Prácticas Profesionales Supervisadas

Leandro Estrella

Desarrollo de un señuelo informático (honeypot) con capacidades extendidas de análisis de tráfico mediante técnicas de aprendizaje automático

Instituto de Ingeniería y Agronomía

2025

Carrera: Ingeniería en Informática

INSTITUTO DE INGENIERÍA Y AGRONOMÍA

INGENIERÍA EN INFORMÁTICA

PROYECTO INTEGRADOR PROFESIONALIZANTE

Informe Final

Desarrollo de un señuelo informático (honeypot) con capacidades extendidas de análisis de tráfico mediante técnicas de aprendizaje automático

Estudiante: Leandro Estrella

Entidad Receptora: Emergencias Salud

Docente Supervisor: Carlos Schenone

Tutor Organizacional: Martín Morales

FLORENCIO VARELA, 2025

Resumen

Este trabajo presenta una solución basada en técnicas de Machine Learning para detectar patrones de ataque en aplicaciones web en parte tanto para recopilación de información de un sitio para posteriormente comprometerse como así también para explotar vulnerabilidades en el mismo con el objetivo de utilizarlo de maneras no autorizadas y obtener un beneficio de ello siendo esto parte de la práctica habitual de las metodologías de hacking web por parte de cibercriminales. Utilizando la arquitectura de Random Forest basada en regresiones lineales y clasificaciones, se implementó y evaluó un modelo capaz de detectar tráfico malicioso, alcanzando una precisión cercana al 80%.

El estudio destaca la importancia de utilizar conjuntos de datos correctamente estructurados y categorizados para evitar malas interpretaciones dentro de la categorización del modelo. Durante la investigación se abordaron desafíos como la transformación de los datos, el ajuste de precisión y el aprendizaje de esta tecnología.

El modelo desarrollado representa un aporte en la lucha contra los ciberdelincuentes y sienta las bases para futuras mejoras, como la ampliación del dataset, la optimización del modelo, el desarrollo de nuevas capacidades y su implementación en diferentes organizaciones.

Palabras Clave: Machine Learning, Phishing, Árboles de decisión, Random Forest, Ciberseguridad, Explotación de vulnerabilidad, Reconocimiento de activos.

Abstract

Development of a Computerized Deception System (Honeypot) with Extended Traffic Analysis Capabilities Using Machine Learning Techniques

This paper presents a solution based on machine learning techniques to detect attack patterns in web applications, partly for gathering information from a site in order to subsequently compromise it, and also to exploit vulnerabilities in it with the aim of using it in unauthorized ways and obtaining a benefit from it, this being part of the usual practice of web hacking methodologies by cybercriminals. Using Random Forest architecture based on linear regressions and classifications, a model capable of detecting malicious traffic was implemented and evaluated, achieving an accuracy of close to 80%.

The study highlights the importance of using properly structured and categorized datasets to avoid misinterpretations within the model's categorization. During the research, challenges such as data transformation, accuracy adjustment, and learning this technology were addressed.

The model developed represents a contribution to the fight against cybercriminals and lays the foundation for future improvements, such as expanding the dataset, optimizing the model, developing new capabilities and implementing it in different organizations.

Keywords: Machine Learning, Phishing, Decision Trees, Random Forest, Cybersecurity, Vulnerability Exploitation, Asset Recognition.

Dedicatoria y agradecimientos

Quiero agradecer antes que nada a la Universidad Nacional Arturo Jauretche que me dió la posibilidad de estudiar esta bella y desafiante carrera. A todos los docentes que han forjado en mí la pasión por la informática y a todos aquellos amigos,compañeros y colegas que me han acompañado durante toda la carrera.

Continuando con los agradecimientos, quiero expresar mi profunda gratitud con el Dr. Carlos Schenone por su acompañamiento, excelente predisposición y sobre todo, por compartir la misma pasión respecto a la seguridad informática, esperando en el corto plazo contribuir junto a él proyectos que enriquezcan la experiencia de la comunidad y los estudiantes en esta área a través de la universidad. Además quiero reconocer y agradecer al coordinador de la carrera Dr. Ing. Martín Morales por permitirme la realización de este proyecto y su compromiso por mejorar continuamente la carrera de Ingeniería en Informática.

Por otra parte, quiero agradecer tanto a mi líder Carolina Carracedo, que ha sido un gran apoyo para alcanzar esta meta y que ,además de ser una gran líder y sobre todo una gran persona, como a mi compañero Luis Gonzalez, que es un excelente ser humano y trabajador, me han permitido enfocar en este proyecto para que pueda finalizarlo en los plazos estipulados.

Quisiera dedicar este logro a mis padres, a mi hermano y a mi pareja Alejandra,a mis amigos y todas las personas que me han apoyado durante todo el trayecto, siendo fundamentales para que persista ante las diferentes dificultades que se me presentaron y que hoy culminan con este título.

Índice de contenido

1. Introducción

El presente trabajo forma parte de la Práctica Profesional Supervisada (PPS) de la carrera Ingeniería en Informática de la Universidad Nacional Arturo Jauretche (UNAJ), siendo parte de los requisitos para la obtención del título.

En un contexto donde las amenazas cibernéticas son cada vez más sofisticadas [1] el objetivo general de esta investigación es desarrollar una solución honeypot genérica basada en docker, python y en modelos creados con Machine Learning a partir de datasets construidos en herramientas de monitoreo de red que fueron confeccionados detalladamente para permitir la detección de patrones de ataque web, contribuyendo a fortalecer las medidas de ciberseguridad en las etapas tempranas de detección de amenazas y prevención de incidentes [2], [3].

Las técnicas de inteligencia artificial han demostrado logros en diversos campos, entre ellos la ciberseguridad, donde con diversas técnicas de entrenamiento podemos detectar y predecir ataques utilizando una fuente de datos confiable y purgada [4].

Considerando lo anterior, las actividades desarrolladas en la práctica se enfocarán en investigar y desarrollar un clasificador basado en árboles de decisión binarios entrenados para identificar patrones de detección de paquetes mediante algoritmos de machine learning [5]. La investigación a desarrollar se enfocará en el entrenamiento y construcción de un modelo basado en un algoritmo de aprendizaje alimentado con un conjunto de datos conformado por datasets en formato csv con datos de paquetes de red correspondientes a diferentes servicios de red [6],[7]. Durante el entrenamiento y validación se utilizarán labels que marcan el tráfico como normal o malicioso, luego se buscará obtener una exactitud global con una cota establecida en 90% para posteriormente validar su desempeño utilizando una función de predicción en tiempo real que categoriza el tráfico utilizando el modelo generado [8],[9].

El tema de estudio seleccionado tiene gran relevancia en el contexto actual de la seguridad informática, donde la prevención de ataques cibernéticos es una necesidad primordial con el objetivo de detener o al menos frenar su incremento sostenido. En este grupo, destacan los ataques compatibles con servicios de red, representando el setenta y siete por ciento de los incidentes de seguridad reportados globalmente según un estudio de la empresa Verizon [10]. Siguiendo la tendencia, en Argentina los ataques que involucran servicios de red se ubican en todo el gráfico , como se puede observar en la imagen siguiente obtenida el

reporte anual para el año 2024 del equipo local de respuesta ante incidentes (ARCERT) [11].

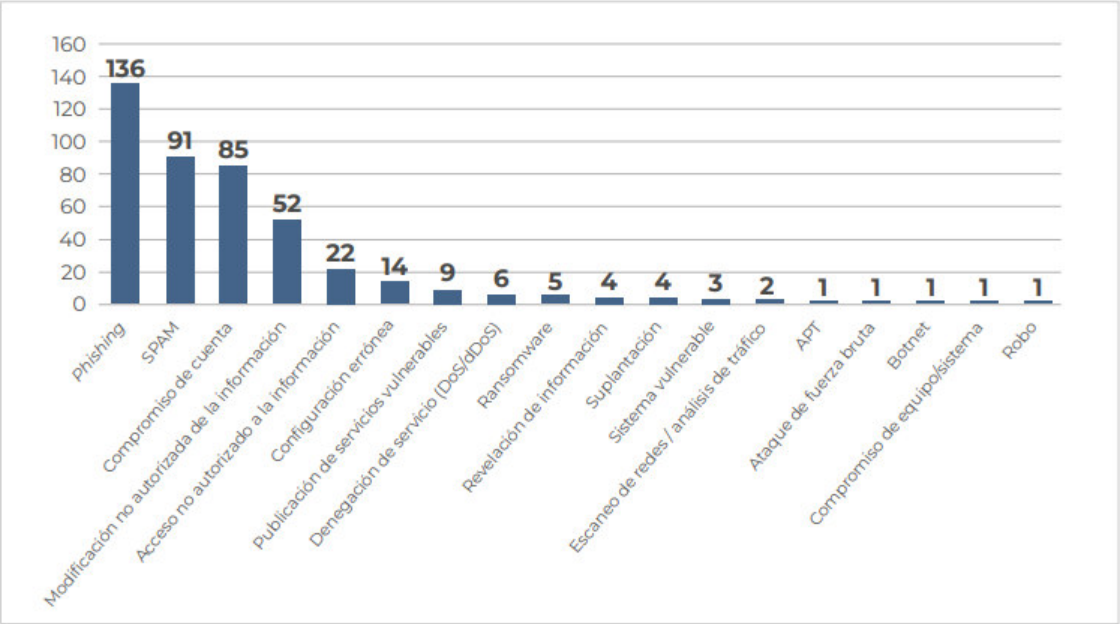

Figura 1. Informe Anual de Incidentes de Seguridad registrados en el 2024 por el equipo de respuesta ante Emergencias Informáticas Nacional, [en línea] Disponible en: *https://www.argentina.gob.ar/sites/default/files/2025/07/informe_cert-ar_2024.pd* (Accedido el 08 de octubre de 2025)

En este contexto, una herramienta automatizada para la detección y alerta de ataques de red fortalece los sistemas colaborando la protección de la infraestructura expuesta a internet como así también permite ser aplicable a la red interna de una organización,promoviendo un entorno digital más seguro mediante el monitoreo y toma de decisión más veloces [9],[12].

1.1 Motivación

Los ataques informáticos siguen creciendo, causando altos impactos en la sociedad [13]. A partir de la penetración de la tecnología en la sociedad de la información, Somersville plantea que la ciberseguridad dejó de ser un problema técnico para ser un problema sociotécnico [14]. A pesar de los continuos esfuerzos para fortalecer los sistemas informáticos, los ataques continúan

creciendo [10]. Con el objetivo de aportar una herramienta en el combate contra la inseguridad informática, cuyas consecuencias persiguen diversos objetivos, entre los cuales podemos mencionar el ataque a la disponibilidad para dañar la operación y/o reputación de la empresa para obtener un beneficio económico por medio de la extorsión, o lograr saltear los mecanismos de autenticación para conseguir información que luego será vendida en el mercado ilegal. La herramienta que proponemos busca simplificar la implementación de un entorno que emule un sitio expuesto a Internet, captando las conexiones del exterior y analizando los datos a través de un modelo de aprendizaje automático entrenado para identificar actividades maliciosas [9],[10], devolviendo una señal de alerta con el objetivo de mitigar el impacto de un potencial ataque informático [8].

En este contexto, surge la pregunta de investigación: **¿Puede crearse un método para fortalecer la seguridad en red a partir de técnicas de Machine Learning que nos brinde una detección proactiva y temprana ante los ataques de red que pueda sufrir nuestra infraestructura expuesta públicamente? [9],[12].**

Este proyecto busca responder a esta cuestión mediante la aplicación de algoritmos de aprendizaje de máquina con el propósito de catalogar el tráfico de red e identificar ataques en tiempo real que esté sufriendo nuestra infraestructura expuesta a internet que ha sido simulada por nosotros [3],[4].

Con el fin de realizar dicha herramienta, utilizamos machine learning, el cual ha mostrado un gran potencial en la creación de modelos entrenados que pueden resolver diferentes cuestiones según los datos que se le brinden [5],[6].

1.2 Objetivos

En concordancia con la motivación de este proyecto, el objetivo general consiste en construir una herramienta que permita automatizar el despliegue de un honeypot con capacidades extendidas de análisis de tráfico mediante técnicas de aprendizaje automático [1],[2],[9]. Para dicha finalidad, también tomamos en cuenta que la herramienta pueda ser replicable para cualquier sitio y cuya infraestructura también pueda ser desplegada con facilidad, velocidad y practicidad permitiendo que esté al alcance de cualquier interesado en utilizarla para su propio entorno, ya sea de manera profesional o para realizar pruebas de concepto controladas [15].

Objetivos Específicos

Con la finalidad de poder alcanzar el objetivo propuesto en esta práctica profesional supervisada, se han delimitado metas prioritarias que en conjunción permiten que se logre el objetivo propuesto, las cuales son las siguientes:

Construir una herramienta que permita automatizar el despliegue de un sitio destinado a cumplir la función de honeypot:

- Crear código que permita clonado estático de sitios y los salve en un directorio listos para ser montados

Incorporar en el honeypot herramientas que brinden la capacidad de capturar tráfico:

- Investigar herramientas que permitan interactuar con el tráfico de red del servidor
- Integrar la opción elegida con la herramienta que estamos desarrollando
- Verificar que los datos sean consistentes y fidedignos

Construir una herramienta que permita analizar los datos capturados basada en la detección de patrones compatibles con una amenaza informática utilizando técnicas de aprendizaje automático:

- Diseñar y entrenar un modelo de Random Forest con capacidad para detectar y categorizar tráfico de red.
- Evaluar diferentes técnicas de optimización para obtener el mejor rendimiento.

2. Marco Teórico

2.1. Inteligencia Artificial (IA)

Es una rama de la informática que se centra en la creación de sistemas capaces de realizar tareas que normalmente requieren inteligencia humana, tales como la interpretación de datos, la toma de decisiones y la resolución de problemas[16]. En términos generales, la IA busca dotar a las máquinas de la capacidad para aprender de su entorno, adaptarse a situaciones nuevas y actuar de manera autónoma, facilitando su aplicación en áreas tan diversas como la medicina, el transporte y la seguridad informática [17].

Los sistemas de IA se dividen en dos categorías principales: IA débil y IA fuerte[18]. La IA débil se limita a realizar tareas específicas, como el reconocimiento de voz o el análisis de patrones en imágenes, en los que un conjunto de algoritmos toma decisiones basadas en reglas predefinidas o datos históricos [19]. Por otro lado, la IA fuerte busca replicar en su totalidad las capacidades cognitivas humanas, a superar habilidades de razonamiento, planificación y comprensión contextual, aunque aún queda un largo camino por recorrer para alcanzar este nivel [20].

Desde su surgimiento, la IA ha evolucionado significativamente, pasando de sistemas que se basaban en reglas lógicas a métodos que involucran el aprendizaje automático (Machine Learning, ML) y el aprendizaje profundo (Deep Learning, DL). Esta transición ha marcado un cambio de paradigma en el que los sistemas ahora pueden aprender y mejorar su rendimiento a partir de datos, en lugar de depender únicamente de la programación específica de cada tarea [21], [22].

2.2. Aprendizaje automático (ML)

Dentro del ámbito de la Inteligencia Artificial, el Machine Learning (ML) o aprendizaje automático destaca como una subdisciplina clave, centrada en el desarrollo de sistemas que pueden aprender automáticamente a partir de datos sin necesidad de una programación específica para cada tarea [21] . En lugar de seguir instrucciones codificadas paso a paso, los sistemas de ML analizan grandes volúmenes de datos, identifican patrones y extraen tendencias que

luego les permiten realizar predicciones o tomar decisiones de manera autónoma [16],[22].

El aprendizaje automático cambia el paradigma de programación ya que se suministran datos de entrada junto con las respuestas que se esperan de ese conjunto y el sistema "aprende" a inferir el modelo o las reglas necesarias para obtener las respuestas deseadas [21]. Esta perspectiva permite que los sistemas construyan su propia contextualización y comprensión de las relaciones entre los datos , aplicando ese conocimiento adquirido a todo nuevo conjunto de datos mejorando su performance a través de la práctica, apoyado por técnicas de análisis estadístico y procesamiento de datos a gran escala, aprovechan la capacidad de cómputo moderno para trabajar con volúmenes de datos masivos que serían muy complejos de procesar con los métodos estadísticos tradicionales [19], [22].

Dentro de machine learning, existen tres tipos principales de algoritmos de aprendizaje automático, cada uno definido por su enfoque para entrenar los modelos de acuerdo a los datos disponibles y la problemática que desee resolverse:

Aprendizaje Supervisado: Este tipo de algoritmo se basa en entrenar un modelo con datos de entrada etiquetados, donde cada ejemplo está acompañado de una clasificación correcta o etiqueta. El sistema utiliza estos pares de datos para aprender a mapear entradas con salidas y posteriormente, realizar predicciones sobre datos nuevos no etiquetados [21], [22]. Los campos de aplicación incluyen sistemas de clasificación de imágenes, predicción de tendencias, regresión, tareas de clasificación y reconocimiento de patrones.

Aprendizaje No Supervisado: Este algoritmo utiliza datos no etiquetados y busca patrones característicos en estos sin contar con una etiqueta o respuesta correcta explícita. El objetivo del mismo es encontrar estructuras subyacentes, que le permitan realizar categorizaciones o agrupaciones de los datos. Este abordaje es útil en análisis exploratorios, detección de anomalías y creación de sistemas de sistemas de recomendación [22].

Aprendizaje por Refuerzo: El algoritmo se basa en la interacción con un entorno dinámico, donde el sistema aprende a tomar decisiones con el objetivo de maximizar una recompensa acumulada. En esta técnica, el modelo no recibe etiquetas o respuestas correctas específicas, sino que aprende iterando en un proceso de prueba y error, ajustando sus acciones de acuerdo a las

recompensas o penalizaciones recibidas. Se utiliza en aplicaciones que requieren la toma de decisiones en tiempo real, como los sistemas de control para robots [23].

El aprendizaje automático representa un avance fundamental en la capacidad de los sistemas para analizar y responder a datos complejos. Al aprender directamente de los datos y refinar su comportamiento a medida que recibe información adicional, los modelos se vuelven cada vez más efectivos y precisos, abriendo su aplicación a un grupo amplio de problemas complejos, desde el medio ambiente hasta la medicina [14], [22], aportando soluciones a desafíos que antes resultaban muy difíciles de abordar con la programación tradicional.

2.3. Random Forest

El algoritmo Random Forest se enmarca dentro de las técnicas de aprendizaje supervisado que emplea métodos de ensamblado para mejorar la capacidad predictiva y la robustez de los modelos, utilizando para este fin un conjunto de árboles de decisión independientes, los cuales fueron entrenados con muestras aleatorias del conjunto de datos original y posteriormente combina sus predicciones mediante votación (en clasificación) o promedio (en regresión) [5], [24].

El procedimiento general del RF se puede describir con la siguiente metodología:

1. Para cada árbol del bosque, se genera un muestreo con reemplazo del conjunto de entrenamiento [5].
2. Durante el crecimiento de cada árbol, en cada nodo de decisión se selecciona al azar un subconjunto de las variables de entrada (atributos) disponibles para determinar la partición óptima. Esta selección aleatoria tiene como fin reducir la correlación entre los árboles y promover la diversidad en el conjunto [5], [25].
3. Cada árbol se construye hasta su crecimiento completo (o hasta algún criterio de parada) sin necesariamente tener que realizar una poda (es decir remover aquellos árboles redundantes u que tienen una performance pobre) [26].
4. Si utilizamos el criterio de clasificación, la predicción del conjunto se obtiene por mayoría de votos entre los árboles; en cambio en regresión, por el promedio de sus salidas [5].

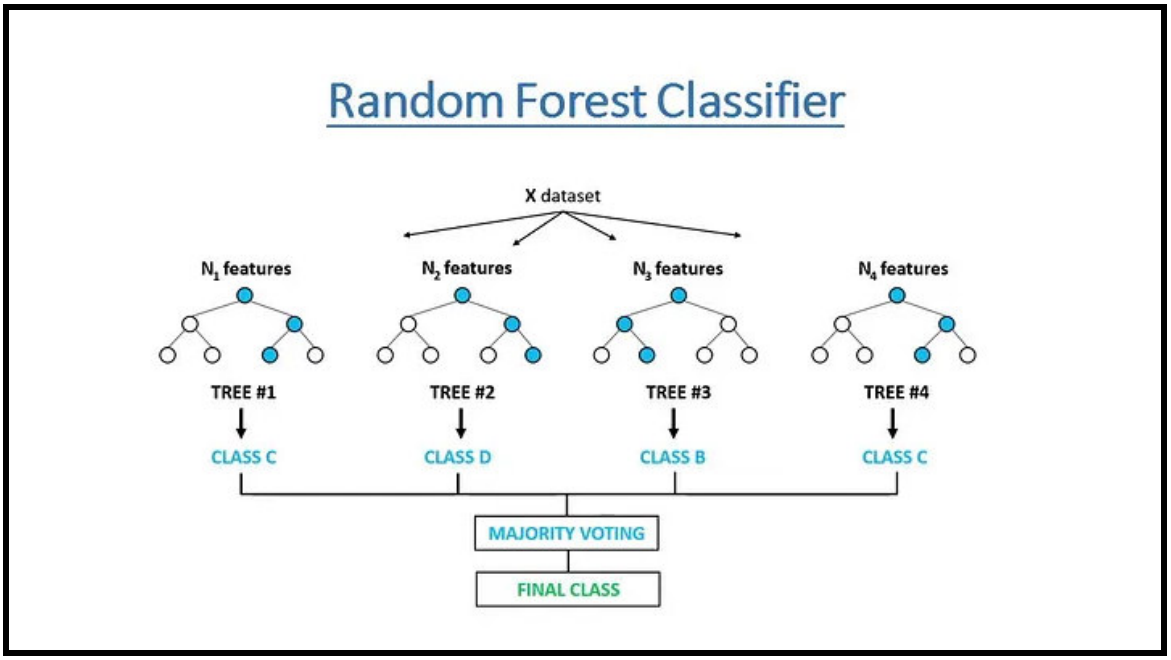

Figura 2. Esquema general de funcionamiento de Random Forest Classifier

2.4. Optimización de hiperparametros

La optimización de hiperparámetros es un paso crítico en el desarrollo de modelos de aprendizaje automático debido a que una correcta selección de estos valores puede mejorar notoriamente la capacidad de generalización del modelo y su desempeño [21], [27]. Dentro de la optimización de hiperparámetros, encontramos el enfoque denominado como “grid search” (búsqueda exhaustiva sobre una malla de valores posibles) [28].

El método Grid Search consiste en definir un espacio de parámetros discretizados para cada hiperparámetro relevante del estimador tales como penalización de una clase, el número de árboles en un bosque aleatorio, etc. A partir de esto, se generan todas las combinaciones posibles del producto cartesiano de dichos valores y se evalúan mediante validación cruzada [29],[30].

El procedimiento para el funcionamiento se puede describir de la siguiente manera:

1. Se selecciona un estimador base y se define un diccionario (o lista de diccionarios) que mapea nombres de hiperparámetros a listas de valores posibles.
2. Grid Search crea internamente todos los puntos de la malla (es decir todas las combinaciones de hiperparametros). Por cada punto obtenido,

el estimador se entrena y evalúa mediante validación cruzada definida [29].

3. Para cada combinación, se calcula una métrica de evaluación (por ejemplo accuracy, F1-score, ROC-AUC) media sobre los folds de la validación cruzada [29].
4. Una vez que se completan todas las pruebas, se selecciona la configuración que alabanza el mejor valor medio de la métrica y configurando este valor obtenido se re-entrena el conjunto de entrenamiento.
5. Luego del ajuste realizado sobre el modelo, este último puede evaluarse sobre un conjunto de prueba independiente para estimar su generalización real [21].

Matemáticamente, si se tienen “n” hiperparámetros y cada uno admite “mi” valores, entonces la cantidad total de combinaciones es

$$\prod_{i=1}^{n} m_i$$

lo cual implica que el coste computacional crece exponencialmente con el número de parámetros y el tamaño de los valores explorados. Es por ello que la técnica es exhaustiva y puede volverse inviable en espacios de búsqueda amplios [28],[30].

2.5.Analisis de trafico

En el entorno actual de redes híbridas y de gran escala, el análisis del tráfico de red representa una función esencial tanto para la gestión de la infraestructura como para la seguridad informática [31],[32]. El tráfico de red puede definirse como el conjunto de datos (paquetes, flujos, conexiones) que circulan entre los dispositivos de una red en un periodo de tiempo determinado. Su estudio permite caracterizar comportamientos normales, detectar anomalías, estimar la carga de la red y anticipar incidentes de seguridad.

Fundamentos del análisis de tráfico

El análisis de tráfico de red parte de los siguientes elementos clave:

- **Captura de datos:** consiste en obtener información de los paquetes (por ejemplo mediante tcpdump / pcap), o de los flujos (flow records, NetFlow, IPFIX). Estos registros pueden proporcionar metadatos como el número de bytes, número de paquetes, tiempos de arribo, puertos, protocolos, dirección origen/destino [33],[34].
- **Características estadísticas:** A partir de la captura se extraen métricas como volumen de bytes, duración de conexión, tasa de paquetes, y distribución de tamaño. Estas características permiten describir patrones de comportamiento de la red [35].
- **Clasificación y detección de patrones:** Una vez obtenida la información, el análisis puede contemplar métodos de clasificación de tráfico (por ejemplo distinguir aplicaciones, tipos de servicio) o detección de anomalías (por ejemplo tráfico malicioso, escaneo, DDoS) [36].
- **Visualización y correlación:** Finalmente, los datos de tráfico pueden visualizarse para entender tendencias (por ejemplo picos de tráfico), correlacionarse con eventos de seguridad o inteligencia de amenazas, y alimentar otros sistemas de monitoreo o respuesta.

Aplicaciones en seguridad y detección de intrusiones

El análisis del tráfico de red es particularmente relevante en contextos de ciberseguridad. Por ejemplo, es un componente fundamental en un sistema de detección de intrusiones basado en flujos, ya que permite:

- Identificar escaneos de puertos, conexiones inusuales o patrones de comunicación que se desvían del comportamiento normal [37].
- Detectar anomalías de tráfico que podrían corresponder a ataques de día-cero, movimientos laterales o filtraciones de datos [38].
- Brindar la base de datos que alimenta modelos de aprendizaje automático o de detección de anomalías [39].

Métodos y técnicas principales

Los principales enfoques del análisis de tráfico incluyen:

- **Clasificación de tráfico basado en métodos tradicionales:** Se han empleado heurísticas sobre puertos, inspección de payload, o comportamiento estadístico. Sin embargo, con el incremento del cifrado y

el cambio de patrones, estos métodos tradicionales presentan limitaciones evidentes [36],[40].

- **Flujo (flow) vs paquete (packet) vs meta datos**: La elección entre analizar cada paquete o los flujos agregados tiene implicaciones en el volumen de datos, complejidad de procesamiento y granularidad de la detección [33],[34].
- **Aplicación de machine learning y deep learning:** Más recientemente se han aplicado técnicas de ML (árboles, SVM, clustering) y deep learning (autoencoders, redes recurrentes) al análisis de tráfico de red, especialmente para detección de anomalías [39],[41].
- **Detección de anomalías:** Dentro del análisis del tráfico se incluye la detección de eventos no habituales. Las anomalías pueden clasificarse como puntos aislados, contextuales o colectivas. Los retos incluyen la heterogeneidad, escalabilidad de datos, latencia y precisión [43],[44].

2.6. Ciberseguridad

La ciberseguridad es el conjunto de procesos, tecnologías y medidas organizativas destinadas a proteger las redes, los sistemas y los datos de accesos no autorizados, alteraciones o destrucción [45]. Sus fundamentos se basan en los pilares de confidencialidad, integridad y disponibilidad junto con otros factores emergentes como la autenticación, no repudio, resiliencia y continuidad operativa [46]. Con el transcurso del tiempo, la digitalización se ha expandido conectando todos los aspectos de la vida cotidiana y la empresarial donde la ciberseguridad ha pasado de ser solamente una protección de activos digitales a ser un requisito fundamental en la sociedad moderna, abarcando múltiples capas de protección distribuidas a través de dispositivos y redes que incluyen desde el software hasta las estrategias organizativas de gestión de riesgos conformando un problema socio-técnico [14],[47].

2.7. Honeypot

La defensa frente a amenazas cibernéticas está evolucionando más allá de los enfoques tradicionales basados únicamente en firmas o reglas estáticas [48]. En este contexto surge la estrategia de la trampa deliberada o “honeypot”, definida como un sistema o recurso informático que está diseñado para parecer un activo legítimo vulnerable, con el fin de atraer al atacante, monitorizar su comportamiento, recopilar inteligencia y, en el mejor de los casos, aislarlo del

entorno de producción [1],[2]. Esta técnica aporta tanto detección temprana como análisis forense de la conducta adversaria, permitiendo complementar los mecanismos de seguridad tradicionales.

Definición, objetivos y clasificaciones

Un honeypot actúa como cebo: se presenta como un recurso aparentemente valioso para un adversario, y su interacción provee datos ricos sobre tácticas, técnicas y procedimientos (TTP) del atacante [2],[49]. Los objetivos principales son:

- Atraer actividad maliciosa para aliviar la presión sobre sistemas críticos.
- Recoger datos de ataque, malware, vectores de explotación y comportamiento adversario [2].
- Mejorar la postura de seguridad mediante análisis, inteligencia y eventual integración con respuesta a incidentes [50].

Según el nivel de interacción y complejidad, los honeypots pueden clasificarse típicamente en las categorías de baja interacción (limitados en su emulación de servicios y muy controlados), alta interacción (que permiten al atacante interactuar casi como si fuera un sistema real, obteniendo mayor visibilidad pero con mayor riesgo) y interacción media o híbrida (que combinan características de ambas) [1],[49]. Esta taxonomía resulta fundamental en el diseño de la estrategia de honeypot, pues afecta la riqueza de datos, el coste operativo y el riesgo inherente.

Fundamentos operativos y arquitectura

El despliegue de un honeypot implica diversas decisiones arquitectónicas: ubicación en la red (DMZ, red interna, nube), aislamiento respecto del entorno de producción, capacidad de monitorización, recolección de logs, generación de alertas y análisis posteriores [51]. De hecho, una arquitectura típica incluye: un recurso señuelo, sensores que monitorizan la interacción, un almacén de datos de eventos y un sistema de análisis (por ejemplo, correlación con SIEM)[49]. Por ejemplo, estudios han señalado que la arquitectura decoy + captor permite clasificar honeypots independientes o cooperativos según el grado de integración con otros sistemas [51].

El nivel de interacción influye directamente en la fidelidad de la emulación (y por tanto en la capacidad de engaño al atacante), pero también en el riesgo de que el honeypot sea detectado o utilizado como plataforma de ataque hacia otros

sistemas [1],[2]. Por tanto, la correcta configuración, segmentación y monitoreo son requisitos críticos [50}.

3. Estado del Arte

3.1. Honeypots contemporáneos y su evolución

Los honeypots han sido desde sus inicios una herramienta de engaño y recopilación de inteligencia frente a actividad maliciosa; trabajan como señuelos computacionales que simulan recursos vulnerables para atraer atacantes, registrar su comportamiento y obtener datos de valor para la defensa [1],[2]. A lo largo del tiempo, estos sistemas han evolucionado considerablemente en cuanto a su arquitectura, nivel de interacción y despliegue. Por ejemplo, las primeras implementaciones se enfocan en emular servicios básicos con bajo nivel de interacción, mientras que en la actualidad existen honeypots de alta interacción, distribuidos en la nube, modularizados y diseñados para integrarse con arquitecturas de contenedores y microservicios [50],[51].

Además de esta evolución en la interacción, también se observa un cambio hacia entornos específicos como IoT, IIoT y sistemas ciber-físicos, donde los honeypots tienen que adaptarse a dispositivos con limitaciones de recursos, protocolos especializados y dinámicas adversas particulares. Esta transformación implica nuevos retos: la detección del honeypot por parte del atacante, la evasión de la emulación y la necesidad de mantener seguras las trampas sin que se conviertan en plataformas de ataque [49]. A través de estas mejoras, los honeypots modernos no solo actúan como centinelas pasivos, sino también como elementos activos de threat-intelligence.

3.2. Implementaciones open source vs comerciales

En cuanto a su implementación, existe una distinción clara entre soluciones open source y comerciales o propietarias. Las herramientas de código abierto permiten una mayor transparencia, adaptabilidad y revisión por la comunidad, lo cual favorece su adopción en entornos de investigación y prototipado [51]. Una encuesta reciente sobre honeypots open source analiza arquitecturas, despliegues en la nube, frameworks y metodologías de detección y evasión. Por su parte, las soluciones comerciales ofrecen soporte, integración con suites de seguridad corporativa, actualizaciones continuas y, a menudo, interfaces más amigables, aunque pueden presentar menor flexibilidad para modificar internamente la lógica del honeypot [50].

La comparación entre estos dos enfoques debe contemplar criterios como nivel de interacción, datos capturados, facilidad de despliegue, capacidad de integración y coste operativo [51]. En la práctica, organizaciones híbridas aprovechan modelos mixtos: honeypots open source para experimentación y

monitoreo interno, y soluciones comerciales para entornos de producción con requisitos de soporte corporativo.

3.3. Aplicaciones de machine learning en detección de intrusiones

En la última década, el uso de técnicas de aprendizaje automático (ML) y aprendizaje profundo (DL) en sistemas de detección de intrusiones ha crecido de forma considerable [39],[41]. En este contexto, los honeypots pueden aportar un volumen significativo de datos maliciosos reales ,que luego alimentan modelos ML, mientras que los sistemas de detección basados en ML permiten superar las limitaciones de los enfoques tradicionales (como firmas o reglas estáticas) al identificar patrones de ataques desconocidos o comportamientos anómalos [37].

Las técnicas empleadas varían entre aprendizaje supervisado, no supervisado y semi-supervisado, lo que facilita la detección de intrusiones en entornos dinámicos [21],[22]. A pesar de los avances, persisten retos como el desequilibrio de clases, la escasez de datos etiquetados y la presencia de ataques adversarios diseñados para engañar a los modelos [44].

3.4. Limitaciones y brechas identificadas

Aunque el estado del arte muestra múltiples avances, también emergen limitaciones y brechas de investigación relevantes. En el ámbito de honeypots, se identifican riesgos como la vulnerabilidad del honeypot a convertirse en plataforma de ataque, la dificultad de lograr una simulación realista completa, y la detección por parte de adversarios lo que reduce su efectividad [49],[42]. Asimismo, la mayoría de los estudios señalan que la integración de honeypots con análisis de datos y ML aún está en fase emergente, con poca estandarización en procesos y escasa cobertura de escenarios industriales [51].

Por otro lado, los sistemas ML en detección de intrusiones enfrentan problemas como la falta de datasets representativos, la generalización en entornos reales muy heterogéneos y la resistencia frente a ataques adversarios (adversarial ML) [43],[44]. Finalmente, cuando estos componentes se integran dentro de arquitecturas híbridas (on-premises + nube) y de contenedores, la visibilidad del tráfico, la gobernanza de datos y la automatización de respuesta constituyen brechas que deben abordarse [51]. Estas limitaciones explicitan el área de contribución de esta tesis: proponer una solución híbrida de honeypot + ML + integración en entorno seguro y automatizado, considerando las lecciones del estado del arte.

4. Metodología y diseño del proyecto

4.1. Metodología del desarrollo

Para la realización de esta prueba de concepto, se ha tenido en cuenta diferentes aspectos en torno a los costos, recursos y tiempo de la herramienta, tomando como referencia que el software debe poder correrse de manera sencilla en cualquier servidor, el mismo debe poder reutilizarse contra diferentes aplicaciones web y obtener el mismo comportamiento sin importar el stack tecnológico de dicha página web y a su vez, que el modelo de entrenamiento puede ser aplicado correctamente independientemente de que aplicación web se encuentre corriendo en la herramienta [52],[53].

Basándonos en este criterio, se ha concluido en un stack tecnológico que permite la replicación de la herramienta en forma parcialmente automatizada, permitiendo llevar la prueba de concepto a cualquier servidor para su posterior implementación [54],[55].

4.2. Arquitectura general del sistema

En el siguiente diagrama se representan los principales componentes que conforman el sistema desarrollado.

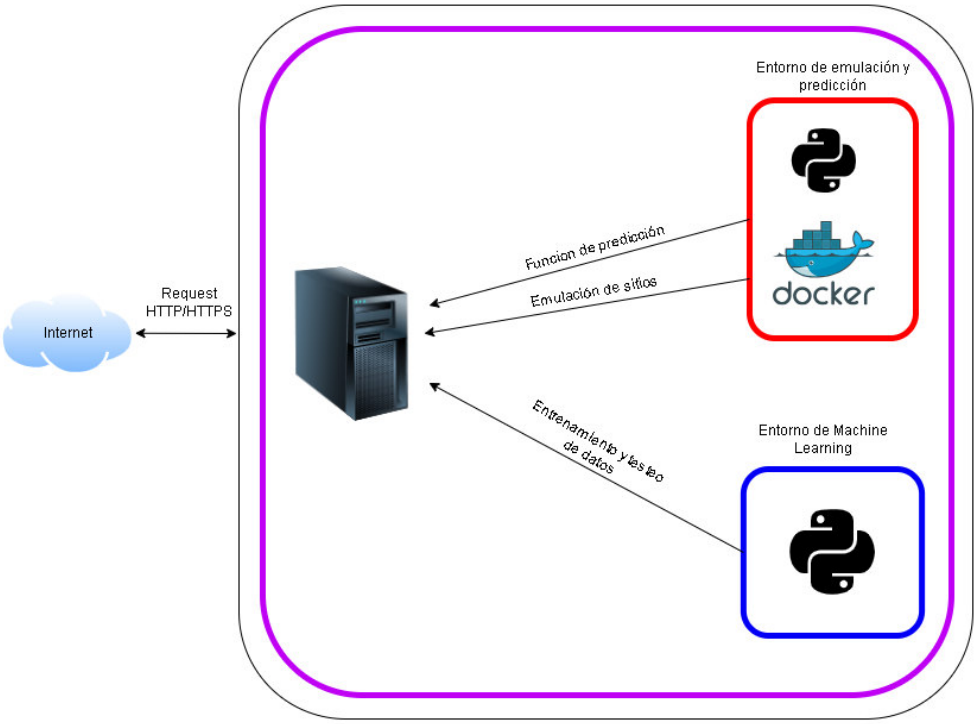

Figura 3. Arquitectura general del sistema

El elemento central es el **servidor**, que constituye el recurso de cómputo encargado de ejecutar las distintas funciones del proyecto, incluyendo la emulación del honeypot, la captura de tráfico y el procesamiento mediante técnicas de aprendizaje automático[56]. Este servidor puede desplegarse tanto en un entorno **on-premise** como en la **nube**, aunque se recomienda su implementación dentro de la **red DMZ**, a fin de reducir la exposición directa del entorno corporativo y facilitar la segmentación del tráfico [57],[58].

Hardware requerido

Dado que el modelo propuesto contempla distintos ambientes de ejecución, los requerimientos de hardware pueden variar según el enfoque adoptado. Se consideran dos criterios posibles para la implementación.

En el primer enfoque, denominado **All-in-One**, todas las funciones del sistema (emulación, captura, entrenamiento y predicción) se ejecutan dentro del mismo servidor. Para este caso, se recomienda una configuración mínima que contemple un procesador de ocho núcleos (8 CPU cores), 16 GB de memoria RAM, una GPU Nvidia 1050 Ti o superior, 250 GB de almacenamiento y dos

interfaces de red físicas [59]. Esta configuración permite garantizar un rendimiento adecuado tanto en las fases de entrenamiento del modelo como en la inferencia en tiempo real [60].

El segundo criterio, denominado **emulación y predicción**, separa el proceso de entrenamiento del modelo, trasladando esa tarea a un equipo de cómputo externo y dedicando el servidor únicamente a las fases de emulación y predicción. En este caso, los requerimientos se mantienen similares (procesador de ocho núcleos, 16 GB de memoria RAM, 250 GB de almacenamiento y dos interfaces de red) aunque sin la necesidad de contar con una GPU dedicada, dado que la carga computacional del entrenamiento se ejecuta en un entorno diferente [61].

Sistema operativo

El proyecto puede implementarse sobre distintos sistemas operativos, ya que **Docker** es compatible tanto con Windows como con Linux. Sin embargo, se recomienda utilizar una distribución basada en Linux por su estabilidad, rendimiento y facilidad de administración en entornos de servidores [62]. En el desarrollo del presente proyecto se empleó **Ubuntu Server 24.04** como sistema operativo base.

Consideraciones de seguridad

Dado que el sistema se encuentra expuesto a Internet, la **seguridad del servidor** constituye un aspecto fundamental para prevenir accesos no autorizados y mantener la integridad de los datos [63].
Se establecieron las siguientes medidas:

- **Interfaces de red:** se configuraron dos interfaces diferenciadas: una interfaz externa, asociada a la dirección IP pública y destinada a la escucha del software de captura de tráfico, y una interfaz interna utilizada exclusivamente para tareas de administración desde una red dedicada [57].

- **Firewall:** se configuraron reglas restrictivas que permiten únicamente el tráfico esencial en los puertos 80 (HTTP) y 443 (HTTPS), bloqueando cualquier otro acceso entrante. Una vez completada la instalación de los servicios, también se restringe el tráfico saliente, permitiendo únicamente las comunicaciones necesarias. Estas políticas pueden implementarse

mediante herramientas como *iptables*, *ufw* o *firewalld* [64],[65].

- **Docker y principio de privilegios mínimos:** los contenedores se ejecutan con los privilegios mínimos posibles, siguiendo el principio de *Zero Trust*. Para ello, se utiliza un usuario de servicio sin permisos de inicio de sesión interactiva, pero con capacidad para instanciar contenedores [52].

- **Hardening del sistema:** se implementaron mecanismos de control de acceso obligatorio mediante **SELinux** o **AppArmor**, además de la desactivación de servicios no esenciales para minimizar la superficie de ataque y reducir el riesgo de escalamiento de privilegios [67].

- **Autenticación segura:** el acceso al sistema se configuró mediante autenticación basada en claves asimétricas, eliminando la posibilidad de conexión por contraseña y fortaleciendo la seguridad del inicio de sesión [63].

Entornos virtuales y dependencias

Para la gestión del entorno de desarrollo y las dependencias, se emplearon **entornos virtuales de Python**, lo que permite aislar las librerías utilizadas y garantizar la reproducibilidad del proyecto [68]. Dependiendo del enfoque de infraestructura adoptado (All-in-One o Solo predicción), pueden configurarse uno o más entornos virtuales dedicados a las tareas específicas de entrenamiento e inferencia. El servidor debe contar con **Python 3.9 o superior**, junto con los paquetes y librerías requeridos para la ejecución de los scripts del proyecto [68].

Red y administración

El servidor debe ubicarse en un **segmento de red aislado**, separado de la red corporativa, con el objetivo de minimizar el riesgo de compromiso de la organización en caso de que un atacante logre tomar control del sistema [57]. Además, se recomienda la utilización de un **host bastión** como punto de acceso seguro, a través del cual se realicen las conexiones de administración por medio de **SSH** u otro protocolo cifrado [69]. Este esquema permite mantener una capa adicional de protección entre el entorno de producción y el servidor honeypot.

4.3. Diseño del honeypot

Para este proyecto, se ha decidido utilizar diversa variedad de herramientas que juntas contemplan la replicación de los sitios que el usuario decide clonar como así también todas las configuraciones respecto al monitoreo de la aplicación y la función de inferencia aplicada en la función de predicción [57].

Desglosando este diseño elegido podemos detallar con precisión los siguientes servicios que corre el honeypot y cual es su finalidad dentro de la herramienta.

Zeek

La herramienta zeek se encarga de realizar el monitoreo de todo el tráfico que está recibiendo el servidor, dejando registro de todos los logs de red en una carpeta del contenedor la cuál comparte con el equipo [70] . Esta herramienta crea archivos que contienen los logs de red capturados en el monitoreo realizado por la herramienta, segmentado por reglas que se encuentran predefinidas dentro del software y que permiten visualizar de una manera rápida y efectiva el tráfico en tiempo real en él. Dentro de estos archivos, utilizaremos puntualmente el log de conexiones para mapear los resultados del mismo con la función de predicción que utiliza el modelo entrenado previamente por nosotros, indicando mediante alertas en pantallas las direcciones ip de origen que han sido categorizadas como maliciosas según el modelo de entrenamiento [37]. El resto de archivos no han sido utilizados en esta versión del proyecto pero pueden ser utilizados para futuras versiones que incluye visado de archivos, comandos, etc.

Nginx

Nginx es un software de código abierto de alto rendimiento que funciona como servidor web, proxy reverso, balanceador de cargas y caché HTTP [71]. En nuestro caso, este mismo es utilizado para servir las aplicaciones web que hemos clonado de los diferentes sitios que se encuentran de manera pública dentro del dominio y subdominios de la organización, representando aquel sitio que busca ser simulado y dar la apariencia de sitio legítimo en un entorno diferente al de producción, buscando que los ciberdelincuentes realicen ataques hacia estos y poder obtener datos valiosos sobre los mismos [49].

Stack ELK

El stack elk consiste en un conjunto de software que en conjunción permiten la transformación, ingesta y visualización de datos en un aplicativo web [72].

A continuación detallaremos su función dentro del diseño de este proyecto.

Elasticsearch

Elasticsearch es un motor de búsqueda que permite centralizar un gran volumen de datos para su indexado y posterior análisis en tiempo real [73]. En esta ocasión, utilizamos elasticsearch para almacenar el contenido de los logs monitoreados por zeek y almacenarlos en este motor de búsqueda para su posterior análisis y correlación, diseñado para que un analista pueda ver que está sucediendo en dicho servidor y tomar decisiones en caso de que así se requiera.

Logstash

Logstash es un herramienta de código abierto que recopila datos de diversas fuentes, los transforma y luego los envía a un destino seleccionado. En nuestra herramienta, este software se utiliza para transformar los logs de zeek a un formato tipo json que luego será enviado a elasticsearch para su posterior visualización, debido a que los logs en crudo no pueden ser procesados correctamente por el motor de datos.

Kibana

Kibana es una herramienta de visualización y exploración de datos que trabaja en conjunto con elasticsearch [75]. Para nuestro proyecto, este software de código abierto nos permite visualizar en un navegador web todos los logs que se han parseado e ingestado en elasticsearch, permitiendo realizar búsquedas y filtrado de eventos para mostrar aquellos datos de interés. Adicionalmente, con este software pueden realizarse tableros de gráficos que nos permiten visualizar en resumen el estado de situación sobre un conjunto de datos relacionados.

Predicción

Esta funcionalidad del proyecto es fundamental, ya que la misma se encarga de catalogar los eventos que están pasando en tiempo real como una amenaza o a su vez, determinar qué es tráfico legítimo [39]. Para lograr dicha finalidad, la función (que ha sido desarrollada en el lenguaje Python) toma los datos que están escribiéndose en el archivo de logs de conexión, realiza las conversión a tipo de dato de dataframe para luego aplicar la función de predicción probabilística y determinar un puntaje de probabilidad que se compara contra el umbral que se define para considerar que nos encontramos ante un datagrama que indique un ataque por parte de un ciberdelincuente, notificando a través de la consola que se ha hecho esta detección [21],[22].

Clonado o Crawler

Con esta función escrita en python, lo que se realiza es un barrido en profundidad de la página web que definamos clonar para luego ser almacenada en el servidor. Para dicha tarea primero se establece un directorio de destino, se procede a recorrer el sitio y a salvar los objetos correspondientes al sitio que luego serán utilizados durante la fase de emulación [75].

4.4. Modelo de aprendizaje automático

El componente de **aprendizaje automático** constituye el núcleo analítico del sistema propuesto, ya que permite transformar los datos de tráfico capturados por el honeypot en información útil para la detección temprana de amenazas [21], [39]. Este modelo se concibe como una herramienta predictiva capaz de clasificar el tráfico en dos categorías principales: **benigno** o **malicioso**, basándose en características previamente extraídas de las conexiones de red. El diseño, entrenamiento y validación del modelo se realizaron siguiendo principios de ingeniería de datos, considerando tanto la calidad del conjunto de entrenamiento como la robustez del algoritmo seleccionado [22], [27].

Selección del dataset

El primer paso en la construcción del modelo consistió en seleccionar un **conjunto de datos (dataset)** representativo y adecuado para el problema de clasificación. En la literatura especializada, diversos datasets se utilizan en investigaciones de detección de intrusiones, como *KDD Cup 99*, *NSL-KDD*, *CICIDS2017* o *UNSW-NB15* [43],[44]. Sin embargo, los conjuntos más antiguos (como *KDD Cup 99*) presentan limitaciones significativas, entre ellas un fuerte desbalance de clases, redundancia de atributos y la presencia de patrones obsoletos que no reflejan los ataques contemporáneos [43].

Por este motivo, el dataset seleccionado para este proyecto fue el **UNSW-NB15**, desarrollado por la Australian Centre for Cyber Security (ACCS) en la University of New South Wales [39]. Este conjunto de datos representa un entorno híbrido con tráfico normal y malicioso capturado en 2015 mediante el emulador IXIA PerfectStorm, e incluye nueve tipos de ataques diferentes, como *Fuzzers, Reconnaissance, Exploits, DoS, Generic, Shellcode, Worms, Backdoors* y *Analysis*. Su principal ventaja radica en que combina tráfico moderno y variado, manteniendo una estructura de características adecuada para la aplicación de técnicas de aprendizaje supervisado [39], [4].

El dataset está compuesto por **45 atributos** [96] (ver anexo Lista de atributos) y más de 2 millones de registros, de los cuales una proporción significativa corresponde a tráfico legítimo. Esto lo convierte en un punto de partida ideal para entrenar modelos de clasificación binaria (ataque / no ataque) y posteriormente validar su desempeño frente a datos reales capturados por el honeypot implementado [39]. Adicionalmente, este dataset cuenta con documentación pública y ha sido ampliamente utilizado en trabajos académicos recientes, lo cual permite comparar los resultados de este proyecto con estudios previos bajo condiciones similares [4],[76].

Durante la etapa experimental, el conjunto original fue sometido a un proceso de **filtrado y reducción**. Se eliminaron atributos redundantes o de baja varianza, así como campos que podían introducir sesgo (por ejemplo, direcciones IP o identificadores específicos de sesión). De igual modo, se equilibró parcialmente la distribución de clases para mitigar el efecto del desbalance, ya sea mediante submuestreo de la clase mayoritaria o mediante técnicas de ponderación de clases en el modelo (*class_weight='balanced_subsample'*).

Selección y construcción de características (features)

La fase de **feature engineering** constituye uno de los pasos más críticos en el desarrollo del modelo, ya que determina en gran medida su capacidad de generalización [21],[22]. El análisis del tráfico capturado, tanto del dataset como del honeypot, permitió identificar un conjunto de **características cuantitativas y cualitativas** relevantes para la detección de actividad anómala.

Entre las **características numéricas** más importantes se destacan:

- **Duración de la conexión (dur):** tiempo total de la sesión TCP o UDP; útil para detectar comportamientos anómalos como sesiones persistentes o de corta duración repetitiva.

- **Bytes enviados y recibidos (sbytes, dbytes):** reflejan el volumen de datos transferidos en cada dirección, indicador frecuente de exfiltración o ataques volumétricos.

- **Paquetes enviados y recibidos (spkts, dpkts):** miden la intensidad del tráfico; valores anómalos suelen asociarse a ataques DoS o escaneo masivo.

- **Tiempos entre paquetes (inter-arrival time):** en muchos ataques automatizados los intervalos tienden a ser regulares o extremadamente cortos.

A nivel **categórico**, se incluyen atributos como:

- **Protocolo (proto):** identifica el tipo de protocolo de transporte utilizado (TCP, UDP, ICMP, etc.).

- **Estado de la conexión (state):** permite reconocer si la sesión fue establecida, rechazada o finalizada abruptamente.

- **Servicio (service):** indica la aplicación o puerto destino (HTTP, FTP, SSH, DNS, entre otros).

Estas variables fueron normalizadas y transformadas según su tipo. Las variables numéricas se estandarizaron mediante escalamiento (*StandardScaler*), y las categóricas se codificaron utilizando *One-Hot Encoding* para su correcta interpretación por parte del modelo. Este proceso se automatizó mediante un *pipeline* en *scikit-learn*, lo que permite repetir las etapas de preprocesamiento en nuevos conjuntos de datos sin reescribir código manualmente [27],[77].

Adicionalmente, se evaluó la importancia relativa de cada característica a través de la métrica de *feature importance* propia del modelo **Random Forest**, lo que permitió confirmar que variables como la duración de la conexión, la cantidad de bytes y los protocolos fueron las más determinantes en la predicción final [5],[6].

Selección del modelo y técnicas de entrenamiento

En la etapa de modelado, se optó por un enfoque de **aprendizaje supervisado** dado que el conjunto de datos cuenta con etiquetas conocidas que indican la naturaleza del tráfico. Entre los algoritmos disponibles, se seleccionó el **Random Forest Classifier**, un método de ensamble basado en árboles de decisión propuesto por Breiman [5]. Este modelo fue elegido por varias razones: su capacidad para manejar datos de alta dimensionalidad, su tolerancia al ruido, su robustez frente al sobreajuste y su interpretación a través de la importancia de variables [6],[24].

El entrenamiento del modelo se realizó mediante un proceso sistemático que incluyó **validación cruzada y búsqueda de hiperparámetros** (*GridSearchCV*) [78],[79]. Este procedimiento permitió explorar combinaciones de parámetros tales como el número de árboles (*n_estimators*), la profundidad máxima de los árboles (*max_depth*), la cantidad de características consideradas en cada división (*max_features*) y el criterio de partición (*gini* o *entropy*).

La búsqueda se efectuó sobre una malla de valores discretos y se evaluó mediante validación cruzada, utilizando como métrica principal el *F1-score*, que equilibra la precisión y el recall. Esta métrica se consideró más adecuada que la simple *accuracy*, dado que el conjunto de datos presenta cierto grado de desbalance entre clases. El uso de *GridSearchCV* permitió determinar la configuración óptima de hiperparámetros sin necesidad de intervención manual, garantizando mayor objetividad en la selección del modelo final.

El modelo resultante fue entrenado sobre un subconjunto de entrenamiento (80 %) y evaluado en un conjunto de prueba independiente (20 %). Los resultados obtenidos mostraron un rendimiento promedio superior al 89 % de *accuracy* y un *F1-score* cercano al 0.91, lo que demuestra su capacidad para distinguir tráfico benigno de malicioso [76], [4]. Sin embargo, el foco principal no se limitó a maximizar el rendimiento estadístico, sino también a asegurar la estabilidad y reproducibilidad del modelo frente a datos nuevos, incluyendo aquellos capturados por el honeypot en producción.

Evaluación y validación del modelo

La validación del modelo se realizó a partir de la **matriz de confusión**, herramienta que permite analizar los errores de clasificación en términos de verdaderos positivos (TP), falsos positivos (FP), verdaderos negativos (TN) y falsos negativos (FN). En el contexto de la detección de intrusiones, los **falsos negativos** representan el caso más crítico, ya que implican que una amenaza no fue detectada [37], [44]. Por ello, se buscó minimizar esta métrica incluso a costa de un ligero incremento en los falsos positivos.

Se complementa el análisis con las métricas **Precision** (es una métrica que mide la proporción de precisiones correctas del total de predicciones realizadas), **Recall** (es una métrica que mide la proporción de predicciones positivas que el modelo logró identificar correctamente), **ROC-AUC** (es una métrica que mide la capacidad del modelo para distinguir clases) y **Curva PR** (es un indicador que muestra la relación entre la precision y el recall en diferentes umbrales de

clasificación), para evaluar la capacidad del modelo bajo distintos umbrales de decisión [37],[43],[44] . Se realizaron además pruebas de robustez aplicando subconjuntos de datos obtenidos directamente del entorno de honeypot, confirmando que el modelo mantenía un comportamiento coherente frente a tráfico real y variado [37].

Implementación e integración en el sistema

Una vez validado, el modelo fue serializado en formato *pickle* y desplegado dentro del entorno de producción, integrado en la arquitectura general del sistema. Este componente se encarga de recibir flujos de datos procesados (por ejemplo, desde Zeek o CICFlowMeter), aplicar el pre-procesamiento correspondiente y generar una predicción en tiempo real sobre la naturaleza del tráfico [79], [80].

5. Implementacion

En esta sección abordaremos todas las operaciones realizadas para la implementación de la herramienta.

5.1. Preparación del entorno

Como primer paso, hemos creado el hardware virtualizado para el equipo utilizando el enfoque **ALL-IN-ONE**, estableciendo un disco de 200 GB, con placa de red, 8 core de CPU, 16 gigabytes de ram y una placa de video dedicada Nvidia modelo 1050 TI [60],[81]. Una vez que se creó el mismo, procedimos a utilizar como iso de instalación de sistema operativo la distribución **Ubuntu Server 24.10**, configurando la instalación con encriptación de disco, creando un usuario con una contraseña robusta, eligiendo correctamente la ubicación geográfica donde está implementado el servidor y una vez finalizado esto, procedemos a instalar el sistema operativo [62].

Una vez que el sistema operativo ha reiniciado, se procede a actualizar los paquetes del mismo utilizando el comando **apt update && apt upgrade -y** como usuario administrador o root dentro de la terminal, siendo utilizados estos comandos para actualizar la información de los paquetes que posee el servidor y la segunda parte para efectivamente realizar la actualización o instalación de los paquetes que se encuentran en el sistema.

Luego de realizada esta acción, se ha procedido a copiar una llave pública creada previamente para que esta misma forme parte de las conexiones de confianza del equipo a través de la metodología de conexión por infraestructura de clave pública, utilizando certificados para este fin [82]. Se procedió, luego de copiadas estos certificados, a modificar la configuración del servicio SSH para permitir conexiones solo por llave pública/privada, evitando así intentos de fuerza bruta de parte de intrusos que quieran tomar control de nuestro sistema, como así también cambiando el puerto del servicio SSH [63],.

Se procede a instalar el servicio de Docker dentro del servidor con el siguiente script:

```
!#/bin/bash

#Sincronizo paquetes
```

```
sudo apt-get update

#Instalo paquetes para certificados y request http

sudo apt-get install ca-certificates curl

sudo install -m 0755 -d /etc/apt/keyrings

#Descargo llave gpg de docker
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg
-o /etc/apt/keyrings/docker.asc


sudo chmod a+r /etc/apt/keyrings/docker.asc


# Add the repository to Apt sources:

echo \

 "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \

 $(. /etc/os-release && echo
"${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \

 sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

#Instalo paquetes necesarios para docker

sudo apt-get install docker-ce docker-ce-cli \
containerd.io docker-buildx-plugin \
docker-compose-plugin -y
```

Figura 4. Visualización de entorno configurado correctamente.

Con el software instalado procedemos a instalar **python3-venv**, el cuál es un paquete de python que nos permite crear ambientes y entornos que son seguros

para instalar paquetes sin causar conflictos con los paquetes que se encuentran actualmente dentro del servidor, a través del comando **apt install python3-venv**.

Como primer paso, procederemos a elegir un directorio donde queremos montar la aplicación. Una vez allí, procederemos a crear un ambiente de python con el comando **python3 -m venv ./honey** y luego activarlo con el comando **source honey/bin/activate** donde se mostrará el nombre del entorno delante del nombre del usuario y hostname en la terminal, indicando que estamos utilizando el entorno que hemos creado [83],[77].

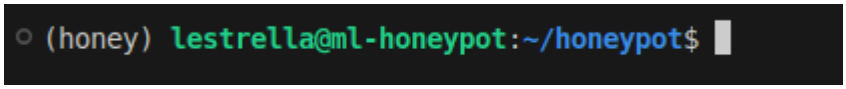

Figura 5. Visualización de entorno configurado correctamente.

Después de haber realizado esta acción, procedemos a instalar los paquetes **requests,bs4,pandas,sklearn,scikit-learn,matplotlib** escritos en python a través del administrador de paquetes **pip** [77].

5.2. Despliegue del honeypot

Luego de la preparación del entorno, durante esta etapa se enfocará en el despliegue de la herramienta en sí, mostrando todos los pasos realizados para el correcto funcionamiento del proyecto.

Como primer paso, se procede a establecer el sitio a clonar y en qué carpeta será guardada esta copia.

```
base_url = "https://www2.emergencias.com.ar/"  # Cambia al sitio objetivo
output_dir = "clon_completo_emervieja"
visited_urls = set()
max_depth = 5  # Profundidad máxima
```

Figura 6. Configuración de sitio a clonar

Luego de realizar esta configuración, se procede a correr el script python que se encarga de recorrer al objetivo y extraer todas las imágenes, archivos javascript y frontend del sitio elegido, guardando todos los objetos clonados y/o descargados en la carpeta que le hemos indicado con anterioridad. En el caso de

este proyecto, al disponer de varios sitios públicos, se ha realizado un clonado de los siguientes sitios:

- https://avion.emergencias.com.ar
- https://www2.emergencias.com.ar
- https://emergencias.com.ar
- https://cursos.emergencias.com.ar
- https://auditoriascume.emergencias.com.ar

Una vez realizada dicha acción, procedimos a copiar todos los resultados a la carpeta **full_integration** y allí a la carpeta **sites**, donde se encuentran los sitios clonados mencionados con anterioridad.

La estructura de directorios relacionados con el servicio web queda de la siguiente manera

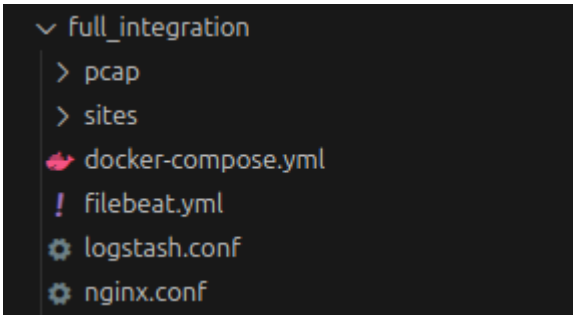

Figura 7. Estructura de directorios de emulación web

El archivo encargado de orquestar la creación de los contenedores posee varios servicios sincronizados que son encargados de realizar diferentes tareas para que la aplicación funciones de manera correcta

Nginx

El servicio definido como nginx dentro del archivo docker-compose.yml se encarga de utilizar el archivo **nginx.conf** para aplicar las configuraciones de permisos y accesos a los sitios que hemos clonado. En este contenedor, se monta la carpeta del servidor que contiene el código de los sitios en el directorio **/usr/share/nginx/sites** y el archivo de configuraciones [84], [85].

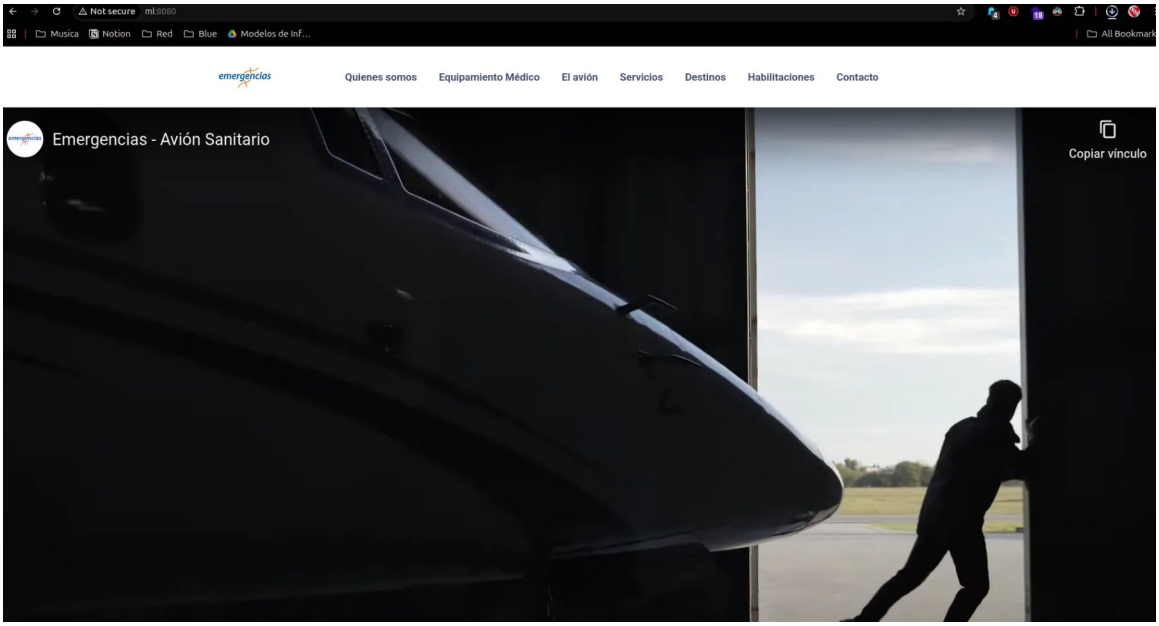

Figura 8. Sitio clonado en funcionamiento

Filebeat

Este contenedor se encarga de setear la recolección de los archivos **log** que se van poblando con logs de red y se envían a logstash para su posterior transformación. El servicio se configura con el archivo **filebeat.yml** que se encuentra en el directorio y este mismo archivo se monta dentro del contenedor, además de montar la carpeta que contiene los logs de red creados por Zeek.

Logstash

La funcionalidad del servicio de logstash es poseer un demonio en escucha esperando datos que sean ingresados por filebeat y, a su vez, transformar y redireccionar esta misma información hacia la base de datos de elasticsearch. Se encuentra configurado por el archivo **logstash.conf** que se monta en el contenedor.

Elasticsearch

Este servicio almacena los logs que transforma logstash y que luego procedemos a visualizar dentro de la aplicación web. Se configuró luego de realizado el inicio del servicio ya que se debió crear un patrón para agregar los eventos recibidos a un índice.

Kibana

El contenedor de este servicio sirve una aplicación web que permite ver todos los datos ingestados dentro de la base de datos de elasticsearch, para su posterior explotación y análisis. Este servicio fue configurado para solo ser accesible por el dirección **loopback** del servidor, por lo que debe accederse a ella a través de una conexión **proxy socks 5** por SSH contra este servidor**.**

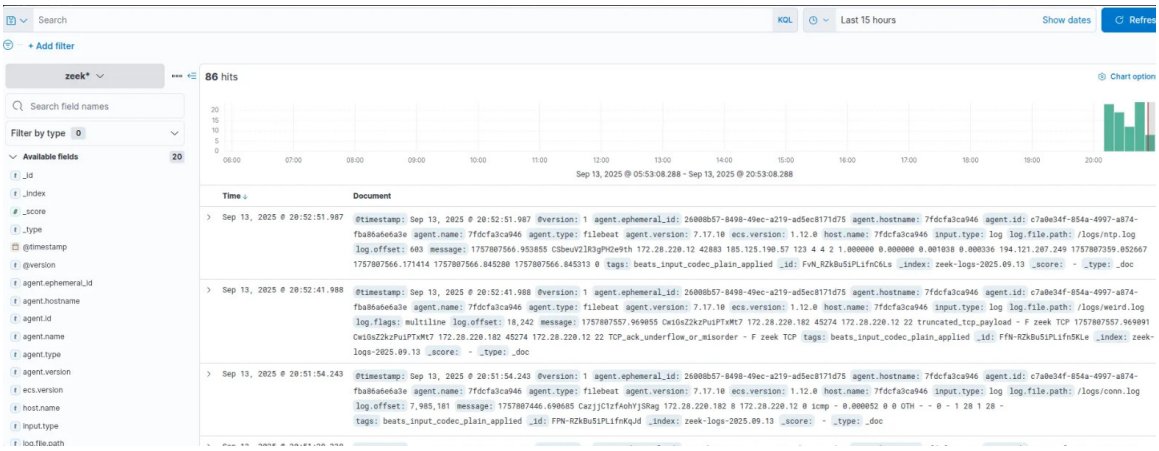

Figura 9. Servicio Kibana mostrando los logs ingestados

Zeek

Este servicio pone un sniffer en la placa de red para ver el tráfico que está aconteciendo en el servidor para luego transformarlo en archivos legibles para el usuario [86],[87]. La configuración del servicio se hace a través del archivo de **docker-compose** donde se procede a configurar que el servicio escuche en una interfaz de red determinada, convierta los logs a un formato tipo JSON, que posea las capacidades de **Administrador de Red** y **Administración de red,** montado con permisos de lectura y escritura dentro de la carpeta **pcap** del servidor y por último, establecer la red en modo **host** [52],[88].

Con este despliegue utilizando el comando **docker compose up -d** se crearon todos los servicios anteriormente mencionados, teniendo dos redes docker (**zeeknet** que se encarga de comunicarse con la interfaz de red del servidor y **elknet** que comunica todos los servicios de almacenamiento de logs) y el puerto **http** sirviendo todas las aplicaciones web usando **nginx** como proxy reverso y mapeando los sitios con diferentes nomenclaturas.

5.3. Captura y procesamiento del tráfico

En este apartado, nos enfocaremos en el proceso de captura y procesamiento de tráfico por parte del honeypot y la herramienta zeek, cuya finalidad es la de realizar esta tarea [70].

El proceso comienza como hemos mencionado en el punto anterior, configurando al contenedor con capacidades de administración de red del sistema operativo permitiendo el acceso a todos los datos relacionados con los procesos de red que se encuentran corriendo en el sistema operativo [88].

Una vez que el servicio comienza la captura y procesamiento de los datos, se crean diferentes archivos de log en formato json que se agrupan según el siguiente criterio [70]:

- **Conn.log:** archivo que concentra la telemetría de las conexiones del equipo
- **Dhcp.log:** archivo que concentra los eventos relacionados al protocolo DHCP
- **Dns.log:** archivo que concentra los eventos relacionados a paquetes DNS
- **Files.log:** archivo que capta los archivos que se han observado durante la captura de paquetes
- **Http.log:** archivo que concentra los eventos de red del protocolo HTTP
- **Ntp.log:** archivo que concentra los eventos de red del servicio NTP
- **Ssh.log:** archivo que contiene los detalles sobre las sesiones SSH
- **Ssl.log:** archivo que contiene la información de los paquetes relacionados con SSL/TLS
- **Weird.log:** archivo que concentra los datos que no han sido posible procesar correctamente.

Todos estos archivos contienen diferentes tipos de datos que se unen mediante el campo **uuid**, el cuál es el valor que correlaciona todos los eventos en todos los archivos que crea la herramienta. En este documento no se detallaran todos los campos y valores de los archivos mencionados con anterioridad sino que nos centraremos en el archivo de conexiones el cual posee los campos **ts, uuid, id.orig_h, id.orig_p, id.resp_h, id.resp_p, proto, duration, orig_bytes,resp_bytes, conn_state, missed_bytes, history ,orig_pkts, orig_ip_bytes, resp_pkts, resp_ip_bytes**.

Estos campos delimitan toda la actividad de conexión en el servidor, tanto entrante como saliente, por lo que mediante la eliminación y transformación de

datos, estos son la base de los atributos que utilizaremos en nuestra herramienta [36].

5.4. Entrenamiento y aplicación del modelo

Con el fin de poder aplicar un modelo de machine learning a la herramienta, como primer paso procedemos a realizar el análisis del dataset utilizado (en este caso USW_NB15), qué atributos del mismo podremos utilizar, como se entrenará el modelo y de qué manera será aplicada la inferencia [3],[76] .

Como primer paso, utilizamos los datos normalizados que nos brinda el dataset USW-NB15 cuya consolidación de datos proviene de fuentes tales como BRO-IDS y Argus siendo el creador de las mismas el doctorado **Nour Moustafa** quien ha dejado público este conjunto de datos para utilizarlos como parte de investigación [70].

Para los propósitos de nuestra herramienta, este conjunto de datos es ideal ya que,además de poseer datos normalizados, categorizados y fidedignos, los mismos se encuentran actualizados y son compatibles con el contexto actual de la tecnología [3],[4].

Una vez que hemos establecido el conjunto de datos que utilizaremos, procedemos a utilizar los archivos **UNSW_NB15-testing-set.csv** y **UNSW_NB15-training-set.csv** [96] correspondientes a los datos de pruebas y datos de entrenamiento respectivamente, teniendo un total de 82.332 registros para el dataset de pruebas y 175.341 registros para el dataset de entrenamiento [3],[76].

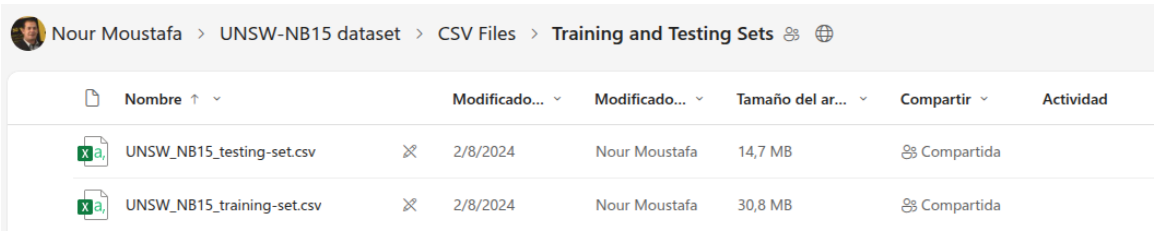

Figura 10. Repositorio compartido de los dataset

Habiendo obtenido los archivos correspondientes, procederemos con el análisis del archivo que contiene el conjunto de datos de entrenamiento, observando que los atributos que presenta el mismo son **45** en total, por lo que debemos observar cuales de todos estos atributos nos sirven para aplicar correctamente

la inferencia posteriormente al entrenamiento ya que el tipo de datos y atributos deben coincidir tanto en el modelo como en los datos a probar en la inferencia [22],[25].

En el proceso que conllevo un riguroso análisis de parte de los datos que posee el dataset, tomamos como piedra angular el atributo **label** el cuál constituye al valor **0** como un registro normal y al valor **1** como un registro anómalo o malicioso [3],[76]. En este proyecto, hemos evitado utilizar el atributo categoría que incluye diferentes tipos de ataque debido a que eso conlleva una mayor correlación y optimización de fuentes que debido a los eventos que registra **zeek** no podían aplicar en esta instancia del trabajo [43].

Con este atributo fundamental elegido, se procede a realizar el proceso de poda correspondiente, divisando los atributos provenientes del log de red del archivo **conn.log** y divisando que atributo del dataset es compatible con los datos que tenemos para no entrar en conflicto al momento de realizar la inferencia, siendo los atributos **dur,sbytes, dbytes, spkts y dpkts** seleccionados como datos numéricos y, a su vez, los atributos **proto, state y service** seleccionados como los atributos categóricos [35],[36],[70] .

Debido al tipo de proyecto realizado, se ha notado que el modelo de machine learning que mejor adapta las necesidades del proyecto con sus capacidades es **RandomForest** debido a que se busca clasificar el tráfico (algo que el algoritmo realiza muy bien), trabajar con atributos numéricos y categóricos, relacionar atributos no lineales entre sí y poseer datos ruidosos y desbalanceados, por lo que se planteó utilizar en la codificación este algoritmo [5],[6].

Durante la fase de diseño, se había planteado como objetivo tener un 90% de exactitud global por lo que esta fue nuestra meta para seleccionar el modelo final a utilizar, por lo que comenzamos a realizar la codificación del script para obtener un modelo acorde a nuestras expectativas.

Como primer paso, se procedió a trabajar sobre una copia en memoria del dataset de entrenamiento para, posteriormente, trabajar sobre el mismo con el fin de obtener el modelo [21]. Luego de realizada la copia, se procede a normalizar los caracteres de las valores del dataset con el fin de evitar errores al momento del procesamiento de datos, aplicando expresiones regulares y cambios de palabras devolviendo un objeto dataframe con los valores limpios y listos para ser utilizados en una siguiente etapa [27],[77].

En consecuencia, los datos tratados en la etapa anterior proceden a pasar por otra etapa de normalización donde se cambian aquellos valores categóricos que no posean una cadena de caracteres válida proceden a reemplazarse con un valor correcto como así también se procede a normalizar los valores numéricos que no estén definidos correctamente dentro del dataframe [21],[25],[77]. Como último paso del proceso de normalización de datos se procede a preparar las etiquetas de los datos para su posterior utilización en el entrenamiento del modelo.

```python
def coerce_types(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()
    for c in ["proto","state","service","attack_cat"]:
        if c in df.columns:
            df[c] = df[c].astype(str).str.strip().str.lower().replace({"nan": np.nan, "-": np.nan})
    for c in BRIDGE_NUM:
        if c in df.columns:
            df[c] = pd.to_numeric(df[c], errors="coerce")
    return df
```

Figura 11. Parte del proceso de normalización de datos

Una vez concluida la normalización de datos, procedemos a aplicar un pre procesamiento de datos donde se realiza el relleno de datos faltantes, se convierte las categorías en columnas binarias y se escalan los valores numéricos para que puedan ser comparables, teniendo así un bloque de preprocesamiento de datos que se toma como pre ingreso antes de entrenar el modelo [77],[6],[22]. Para el entrenamiento del modelo, hemos erigido la configuración de random forest con los valores **balanced subsample** en peso de clase, **800** en número de estimadores y con un random state de **42** que es utilizado sólo por convención cultural, ya que el entrenamiento se realiza de igual manera con cualquier número entero [5],[24].

En esta etapa del entrenamiento luego de varios procesos de entrenamiento donde se buscaba obtener el mejor ajuste posible, se aplicó el método de **GridsearchCV** para optimización de hiperparametros para obtener la mejor combinación del modelo y aplicar el entrenamiento con este [28],[29]. Para obtener la mejor configuración se han realizado diversas pruebas internas con foco en la clase **1 (malicioso)** y priorización en el **F1 score** con diferentes valores en los siguientes parámetros:

- **n_estimators:** cantidad de árboles que puede albergar el modelo (valores utilizados en la prueba: 200,400,800)

- **max_depth:** profundidad de las ramas de los árboles (valores utilizados: None, 10,20,30)
- **min_split:** cantidad de datos mínimos necesarios para poder dividir una rama (valores utilizados: 2,5,10)
- **min_leaf:** cantidad de datos mínimos que debe poseer la predicción final del arbol (valores utilizados: 1,2,4)
- **max_features:** cuantas columnas utiliza el árbol para realizar la predicción (valores utilizados: sqrt,log)

Luego de la prueba realizada, donde internamente el random forest realiza todas las combinaciones posibles utilizando los parámetros que definimos para devolver la combinación que da el mejor score global entre la combinación de la precision y el recall dando como resultado la configuración **scorer (beta=0.8,pos_label=1), param_grid (n_estimators=800,max_depth=30,split=2,leaf=1,features=log2), cv (n_splits=3,shuffle=true,random_state=42)**

Luego de esto determinamos el mejor umbral con una función que fue probando distintos valores y dando un valor óptimo de **0.55**. Luego de construir el modelo obtenemos el siguiente gráfico de la matriz de confusión, la cuál nos indica el número de clasificaciones acertadas, los falsos positivos y los falsos negativos

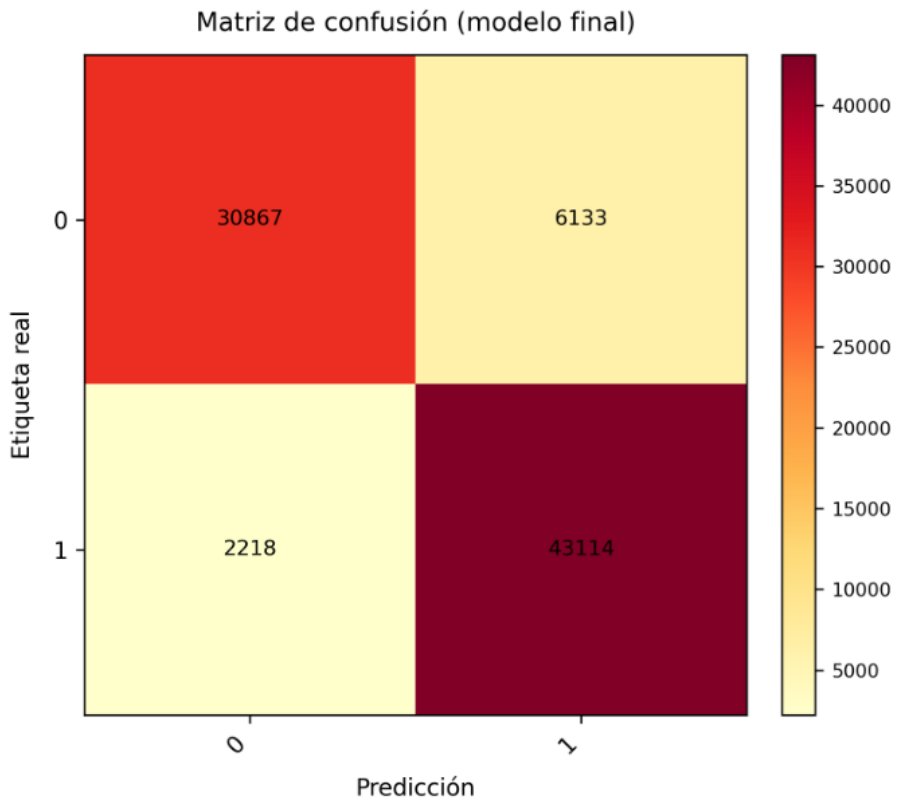

Figura 12. Matriz de confusión creada con los resultados del entrenamiento

Además, en el siguiente gráfico se observa la diagonal principal con un valor cercano a **1** lo cuál indica una gran precisión en la predicción y un valor cercano a **0** en la diagonal secundaria que denota una baja tasa de predicciones incorrectas.

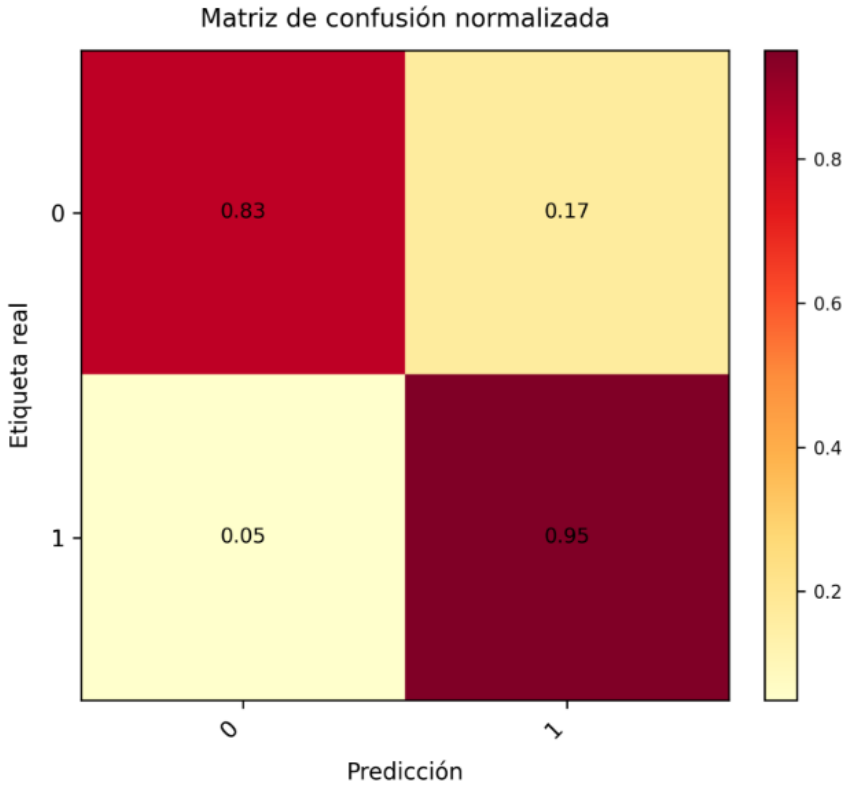

Figura 13. Normalización de matriz de confusión

A su vez, obtenemos el reporte de clasificación basado en las dos etiquetas que definimos en un principio, determinado los porcentajes obtenidos en **Precision, Recall y F1-Score**, siendo la **clase 0** aquella clasificada como **tráfico normal** y la **clase 1** aquella clasificada como **tráfico malicioso**. En el siguiente gráfico se puede observar los resultados de la matriz de confusión obtenida, siendo visible que la prioridad fue obtener un **Recall** alto para la **clase 1** ya que es la que indica el tráfico proveniente de un atacante.

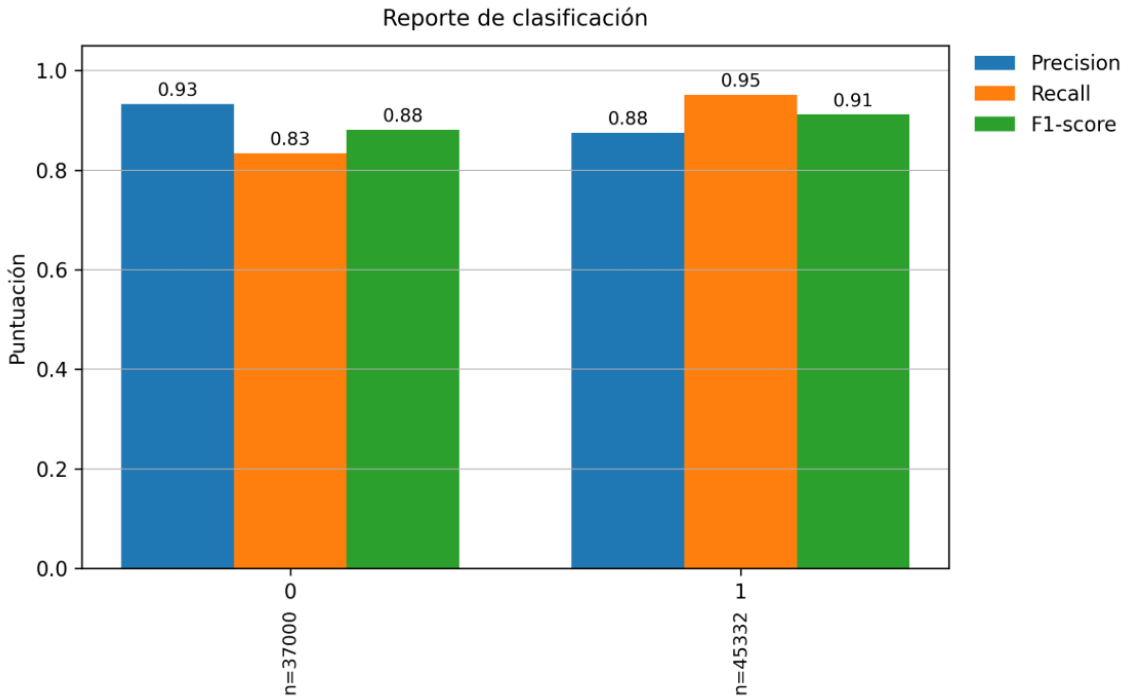

Figura 14. Gráfico de barras de reporte de clasificación

Como última parte del proceso, procedemos a realizar un script en python que utiliza el modelo que hemos obtenido en el procedimiento anterior y cree un bucle infinito donde realiza una lectura del archivo de conexiones creado por **zeek** y nos indique por pantalla cada **0,5** segundos si el tráfico que acaba de leer es benigno o malicioso, indicando la ip de origen, el score que le dio la probabilidad, la ip destino y el uid del registro, obteniendo en tiempo real una alerta de clasificación basado en el modelo.

6. Resultados y evaluación

En este apartado se detallan los resultados obtenidos durante la implementación del servidor y como este ha respondido durante las jornadas en que el mismo se encontraba operativo.

6.1. Desempeño del modelo de detección

Una vez concluido el proceso de entrenamiento y validación mediante **gridsearchCV**, se seleccionó el modelo con mejor desempeño priorizando la reducción de falsos positivos sin sacrificar una sensibilidad adecuada.
El modelo óptimo correspondió a un **Random Forest** configurado con los hiperparámetros **n_estimators = 800, max_depth = 30, max_features = "log2", class_weight = "balanced_subsample", min_samples_leaf = 1 y min_samples_split = 2.**

La evaluación final se realizó sobre el conjunto de prueba independiente, utilizando el umbral de decisión ajustado con un recall mínimo del 95%.

Métrica	Clase 0 (Benigno)	Clase 1 (Malicioso)	Promedio ponderado
Precisión	0.93	0.88	0.90
Recall	0.83	0.95	0.89
F1-score	0.88	0.91	0.90
Accuracy global			0.89

Figura 15. Matriz de confusión

Estos valores indican que el modelo logró una precisión promedio del 90 %, con una capacidad de detección de ataques (recall) del 95 %.
La matriz de confusión muestra que la mayoría de los falsos positivos se concentraron en tráfico benigno clasificado erróneamente como malicioso, mientras que los falsos negativos (ataques no detectados) fueron mínimos, cumpliendo con el objetivo de mantener un *recall* elevado.

En términos de equilibrio entre *precision* y *recall*, el *F1-score* alcanzado evidencia un modelo robusto y generalizable, adecuado para ser desplegado dentro del entorno de detección del honeypot.

El análisis de importancia de atributos evidencia que las variables más determinantes en la predicción corresponden a métricas de flujo tales como duración de la conexión, bytes enviados y recibidos, número de paquetes, y protocolo de transporte.

Estos resultados coinciden con la investigación sobre detección de intrusiones basada en flujos de red, donde dichas características son indicativas de comportamientos anómalos y patrones de ataque.

Finalmente, el modelo se integró al pipeline de inferencia del sistema, procesando en tiempo real los flujos capturados por Zeek.

6.2. Comportamiento del honeypot en producción

En este aparte, relataremos el comportamiento del honeypot en producción expuesto públicamente en la red internet.

En cuánto a lo que fueron las pruebas en tiempo real al principio del despliegue, se ha utilizado una máquina virtual perteneciente al mismo segmento de red para asegurarnos que los equipos se vean entre sí, asegurando que no se pierdan paquetes al momento de realizar los ataques planificados. Para este caso, se realizaron pruebas utilizando herramientas de descubrimiento de puertos como nmap como así también utilidades de escaneo de aplicación tales como wpscan y nikto, los cuáles fueron detectados correctamente por la herramienta en cuánto el código de inferencia alcanzó dicha actividad durante el procesamiento del archivo [89],[90],[91].

El siguiente paso ha sido la publicación del honeypot a través de la utilización de una instancia ec2 en AWS donde se ha implementado el servicio de docker y se ha desplegado el directorio que contiene el modelo, los archivos de configuración de los servicios, un archivo adicional de un servicio que establece la manipulación de las alertas generadas y los códigos relacionados a la inferencia del modelo. Como primera parte, se ha generado una instancia que posee un hardware similar al descrito con anterioridad en la sección de diseño, se han aplicado medidas de prevención tales como permitir el acceso a través de un host bastión con un puerto de conexión segura en escucha en un puerto no convencional y permitiendo conexiones administrativas sólo a través de este

servidor hacia la máquina que contiene la simulación de las páginas de la empresa. Una vez realizadas estas configuraciones, se procede a ingresar al servidor a través de su puerto de conexión segura (ssh) y una vez allí, se realiza la instalación del software **docker** que permite que se puedan instanciar los servicios de la herramienta, después se procede a cambiar los permisos y ownership del archivo de **filebeat.yml** que se encarga de la ingesta de logs en **elasticsearch** y luego procedemos a utilizar el comando **docker compose up -d** para poder correr todos los servicios como se ha planificado con anterioridad. Adicionalmente, para evitar correr manualmente el archivo python encargado de la predicción, se creó un servicio adicional que realiza esta tarea como un servicio, detectando direcciones ip maliciosas y guardandolas en un archivo para su posterior análisis. Cabe destacar que también se han agregado registros **DNS** para que los subdominios creados bajo el dominio **grupoemergencias.com** redirijan su tráfico a nuestra instancia **ec2** al ser solicitados.

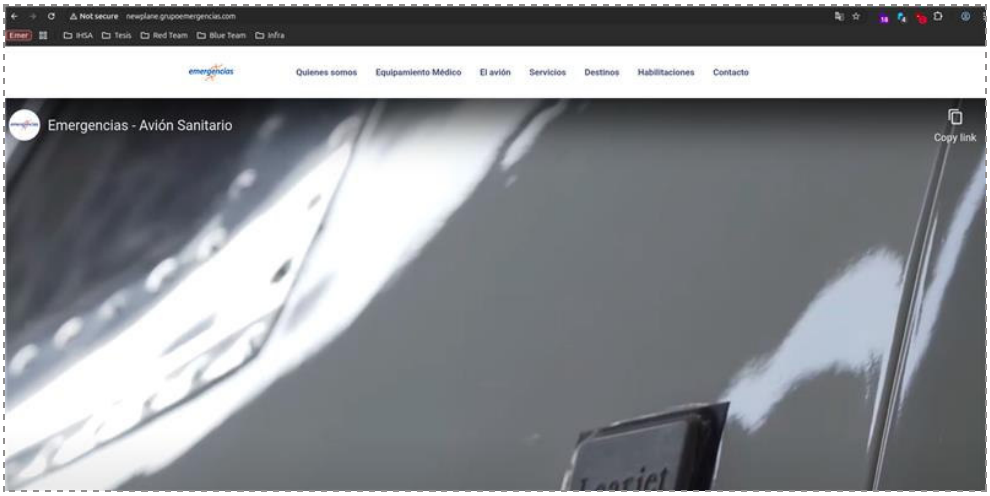

Figura 16. Sitio wordpress de Avión simulado .Ver en Avion Emergencias – Vuelos sanitarios

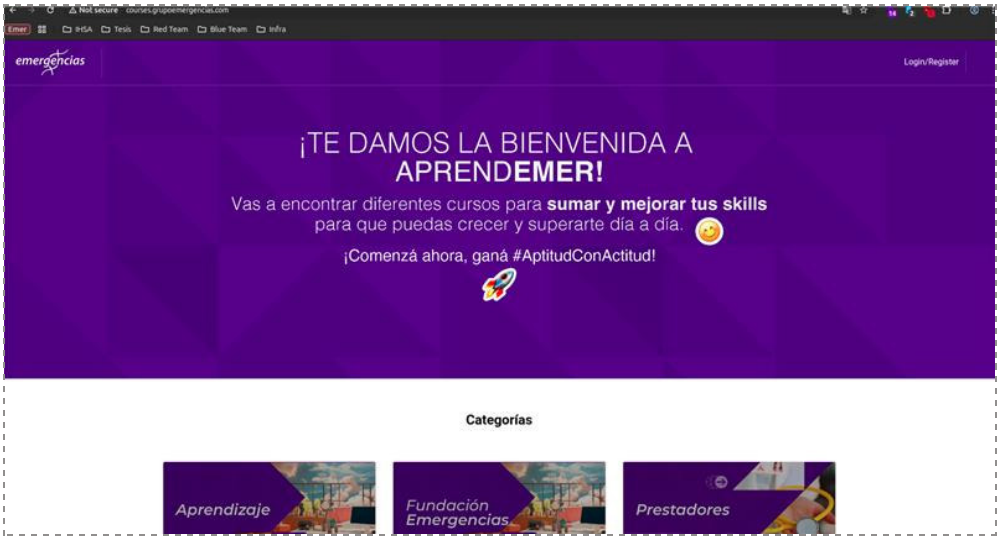

Figura 17. Sítio Moodle de cursos. Ver en cursos.fundacionemergencias.org

Una vez realizada la implementación del servicio comenzamos a monitorizar el comportamiento del software, observando en tiempo real si se comporta según lo esperado y como fue diseñado, observando que los logs de red se estén guardando según lo previsto.

6.3. Discusión de resultados y validación

En esta sección, describiremos los resultados obtenidos luego de haber dejado en producción el servidor que posee el despliegue de nuestro honeypot con sus sitios simulados y monitorizados.

Como primer dato a destacar, se ha dejado el servidor corriendo por algunas horas, con vigilancia activa y un equipo de respuesta activo en caso de que el servidor resultase comprometido. Durante el transcurso de tiempo en el que el servidor estuvo operativo y respondiendo las solicitudes que le llegaban desde internet, pudimos observar diferentes comportamientos en torno a la actividad del servidor [92].

Antes que nada, nuestro equipo ha realizado un escaneo a los sitios a modo de prueba de que este detectando correctamente la detección de puertos y los análisis de aplicación realizados por herramientas que nosotros controlamos tal como hemos hecho con anterioridad cuyo resultado fue satisfactorio, indicando que detecta de manera correcta ataques dirigidos.

Con el paso del tiempo, se observaron diferentes comportamientos, por ejemplo que los javascripts que vienen en las páginas clonadas realizaban comunicaciones contra los servidores de google u otros terceros. En cuánto a los paquetes de entrada que estaba recibiendo el servidor, ha podido observarse el comportamiento de los bots especializados de los ciberdelincuentes que realizan un barrido inicial, donde se comunican con el dominio pero no por el servicio http, dejando así muy pocos paquetes para análisis aunque en futuras validaciones se ha visualizado que estas direcciones ip están actualmente reportadas [93],[94],[95].

Figura 18. Ip atacante con escaneo especializado.

Figura 19. Análisis de ip identificada en la figura 19.

Por otra parte, en cuánto los atacantes intentaron realizar un barrido de puertos y un análisis de aplicación, la aplicación los detectó correctamente y fueron incluidos en la lista de direcciones origen identificadas, validando luego que esta dirección se encontraba reportada como maliciosa.

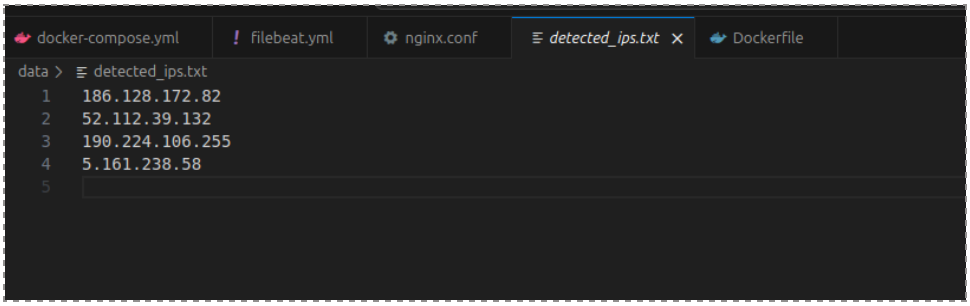

Figura 20. Listado de ips maliciosas detectadas.

Figura 21. Analisis de reputación de ip 5.161.238.58

Por lo tanto, el comportamiento en producción fué satisfactorio mostrando un sistema eficaz y eficiente para la identificación de potenciales ataques.

7. Conclusiones y próximos pasos

En esta sección abordaremos las conclusiones relacionadas al proyecto, los percances que tuvieron lugar durante la realización del mismo y que funcionalidades deberían incluir en próximas versiones en caso de seguir desarrollando el proyecto.

7.1. Conclusiones

Luego de la realización de este proyecto podemos concluir que la inteligencia artificial es una herramienta muy poderosa y de gran utilidad pero que a su vez debemos contar con conjuntos de datos relacionados a la finalidad del proyecto que estemos realizando, siendo este caso, un caso de éxito debido a que se contó desde un principio con datos normalizados y estandarizados que solamente necesitan ser explotados mediante la transformación de los atributos a aquellos que necesitemos para que dicha solución funcione de una manera correcta. En cuanto al aprendizaje automático, podemos concluir que es una herramienta que puede presentar dificultad en cuanto a su aprendizaje en instancias preliminares pero que luego de realizar diversas pruebas y aprender de los errores, posee una gran capacidad para moldear ideas y proyectos de una manera muy simple y precisa, adaptando nuestras necesidades y/o objetivos a los datos y al modelo que elijamos para resolver la problemática que nos planteamos.

Además de esto, también la portabilidad y sencillez de Docker permiten replicar cualquier tipo de aplicación en diferentes entornos tales como un servidor dedicado o como parte de un cluster de kubernetes, permitiendo agregar diversas herramientas que nos permiten realizar diferentes funciones según nuestra necesidades y si aquellas funciones no existiesen, es posible crearlas y que las mismas sean portables. Sumado a lo mencionado anteriormente, los requerimientos para la simulación de las aplicaciones son realmente bajos, pudiendo configurar varias aplicaciones al mismo tiempo administradas bajo un solo servicio,como es el caso de nginx, y ,a su vez, estandarizando código y funcionalidad que permite la reutilización por parte de terceros de la aplicación y/o funcionalidades como así también pudiendo incorporar herramientas que sirvan para el monitoreo de los diferentes sitios clonados y que incluso pueden ser utilizadas en servicios de observabilidad de operaciones de centros de seguridad. También es de destacar que se disminuye el riesgo de compromiso del servidor utilizando servicios siguiendo el principio de Zero Trust, reduciendo el riesgo de escalamiento de privilegios en caso de compromiso externo.

Por último, el lenguaje de programación utilizado para todo el proyecto ha sido python, el cuál es muy versátil, posee una diversa variedad de librerías para todo tipo de acciones y funciones y es muy útil para proyectos como este donde debe codificarse scripts de entrenamiento de modelos de Machine Learning u otros tales como la copia de todos los elementos de un aplicativo web.

7.2. Limitaciones encontradas y próximos pasos

Durante la ejecución del proyecto se han encontrado escollos que han dificultado el avance. Comenzamos con la curva de aprendizaje de Machine Learning debido a que puede ser abrumadora para aquellos profesionales que no la utilizan en su labor diaria profesional o que no han interactuado con el mundo de la inteligencia artificial en su tiempo libre, representando un esfuerzo considerable para comprender correctamente las acciones involucradas, especialmente aquellas destinadas a analizar cómo mejorar los resultados obtenidos, donde ajustar para obtener un mejor rendimiento del modelo y cómo utilizar las evidencias para verificar su correcto funcionamiento.

Otra de las limitaciones encontradas fue el tiempo para realizar el proyecto, debido a la carga que representa el aprendizaje de ML y la comprensión de sus actividades, impactando en el recorte de las features planteadas originalmente, planteando para los próximos pasos varias ideas que podrían enriquecer aún más este proyecto.

Luego, consideramos necesario plantear los recursos necesarios, presentando restricciones al momento de implementar el proyecto en un entorno empresarial debido a los presupuestos acotados y la dificultad para persuadir a la gerencia respecto a inversiones en proyectos de ciberseguridad los cuales no son considerados como un beneficio potencial para la compañía sino que son catalogados como un gasto debido a que no les otorga un retorno. En el caso particular del proyecto, el entrenamiento del modelo requiere disponer de un servidor con recursos de cómputo elevados que en el caso de ciertas empresas o entornos requiere un costo imposible de abordar por lo cual se deben realizar recortes en las características de los modelos impactando en el desempeño final del sistema o en el tiempo necesario para realizar el procesamiento de los conjuntos de datos y entrenamiento del modelo.

Adicionalmente, podemos mencionar la securización del entorno de implementación de la herramienta. En este caso, las restricciones de tiempo en las etapas finales del proyecto impidieron el desarrollo de la automatización de

la seguridad del servidor, debiendo aplicar un procedimiento manual a medida que se realizaron las diferentes actividades del proyecto.

En cuanto a acciones a realizar con el proyecto podemos establecer varios puntos que pueden mejorar las tareas que realiza la herramienta. Como primer punto, podemos tomar la implementación mediante infraestructura como código lo cuál nos permite automatizar tanto el despliegue del esqueleto de la máquina virtual o servidor como así también su configuración con todas las herramientas y características que permiten una correcta implementación de manera automática.

El segundo punto es la tratativa de registros DNS y certificados, por ejemplo que se permite buscar dominios similares al clonado y comprarlos para su posterior utilización y despliegue de dominios DNS o en su defecto, la automatización para utilizar servicios de creación de certificados SSL para poder adjuntos al servidor y simular aún más la veracidad y autenticidad de los sitios.

Como tercer punto y que viene en combinación del punto anterior, se puede utilizar traefik para manejar los nombres y las redirecciones de los sitios simulados, pudiendo centralizar las configuraciones en un solo servicio.

En cuarto punto colocamos la implementación de un LLM dentro de los servicios para poder crear configuraciones de docker basados en la tecnología con la que fueron programados los sitios y así poder tener un sitio con fidelidad más alta respecto al original. A su vez, esta herramienta también permitirá crear código backend para los maquetados clonados para que estos sitios simulan tener una funcionalidad completa y que no lancen errores al querer interactuar con los diferentes elementos del sitio.

Por último podemos nombrar a la posibilidad de integrar los servicios preexistentes con una solución SOAR que nos permita realizar tareas de respuesta a incidentes u detecciones en el servidor, evitando el uso de recursos humanos adicionales que pueden estar colaborando en otras tareas con el equipo.

8. Bibliografia

[1] P. Mell, K. Kent and J. Nusbaum, “Guide to Intrusion Detection and Prevention Systems (IDPS)”, NIST SP 800-94, 2007.
[2] L. Spitzner, Honeypots: Tracking Hackers. Boston, MA, USA: Addison-Wesley, 2003.
[3] N. Provos and T. Holz, Virtual Honeypots: From Botnet Tracking to Intrusion Detection. Upper Saddle River, NJ, USA: Addison-Wesley, 2007.
[4] T. Hastie, R. Tibshirani and J. Friedman, The Elements of Statistical Learning. Springer, 2009.
[5] L. Breiman, “Random Forests”, Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.
[6] N. Moustafa and J. Slay, “UNSW-NB15: A Comprehensive Data Set for Network Intrusion Detection Systems”, MilCIS, 2015.
[7] I. Sharafaldin, A. H. Lashkari and A. A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization”, ICISSP, 2018.
[8] S. J. Stolfo et al., “Cost-Based Modeling and Evaluation…”, ACM SIGMOD Record, 2000.
[9] C. Modi *et al.*, “A Survey of Intrusion Detection Techniques in Cloud”, *Journal of Network and Computer Applications*, 2013.
[10] Verizon, *2024 Data Breach Investigations Report (DBIR)*, Verizon Enterprise, 2024.
[11] Ministerio de Seguridad de la República Argentina – AR-CERT, *Reporte Anual de Incidentes de Seguridad 2024*, Centro de Respuesta a Incidentes de Seguridad en Tecnologías de la Información, 2024.
[12] S. Axelsson, “Intrusion Detection Systems: A Survey and Taxonomy”, *Technical Report*, Chalmers University of Technology, 2000.
[13] ENISA, *ENISA Threat Landscape 2023*, European Union Agency for Cybersecurity, 2023.
[14] D. Somerville, “Cyber-Security as a Socio-Technical Problem”, *European Conference on Cyber Warfare and Security*, 2015.
[15] S. Zargar, J. Joshi and D. Tipper, “A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks”, *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.
[16] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed., Prentice Hall, 2010.
[17] A. L. Samuel, “Some Studies in Machine Learning Using the Game of Checkers”, *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.

[18] J. Searle, “Minds, Brains, and Programs”, *Behavioral and Brain Sciences*, vol. 3, no. 3, pp. 417–457, 1980.
[19] Y. LeCun, Y. Bengio and G. Hinton, “Deep Learning”, *Nature*, vol. 521, pp. 436–444, 2015.
[20] B. Goertzel and C. Pennachin, *Artificial General Intelligence*, Springer, 2007.
[21] T. Mitchell, *Machine Learning*, McGraw-Hill, 1997.
[22] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016.
[23] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 2018.
[24] T. K. Ho, “Random Decision Forests”, *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, Montreal, Canada, pp. 278–282, 1995.
[25] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, Springer, 2009.
[26] L. Rokach, “Ensemble-based classifiers”, *Artificial Intelligence Review*, vol. 33, pp. 1–39, 2010.
[27] S. Raschka and V. Mirjalili, *Python Machine Learning*, 3rd ed., Packt Publishing, 2019.
[28] J. Bergstra and Y. Bengio, “Random Search for Hyper-Parameter Optimization”, *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.
[29] R. Kohavi, “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection”, *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1137–1145, 1995.
[30] S. B. Kotsiantis, “Supervised Machine Learning: A Review of Classification Techniques”, *Informatica*, vol. 31, no. 3, pp. 249–268, 2007.
[31] A. Callado *et al.*, “A Survey on Internet Traffic Identification”, *IEEE Communications Surveys & Tutorials*, vol. 11, no. 3, pp. 37–52, 2009.
[32] K. Salah, “Understanding Internet Traffic Streams”, *IEEE Communications Surveys*, vol. 7, no. 1–4, pp. 70–78, 2005.
[33] B. Claise, “RFC 5101: Specification of the IPFIX Protocol”, IETF/IEEE Standard, 2008.
[34] C. Estan and G. Varghese, “New Directions in Traffic Measurement and Accounting”, *ACM SIGCOMM*, 2002.
[35] M. Thottan and C. Ji, “Adaptive Traffic Anomaly Detection Using Flow-Level Information”, *IEEE INFOCOM*, 2003.
[36] T. Nguyen and G. Armitage, “A Survey of Techniques for Internet Traffic Classification Using Machine Learning”, *IEEE Communications Surveys & Tutorials*, vol. 10, no. 1, pp. 56–76, 2008.

[37] R. Sommer and V. Paxson, “Outside the Closed World: On Using Machine Learning for Network Intrusion Detection”, *IEEE Symposium on Security and Privacy*, 2010.
[38] J. Gao *et al.*, “Detecting Sophisticated Attacks via Behavioral Anomaly Detection”, *IEEE Transactions on Dependable and Secure Computing*, 2019.
[39] N. Moustafa and J. Slay, “UNSW-NB15 Dataset”, *MilCIS*, 2015.
[40] A. Moore and D. Zuev, “Internet Traffic Classification Using Bayesian Analysis”, *ACM SIGMETRICS*, 2005.
[41] R. Vinayakumar *et al.*, “Deep Learning Approaches for Network Intrusion Detection”, *IEEE Access*, 2019.
[42] S. Potluri and C. Diedrich, “Evaluation of Honeypot Systems for Industrial Control Systems”, *IEEE ETFA*, 2013.
[43] V. Chandola, A. Banerjee and V. Kumar, “Anomaly Detection: A Survey”, *ACM Computing Surveys*, 2009.
[44] L. Portnoy, E. Eskin and S. Stolfo, “Intrusion Detection with Unlabeled Data Using Clustering”, *ACM Workshop on Data Mining Applied to Security*, 2001.
[45] National Institute of Standards and Technology (NIST), “Framework for Improving Critical Infrastructure Cybersecurity”, NIST, 2018.
[46] ISO/IEC 27001:2022, *Information Security, Cybersecurity and Privacy Protection — Information Security Management Systems — Requirements*, International Organization for Standardization, 2022.
[47] ENISA, *Cybersecurity Threat Landscape 2023*, European Union Agency for Cybersecurity, 2023.
[48] R. Bace and P. Mell, *Intrusion Detection Systems*, NIST SP 800-31, 2001.
[49] C. Seifert, I. Welch and P. Komisarczuk, “HoneyC — A Honeypot for Client-Side Attacks”, *2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*.
[50] M. Nawrocki *et al.*, “A Survey on Honeypots, Honeynets and Their Applications on Smart Grid”, *IEEE Communications Surveys & Tutorials*, 2016.
[51] G. Kaur and A. Singh, “Taxonomy of Honeypots and Intrusion Detection Systems”, *2016, 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*.
[52] D. Merkel, “Docker: Lightweight Linux Containers for Consistent Development and Deployment”, *Linux Journal*, no. 239, 2014.
[53] B. Burns, B. Grant, D. Oppenheimer, E. Brewer and J. Wilkes, “Borg, Omega, and Kubernetes”, *Communications of the ACM*, vol. 59, no. 5, pp. 50–57, 2016.
[54] J. Turnbull, *The Docker Book: Containerization Is the New Virtualization*, 2014.
[55] M. Fowler and J. Lewis, “Microservices: a Definition of this New Architectural Term”, *IEEE Software*, vol. 33, no. 1, pp. 22–27, 2016.

[56] P. Mell and T. Grance, “The NIST Definition of Cloud Computing”, NIST SP 800-145, 2011.
[57] NIST, “Guide to Enterprise Network Security Architecture”, NIST SP 800-160, 2022.
[58] SANS Institute, “DMZ Network Design”, SANS Whitepaper, 2019.
[59] NVIDIA, “CUDA GPUs and Compute Requirements,” NVIDIA Documentation, 2020.
[60] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” *Communications of the ACM*, 2008.
[61] F. Chollet, *Deep Learning with Python*, 2nd ed., Manning, 2021.
[62] Red Hat, “Linux Performance Tuning for Application Workloads,” Whitepaper, 2020.
[63] NIST, “Zero Trust Architecture,” NIST SP 800-207, 2020.
[64] M. Rash, *Linux iptables Pocket Reference*, O’Reilly, 2017.
[65] Red Hat, “Securing Linux Systems with firewalld”, RHCSA/RHCE Documentation, 2021.
[66] Forrester, “Zero Trust eXtended Framework”, Forrester Research, 2018.
[67] Tresys, “SELinux Policy Guide”, Tresys Security Research, 2017.
[68] Python Software Foundation, “Python Virtual Environments and Packaging”, PSF Documentation, 2023.
[69] AWS, “Bastion Host Security Best Practices”, AWS Security Whitepaper, 2020.
[70] V. Paxson, “Bro: A System for Detecting Network Intruders in Real-Time”, *Computer Networks*, 1999.
[71] I. Sysoev, “Nginx High-Performance Web Server”, Nginx Conference, 2014.
[72] Elastic.co, “The Elastic Stack: Elasticsearch, Logstash, Kibana”. Whitepaper, 2021.
[73] S. Ghemawat, “Elasticsearch Architecture”, *IEEE BigData*, 2017.
[74] Elastic, “Kibana: Data Visualization for Elasticsearch”, 2021.
[75] S. Chakrabarti, *Mining the Web: Discovering Knowledge from Hypertext Data*, *IEEE Press*, 2003.
[76] F. Haddadi and A. N. Zincir-Heywood, “Benchmarking the UNSW-NB15 Dataset”, *IEEE CNS*, 2019.
[77] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python”, *JMLR*, 2011
[78] R. Kohavi, “Cross-Validation and Model Selection”, *IJCAI*, 1995
[79] J. Bergstra and Y. Bengio, “Random Search for Hyperparameter Optimization”, *JMLR*, 2012.
[80] M. S. Keogh and E. Soria, “Model Deployment in Production Systems”, *IEEE Access*, 2020.

[81] VMware, “Best Practices for Virtual Machine CPU and Memory Allocation”, VMware Technical Guide, 2022.
[82] NIST, “Recommendation for Key Management – Part 1”, NIST SP 800-57, 2020.
[83] K. Reitz and T. Schlusser, *The Hitchhiker’s Guide to Python: Best Practices for Development*, O’Reilly Media, 2016.
[84] Docker Inc., “Docker Documentation: Volumes, Bind Mounts and Configuration Management”, Docker Technical Docs, 2023.
[85] NGINX, “NGINX Configuration Guide: Best Practices for Reverse Proxy and Static Content”, NGINX Technical Documentation, 2022
[86] V. Jacobson, C. Leres, and S. McCanne, “tcpdump—The Protocol Packet Capture and Dumper Program”, *Lawrence Berkeley Laboratory*, 1993.
[87] A. Luna, R. Torres and A. Guzmán, “Network Traffic Monitoring Using Packet Capture Techniques”, *IEEE LATINCOM*, 2017.
[88] A. Prakash and S. Singh, “Security Implications of Linux Capabilities in Containerized Environments”, *IEEE ICCC*, 2020.
[89] G. Lyon, “Nmap: A Network Mapper”, *IEEE Security & Privacy*, 2009.
[90] S. Christey, “Web Vulnerability Scanning Tools”, *IEEE S&P*, 2012.
[91] WPScan Team, “WPScan: WordPress Security Scanner”, *BlackHat EU*, 2015.
[92] S. Alqahtani & K. Alsubhi, “Survey on Honeypot-Based Intrusion Detection Systems”, *IEEE Access*, 2022.
[93] Z. Durumeric et al., “A Search Engine for Internet-Wide Scanning”, *ACM CCS*, 2014.
[94] S. Almotiri and M. Elleithy, “Malicious Bot Detection in Internet Traffic Using ML,” *IEEE LCN*, 2021.
[95] K. Thomas et al., “The Abuse of Online Bots and Automated Scanning,” *IEEE S&P Workshops*, 2015.
[96] N. Moustafa, “UNSW-NB15 dataset,” University of New South Wales (UNSW). Jun. 2021. [En línea]. Disponible: https://research.unsw.edu.au/projects/unsw-nb15-dataset

Anexos

Anexo I. Código Unsw_nb15_train_gridsearch.py

Código que realiza la normalización y transformación de datos, entrenamiento, prueba y creación del modelo, a su vez que crea las gráficas de la matriz de confusión

```
#!/usr/bin/env python3
import argparse, os, sys, json
import numpy as np
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV,StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix,
precision_score, recall_score, fbeta_score, make_scorer
from graficos_reporte import (
    plot_confusion_matrix_figure,
    plot_classification_report_bars
)

import joblib

RANDOM_STATE = 42
BRIDGE_NUM = ["dur","sbytes","dbytes","spkts","dpkts"]
BRIDGE_CAT = ["proto","state","service"]

def read_csv_safely(path: str) -> pd.DataFrame:
    try:
        return pd.read_csv(path)
    except Exception:
        return pd.read_csv(path, engine="python")

def clean_columns(df: pd.DataFrame) -> pd.DataFrame:
```

```
    df = df.copy()
    df.columns = (
        df.columns
          .str.strip()
          .str.lower()
          .str.replace(r"[^\w]+", "_", regex=True)
          .str.replace("__+", "_", regex=True)
          .str.strip("_")
    )
    # Common UNSW typos
    fixes = {
        "attack_cat_": "attack_cat",
        "label_": "label",
    }
    df.rename(columns=fixes, inplace=True)
    return df

def coerce_types(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()
    for c in ["proto","state","service","attack_cat"]:
        if c in df.columns:
            df[c] = df[c].astype(str).str.strip().str.lower().replace({"nan": np.nan, "-":
np.nan})
    for c in BRIDGE_NUM:
        if c in df.columns:
            df[c] = pd.to_numeric(df[c], errors="coerce")
    return df

def prepare_labels(df: pd.DataFrame, mode: str) -> pd.DataFrame:
    df = df.copy()
    if "attack_cat" in df.columns:
        df["attack_cat"] = df["attack_cat"].fillna("normal").replace({"-":
"normal"}).str.lower()
    if "label" in df.columns:
        df["label"] = pd.to_numeric(df["label"], errors="coerce").fillna(0).astype(int)
    else:
        df["label"] = (df.get("attack_cat","normal").astype(str).str.lower() !=
"normal").astype(int)
```

```
    if mode == "binary":
        df["y"] = df["label"]
    else:
        known =
["normal","fuzzers","analysis","backdoor","dos","exploits","generic","reconnaissanc
e","shellcode","worms"]
        ac = df.get("attack_cat", "normal").astype(str).str.lower().fillna("normal")
        df["y"] = ac.where(ac.isin(known), other="other")
    return df

def build_preproc():
    num_pipe = Pipeline([
        ("imp", SimpleImputer(strategy="median")),
        ("sc", StandardScaler(with_mean=False)),
    ])
    cat_pipe = Pipeline([
        ("imp", SimpleImputer(strategy="most_frequent")),
        ("oh", OneHotEncoder(handle_unknown="ignore")),
    ])
    pre = ColumnTransformer([
        ("num", num_pipe, BRIDGE_NUM),
        ("cat", cat_pipe, BRIDGE_CAT),
    ])
    return pre

def main():
    ap = argparse.ArgumentParser(description="Train UNSW-NB15 bridge model
compatible with Zeek conn.log")
    ap.add_argument("--train", required=True)
    ap.add_argument("--test", required=True)
    ap.add_argument("--mode", choices=["binary","multiclass"], default="binary")
    ap.add_argument("--out", default="unsw_bridge_model.joblib")
    args = ap.parse_args()

    train = coerce_types(clean_columns(read_csv_safely(args.train)))
    test  = coerce_types(clean_columns(read_csv_safely(args.test)))
```

```
    needed = set(BRIDGE_NUM + BRIDGE_CAT + ["attack_cat","label"])
    train = train[[c for c in train.columns if c in needed]].copy()
    test  = test [[c for c in test.columns  if c in needed]].copy()

    train = prepare_labels(train, args.mode)
    test  = prepare_labels(test, args.mode)

    pre = build_preproc()
    clf = Pipeline([
        ("pre", pre),
        ("rf",
RandomForestClassifier(class_weight='balanced_subsample',n_jobs=-1,n_esti
mators=800 , random_state=RANDOM_STATE
        ))
    ])

    Xtr = train.drop(columns=["y"])
    ytr = train["y"]
    Xte = test.drop(columns=["y"])
    yte = test["y"]


    scorers = {
        "recall_pos": make_scorer(recall_score, pos_label=1),
        "precision_pos": make_scorer(precision_score, pos_label=1),
        "fbeta_08": make_scorer(fbeta_score, beta=0.8, pos_label=1),
        "fbeta_10": make_scorer(fbeta_score, beta=1.0, pos_label=1),  # F1 clásico
    }

   #Parametros
    param_grid = {
        "rf__n_estimators": [800],
        "rf__max_depth": [30],
        "rf__min_samples_split": [2],
        "rf__min_samples_leaf": [1],
        "rf__max_features": ["log2"]
    }
```

```
   # CV estratificado para mantener proporciones
   cv = StratifiedKFold(n_splits=3, shuffle=True,
random_state=RANDOM_STATE)

   #Gridsearch con fbeta_08
   grid = GridSearchCV(
   clf,
   param_grid=param_grid,
   scoring=scorers,
   refit="fbeta_08",
   cv=cv,
   n_jobs=-1,
   verbose=2
   )

 #Obtener mejor recall
 def pick_threshold(y_true, y_proba, recall_min=0.95):
      thr_list = np.linspace(0.20, 0.80, 25)
      best = (0.5, 0.0, 0.0, 0.0)  # (t, precision, recall, f1)
      for t in thr_list:
         yp = (y_proba >= t).astype(int)
         pre = precision_score(y_true, yp, pos_label=1, zero_division=0)
         rec = recall_score(y_true, yp, pos_label=1)
         f1  = fbeta_score(y_true, yp, beta=1.0, pos_label=1)
         if rec >= recall_min:
            if pre > best[1]:  # max precision bajo la restricción de recall
               best = (t, pre, rec, f1)
      if best[1] == 0.0:  # si no alcanza recall_min, tomar mejor F1 global
         for t in thr_list:
            yp = (y_proba >= t).astype(int)
            f1  = fbeta_score(y_true, yp, beta=1.0, pos_label=1)
            if f1 > best[3]:
               pre = precision_score(y_true, yp, pos_label=1, zero_division=0)
               rec = recall_score(y_true, yp, pos_label=1)
               best = (t, pre, rec, f1)
      return best

   #Entrenar GridSearch
```

```
    grid.fit(Xtr, ytr)

    print("=== Mejor combinación (según fbeta_08) ===")
    print(grid.best_params_)
    print("Mejor score (fbeta_08):", grid.best_score_)

    best_model = grid.best_estimator_

    y_proba_te = best_model.predict_proba(Xte)[:, 1]
    t_opt, pre_v, rec_v, f1_v = pick_threshold(yte, y_proba_te, recall_min=0.95)

    yp = (y_proba_te >= t_opt).astype(int)

    #Creación de imagenes de gráficas
    class_names = [0,1]
    labels = [0,1]
    # Matriz de confusión cruda
    plot_confusion_matrix_figure(yte, yp, class_names,
                        normalize=False,
                        title="Matriz de confusión (modelo final)",
                        save_path="matriz_confusion.png")

# Matriz de confusión normalizada
    plot_confusion_matrix_figure(yte, yp, class_names,
                        normalize=True,
                        title="Matriz de confusión normalizada",
                        save_path="matriz_confusion_normalizada.png")

    # Reporte de clasificación por clase
    plot_classification_report_bars(yte, yp, class_names,
                          title="Reporte de clasificación",
                          save_path="reporte_clasificacion.png")

   #Reporte por consola
    print("\n=== Classification report (umbral óptimo) ===")
    print(classification_report(yte, yp, digits=4))

    print("\n=== Confusion matrix (counts) ===")
```

```
    print(confusion_matrix(yte, yp))

    #Obtener peso de atributos
    rf = grid.best_estimator_.named_steps['rf']
    importances = rf.feature_importances_
    feat_names =
grid.best_estimator_.named_steps['pre'].get_feature_names_out()

    df = pd.DataFrame({'feature': feat_names, 'importance': importances})
    print(df.sort_values('importance', ascending=False).head(15))

    # Save model + metadata
    meta = {
    "mode": args.mode,
    "bridge_num": BRIDGE_NUM,
    "bridge_cat": BRIDGE_CAT,
    "best_params": grid.best_params_,
    "threshold": float(t_opt),
    "version": 2,
    }
    joblib.dump({"pipeline": best_model, "meta": meta}, args.out)
    with open(args.out + ".meta.json","w") as f:
        json.dump(meta, f, indent=2)
    print(f"[OK] Saved model to {args.out} and metadata to {args.out}.meta.json")

if __name__ == "__main__":
    main()
```

Anexo II. Código Zeek_predict_bridge.py

Descripción: Código que aplica inferencia sobre archivo de extensión .log proveniente de Zeek utilizando el modelo para realizar predicciones sobre dicho conjunto de registros y devuelve un archivo csv con los resultados.

```
```

```
#!/usr/bin/env python3
import argparse, json, sys, os, re
import joblib
import pandas as pd
import numpy as np
from datetime import datetime

BRIDGE_NUM = ["dur","sbytes","dbytes","spkts","dpkts"]
BRIDGE_CAT = ["proto","state","service"]

def load_model(path):
    obj = joblib.load(path)
    return obj["pipeline"], obj.get("meta", {})

def parse_zeek_json(path):
    # Zeek JSON: one JSON object per line
    with open(path, "r", encoding="utf-8") as f:
        for line in f:
            line = line.strip()
            if not line:
                continue
            if not line.startswith("{"):
                continue
            try:
                yield json.loads(line)
            except Exception:
                continue

#Función que mapea atributos de conn.log con los de unsw
def zeek_to_bridge(rec: dict):
    out = {}
    out["dur"] = rec.get("duration")
    out["sbytes"] = rec.get("orig_bytes")
    out["dbytes"] = rec.get("resp_bytes")
    out["spkts"] = rec.get("orig_pkts")
    out["dpkts"] = rec.get("resp_pkts")
    out["proto"] = (rec.get("proto") or "").lower() or None
    cs = rec.get("conn_state")
```

```
    out["state"] = cs.lower() if isinstance(cs, str) else None
    svc = rec.get("service")
    # Zeek uses "-" for unknown service
    if svc == "-":
        svc = None
    out["service"] = (svc or "").lower() or None
    return out

def batch_dataframe(json_path):
    rows = []
    meta = []
    for rec in parse_zeek_json(json_path):
        bridge = zeek_to_bridge(rec)
        rows.append(bridge)
        meta.append({
            "ts": rec.get("ts"),
            "uid": rec.get("uid"),
            "src": rec.get("id.orig_h"),
            "dst": rec.get("id.resp_h"),
            "sport": rec.get("id.orig_p"),
            "dport": rec.get("id.resp_p"),
        })
    if not rows:
        return pd.DataFrame(), pd.DataFrame()
    X = pd.DataFrame(rows)
    M = pd.DataFrame(meta)
    # coerce numerics
    for c in BRIDGE_NUM:
        if c in X.columns:
            X[c] = pd.to_numeric(X[c], errors="coerce")
    for c in BRIDGE_CAT:
        if c in X.columns:
            X[c] = X[c].astype(object)
    return X, M

def main():
    ap = argparse.ArgumentParser(description="Predict malice from Zeek
conn.log (JSON) using UNSW bridge model")
```

```
    ap.add_argument("--model", required=True, help="Path to
unsw_bridge_model.joblib")
    ap.add_argument("--conn_json", required=True, help="Path to Zeek conn.log
in JSON")
    ap.add_argument("--out", help="Optional CSV to write predictions")
    args = ap.parse_args()

    pipe, meta = load_model(args.model)
    X, M = batch_dataframe(args.conn_json)
    if X.empty:
        print("[WARN] No Zeek records parsed.")
        sys.exit(0)

    preds = pipe.predict(X)
    try:
        proba = pipe.predict_proba(X)[:,1]
    except Exception:
        proba = None

    out_df = M.copy()
    out_df["pred"] = preds
    if proba is not None:
        out_df["score"] = proba

    if args.out:
        out_df.to_csv(args.out, index=False)
        print(f"[OK] Wrote predictions to {args.out}")
    else:
        show = out_df.sort_values("score", ascending=False) if "score" in
out_df.columns else out_df
        print(show.head(20).to_string(index=False))

if __name__ == "__main__":
    main()
```

Anexo III. Código Zeek_tail_predict.py

Descripción: Código que aplica inferencia sobre los logs de red de Zeek en tiempo real y devuelve alertas al detectar tráfico de red catalogado como malicioso.

```
#!/usr/bin/env python3
import argparse, time, os, json, joblib, pandas as pd, numpy as np
from zeek_predict_bridge import zeek_to_bridge, BRIDGE_NUM, BRIDGE_CAT


def tail_f(path):
    with open(path, "r", encoding="utf-8") as f:
        f.seek(0, os.SEEK_END)
        while True:
            line = f.readline()
            if not line:
                time.sleep(0.5)
                continue
            yield line

def main():
    #Menu
    ap = argparse.ArgumentParser(description="Tail Zeek conn.log (JSON) and
live-predict")
    ap.add_argument("--model", required=True)
    ap.add_argument("--conn_json", required=True)
    ap.add_argument("--threshold", type=float, default=0.8, help="Alert threshold on
probability (binary mode)")
    args = ap.parse_args()
    obj = joblib.load(args.model)
    pipe = obj["pipeline"]

    buf = []
    for line in tail_f(args.conn_json):
        try:
            rec = json.loads(line)
        except Exception:
```

```
            continue
        bridge = zeek_to_bridge(rec)
        X = pd.DataFrame([bridge])
        for c in BRIDGE_NUM:
            X[c] = pd.to_numeric(X[c], errors="coerce")
        for c in BRIDGE_CAT:
            X[c] = X[c].astype(object)
        try:
            proba = pipe.predict_proba(X)[:,1][0]
        except Exception:
            pred = pipe.predict(X)[0]
            proba = None
        uid = rec.get("uid")
        src = rec.get("id.orig_h"),
        dst = rec.get("id.resp_h")
        if proba is not None and proba >= args.threshold:
            print(json.dumps({"uid": uid, "src": src, "dst": dst, "score": proba, "alert":
True}))
        elif proba is None and pred == 1:
            print(json.dumps({"uid": uid, "src": src, "dst": dst, "pred": int(pred), "alert":
True}))


if __name__ == "__main__":
    main()
```

Anexo IV. Archivo de implementación Docker-compose.yml

Descripción: Archivo que contiene la configuración de los servicios del proyecto con sus configuraciones de servicio correspondientes.

```
services:
  nginx:
```

```
  image: nginx:alpine
  container_name: honeypot_nginx
  ports:
   - "80:80"
  volumes:
   - ./sites:/usr/share/nginx/sites:ro
   - ./nginx.conf:/etc/nginx/nginx.conf:ro
  networks:
   zeeknet:
    ipv4_address: 172.20.0.10
  restart: unless-stopped

zeek:
  image: blacktop/zeek
  container_name: zeek_monitor
  command: ["-i", "ens18","LogAscii::use_json=T"]
  network_mode: "host"
  cap_add:
   - NET_RAW
   - NET_ADMIN
  volumes:
   - ./pcap:/pcap:rw
  restart: unless-stopped

elasticsearch:
  image: docker.elastic.co/elasticsearch/elasticsearch:7.17.10
  container_name: es
  environment:
   - discovery.type=single-node
   - xpack.security.enabled=false
  ulimits:
   memlock:
    soft: -1
    hard: -1
  mem_limit: 1g
  ports:
   - "9200:9200"
```

```
  networks:
    elknet:
      ipv4_address: 172.30.0.10
  restart: unless-stopped

logstash:
  image: docker.elastic.co/logstash/logstash:7.17.10
  container_name: logstash
  volumes:
    - ./logstash.conf:/usr/share/logstash/pipeline/logstash.conf:ro
  ports:
    - "5044:5044"
  depends_on:
    - elasticsearch
  networks:
    elknet:
      ipv4_address: 172.30.0.11
  restart: unless-stopped

kibana:
  image: docker.elastic.co/kibana/kibana:7.17.10
  container_name: kibana
  environment:
    ELASTICSEARCH_HOSTS: http://elasticsearch:9200
  ports:
    - "127.0.0.1:5601:5601"
  depends_on:
    - elasticsearch
  networks:
    elknet:
      ipv4_address: 172.30.0.12
  restart: unless-stopped

filebeat:
  image: docker.elastic.co/beats/filebeat:7.17.10
  container_name: filebeat
  user: root
```

```
    volumes:
      - ./filebeat.yml:/usr/share/filebeat/filebeat.yml:ro
      - ./pcap:/logs:ro
    depends_on:
      - logstash
    networks:
      - elknet
    restart: unless-stopped


networks:
  zeeknet:
    driver: bridge
    ipam:
      config:
        - subnet: 172.20.0.0/24
  elknet:
    driver: bridge
    ipam:
      config:
        - subnet: 172.30.0.0/24
```

Anexo V. Archivo de configuración Logstash.conf

Descripción: Archivo que contiene la configuración del servicio de encolamiento de datos.

```
input {
  beats {
    port => 5044
  }
}
filter {}
output {
  elasticsearch {
```

```
    hosts => ["http://elasticsearch:9200"]
    index => "zeek-logs-%{+YYYY.MM.dd}"
  }
}
```

Anexo VI. Archivo de configuración Nginx.conf

Descripción: Archivo que contiene la configuración del servicio de proxy reverso del proyecto.

```
events {}
http {
    server {
        listen 80;
        server_name avion.emer;
        root /usr/share/nginx/sites/avion;
        index index.html;

        ## Servir assets estáticos con cache
        location ~* \.(?:ico|css|js|gif|jpe?g|png|webp|woff2?|ttf|svg|eot|json|php)$ {
            expires 6M;
            access_log off;
            add_header Cache-Control "public";
            try_files $uri =404;
            }

        # fallback: si no existe archivo → index.html
        location / {
            try_files $uri $uri/ /index.html;
        }
    }

    server {
        listen 80;
        server_name cume.emer;
```

```
    root /usr/share/nginx/sites/cume;
    index index.html;

    ## Servir assets estáticos con cache
    location ~* \.(?:ico|css|js|gif|jpe?g|png|webp|woff2?|ttf|svg|eot|json|php)$ {
        expires 6M;
        access_log off;
        add_header Cache-Control "public";
        try_files $uri =404;
        }

    # fallback: si no existe archivo → index.html
    location / {
        try_files $uri $uri/ /index.html;
    }
}

server {
    listen 80;
    server_name cursos.emer;
    root /usr/share/nginx/sites/cursos;
    index index.html;

    ## Servir assets estáticos con cache
    location ~* \.(?:ico|css|js|gif|jpe?g|png|webp|woff2?|ttf|svg|eot|json|php)$ {
        expires 6M;
        access_log off;
        add_header Cache-Control "public";
        try_files $uri =404;
        }

    # fallback: si no existe archivo → index.html
    location / {
        try_files $uri $uri/ /index.html;
    }
}
server {
```

```
    listen 80;
    server_name emergencias.emer;
    root /usr/share/nginx/sites/emergencias;
    index index.html;

    ## Servir assets estáticos con cache
    location ~* \.(?:ico|css|js|gif|jpe?g|png|webp|woff2?|ttf|svg|eot|json|php)$ {
        expires 6M;
        access_log off;
        add_header Cache-Control "public";
        try_files $uri =404;
        }

    # fallback: si no existe archivo → index.html
    location / {
        try_files $uri $uri/ /index.html;
    }
}
server {
    listen 80;
    server_name landing.emer;
    root /usr/share/nginx/sites/landing;
    index index.html;

    ## Servir assets estáticos con cache
    location ~* \.(?:ico|css|js|gif|jpe?g|png|webp|woff2?|ttf|svg|eot|json)$ {
        expires 6M;
        access_log off;
        add_header Cache-Control "public";
        try_files $uri =404;
        }

    # fallback: si no existe archivo → index.html
    location / {
        try_files $uri $uri/ /index.html;
    }
}
```

```
}
```

Anexo VII. Código urlCloner.py

Descripción: Código encargado de la clonación del sitio objetivo especificado en el archivo mismo.

```
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin, urlparse
import os
import sys
import signal

def signal_handler(sig, frame):
    print('You pressed Ctrl+C!')
    sys.exit(0)

signal.signal(signal.SIGINT, signal_handler)



# Configuración
base_url = "sitio"  # Cambia al sitio objetivo
output_dir = "carpeta"
visited_urls = set()
max_depth = 5  # Profundidad máxima
os.makedirs(output_dir, exist_ok=True)

def save_file(url, content):
    parsed = urlparse(url)
    path = parsed.path.lstrip('/')
    if not path or path.endswith('/'):
        path += 'index.html'
    local_path = os.path.join(output_dir, path)
    os.makedirs(os.path.dirname(local_path), exist_ok=True)
```

```
    with open(local_path, 'wb') as f:
        f.write(content)
    print(f"[+] Guardado: {local_path}")

def crawl(url, depth=0):
    if url in visited_urls or depth > max_depth:
        return
    visited_urls.add(url)

    try:
        response = requests.get(url)
        content_type = response.headers.get('Content-Type', '')
        if 'text/html' in content_type:
            save_file(url, response.content)
            soup = BeautifulSoup(response.text, 'html.parser')

            # Descargar imágenes, CSS, JS
            for tag in soup.find_all(['img', 'script', 'link']):
                attr = 'src' if tag.name in ['img', 'script'] else 'href'
                resource = tag.get(attr)
                if resource:
                    resource_url = urljoin(url, resource)
                    parsed_resource = urlparse(resource_url)
                    if parsed_resource.netloc == urlparse(base_url).netloc:
                        try:
                            r = requests.get(resource_url)
                            save_file(resource_url, r.content)
                        except:
                            print(f"[!] Falló la descarga de {resource_url}")

            # Seguir links internos
            for a in soup.find_all('a', href=True):
                link = urljoin(url, a['href'])
                parsed_link = urlparse(link)
                if parsed_link.netloc == urlparse(base_url).netloc:
                    crawl(link, depth + 1)
        else:
```

```
            save_file(url, response.content)
        except Exception as e:
            print(f"[!] Error accediendo a {url}: {e}")

# Iniciar el clonado

crawl(base_url)
```

Anexo VIII. Archivo de configuración filebeat.yml

Descripción: Archivo encargado de la recolección de registros de red y su posterior envío a logstash.

```
# filebeat.yml
filebeat.inputs:
  - type: log
    enabled: true
    paths:
      - /logs/conn.log
      - /logs/http.log
    multiline.pattern: '^\d{4}-\d{2}-\d{2}'
    multiline.negate: true
    multiline.match: after

output.logstash:
  hosts: ["logstash:5044"]

setup.kibana.enabled: false
monitoring.enabled: false
```

Anexo IX. Lista de atributos

Descripción: Lista de los atributos del dataset junto a una breve descripción de su finalidad.

1. id

Identificador único del flujo de red generado por el dataset.

2. dur

Duración total del flujo (segundos).

3. proto

Protocolo de transporte utilizado (TCP, UDP, ICMP, etc.).

4. service

Servicio de aplicación asociado al puerto destino (HTTP, FTP, SSH, etc.).

5. state

Estado de la conexión según el firewall (ESTABLISHED, FIN, REJ, etc.).

6. spkts

Cantidad de paquetes enviados (s) en el flujo.

7. dpkts

Cantidad de paquetes recibidos (d) en el flujo.

8. sbytes

Bytes enviados (s) en el flujo.

9. dbytes

Bytes recibidos (d) en el flujo.

10. rate

Tasa de transferencia total del flujo (bytes por segundo).

11. sttl

Valores TTL de paquetes enviados (s).

12. dttl

Valores TTL de paquetes recibidos (d).

13. sload

Carga de tráfico enviada en bits por segundo.

14. dload

Carga de tráfico recibida en bits por segundo.

15. sloss

Cantidad de paquetes perdidos enviados.

16. dloss

Cantidad de paquetes perdidos recibidos.

17. sinpkt

Tiempo promedio entre paquetes enviados.

18. dinpkt

Tiempo promedio entre paquetes recibidos.

19. sjit

Jitter (variación del tiempo entre paquetes) enviado.

20. djit

Jitter (variación del tiempo entre paquetes) recibido.

21. swin

Tamaño de ventana TCP del emisor.

22. stcpb

Número de secuencia TCP inicial enviado.

23. dtcpb

Número de secuencia TCP inicial recibido.

24. dwin

Tamaño de ventana TCP del receptor.

25. tcprtt

Tiempo total de ida y vuelta (RTT) del flujo.

26. synack

Tiempo entre paquetes SYN y ACK (latencia de handshake).

27. ackdat

Tiempo entre ACK y el primer dato transmitido.

28. smean

Tamaño promedio de paquetes enviados.

29. dmean

Tamaño promedio de paquetes recibidos.

30. trans_depth

Profundidad de transacciones HTTP dentro de la sesión.

31. response_body_len

Tamaño del cuerpo de respuesta HTTP.

32. ct_srv_src

Cantidad de conexiones desde la misma IP origen al mismo servicio destino.

33. ct_state_ttl

Conteo de flujos con igual par (state, TTL).

34. ct_dst_ltm

Cantidad de conexiones recientes hacia el mismo destino.

35. ct_src_dport_ltm

Cantidad de conexiones recientes desde la misma fuente al mismo puerto destino.

36. ct_dst_sport_ltm

Cantidad de conexiones recientes al mismo destino desde el mismo puerto origen.

37. ct_dst_src_ltm

Cantidad total de conexiones previas entre este par IP origen–destino.

38. is_ftp_login

Indica si hubo un login FTP exitoso (0/1).

39. ct_ftp_cmd

Número de comandos FTP en la sesión.

40. ct_flw_http_mthd

Cantidad de métodos HTTP detectados (GET, POST, etc.).

41. ct_src_ltm

Número de conexiones recientes desde la misma IP origen.

42. ct_srv_dst

Conexiones que llegaron al mismo servicio desde distintas IP.

43. is_sm_ips_ports

Indica si el par (IP origen, IP destino, puertos) aparece repetido (0/1).

44. attack_cat

Categoría de ataque (DoS, Reconnaissance, Fuzzers, etc.).

45. label

Etiqueta binaria: 0 (benigno), 1 (malicioso).

Anexo X. Código graficos_reporte.py

Descripción: Código encargado de la realización de los gráficos de la matriz de confusión.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

def plot_confusion_matrix_figure(y_true, y_pred, class_names, normalize=False,
                                 title="Matriz de confusión",
                                 save_path=None, dpi=300):
    """
    Dibuja una matriz de confusión con anotaciones.
    - normalize=True para porcentajes por fila.
    - save_path para guardar PNG (alta calidad).
    """
    # Orden consistente según class_names
    cm = confusion_matrix(y_true, y_pred, labels=class_names)
```

```
    if normalize:
        with np.errstate(all='ignore'):
            cm = cm.astype(float) / cm.sum(axis=1, keepdims=True)
            cm = np.nan_to_num(cm)

    fig = plt.figure(figsize=(6, 5), dpi=dpi)
    ax = fig.add_subplot(111)
    im = ax.imshow(cm, interpolation='nearest',cmap="YlOrRd")  # colormap por
defecto

    ax.set_title(title, pad=10, fontsize=11)
    ax.set_xticks(np.arange(len(class_names)))
    ax.set_yticks(np.arange(len(class_names)))
    ax.set_xticklabels(class_names, rotation=45, ha='right')
    ax.set_yticklabels(class_names)
    ax.set_xlabel("Predicción", labelpad=8)
    ax.set_ylabel("Etiqueta real", labelpad=8)

    fmt = ".2f" if normalize else "d"
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center", fontsize=9)

    # Barra lateral (opcional, se ve bien sin tocar colores)
    cbar = fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
    cbar.ax.tick_params(labelsize=8)

    fig.tight_layout()
    if save_path:
        fig.savefig(save_path, bbox_inches="tight")
    return fig


def plot_classification_report_bars(y_true, y_pred, labels, display_names=None,
                                    title="Reporte de clasificación",
                                    save_path=None, dpi=300):
    from sklearn.metrics import classification_report
```

```
import numpy as np, matplotlib.pyplot as plt

if display_names is None:
    display_names = [str(l) for l in labels]

rep = classification_report(y_true, y_pred, labels=labels,
                            output_dict=True, zero_division=0)

precisions = [rep[str(lbl)]['precision'] if str(lbl) in rep else 0.0 for lbl in labels]
recalls    = [rep[str(lbl)]['recall']    if str(lbl) in rep else 0.0 for lbl in labels]
f1s        = [rep[str(lbl)]['f1-score']  if str(lbl) in rep else 0.0 for lbl in labels]
supports   = [int(rep[str(lbl)]['support']) if str(lbl) in rep else 0 for lbl in labels]

x = np.arange(len(labels)); width = 0.25

fig = plt.figure(figsize=(8, 5), dpi=dpi)
ax = fig.add_subplot(111)

b1 = ax.bar(x - width, precisions, width, label='Precision')
b2 = ax.bar(x,         recalls,   width, label='Recall')
b3 = ax.bar(x + width, f1s,       width, label='F1-score')

ax.set_title(title, pad=10, fontsize=11)
ax.set_ylabel('Puntuación')
ax.set_xticks(x)
ax.set_xticklabels(display_names, rotation=0)
ax.set_ylim(0, 1.05)
ax.grid(axis='y', linewidth=0.5)
ax.margins(y=0.03)  # un pelín de aire

# Etiquetas encima de cada barra
for bars in (b1, b2, b3):
    ax.bar_label(bars, fmt='%.2f', padding=2, fontsize=9)

# Soporte: ponelo debajo del tick con offset (no dentro del gráfico)
for xi, s in zip(x, supports):
    ax.annotate(f"n={s}", xy=(xi, 0), xytext=(0, -20),
                textcoords='offset points', ha='center', va='top',
```

```
                fontsize=9, rotation=90, clip_on=False)

    # Leyenda afuera (no tapa barras)
    ax.legend(loc='upper left', bbox_to_anchor=(1.02, 1.00), borderaxespad=0.,
frameon=False, ncol=1)

    fig.tight_layout()
    if save_path:
        fig.savefig(save_path, bbox_inches='tight', pad_inches=0.2)
    return fig
```

.