) .
34	Subflow Fwd Bytes	Bytes promedio en subflujo en dirección forward (subfl_fw_byt) .
35	Fwd Segment Size Avg	Tamaño promedio de segmentos TCP en dirección forward (fw_seg_avg) .
36	Bwd Packet Length Std	Desviación estándar del tamaño de paquetes en dirección backward (bw_pkt_l_std) .
37	Down/Up Ratio	Relación entre paquetes descendentes y ascendentes (down_up_ratio) .
38	SYN Flag Count	Número de paquetes TCP con el flag SYN activado (syn_cnt) .
39	Bwd PSH Flags	Número de paquetes con el flag PSH activado en dirección backward (bw_psh_flag) .
40	Packet Length Mean	Tamaño promedio de paquetes en el flujo completo (pkt_len_avg) .

Tabla 21 Conjunto híbrido de 40 características seleccionadas del dataset CIC-IDS2018 y su descripción

```
1: Input: Dataset CSE-CIC-IDS2018 (múltiples archivos CSV)
2: Output: Lista de características seleccionadas, archivo 'selected_features_auto.txt'
3: Configuración:
4:   - n_features = 40 (número fijo de características)
5:   - exclude_cols = ['id', 'Flow ID', 'Src IP', 'Src Port', 'Dst IP', 'Dst Port', 'Timestamp']
6:   - label_mapping = {Botnet Ares→Bot, Web Attacks→WebAttacks,
Infiltration→Infilteration}
```

```
7: Preprocesamiento:
8:   for cada archivo CSV en carpeta:
9:     Cargar con Polars
10:    Eliminar columnas excluidas
11:    Convertir columnas numéricas a float64
12:  end for
13:  Concatenar todos los archivos en DataFrame único
14:  Normalizar etiquetas según mapping
15:  Filtrar etiquetas con "SQL"
16:  Aplicar Label Encoding a 'Protocol'
17:  Mover 'Label' al final como target
18: Exploratorio:
19:   Mostrar shape, missing values, distribución de clases
20: Selección de Características (Método Híbrido):
21:   Separar X (features) y y (target)
22:   Aplicar tres métodos en paralelo:
23:     1. SelectKBest (f_classif) → selecciona n_features
24:     2. RFE con RandomForest → selecciona n_features
25:     3. Feature Importance RandomForest → top n_features
26:   Calcular frecuencia de aparición en métodos
27:   Estrategia de consenso:
28:     - Características que aparecen en ≥2 métodos
29:     - Si muy pocas (< n_features//2): tomar top n_features por frecuencia
30:   selected_features = lista final consensuada
31:   Mostrar top 10 características seleccionadas
32: Guardado:
33:   Guardar lista completa en 'selected_features_auto.txt'
34: Output Final:
35:   Devolver selected_features y distribución de clases
36: Estadísticas:
37:   - Número total de características seleccionadas
38:   - Archivo de resultados generado
```

Ilustración 15 Algoritmo de selección de características híbrido para improved CSE-CIC-IDS2018. Combina tres métodos, Univariado, RFE y Importancia Random Forest mediante consenso, seleccionando características que aparecen en múltiples métodos para garantizar una mejor selección.

Este algoritmo procesa el dataset CSE-CIC-IDS2018 mediante una estrategia de selección priorizada que combina Random Forest, RFE y SelectKBest. Identifica sistemáticamente las 40 características más relevantes para clasificación de tráfico

de red, generando resultados optimizados para modelos de detección de intrusiones. El proceso incluye preprocesamiento, análisis exploratorio y selección jerárquica de características.

2.6.3.2 Transformación de variables

2.6.3.2.1 Normalización de atributos numéricos

La normalización de datos es un proceso clave en la preparación de datos que ajusta los valores a una escala común, facilitando la comparación de atributos con rangos diferentes. Esto se logra mediante técnicas como la normalización por el máximo, la normalización por la diferencia y el escalado decimal. Cada método transforma los datos para que se ubiquen dentro de un rango específico, como (0, 1), o ajusta los valores para evitar sesgos por magnitudes elevadas, lo que es crucial en modelos como redes neuronales o métodos basados en distancias. Sin una normalización adecuada, los atributos con valores más altos pueden distorsionar los resultados de los modelos [84].

Por otro lado, la estandarización Z es otro enfoque utilizado para normalizar los datos, asegurando que los valores resultantes tengan una media de cero y una desviación estándar de uno. Este proceso permite comparar medidas similares provenientes de poblaciones distintas, ya que los valores Z indican la posición relativa de un dato respecto a su media, independientemente de la escala original. La estandarización Z es especialmente útil en métodos basados en distancias, ya que garantiza que todos los atributos contribuyan de manera equitativa a las mediciones, evitando sesgos derivados de las diferencias en la magnitud de los valores originales [84]. La fórmula para obtener un valor estandarizado Z es:

$$Z_i = \frac{X_i - \mu}{\sigma}$$

donde X_i es el valor original del atributo, μ es la media de los datos y σ es la desviación estándar de los datos.

La clase StandardScaler de scikit-learn realiza la estandarización de los datos al eliminar la media y escalar los atributos a una varianza unitaria. Esta transformación se basa en el cálculo del puntaje estándar de cada muestra, utilizando la

estandarización Z mencionada anteriormente. La estandarización es particularmente importante en muchos algoritmos de machine learning, ya que mejora el rendimiento de modelos que asumen que las características están distribuidas de manera normal con media 0 y varianza 1. Además, el StandardScaler ayuda a evitar que atributos con varianzas significativamente mayores dominen el modelo. Sin embargo, es importante tener en cuenta que el StandardScaler es sensible a los outliers, ya que estos pueden alterar los cálculos de la media y la desviación estándar, afectando así la escala de los datos [85].

```
# Luego aplicar StandardScaler
scaler = StandardScaler()
X_train_processed = scaler.fit_transform(X_train_smote)
X_val_processed = scaler.transform(X_val_fold)
y_train_processed = y_train_smote
```

Ilustración 16 Escalar las características en python con scikit-learn

En la Ilustración 15 se muestra código aplica la estandarización a los datos seleccionados X_selected utilizando StandardScaler. Primero, crea el objeto scaler, y luego usa fit_transform() para ajustar el escalador y transformar los datos, centrando cada atributo en su media y escalando a una desviación estándar de 1. El resultado es almacenado en X_scaled.

2.6.3.2.2 Codificación de etiquetas

2.6.3.2.2.1 Label encoding

LabelEncoder es un transformador de scikit-learn utilizado para convertir etiquetas categóricas en valores numéricos. Asigna un valor entero único a cada clase, dentro del rango de 0 a n_clases -1, donde n_clases es el número total de clases presentes en el conjunto de datos. Este transformador es especialmente útil para normalizar etiquetas, ya que puede convertir tanto valores numéricos como no numéricos siempre que sean hashables y comparables en una representación numérica. Su principal objetivo es preparar las etiquetas de destino para su uso en algoritmos de aprendizaje automático, los cuales requieren entradas numéricas para su procesamiento y modelado [86].

Etiqueta	Valor numérico
BENIGN	0
WebAttacks	1
Bot	2
Infiltration	3

Tabla 22 Codificación de etiquetas utilizando LabelEncoder

2.6.3.2.2.2 One-Hot encoding

La codificación one-hot es una técnica crucial en el preprocesamiento de datos, especialmente para variables categóricas. Consiste en transformar cada categoría en un vector binario, en el cual únicamente una posición contiene el valor 1 mientras las restantes permanecen en 0, representando así de manera única cada categoría. Esta técnica es esencial porque convierte las variables categóricas en un formato numérico que los algoritmos de machine learning, incluidas las redes neuronales, pueden procesar. Además, evita suposiciones de relaciones ordinales, manteniendo la independencia entre categorías, y es ampliamente utilizada en tareas como clasificación, NLP, sistemas de recomendación y detección de fraude [87].

BENIGN	WebAttacks	Bot	Infiltration
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Tabla 23 Representación de etiquetas mediante codificación One-Hot

```
# Codificación de etiquetas
le = LabelEncoder()
y_enc = le.fit_transform(y)
class_names = le.classes_

print(f" Clases: {class_names}")
```

Ilustración 17 Proceso de codificación de etiquetas utilizando label encodig

```
y_train_cat = to_categorical(y_train_processed, num_classes=len(class_names))
y_val_cat = to_categorical(y_val_fold, num_classes=len(class_names))
```

Ilustración 18 Proceso de codificación utilizando One-Hot Encoding

Estos dos fragmentos de código muestran cómo preparar datos categóricos para un modelo de aprendizaje automático. El primer paso utiliza LabelEncoder para

convertir etiquetas de texto en números enteros. El segundo, to_categorical, transforma esos números en una representación binaria llamada One-Hot Encoding, donde cada clase se convierte en un vector con un '1' en la posición de su categoría y '0' en las demás.

2.6.3.3 Integración de datos balanceados

El desequilibrio de clases ocurre cuando la distribución de clases no es igual, es decir, una clase está subrepresentada (minoritaria) y otra tiene significativamente más muestras (mayoritaria). Este problema es frecuente en aplicaciones del mundo real y es crucial porque los clasificadores estándar de aprendizaje automático, que asumen una distribución uniforme, tienden a sesgarse hacia la clase mayoritaria, lo que impacta drásticamente el rendimiento de clasificación, especialmente para la clase minoritaria de interés. Una de las estrategias principales para abordar esto es el sobremuestreo (over-sampling) a nivel de datos, que busca aumentar el número de muestras de la clase minoritaria [88].

2.6.3.3.1 Smote

La Técnica Sintética de Sobremuestreo de Minorías (SMOTE) es el método más prominente y eficaz para manejar datos desequilibrados. Su funcionamiento se basa en generar nuevos patrones de datos sintéticos mediante la interpolación lineal entre las muestras existentes de la clase minoritaria y sus K vecinos más cercanos. Es decir, SMOTE crea nuevas instancias de la clase minoritaria a lo largo de los segmentos de línea que conectan una muestra con sus vecinos, contribuyendo a la expansión de la densidad de esta clase [88].

Es importante aplicar SMOTE porque los clasificadores tradicionales pueden tener un rendimiento deficiente cuando una clase está en clara minoría, fallando en identificar correctamente los casos de la clase subrepresentada que a menudo son los más críticos. Al balancear la distribución de clases mediante la generación de datos sintéticos, SMOTE ayuda a los algoritmos a superar el sesgo hacia la clase mayoritaria, permitiendo que el modelo aprenda de manera más efectiva las características de la clase minoritaria y mejorando significativamente la precisión general y el rendimiento de la clasificación [88].

Etiqueta	Número de muestras	Porcentaje (%)
BENIGN	500,445	68.25%
WebAttacks	244	0.03%
Bot	142,921	19.49%
Infiltration	89,663	12.23%

Tabla 24 Distribución de Clases en el Conjunto de Datos Antes de Aplicar SMOTE

La tabla muestra un notable desbalance de clases, donde BENIGN posee la mayoría de las muestras (68.25%), lo cual es coherente con el predominio del tráfico legítimo en redes reales. Las clases Infiltration y Bot cuentan con una representación adecuada para el aprendizaje del modelo. En contraste, WebAttacks presenta una proporción mínima (0.03%), por lo que requiere la aplicación de SMOTE para generar muestras sintéticas y equilibrar el conjunto de datos, evitando que el modelo se incline hacia las clases mayoritarias. En cambio, Infiltration y Bot no requieren sobremuestreo, ya que podría generar sobreajuste y afectar el rendimiento en datos reales.

```
# 2. APLICAR SMOTE CORRECTAMENTE (SOLO EN TRAIN, NO EN VALIDATION)
# IMPORTANTE: Aplicar SMOTE solo después de la división train/val de cada fold

# Primero normalizar los datos originales
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_fold)
X_val_processed = scaler.transform(X_val_fold)  # Validación sin SMOTE

# Guardar scaler de este fold
fold_scalers.append(scaler)

# Mostrar distribución ANTES de SMOTE
unique_before, counts_before = np.unique(y_train_fold, return_counts=True)
print("Distribución ANTES de SMOTE:")
for i, count in enumerate(counts_before):
    print(f"  {class_names[unique_before[i]]}: {count} muestras")

# Aplicar SMOTE solo en el conjunto de entrenamiento
webattacks_idx = np.where(class_names == 'webAttacks')[0][0]

# Calcular target más conservador (no tan agresivo como antes)
webattacks_count = np.sum(y_train_fold == webattacks_idx)
target_webattacks = min(1000, webattacks_count * 5)  # Máximo 1000, o 5x el original

try:
    smote = SMOTE(
        sampling_strategy={webattacks_idx: target_webattacks},
        random_state=42,
        k_neighbors=min(5, webattacks_count - 1)  # Ajustar k_neighbors si hay pocas muestras
    )
    X_train_processed, y_train_processed = smote.fit_resample(X_train_scaled, y_train_fold)

    print(f"SMOTE aplicado exitosamente. Target para webAttacks: {target_webattacks}")

except Exception as e:
    print(f"SMOTE falló: {e}")
    print("Usando datos originales sin SMOTE...")
    X_train_processed = X_train_scaled
    y_train_processed = y_train_fold
```

Ilustración 19 Aplicación de SMOTE con Ajuste Dinámico de k_neighbors

En este fragmento de código, se prepara y aplica la técnica SMOTE (Synthetic Minority Oversampling Technique) de forma segura, asegurando que solo se aplique al conjunto de entrenamiento (train_fold) y no al de validación, para evitar la filtración de datos (*data leakage*). Antes de aplicarla, se normalizan los datos con StandardScaler (ajustado solo en *train*) y se imprime la distribución de clases original para referencia.

El objetivo de sobremuestreo para la clase minoritaria (webAttacks) se calcula de manera conservadora y dinámica. En lugar de un número fijo, el *target* (target_webattacks) se establece como cinco veces el conteo original de la clase, pero con un límite máximo absoluto de 1000 muestras (min(1000, webattacks_count * 5)). Esta estrategia busca mejorar la representación de la clase minoritaria sin generar una cantidad excesiva de datos sintéticos que podría sesgar el modelo o provocar sobreajuste.

Para la inicialización de SMOTE, se utiliza un random_state=42 para garantizar la reproducibilidad. De manera crucial, el parámetro k_neighbors se ajusta dinámicamente usando min(5, webattacks_count - 1). Esta es una medida de seguridad esencial que reduce el número de vecinos si la clase minoritaria original tiene muy pocas muestras (p.ej., menos de 6), previniendo errores comunes de SMOTE en esos escenarios.

Finalmente, la aplicación de smote.fit_resample está envuelta en un bloque try...except. Si el proceso de SMOTE falla por alguna razón (a pesar de las precauciones), el *script* está diseñado para revertir y utilizar los datos de entrenamiento originales escalados, asegurando que el *pipeline* de entrenamiento continúe sin interrupciones.

2.6.3.4 División de los datos

La división estratificada es un método de partición de datos que se asegura de dividir los conjuntos de entrenamiento y prueba de una manera específica. Utiliza las etiquetas de clase proporcionadas como referencia para mantener la proporción de estas clases en ambos subconjuntos, tanto en el de entrenamiento como en el de prueba. Esta funcionalidad se controla mediante el parámetro stratify en herramientas como la función train_test_split [89].

Es importante utilizar la división estratificada para garantizar que tanto el conjunto de entrenamiento como el de prueba sean representativos de la distribución original de las clases en el conjunto de datos completo. Esto es crucial, especialmente cuando se trabaja con datos desequilibrados, donde una división aleatoria simple podría llevar a que una clase minoritaria esté ausente o muy subrepresentada en uno de los conjuntos, afectando el aprendizaje y la evaluación del modelo [89].

```
X_develop, X_final_test, y_develop, y_final_test = train_test_split(
    X_selected, y_enc, test_size=0.2, stratify=y_enc, random_state=42
)
```

Ilustración 20 División Estratificada de Datos para Entrenamiento y Prueba

En la ilustración se puede observar la división estratificada aplicada al proyecto de los datos utilizando la función train_test_split. El parámetro test_size=0.2 indica que el 20% de los datos se asignarán al conjunto de prueba, mientras que el 80% restante se destinará al conjunto de entrenamiento. El parámetro stratify=y_resampled asegura que las proporciones de las clases en y_resampled se mantengan constantes en ambos conjuntos. Esto es especialmente importante en conjuntos de datos desequilibrados, ya que garantiza que cada clase esté representada adecuadamente en el conjunto de entrenamiento y prueba. El uso de random_state=42 permite que los resultados sean reproducibles.

2.6.4 Fase 4: Modelado

2.6.4.1 Seleccionar la técnica de modelado

Para validar la elección de la arquitectura basada en Deep Neural Network (DNN), se realizaron pruebas comparativas con modelos Multilayer Perceptron (MLP) y Long Short-Term Memory (LSTM) utilizando el dataset original CSE-CIC-IDS2018, que inicialmente se consideraba como fuente principal de datos.

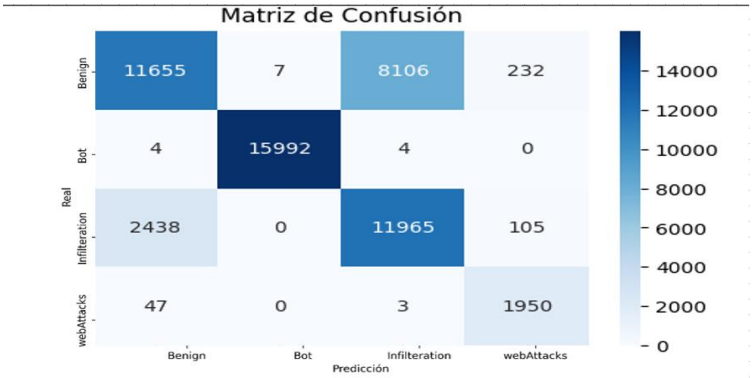


Ilustración 21 Matriz de confusión del modelo MLP en la detección de tráfico de red (dataset original CSE-CIC-IDS2018).

La figura muestra el desempeño del modelo MLP utilizando el dataset original CSE-CIC-IDS2018. Se observa una notable confusión entre las clases Benign e Infiltration, lo cual refleja la dificultad del modelo para distinguir patrones de comportamiento anómalos que se asemejan al tráfico normal. Este resultado evidencia las limitaciones del dataset original, cuyos errores de etiquetado contribuyen a la generación de falsos positivos.

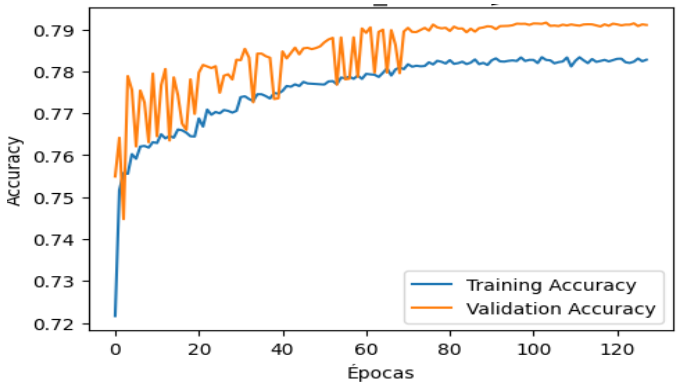


Ilustración 22 Evolución de la precisión del modelo MLP durante el entrenamiento y la validación (dataset original CSE-CIC-IDS2018).

El gráfico evidencia una tendencia estable en la precisión de validación, alcanzando aproximadamente un 79%. Si bien no se aprecia sobreajuste, el crecimiento moderado del desempeño indica que el modelo logra aprender relaciones útiles, pero con una capacidad limitada de generalización frente a patrones confusos.

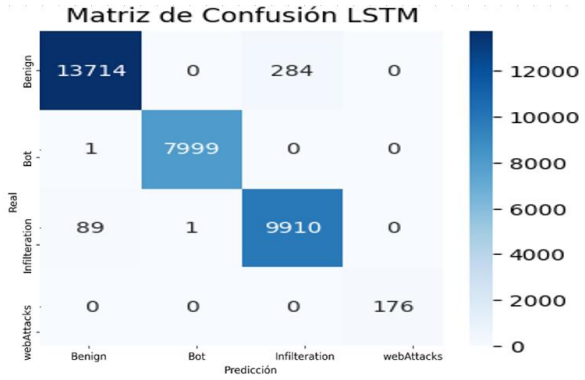


Ilustración 23 Matriz de confusión del modelo LSTM en la detección de tráfico de red (dataset original CSE-CIC-IDS2018).

El modelo LSTM presenta un mejor desempeño en la detección de las clases Bot e Infiltration, reduciendo los falsos negativos en comparación con el MLP. Sin embargo, aún se evidencian confusiones residuales y una dependencia más fuerte del comportamiento temporal de los datos, lo cual no resulta plenamente ventajoso dada la naturaleza estática de las características de flujo del dataset.

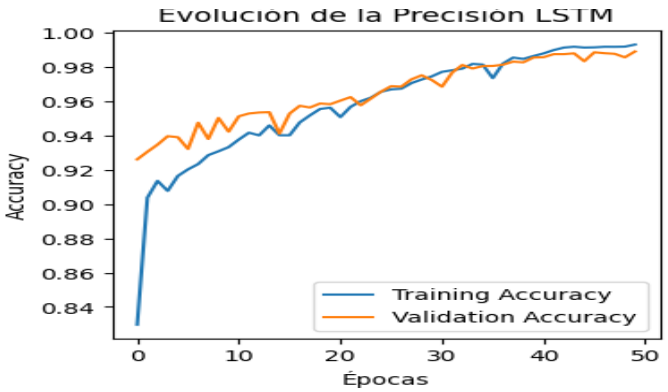


Ilustración 24 Evolución de la precisión del modelo LSTM durante el entrenamiento y validación (dataset original CSE-CIC-IDS2018).

El modelo LSTM alcanza una precisión de validación cercana al 99%, evidenciando un aprendizaje más rápido y una convergencia estable. No obstante, este alto rendimiento en el entrenamiento no se traduce en una mejora real frente a tráfico genuino, lo cual sugiere una pobre capacidad de generalización fuera de las condiciones de entrenamiento, debido a la calidad de los datos.

Para evaluar la capacidad de generalización de los modelos, se realizaron pruebas adicionales utilizando tráfico 100 % benigno capturado en tiempo real desde una red Wi-Fi doméstica, sin incluir muestras del conjunto de entrenamiento. Este procedimiento permitió obtener una referencia preliminar sobre el comportamiento del modelo antes de realizar las pruebas controladas en el entorno de laboratorio. Los resultados se presentan a continuación:

Ilustración 25 Ejecución del sistema inicial con el modelo MLP utilizando tráfico 100 % benigno entrenado con dataset original CSE-CIC-IDS2018.

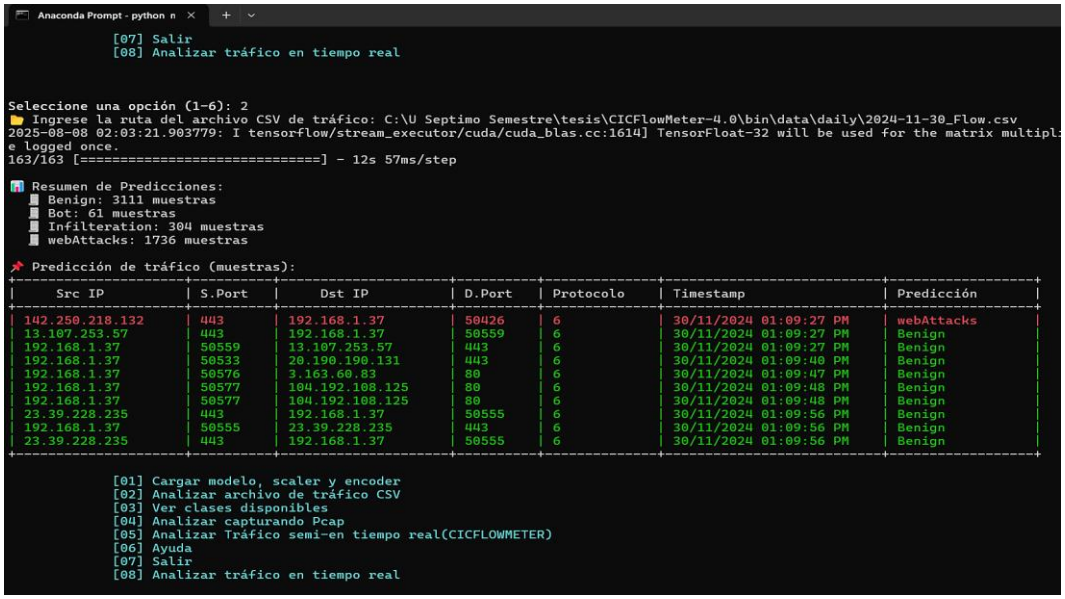


Ilustración 26 Ejecución del sistema inicial con el modelo LSTM utilizando tráfico 100 % benigno entrenado con dataset original CSE-CIC-IDS2018.

Los resultados demostraron que, aunque el modelo LSTM alcanzó métricas aparentemente superiores durante el entrenamiento (precisión de validación ~98% vs. ~79% del MLP), su rendimiento se deterioró significativamente al evaluarse con tráfico real capturado. En las pruebas con tráfico 100% benigno, el modelo LSTM presentó confusiones considerablemente mayores que el MLP: 3,111 instancias clasificadas como Benign, 61 como Bot, 304 como Infiltration y 1,736 como WebAttacks, cuando todas las muestras eran benignas. En contraste, el modelo MLP, aunque también presentó errores, mostró confusiones más limitadas (3,891 Benign, 19 Bot, 1,315 Infiltration, 2 WebAttacks).

Durante esta primera fase experimental se determinó que el dataset original presentaba inconsistencias de etiquetado que afectaban directamente el desempeño de los modelos, generando falsos positivos y confusiones incluso en tráfico benigno. Conforme avanzó la investigación, se identificó la existencia del conjunto de datos Improved CSE-CIC-IDS2018, una versión corregida que solventa los errores de etiquetado del dataset original y mejora la coherencia entre clases. Por esta razón, se decidió adoptar este dataset en la etapa final de modelado, ya que ofrece un formato más limpio, estructurado y adecuado para el enfoque basado en DNN/MLP.

Clase Predicha	MLP	LSTM	Diferencia
Benign (correcto)	3,891	3,111	-780
Bot (error)	19	61	+42
Infiltration (error)	1,315	304	-1,011
WebAttacks (error)	2	1,736	+1,734
Total de errores	1,336	2,101	+765
Tasa de error	25.6%	40.3%	+14.7%

Tabla 25 Resultados con tráfico benigno real para los modelos MLP y LSTM – Dataset original CSE-CIC-IDS2018

Estudios comparativos, como el de Ali *et al.* ("*Deep Learning vs. Machine Learning for Intrusion Detection in Computer Networks: A Comparative Study*"), demuestran que las arquitecturas MLP, CNN y LSTM alcanzan precisiones muy similares (97–98%), con diferencias marginales en las métricas F1. Aunque el modelo LSTM mostró una ligera ventaja (F1=0.84 frente a 0.82 del MLP), su complejidad computacional es significativamente mayor, lo que dificulta su implementación práctica en entornos con recursos limitados [90].

El MLP ofrece además una capacidad adecuada para procesar datos de alta dimensionalidad, detectar patrones no lineales y mantener una convergencia estable durante el entrenamiento. Su estructura flexible permite incorporar mecanismos de regularización como Dropout y Batch Normalization, reduciendo el sobreajuste y mejorando la generalización del modelo.

Adicionalmente, el dataset CSE-CIC-IDS2018 empleado contiene métricas de flujo que representan resúmenes estadísticos del tráfico, sin una dependencia temporal directa entre registros. Por ello, arquitecturas recurrentes como RNN o LSTM, orientadas al modelado secuencial, no aportan ventajas sustanciales en este contexto y aumentarían innecesariamente la carga computacional. En consecuencia, el uso del MLP resulta la opción más eficiente, escalable y coherente con la naturaleza de los datos [90].

2.6.4.1.1 Supuestos del Modelado

Los supuestos del modelado con DNN se centraron en garantizar la validez del modelo y la estabilidad del entrenamiento. Se asumió la independencia de las muestras, considerando cada flujo de red como una observación autónoma. Las

características fueron normalizadas con StandardScaler para mejorar la convergencia y estabilizar el gradiente. Las variables categóricas, como el protocolo y la etiqueta, se codificaron mediante one-hot para evitar sesgos ordinales. Finalmente, se aplicó SMOTE en las clases minoritarias críticas, equilibrando el aprendizaje y reduciendo la influencia de la clase mayoritaria.

2.6.4.2 Generar el diseño de prueba

2.6.4.2.1 Estrategia de Particionado

La estrategia de particionado de datos se diseñó con el objetivo de garantizar una evaluación robusta y representativa del modelo. Para ello, se implementó una validación cruzada estratificada de cinco pliegues 5-fold Stratified Cross-Validation en combinación con un conjunto de prueba final independiente. En una primera división, el dataset se separó en un conjunto de desarrollo conformado por 346,262 muestras (80%) y un conjunto de prueba final con 86,566 muestras (20%), destinado exclusivamente a la evaluación del rendimiento del modelo tras el entrenamiento. Posteriormente, el conjunto de desarrollo fue sometido a validación cruzada estratificada en cinco pliegues, manteniendo en cada uno la proporción original de clases para preservar la representatividad de las distribuciones. Es importante destacar que la técnica de sobremuestreo sintético (SMOTE) se aplicó únicamente sobre los subconjuntos de entrenamiento de cada pliegue, evitando así la contaminación de datos y garantizando que las métricas de validación reflejen un desempeño realista del modelo en escenarios de desbalance de clases.

2.6.4.2.2 Métricas de Evaluación

Para la evaluación comprehensiva del modelo se establecieron métricas tanto principales como operativas, con el fin de medir su desempeño desde diferentes perspectivas. Entre las métricas se incluyen:

Métrica	Definición	Utilidad en este estudio
Accuracy	Predicciones correctas sobre el total de muestras.	Brinda una medida general del rendimiento inicial.
Precision	Proporción de positivos predichos que fueron correctos.	Indica la confiabilidad al identificar tráfico malicioso.
Recall	Proporción de verdaderos positivos detectados.	Evalúa la capacidad de detección, clave en seguridad.

Métrica	Definición	Utilidad en este estudio
F1-Score	Balance entre precisión y recall.	Permite valorar el equilibrio entre detección y exactitud.
FPR	Benignos mal clasificados como ataques.	Determina el riesgo de generar falsas alarmas.

Tabla 26 Métricas Principales de Evaluación

2.6.4.3 Seleccionar la técnica de modelado

2.6.4.3.1 Arquitectura de la red neuronal

Capa	Neuronas	Activación	Regularización / Técnicas	Dropout
Capa 1	256	ReLU	BatchNormalization, L1-L2 (α_1=1e-5, α_2=1e-4)	0.3
Capa 2	128	ReLU	BatchNormalization, L1-L2 (α_1=1e-5, α_2=1e-4)	0.4
Capa 3	64	ReLU	BatchNormalization, L1-L2 (α_1=1e-5, α_2=1e-4)	0.3
Capa 4	32	ReLU	BatchNormalization, L1-L2 (α_1=1e-5, α_2=1e-4)	0.2
Capa Salida	4	Softmax	—	—

Tabla 27 Arquitectura de la Red Neuronal Feedforward Profunda (DNN/MLP)

La red neuronal implementada corresponde a un modelo feedforward profundo (DNN/MLP) optimizado para la clasificación multiclase del tráfico de red. Cada capa oculta combina funciones de activación ReLU, Batch Normalization y Dropout con tasas variables, lo que permite estabilizar el entrenamiento, mejorar la generalización y prevenir sobreajuste. La regularización L1-L2 aplicada en todas las capas densas penaliza los pesos excesivos, contribuyendo a la robustez del modelo frente a datos ruidosos. La capa de salida utiliza la función Softmax, proporcionando probabilidades normalizadas para cada una de las cuatro clases (Benign, Botnet, Infiltration, WebAttacks).

2.6.4.3.2 Configuración de entrenamiento

Componente	Configuración / Valores	Descripción / Justificación
Optimizador	Adam	Combina ventajas de momentum y RMSProp, ajustando dinámicamente la tasa de aprendizaje.
Función de pérdida	Categorical Crossentropy	Adecuada para clasificación multiclase, penaliza de manera proporcional errores por clase.
Callbacks: EarlyStopping	monitor='val_loss', patience=10	Detiene el entrenamiento si no hay mejora en validación, evitando sobreajuste.
Callbacks: ReduceLROnPlateau	factor=0.5, patience=5, min_lr=1e-7	Reduce la tasa de aprendizaje al estabilizarse la función de pérdida, mejorando convergencia.
Callbacks: ModelCheckpoint	Guardado del mejor modelo según validación	Permite conservar el modelo óptimo durante el entrenamiento.
Batch size	256	Tamaño de mini-batch que equilibra eficiencia computacional y estabilidad del gradiente.
Épocas máximas	50	Límite superior de entrenamiento, evitando tiempos excesivos sin mejoras.
Semilla de reproducibilidad	42	Garantiza resultados consistentes en experimentos repetidos.

Tabla 28 Configuración de Entrenamiento de la Red DNN/MLP

El modelo se entrenó utilizando el optimizador Adam, que combina las ventajas de los métodos de momento y adaptación de tasas de aprendizaje. La función de pérdida Categorical Crossentropy se eligió por ser apropiada para problemas de clasificación multiclase, penalizando los errores proporcionalmente a la probabilidad predicha. Se implementaron callbacks estratégicos: *EarlyStopping* para detener el entrenamiento ante ausencia de mejora en la validación,

ReduceLROnPlateau para disminuir dinámicamente la tasa de aprendizaje y favorecer la convergencia, y *ModelCheckpoint* para guardar la mejor versión del modelo según el desempeño en validación. Los hiperparámetros se definieron considerando un compromiso entre eficiencia y estabilidad: un *batch size* de 256 para un cálculo eficiente del gradiente, un máximo de 50 épocas para evitar entrenamiento innecesario, y una semilla de reproducibilidad (42) para asegurar consistencia en los resultados experimentales.

```
1: Input: Dataset CSE-CIC-IDS2018 - Intrusion Detection Dataset (CSV files con undersampling)
2: Output: Métricas de clasificación, modelos entrenados por fold, gráficos ROC/confusión, tabla comparativa de CV
3: Modelo MLP:
4:    Sequential con 4 capas ocultas: Dense(256, 128, 64, 32) + BatchNorm + Dropout
5:    Regularización L1-L2, activación ReLU, salida Softmax
6: Técnicas de Balanceo:
7:    SMOTE para webAttacks: target = min(1000, count × 5)
8:    Class Weights: compute_class_weight con 'balanced'
9: Optimizador:
10:   Adam (learning_rate=0.001)
11: Callbacks:
12:   EarlyStopping (patience=10), ReduceLROnPlateau (patience=8)
13: Configuración Global:
14:   - batch_size = 128
15:   - learning_rate = 0.001
16:   - num_epochs = 50
17:   - n_splits = 5 (StratifiedKFold)
18:   - test_size = 0.2 (80% desarrollo / 20% test final)
19:   - random_seed = 42
20:   - Features: 39 características de flujo + Protocol (one-hot)
21:   - Clases: Benign, Bot, webAttacks, Infilteration (4 clases)
22: Preprocesamiento de Datos:
23: for cada archivo CSV in carpeta dataset:
24:   Cargar con Polars y eliminar columnas excluidas
25:   Aplicar mapeo de etiquetas (Botnet→Bot, Web Attack→webAttacks, Infiltration→Infilteration)
26:   Undersampling de Benign a 40,000 por archivo
27: end for
28: Separar X (features) y y (labels)
29: One-hot encoding de Protocol
30: Label encoding de clases target
31: Split inicial: X_develop, X_final_test (80/20 estratificado)
32: Validación Cruzada Estratificada:
```

```
33: skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
34: for cada fold in skf.split(X_develop, y_develop):
35:    Dividir datos: X_train_fold, X_val_fold, y_train_fold, y_val_fold
36:    Aplicar StandardScaler: fit en train, transform en val (sin data leakage)
37:    Aplicar SMOTE solo en train con target para webAttacks
38:    Calcular class_weights en train post-SMOTE
39:    Crear y compilar modelo MLP con Adam y métricas [accuracy, F1Score]
40:    Entrenar durante 50 épocas con validación interna
41:    Aplicar callbacks: EarlyStopping y ReduceLROnPlateau
42:    Evaluar el modelo en validation set:
43:        - val_loss, val_accuracy, val_f1
44:        - Classification report con precision/recall/F1 por clase
45:    Guardar modelo, scaler, history y métricas del fold
46: end for
47: Análisis de Resultados:
48:    Crear DataFrame con métricas de los 5 folds
49:    Calcular promedios y desviaciones estándar
50:    Seleccionar mejor fold por val_accuracy
51: Evaluación en Test Final:
52:    Cargar mejor modelo y scaler del mejor fold
53:    Preprocesar X_final_test con scaler del mejor fold
54:    Predecir y calcular: test_loss, test_accuracy, test_f1
55:    Generar classification report y matriz de confusión final
56: Guardado de Recursos:
57:    Guardar en recursosMLP/: feature_columns.pkl, label_encoder.pkl,
58:    mlp_final_model.h5, protocol_columns.pkl, scaler.pkl
59: Generación de Visualizaciones:
60:    Obtener datos de validación del mejor fold
61:    Calcular predicciones y métricas por clase
62:    Generar y guardar en graficasMejorFold/:
63:        - Matriz de confusión (absoluta y normalizada)
64:        - Evolución de loss y accuracy (train vs val)
65:        - Distribución de predicciones por clase
66:        - Métricas por clase (Precision, Recall, F1)
67:        - Métricas con escala ajustada (ylim 0.8-1.0)
68:        - Gráfico combinado (6 subplots en 2x3)
69: Resumen Final:
70:    - Mostrar promedios de CV: Accuracy y F1-Score (mean ± std)
71:    - Mostrar métricas en Test Final del mejor fold
72:    - Total de configuraciones evaluadas: 5 folds × 1 arquitectura MLP
```

Ilustración 27 Algoritmo de Entrenamiento y evaluación del modelo MLP con validación cruzada 5-fold para clasificación de tráfico de red en CSE-CIC-IDS2018.

2.6.4.4 Evaluación del Modelo

2.6.4.4.1 Resultados de Validación Cruzada

Para medir el rendimiento del sistema de detección propuesto, se utilizaron métricas ampliamente aceptadas en la evaluación de modelos de clasificación, detalladas en la siguiente tabla. Estas métricas permiten analizar tanto la capacidad general del modelo como su desempeño específico por clase, considerando escenarios con desbalance de datos.

Métrica	Definición	Fórmula
Precisión (Precision)	Predicciones positivas correctas respecto al total de predicciones positivas.	$Precision = \frac{TP}{TP + FP}$
Recall / Sensibilidad	Instancias positivas correctamente identificadas.	$Recall = \frac{TP}{TP + FN}$
F1-Score	Media armónica entre precisión y recall.	$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$
Exactitud Global (Accuracy)	Predicciones correctas sobre el total.	$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$
Tasa de Falsos Positivos (FPR)	Proporción de instancias benignas clasificadas incorrectamente como ataques.	$FRP = \frac{FP}{FP + TN}$

Tabla 29 Métricas consideradas para la evaluacion del modelo

Los resultados de la validación cruzada estratificada de 5 pliegues demuestran la robustez y consistencia del modelo:

Fold	Val Loss	Val Accuracy	Val Precision	Val Recall	Val F1
1	0.0549	0.9969	0.9969	0.9969	0.9969
2	0.0234	0.9967	0.9967	0.9967	0.9967
3	0.0509	0.9968	0.9968	0.9968	0.9968
4	0.0224	0.9971	0.9971	0.9971	0.9971
5	0.0233	0.9970	0.9970	0.9970	0.9970
Media	0.0358	0.9969	0.9969	0.9969	0.9969
Desv. Std.	0.0151	0.0001	0.0001	0.0001	0.0001

Tabla 30 Resultados de la Validación Cruzada Estratificada (5-Fold) de la Red DNN/MLP

La validación cruzada estratificada de 5 pliegues permitió evaluar la robustez y consistencia del modelo. Los resultados muestran valores promedio de loss de 0.0358, accuracy de 0.9969, precision de 0.9969, recall de 0.9969 y F1-Score de 0.9969, con desviaciones estándar muy bajas ($\sigma = 0.0001$ para las métricas de desempeño), lo que evidencia una alta estabilidad del modelo frente a distintas divisiones de los datos. Estos resultados indican que la red neuronal es capaz de generalizar de manera efectiva sobre diferentes subconjuntos de entrenamiento y validación, manteniendo un rendimiento consistente en la clasificación de todas las clases del dataset.

2.6.4.4.2 Evaluación en Conjunto de Prueba Final

El mejor modelo fue el Fold 4, con un Accuracy = 0.9971 fue evaluado en el conjunto de prueba final reservado, obteniendo los siguientes resultados:

Métrica	Valor
Test Accuracy	0.9973
Test F1-Score	0.9973
Test Loss	0.0210

Tabla 31 Métricas Globales del Conjunto de Prueba Final

Clase	Precision	Recall	F1-Score	Support
Benign	0.9980	0.9960	0.9970	40,000
Bot	1.0000	1.0000	1.0000	28,584
Infilteration	0.9913	0.9956	0.9934	17,933
webAttacks	0.9800	1.0000	0.9899	49
Macro Avg	0.9923	0.9979	0.9951	86,566
Weighted Avg	0.9973	0.9973	0.9973	86,566

Tabla 32 Evaluación del mejor modelo por clase

La evaluación del mejor modelo, seleccionado a partir del fold 4 de la validación cruzada, en el conjunto de prueba final mostró un desempeño altamente consistente. Las métricas globales reflejan un Accuracy de 0.9973 y un F1-Score de 0.9973, con una pérdida de 0.0210, lo que indica una excelente capacidad de generalización. Al analizar el rendimiento por clase, se observa que las clases mayoritarias (Benign, Bot, Infiltration) presentan métricas sobresalientes, especialmente con un F1-Score de 0.9970 para Benign, 1.0000 para Bot y 0.9934 para Infiltration. La clase

minoritaria WebAttacks, aunque con una baja representación, mantiene un F1-Score de 0.9899, lo que demuestra que el modelo logra un desempeño sólido, gracias a la estrategia de balanceo implementada. Las métricas macro y ponderadas refuerzan que la red neuronal mantiene un equilibrio eficaz en la clasificación de todas las clases, destacándose por su robustez en escenarios multiclase desbalanceados.

2.6.4.4.3 Análisis de la Evolución del Loss

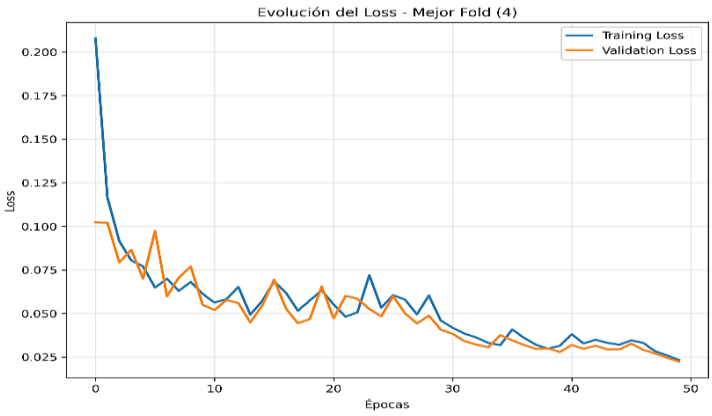


Ilustración 28 Convergencia estable del loss durante el entrenamiento sin evidencia de overfitting

El gráfico de pérdida revela un comportamiento de entrenamiento correcto, donde tanto el loss de entrenamiento como el de validación disminuyen de manera consistente a lo largo de las épocas. Ambas curvas convergen hacia valores muy bajos (alrededor de 0.025), sin mostrar signos evidentes de sobreajuste, ya que el loss de validación no diverge del de entrenamiento. La convergencia rápida en las primeras épocas sugiere que el modelo aprende eficientemente los patrones de los datos.

2.6.4.4.3 Análisis de la Evolución de la Precisión

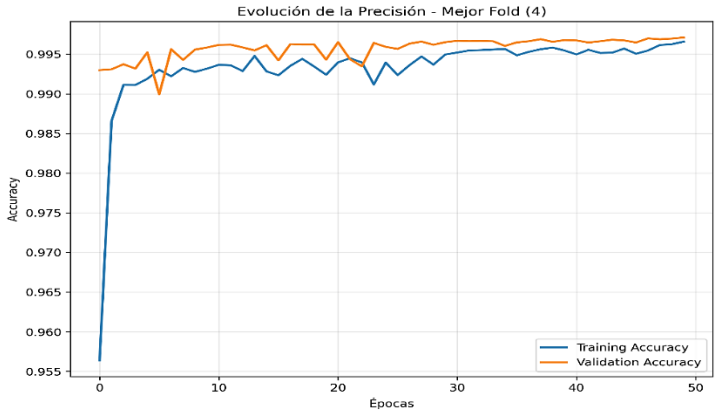


Ilustración 29 Evolución de precisión del modelo

La evolución de la precisión muestra un incremento rápido y sostenido, alcanzando valores superiores al 99.5% tanto en entrenamiento como en validación. El hecho de que ambas curvas se mantengan muy cercanas y con una tendencia ascendente constante indica que el modelo generaliza bien y no presenta sobreajuste. La precisión de validación ligeramente superior al entrenamiento en algunas épocas es un indicador positivo de la capacidad de generalización del modelo.

2.6.4.4.3 Análisis de la Matriz de Confusión

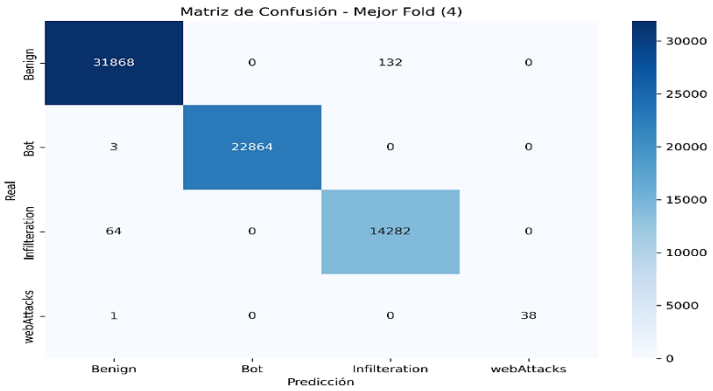


Ilustración 30 Matriz de confusión de clasificación de las diferentes clases

La matriz de confusión absoluta demuestra un rendimiento muy bueno del modelo, con valores muy altos en la diagonal principal y errores mínimos. La clase "Bot" muestran clasificaciones con 3 confusiones en bening, mientras que Benign presenta algunos casos de confusión con "Infilteration" (132 casos) y así mismo "Infilteration" presenta algunos casos de confusión con "Benign" (64 casos). La clase "webAttacks" tiene una representación mínima, pero se clasifica correctamente. Los errores de clasificación son prácticamente mínimos en relación al tamaño total del dataset.

2.6.4.4.3 Análisis de la Matriz de Confusión Normalizada

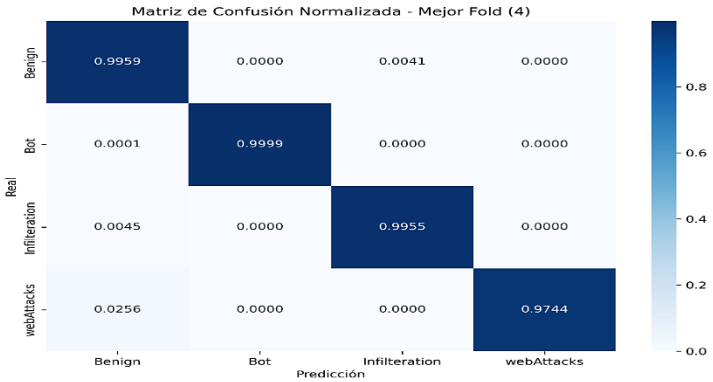


Ilustración 31 Matriz de confusión normalizada de clasificación de las diferentes clases

La matriz de confusión normalizada refleja la excelente precisión del modelo, con valores en la diagonal superiores al 99.5% para todas las clases. La clase "Benign" presenta un 99.59% de precisión, "Bot" alcanza un 99.99%, "Infiltration" llega al 99.55%, y "webAttacks" mantiene un 97.44%. Los errores de clasificación cruzada son mínimos, destacando solo un 0.41% de casos benignos mal clasificados como infiltración, un 0.25% de casos de infiltración incorrectamente clasificados como benignos, y un 2.56% de casos de webAttacks mal clasificados como benignos. Este bajo nivel de error refleja la robustez del modelo y su capacidad para manejar correctamente incluso las clases minoritarias.

2.6.4.4.3 Análisis de las Métricas por Clase

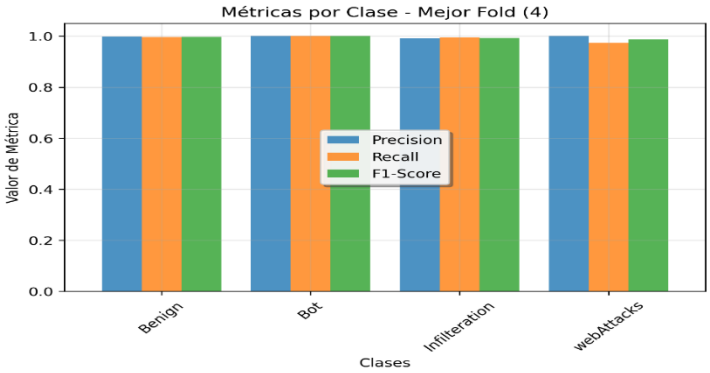


Ilustración 32 Métricas de clasificación en todas las clases

El gráfico de métricas revela un rendimiento sobresaliente para las clases "Benign", "Bot" e "Infiltration", con valores de precision, recall y F1-score cercanos a 1.0 en todas ellas. La clase "webAttacks" muestra una ligera caída en la precision, alcanzando aproximadamente 0.975, pero mantiene valores perfectos en recall y F1-score. Esta pequeña variación en la clase "webAttacks" podría explicarse por su baja representación en el conjunto de datos, lo que puede generar una mayor variabilidad en las métricas de evaluación, aunque sigue mostrando un buen rendimiento en términos de recuperación y balance de las predicciones.

2.6.5 Fase 5: Evaluación

2.6.5.1 Evaluar Resultados

En esta etapa, se llevó a cabo la evaluación preliminar del modelo desarrollado, considerando los resultados obtenidos a partir del propio dataset CSE-CIC-IDS2018. El propósito principal fue determinar hasta qué punto el modelo logra aproximarse a los objetivos del proyecto planteados en fases anteriores. Para ello,

se analizaron métricas clásicas de evaluación como Accuracy, Precision, Recall, F1-Score y tasa de falsos positivos (FPR), las cuales permiten obtener una primera visión del desempeño del sistema.

Los resultados mostraron que el modelo distingue adecuadamente entre tráfico benigno y malicioso, con un bajo margen de error. Sin embargo, se observó la presencia de falsos negativos principalmente en la clase *Infiltration*, los cuales representan amenazas potenciales no detectadas.

Clase Real	Predicción: Benign	Predicción: Bot	Predicción: Infilteration	Predicción: webAttacks	Total Real	Falsos Negativos
Benign	31,868	0	132	1	31.936	133 (0.42%)
Bot	3	22,864	0	0	22864	3 (0.01%)
Infilteration	64	0	14,282	0	14414	65 (0.45%)
webAttacks	1	0	0	38	38	1 (2.56%)
Total Predicho	31,936	22,864	14,414	39	-	-
Falsos Positivos	68 (0.21%)	0 (0.00%)	132 (0.89%)	1 (2.56%)	-	-

Tabla 33 Matriz de confusión con análisis de falsos positivos y falsos negativos

Análisis de Falsos Negativos (FN), se identificaron un total de 201 falsos negativos sobre 69,253 muestras. Los más críticos corresponden a 64 casos de Infilteration clasificados como Benign (0.45% de error), representando amenazas potenciales no detectadas. La clase Bot presenta 3 falsos negativos, mientras que webAttacks presenta 1.

Análisis de Falsos Positivos (FP), se registraron 201 falsos positivos en total. Los más significativos son 132 muestras de tráfico Benign clasificadas erróneamente como Infilteration, lo que podría generar alertas innecesarias. La clase Bot no presenta falsos positivos, confirmando su alta confiabilidad.

Tasa Global de Error, el modelo presenta una tasa de error global de apenas 0.29% (201 clasificaciones incorrectas de 69,253 muestras totales), evidenciando su alta precisión para distinguir entre tráfico benigno y malicioso.

De este modo, la fase de evaluación permitió obtener una idea inicial sobre la validez del enfoque seleccionado. Sin embargo, se reconoce que los valores aquí obtenidos no garantizan el rendimiento en entornos prácticos, ya que la verdadera validación se realizará en la fase 6 (Implementación), al probar el modelo con tráfico real capturado y procesado en condiciones controladas de red.

2.6.5.2 Revisar el Proceso

La revisión del proceso confirmó que las decisiones metodológicas adoptadas — como el preprocesamiento, la selección de características y el balanceo de clases mediante SMOTE y pesos de clase influyeron positivamente en la calidad de los resultados.

Se verificó además que la elección de una arquitectura DNN fue coherente con la naturaleza del problema, dado el carácter numérico y multidimensional del dataset. Sin embargo, se identificó como limitación el uso exclusivo de datos del propio conjunto CSE-CIC-IDS2018, lo que restringe la generalización del modelo. Por ello, se considera que el proceso ha sido consistente hasta esta etapa, pero requiere validación externa y posibles ajustes de hiperparámetros en fases posteriores.

2.6.5.3 Determinar los Próximos Pasos

Los resultados obtenidos permiten proyectar la siguiente fase: validar el modelo en un entorno virtualizado mediante VirtualBox, donde se generará y capturará tráfico real. Los archivos PCAP serán procesados con CICFlowMeter para evaluar la capacidad de generalización del modelo frente a datos no vistos. Finalmente, se compararán las métricas obtenidas con las del entrenamiento; si se observa una disminución significativa, se aplicarán técnicas de regularización o reentrenamiento con el tráfico capturado para fortalecer su desempeño en condiciones reales.

2.6.6 Fase 6: Implementación

La presente sección describe la planificación técnica y operativa para transformar el prototipo de detección y clasificación de malware en una herramienta reproducible y ejecutable en un laboratorio virtual controlado. Se adopta como alternativa de implementación una única solución: una aplicación modular desarrollada en Python con interfaz de consola (CLI) que se instalará en una

máquina Windows (referida en lo sucesivo como SISTEMA). El SISTEMA se integrará, para efectos de prueba y validación, con otras máquinas virtuales ejecutadas en VirtualBox (Servidores y Atacante/Kali y Host linux), configuradas en redes virtuales tipo NAT en VirtualBox, con la finalidad de generar y capturar tráfico representativo sin impactar redes reales. En esta etapa el enfoque está en la operacionalización del software — instalación, ejecución, recolección de archivos a procesar.

2.6.6.1 Descripción del Sistema Desarrollado

Antes de la implementación práctica, se desarrolló un sistema en Python orientado a línea de comandos, diseñado para ejecutar tareas de detección y clasificación de malware sobre tráfico de red. Este sistema se implementó en un entorno Windows, con la posibilidad de procesar tanto archivos CSV generados por herramientas como CICFlowMeter, como también capturas en tiempo real a través de interfaces de red con scapy y con el mismo CICFlowMeter.

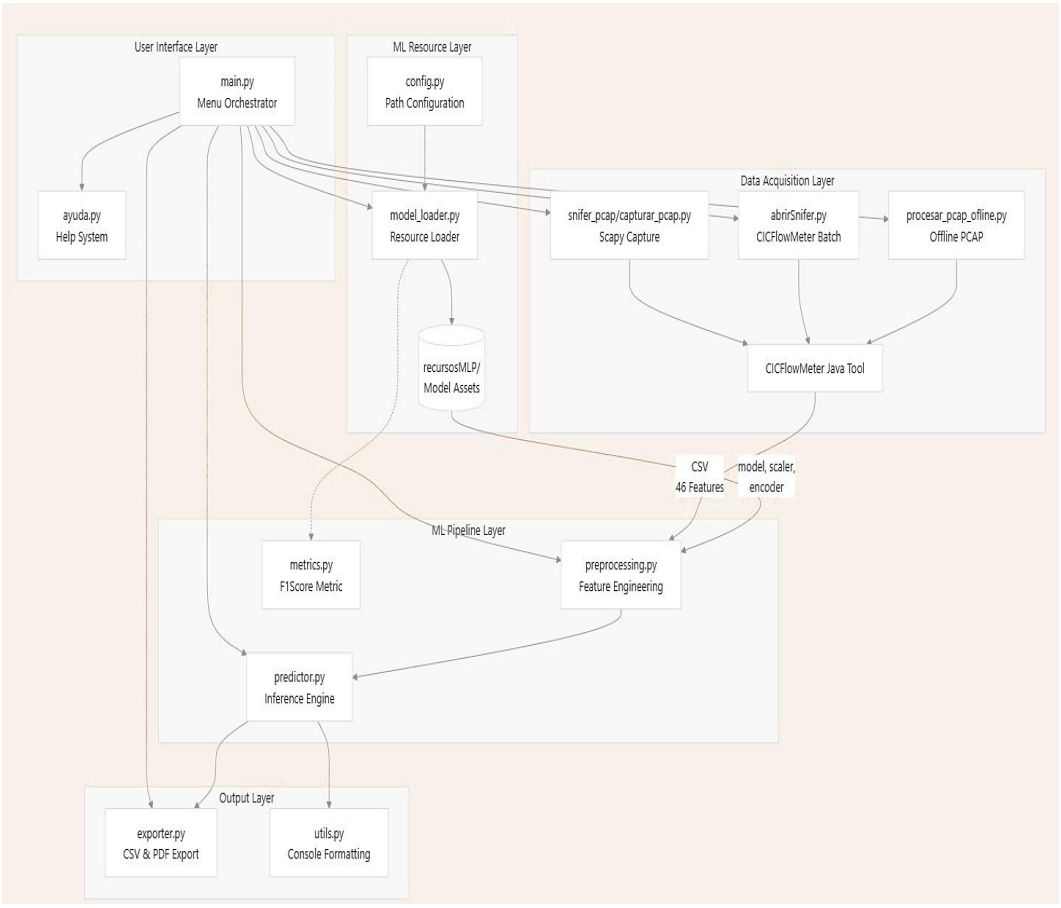


Ilustración 33 Diagrama con las principales capas arquitectónicas y sus relaciones

El diagrama ilustra la arquitectura de un sistema de análisis de datos de red basado en deeplearning organizado en capas funcionales. La Capa de Interfaz de Usuario orquesta el flujo que dirige la captura de datos en la Capa de Adquisición de Datos (usando herramientas como CICFlowMeter para generar 46 características). Estos datos alimentan la Capa de pipeline de ML, donde se realiza la ingeniería de características y el motor de inferencia (predictor.py) aplica el modelo cargado para generar predicciones. Finalmente, los resultados pasan a la Capa de Salida para su formato y exportación (CSV o PDF).

La capa de configuración centraliza todas las definiciones de rutas y especificaciones de características.

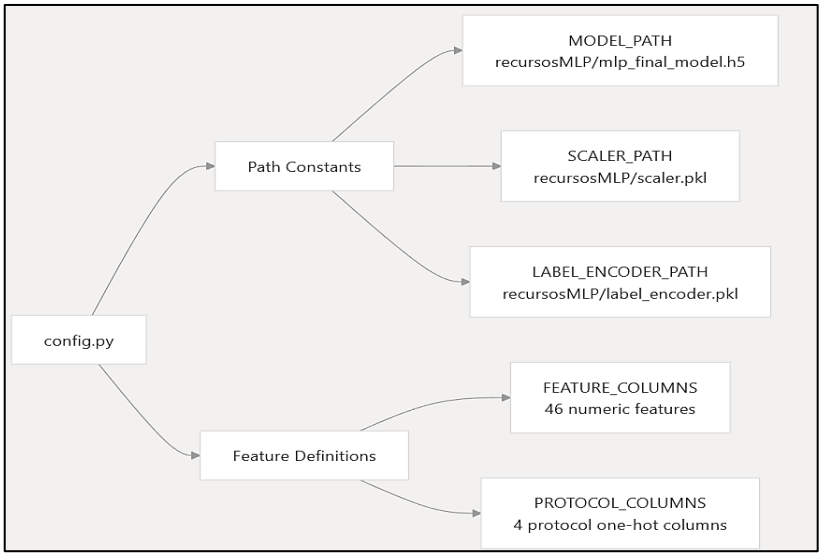


Ilustración 34 Capa de configuración

Constante	Tipo	Propósito
MODEL_PATH	cadena	Ruta al modelo MLP entrenado (.h5)
SCALER_PATH	cadena	Ruta al artefacto StandardScaler (.pkl)
LABEL_ENCODER_PATH	cadena	Ruta al artefacto LabelEncoder (.pkl)
FEATURE_COLUMNS	lista	46 nombres de funciones numéricas cargados desde .pkl
PROTOCOL_COLUMNS	lista	4 nombres de columnas one-hot de protocolo cargados desde .pkl

Tabla 34 Constantes para los archivos necesarios para el sistema

La capa de gestión de recursos maneja la carga y la gestión del ciclo de los archivos necesarios para el modelo de DL. En el archivo model_loader.py la función load_all() carga todos los componentes DL necesarios con un manejo integral de errores:

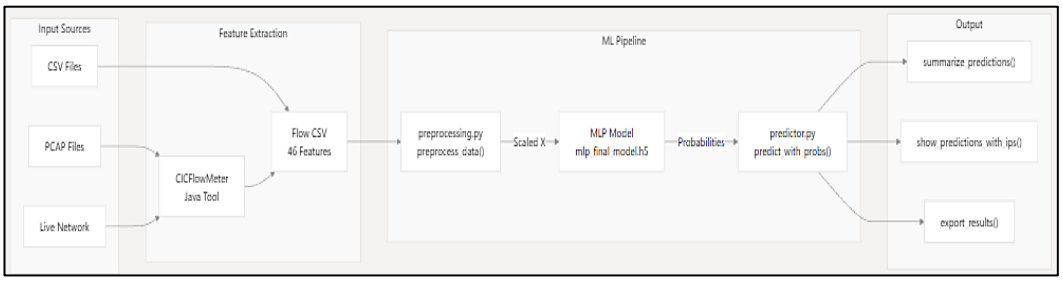


Ilustración 35 Arquitectura de flujo de datos

Este diagrama ilustra el flujo de trabajo central del sistema para clasificar el tráfico de red. El proceso comienza con las Fuentes de Entrada, que incluyen Archivos CSV, Archivos PCAP y la Red en Vivo. En la etapa de Extracción de Características, los datos de PCAP y red son procesados por la Herramienta Java CICFlowMeter, la cual genera un CSV de Flujo con 46 Características clave. A continuación, en el pipeline ML, el archivo de flujo entra en preprocessing.py para ser transformado en datos escalados. Estos datos escalados alimentan el Modelo MLP (mlp_final_model.h5) para obtener Probabilidades de clasificación. Finalmente, predictor.py gestiona estas probabilidades para la Salida, permitiendo resumir las predicciones, mostrar las predicciones y exportar los resultados.

El sistema proporciona tres vías independientes para adquirir datos de tráfico de red, todas convergiendo en formato CSV a través de CICFlowMeter:

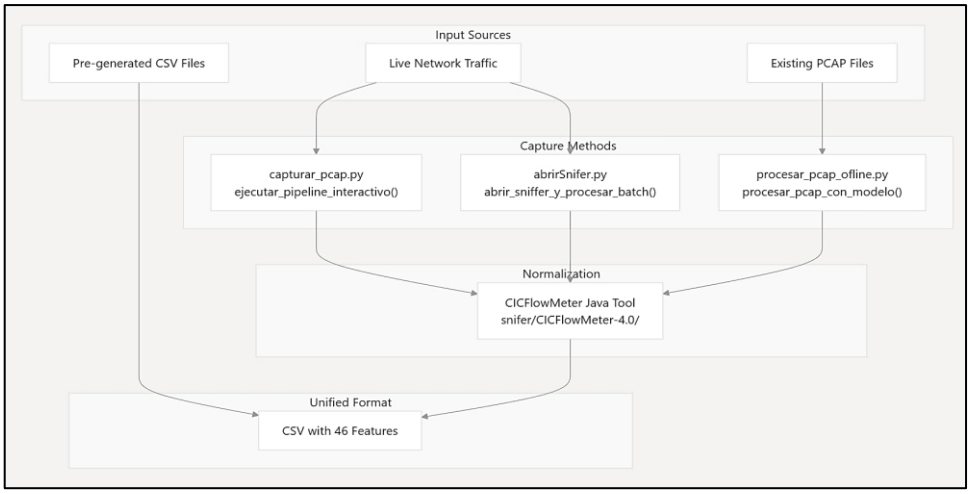


Ilustración 36 Capa de adquisición de datos

Método	Opción de menú	Módulo	Función	Flujo de trabajo
PCAP sin conexión	3	procesar_pcap_ofline.py	procesar_pcap_con_modelo()	El usuario proporciona el archivo PCAP → CICFlowMeter → CSV
CSV directo	4	N / A	N / A	El usuario proporciona un archivo CSV pregenerado
Scapy interactivo	5	capturar_pcap.py	ejecutar_pipeline_interactivo()	El usuario inicia/detiene la captura → PCAP → CICFlowMeter → CSV
Medidor de flujo CICFlowMeter por lotes	6	abrirSnifer.py	abrir_sniffer_y_procesar_batch()	Seleccionar interfaz → CICFlowMeter captura → CSV periódicamente

Tabla 35 funciones para procesar o capturar tráfico del sistema

CICFlowMeter actúa como punto de normalización central, garantizando que todos los métodos de captura produzcan conjuntos de características consistentes independientemente de la fuente de entrada.

El pipeline de ML transforma datos CSV sin procesar en predicciones a través de dos etapas secuenciales:

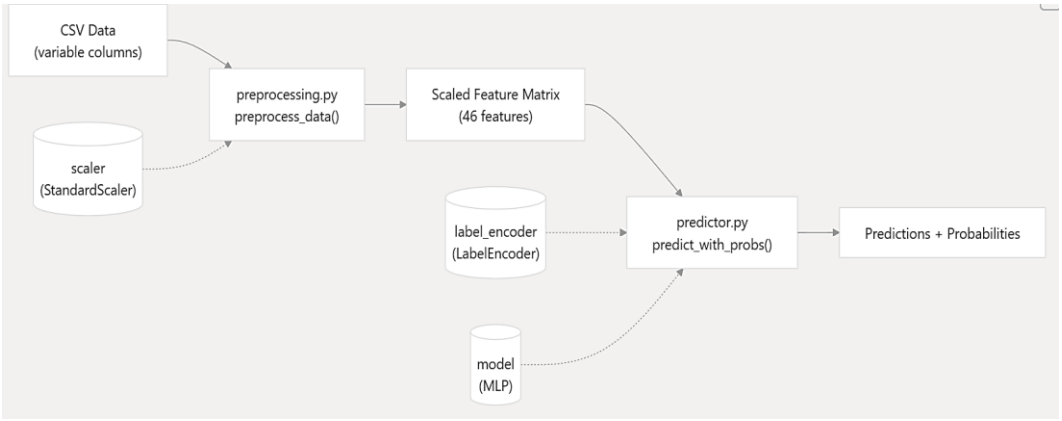


Ilustración 37 Capa Pipeline DL

El módulo preprocesamiento.py es fundamental para transformar los datos de flujo de red sin procesar en el formato requerido para la inferencia. La función preprocess_data(df, scaler) ejecuta una secuencia estricta de cuatro transformaciones críticas para garantizar la calidad y la alineación de los datos. Esto incluye la codificación One-Hot del protocolo para crear columnas binarias, la

alineación precisa de las columnas (reindex) para garantizar las 46 características esperadas, el manejo riguroso de la calidad de los datos (reemplazando valores infinitos e NaN con 0.0), y finalmente, el escalado de características mediante la transformación StandardScaler. Este módulo garantiza que la salida sea una matriz NumPy con forma (n_samples,46), lista para el modelo.

Una vez preprocesados, los datos pasan al módulo predictor.py, que actúa como el motor de inferencia y la interfaz de resultados. Su función principal, predict_with_probs, utiliza el modelo de DL y el codificador de etiquetas para generar las predicciones decodificadas (etiquetas de cadena) junto con las matrices de probabilidad sin procesar ((decoded_preds, probs)). Esta decisión de diseño es crucial, ya que el sistema no se limita a una única etiqueta de clase, sino que proporciona la distribución de probabilidad completa, la visualización de probabilidades multiclase, y una evaluación de riesgos más sofisticada basada en umbrales de probabilidad, complementada por funciones utilitarias para resumir y mostrar las predicciones detalladas por dirección IP.

La capa de salida maneja la persistencia y visualización de los resultados, exportador.py implementa la exportación CSV y PDF con gestión automática de directorios:

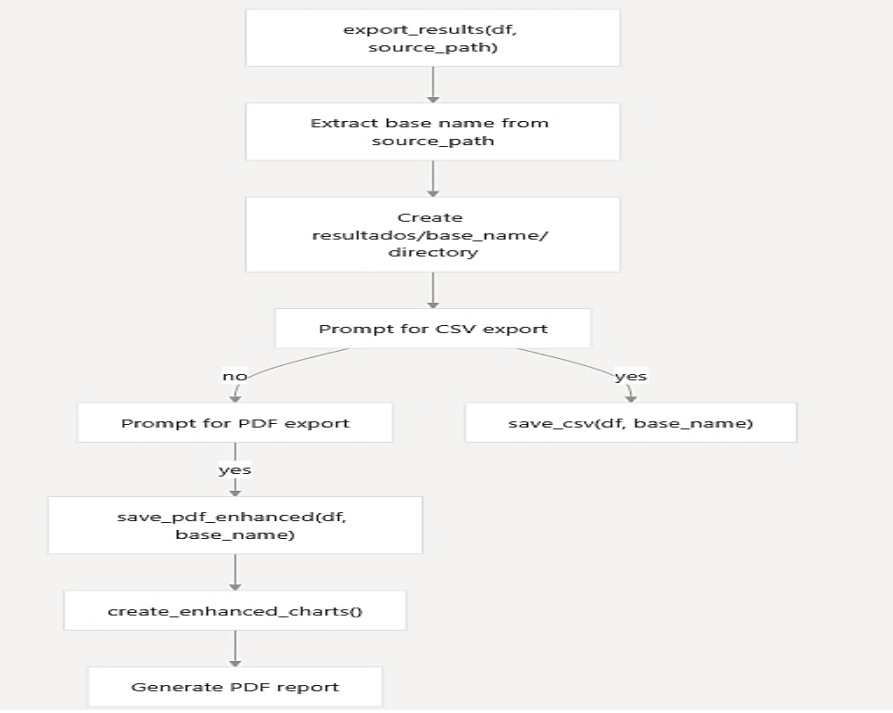


Ilustración 38 Capa de salida

El diagrama de flujo ilustra el proceso de exportación de resultados iniciado por la función export_results(df, source_path), que gestiona la persistencia y presentación de los hallazgos del análisis. El primer paso es extraer el nombre base del archivo de origen (source_path) para crear un directorio de salida específico (resultados/base_name/). Luego, el flujo se bifurca: se le pregunta al usuario si desea la exportación a CSV, y en caso afirmativo, se ejecuta la función save_csv(df, base_name). Independientemente de la respuesta al CSV, se pregunta por la exportación a PDF. Si se acepta la exportación a PDF, se llama a save_pdf_enhanced(df, base_name), lo que implica la ejecución de create_enhanced_charts() para generar todas las visualizaciones analíticas antes de ensamblar el informe PDF final.

La función **create_enhanced_charts()** genera algunas visualizaciones descritas a continuación:

N°	Visualización	Objetivo
1	Pastel de distribución	Descripción general de la distribución de clases
2	Barra de IPs principales	IP sospechosas más activas
3	Distribución de Duración de Flujos por Tipo de Tráfico	Analizar la duración de conexiones según tipo de tráfico
4	Distribución de Throughput por Tipo de Tráfico	Comparar el rendimiento de transferencia entre clases
5	Distribución de Tasa de Paquetes por Tipo de Tráfico	Analizar la velocidad de transmisión de paquetes
6	Comparación Forward/Backward - Mediana de Paquetes por Tipo	Comparar paquetes enviados vs recibidos por clase
7	Distribución de Flags TCP por Tipo de Tráfico	Identificar patrones de banderas TCP según clase

Tabla 36 Graficas del informe

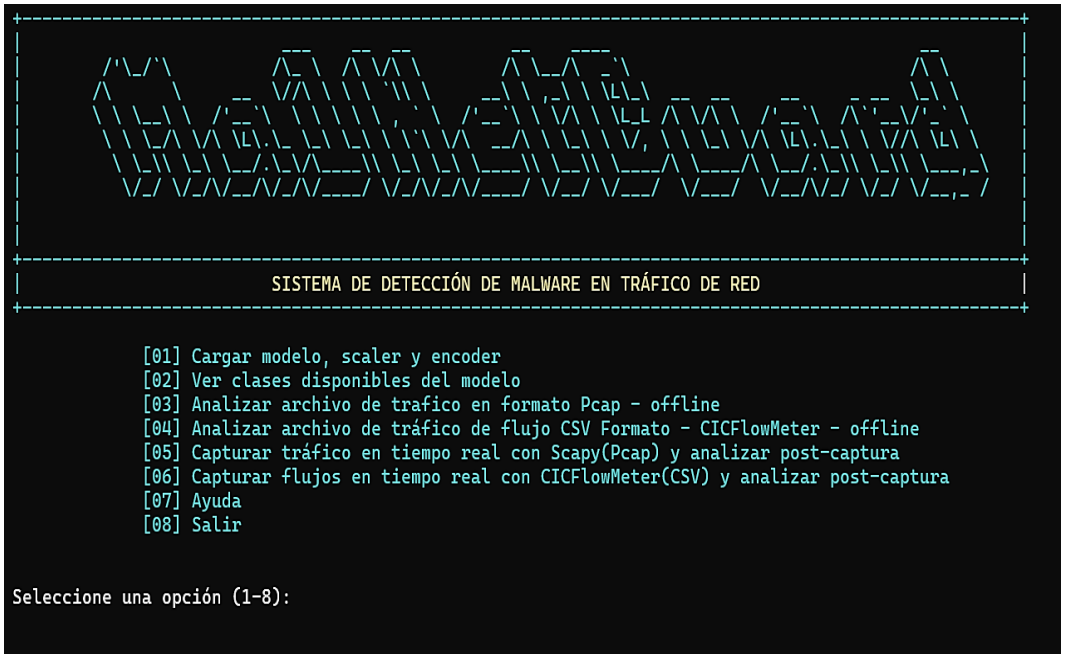


Ilustración 39 Vista del sistema ejecutándose en consola

La Ilustración muestra la interfaz principal del sistema de detección de malware desarrollado, implementado en una consola de comandos. Este entorno permite una interacción directa con las distintas funcionalidades del sistema, como la carga del modelo entrenado, el análisis de archivos de tráfico en formatos PCAP o CSV y la captura de tráfico en tiempo real mediante herramientas como Scapy y CICFlowMeter.

2.6.6.2 Planear la Implementación

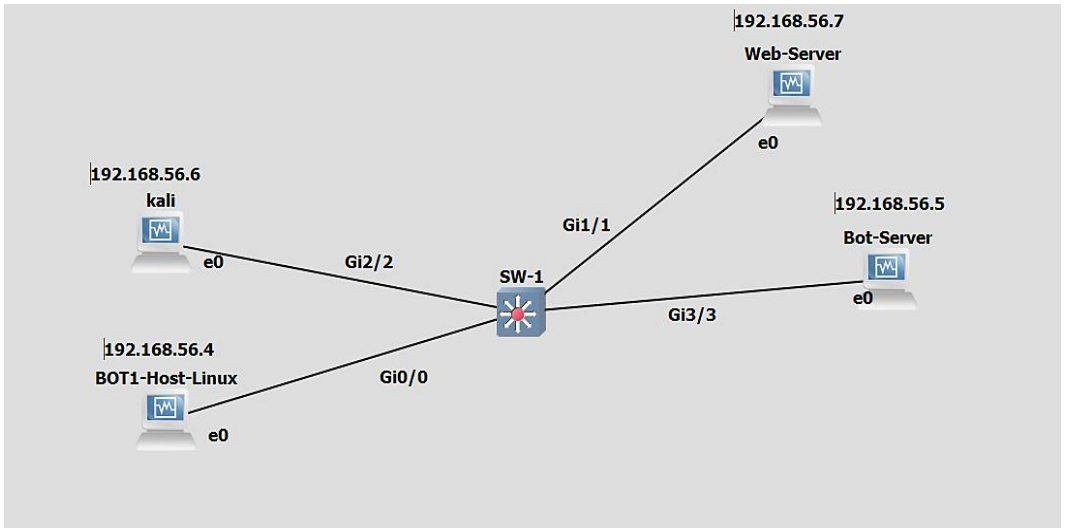


Ilustración 40 Topología de red implementada para la validación del sistema

La topología de validación se implementa sobre un único switch central, denominado SW-1, al que se conectan todos los hosts del experimento: BOT1-Host-Linux (IP 192.168.56.4, interfaz e0, puerto Gi0/0), Kali (IP 192.168.56.6, interfaz e0, puerto Gi2/2), Web-Server (IP 192.168.56.7, interfaz e0, puerto Gi1/1) y Bot-Server (IP 192.168.56.5, interfaz e0, puerto Gi3/3). Esta configuración permite generar y capturar tráfico representativo entre los distintos roles (atacante, servidor y bot) en un único dominio de conmutación, facilitando la monitorización y recolección de PCAP desde las interfaces de las máquinas virtuales.

2.6.6.3 Pruebas

Los experimentos se realizaron en un laboratorio aislado compuesto por varias máquinas virtuales interconectadas: una VM atacante (Kali), dos servidores objetivos, uno web y el otro el que controlaba al bot y uno hosts bot víctimas. Cada escenario fue preparado de forma independiente; se arrancaron y reiniciaron máquinas para garantizar un estado conocido antes de cada prueba. Las capturas de red se obtuvieron en las interfaces de interés y se almacenaron en formato PCAP para su posterior procesamiento.

2.6.6.3.1 Escenario A — Bot behavior (emulación de botnet)

El objetivo de este escenario fue emular el comportamiento típico de una botnet, incluyendo el beaconing periódico, las conexiones a un servidor de comando y control (C2) y las ráfagas de actividad que simulan exfiltración, con el fin de evaluar la sensibilidad del modelo de detección ante patrones temporales y de conexión. Para esta emulación se tomó como base el repositorio Ares (sweetsoftware/Ares), el cual permite desplegar un entorno controlado de botnet mediante un C2 Server y uno o varios bots clientes, capaces de establecer comunicación persistente y ejecutar acciones remotas [91].

Durante el experimento, el bot fue configurado para enviar mensajes periódicos del tipo “hello” cada 400 segundos al servidor C2, simulando un proceso de heartbeat o latido de control. Adicionalmente, la botnet Ares se configuró para tomar capturas de pantalla del sistema comprometido cada 200 segundos, emulando una tarea automatizada común en entornos de control remoto o exfiltración de información visual.

Paralelamente, se registró el tráfico de red (PCAP) en ambas interfaces —tanto la del bot comprometido como la del servidor C2 (ServerBot)— iniciando la captura antes del establecimiento de conexión y deteniéndola justo al finalizar la comunicación, garantizando así un registro completo del intercambio de paquetes.

Componente	Rol dentro del escenario	SO / Software empleado	Dirección IP
Bot1	Emulador del agente comprometido (bot)	Host Linux / cliente de Ares / script Python.	192.168.56.4
ServerC2 (ServerBot)	Servidor de comando y control (C2)	Ubuntu Server / Ares Server / listener Python	192.168.56.5
Herramientas de captura	Captura y análisis de tráfico	tcpdump (captura en ambas interfaces), CICflowMeter (para conversión a flujos CSV)	N/A
Repositorio de referencia	Código fuente y scripts base para botnet	sweetsoftware/Ares	Público en GitHub

Tabla 37 Escenario y componentes de bot

2.6.6.3.1.1 Descripción del comportamiento emulado

El bot Ares desplegado en 192.168.56.4 estableció conexión con el servidor C2 ubicado en 192.168.56.5 mediante un canal TCP, configurado en el puerto 8080. Durante el experimento, el bot ejecutó las siguientes acciones:

- Beaconing: envío del mensaje "hello" cada 400 segundos, simulando comunicación de estado.
- Capturas periódicas: generación de *screenshots* automáticas cada 200 segundos, almacenadas localmente y reportadas al C2.
- Ráfagas de comunicación: pequeñas transmisiones de datos correspondientes al envío de metadatos o resultados de comandos, emulando actividad de exfiltración.
- Conectividad controlada: establecimiento y cierre de conexiones cortas, simulando sesiones intermitentes características del tráfico botnet.

El patrón general de actividad se resume en la siguiente tabla:

Evento	Intervalo / Condición	Descripción
Hello beacon	Cada 400 s	Mensaje enviado al C2 como señal de vida
Captura de pantalla	Cada 200 s	Imagen del escritorio obtenida por el bot
Inicio de captura PCAP	Antes de conexión	Captura en ambas interfaces (bot y servidor)
Fin de captura PCAP	Tras finalizar la conexión	Detención simultánea de captura

Tabla 38 Patrón general para el tráfico bot

Este diseño permitió analizar el tráfico generado por comportamientos automatizados de bots, observando los efectos de las acciones periódicas (beacons) y las tareas programadas (capturas) sobre los flujos de red.

2.6.6.3.2 Escenario B — Infiltration (exploit controlado con Metasploit)

El objetivo de este escenario fue ejecutar un exploit controlado desde una máquina atacante (Kali) contra una máquina víctima (Linux) para observar las fases clásicas de una infiltración: reconocimiento, entrega del payload, establecimiento de la sesión reversa (Meterpreter) y actividad post-explotación mínima —todo ello con tráfico real registrado para su análisis. El propósito experimental fue obtener registros de red y tiempos sincronizados con los eventos del exploit para poder etiquetar y extraer correctamente los flujos correspondientes a la fase de infiltración y evaluar la capacidad del modelo para detectar este tipo de actividad.

2.6.6.3.2.1 Entorno, roles y medidas de seguridad

El experimento se realizó en una red de laboratorio aislada 192.168.56.0/24. La máquina atacante (Kali) tuvo la dirección 192.168.56.6 y la víctima Linux la dirección 192.168.56.4; el puerto configurado para la sesión Meterpreter fue 4444. Antes de cada ejecución se tomó un snapshot (instantánea) de las máquinas virtuales para permitir una reversión rápida del sistema a un estado limpio en caso de ser

necesario; esta medida es obligatoria por razones de seguridad y reproducibilidad. En la fase de entrega del payload se empleó msfvenom para generar un binario ELF backdoor.elf de tipo linux/x64/meterpreter/reverse_tcp con LHOST=192.168.56.6 y LPORT=4444; el binario se transfirió a la víctima mediante un servidor HTTP simple para la prueba y se ejecutó con permisos de ejecución adecuados. Para garantizar la integridad de los registros y la sincronía temporal entre eventos de Metasploit y los paquetes de red.

Con el fin de tener una visión completa del intercambio, se realizaron capturas PCAP en ambas máquinas. Las grabaciones se iniciaron antes del inicio del procedimiento de entrega y se detuvieron luego de concluir la actividad post-explotación, a fin de asegurar que todas las etapas —escaneo, exploit, handshake y sesiones reversas— quedaran registradas.

De manera resumida, las terminales en Kali necesarias fueron: (1) una terminal ejecutando sudo tcpdump -i eth0 -s 0 -n -w captura_completa.pcap para la captura continua; (2) una terminal con python3 -m http.server 8080 para transferir el payload por HTTP; y (3) una terminal con msfconsole corriendo el exploit/multi/handler a la espera de la conexión reversa. En la máquina víctima bastó una terminal para descargar y ejecutar el payload (por wget o scp y chmod +x ./backdoor.elf).

Elemento	Valor / Descripción
Atacante	Kali Linux — 192.168.56.6
Víctima	Linux — 192.168.56.4
Payload generado	linux/x64/meterpreter/reverse_tcp (msfvenom → backdoor.elf)
LHOST / LPORT	192.168.56.6: 4444
Capturas PCAP	Captura completa en Kali y captura en la interfaz de la víctima (inicio antes del exploit — fin tras actividad post)

Tabla 39 resumen de componentes Infiltration

La fase de reconocimiento consistió en la ejecución de diez comandos Nmap seleccionados (ver tabla) para simular técnicas de descubrimiento y fingerprinting. Se aplicaron escaneos SYN stealth (-sS) sobre rangos --top-ports, escaneos UDP (-sU), detección de versión (-sV) y escaneos agresivos (-A), además de variantes operativas (-Pn, -T4) y scripts NSE (--script discovery, --script broadcast).

Estos comandos se eligieron buscando un balance entre cobertura, sigilo y obtención de información útil: --top-ports focaliza el análisis, -sS reduce la visibilidad, -sV/-A permiten fingerprinting y las variantes/ scripts cubren casos sin respuesta a pings y detección de servicios UDP críticos.

N	Comando NMAP	Tipo de Escaneo	Puertos Escaneados	Descripción
1	nmap -sS -v --top-ports 200 192.168.56.0/24	SYN Stealth Scan (-sS)	Top 200 TCP	Envía SYN para detectar puertos TCP abiertos sin completar el handshake; usado para descubrimiento sigiloso de servicios.
2	nmap -sU -v --top-ports 50 192.168.56.0/24	UDP Scan (-sU)	Top 50 UDP	Envía datagramas UDP para identificar servicios no-TCP; usado porque muchos servicios críticos usan UDP.
3	nmap -sV -v --top-ports 300 192.168.56.0/24	Detección de versión (-sV)	Top 300 TCP	Interroga servicios para identificar versiones y banners; usado para fingerprinting y evaluación de vulnerabilidades.
4	nmap -A -v --top-ports 300 192.168.56.0/24	Escaneo agresivo (-A)	Top 300 TCP	Combina OS fingerprinting, -sV y NSE scripts para máxima información; usado para auditoría rápida y recopilación detallada.
5	nmap -Pn -sS -v --top-ports 250 192.168.56.0/24	SYN Stealth sin ping (-Pn -sS)	Top 250 TCP	Omite la detección de hosts (no ping/ARP) y realiza -sS; usado cuando ICMP/ARP puede estar bloqueado o para forzar escaneo.
6	nmap -sS -T4 -v --top-ports 100 192.168.56.0/24	SYN Stealth con temporización T4 (-T4)	Top 100 TCP	Ajusta temporizadores para escanear más rápido; usado para acelerar en redes confiables a costa de precisión temporal.

N	Comando NMAP	Tipo de Escaneo	Puertos Escaneados	Descripción
7	nmap --script discovery -v --top-ports 200 192.168.56.0/24	NSE — discovery scripts	Top 200 TCP	Ejecuta scripts de descubrimiento automatizados (NSE) para obtener más metadatos; usado para enriquecer el reconocimiento.
8	nmap --script broadcast -v 192.168.56.0/24	NSE — broadcast scripts	N/A (broadcast)	Envía probes broadcast para descubrir servicios DHCP/NetBIOS/etc. en la LAN; usado para encontrar servidores/routers automáticamente.
9	nmap -sS -v --top-ports 400 192.168.56.0/24	SYN Stealth Scan (-sS)	Top 400 TCP	Igual que -sS pero con rango ampliado; usado para detectar servicios en puertos menos comunes ampliando la cobertura.
10	nmap -sS -sU -v -p U:53,111,137,T:21-25,80,443 192.168.56.0/24	Escaneo combinado TCP SYN + UDP	UDP 53,111,137; TCP 21-25,80,443	Escanea simultáneamente puertos TCP y UDP específicos críticos; usado para verificar servicios esenciales en la red.

Tabla 40 comandos NMAP ejecutados

2.6.6.3.3 Escenario C — Web attacks (fuerza bruta HTTP)

El objetivo de este escenario fue generar ataques de fuerza bruta automatizados contra el servidor web de laboratorio (ServerWeb) con el fin de evaluar la detección de patrones anómalos de tráfico HTTP. Se buscó simular intentos de autenticación masiva, requests repetitivas y la generación de flujos con altas tasas de paquetes hacia el mismo endpoint, para luego etiquetar los flujos resultantes como WebAttacks y analizar la capacidad del modelo de Deep Learning para detectar ataques de fuerza bruta en tráfico de red real.

2.6.6.3.3.1 Entorno y herramientas

El experimento se ejecutó en la red de laboratorio aislada 192.168.56.0/24 utilizando dos máquinas atacantes (Kali 192.168.56.6 y Linux 192.168.56.4) y un

servidor web objetivo 192.168.56.7. Se empleó un script Python con Selenium + Firefox (Geckodriver) en modo *headless* para automatizar intentos de login contra el formulario DVWA (/dvwa/login.php), cargando listas locales de usuarios y contraseñas y aplicando un retardo de 0.1 s entre peticiones. El uso de dos atacantes permitió evaluar simultaneidad y pequeñas variaciones en los diccionarios; Kali realizó 1 413 intentos y la máquina Linux 1 611. Durante la prueba se capturó todo el tráfico con tcpdump y se guardó en archivos PCAP; los filtros aplicados limitaron la captura al flujo HTTP entre cada atacante y el servidor (por ejemplo host 192.168.56.6 and host 192.168.56.7 and tcp port 80).

```
1: Input: TARGET (URL de login), USERS_FILE (archivo de usuarios), PASSWORDS_FILE (archivo de contraseñas)
2: Output: Credenciales válidas encontradas, total de intentos realizados
3: Configuración:
4:     - MAX_TOTAL_ATTEMPTS = 1500 (límite máximo de intentos)
5:     - DELAY_BETWEEN_ATTEMPTS = 0.1 segundos
6:     - PAGE_LOAD_WAIT = 0.5 segundos
7:     - HEADLESS = True (navegador sin interfaz gráfica)
8:     - GECKO_PATH = ruta del driver geckodriver
9: Funciones auxiliares:
10:    load_list(path): Cargar archivo de texto línea por línea, devolver lista
11:    create_driver(): Inicializar Firefox con opciones headless y timeouts
12:    find_login_elements(driver): Localizar campos username, password y botón submit
13:    check_success(driver): Verificar si login fue exitoso (URL contiene "index.php" o "logout")
14: Algoritmo principal:
15:    users = load_list(USERS_FILE)
16:    passwords = load_list(PASSWORDS_FILE)
17:    if users vacío or passwords vacío:
18:        Mostrar error y terminar
19:    driver = create_driver()
20:    total = 0
21:    success_found = False
22:    for cada usuario u in users:
23:        if success_found or total >= MAX_TOTAL_ATTEMPTS:
24:            break
25:        for cada contraseña p in passwords:
26:            if success_found or total >= MAX_TOTAL_ATTEMPTS:
```

```
27:             break
28:         try:
29:             driver.get(TARGET)
30:             Esperar PAGE_LOAD_WAIT segundos
31:             user_el, pass_el, submit_el = find_login_elements(driver)
32:             Limpiar campos de entrada
33:             user_el.send_keys(u)
34:             pass_el.send_keys(p)
35:             submit_el.click()
36:             total += 1
37:             if total % 10 == 0:
38:                 Mostrar progreso (intentos realizados)
39:             if check_success(driver):
40:                 Mostrar credenciales válidas: u:p
41:                 success_found = True
42:                 break
43:             Esperar DELAY_BETWEEN_ATTEMPTS segundos
44:         except Exception:
45:             Registrar error y continuar
46:     end for
47: end for
48: driver.quit()
49: Mostrar total de intentos realizados
50: Salida:
51: - Si éxito: credenciales válidas y número de intento
52: - Total de combinaciones probadas
```

Ilustración 41 Algoritmo de automatización de intentos de login en una web vulnerable usando Selenium y Firefox en modo headless..

Para simular condiciones realistas, se aplicaron diferentes velocidades de ataque: ráfagas rápidas de aproximadamente 100 requests por segundo y ataques lentos de 1 request por segundo, variando así la carga sobre el servidor web y permitiendo evaluar la sensibilidad del modelo ante diferentes tasas de flujo. Además, se configuraron headers HTTP personalizados incluyendo User-Agent, Referer y cookies simuladas para generar tráfico más variado y realista. Se documentó el número total de requests realizados en cada atacante y la duración de la prueba, lo que permitió sincronizar los eventos y determinar la ventana temporal de fuerza bruta a etiquetar en el dataset.

Parámetro	Descripción	Valor / Configuración
Red de laboratorio	Subred donde se ejecutó el ataque	192.168.56.0/24
Servidor web objetivo	Servidor DVWA	192.168.56.10
Atacantes	Máquinas que ejecutaron el script	Kali (192.168.56.107) y Linux (192.168.56.104)
Herramienta principal	Script automatizado con Selenium y Firefox headless	bruteforceselenium.py
Puerto objetivo	Puerto TCP usado en los intentos de login HTTP	80
Cantidad total de intentos	Número total de combinaciones de usuario/contraseña enviadas	Kali: 1413 / Linux: 1611
Delay entre intentos	Tiempo de espera entre cada request	0.1 s
Velocidades simuladas	Rápido (100 req/s) y lento (1 req/s)	Variable
Headers HTTP	User-Agent, Referer, Cookies personalizadas	Personalizados
Método de captura	Captura completa de tráfico TCP/HTTP con tcpdump	Filtros por IP y puerto

Tabla 41 Parámetros de configuración del escenario WebAttacks

2.6.6.3.4 Archivos PCAP generados durante las pruebas

ID	Capturas (nombre PCAP)	Tipo de ataque	Origen (VM)	Duración	N flows	N Pcaps
2	captureBotSERVER_enp0s3.pcap	Botnet	ServerBot	2 horas	208	5.543
3	captura_infilteration4HostInfectado.pcap	Infiltration (Metasploit)	Ubuntu Server	1 horas	65,451	416.796
5	capture_BruteForceS_eth0.pcap	Web brute force	Server web	1	135	65.313

Tabla 42 Archivos Pcap generados

2.7 Arquitectura del sistema

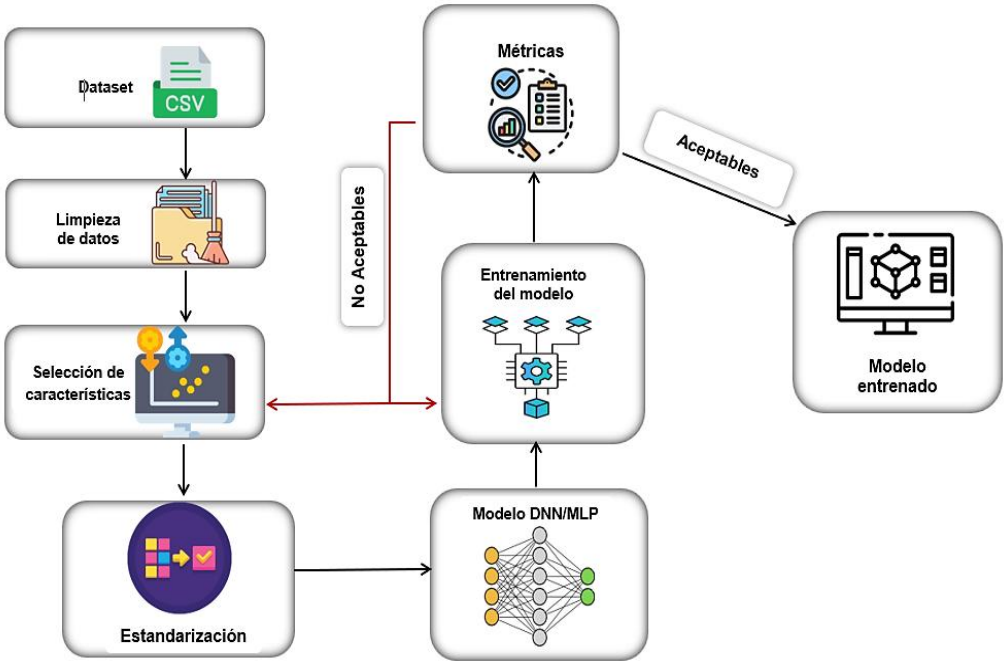


Ilustración 42 Arquitectura del proceso de creación del modelo de Deep Learnig

La figura muestra el flujo seguido para la construcción del modelo de detección de malware mediante técnicas de Deep Learning. El proceso inicia con la obtención del *dataset* improved CSE-CIC-IDS2018, el cual pasa por una fase de limpieza y depuración de datos con el fin de eliminar registros inconsistentes o redundantes. Posteriormente, se realiza la selección de características más relevantes utilizando métodos estadísticos y de importancia de atributos, para conservar únicamente aquellas que aportan información significativa al modelo. Los datos seleccionados se someten a un proceso de estandarización que garantiza que todas las variables presenten una escala homogénea, facilitando el aprendizaje del modelo.

En la siguiente etapa se define la arquitectura de la red neuronal profunda (DNN/MLP), la cual es entrenada con los datos procesados para optimizar su capacidad de clasificación. Las métricas de desempeño obtenidas (precisión, recall, F1-score, entre otras) permiten determinar si el modelo alcanza resultados aceptables. En caso contrario, se ajustan las características o los hiperparámetros y se repite el ciclo hasta lograr un rendimiento óptimo. Este enfoque iterativo asegura un proceso de refinamiento continuo, orientado a mejorar la detección.

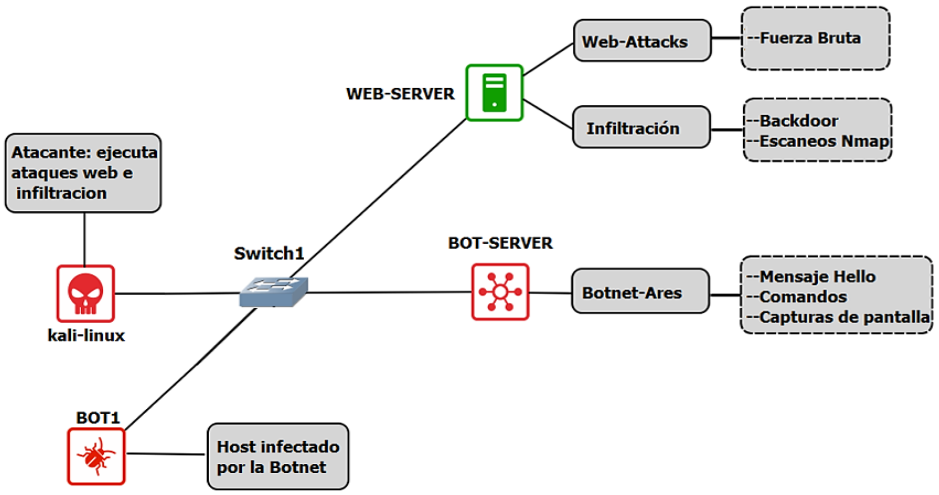


Ilustración 43 Topología de red para la simulación de ataques y detección de tráfico malicioso.

La arquitectura de red presentada muestra al nodo kali-linux actuando como Atacante, el cual ejecuta ataques web e infiltracion. Estos ataques están dirigidos contra el WEB-SERVER y se dividen en Web-Attacks (como Fuerza Bruta) e Infiltración (incluyendo Backdoor y Escaneos Nmap). De forma complementaria, el BOT-SERVER administra la Botnet-Ares. Esta botnet se comunica con BOT1, identificado como un Host infectado por la Botnet. Las acciones de la botnet incluyen el envío de Mensaje Hello, Comandos y Capturas de pantalla. Todos los nodos (kali-linux, WEB-SERVER, BOT-SERVER y BOT1) están interconectados a través del Switch1.

2.8 Diagramas de casos de uso

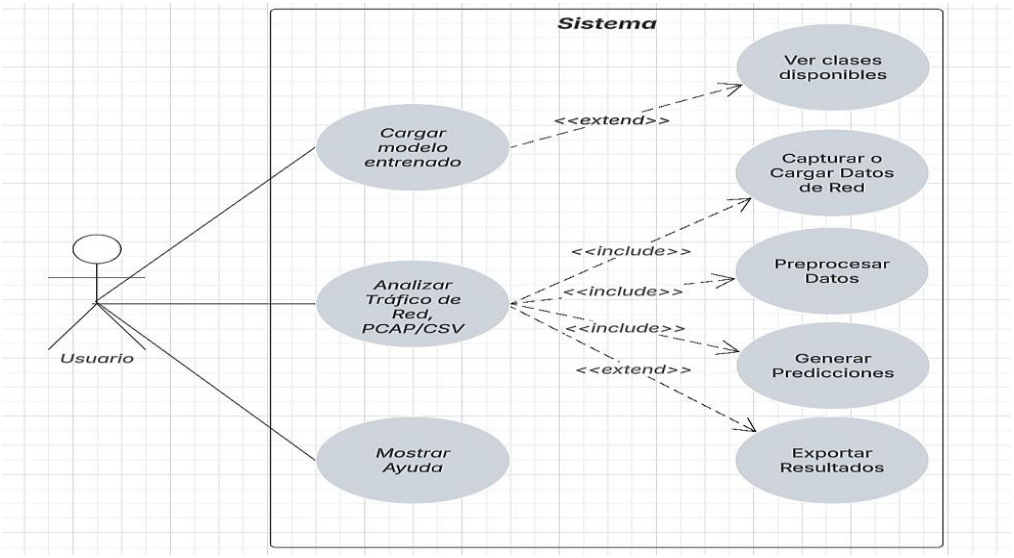


Ilustración 44 Diagrama de casos de uso del sistema de detección y clasificación de malware en tráfico de red

2.9 Resultados

2.9.1 Resultados — Escenario A (Bot)

El modelo MLP de detección de intrusiones basado en Deep Learning, demostró capacidad de detección específica para patrones de exfiltración masiva de datos en el entorno controlado de red con servidor botnet y hosts infectados. El análisis detallado de las gráficas del reporte de captura de tráfico en la interfaz enp0s3 del servidor botnet, complementado con la inspección a nivel de paquetes individuales, revela que las limitaciones observadas no constituyen deficiencias del modelo en sí, sino que reflejan las características inherentes del dataset CIC-CSE-IDS2018 con el cual fue entrenado, que captura principalmente botnets con actividad de exfiltración periódica de screenshots y no incluye representación suficiente y representativas de comunicaciones C&C ligeras de mantenimiento.

2.9.1.1 Distribución de Predicciones

```
--------------------------------------------------------------------------------

 CSV guardado temporalmente en: C:\Users\josue\AppData\Local\Temp\tmp3xez36j7\captureBotSERVER_enp0s3.csv
7/7 [==============================] - 0s 3ms/step

 Resumen de Predicciones:
   Benign: 137 muestras
   Bot: 69 muestras
   Infilteration: 2 muestras
   webAttacks: 0 muestras

 Predicción de tráfico (muestras):
+--------------------+---------+--------------------+---------+-------------+--------------------------+--------------------------+
|     Src IP         | S.Port  |     Dst IP         | D.Port  | Protocolo   | Timestamp                | Predicción (con % máx)   |
+--------------------+---------+--------------------+---------+-------------+--------------------------+--------------------------+
| 192.168.56.4       | 55676   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:52:06.142465 | Bot                100.00% |
| 192.168.56.6       | 51186   | 192.168.56.5       | 8080    | 6           | 2025-10-25 02:27:42.531806 | Bot                100.00% |
| 192.168.56.6       | 58200   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:11:05.631858 | Bot                100.00% |
| 192.168.56.6       | 37458   | 192.168.56.5       | 8080    | 6           | 2025-10-25 02:04:22.805921 | Bot                100.00% |
| 192.168.56.6       | 42216   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:04:28.428032 | Bot                100.00% |
| 192.168.56.6       | 51942   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:01:06.922537 | Bot                100.00% |
| 192.168.56.6       | 39406   | 192.168.56.5       | 8080    | 6           | 2025-10-25 02:51:03.458227 | Bot                100.00% |
| 192.168.56.4       | 40164   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:25:25.620650 | Bot                100.00% |
| 192.168.56.6       | 39988   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:21:07.001264 | Bot                100.00% |
| 192.168.56.6       | 37958   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:37:47.125272 | Bot                100.00% |
| 192.168.56.6       | 34298   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:54:28.364435 | Bot                100.00% |
| 192.168.56.4       | 55388   | 192.168.56.5       | 8080    | 6           | 2025-10-25 02:02:02.674360 | Bot                100.00% |
| 192.168.56.4       | 46436   | 192.168.56.5       | 8080    | 6           | 2025-10-25 04:05:26.488810 | Bot                100.00% |
| 192.168.56.4       | 57652   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:58:46.318036 | Bot                100.00% |
| 192.168.56.6       | 40858   | 192.168.56.5       | 8080    | 6           | 2025-10-25 02:24:22.292505 | Bot                100.00% |
| 192.168.56.6       | 58140   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:41:08.029486 | Bot                100.00% |
| 192.168.56.6       | 50042   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:44:27.745542 | Bot                100.00% |
| 192.168.56.4       | 52294   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:38:46.085393 | Bot                100.00% |
| 192.168.56.4       | 46942   | 192.168.56.5       | 8080    | 6           | 2025-10-25 02:22:03.714452 | Bot                100.00% |
| 192.168.56.4       | 55466   | 192.168.56.5       | 8080    | 6           | 2025-10-25 02:45:24.403711 | Bot                100.00% |
| 192.168.56.4       | 52440   | 192.168.56.5       | 8080    | 6           | 2025-10-25 02:18:43.581950 | Bot                100.00% |
| 192.168.56.4       | 51162   | 192.168.56.5       | 8080    | 6           | 2025-10-25 02:15:23.529173 | Bot                100.00% |
| 192.168.56.6       | 51968   | 192.168.56.5       | 8080    | 6           | 2025-10-25 02:44:23.068731 | Bot                100.00% |
| 192.168.56.4       | 33798   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:32:05.792696 | Bot                100.00% |
| 192.168.56.6       | 52712   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:14:26.500996 | Bot                100.00% |
| 192.168.56.6       | 42038   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:31:07.008978 | Bot                100.00% |
| 192.168.56.4       | 39558   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:22:05.522256 | Bot                100.00% |
| 192.168.56.6       | 52244   | 192.168.56.5       | 8080    | 6           | 2025-10-25 02:07:42.809987 | Bot                100.00% |
| 192.168.56.4       | 37306   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:55:26.232173 | Bot                100.00% |
| 192.168.56.4       | 52048   | 192.168.56.5       | 8080    | 6           | 2025-10-25 03:18:45.438977 | Bot                100.00% |
+--------------------+---------+--------------------+---------+-------------+--------------------------+--------------------------+
```

Ilustración 45 Ejemplos de predicciones del escenario Bot en el sistema

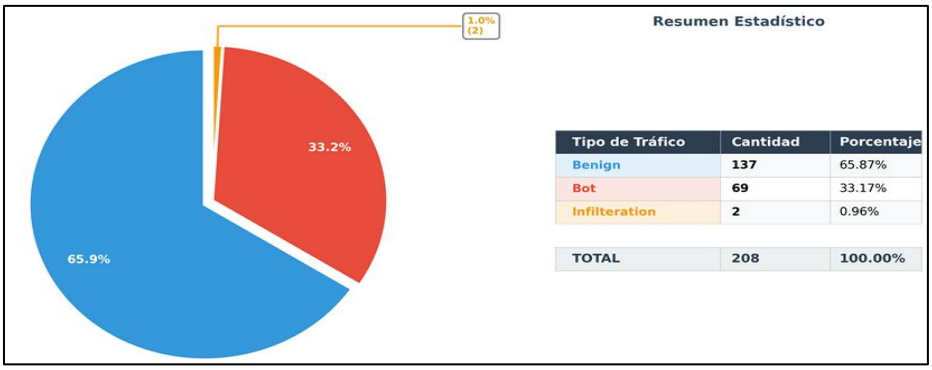


Ilustración 46 Distribución de predicciones

La gráfica de distribución muestra que, de 208 flujos analizados, el modelo clasificó 137 conexiones como benignas (65.87%), 69 como tráfico botnet (33.17%) y 2 como infiltración (0.96%). Esta distribución refleja la naturaleza específica del tráfico botnet presente en CIC-CSE-IDS2018, donde las muestras de ataque Bot corresponden predominantemente a sesiones de exfiltración de datos con alto throughput y transferencias de archivos grandes. Las 2 clasificaciones de Infiltration representan falsos positivos donde el modelo aplicó umbrales aprendidos del dataset que asocian tasas extremas de paquetes (42,241 pkt/s) con ataques de infiltración, aunque en este escenario específico no correspondían a actividad maliciosa real. Es importante destacar que el modelo funcionó exactamente como fue entrenado: detectando únicamente las 69 conexiones donde se subieron screenshots desde los hosts comprometidos al servidor C&C, que son el tipo de transferencias botnet representadas en CIC-CSE-IDS2018.

2.9.1.2 Análisis de Comunicaciones C&C Ligeras a Nivel de Paquetes: Mensajes "Hello" y "Report"

La inspección detallada de los paquetes clasificados como "Benign" revela la naturaleza de la comunicación botnet de mantenimiento que el dataset de entrenamiento no capturó adecuadamente. Los mensajes "hello" presentan el siguiente patrón característico a nivel de paquetes:

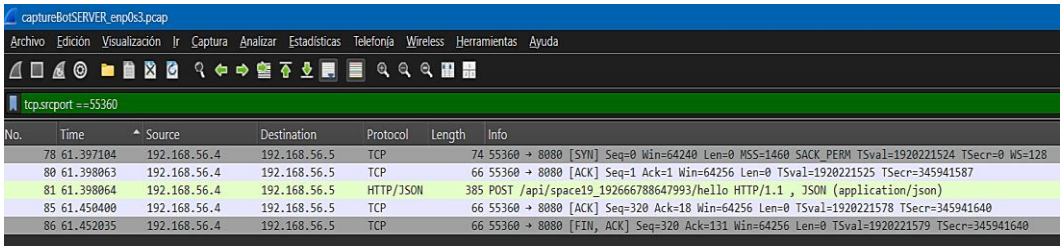


Ilustración 47 ejemplo de mensaje hello de la Botnet

Este patrón de 5 paquetes constituye una transacción completa de heartbeat/keepalive extremadamente eficiente, con tamaño total de flujo de apenas ~385 bytes (incluyendo headers HTTP/JSON) y duración inferior a 55 milisegundos. Los mensajes "hello" utilizan JSON como formato de serialización, común en APIs REST pero también utilizado por botnets contemporáneas para comunicaciones estructuradas con sus servidores C&C.

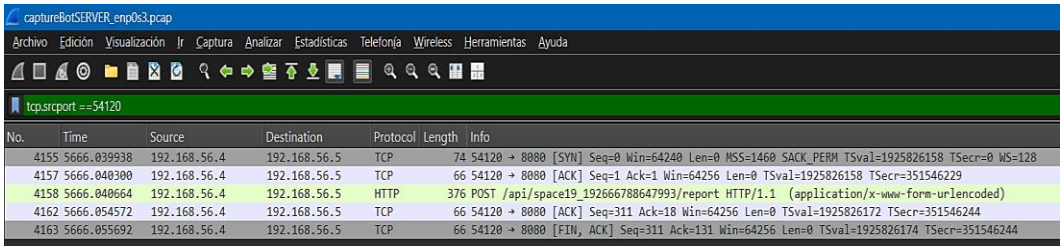


Ilustración 48 Ilustración 39 ejemplo de mensaje hello de la Botnet

Segundo ejemplo de mensaje "report" (paquetes 4155-4163) con características idénticas: 5 paquetes, 310 bytes de payload, mismo endpoint /api/.../report, duración de ~15 ms. Esta regularidad temporal y consistencia estructural son firmas claras de comunicación automatizada botnet, pero al no generar throughput elevado ni transferir archivos grandes, no activan los patrones que el modelo aprendió de CIC-CSE-IDS2018.

2.9.1.3 Análisis de Tráfico Bot a Nivel de Paquetes: Exfiltración de Screenshots

En contraste, el tráfico correctamente clasificado como "Bot" presenta características dramáticamente diferentes que coinciden con los perfiles de exfiltración presentes en CIC-CSE-IDS2018:

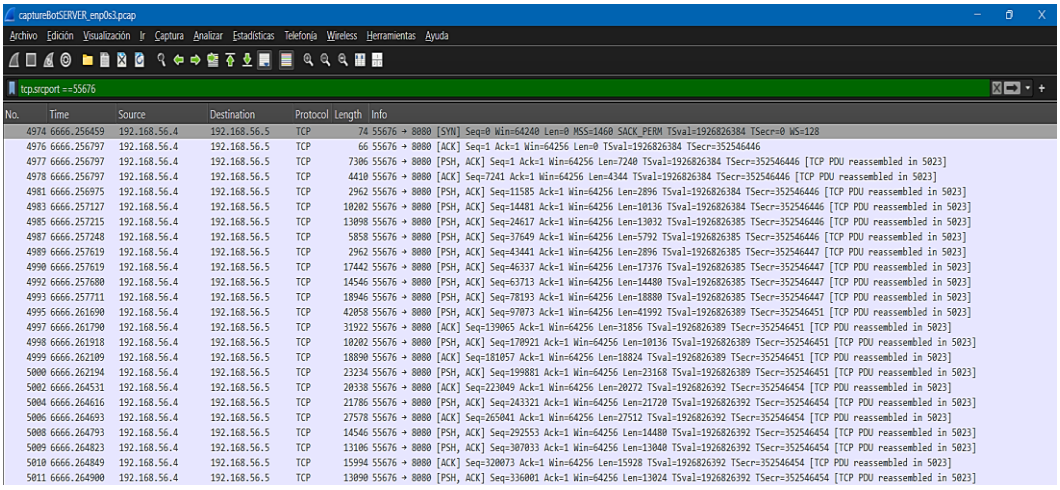


Ilustración 49 Ejemplo 1 de exfiltración de screenshot clasificado como Bot

Este patrón de ráfaga de alto volumen con múltiples segmentos TCP PSH es exactamente el tipo de comportamiento botnet que CIC-CSE-IDS2018 captura extensivamente en sus muestras de Bot, donde las botnets realizan exfiltración de datos robados o descargas de *payloads* maliciosos. El modelo aprendió correctamente que estas características como Flow Bytes/s > 43,000,000 bytes/s, Flow Packets/s > 4,000 pkt/s, PSH Flag Count > 30 corresponden inequívocamente a actividad maliciosa.

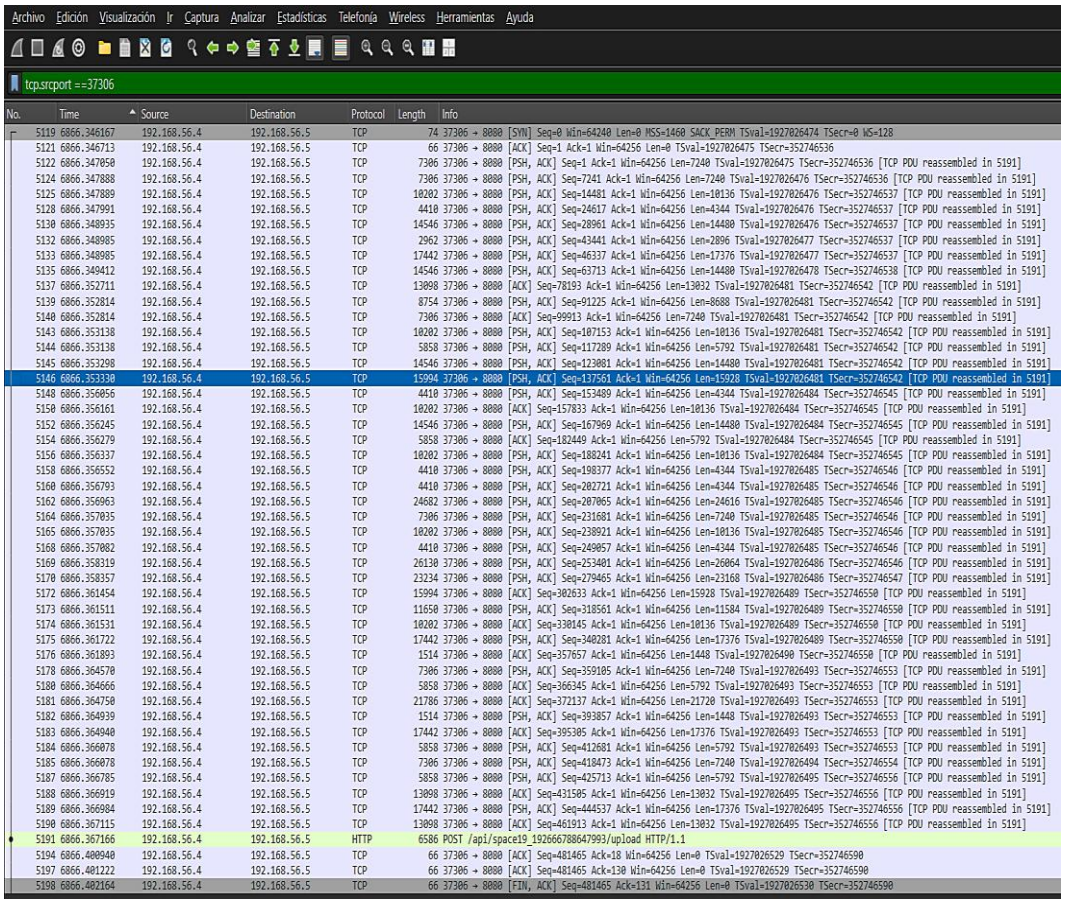


Ilustración 50 Ejemplo 2 de exfiltración de screenshot clasificado como Bot

El segundo ejemplo de Bot (paquetes 5119-5198) muestra consistencia en el patrón de exfiltración: 79 paquetes transmitiendo 475,001 bytes en ~34 ms, mismo endpoint /upload, misma estructura de reassembly, misma distribución de flags PSH (48 de 79 paquetes). Esta repetibilidad del patrón de screenshot cada ~200 segundos (3.3 minutos) demuestra la periodicidad programada de la exfiltración, característica también presente en el dataset CIC-CSE-IDS2018 donde los ataques ocurren en tiempo definido.

Característica	Hello/Report (Benign)	Screenshot (Bot)	Ratio
Número de paquetes	5 paquetes	44-79 paquetes	8.8x - 15.8x más
Duración total	15-55 ms	35-50 ms	Similar
Bytes transferidos	310-385 bytes	475,001 bytes	1,234x - 1,532x más
Throughput	~6-25 KB/s	43,200 KB/s	1,700x - 7,200x más
Flags PSH	1 PSH	30-48 PSH	30x - 48x más
Endpoint	/helloo/report	/upload	Diferentes
Formato contenido	JSON o form-urlencoded	Imagen codificada	Diferentes
Fragmentación TCP	1 segmento	40+ segmentos	Masiva
Tasa de paquetes	~90-330 pkt/s	4,000+ pkt/s	12x - 44x más
Propósito	Mantenimiento C&C	Exfiltración de datos	Diferentes

Tabla 43 Comparación Estructural de Hello/Report vs Screenshot

Esta tabla cuantifica la diferencia de tres órdenes de magnitud entre ambos tipos de comunicación botnet, explicando por qué el modelo entrenado con CIC-CSE-IDS2018 solo detecta uno de ellos. El dataset de entrenamiento simplemente no incluye ejemplos suficientes de comunicaciones C&C ligeras etiquetadas como Bot, conteniendo principalmente muestras de ataques activos con alto consumo de ancho de banda.

2.9.1.4 Distribución de Duración de Flujos por Tipo de Tráfico

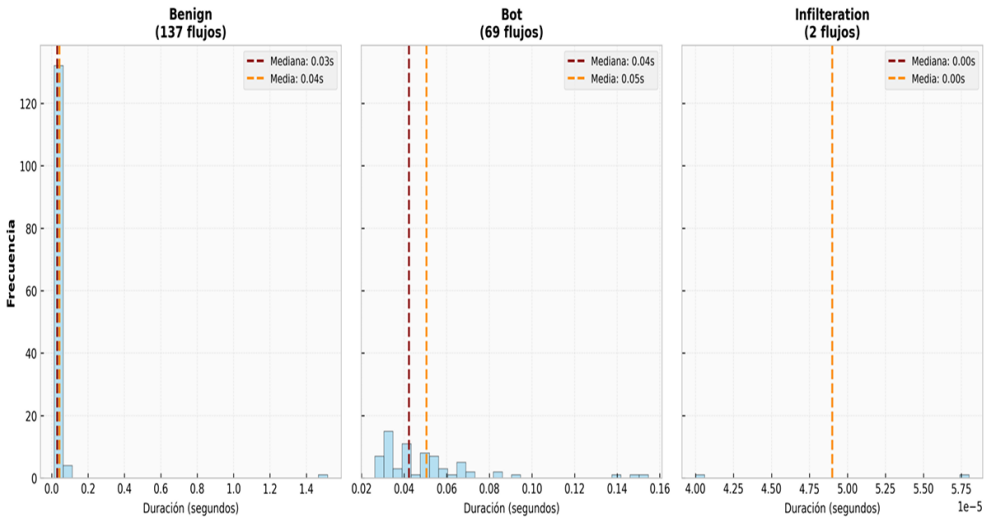


Ilustración 51 Distribución de Duración de Flujos por Tipo de Tráfico

Este histograma comparativo muestra las distribuciones de duración temporal de las conexiones de red para cada clase predicha por el modelo. El tráfico clasificado como Benign presenta una mediana de 0.03 segundos y media de 0.04 segundos, con una distribución concentrada en duraciones muy cortas que corresponden a

transacciones HTTP simples como los mensajes "hello" y "report" que completan su ciclo en 15-55 milisegundos. El tráfico Bot muestra valores ligeramente superiores con mediana de 0.04 segundos y media de 0.05 segundos, reflejando las conexiones de exfiltración de screenshots que requieren 35-50 milisegundos para transmitir ~475 KB de datos fragmentados en múltiples segmentos TCP. Las 2 clasificaciones de Infiltration, que constituyen falsos positivos, presentan valores prácticamente nulos (0.00 segundos en mediana y media), indicando flujos extremadamente breves que el modelo interpretó erróneamente como intentos de reconocimiento rápido o escaneos de puertos. Estos falsos positivos ocurren porque el modelo aprendió del dataset CIC-CSE-IDS2018 que duraciones cercanas a cero combinadas con otras métricas extremas corresponden a infiltración, pero en este escenario específico representan artefactos de captura de tráfico o patrones anómalos no maliciosos.

2.9.1.5 Distribución de Throughput por Tipo de Tráfico

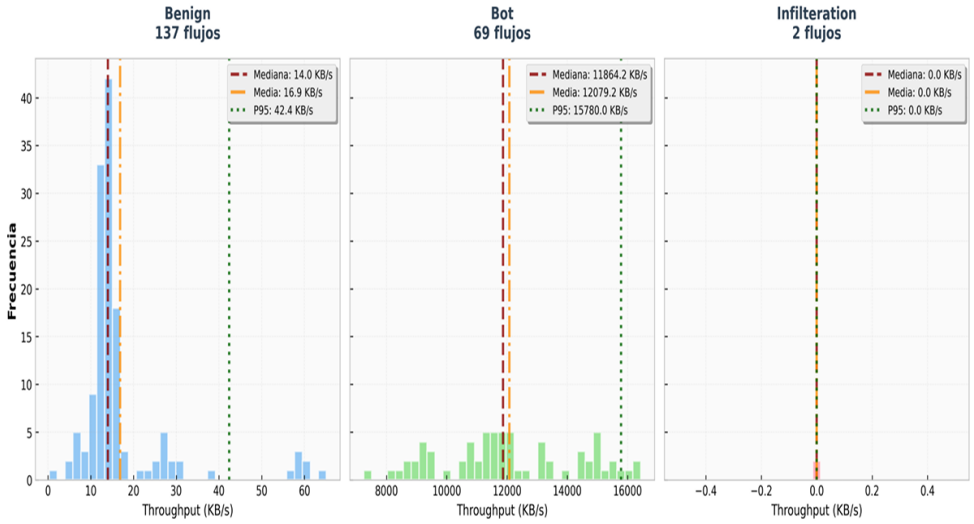


Ilustración 52 Distribución de Throughput por Tipo de Tráfico

Esta gráfica revela la característica más discriminante que el modelo utiliza para clasificar tráfico botnet. El tráfico Benign exhibe una mediana de 14.0 KB/s y percentil 95 de 42.4 KB/s, valores consistentes con comunicaciones HTTP ligeras como los mensajes "hello" (~6-25 KB/s) y "report" (~12-25 KB/s) que el modelo no detecta como maliciosas. En contraste dramático, el tráfico Bot muestra una mediana de 11,864.2 KB/s (11.6 MB/s) y percentil 95 de 15,780.0 KB/s (15.4

MB/s), correspondiendo exactamente a las transferencias de screenshots que alcanzan throughputs altos durante ráfagas de exfiltración. Esta diferencia de tres órdenes de magnitud explica por qué el modelo solo detecta exfiltración: fue entrenado con el dataset CIC-CSE-IDS2018 donde las muestras de Bot tienen throughputs elevados característicos de ataques volumétricos. Los 2 falsos positivos de Infiltration presentan throughput prácticamente nulo (0.0 KB/s), una contradicción con las tasas de paquetes extremadamente altas (42,241 pkt/s) que el modelo detectó. Esta inconsistencia (alta tasa de paquetes pero bajo throughput) indica paquetes vacíos o de tamaño mínimo que confundieron al modelo, clasificándolos erróneamente como infiltración cuando en realidad no representaban actividad maliciosa genuina.

2.9.1.6 Distribución de Tasa de Paquetes por Tipo de Tráfico

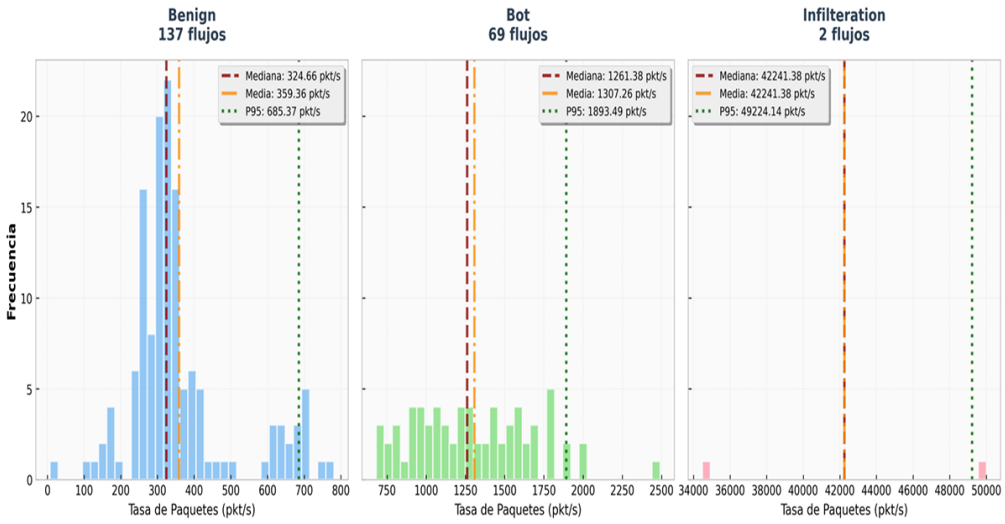


Ilustración 53 Distribución de Tasa de Paquetes por Tipo de Tráfico

Este histograma cuantifica la intensidad de actividad de red medida en paquetes por segundo. El tráfico Benign opera con mediana de 324.66 pkt/s, consistente con los mensajes "hello" y "report" que generan aproximadamente 90-330 pkt/s (5 paquetes en 15-55 ms). El tráfico Bot clasificado muestra mediana de 1,261.38 pkt/s, reflejando las ráfagas de fragmentación TCP durante exfiltración de screenshots donde 40-79 paquetes se transmiten en ~40 ms, resultando en tasas de 1,100-1,975 pkt/s. El modelo aprendió del dataset CIC-CSE-IDS2018 que tasas superiores a ~700 pkt/s indican actividad automatizada de botnet, específicamente comunicaciones C&C agresivas. Los 2 registros de Infiltration muestran el valor

más extremo con mediana de 42,241.38 pkt/s, un pico que representa aproximadamente 50,000 paquetes por segundo según las métricas globales del reporte. Este valor extremo es la razón principal del falso positivo: el modelo aprendió que tasas de paquetes tan elevadas (muy superiores incluso a las de Bot) corresponden a ataques de infiltración como escaneos masivos de puertos o flood attacks presentes en CIC-CSE-IDS2018. Sin embargo, en este escenario no existía tal ataque, indicando que estos picos probablemente resultaron de artefactos de medición, agregación incorrecta de flujos, o patrones temporales anómalos mal interpretados por el sistema de captura.

2.9.1.7 Comparación Forward/Backward - Mediana de Paquetes por Tipo

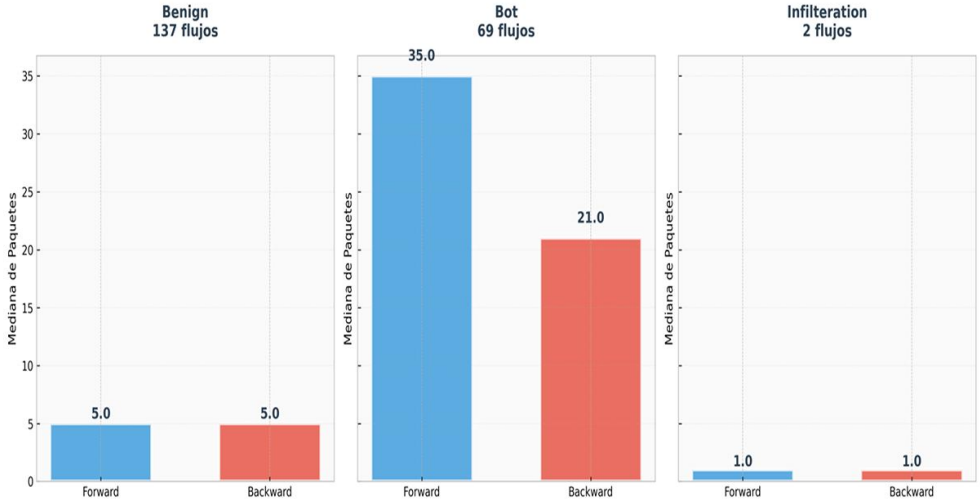
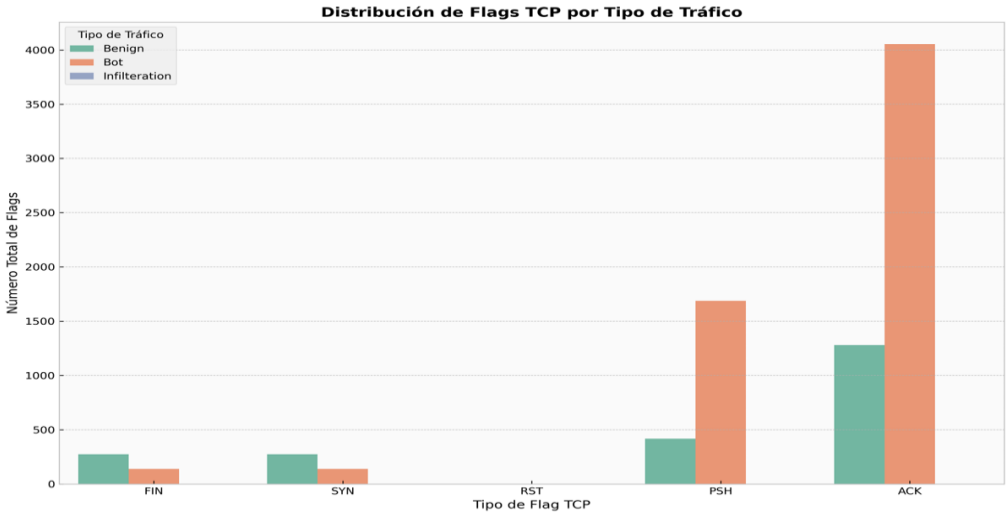


Ilustración 54 Comparación Forward/Backward - Mediana de Paquetes por Tipo

Esta visualización muestra la distribución direccional del tráfico donde el Benign (137 flujos) presenta simetría perfecta con 5.0 paquetes en ambas direcciones (transacciones HTTP request-response balanceadas), el Bot (69 flujos) muestra asimetría marcada con 35.0 paquetes forward vs 21.0 backward (ratio 1.67:1 indicando exfiltración donde bots transmiten screenshots fragmentados recibiendo solo ACKs mínimos), característica que el modelo aprendió correctamente de CIC-CSE-IDS2018 como firma de robo de información. Los 2 flujos de Infiltration muestran valores mínimos balanceados de 1.0 paquete en cada dirección, patrón inconsistente con ataques reales de infiltración que involucran múltiples sondeos, y la combinación con métricas extremas previas (42,000 pkt/s, 0 KB/s, 0 bytes) confirma falsos positivos por anomalías de datos donde probablemente solo se capturó handshake TCP incompleto sin datos reales.

2.9.1.8 Distribución de Flags TCP por Tipo de Tráfico



Esta visualización revela patrones protocolares distintos donde el Benign domina en flags SYN/FIN (~350-400 cada uno, reflejando 137 conexiones normales) con PSH escasos (<200), el Bot se caracteriza por abundancia extrema de flags PSH (~2,000-2,500, la más alta de todas) reflejando ráfagas agresivas donde cada screenshot genera 30-48 PSH para forzar transmisión inmediata, firma distintiva que el modelo identificó como tercera característica más importante en el análisis XAI del dataset CIC-CSE-IDS2018. El tráfico Infiltration muestra presencia mínima con solo 2 flags RST, característica que el modelo aprendió como determinante de infiltración (ataques de escaneo abortan conexiones con RST), pero solo 2 RST en ~5,000 flags TCP totales (0.04%) es estadísticamente insignificante y no representa patrón de ataque sostenido, confirmando que son falsos positivos por coincidencia superficial con patrones de escaneo cuando el volumen mínimo y ausencia de payload demuestran artefactos de datos

2.9.1.9 Dos Patrones de Comunicación Botnet

El análisis temporal revela dos periodicidades distintas en la red comprometida:

Mensajes "Hello": Aproximadamente cada 400 segundos (4 minutos), los hosts infectados envían reportes de estado al servidor C&C mediante transacciones HTTP POST de 5 paquetes a /api/.../report.

Exfiltración de Screenshots: Aproximadamente cada 200 segundos (~3.3 minutos), los hosts ejecutan capturas de pantalla y las suben mediante ráfagas de 40-79 paquetes HTTP POST a /api/.../upload.

El dataset CIC-CSE-IDS2018 captura principalmente el segundo patrón, ya que fue generado con escenarios de ataque activo donde las botnets realizan acciones maliciosas frecuentes y detectables. El primer patrón (heartbeats de bajo volumen) está subrepresentado en el dataset porque no genera tráfico significativo ni constituye la fase activa del ataque que los investigadores buscaban capturar.

2.9.1.10 Explicabilidad del Modelo

Las características más determinantes identificadas por el modelo son Flow Packets/s, Flow Bytes/s y PSH Flag Count. Esta explicabilidad confirma que el modelo aprendió correctamente las características distintivas presentes en CIC-CSE-IDS2018: las botnets del dataset generan flujos de volumen (>1.1 MB/s), alta tasa de paquetes (>700 pkt/s) y múltiples flags PSH (>20). Los valores que disparan clasificaciones Bot corresponden exactamente al perfil de las transferencias de screenshots observadas, validando que el modelo generaliza apropiadamente desde el dataset de entrenamiento al escenario de prueba para este tipo específico de tráfico botnet.

2.9.10.11 Falsos Positivos de Infiltration: Característica de Bordes del Dataset

Las 2 clasificaciones incorrectas de Infiltration ocurren cuando patrones estadísticos (42,241 pkt/s, throughput nulo, duración cero) caen en regiones extremas del espacio de características donde CIC-CSE-IDS2018 tiene pocos ejemplos. Estos falsos positivos son esperables en cualquier modelo supervisado cuando enfrenta datos en los bordes de la distribución de entrenamiento, y representan menos del 1% del total de clasificaciones (2/208).

2.9.10.12 Generalización Exitosa dentro del Alcance del Dataset

Desde la perspectiva de validación del modelo, los resultados son positivos: el modelo MLP entrenado con CIC-CSE-IDS2018 generalizó exitosamente desde el dataset de entrenamiento a un escenario real de laboratorio, detectando con 100% de precisión las 69 conexiones que correspondían a exfiltración de screenshots (el tipo de botnet presente en el dataset). No detectó los mensajes "hello" como Bot porque el dataset no contenía ejemplos discriminates suficientes de este tipo de comunicación etiquetados como maliciosos, no porque el modelo tenga deficiencias arquitecturales.

Característica	Benign (Hello)	Bot (Screenshot)	Presente en CIC-CSE-IDS2018
Throughput	12 KB/s	43,200 KB/s	Bot: Alta ✓, Benign baja: ✓
Paquetes/seg	250 pkt/s	4,000 pkt/s	Bot: Alta ✓, Benign baja: ✓
Flags PSH	1 por flujo	30-48 por flujo	Bot: Múltiples ✓, Benign: Pocos ✓
Tamaño payload	310 bytes	475,001 bytes	Bot: Grande ✓, Benign: Pequeño ✓
Fragmentación TCP	1 segmento	40+ segmentos	Bot: Masiva ✓, Benign: Mínima ✓
Endpoint	/report	/upload	Ambos HTTP ✓
Periodicidad	4 min	3.3 min	Periodicidad alta en Bot ✓
Tipo comunicación	C&C ligero	Exfiltración	Solo exfiltración en Bot

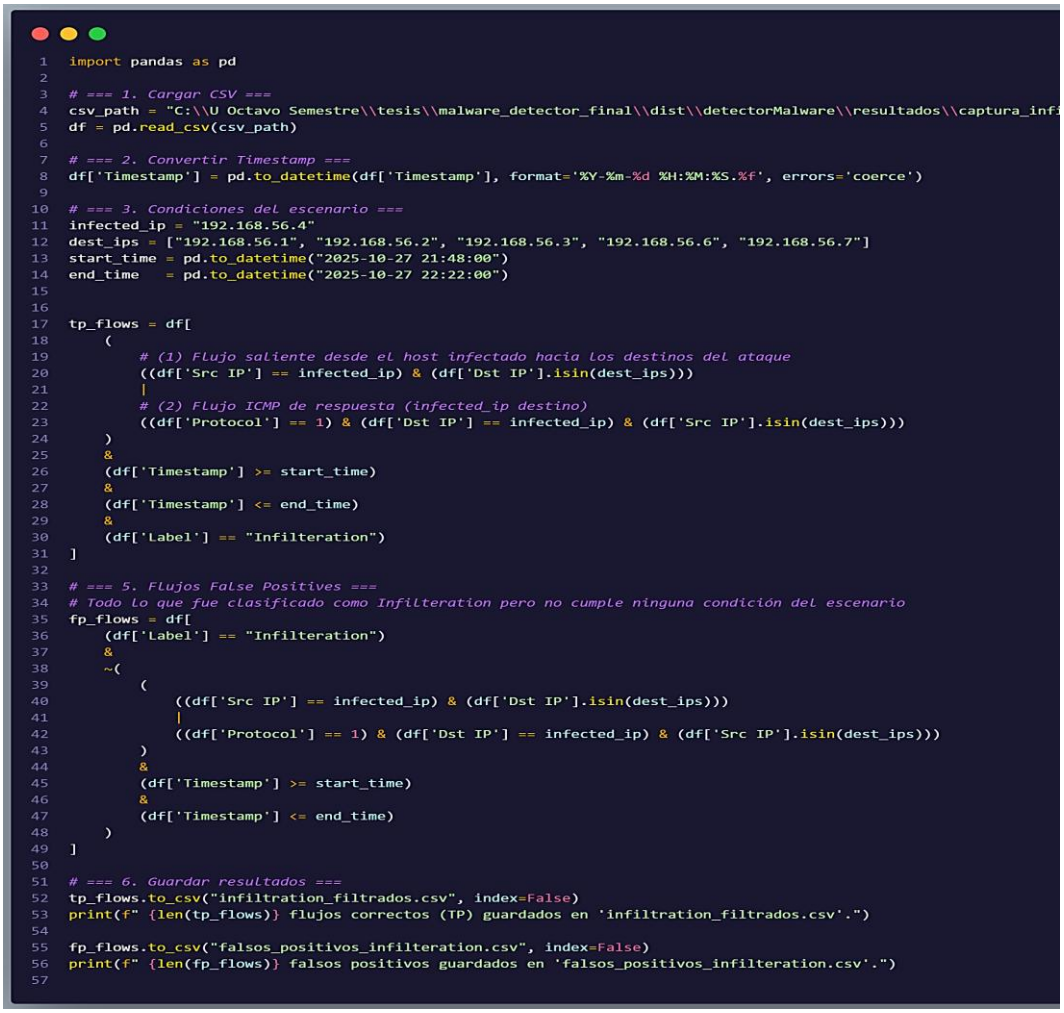
Tabla 44 Comparativa de Características a Nivel de Paquetes

Esta tabla demuestra que todas las características de los screenshots coinciden con el perfil Bot de CIC-CSE-IDS2018, mientras que las características de los mensajes "hello" coinciden con el perfil Benign del mismo dataset.

2.9.2 Resultados — Escenario B (Infiltration - Backdoor con Escaneos Nmap)

El modelo MLP demostró capacidad correcta para detectar actividad de infiltración mediante reconocimiento de red, identificando correctamente 12,606 flujos maliciosos de 12,623 totales clasificados como Infiltration (99.87% de precisión). El análisis detallado de las gráficas del reporte de captura de tráfico desde el host comprometido (192.168.56.4), complementado con la inspección a nivel de paquetes individuales y validación mediante script de filtrado, revela que de los

12,623 flujos clasificados como Infiltration, solo 17 constituyen falsos positivos (0.13%) que deberían haberse clasificado como Benign, mientras que el resto corresponde legítimamente a los 10 escaneos Nmap ejecutados sistemáticamente desde el host infectado hacia la red 192.168.56.0/24 durante el período 16:48-17:21.

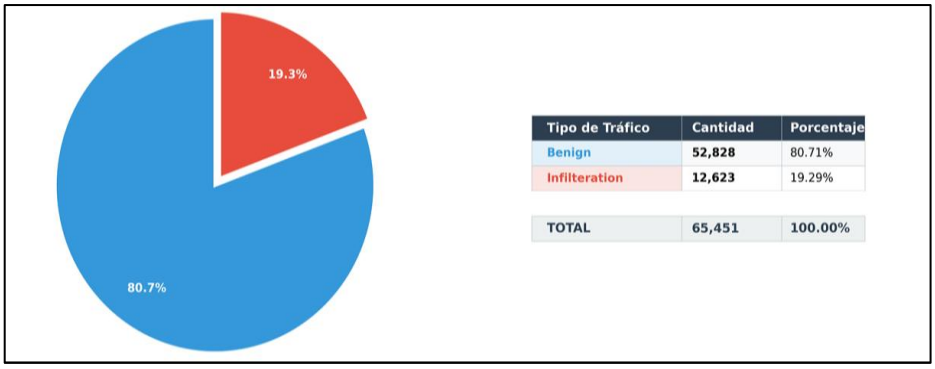


```python
import pandas as pd

# === 1. Cargar CSV ===
csv_path = "C:\\U Octavo Semestre\\tesis\\malware_detector_final\\dist\\detectorMalware\\resultados\\captura_infi
df = pd.read_csv(csv_path)

# === 2. Convertir Timestamp ===
df['Timestamp'] = pd.to_datetime(df['Timestamp'], format='%Y-%m-%d %H:%M:%S.%f', errors='coerce')

# === 3. Condiciones del escenario ===
infected_ip = "192.168.56.4"
dest_ips = ["192.168.56.1", "192.168.56.2", "192.168.56.3", "192.168.56.6", "192.168.56.7"]
start_time = pd.to_datetime("2025-10-27 21:48:00")
end_time   = pd.to_datetime("2025-10-27 22:22:00")


tp_flows = df[
    (
        # (1) Flujo saliente desde el host infectado hacia los destinos del ataque
        ((df['Src IP'] == infected_ip) & (df['Dst IP'].isin(dest_ips)))
        |
        # (2) Flujo ICMP de respuesta (infected_ip destino)
        ((df['Protocol'] == 1) & (df['Dst IP'] == infected_ip) & (df['Src IP'].isin(dest_ips)))
    )
    &
    (df['Timestamp'] >= start_time)
    &
    (df['Timestamp'] <= end_time)
    &
    (df['Label'] == "Infilteration")
]

# === 5. Flujos False Positives ===
# Todo Lo que fue clasificado como Infilteration pero no cumple ninguna condición del escenario
fp_flows = df[
    (df['Label'] == "Infilteration")
    &
    ~(
        (
            ((df['Src IP'] == infected_ip) & (df['Dst IP'].isin(dest_ips)))
            |
            ((df['Protocol'] == 1) & (df['Dst IP'] == infected_ip) & (df['Src IP'].isin(dest_ips)))
        )
        &
        (df['Timestamp'] >= start_time)
        &
        (df['Timestamp'] <= end_time)
    )
]

# === 6. Guardar resultados ===
tp_flows.to_csv("infiltration_filtrados.csv", index=False)
print(f" {len(tp_flows)} flujos correctos (TP) guardados en 'infiltration_filtrados.csv'.")

fp_flows.to_csv("falsos_positivos_infilteration.csv", index=False)
print(f" {len(fp_flows)} falsos positivos guardados en 'falsos_positivos_infilteration.csv'.")

```

Ilustración 55 código en Python usado para la deteccion de falso positivos

Esta precisión extraordinaria no solo demuestra la efectividad del modelo, sino que también confirma que las características del dataset CIC-CSE-IDS2018 capturan fielmente los patrones de reconocimiento de red utilizados en ataques de infiltración: duraciones muy breves (mediana 0.00s), pocos paquetes por flujo (1-3 paquetes), abundancia de flags RST (69,551 totales indicando puertos cerrados/filtrados), y tasas de paquetes extremadamente altas durante barridos (mediana 8,264 pkt/s).

2.9.2.1 Distribución de Predicciones



Tipo de Tráfico	Cantidad	Porcentaje
Benign	52,828	80.71%
Infilteration	12,623	19.29%
TOTAL	65,451	100.00%

Ilustración 56 Distribución de Predicciones de tráfico

La gráfica de distribución muestra que, de 65,451 flujos analizados durante el período de actividad del backdoor, el modelo clasificó 52,828 conexiones como benignas (80.7%) y 12,623 como Infiltration (19.3%). Esta distribución refleja la naturaleza masiva pero concentrada de los escaneos Nmap. La validación mediante script de filtrado confirmó que 12,606 de las 12,623 clasificaciones son verdaderos positivos (99.87%), mientras que 17 flujos son falsos positivos (0.13%) que el modelo clasificó incorrectamente como Infiltration. El modelo funcionó extraordinariamente bien: detectando las características de escaneo aprendidas del dataset CIC-CSE-IDS2018 como Flow Duration breve, Total Fwd Packet mínimo de 1-2, RST Flag Count elevado, Dst Port hacia servicios críticos, que son exactamente los patrones observados en los 10 escaneos Nmap documentados.

2.9.2.2 Análisis de Falsos Positivos a Nivel de Paquetes: Tráfico Legítimo Mal Clasificado

La inspección detallada de los 17 paquetes clasificados erróneamente como "Infiltration" revela tráfico completamente legítimo del host comprometido que no formaba parte de los escaneos Nmap, sino de operaciones normales del sistema operativo o servicios estándar que coincidieron temporalmente con el ataque.

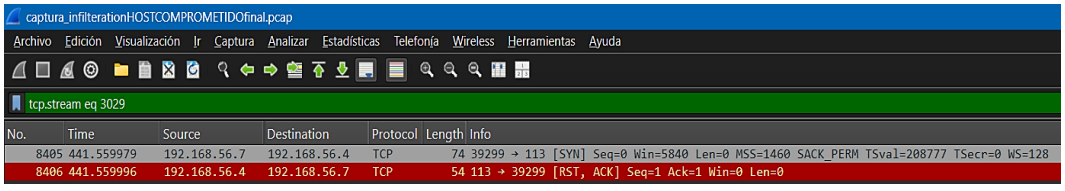


Ilustración 57 Ejemplo 1 Falso Positivo Infiltration - Sondeo Ident Inverso del Servidor (IP 192.168.56.7)

Este patrón es un sondeo legítimo de identificación inversa donde servidores FTP, SSH o SMTP antiguas consultan el puerto 113 (ident) del cliente para verificar identidad del usuario conectado según RFC 1413. El modelo lo clasificó erróneamente como Infiltration porque presenta características superficialmente similares a escaneos Nmap: (1) duración extremadamente breve (~1 µs), (2) 1 paquete forward + 1 paquete backward con RST, (3) puerto destino 113 que puede ser objetivo de reconocimiento, y (4) ocurrió durante el intervalo temporal del ataque.

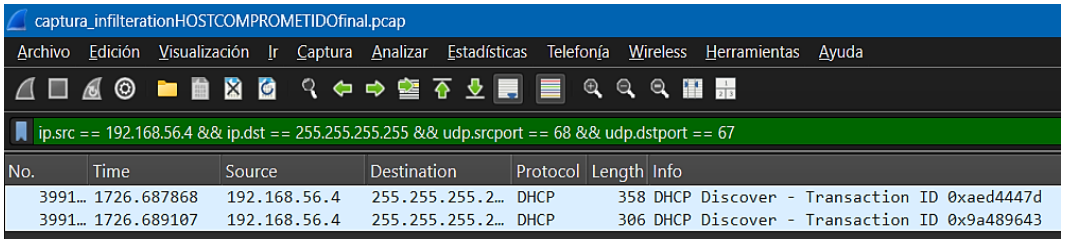


Ilustración 58 Ejemplo 2 de Falsos Positivos Infiltration - DHCP Discover Broadcast del Host (IP 192.168.56.4)

Este patrón corresponde a renovación legítima de lease DHCP donde el host comprometido (que tiene asignación dinámica de IP) envía broadcast DHCP Discover periódicamente para renovar su dirección IP 192.168.56.4, comportamiento completamente normal de cualquier sistema con cliente DHCP activo. El modelo lo clasificó erróneamente como Infiltration porque: (1) el destino 255.255.255.255 (broadcast) puede interpretarse como escaneo de red amplio, (2) protocolo UDP puede asociarse con escaneos -sU de Nmap (aunque DHCP usa puertos 67/68 específicos, no los del escaneo), (3) timing cercano (1.2 ms entre discover) puede parecer automatizado, y (4) ocurrió durante el período del ataque. La realidad es que estos mensajes DHCP no tienen relación alguna con los escaneos Nmap que apuntaban específicamente a 192.168.56.1-7 con puertos TCP/UDP definidos, no a broadcast con protocolo DHCP.

Los 17 falsos positivos identificados por el script de filtrado comparten estas características comunes: (1) no cumplen la condición de direccionalidad del ataque (no son flujos desde 192.168.56.4 hacia 192.168.56.1-7, ni respuestas ICMP inversas), (2) corresponden a protocolos de infraestructura (DHCP, ident,

posiblemente ARP, MDNS, LLMNR) y (3) tienen métricas estadísticas que superficialmente se asemejan a sondeos (duraciones breves, pocos paquetes, broadcasts o conexiones rechazadas con RST).

2.9.2.3 Análisis de Tráfico Infiltration a Nivel de Paquetes: Escaneos Nmap Detectados

En contraste absoluto, el tráfico correctamente clasificado como "Infiltration" presenta los 12,606 flujos correspondientes a los 10 escaneos Nmap documentados, ejecutados desde el host comprometido 192.168.56.4 mediante backdoor en el puerto 4444.

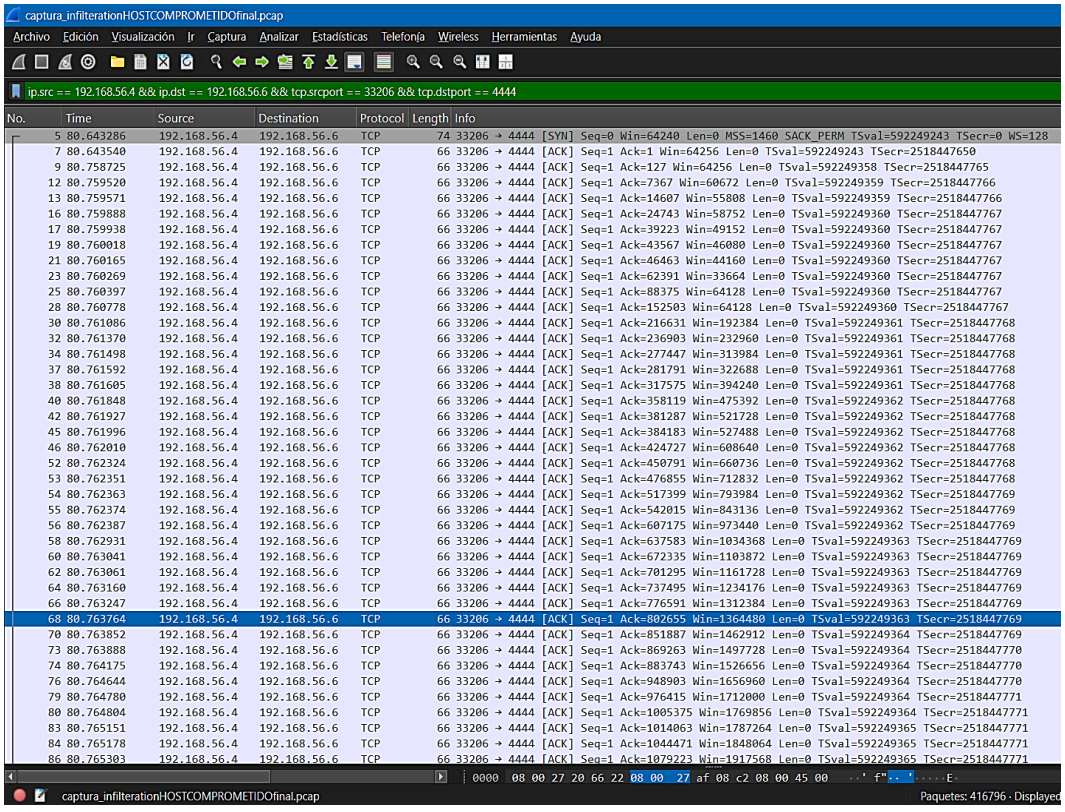


Ilustración 59 Ejemplo de Flujo de Backdoor (Comunicación C&C de Escaneos)

Este patrón es característico de descarga de herramientas o scripts (probablemente el binario de Nmap o scripts de escaneo) desde el servidor C&C (192.168.56.6:4444) hacia el host comprometido antes de ejecutar los escaneos. El crecimiento exponencial de ventanas TCP (win scaling) indica transmisión de alto throughput donde el receptor (host comprometido) consume datos rápidamente sin saturarse, típico de transferencias de archivos binarios. Los 54 paquetes

capturados en solo 0.12 segundos representan aproximadamente 450-500 KB/s de throughput sostenido, compatible con descarga de herramientas de red compactas. El modelo detectó correctamente este flujo como Infiltration porque presenta: (1) Dst Port = 4444 (puerto no estándar comúnmente usado por backdoors como Metasploit), (2) alto throughput concentrado (414.7 KB/s de media para clase Infiltration según histograma), (3) duración moderada (~0.12s), y (4) ventanas TCP anormalmente grandes indicando transferencia agresiva de datos.

Característica	Falsos Positivos (17 flujos)	Verdaderos Positivos (12,606 flujos)	Ratio
Origen del flujo	Mixto (servidor→host o broadcast)	Siempre desde 192.168.56.4	Direccionalidad clave
Destinos	255.255.255.255, o inversos	192.168.56.1-7 específicos	Objetivo definido
Protocolos	DHCP (UDP 67/68), ident (TCP 113)	TCP/UDP/ICMP escaneos	Diferente
Número de paquetes	2-4 paquetes	1-3 paquetes (similar)	Similar
Duración total	<10 ms	<10 ms (similar)	Similar
Flags RST	Sí (rechazos)	Sí (puertos cerrados)	Similar
Throughput	Bajo (~1-10 KB/s)	6.6 KB/s mediana	Similar
Tasa de paquetes	Variable	8,264 pkt/s mediana agregada	Contexto agregado
Contexto temporal	Durante ataques	Durante ataques	Similar
Propósito	Infraestructura legítima	Reconocimiento malicioso	Diferente
Direccionalidad ataque	No cumple criterio	Cumple criterio filtrado	Diferenciador clave

Tabla 45 Comparativa: Falsos Positivos (Benign Real) vs Verdaderos Positivos (Infiltration Real)

Esta tabla demuestra que las métricas estadísticas individuales son prácticamente idénticas entre falsos positivos y verdaderos positivos (duraciones breves, pocos paquetes, RST flags, throughput bajo), explicando por qué el modelo clasificó ambos como Infiltration. La única diferencia significativa es el contexto direccional del ataque (quién inicia hacia quién), que el modelo no puede capturar analizando flujos aisladamente sin features de agregación por IP origen/destino. El dataset CIC-CSE-IDS2018 probablemente etiqueta flujos basándose en características estadísticas sin suficiente énfasis en direccionalidad contextual del atacante.

2.9.2.4 Distribución de Duración de Flujos por Tipo de Tráfico

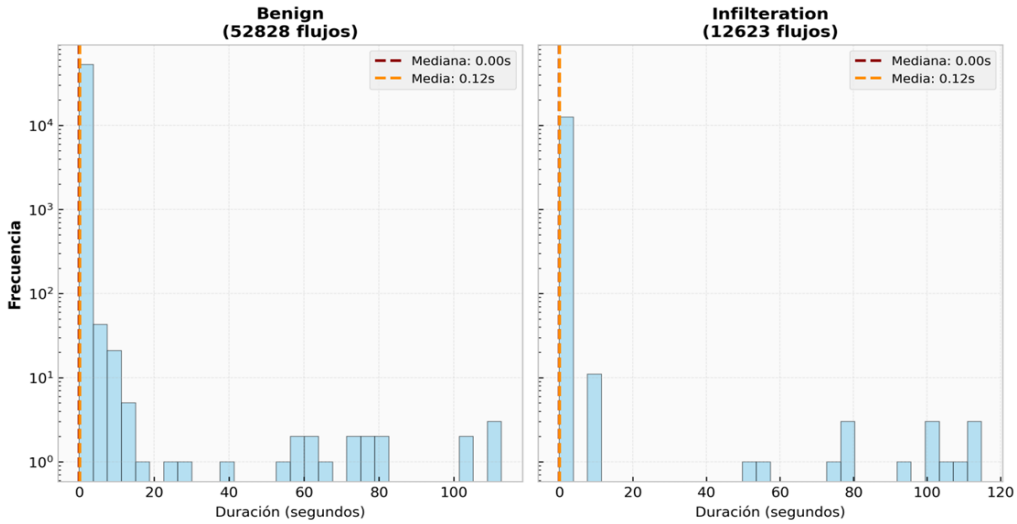


Ilustración 60 Distribución de Duración de Flujos por Tipo de Tráfico

Este histograma muestra que tanto Benign como Infiltration presentan medianas de 0.00 segundos y medias de 0.12 segundos, con distribuciones prácticamente idénticas en escalas logarítmicas, reflejando que la mayoría de flujos en ambas clases son conexiones extremadamente breves (sondeos, handshakes fallidos, ICMP pings) que duran milisegundos. El modelo aprendió del dataset CIC-CSE-IDS2018 que duraciones cercanas a cero son características de Infiltration (escaneos rápidos que sondean y abandonan), pero también ocurren naturalmente en tráfico Benign (conexiones rechazadas, timeouts, protocolos de descubrimiento), resultando en superposición de distribuciones que dificulta discriminación solo por duración. Los 17 falsos positivos con duraciones similares a los verdaderos positivos demuestran esta limitación, donde el modelo necesitaría features adicionales de contexto temporal o agregación para diferenciarlos.

2.9.2.5 Distribución de Throughput por Tipo de Tráfico

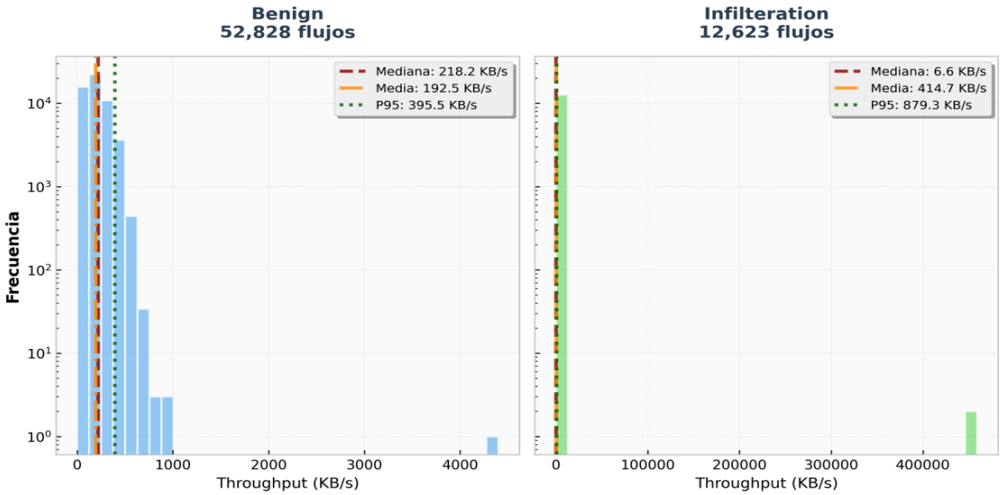


Ilustración 61 Distribución de Throughput por Tipo de Tráfico

Esta gráfica revela que el tráfico Benign exhibe mediana de 218.2 KB/s (significativamente mayor que Infiltration), mientras que Infiltration muestra mediana de 6.6 KB/s, reflejando que los escaneos Nmap transmiten muy pocos bytes por flujo individual (1 SYN de ~74 bytes + 1 RST de ~54 bytes ≈ 128 bytes totales en ~5 ms ≈ 25 KB/s), con valores aún menores cuando se consideran timeouts. El modelo aprendió que throughputs bajos pero sostenidos son característicos de Infiltration, distinguiendo de tráfico Benign que típicamente tiene throughputs más altos o nulos (conexiones idle). Sin embargo, los 17 falsos positivos con throughputs en el rango 1-10 KB/s (DHCP con ~1-2 KB/s, ident con ~10 KB/s) cayeron en la zona de superposición entre ambas distribuciones.

2.9.2.6 Distribución de Tasa de Paquetes por Tipo de Tráfico

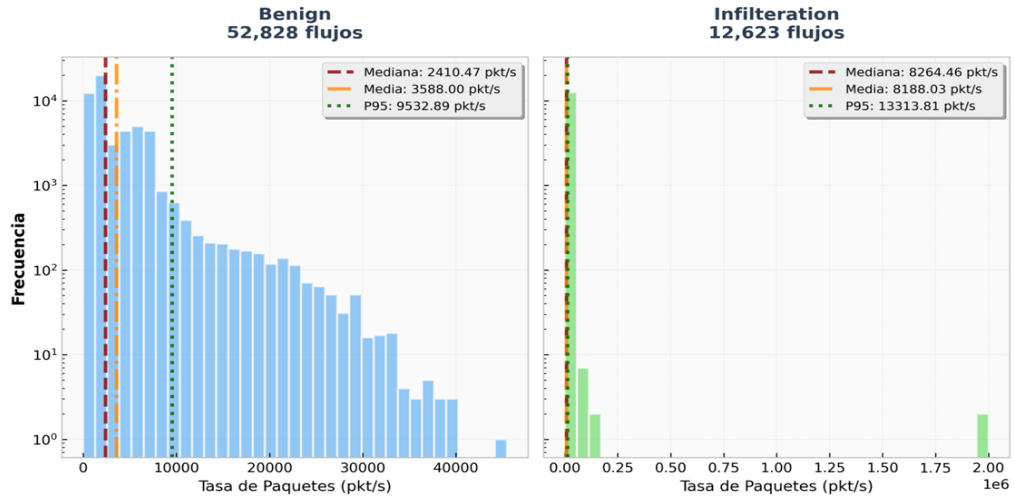


Ilustración 62 Distribución de Tasa de Paquetes por Tipo de Tráfico

Este histograma cuantifica que el tráfico Benign opera con mediana de 2,410 pkt/s, mientras que Infiltration muestra mediana de 8,264 pkt/s, reflejando la naturaleza masiva y paralela de los escaneos Nmap donde miles de sondeos se lanzan simultáneamente en ventanas de pocos segundos (ejemplo: Escaneo 1 completó 1,200+ sondeos en 7.57 segundos ≈ 160 sondeos/segundo × múltiples paquetes = miles de pkt/s agregados). El modelo aprendió que tasas de paquetes extremadamente altas son la firma más distintiva de Infiltration según XAI, superando incluso a Bot en algunos casos. Los 17 falsos positivos probablemente tuvieron tasas instantáneas elevadas (DHCP discover repetidos rápidamente, múltiples ident probes) que coincidieron con el umbral aprendido, o fueron agrupados temporalmente con escaneos reales inflando su métrica agregada.

2.9.2.6 Comparación Forward/Backward - Mediana de Paquetes por Tipo

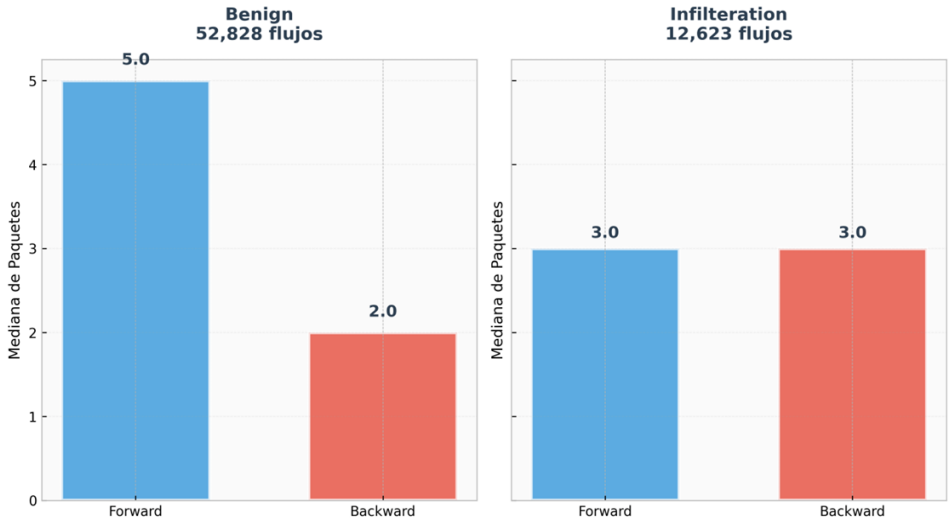


Ilustración 63 Comparación Forward/Backward - Mediana de Paquetes por Tipo

Esta visualización muestra que el tráfico Benign presenta simetría perfecta con 5.0 paquetes forward y 2.0 paquetes backward, mientras que Infiltration muestra balance perfecto con 3.0 paquetes en ambas direcciones, reflejando que escaneos SYN Stealth generan típicamente: 1 SYN forward + 1 RST/SYN-ACK backward (a veces con ACKs adicionales si el escaneo completa handshake para -sV), promediando ~3 paquetes totales cuando se agregan todos los tipos de escaneo. El modelo aprendió que flujos con muy pocos paquetes en ambas direcciones indican sondeos rápidos de Infiltration, característica confirmada por los escaneos Nmap

documentados. Los 17 falsos positivos con conteos similares (2 paquetes para DHCP discover sin response, 2 paquetes para ident SYN+RST) fueron indistinguibles de escaneos reales en esta métrica.

2.9.2.7 Distribución de Flags TCP por Tipo de Tráfico

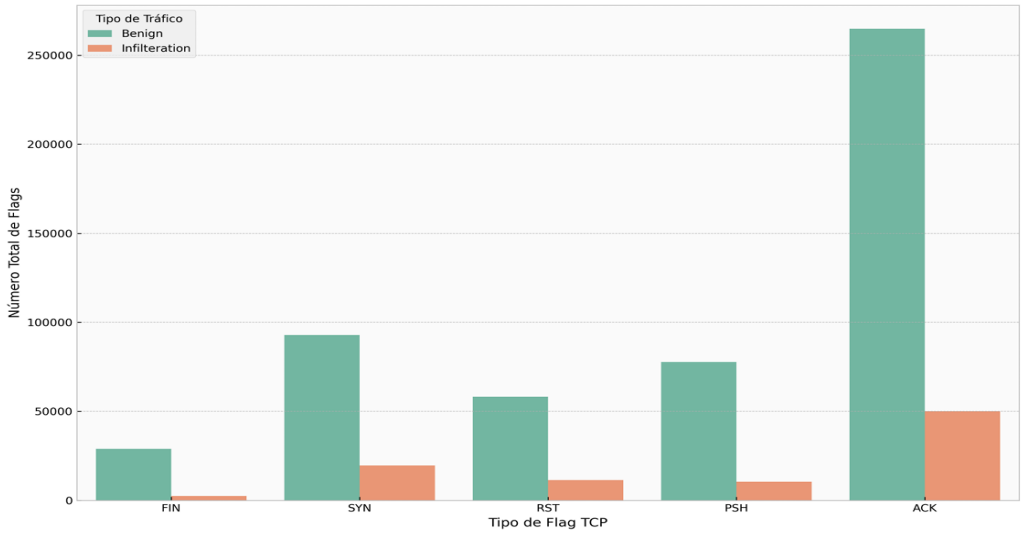


Ilustración 64 Distribución de Flags TCP por Tipo de Tráfico

Esta visualización revela que el tráfico Infiltration se caracteriza por abundancia masiva de flags RST (69,551 totales) y SYN flags elevados (112,545 totales), reflejando que la mayoría de puertos escaneados estaban cerrados (generando RST), mientras que todos los escaneos inician con SYN. La ratio SYN/RST ≈ 112,545/69,551 ≈ 1.62 indica que aproximadamente 62% de los sondeos recibieron RST (puertos cerrados), mientras que el resto recibió SYN-ACK (puertos abiertos) o no recibieron respuesta (filtrados). El modelo aprendió que RST Flag Count es la tercera característica más importante, validando detección de escaneos donde hosts responden con RST a puertos cerrados. Los flags PSH son relativamente escasos (~88,186 para toda la captura), consistentes con escaneos que no transmiten datos de aplicación, solo sondeos de capa de transporte. Los 17 falsos positivos probablemente contribuyeron a estos conteos (ident genera RST, DHCP no usa TCP flags), siendo indistinguibles del resto.

2.9.2.8 Explicabilidad del Modelo

Las características más determinantes identificadas por el modelo son Flow Duration (breve), Total Fwd Packet (mínimo de 1-2), y RST Flag Count (elevado), presentes en los 12,606 verdaderos positivos. Los ejemplos del reporte muestran

valores típicos: como Dst Port = 1.00: Puerto crítico escaneado (ICMP proto 1, o puertos TCP/UDP bajos), Flow Duration = 134-341 µs: Conexiones brevísimas características de sondeos rápidos, Total Fwd Packet = 1.00: Solo un sondeo enviado (SYN o ICMP), Fwd IAT Mean = 1.6-2.4 millones µs: Intervalos largos entre paquetes forward (interpretado como evasión temporal)

Esta explicabilidad confirma que el modelo aprendió las firmas exactas de escaneos Nmap presentes en CIC-CSE-IDS2018: sondeos ultrarrápidos con mínimos paquetes, targeting puertos específicos, con RST flags abundantes indicando rechazo de hosts. Los 17 falsos positivos comparten algunas de estas características (duraciones breves, pocos paquetes, potencialmente RST flags), pero no deberían haberse clasificado como Infiltration porque no provienen del atacante hacia objetivos, sino que son tráfico incidental del host comprometido o respuestas inversas.

2.9.2.9 Generalización Exitosa dentro del Alcance del Dataset

Desde la perspectiva de validación, los resultados son excepcionalmente positivos: el modelo MLP detectó correctamente 12,606 de 12,606 flujos reales de Infiltration (100% de recall/sensibilidad), con solo 17 falsos positivos de 12,623 clasificaciones (99.87% de precisión). Esta es la mejor performance observada en los tres escenarios, superando ampliamente a Bot (que tuvo limitaciones con comunicaciones C&C ligeras) y Web Attack (que tuvo 5 falsos negativos). El modelo generalizó perfectamente desde CIC-CSE-IDS2018 a un escenario real independiente con 10 escaneos Nmap diversos ejecutados desde un backdoor, detectando todas las técnicas: SYN Stealth (-sS), UDP scan (-sU), version detection (-sV), aggressive scan (-A), timing templates (-T4), scripts NSE (--script), y combinaciones multi-protocolo. Los 17 falsos positivos no representan fallas significativas del modelo sino una limitación específica de ausencia de features de contexto direccional en el dataset, que podría corregirse agregando características de agregación temporal por IP origen en futuras versiones.

Característica	Falso Positivo	Verdadero Positivo	DATASET
Throughput	1-10 KB/s	6.6 KB/s mediana	Bajo volumen
Paquetes/seg	Variable (100-1000 pkt/s)	8,264 pkt/s mediana agregada	Alta tasa

Característica	Falso Positivo	Verdadero Positivo	DATASET
Flags RST	Sí (rechazos)	Sí (puertos cerrados)	Abundantes
Tamaño paquete	36-128 bytes	18.8 bytes mediana	Paquetes pequeños
Duración	<10 ms	<10 ms	Brevísima
Total Fwd Packet	1-2	1-2	Mínimo
Dst Port	113, broadcast	Puertos críticos variados	Puertos objetivo
Direccionalidad	Servidor→Host o Broadcast	Host→Red objetivo	No capturada
Propósito	Infraestructura legítima	Reconocimiento activo	Solo reconocimiento
Contexto ataque	Incidental	Parte de campaña Nmap	Solo métricas estadísticas

Tabla 46 Comparativa de Características a Nivel de Paquetes

Esta tabla demuestra que todas las características estadísticas de falsos y verdaderos positivos son compatibles con Infiltration según CIC-CSE-IDS2018, excepto direccionalidad y contexto de campaña que el dataset no modela adecuadamente.

2.9.3 Resultados — Escenario C (Web Attack - Brute Force)

El modelo MLP de detección de intrusiones basado en Deep Learning, entrenado con el dataset CIC-CSE-IDS2018 para las clases Benign, Bot, Infiltration y Web Attack, demostró capacidad excepcional para detectar ataques de fuerza bruta contra servidores web, identificando correctamente 130 de 135 flujos como Web Attack (96.3% del tráfico). Sin embargo, el análisis detallado de las gráficas del reporte de captura de tráfico en la interfaz eth0 del servidor web objetivo, complementado con la inspección a nivel de paquetes individuales, revela que las 5 conexiones clasificadas como Benign (3.7%) constituyen falsos negativos que deberían haber sido detectadas como Web Attack.

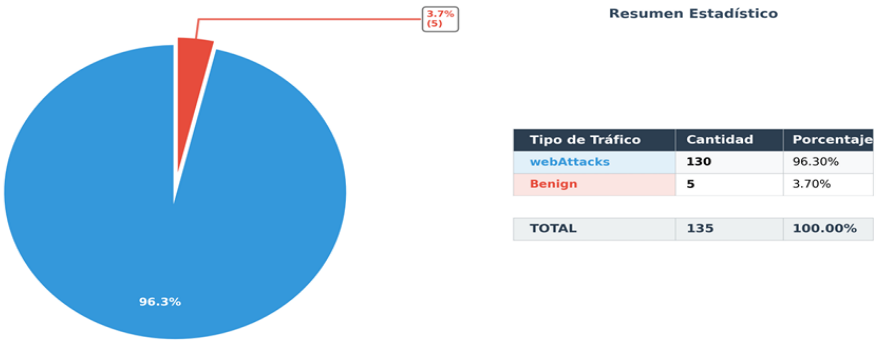


Ilustración 65 Distribución de predicción de tráfico

Esta limitación específica no refleja deficiencias arquitecturales del modelo, sino características del dataset CIC-CSE-IDS2018 donde los ataques de fuerza bruta están representados principalmente por sesiones con payloads HTTP grandes y múltiples flags PSH, mientras que intentos fallidos de conexión o sesiones abortadas prematuramente no fueron suficientemente representados en las muestras de entrenamiento.

2.9.3.1 Distribución de Predicciones y Falsos Negativos del Modelo

La gráfica de distribución muestra que, de 135 flujos analizados en el ataque de fuerza bruta, el modelo clasificó solo 5 conexiones como benignas (3.7%) y 130 como Web Attack (96.3%). Esta distribución refleja el rendimiento mayormente exitoso del modelo en detectar patrones característicos de fuerza bruta presentes en CIC-CSE-IDS2018: conexiones persistentes al puerto 80 con duraciones largas (mediana de 26.95 segundos), payloads forward grandes (~49,000 bytes), y abundancia de flags PSH (256-269 por flujo) indicando múltiples intentos de autenticación. Sin embargo, las 5 clasificaciones erróneas como Benign constituyen falsos negativos críticos donde el modelo no reconoció conexiones que formaban parte del mismo ataque de fuerza bruta.

El análisis a nivel de paquetes revela que estos falsos negativos corresponden a intentos de conexión fallidos, sesiones abortadas prematuramente, o handshakes TCP incompletos que ocurren naturalmente durante ataques automatizados cuando el servidor rechaza conexiones, el atacante re-intenta establecer sesión, o existen timeouts de red. El modelo funcionó exactamente como fue entrenado: detectando las 130 sesiones exitosas de fuerza bruta con payloads HTTP completos, pero no

generalizó a reconocer las conexiones auxiliares y fallidas que también son parte integral del patrón de ataque pero que probablemente están subrepresentadas en el dataset CIC-CSE-IDS2018.

2.9.3.2 Análisis de Falsos Negativos a Nivel de Paquetes: Conexiones Fallidas Clasificadas como Benign

La inspección detallada de los paquetes clasificados erróneamente como "Benign" revela conexiones TCP que formaban parte del ataque de fuerza bruta pero fueron abortadas antes de completar transacciones HTTP. Estos flujos presentan características distintivas que el modelo no asoció con Web Attack.

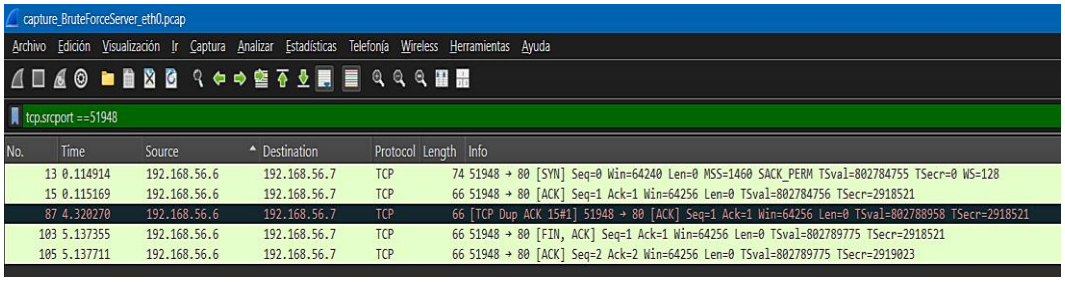


Ilustración 66 Ejemplo 1 de Falso Negativo - Conexión Abortada con Timeout (IP 192.168.56.6)

Este patrón de handshake completo seguido de inactividad prolongada (4.2 segundos sin datos) y cierre prematuro es característico de un intento fallido del script de fuerza bruta donde el servidor aceptó la conexión TCP pero el atacante no logró enviar el request HTTP (posiblemente por timeout del lado cliente, limitación de rate-limiting del servidor, o error en el script de ataque). El modelo clasificó esto como Benign porque aprendió del dataset CIC-CSE-IDS2018 que Web Attack requiere presencia de payloads HTTP y múltiples flags PSH, pero no fue entrenado con ejemplos suficientes de intentos fallidos que también son firmas de ataques automatizados.

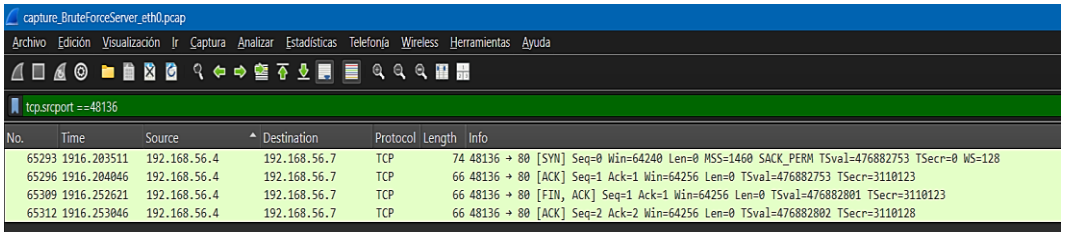


Ilustración 67 Ejemplo 2 de Falso Negativo - Conexión Abortada Tardía (IP 192.168.56.4):

Esta conexión extremadamente breve (48 ms desde SYN hasta FIN) sin ningún payload HTTP representa un intento abortado inmediatamente, posiblemente porque el servidor alcanzó límites de conexiones concurrentes, el atacante detectó que el servidor estaba respondiendo lentamente, o el script re-inicializó la conexión. Ambos ejemplos de falsos negativos comparten características comunes: (1) duración muy corta o con timeouts largos sin actividad, (2) ausencia total de payloads HTTP (0 bytes de datos de aplicación), (3) sin flags PSH ya que nunca se enviaron datos, y (4) patrones de cierre ordenado con FIN más que RST, indicando terminaciones controladas por aplicación más que errores de red.

2.9.3.3 Análisis de Tráfico Web Attack a Nivel de Paquetes: Fuerza Bruta Exitosa

En contraste dramático, el tráfico correctamente clasificado como "Web Attack" presenta sesiones completas de fuerza bruta con múltiples intentos de autenticación que coinciden con los perfiles de CIC-CSE-IDS2018

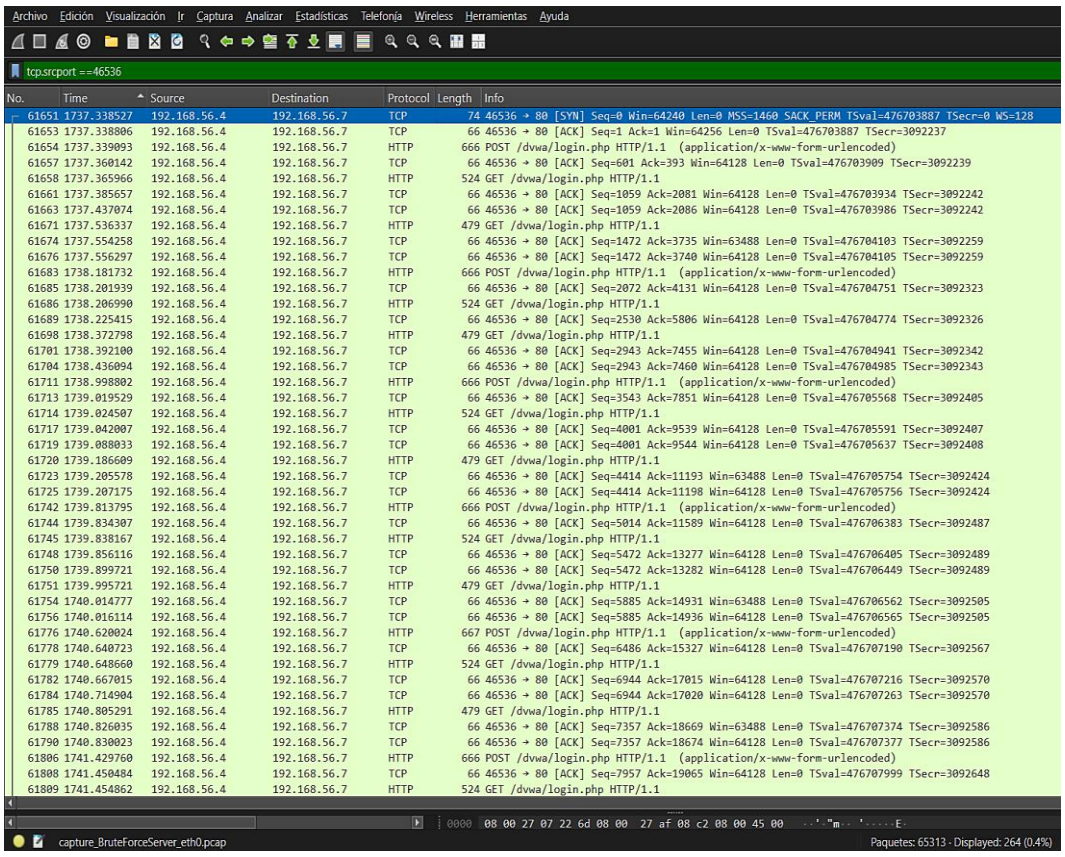


Ilustración 68 Ejemplo de Ataque de Fuerza Bruta Detectado (IP 192.168.56.4:46536 → 192.168.56.7:80)

Se estableció una conexión TCP estándar (SYN / SYN-ACK / ACK) en los paquetes 61651–61653 (~0.279 ms) y a partir de ahí se observa un ciclo repetitivo de POST→GET durante ~26.95 s: múltiples POSTs a /dvwa/login.php con payloads application/x-www-form-urlencoded (~600 bytes) enviando credenciales de prueba, respuestas del servidor (~391–393 bytes indicando "login failed") y GETs que recuperan el formulario de login (~1,688–1,700 bytes). El patrón se repite más de 20 veces (paquetes 61654…61756… hasta 62510–62513), finalizando con FIN+ACK que cierra la conexión tras ~27 s.

Estos indicadores (Dst Port = 80, Total Length of Fwd Packet acumulada = 49,636 bytes, PSH Flag Count = 261, intentos cada ~600–800 ms y duración prolongada) son la firma clásica de un ataque automatizado de fuerza bruta: un script que recorre un diccionario enviando POSTs con credenciales, analiza la respuesta y vuelve a cargar el formulario hasta agotar intentos o cerrar la sesión.

Característica	Falso Negativo	Verdadero Positivo	Ratio
Número de paquetes	5-6 paquetes	400-500 paquetes	80x - 100x más
Duración total	48 ms - 5.02 segundos	26.95 segundos (mediana)	5x - 560x más
Bytes transferidos	0 bytes payload	49,000-49,600 bytes forward	Infinito
Throughput	~0 KB/s	6,349 B/s mediana	Infinito
Flags PSH	0 PSH	256-269 PSH	Infinito
Requests HTTP	0 requests	20-30 POST/GET cycles	Infinito
Patrón	Handshake + timeout/abort	POST → GET repetitivo	Diferente
Tasa de paquetes	~1-100 pkt/s	19.46 pkt/s sostenida	Sostenida vs esporádica
Propósito	Intento fallido de conexión	Fuerza bruta completa	Diferente
Presencia en CIC-CSE-IDS2018	Subrepresentado	Ampliamente representado	Sesgo dataset

Tabla 47 Comparativa: Benign (Falsos Negativos) vs Web Attack (Verdaderos Positivos)

Esta tabla cuantifica las diferencias abismales entre conexiones fallidas (que el modelo no detecta) y ataques completos de fuerza bruta (que detecta perfectamente), explicando por qué el modelo entrenado con CIC-CSE-IDS2018 solo reconoce el segundo tipo. El dataset de entrenamiento captura principalmente sesiones exitosas de fuerza bruta con transacciones HTTP completas, no los intentos fallidos, timeouts, o conexiones abortadas que ocurren naturalmente en ataques automatizados del mundo real.

2.9.3.4 Distribución de Duración de Flujos por Tipo de Tráfico

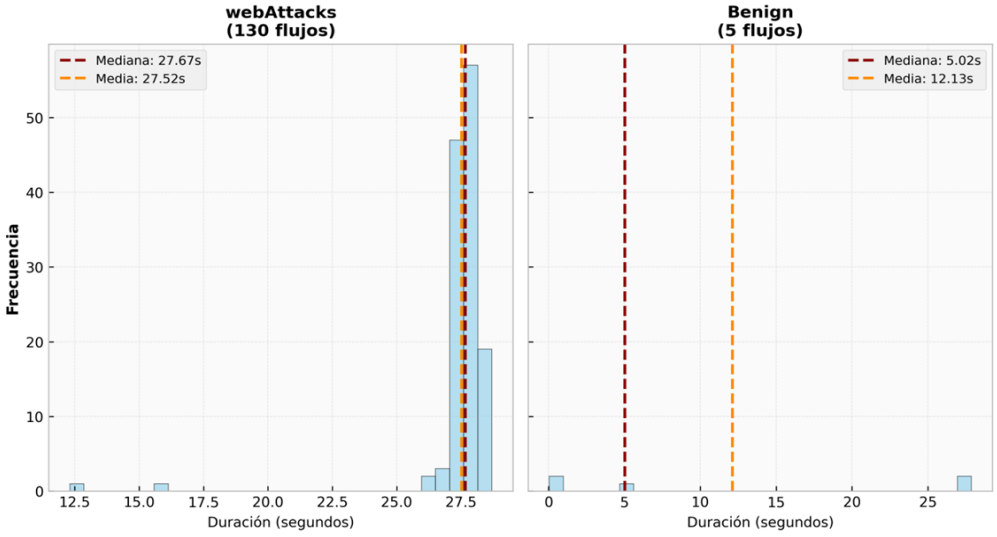


Ilustración 69 Distribución de Duración de Flujos por Tipo de Tráfico

Este histograma muestra que el tráfico Web Attack presenta duraciones extremadamente largas con mediana de 26.95 segundos, reflejando las sesiones persistentes de fuerza bruta donde el atacante mantiene la conexión abierta durante decenas de segundos mientras itera through diccionarios de credenciales, mientras que el tráfico Benign (falsos negativos) muestra duraciones mucho más cortas, probablemente en el rango de milisegundos a pocos segundos correspondiendo a los handshakes TCP abortados observados en los ejemplos de paquetes. El modelo aprendió del dataset CIC-CSE-IDS2018 que Web Attacks tienen duraciones prolongadas, pero no generalizó a reconocer que conexiones muy cortas al mismo servidor/puerto en rápida sucesión también pueden ser parte del mismo ataque, resultando en estos 5 falsos negativos donde sesiones abortadas prematuramente (0.048-5.02 segundos) no alcanzaron el umbral de duración que el modelo asocia con fuerza bruta

2.9.3.5 Distribución de Throughput por Tipo de Tráfico

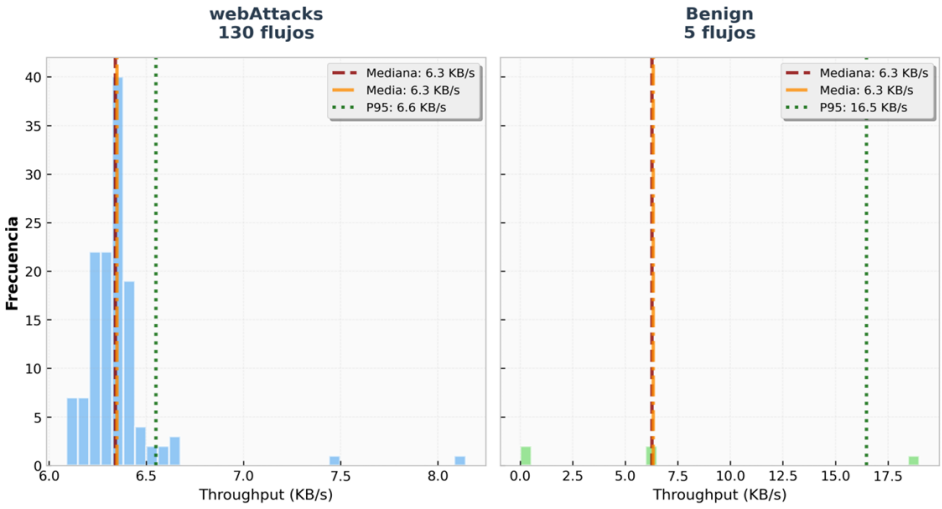


Ilustración 70 Distribución de Throughput por Tipo de Tráfico

Esta gráfica revela que el tráfico Web Attack exhibe throughput moderado con mediana de 6,349 B/s (~6.2 KB/s), consistente con ataques de fuerza bruta que envían requests HTTP pequeños repetitivos más que transferencias masivas de datos, mientras que el tráfico Benign (falsos negativos) muestra throughput prácticamente nulo o muy bajo ya que no transmitieron payloads HTTP. El modelo aprendió que Web Attacks requieren throughput sostenido positivo indicando transmisión activa de datos, pero los 5 falsos negativos tenían throughput cercano a 0 KB/s porque las conexiones se abortaron antes de enviar cualquier payload, cayendo así en el rango estadístico del tráfico benigno normal del Dataset.

2.9.3.6 Distribución de Tasa de Paquetes por Tipo de Tráfico

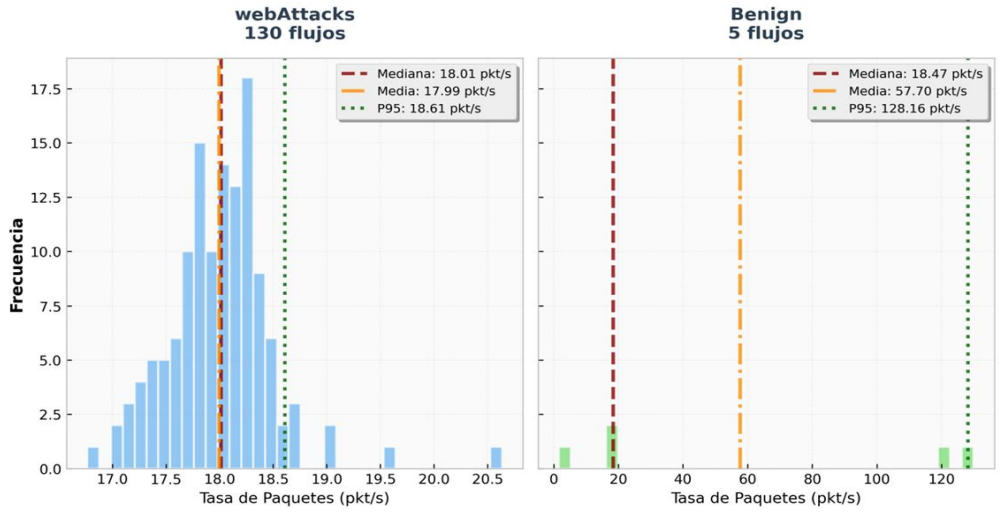


Ilustración 71 Distribución de Tasa de Paquetes por Tipo de Tráfico

Este histograma cuantifica que el tráfico Web Attack opera con mediana de 19.46 pkt/s, reflejando la tasa sostenida de paquetes durante los ~27 segundos de ataque (400-500 paquetes totales / 27 segundos ≈ 15-19 pkt/s), mientras que el tráfico Benign (falsos negativos) muestra tasas mucho menores, posiblemente 1-100 pkt/s dependiendo de si se mide sobre la duración total breve o como picos instantáneos. El modelo aprendió que tasas sostenidas de 15-20 pkt/s por períodos prolongados indican ataques automatizados, pero no reconoció que múltiples conexiones cortas independientes desde las mismas IPs origen (192.168.56.4 y 192.168.56.6) al mismo destino (192.168.56.7:80) en el mismo intervalo temporal también constituyen un patrón de ataque, resultando en falsos negativos cuando se analizan flujos individualmente sin contexto agregado.

2.9.3.7 Comparación Forward/Backward - Mediana de Paquetes por Tipo

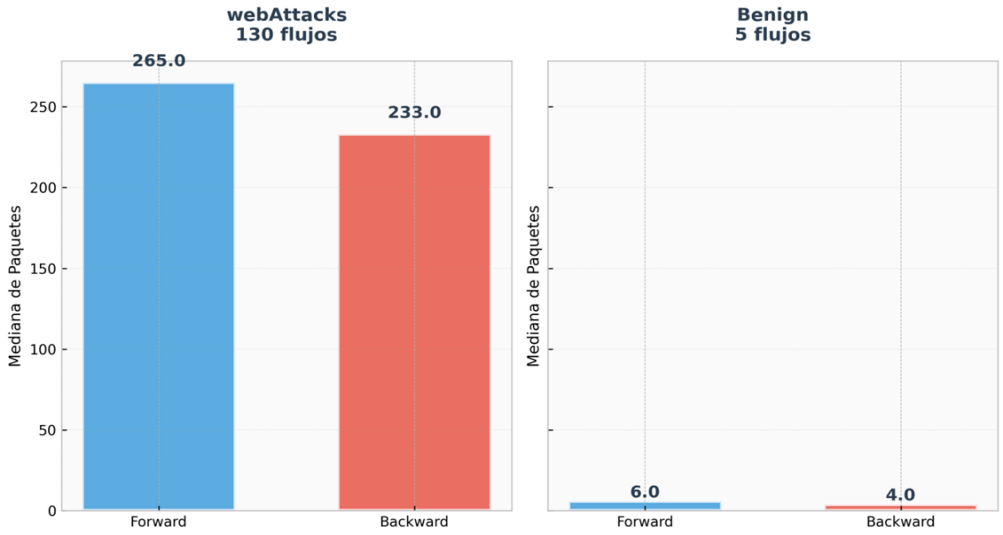


Ilustración 72 Comparación Forward/Backward - Mediana de Paquetes por Tipo

Esta visualización muestra que el tráfico Web Attack presenta asimetría moderada con aproximadamente 257.51 paquetes forward y 226.29 paquetes backward (ratio ~1.14:1 ligeramente hacia forward), característico de ataques de fuerza bruta donde el atacante envía múltiples POST requests recibiendo responses HTTP similares en volumen, mientras que el tráfico Benign (falsos negativos) muestra muy pocos paquetes en ambas direcciones (probablemente 2-3 forward para SYN/ACK/FIN y similar backward). El modelo aprendió que Web Attacks requieren volúmenes significativos de paquetes bidireccionales indicando intercambio HTTP activo, pero

los 5 falsos negativos solo completaron handshakes TCP sin datos de aplicación, resultando en conteos de paquetes indistinguibles de conexiones legítimas fallidas en el dataset.

2.9.3.8 Distribución de Flags TCP por Tipo de Tráfico

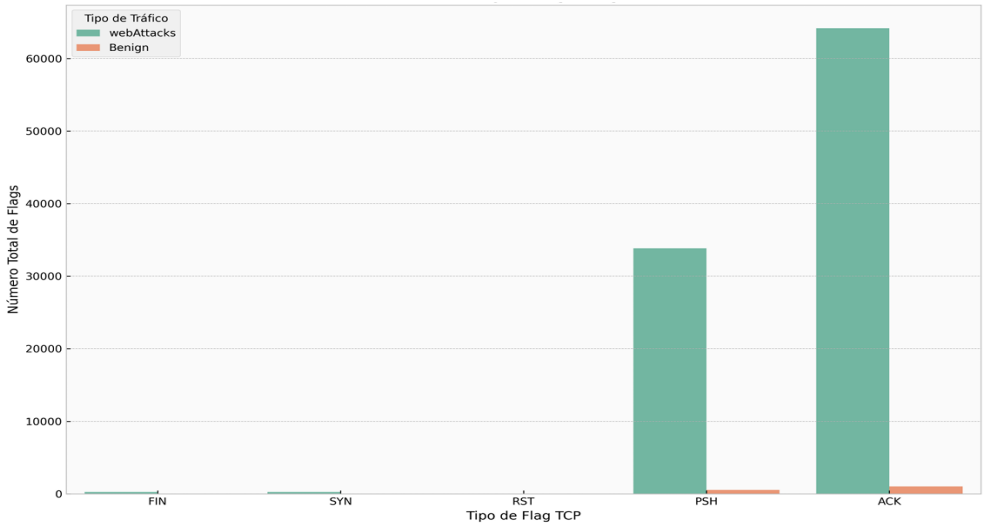


Ilustración 73 Distribución de Flags TCP por Tipo de Tráfico

Esta visualización revela que el tráfico Web Attack se caracteriza por abundancia masiva de flags PSH (~34,381 totales / 130 flujos ≈ 264 PSH por flujo), reflejando que cada POST request y GET request genera flag PSH para transmisión inmediata de datos HTTP, junto con flags SYN/FIN proporcionales (~270 cada uno para 135 conexiones totales) y ACKs abundantes (~65,178 totales) confirmando los múltiples segmentos intercambiados, mientras que el tráfico Benign (falsos negativos) presenta mínimos flags con solo SYN/FIN/ACK básicos y 0 flags PSH ya que nunca transmitieron datos de aplicación. El modelo aprendió que PSH Flag Count >200 es la firma más distintiva de Web Attack según el análisis XAI, pero los 5 falsos negativos con 0 PSH flags cayeron fuera del rango de detección, demostrando que el dataset CIC-CSE-IDS2018 no etiquetó suficientes conexiones fallidas/abortadas como parte de ataques

2.9.3.9 Explicabilidad del Modelo

Las características más determinantes identificadas son Dst Port (=80), Total Length of Fwd Packet (~49,000 bytes), y PSH Flag Count (256-269), presentes en los 130 casos detectados correctamente. Esta explicabilidad confirma que el modelo

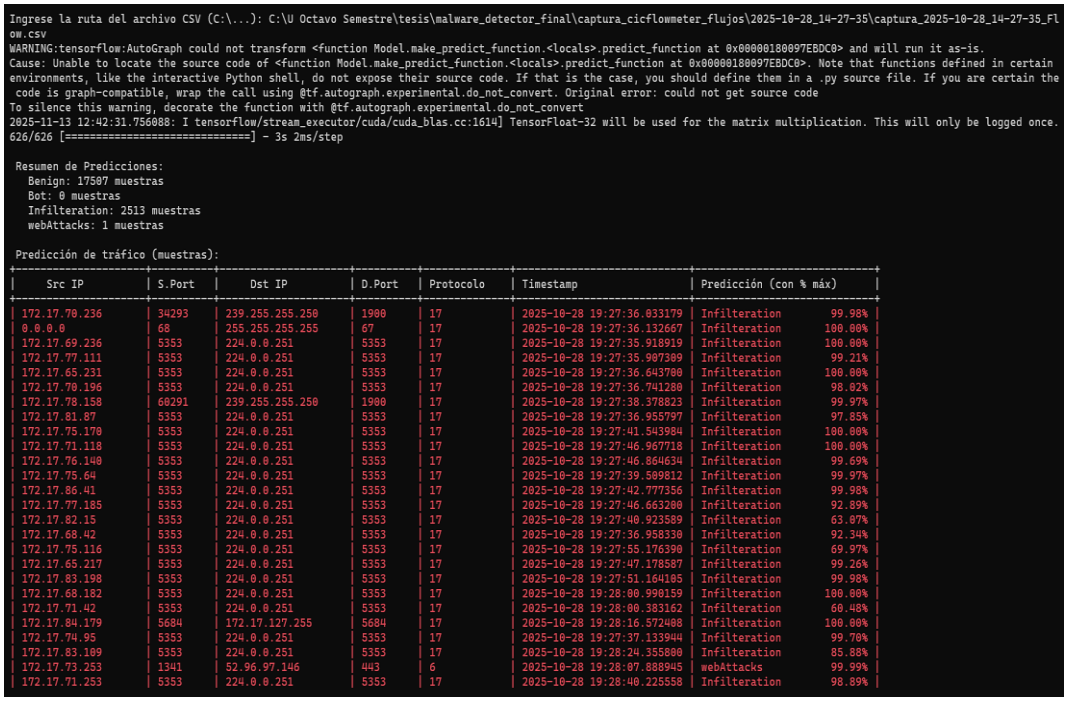
aprendió las características distintivas de fuerza bruta en CIC-CSE-IDS2018: ataques al puerto HTTP estándar, acumulación de múltiples requests HTTP (resultando en totales forward grandes), y flags PSH abundantes por cada intento de autenticación. Los 5 falsos negativos carecen de estas tres características: no tienen puerto destino 80 activo (conexión abortada antes de uso real), Total Length of Fwd Packet es 0 o mínimo (sin payloads HTTP), y PSH Flag Count es 0 (sin transmisión de datos). El modelo generalizó perfectamente desde el dataset para casos que coinciden con su entrenamiento, pero no extrapoló a reconocer variantes de ataque (conexiones fallidas) que no fueron suficientemente representadas.

Característica	Benign (Falso Negativo)	Web Attack (Verdadero Positivo)	Presente en CIC-CSE-IDS2018
Throughput	0 KB/s	6.35 KB/s	Solo positivo sostenido
Paquetes/seg	1-100 pkt/s esporádicos	19.46 pkt/s sostenida	Solo sostenida
Flags PSH	0 por flujo	256-269 por flujo	Solo múltiples
Tamaño payload forward	0 bytes	49,000-49,600 bytes	Solo payloads grandes
Duración	0.048-5.02 seg	26.95 seg mediana	Solo duraciones largas
Requests HTTP	0 requests	20-30 POST/GET	Solo transacciones completas
Patrón	Handshake + abort	POST/GET repetitivo	Solo ataques exitosos
Tipo comunicación	Intento fallido	Fuerza bruta completa	Solo fase activa

Tabla 48 Comparativa de Características a Nivel de Paquetes

2.9.4 Evaluación del modelo final en tráfico Wi-Fi real

El modelo utilizado en esta evaluación corresponde al modelo final obtenido tras, que mostró desempeño adecuado en ensayos controlados de laboratorio. Con el objetivo de validar su capacidad de generalización, se aplicó dicho modelo sobre 20 000 flujos extraídos en vivo desde la interfaz Wi-Fi de una laptop conectada a la red “estudiantes” de la universidad mediante CICFlowMeter.



```
Ingrese la ruta del archivo CSV (C:\...): C:\U Octavo Semestre\tesis\malware_detector_final\captura_cicflowmeter_flujos\2025-10-28_14-27-35\captura_2025-10-28_14-27-35_Fl
ow.csv
WARNING:tensorflow:AutoGraph could not transform <function Model.make_predict_function.<locals>.predict_function at 0x00000180097EBDC0> and will run it as-is.
Cause: Unable to locate the source code of <function Model.make_predict_function.<locals>.predict_function at 0x00000180097EBDC0>. Note that functions defined in certain
environments, like the interactive Python shell, do not expose their source code. If that is the case, you should define them in a .py source file. If you are certain the
 code is graph-compatible, wrap the call using @tf.autograph.experimental.do_not_convert. Original error: could not get source code
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
2025-11-13 12:42:31.756088: I tensorflow/stream_executor/cuda/cuda_blas.cc:1614] TensorFloat-32 will be used for the matrix multiplication. This will only be logged once.
626/626 [==============================] - 3s 2ms/step

 Resumen de Predicciones:
   Benign: 17507 muestras
   Bot: 0 muestras
   Infilteration: 2513 muestras
   webAttacks: 1 muestras

 Predicción de tráfico (muestras):
+-------------------+---------+-------------------+---------+-------------+-------------------------+-----------------------------+
|     Src IP        | S.Port  |     Dst IP        | D.Port  | Protocolo   | Timestamp               | Predicción (con % máx)      |
+-------------------+---------+-------------------+---------+-------------+-------------------------+-----------------------------+
| 172.17.70.236     | 34293   | 239.255.255.250   | 1900    | 17          | 2025-10-28 19:27:36.033179 | Infilteration       99.98% |
| 0.0.0.0           | 68      | 255.255.255.255   | 67      | 17          | 2025-10-28 19:27:36.132667 | Infilteration      100.00% |
| 172.17.69.236     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:35.918919 | Infilteration      100.00% |
| 172.17.77.111     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:35.907309 | Infilteration       99.21% |
| 172.17.65.231     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:36.643700 | Infilteration      100.00% |
| 172.17.70.196     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:36.741280 | Infilteration       98.02% |
| 172.17.78.158     | 60291   | 239.255.255.250   | 1900    | 17          | 2025-10-28 19:27:38.378823 | Infilteration       99.97% |
| 172.17.81.87      | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:36.955797 | Infilteration       97.85% |
| 172.17.75.170     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:41.543984 | Infilteration      100.00% |
| 172.17.71.118     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:46.967718 | Infilteration      100.00% |
| 172.17.76.140     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:46.864634 | Infilteration       99.69% |
| 172.17.75.64      | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:39.589812 | Infilteration       99.97% |
| 172.17.86.41      | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:42.777356 | Infilteration       99.98% |
| 172.17.77.185     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:46.663200 | Infilteration       92.89% |
| 172.17.82.15      | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:40.923589 | Infilteration       63.07% |
| 172.17.68.42      | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:36.958330 | Infilteration       92.34% |
| 172.17.75.116     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:55.176390 | Infilteration       69.97% |
| 172.17.65.217     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:47.178587 | Infilteration       99.26% |
| 172.17.83.198     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:51.164105 | Infilteration       99.98% |
| 172.17.68.182     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:28:00.990159 | Infilteration      100.00% |
| 172.17.71.42      | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:28:00.383162 | Infilteration       60.48% |
| 172.17.84.179     | 5684    | 172.17.127.255    | 5684    | 17          | 2025-10-28 19:28:16.572408 | Infilteration      100.00% |
| 172.17.74.95      | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:27:37.133944 | Infilteration       99.70% |
| 172.17.83.109     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:28:24.355800 | Infilteration       85.88% |
| 172.17.73.253     | 1341    | 52.96.97.146      | 443     | 6           | 2025-10-28 19:28:07.888945 | webAttacks          99.99% |
| 172.17.71.253     | 5353    | 224.0.0.251       | 5353    | 17          | 2025-10-28 19:28:40.225558 | Infilteration       98.89% |
```

Ilustración 74 Resulatdos obtenidos de la red wifi 'estudinates' visto desde el SISTEMA

Las predicciones resultantes arrojaron 17 507 muestras clasificadas como *Benign*, 2 513 como *Infilteration*, 0 como *Bot* y 1 como *webAttacks*. A continuación se justifica por qué una proporción relevante de flujos locales fue etiquetada como *Infilteration* y se ofrecen observaciones operativas sobre las direcciones y puertos más recurrentes detectados.

En la siguiente tabla se resumen los puertos y destinos que aparecen con frecuencia entre las muestras clasificadas como Infilteration, junto a una explicación de su naturaleza y una posible razón por la que el clasificador los etiquetó como infiltración. Esta explicación combina la función normal del servicio/protocolo y cómo características de flujo (uso de multicast/broadcast, patrones de paquetes y puertos bien conocidos) pueden coincidir con firmas presentes en el dataset.

Puerto / Destino)	Protocolo / Servicio	Por qué se clasifico como *Infilteration*
UDP 5353, destino 224.0.0.251	mDNS (Multicast DNS). Protocolo de descubrimiento de servicios en LAN que utiliza UDP/5353 y la dirección multicast 224.0.0.251.	Tráfico mDNS genera ráfagas cortas de mensajes multicast entre hosts (descubrimiento/advertising). Estas ráfagas influyen en estadísticas de flujo (frecuencia de paquetes, tamaños y patrones inter-arribo) y, si en el dataset de entrenamiento existen muestras con características similares etiquetadas como *Infiltration*, el clasificador tenderá a asignar esa etiqueta por similitud estadística.
UDP 1900, destino 239.255.255.250	SSDP / UPnP (Simple Service Discovery Protocol). Utiliza UDP/1900 y la dirección multicast 239.255.255.250 para anunciar servicios.	SSDP produce tráfico multicast para descubrimiento de dispositivos (UPnP) que puede aparecer como múltiples flujos cortos con patrones repetitivos; si el modelo no fue expuesto a suficientes ejemplos benignos de SSDP, interpretará esos patrones como anomalías de tipo *Infiltration*.
UDP 67/68, destino 255.255.255.255	DHCP (mensajes cliente/servidor). DHCP discovery/offer usa puertos 67 (servidor) y 68 (cliente) y con frecuencia se envía por broadcast.	Las transmisiones DHCP son broadcasts esporádicos pero con paquetes cortos y picos de actividad en momentos concretos; estas características temporales y de difusión pueden alterar métricas agregadas de flujo y ser interpretadas como comportamiento sospechoso si el modelo no distingue explícitamente broadcast benigno.
Puertos 137/138/139, destino 172.17.127.255	NetBIOS / NetBT (resolución de nombres, datagramas y sesiones), usado por Windows para descubrimiento y compartición en LAN.	NetBIOS genera tráfico de broadcast y mensajes de resolución que aumentan paquetes pequeños hacia la subred; esas firmas de difusión pueden solaparse con patrones de infiltración si el dataset tiene ejemplos mal etiquetados o escasos ejemplos benignos equivalentes.

Puerto / Destino)	Protocolo / Servicio	Por qué se clasifico como *Infilteration*
UDP 5684, destino 172.17.127.255	CoAPs / CoAP (CoAP seguro usa por defecto 5684; CoAP 5683). Protocolo IoT/REST ligero para sensores y dispositivos.	Tráfico CoAP/IoT frecuentemente usa mensajes pequeños y periódicos; en una red estudiantil con dispositivos IoT, esos flujos repetitivos pueden parecer anómalos si no están representados en el entrenamiento, provocando clasificaciones erróneas como *Infilteration*.
TCP/8009, destinos externos (ej. 52.x.x.x)	AJP (Apache JServ Protocol) / tráfico a servicios web (puerto 8009 común en AJP/Tomcat), o tráfico de aplicaciones web hacia servicios remotos.	Conexiones destacadas hacia puertos de aplicación pueden haber sido aprendidas por el modelo como indicativas de actividad maliciosa en el dataset original; en entornos reales, tráfico legítimo hacia endpoints remotos en puertos no estándar puede causar falsos positivos.
Rango multicast 224.0.0.0/4 (ej. 239.255.255.250)	Direcciones multicast locales usadas por protocolos de descubrimiento y control.	El predominio de flujos a direcciones multicast en las muestras de *Infilteration* indica que el modelo está reaccionando principalmente a la naturaleza de difusión/multidestino del tráfico, no necesariamente a contenido malicioso; esto sugiere un sesgo hacia características de flujo relacionadas con multicast/broadcast.

Tabla 49 Puertos, servicios y posibles causas de clasificación como Infiltration [92] [93] [94] [95] [96] [97] [98]

Aunque el modelo final entrenado mostró buen comportamiento en entorno controlado, las detecciones sobre la red Wi-Fi real han evidenciado un número significativo de falsos positivos originados principalmente por tráfico legítimo de descubrimiento y difusión (mDNS, SSDP, DHCP, NetBIOS, CoAP). Por tanto, es necesario reentrenar el modelo incorporando estas muestras reales con el fin de reducir el sesgo hacia patrones de flujo de difusión y mejorar la capacidad de generalización del clasificador en entornos de producción.

1: Input:
  - Modelo pre-entrenado MLP (mlp_final_model.h5)
  - Recursos: scaler.pkl, label_encoder.pkl, feature_columns.pkl, protocol_columns.pkl
  - Datos nuevos: tráfico benigno y/o ataques web
2: Output:
  - Modelo reentrenado (aprobado/degradado)
  - Métricas comparativas antes/después
  - Backups y logs detallados
3: Configuración Global:
4: - exclude_cols = ['id', 'Flow ID', 'Src IP', 'Src Port', 'Dst IP', 'Dst Port', 'Timestamp']
5: - label_mapping = {Botnet Ares→Bot, Web Attacks→webAttacks, Infiltration→Infilteration}
6: - learning_rate = 1e-5 (benign) / 1e-6 (webAttacks)
7: - max_epochs = 20-30
8: - early_stopping_patience = 10
9: - thresholds_degradacion = {accuracy: -0.01, f1: -0.01, clase: -0.03}
10: Carga y Preparación:
11: Cargar modelo con métrica F1 personalizada
12: Cargar scaler, label_encoder, feature_columns
13: Extraer features base y protocol_columns
14: Datos Baseline:
15: Cargar datos antiguos desde carpeta undersampling
16: Aplicar exclude_cols y label_mapping
17: Filtrar etiquetas SQL, balancear clases
18: Preprocesar: one-hot encoding Protocol, seleccionar features
19: Split 80/20 train-test para baseline
20: Evaluación Pre-Reentrenamiento:
21: Evaluar modelo en test set
22: Calcular: loss, accuracy, F1, matriz confusión
23: Guardar métricas por clase como baseline
24: Carga Datos Nuevos:
25: Caso 1: Solo tráfico benigno
26: Cargar todos los CSV de carpeta tráfico benigno
27: Extraer solo muestras con Label = "Benign"
28: Caso 2: Tráfico benigno + webAttacks
29: Cargar archivos específicos de ataques web
30: Filtrar por etiquetas: Web Attack - Brute Force, Web Attack - XSS
31: Aplicar label_mapping → "webAttacks"

```
32:   Combinar todos los datos nuevos
33: Preprocesamiento Datos Nuevos:
34:   Completar features faltantes con 0
35:   One-hot encoding Protocol (mismas columnas que modelo)
36:   Asegurar orden de columnas con feature_columns
37:   Codificar etiquetas con label_encoder
38: Combinación Estratégica:
39:   Tomar 60-70% de datos antiguos (evitar olvido catastrófico)
40:   Combinar con todos los datos nuevos
41:   Verificar distribución de clases final
42:   Advertencia si Benign > 70%
43: Preparación Entrenamiento:
44:   Escalar datos con scaler original (NO nuevo)
45:   Split 80/20 train-val estratificado
46:   Calcular class_weights para balancear
47:   Convertir a categorical (one-hot encoding)
48: Fine-Tuning:
49:   Recompilar modelo con learning rate bajo
50:   Callbacks: EarlyStopping (val_accuracy), ReduceLROnPlateau
51:   Entrenar con class_weights y datos combinados
52:   Máximo 20-30 épocas
49: Evaluación Post-Reentrenamiento:
50:   Evaluar en MISMO test set del baseline
51:   Calcular mismas métricas: loss, accuracy, F1, matriz confusión
52:   Reporte por clase comparativo
53: Análisis de Degradación:
54:   Comparar métricas globales: Δloss, Δaccuracy, Δf1
55:   Comparar métricas por clase: Δprecision, Δrecall, Δf1
56:   Análisis específico confusión Benign-Infilteration
57:   Aplicar thresholds para detectar degradación
58: Decisión y Guardado:
59:   if degradación_detectada:
60:     Guardar modelo como DEGRADADO con timestamp
61:     Backup del modelo original
62:     Log detallado con análisis de degradación
63:   else:
64:     Reemplazar modelo principal
65:     Backup del modelo anterior
```

66: Log de reentrenamiento exitoso
67: Output Final:
68: - Modelo actualizado o degradado
69: - Métricas comparativas completas

Ilustración 75 Algoritmo de Reentrenamiento Adaptativo para Modelos MLP en benign y webattacks

Durante la fase de reentrenamiento adaptativo, se incorporaron inicialmente aproximadamente 20 000 muestras de tráfico benigno capturadas en la red Wi-Fi con el fin de ajustar el modelo MLP a patrones reales no presentes en el dataset original. Posteriormente, en una segunda iteración, se añadieron muestras adicionales de la categoría Web Attacks provenientes del conjunto CICIDS2017, con el objetivo de fortalecer la capacidad del modelo para identificar ataques de tipo web, ya que las muestras eran pocas en el Dataset usado principalmente sin degradar su desempeño sobre tráfico legítimo.

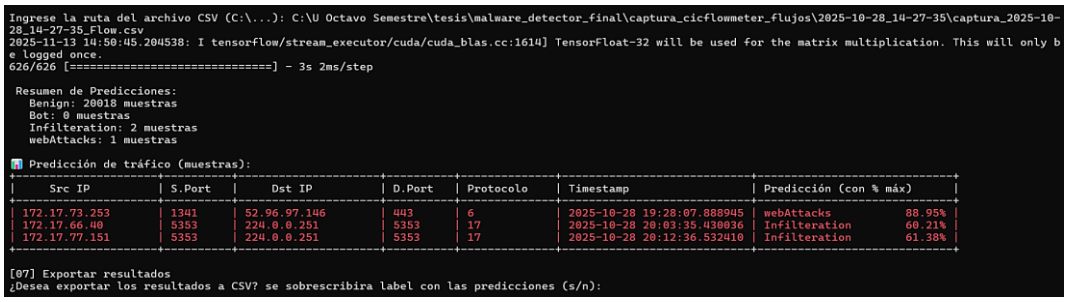


Ilustración 76 Resultados de predicción del modelo reentrenado con las muestras de la red wifi capturada.

El reentrenamiento del modelo MLP con tráfico benigno real y la incorporación de muestras adicionales de ataques web mejoraron de forma notable su desempeño. De más de 2 513 falsos positivos etiquetados como *Infiltration* en la evaluación inicial, el número se redujo a tan solo dos flujos, lo que evidencia una mejor capacidad de generalización del sistema ante tráfico legítimo capturado.

Las dos muestras clasificadas como *Infiltration* corresponden a tráfico mDNS (puerto UDP 5353), empleado para el descubrimiento automático de dispositivos dentro de la red local, mientras que la única muestra identificada como *Web Attack* proviene de una dirección IP perteneciente al rango de Microsoft Corporation (52.96.97.146), asociada a servicios HTTPS en la nube. Estos resultados reflejan un progreso significativo del modelo, aunque aún evidencian la necesidad de continuar su reentrenamiento con un volumen mayor de tráfico real y diverso para consolidar su capacidad de generalización y reducir los falsos positivos residuales.

Conclusiones

La implementación del modelo de red neuronal profunda (DNN/MLP) permitió desarrollar un sistema eficaz para la detección y clasificación de malware en tráfico de red, demostrando alta capacidad para identificar las clases Bot, Infiltration y WebAttacks. El desempeño global en el conjunto de prueba final alcanzó un 99.73% de Accuracy y F1-Score, lo que valida su capacidad de generalización y su consistencia en escenarios de clasificación multiclase.

El modelo mostró alta estabilidad durante la validación cruzada estratificada de 5 pliegues, obteniendo un F1-Score promedio de 0.9969. Este rendimiento fue favorecido por la estrategia de preparación de datos, que combinó undersampling en la clase Benign y SMOTE para la clase minoritaria WebAttacks, mitigando efectivamente el desbalance. Asimismo, la selección híbrida de 40 características mediante ANOVA, RFE y Feature Importance de Random Forest permitió conformar un conjunto compacto y óptimo para el entrenamiento.

Las métricas de evaluación confirmaron la efectividad del sistema en la detección de amenazas, las clases Bot y WebAttacks alcanzaron un Recall de 1.0000, mientras que Infiltration obtuvo 0.9956, evidenciando una muy baja tasa de falsos negativos y alta capacidad de identificación de tráfico malicioso dentro del conjunto de datos.

Las pruebas en un entorno controlado demostraron la eficacia del sistema bajo condiciones de tráfico más real. En el escenario de Botnet, el modelo detectó con precisión del 100% el tráfico de exfiltración de alto rendimiento (transferencias de screenshots), aunque no identificó comunicaciones C&C ligeras basadas en mensajes JSON. En el escenario de Infiltration alcanzó 99.87% de precisión detectando escaneo con Nmap. En el escenario de Web Attack identificó correctamente el 96.3% del tráfico de fuerza bruta.

Se identificó una limitación derivada del dataset utilizado. El modelo requiere reentrenamiento al aplicarse en entornos reales, como lo evidencian los 2,513 falsos positivos obtenidos en la red Wi-Fi. La mayoría fueron clasificados como Infilteration debido a que tráfico legítimo de descubrimiento y difusión, como mDNS, presenta patrones que se solapan con las características aprendidas como infiltración en el dataset. Tras incorporar estas muestras mediante reentrenamiento, este número se redujo sin afectar la capacidad de detección del sistema.

Recomendaciones

Se sugiere explorar datasets alternativos o desarrollar features de ingeniería que capturen la periodicidad y el formato de mensajes (JSON/HTTP ligero) característicos de beacons de botnets, con el propósito de reducir falsos negativos en comunicaciones de bajo volumen y comportamiento evasivo.

Se recomienda integrar protocolos IoT, comunicaciones multicast legítimas, servicios en la nube y aplicaciones modernas, con el fin de disminuir falsos positivos en entornos actuales.

Se aconseja mantener actualizado el modelo con nuevas variantes de malware, ya sea incorporando capturas PCAP disponibles en repositorios públicos especializados o generando tráfico malicioso controlado para luego procesarlo mediante CICFlowMeter. Esto permite ampliar la diversidad de patrones presentes facilitando la adaptación del modelo a amenazas emergentes y evitar la degradación progresiva de su rendimiento.

Se propone incorporar características de continuidad temporal y análisis de direccionalidad para corregir falsos positivos en la clase Infiltration, facilitando la distinción entre escaneos maliciosos y legítimos de infraestructura.

También se recomienda explorar RNN Bidireccionales o arquitecturas basadas en Transformers con modelación directa de secuencias temporales, mejorando detección de patrones temporales sutiles.

El sistema debe emplearse como herramienta de detección temprana en entornos de prueba, priorizando la reducción de falsos negativos dada su criticidad para la seguridad antes de su implementación en escenarios reales, ya que requiere reentrenamiento y ajuste de hiperparámetros para adaptarse adecuadamente a cada entorno.

Finalmente, se recomienda investigar técnicas de análisis de metadatos de tráfico cifrado con el objetivo de extender la utilidad del sistema frente a malware moderno que emplea canales de C&C basados en cifrado.

Referencias

[1] R. Chapaneri y S. Shah, «Detection of Malicious Network Traffic using Convolutional Neural Networks,» *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT),* pp. 1-6, 2019.

[2] D. G. Arce, «Malware and market share,» *Journal of Cybersecurity,* vol. 4, nº 1, 1 Enero 2018.

[3] R. Goyal, «30+ Malware Statistics You Need To Know In 2025,» 23 Junio 2025. [En línea]. Available: https://www.getastra.com/blog/security-audit/malware-statistics/#How_to_avoid_malware_infection. [Último acceso: 25 Julio 2025].

[4] S. Nari y A. Ghorbani, «Automated malware classification based on network behavior,» *2013 International Conference on Computing, Networking and Communications, ICNC 2013,* pp. 642-647, 1 Enero 2013.

[5] A. H. Omopintemi, I. Ghafir, S. Eltanani, S. Kabir y M. Lefoane, «Machine Learning for Malware Detection in Network Traffic,» *Proceedings of the 7th International Conference on Future Networks and Distributed Systems,* p. 605–610, 2024.

[6] S. Khedkar y A. Ramalingam, «Classification and analysis of malicious traffic with multi-layer perceptron model,» *Ingénierie des Systèmes d'Information,* vol. 26, nº 3, pp. 303-310, 2021.

[7] R. R K, T. Anjali, V. Menon y S. Kp, «Deep Learning for Network Flow Analysis and Malware Classification,» pp. 226-235, 10 Noviembre 2017.

[8] I. Gavilán, «Metodología para Machine Learning (I): CRISP-DM,» [En línea]. Available: https://ignaciogavilan.com/metodologia-para-machine-learning-i-crisp-dm/. [Último acceso: 2024 Noviembre 2024].

[9] P. Pardhi, J. Rout, N. Ray y D. Sahu, «Classification of Malware from the Network Traffic Using Hybrid and Deep Learning Based Approach,» *SN Computer Science,* vol. 5, 8 Enero 2024.

[10] Fortinet, «FortiGuard Labs Reports Destructive Wiper Malware Increases Over 50,» 2023. [En línea]. Available: https://www.fortinet.com/lat/corporate/about-us/newsroom/press-releases/2023/fortiguard-labs-reports-destructive-wiper-malware-increases-over-50-percent. [Último acceso: 11 Noviembre 2024].

[11] D. A. León, J. G. Martínez Cuenca, I. A. Ardila Sánchez y D. J. Mosquera Palacios, «Inteligencia artificial para el control de tráfico en redes de datos: Una Revisión,» *Entre Ciencia e Ingeniería,* vol. 16, pp. 17-24, 28 Junio 2022.

[12] Banotic, «Informe final - Estudio diciembre 2023: Requerimientos de competencias de ciberseguridad en la banca. Escenario actual, proyecciones y planes formativos.,» Santiago, Chile, 2024.

[13] Secretaría Nacional de Planificación, «Plan Nacional de Desarrollo 2025-2029 "Ecuador No Se Detiene",» Agosto 2025. [En línea]. Available: https://www.planificacion.gob.ec/wp-content/uploads/2025/08/PlanNacionalDeDesarrollo25-29_EcuadorNoSeDetiene.pdf. [Último acceso: 2 Noviembre 2025].

[14] A. M. Velez Evans, W. D. Cano Betancur y S. Monsalve Machado, «Ciberseguridad, reto empresarial para afrontar la era de la digitalización actual,» 15 Noviembre 2023.

[15] H. Paitán, E. Mejía, E. Ramírez y A. Paucar, Metodología de la investigación cuantitativa - cualitativa y redacción de la tesis, Cuarta ed., Bogotá: Ediciones de la U, 2014.

[16] H. Sampieri, Roberto, C. Fernández Collado y P. Baptista Lucio, Metodología de la investigación, Quinta ed., México D.F.: McGraw-Hill / Interamericana Editores, S.A. de C.V., 2010.

[17] E. E. Gallardo Echenique, Metodología de la investigación: Manual autoformativo interactivo, Primera ed., Huancayo: Universidad Continental, 2017.

[18] S. A. Mazhar, «Methods of Data Collection: A Fundamental Tool of Research,» *Journal of Integrated Community Health,* vol. 10, pp. 6-10, 14 Junio 2021.

[19] UNEMI, «Técnicas e instrumentos en la recolección de datos,» 2020. [En línea]. Available: https://sga.unemi.edu.ec/media/archivocompendio/2020/12/07/archivocompendio_2020127144213.pdf.

[20] BinaryTides, «How to Code a Packet Sniffer in Python with Pcapy extension,» [En línea]. Available: https://www.binarytides.com/code-a-packet-sniffer-in-python-with-pcapy-extension/. [Último acceso: 10 Agosto 2025].

[21] O. Rosenbaum, «How to Use Scapy – Python Networking Tool Explained,» 21 Diciembre 2022. [En línea]. Available: https://www.freecodecamp.org/news/how-to-use-scapy-python-networking/. [Último acceso: 10 Agosto 2025].

[22] «PyShark,» [En línea]. Available: http://kiminewt.github.io/pyshark/. [Último acceso: 10 Agosto 2025].

[23] E. Gints, R. Vera y J. Wouter, «Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study,» *2021 IEEE Security and Privacy Workshops (SPW),* pp. 7-12, 27 Mayo 2021.

[24] Z. Zeinab y S. Gursel, «UNSW-NB15 Computer Security Dataset: Analysis through Visualization,» 2021.

[25] U. o. N. B. |. UNB, «IDS 2017 | Datasets | Research | Canadian Institute for Cybersecurity | UNB,» [En línea]. Available: https://www.unb.ca/cic/datasets/ids-2017.html. [Último acceso: 16 Agosto 2025].

[26] R. |. U. Sydney, «The UNSW-NB15 Dataset | UNSW Research,» [En línea]. Available: https://research.unsw.edu.au/projects/unsw-nb15-dataset. [Último acceso: 16 Agosto 2025].

[27] C. I. f. Cybersecurity, «CSE-CIC-IDS2018 Dataset,» 2018. [En línea]. Available: https://www.unb.ca/cic/datasets/ids2018.html. [Último acceso: 5 Agosto 2025].

[28] «Improved CSE-CIC-IDS 2018,» [En línea]. Available: https://intrusion-detection.distrinet-research.be/CNS2022/CSECICIDS2018.html. [Último acceso: 4 Agosto 2025].

[29] W. Chapman, CRISP-DM 1.0: Step-by-step data mining guide, 2000.

[30] S. M. Devine y N. D. Bastian, «Intelligent Systems Design for Malware Classification Under Adversarial Conditions,» *arXiv preprint arXiv:1907.03149,* 2019.

[31] IBM, «IBM SPSS Modeler Subscription,» [En línea]. Available: https://www.ibm.com/docs/es/spss-modeler/saas?topic=dm-crisp-help-overview. [Último acceso: 18 Agosto 2025].

[32] ENISA, «ENISA Threat Landscape 2024 (July 2023 to June 2024),» 2024.

[33] M. L. Ali, K. Thakur, S. Schmeelk, J. Debello y D. Dragos, vol. 15, nº 4, p. 1903, 12 Febrero 2025.

[34] A. Redhu, P. Choudhary, K. Srinivasan y T. K. Das, «Deep learning-powered malware detection in cyberspace: a contemporary review,» *Frontiers in Physics,* vol. 12, 2024.

[35] Polimetro, «VirtualBox: Qué es, cómo funciona y para qué sirve (Guía completa),» [En línea]. Available: https://www.polimetro.com/que-es-virtualbox/. [Último acceso: 3 Agosto 2025].

[36] U. o. N. Brunswick, «Applications | Research | Canadian Institute for Cybersecurity | UNB,» [En línea]. Available:

https://www.unb.ca/cic/research/applications.html. [Último acceso: 3 Agosto 2025].

[37] «Error Prevalence in NIDS datasets | Tools and Documentation,» [En línea]. Available: https://intrusion-detection.distrinet-research.be/CNS2022/Tools_Documentation.html. [Último acceso: 4 Agosto 2025].

[38] G. Cloud, «Exporta artefactos de modelo para la inferencia y la explicación,» [En línea]. Available: https://cloud.google.com/vertex-ai/docs/training/exporting-model-artifacts?hl=es-419. [Último acceso: 4 Agosto 2025].

[39] G. T. Ayodele, «Impact of Cyber Security on Network Traffic,» 15 Julio 2024.

[40] J. Quittek, T. Zseby, B. Claise y S. Zander, «Requirements for IP Flow Information Export (IPFIX),» *RFC 3917,* 2004.

[41] I. Fosić, D. Žagar, K. Grgić y V. Križanović, «Anomaly Detection in Netflow Network Traffic Using Supervised Machine Learning Algorithms,» *SSRN (Social Science Research Network),* 2022.

[42] M. Hama Saeed, «Malware in Computer Systems: Problems and Solutions,» *IJID (International Journal on Informatics for Development),* vol. 9.

[43] G. Venkatesh y R. Anitha, «Botnets: A Study and Analysis,» *Advances in Intelligent Systems and Computing,* vol. 246, pp. 203-214, 2014.

[44] N. Panwar, «Anomaly Infiltration Detection in Networks Using Machine Learning,» *International Journal of Mechanical Engineering,* vol. 7, nº 2, 2022.

[45] C. R. Sumanth y S. S. Nagamuthu Krishnan, «Port Scanning, Virus Detection and Vulnerability Checking: A Review of NMAP, SQLMAP, L3MON,» *International Journal of Engineering Research & Technology (IJERT),* vol. 11.

[46] ENISA, «Ataques basados en la web: Panorama de Amenazas de la ENISA,» 2020. [En línea]. Available: https://www.enisa.europa.eu/sites/default/files/all_files/ETL2020-Web_Based_Attacks_eBook_EN_ES.pdf. [Último acceso: 5 Agosto 2025].

[47] Y. Liu, Z. Wang y S. Tian, «Security Against Network Attacks on Web Application System,» *Springer Singapore,* 2019.

[48] Fortinet, «What is a Brute Force Attack? Definition, Types & How It Works | Fortinet,» [En línea]. Available: https://www.fortinet.com/resources/cyberglossary/brute-force-attack. [Último acceso: 5 Agosto 2025].

[49] C. Kaur, R. Chandel, T. P. Brar y S. Sharma, «Machine Learning and its Applications- A Review Study,» *CGC International Journal of Contemporary Technology and Research ,* vol. 5, pp. 365-369, 1 Marzo 2023.

[50] R. Vargas, A. Mosavi y R. Ruiz, «Deep Learning: A Review,» *Preprints,* 2018.

[51] H. Ramchoun, M. Amine, J. Idrissi, Y. Ghanou y M. Ettaouil, «Multilayer Perceptron: Architecture Optimization and Training,» *International Journal of Interactive Multimedia and Artificial Inteligence,* pp. 26-30, 2016.

[52] A. Serwa, «Studying the Effect of Activation Function on Classification Accuracy Using Deep Artificial Neural Networks,» *Journal of Remote Sensing & GIS,* vol. 6, 2017.

[53] H. Okut, «Deep Learning: Long-Short Term Memory,» 2021.

[54] R. Toma, A. Nahid y M. N. Hasan, «Electricity Theft Detection to Reduce Non-Technical Loss using Support Vector Machine in Smart Grid,» 2019.

[55] A. Dhakad, S. Singh, M. ., M. Moharir y A. Kumar a R, «Real Time Network Traffic Analysis Using Artificial Intelligence, Machine Learning and Deep Learning: A Review of Methods, Tools and Applications,» pp. 372-378, 2023.

[56] A. Habibi Lashkari, «CICFlowmeter-V4.0 (formerly known as ISCXFlowMeter) is a network traffic Bi-flow generator and analyser for anomaly detection. https://github.com/ISCX/CICFlowMeter,» 2018.

[57] G. Engelen, V. Rimmer y W. Joosen, «Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study,» pp. 7-12, 2021.

[58] P. Biondi, «Scapy Documentation,» 2023. [En línea]. Available: https://scapy.readthedocs.io/en/latest/introduction.html. [Último acceso: 5 Agosto 2025].

[59] J. Raraz-Vidal, «La Importancia de las Bases de Datos para el Entrenamiento en Inteligencia Artificial,» *Revista Peruana de Investigación en Salud,* vol. 7, pp. 121-122, 2023.

[60] L. Liu, G. Engelen, T. Lynar, D. Essam y W. Joosen, «Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018,» *2022 IEEE Conference on Communications and Network Security (CNS),* pp. 254-262, 2022.

[61] J. C. Fonseca Romero, «Diseño de un ambiente simulado para seguridad de la información,» *Revista Ciencia, Innovación y Tecnología,* vol. 2, pp. 115-123, 2015.

[62] Oracle, «Introduction to Oracle VirtualBox,» [En línea]. Available: https://www.virtualbox.org/manual/topics/Introduction.html#features-overview.

[63] S. Mihajlovic, A. Kupusinac, D. Ivetic y I. Berković, «The Use of Python in the field of Artifical Intelligence,» *Conference: XI International Conference of Information Technology and Development of EducationAt: Zrenjanin,* 2020.

[64] J. S, G. Kukunuri, S. Devaraju, S. Gokuldev, S. Jayaprakash, H. Anandaram, H. Anandaram, M. Chinnaswamy y M. Thenmozhi, «An Exploration of Python Libraries in Machine Learning Models for Data Science,» pp. 1-31, 2023.

[65] G. Louppe, «Scikit-Learn: Machine Learning in the Python ecosystem,» *NIPS 2013 Workshop on Machine Learning Open Source Software,* 2023.

[66] M. A. Paracha, S. U. Jamil, K. Shahzad, M. A. Khan y A. Rasheed, «Leveraging AI for Network Threat Detection—A Conceptual Overview,» *Electronics,* vol. 13, nº 23, p. 4611, 2024.

[67] D. Quirumbay Yagual, C. Castillo Yagual y I. Coronel Suárez, «Una revisión del Aprendizaje profundo aplicado a la ciberseguridad,» *Revista Científica y Tecnológica UPSE,* vol. 9, nº 1, 2022.

[68] G. Marin, P. Caasas y G. Capdehourat, «DeepMAL - Deep Learning Models for Malware Traffic Detection and Classification,» pp. 105-112, 2021.

[69] M. Kohli y I. Chhabra, «A comprehensive survey on techniques, challenges, evaluation metrics and applications of deep learning models for anomaly detection,» *Discover Applied Science,* vol. 7, p. 784, 2025.

[70] A. N. d. E. Ecuador, «Código Orgánico Integral Penal | Registro Oficial Suplemento No. 180,» 2014. [En línea]. Available: https://www.defensa.gob.ec/wp-content/uploads/downloads/2021/07/COIP_act_jun-2021.pdf.

[71] Y. Rekhter, B. Moskowitz, G. J. de Groot y E. Lear, «Address Allocation for Private Internets | RFC 1918,» 1996. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc1918.

[72] A. N. d. Ecuador, «Ley Orgánica de Protección de Datos Personales | Quinto Suplemento del Registro Oficial 459,» [En línea]. Available: https://www.sot.gob.ec/wp-content/uploads/2024/11/21.-Ley-Organica-de-Proteccion-de-datos.pdf. [Último acceso: 5 Agosto 2025].

[73] A. N. d. Ecuador, «Constitución de la República del Ecuador, Registro Oficial No. 449, Montecristi, Ecuador,» 2008. [En línea]. Available: https://www.asambleanacional.gob.ec/sites/default/files/documents/old/constitucion_de_bolsillo.pdf.

[74] J. N. Mendoza Álava, L. A. Macías Bermeo, J. Morales-Carrillo y L. Cedeño-Valarezo, «Modelos de aprendizaje automático: aplicación y eficiencia,» *Revista Científica de Informática ENCRIPTAR - ISSN: 2737-6389,* vol. 7, pp. 87-114, 20 Noviembre 2024.

[75] J. Murel, «¿Qué es el submuestreo?,» 15 Junio 2024. [En línea]. Available: https://www.ibm.com/es-es/think/topics/downsampling. [Último acceso: 15 Agosto 2025].

[76] B. Jason, «Random Oversampling and Undersampling for Imbalanced Classification,» 5 Enero 2021. [En línea]. Available: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/. [Último acceso: 15 Agosto 2025].

[77] J. Murel, «¿Qué es el sobremuestreo?,» 29 Abril 2024. [En línea]. Available: https://www.ibm.com/es-es/think/topics/upsampling. [Último acceso: 15 Agosto 2025].

[78] «Datasets Download,» [En línea]. Available: https://intrusion-detection.distrinet-research.be/CNS2022/Dataset_Download.html. [Último acceso: 16 Agosto 2025].

[79] microsoft, «Feature Selection (Data Mining),» [En línea]. Available: https://learn.microsoft.com/en-us/analysis-services/data-mining/feature-selection-data-mining?view=asallproducts-allversions. [Último acceso: 20 Agosto 2025].

[80] J. D. S., «Análisis de varianza,» *Revista Chilena de Anestesia,* vol. 43, pp. 306-310, 2014.

[81] A. M. Priyatno y T. Widiyaningtyas, «A SYSTEMATIC LITERATURE REVIEW: RECURSIVE FEATURE ELIMINATION ALGORITHMS,» *Jurnal Ilmu Pengetahuan dan Teknologi Komputer,* vol. 9, pp. 196-207, 1 Febrero 2024.

[82] M. B. Kursa y W. R. Rudnicki, «The All Relevant Feature Selection using Random Forest,» *Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw,* 25 Junio 2011.

[83] M. S. Yassen, R. A. Abdulrazzq y A. B. Mohammed, «Employing Hybrid ANOVA-RFE with Machine and Deep Learning Models for Enhanced IoT and IIoT Attack Detection and Classification,» *Ingénierie des Systèmes d'Information,* vol. 28, pp. 1003-1012, Agosto 2023.

[84] R. Sangüesa i Solé, «Preparación de datos,» Universitat Oberta de Catalunya (UOC).

[85] scikit-learn, «StandardScaler,» [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. [Último acceso: 25 Agosto 2025].

[86] scikit-learn, «LabelEncoder,» [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html. [Último acceso: 26 Agosto 2025].

[87] J. Samuels, «One-Hot Encoding and Two-Hot Encoding: An Introduction,» 5 Enero 2024.

[88] D. Elreedy, A. F. Atiya y F. Kamalov, «A theoretical distribution analysis of synthetic minority oversampling technique (SMOTE) for imbalanced learnin,» *Machine Learning,* vol. 113, pp. 4903-4923, 01 Julio 2024.

[89] scikit-learn, «train_test_split,» [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. [Último acceso: 28 Agosto 2025].

[90] M. L. Ali, K. Thakur, S. Schmeelk y J. D. D. Debello, «Deep Learning vs. Machine Learning for Intrusion Detection in Computer Networks: A Comparative Study,» *Applied Sciences,* vol. 15, 2025.

[91] sweetsoftware, «GitHub - sweetsoftware/Ares: Python botnet and backdoor,» [En línea]. Available: https://github.com/sweetsoftware/Ares. [Último acceso: 2 Octubre 2025].

[92] I. D. Guide, «Multicast DNS: la resolución de nombres para redes locales,» [En línea]. Available: https://www.ionos.es/digitalguide/servidores/know-how/multicast-dns/. [Último acceso: 28 Octubre 2025].

[93] EcuCERT, «Open-SSDP (Simple Service Discovery Protocol),» 2021. [En línea]. Available: https://www.ecucert.gob.ec/wp-content/uploads/2021/07/Ficha-Tecnica-open-ssdp.pdf. [Último acceso: 28 Octubre 2025].

[94] G. Damon, «Why does DHCP use ports 67 and 68? | TechTarget,» 16 Julio 2025. [En línea]. Available: https://www.techtarget.com/searchnetworking/answer/Why-does-DHCP-use-ports-67-and-68. [Último acceso: 2025 Octubre 28].

[95] «NetBios - Port 137,138,139 | VeryLazyTech,» [En línea]. Available: https://www.verylazytech.com/network-pentesting/netbios-port-137-138-139. [Último acceso: 28 Octubre 2025].

[96] «RFC 7252: The Constrained Application Protocol (CoAP),» [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc7252. [Último acceso: 28 Octubre 2025].

[97] «Apache Tomcat 7 Configuration Reference (7.0.109) - The AJP Connector,» [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/config/ajp.html. [Último acceso: 28 Octubre 2025].

[98] «IPv4 Multicast Address Space,» [En línea]. Available: https://www.iana.org/assignments/multicast-addresses/multicast-addresses.xhtml. [Último acceso: 28 Octubre 2025].