TRABAJO FIN DE GRADO

Curso 2025/26

DETECCIÓN DE AMENAZAS EN TIEMPO REAL MEDIANTE INTELIGENCIA ARTIFICIAL

Alumno:

PABLO ENRIQUE GUNTÍN GARRIDO

Tutores:

DIEGO RAMÍREZ JIMÉNEZ

ROBERTO MATHEUS PINHEIRO PEREIRA

UNIVERSIDAD INTERNACIONAL ISABEL I DE CASTILLA

FACULTAD DE CIENCIAS Y TECNOLOGÍA

GRADO EN INGENIERÍA INFORMÁTICA

Pablo Enrique Guntín Garrido

Trabajo Fin de Grado

Curso 2025/26

Detección de Amenazas en Tiempo Real Mediante Inteligencia artificial

Tutores: Diego Ramírez Jiménez

Roberto Matheus Pinheiro Pereira

Universidad Internacional Isabel I de Castilla

Facultad de Ciencias y Tecnología

Grado en Ingeniería Informática

Universidad Isabel I

Resumen

Este Trabajo Fin de Grado aborda el diseño, implementación y evaluación de un sistema de detección de amenazas en tiempo real basado en técnicas de inteligencia artificial e integrado en un enfoque DevSecOps. El objetivo es analizar la viabilidad de combinar modelos de aprendizaje automático con procesos de automatización, monitorización continua y seguridad integrada para mejorar la detección de intrusiones en entornos de red modernos.

En primer lugar, se realiza un estudio del estado del arte en ciberseguridad, sistemas de detección de intrusiones y aplicaciones de la inteligencia artificial, identificando las limitaciones de los enfoques tradicionales basados en firmas. A partir de este análisis, se diseña una arquitectura modular que integra la captura, el preprocesamiento de datos, el análisis mediante modelos de aprendizaje automático y la generación automatizada de alertas dentro de un pipeline de integración y despliegue continuo.

El sistema se implementa en un entorno experimental basado en contenedores Docker y se evalúa utilizando los conjuntos de datos CICIDS2017 y UNSW-NB15. El modelo Random Forest obtiene resultados satisfactorios según métricas estándar de evaluación. Finalmente, se analizan las limitaciones del sistema y se proponen líneas futuras orientadas a la detección en tiempo real y al uso de modelos más avanzados.

Palabras clave: ciberseguridad, detección de intrusiones, inteligencia artificial, aprendizaje automatizado, DevSecOps.

Abstract

This Final Degree Project addresses the design, implementation, and evaluation of a real-time threat detection system based on artificial intelligence techniques and integrated within a DevSecOps approach. The objective is to analyze the feasibility of combining machine learning models with automation processes, continuous monitoring, and integrated security to improve intrusion detection in modern network environments.

First, a state-of-the-art study in cybersecurity, intrusion detection systems, and artificial intelligence applications is conducted, identifying the limitations of traditional signature-based approaches. Based on this analysis, a modular architecture is designed that integrates data acquisition, preprocessing, analysis using machine learning models, and automated alert generation within a continuous integration and deployment pipeline.

The system is implemented in an experimental environment based on Docker containers and evaluated using the CICIDS2017 and UNSW-NB15 datasets. The Random Forest model achieves satisfactory results according to standard evaluation metrics. Finally, the limitations of the system are analyzed, and future research directions are proposed, focusing on real-time detection and the use of more advanced models.

Keywords: Cybersecurity, intrusion detection, artificial intelligence, machine learning, DevSecOps.

Índice

Índice de figuras

Índice de tablas

Capítulo 1: Introducción

1.1. Contexto general de la ciberseguridad y su evolución

En las últimas décadas, la ciberseguridad ha pasado de ser una preocupación técnica limitada a los departamentos informáticos a convertirse en un pilar estratégico esencial para la continuidad operativa, la protección de los activos digitales y la confianza institucional. El avance de la transformación digital, impulsado por la masificación de Internet, la adopción de la computación en la nube, la expansión del Internet de las Cosas (IoT) y la incorporación progresiva de la inteligencia artificial (IA) en los procesos organizativos, como anunciaba, ha generado un entorno interconectado con una superficie de ataque cada vez más amplia y compleja [1].

Las organizaciones modernas, como pueden ser empresas, instituciones públicas o usuarios domésticos, dependen de infraestructuras digitales distribuidas y de sistemas en red para operar. Esta interconexión, a pesar de aportar eficiencia y escalabilidad, también introduce vulnerabilidades que pueden ser explotadas de manera remota. Como consecuencia de esto, la ciberseguridad ha dejado de ser un componente meramente técnico para integrarse como una función transversal dentro de la gestión global del riesgo organizativo [2].

Históricamente, la protección informática se centraba en el perímetro de seguridad, basada en mecanismos como podían ser antivirus, cortafuegos o filtros de contenido. Sin embargo, la sofisticación de las amenazas contemporáneas ha impulsado un cambio hacia un modelo de defensa en profundidad, donde la prevención, la detección temprana y la respuesta automatizada son elementos clave. Los ataques actuales, como el *ransomware*, la exfiltración de datos o la explotación de vulnerabilidades en infraestructuras críticas, demuestran un alto grado de planificación, automatización y persistencia. Esto todo obliga a las organizaciones a adoptar estrategias más proactivas y adaptativas [3].

Entre los casos más notorios de los últimos años se encuentran ataques masivos que afectaron tanto a corporaciones privadas como a organismos públicos, provocando interrupciones operativas y pérdidas económicas significativas. Este panorama evidencia que la seguridad no puede abordarse de manera reactiva, si no que requiere sistemas capaces de aprender y anticiparse a nuevas amenazas en un entorno digital dinámico y globalizado [1].

1.1.1. Tendencias y desafíos contemporáneos

En la actualidad, la ciberseguridad se caracteriza por la profesionalización del ciberdelito y el crecimiento de economías ilícitas en línea. Modelos como el *cybercrime-as-a-service*¸CaaS (Ciberdelito como Servicio), ampliamente documentados [4], han facilitado el acceso a kits de ataque, redes de bots y malware sofisticado, reduciendo las barreras de entrada para potenciales atacantes. Este fenómeno ha derivado en una democratización del delito informático, en la que incluso actores con conocimientos técnicos limitados pueden ejecutar ataques de alto impacto mediante plataformas automatizadas.

A su vez, la creciente dependencia de servicios en la nube, el auge del trabajo remoto y la movilidad digital han multiplicado los vectores de riesgo, introduciendo nuevos desafíos en la gestión de la identidad, el acceso y la integridad de los datos. La detección temprana de intrusiones debe enfrentarse hoy a la complejidad de procesar volúmenes masivos de información generada por sistemas, dispositivos y usuarios, donde los comportamientos maliciosos pueden quedar ocultos entre millones de eventos legítimos [3], [5].

Ante esta situación, la capacidad de identificar anomalías en tiempo real y de forma autónoma se ha vuelto un elemento esencial para la resiliencia digital. Los enfoques tradicionales, basados exclusivamente en reglas o firmas, resultan insuficientes frente a ataques emergentes o desconocidos. Por ello, la incorporación de técnicas de aprendizaje automático y análisis inteligente de datos representa una evolución natural hacia un modelo de seguridad más dinámico, predictivo y autorregulado [1], [4], [6]

1.2. Limitaciones de los métodos tradicionales de detección

Los sistemas tradicionales de detección de intrusiones (IDS, *Intrusion Detection Systems*) y de prevención de intrusiones (IPS, *Intrusion Prevention Systems*) se han constituido a lo largo de la historia como los pilares fundamentales en la defensa de las infraestructuras digitales[7]. Estos mecanismos, están generalmente basados en firmas o reglas heurísticas, operan mediante la comparación del tráfico de red o de los registros de actividad con bases de datos que contienen patrones de comportamiento malicioso previamente identificados[3], [8]

Este enfoque, aunque eficaz frente a amenazas conocidas y catalogadas, presenta limitaciones estructurales frente a ataques nuevos, desconocidos o de tipo *zero-day*, es decir, aquellos que explotan vulnerabilidades aún no documentadas o sin un parche de seguridad descubierto. La dependencia de bases de datos actualizadas manualmente implica un desfase entre la aparición de la amenaza y su detección efectiva, lo que deja un margen de exposición considerable para los sistemas afectados [9].

En primer lugar, los sistemas basados en firmas requieren un mantenimiento continuo y la intervención constante de especialistas en seguridad para la actualización de sus repositorios de patrones. Esta dependencia humana ralentiza la capacidad de respuesta ante incidentes emergentes, especialmente cuando las amenazas evolucionan con gran rapidez o utilizan técnicas de ofuscación (modificación de código para dificultar la lectura, renombrado de variables, eliminación de comentarios etc) y cifrado para modificar su apariencia. En segundo lugar, los métodos heurísticos o de detección basada en reglas, auqnue permiten identificar comportamientos atípicos, tienden a generar un número elevado de falsos positivos, lo que provoca una sobrecarga de alertas en los Centros de Operaciones de Seguridad (SOC) y reduce la eficiencia del personal analista [1].

Del mismo modo, la escalabilidad de estos sistemas tradicionales resulta limitada frente al crecimiento exponencial del tráfico de red y la diversidad de dispositivos conectados. En entornos corporativos modernos, donde conviven infraestructuras híbridas y servicios en la nube, los IDS e IPS convencionales se enfrentan a la dificultad de procesar volúmenes masivos de datos en tiempo real sin degradar su rendimiento. Además, el aumento de la automatización de los ciberataques y la aplicación de técnicas avanzadas de evasión, como el cifrado de payloads o la fragmentación de paquetes, han disminuido drásticamente la eficacia de los mecanismos tradicionales [2].

Como consecuencia, los métodos clásicos de detección carecen de la capacidad adaptativa y predictiva necesaria para hacer frente a un panorama de amenazas cada vez más dinámico y cambiante. Frente a este escenario, la incorporación de modelos inteligentes, basados en aprendizaje automático (Machine Learning) y aprendizaje profundo (Deep Learning), se presenta como una alternativa más robusta y flexible. Estos modelos no dependen exclusivamente del conocimiento previo de los ataques, sino que pueden aprender patrones de comportamiento, generalizar ante situaciones desconocidas y adaptarse progresivamente a nuevas condiciones del entorno. Este cambio de paradigma marca el paso de una seguridad reactiva a una seguridad proactiva y adaptativa, acorde con las necesidades de los sistemas digitales actuales [3], [5], [10].

1.3. Rol de la inteligencia artificial en la ciberdefensa moderna

La inteligencia artificial se ha consolidado en los últimos años como un eje transformador dentro de la ciberdefensa moderna, impulsando un cambio de paradigma hacia modelos de seguridad más autónomos, adaptativos y predictivos, tal y como señalan diversos estudios en el ámbito de la seguridad informática[10]. Su contribución resulta especialmente relevante en un contexto caracterizado por la creciente sofisticación de las amenazas, la expansión de las superficies de ataque y el volumen masivo de datos que deben analizarse para detectar comportamientos anómalos en tiempo real. Frente a este panorama, las técnicas basadas en firmas o reglas estáticas muestran limitaciones evidentes, especialmente ante ataques desconocidos o de tipo *zero-day*, lo que ha generado una transición natural hacia mecanismos inteligentes capaces de aprender y generalizar a partir de los datos [10].

La incorporación de algoritmos de aprendizaje automático y aprendizaje profundo ha permitido que los sistemas de seguridad puedan identificar patrones sutiles y no triviales en flujos de red complejos, extrayendo características relevantes que no serían detectables mediante métodos tradicionales, como se recoge en distintos títulos [3], [5]. A diferencia de los modelos estáticos, la IA ofrece una capacidad de análisis adaptativo que facilita la identificación temprana de incidentes, incluso en entornos altamente distribuidos, dinámicos y sometidos a constantes variaciones. En este sentido, la IA no solo complementa los mecanismos convencionales de defensa, sino que los potencia al dotarlos de autonomía, velocidad de respuesta y capacidad para anticiparse a comportamientos maliciosos emergentes [10].

Desde una perspectiva técnica, las aplicaciones de la IA en ciberseguridad pueden agruparse en tres ejes principales. En primer lugar, el aprendizaje supervisado permite entrenar modelos con ejemplos etiquetados de tráfico legítimo y malicioso, dando lugar a clasificadores capaces de distinguir entre ambos con altos niveles de precisión. Algoritmos como Random Forest, *Support Vector Machines* o las Redes Neuronales Artificiales se han posicionado como referentes en la identificación de intrusiones y en la categorización de ataques específicos, como se reflejan en multiples trabajos académicos en este ámbito [2], [3]. En segundo lugar, los enfoques de aprendizaje no supervisado y detección de anomalías cobran especial relevancia en escenarios donde no existen etiquetas confiables. Métodos como *clustering*, *autoencoders* o modelos basados en densidad permiten descubrir comportamientos inusuales que podrían corresponderse con actividades maliciosas previamente desconocidas. Finalmente, el aprendizaje amplía aún más estas capacidades al capturar patrones complejos y relaciones temporales en grandes volúmenes de datos, consolidándose como una herramienta idónea para el análisis de tráfico en tiempo real [2], [3].

La combinación de estas técnicas ha permitido mejorar significativamente la precisión de los sistemas de detección, reducir la tasa de falsos positivos y automatizar tareas que antes requerían una supervisión intensiva por parte de analistas humanos. Además, el uso de IA facilita la correlación de eventos distribuidos, la priorización automática de alertas y la generación de respuestas tempranas, contribuyendo a la reducción del tiempo de exposición ante ataques. Estas capacidades resultan especialmente valiosas en los Centros de Operaciones de Seguridad (SOC), donde la carga de trabajo y la complejidad de los incidentes hacen indispensable el apoyo de herramientas basadas en inteligencia artificial [5], [9].

No obstante, la integración de IA en la ciberdefensa plantea también desafíos significativos. La necesidad de *datasets* representativos y actualizados, la dificultad de interpretar decisiones emitidas por modelos de tipo *black box*, o la exposición a técnicas de ataque adversarial son factores que deben ser gestionados cuidadosamente para garantizar la fiabilidad de los sistemas. Asimismo, su despliegue en entornos de producción requiere combinar el uso de IA con metodologías robustas de ingeniería de software y seguridad continua, especialmente en arquitecturas críticas donde la latencia y la disponibilidad son determinantes [8].

La inteligencia artificial no solo representa una mejora incremental respecto a los enfoques tradicionales, sino que constituye un pilar estratégico de la ciberdefensa contemporánea. Su capacidad para analizar datos a gran escala, adaptarse a contextos cambiantes y detectar patrones emergentes la convierte en un elemento indispensable para el desarrollo de soluciones de seguridad resilientes, escalables y capaces de operar en tiempo real. El presente trabajo de fin de grado se apoya precisamente en esta visión, integrando técnicas de IA dentro de un entorno DevSecOps que permite automatizar cada fase del ciclo de vida del modelo, entrenamiento, despliegue, monitorización y actualización, consolidando un enfoque de seguridad inteligente y continua.

1.4. Incorporación del paradigma DevSecOps en la seguridad

La creciente automatización de la ciberseguridad y el uso de inteligencia artificial enlazan de forma natural con las prácticas actuales de ingeniería de software. Dado este contexto, DevOps, modelo que promueve la integración continua entre desarrollo y operaciones mediante automatización, colaboración y despliegues frecuentes, ha transformado la forma en la que las organizaciones construyen y mantienen sus sistemas [11].

A partir de este enfoque, surge DevSecOps, que incorpora la seguridad como un componente transversal dentro del ciclo de vida del desarrollo. Su objetivo no es añadir controles al final del proceso, sino integrarlos desde el diseño inicial mediante el principio de *shift-left security*, ampliamente adoptado en entornos de desarrollo moderno. Gracias a esta integración temprana, es posible detectar vulnerabilidades de forma anticipada y garantizar que cada nueva versión del sistema cumpla con los estándares de seguridad establecidos antes de su despliegue en producción [11].

Este cambio no solo implica la introducción de nuevas herramientas, sino también una transformación cultural y organizativa, en la que los equipos de desarrollo, operaciones y seguridad trabajan de manera colaborativa y continua. El objetivo es eliminar las barreras tradicionales entre departamentos, promoviendo la responsabilidad compartida sobre la seguridad y la calidad del producto final [6].

En la práctica, DevSecOps incorpora la automatización de tareas críticas dentro del pipeline de integración y despliegue continuo (CI/CD, *Continuous Integration/Continuous Deployment*). Estas tareas incluyen el análisis de vulnerabilidades, el escaneo de dependencias, la verificación de configuraciones seguras, el análisis estático y dinámico del código y la monitorización continua de los sistemas. Con todo esto, se consigue una detección temprana de fallos o vulnerabilidades, minimizando el riesgo de exposición en entornos de producción [11].

La aplicación de DevSecOps en proyectos de detección de amenazas resulta especialmente valiosa, ya que permite diseñar sistemas resilientes, escalables y auditables. Gracias a la integración de herramientas como Docker y Kubernetes, los modelos de inteligencia artificial pueden ser entrenados, validados y desplegados en entornos controlados y, así, garantizar la reproducibilidad y el aislamiento de los procesos. También, plataformas de automatización como Jenkins, GitHub Actions o GitLab CI facilitan la actualización y mantenimiento continuo de los modelos de IA sin interrupción del servicio [1].

Un ejemplo práctico de esta integración se observa en sistemas de detección de intrusiones basados en IA: mientras el modelo de análisis del tráfico se entrena o se actualiza automáticamente dentro de un pipeline CI/CD, herramientas como SonarQube, Trivy o Dependency-Check verifican en paralelo la seguridad del código, los contenedores y las librerías empleadas. Esto permite desplegar modelos nuevos o corregidos sin interrumpir el servicio y con garantía de seguridad.

La integración de inteligencia artificial y DevSecOps representa un avance significativo hacia la seguridad continua y automatizada. Este modelo no solo incrementa la eficiencia operativa del desarrollo, también fortalece la capacidad de las organizaciones para detectar, responder y mitigar amenazas en tiempo real, reduciendo la dependencia de la intervención manual y mejorando la resiliencia frente a entornos hostiles. Así, DevSecOps se consolida como un enfoque esencial para garantizar la seguridad integral en la era de la automatización y la inteligencia computacional.

Capítulo 2: Objetivos del trabajo

El presente trabajo tiene como finalidad el diseño e implementación de un sistema de detección de amenazas en tiempo real, apoyado en técnicas de inteligencia artificial y enmarcado dentro del paradigma DevSecOps**.** Este enfoque busca integrar la seguridad como parte del ciclo de vida del desarrollo, favoreciendo la automatización, la eficiencia operativa y la capacidad de respuesta temprana ante incidentes de seguridad.

2.1. Objetivo general

Diseñar e implementar un sistema de detección de amenazas en tiempo real basado en modelos de inteligencia artificial**,** integrado en un entorno DevSecOps que automatice las fases de entrenamiento, despliegue y monitorización**,** y así poder garantizar una respuesta oportuna y adaptativa frente a ciberataques.

2.2. Objetivos específicos

1. Analizar el estado del arte de los sistemas de detección de intrusiones y de las aplicaciones de inteligencia artificial en la ciberseguridad.
2. Seleccionar y preparar un conjunto de datos representativo para el entrenamiento de modelos de detección de anomalías.
3. Desarrollar y comparar distintos modelos de aprendizaje automático y profundo, evaluando su rendimiento mediante métricas de precisión, *recall, F1-score* y latencia.
4. Diseñar e implementar un entorno de integración y despliegue continuo (CI/CD) que automatice el ciclo de vida del modelo.
5. Incorporar herramientas de monitorización y *observability* que permitan analizar el comportamiento del sistema en tiempo real.
6. Explorar la integración de un asistente inteligente para consultas automáticas de vulnerabilidades y resultados del sistema.
7. Documentar la arquitectura y los resultados, proponiendo mejoras orientadas a la escalabilidad, resiliencia y seguridad continua.

2.3. Alcance y estructura del documento

El presente Trabajo deFin de Grado aborda el desarrollo de un sistema experimental para la detección de amenazas en tiempo real mediante técnicas de inteligencia artificial, con un enfoque basado en los principios de DevSecOps. El alcance del proyecto incluye el diseño conceptual, la implementación de un prototipo funcional, la evaluación de su rendimiento y la integración en un pipeline automatizado de despliegue y monitorización.

El proyecto no busca reemplazar las soluciones comerciales de detección de intrusiones existentes, sino demostrar la viabilidad técnica y metodológica de combinar modelos de IA con prácticas DevSecOps para lograr una seguridad continua y adaptativa. Se pretende que la propuesta sirva como base para futuras investigaciones y aplicaciones en entornos empresariales, contribuyendo al avance de la ciberseguridad inteligente.

Capítulo 3. Marco teórico y estado del arte

Este tercer capítulo tiene como propósito establecer los fundamentos conceptuales, técnicos y metodológicos que sustentan el desarrollo del sistema propuesto de detección de amenazas en tiempo real mediante técnicas de inteligencia artificial. A través de este marco teórico se construye la base científica necesaria para comprender las decisiones de diseño, las estrategias de implementación y las tecnologías empleadas en las etapas posteriores del proyecto.

En los siguientes apartados se abordan los puntos teóricos fundamentales a seguir sobre los que se apoya esta investigación: los principios de la ciberseguridad y su evolución frente al panorama actual de amenazas; los mecanismos de detección de intrusiones y sus diferentes enfoques; las aplicaciones de la inteligencia artificial, tanto en el aprendizaje automático (*machine learning*) como en el aprendizaje profundo (*deep learning*), orientadas a la defensa cibernética; la importancia de los datos en la construcción de modelos de detección eficaces; y

la integración del paradigma DevSecOps, que permite automatizar los procesos de seguridad dentro del ciclo de vida del *software* [9], [11].

Además, se lleva a cabo una revisión de investigaciones previas, proyectos académicos y soluciones comerciales relevantes, con el objetivo de contextualizar la propuesta dentro del estado del arte de la disciplina. Este análisis comparativo permitirá identificar los avances más destacados, las limitaciones persistentes y las tendencias actuales en la aplicación de técnicas de IA para la ciberseguridad [3], [9].

Este capítulo servirá como una base de referencia teórica y técnica sobre la cual se justifica la pertinencia del enfoque adoptado, proporcionando los argumentos necesarios para sustentar las decisiones de diseño, selección de herramientas y métodos de evaluación presentados en los capítulos siguientes [3], [11].

3.1. Ciberseguridad y evolución de las amenazas

La ciberseguridad puede definirse como el conjunto de medidas, políticas, procesos y tecnologías orientadas a proteger los sistemas informáticos, las redes de comunicación y los datos frente a accesos no autorizados, daños, alteraciones o interrupciones que puedan comprometer la disponibilidad, integridad o confidencialidad de la información [9]. En la actualidad, esta disciplina se ha convertido en un componente esencial para el funcionamiento de las organizaciones, tanto públicas como privadas, debido a su dependencia creciente de infraestructuras digitales y servicios interconectados.

El proceso de transformación digital de todo el mundo ha incrementado de forma significativa la superficie de exposición a riesgos tecnológicos: factores como la expansión del Internet de las Cosas (IoT), la computación en la nube, la movilidad laboral y la automatización industrial han multiplicado los puntos de vulnerabilidad. En consecuencia, las amenazas informáticas se han vuelto más frecuentes, sofisticadas y persistentes, obligando a las organizaciones a adoptar enfoques de defensa más integrales y basados en inteligencia [1].

Históricamente, los ataques informáticos se limitaban a acciones relativamente simples, como la propagación de virus, gusanos o troyanos, cuyo impacto era principalmente local o aislado. Conforme ha ido pasado el tiempo, sin embargo, en las últimas décadas, se ha producido una evolución hacia campañas organizadas de ciberespionaje, *ransomware* y sabotaje digital, dirigidas contra infraestructuras críticas y sectores estratégicos como la salud, la energía, la administración pública o los servicios financieros [3]. Este cambio de escala y propósito ha motivado la transición desde modelos de defensa perimetral hacia estrategias proactivas y automatizadas, capaces de detectar anomalías y responder antes de que el ataque se materialice.

En el panorama actual, las principales categorías de amenazas cibernéticas pueden agruparse en los siguientes tipos:

- Ataques de denegación de servicio (DoS/DDoS): buscan saturar los recursos de un sistema o red, impidiendo el acceso de los usuarios. Existen muchos ejemplos al respecto, uno de ellos es el ataque que recibió la compañía de videojuegos Activision Blizzard [12].
- Ataques de inyección o explotación, como *SQL Injection* o *Cross-Site Scripting* (XSS), que aprovechan vulnerabilidades en aplicaciones web para ejecutar código malicioso.

Un ejemplo de esto podría ser el ataque *SQL Injection* que sufrió WordPress por lo que tuvieron que actualizar su versión 5.8 en el 2022 [13].

- *Malware* avanzado y *ransomware* diseñados para cifrar, robar o exfiltrar información confidencial, afectando tanto a usuarios individuales como a grandes corporaciones. En enero del 2020, el fabricante GE detectó que la lista de procesos objetivo del *ransomware* EKANS incluye numerosas referencias a los productos Proficy. Dicha infección sería originada por el *ransomware* EKANS: los atacantes que utilizan este tipo de *malware* serían capaces de cifrar el disco duro, por ejemplo [14].
- Amenazas persistentes avanzadas (APT), caracterizadas por campañas prolongadas que combinan técnicas de intrusión, movimiento lateral y espionaje sostenido dentro de las redes corporativas de las empresas. El grupo Orangeworm utilizó un *malware* en 2015 para acceder a dispositivos médicos como máquinas de resonancia y robar información confidencial con fines de espionaje industrial [15].

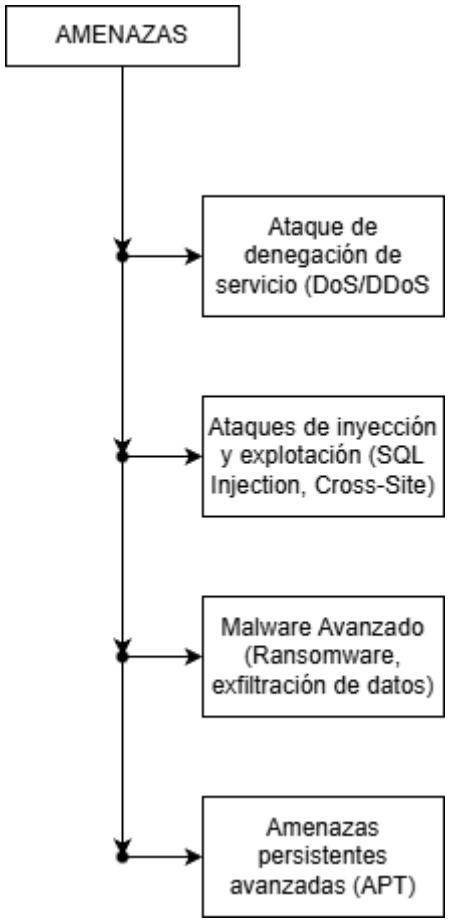

Figura 1. Taxonomía general de las amenazas cibernéticas en entornos de red.

Este esquema representado en la Figura 1, se muestras las principales categorías de ataques descritas en el panorama actual de la ciberseguridad. La magnitud y la sofisticación de estas amenazas demuestran que la protección perimetral tradicional ya no es suficiente para garantizar la seguridad de los sistemas. Actualmente, la defensa debe basarse en inteligencia y automatización, utilizando herramientas capaces de analizar grandes volúmenes de datos, correlacionar eventos y detectar comportamientos anómalos de forma autónoma. Esta evolución ha impulsado el desarrollo de sistemas inteligentes de detección y respuesta,

apoyados en inteligencia artificial y analítica avanzada, que permiten anticiparse a los incidentes y fortalecer la resiliencia de las infraestructuras digitales [1], [2].

3.2. Fundamentos de la detección de intrusiones

Los sistemas de detección de intrusiones, IDS, y de prevención de intrusiones, IPS, constituyen componentes fundamentales dentro de las estrategias modernas de ciberdefensa. Su principal función consiste en identificar comportamientos sospechosos o no autorizados en redes, sistemas o aplicaciones, y en algunos casos, responder de manera automática para bloquear o mitigar la amenaza antes de que se produzcan daños [2].

Estos sistemas analizan el tráfico de red, los registros de actividad y otros indicadores con el fin de detectar patrones anómalos o coincidencias con firmas conocidas de ataque. A lo largo del tiempo, se han consolidado tres enfoques principales de detección, que se diferencian en su metodología y capacidad de adaptación:

1. Detección basada en firmas: este enfoque consiste en comparar el tráfico de red con un conjunto de patrones o firmas ya conocidas que representan comportamientos maliciosos específicos, como secuencias de bytes o comandos asociados a ataques previos. Herramientas ampliamente utilizadas como Snort o Suricata aplican este método, el cual ofrece alta precisión frente a amenazas identificadas [16]. Sin embargo, su principal limitación radica en la incapacidad para detectar ataques *zero-day*, es decir, aquellos que explotan vulnerabilidades desconocidas o sin parche disponible [9], [16].
2. Detección basada en anomalías: este tipo de detección establece una línea base de comportamiento normal del sistema y evalúa las desviaciones respecto a dicha referencia. Los eventos que se alejan de lo esperado se clasifican como potencialmente maliciosos. Este método resulta útil para detectar amenazas nuevas o no documentadas, aunque tiende a generar falsos positivos, especialmente en entornos dinámicos donde el comportamiento legítimo puede variar con frecuencia [1].
3. Sistemas híbridos: buscan combinar las ventajas de los dos enfoques anteriores. Emplean reglas de firma para identificar amenazas conocidas y modelos de comportamiento para detectar ataques desconocidos o variantes emergentes. De esta forma, se logra un equilibrio entre precisión y capacidad adaptativa, reduciendo la dependencia de las actualizaciones manuales y mejorando la detección de anomalías sutiles.

Además de la metodología de detección, los IDS se clasifican según su ámbito de observación:

- Los IDS basados en *host* (HIDS), que supervisan el comportamiento interno de sistemas individuales, como el uso de recursos, las modificaciones de archivos o los accesos no autorizados.
- Los IDS basados en red (NIDS), que, por otra parte, analizan el tráfico entre dispositivos, inspeccionando los paquetes de datos en busca de indicios de intrusión o manipulación.

En los últimos años, el interés se ha desplazado hacia sistemas de detección distribuidos e inteligentes, apoyados en técnicas de aprendizaje automático y profundo, que permiten procesar grandes volúmenes de información y adaptarse a entornos cambiantes y de alta velocidad. Estos sistemas, además de aumentar la precisión de detección, reducen los falsos

positivos y facilitan la automatización de la respuesta, representando una evolución natural hacia una ciberdefensa predictiva y proactiva [9].

Un falso positivo podríamos definirlo como como una alerta que indica un ataque inexistente. Por ejemplo, un empleado sube un archivo de gran tamaño a la nube corporativa y puede ser marcado erróneamente como una actividad de exfiltración de documentación. Conseguir una reducción de falsos positivos mejoraría la eficiencia personal del Centro de Operaciones de Seguridad y favorece la automatización de la respuesta.

3.3. Aplicación de la inteligencia artificial en la ciberseguridad

La aplicación de la inteligencia artificial en el ámbito de la ciberseguridad ha experimentado un crecimiento acelerado en los últimos años, impulsada por la necesidad de responder a amenazas cada vez más sofisticadas, automatizadas y en constante evolución. Las técnicas de *Machine Learning* y *Deep Learning* se han consolidado como herramientas esenciales para mejorar la precisión, la escalabilidad y la adaptabilidad de los sistemas de detección de intrusiones, aportando capacidades que superan ampliamente a los enfoques tradicionales basados en reglas o firmas [3], [10].

A diferencia de los métodos convencionales, que dependen de patrones previamente conocidos, los modelos de IA son capaces de aprender comportamientos complejos a partir de grandes volúmenes de datos, identificar anomalías sutiles y detectar actividades maliciosas incluso cuando no existe información previa sobre el ataque. Esta capacidad de análisis adaptativo facilita el desarrollo de mecanismos de defensa dinámicos, predictivos y capaces de evolucionar junto con las amenazas emergentes, lo que resulta especialmente valioso en entornos distribuidos y de alta velocidad [9].

Diversas investigaciones recientes demuestran la eficacia de técnicas de aprendizaje automático y profundo en la detección de intrusiones: las redes LSTM han destacado en la identificación de ataques secuenciales en flujos de red, superando a modelos clásicos al capturar dependencias temporales entre eventos [17]. También, enfoques híbridos basados en CNN-RNN han mostrado mejoras significativas en precisión y reducción de falsos positivos en *datasets* como CICIDS2017 y UNSW-NB15 [18]. Este cuerpo de evidencia científica confirma el potencial de la inteligencia artificial para fortalecer la resiliencia de los sistemas modernos de ciberdefensa.

La revisión sistemática publicada por ACM Computing Surveys proporciona una clasificación exhaustiva de los enfoques basados en aprendizaje automático para la detección de intrusiones, analizando *datasets*, metodologías y retos actuales en la disciplina [19].

3.3.1. Enfoques de aprendizaje supervisado

A diferencia de los enfoques tradicionales basados en reglas o firmas, los modelos de inteligencia artificial son capaces de aprender patrones de comportamiento a partir de grandes volúmenes de datos, lo que les permite identificar anomalías o actividades maliciosas incluso en ausencia de información previa sobre el ataque. Este enfoque proporciona una defensa más dinámica y predictiva, capaz de evolucionar junto con las amenazas emergentes.

El aprendizaje supervisado constituye uno de los métodos más extendidos en la detección de intrusiones, ya que permite entrenar modelos utilizando conjuntos de datos etiquetados donde cada muestra se clasifica previamente como "normal" o "maliciosa". De esta forma, el algoritmo aprende las características distintivas de cada categoría, pudiendo predecir con precisión el comportamiento de nuevos eventos una vez desplegado en producción [9]. Este enfoque se ha aplicado ampliamente sobre *datasets* como UNSW-NB15, CICIDS2017 o CTU-13, que contienen múltiples tipos de ataques y tráfico legítimo representativo.

Entre los algoritmos más empleados destacan:

Árboles de decisión y Random Forest (RF): valorados por su interpretabilidad y robustez ante ruido, estos modelos ofrecen un equilibrio adecuado entre precisión y eficiencia computacional. Diversos estudios han demostrado que Random Forest obtiene resultados competitivos en *datasets* modernos de intrusiones, superando el 95% de exactitud en escenarios de tráfico mixto y manteniendo tiempos de inferencia reducidos. Su capacidad para manejar datos numéricos y categóricos los convierte en una opción fiable en entornos operativos variados [20].

Máquinas de Vectores de Soporte (SVM): las SVM son especialmente útiles en clasificaciones binarias o multiclase con datos de alta dimensionalidad, característica habitual en flujos de red contemporáneos. Investigaciones recientes destacan su rendimiento estable en la detección de ataques de reconocimiento, explotación y escaneo, ofreciendo márgenes de error reducidos incluso con conjuntos de características complejos [21].

Redes Neuronales Artificiales (ANN): las ANN permiten modelar relaciones no lineales complejas, lo que las hace adecuadas para escenarios donde los patrones maliciosos no son evidentes o se encuentran altamente dispersos en el espacio de características. Cuando se combinan con técnicas de preprocesamiento como la normalización o la reducción de dimensionalidad, las ANN logran precisiones superiores al 95% en diversos estudios, mostrando una notable capacidad para adaptarse a variaciones sutiles en el comportamiento del tráfico de red [22].

Si bien los enfoques supervisados destacan por su precisión, su rendimiento depende en gran medida de la calidad y representatividad del conjunto de entrenamiento. Un preprocesamiento adecuado, que incluya normalización, selección de características y balanceo de clases, resulta esencial para evitar sesgos, reducir el sobreajuste y garantizar un desempeño fiable en contextos reales.

Por tanto, el aprendizaje supervisado constituye una de las ramas más extendidas de la inteligencia artificial aplicada a la ciberseguridad. En este enfoque, los modelos se entrenan con conjuntos de datos etiquetados que contienen ejemplos de comportamientos normales y maliciosos. Entre los algoritmos más empleados destacan las Máquinas de Vectores de Soporte (SVM), los Random Forests y las Redes Neuronales Artificiales (ANN), ampliamente utilizados en la clasificación de tráfico de red, la detección de intrusiones y el análisis de *malware*. Estos modelos aprenden a reconocer las características distintivas de cada categoría, permitiendo clasificar nuevas muestras con un alto grado de exactitud.

3.3.2. Enfoques no supervisados y de detección de anomalías

En los escenarios donde los datos no están etiquetados o las categorías de comportamiento no son conocidas, los sistemas de detección de intrusiones recurren a técnicas no supervisadas y a métodos orientados al descubrimiento de anomalías. Estos enfoques resultan especialmente valiosos en entornos donde las amenazas evolucionan rápidamente, en particular ante ataques desconocidos o de tipo *zero-day*, ya que no dependen de firmas o patrones previamente definidos y pueden adaptarse a condiciones cambiantes.

Entre los métodos más utilizados se encuentran los algoritmos de *clustering*, como K-Means, DBSCAN o Gaussian Mixture Models (GMM), cuyo objetivo es agrupar las observaciones en función de su similitud. Una vez formados los grupos, las instancias que se alejan significativamente de los centroides o de la densidad del conjunto se consideran posibles anomalías. Este enfoque permite identificar comportamientos atípicos que podrían corresponderse con intrusiones, movimientos laterales o actividad automatizada no registrada previamente [23].

Otro pilar fundamental en la detección de anomalías lo constituyen los *autoencoders*, modelos basados en redes neuronales capaces de aprender una representación eficiente del tráfico normal mediante un proceso de reconstrucción. Durante su entrenamiento, el *autoencoder* se optimiza para reproducir con precisión las entradas que representan comportamientos legítimos. En fase de inferencia, una desviación significativa en el error de reconstrucción, superior a un umbral establecido, puede indicar la presencia de actividad maliciosa. Esta técnica es especialmente eficaz en entornos donde los patrones de ataque son sutiles o difíciles de observar directamente, permitiendo detectar anomalías incluso en tráfico cifrado o altamente ruidoso [24].

Los enfoques no supervisados y basados en anomalías proporcionan una capa de defensa flexible y adaptativa, complementando a los modelos supervisados y ampliando la capacidad del sistema para identificar ataques emergentes sin conocimiento previo de su naturaleza. Su combinación con técnicas de preprocesamiento adecuadas y análisis continuo contribuye a fortalecer la resiliencia de los sistemas modernos de ciberdefensa [9].

En definitiva, el aprendizaje no supervisado se emplea en escenarios donde los datos no están previamente etiquetados. Los algoritmos de *clustering*, como K-Means o DBSCAN, y las arquitecturas de *autoencoders*, permiten identificar desviaciones respecto al comportamiento esperado, siendo especialmente útiles en la detección de anomalías o de ataques desconocidos (zero-day). Estos métodos ofrecen flexibilidad en entornos dinámicos donde los patrones de ataque evolucionan constantemente

3.3.3. Aprendizaje profundo y arquitecturas avanzadas

El *Deep Learning* representa una evolución significativa respecto a los enfoques de aprendizaje supervisado y no supervisado, ya que se basa en redes neuronales de múltiples capas capaces de aprender automáticamente representaciones jerárquicas y abstractas directamente de los datos, sin necesidad de realizar una ingeniería manual de características. Esta capacidad de extracción automática permite abordar con mayor eficacia la complejidad y variabilidad inherentes al tráfico de red moderno [9].

Entre las arquitecturas más destacadas se encuentran las Convolutional Neural Networks, CNN, ampliamente utilizadas en tareas de clasificación de tráfico. Estas redes son capaces de transformar secuencias de red en matrices o imágenes de características, identificando patrones espaciales que reflejan estructuras propias del comportamiento malicioso. Diversos estudios han demostrado su efectividad; por ejemplo, investigaciones recientes señalan que una CNN entrenada sobre el *dataset* CICIDS2017 obtuvo precisiones del 98,9%, evidenciando su capacidad para discriminar entre tráfico normal y ataques.

Otro grupo de modelos relevantes lo conforman las Recurrent Neural Networks, RNN, y sus variantes avanzadas, como LSTM, Long Short-Term Memory. Estas arquitecturas están diseñadas para capturar dependencias temporales en los datos, lo que las hace especialmente adecuadas para analizar secuencias de eventos y detectar patrones asociados a ataques persistentes o distribuidos. Estudios aplicados en entornos IoT han evidenciado que los modelos LSTM superan a algoritmos clásicos como SVM o KNN en la detección de amenazas secuenciales, debido a su capacidad para retener y procesar información a largo plazo.

A partir de la evolución natural de estos enfoques han surgido modelos híbridos CNN–LSTM, que combinan la fortaleza espacial de las CNN con la capacidad temporal de las LSTM.[18] Esta integración permite capturar simultáneamente características locales del tráfico y dependencias entre eventos, lo que se traduce en mejoras tangibles en precisión y reducción de falsos positivos. En estudios comparativos recientes, estos modelos híbridos lograron disminuir en torno a un 30% la tasa de falsas alarmas frente a aproximaciones tradicionales.

Gracias a estas capacidades, las arquitecturas de *Deep Learning* permiten desarrollar sistemas de detección autónomos, adaptativos y capaces de mejorar su rendimiento conforme se incorporan nuevos datos. Su naturaleza evolutiva, derivada de procesos de aprendizaje continuo, reduce la dependencia de intervención humana y las convierte en una herramienta esencial para la ciberdefensa moderna.

Con esto, el aprendizaje profundo representa una evolución significativa al utilizar redes neuronales de múltiples capas que son capaces de aprender automáticamente características complejas y jerárquicas a partir de los datos. Los modelos, como las Convolutional Neural Networks (CNN) y las Recurrent Neural Networks (RNN), han demostrado resultados sobresalientes en la clasificación de tráfico de red, la detección secuencial de eventos y el análisis de comportamiento temporal, al poder identificar relaciones no lineales y dependencias entre observaciones consecutivas. Gracias a estas arquitecturas es posible desarrollar sistemas de detección más precisos, autónomos y capaces de adaptarse a nuevos patrones maliciosos sin intervención humana directa.

3.3.4. Desafíos de la IA en la detección de intrusiones

A pesar de los avances significativos logrados por la inteligencia artificial en la detección de intrusiones, su aplicación práctica continúa enfrentándose a diversos desafíos que deben ser abordados para garantizar la fiabilidad, robustez y eficacia de los sistemas de ciberdefensa. Estos retos no solo afectan al rendimiento técnico de los modelos, sino también a su integración operativa dentro de infraestructuras reales.

- **Dependencia de *datasets* amplios, diversos y actualizados:** los modelos de IA requieren conjuntos de datos representativos que reflejen tanto tráfico legítimo como patrones de ataque actuales. No obstante, muchos de los *datasets* públicos disponibles

presentan problemas como desbalance de clases, ruido o falta de realismo en la captura de tráfico, lo que puede limitar la capacidad de generalización del sistema. Además, al evolucionar las tácticas de ataque, los modelos deben ser entrenados periódicamente para evitar quedarse obsoletos [25].

- **Dificultad de interpretabilidad en modelos complejos:** las arquitecturas profundas, como CNN o LSTM, suelen comportarse como "cajas negras", dificultando la comprensión de las razones detrás de una predicción concreta. Esta falta de compresión complica las auditorías de seguridad y la validación de resultados, especialmente en entornos regulados. Actualmente se están incorporando metodologías de interpretabilidad como LIME o SHAP para aportar transparencia al proceso de toma de decisiones [26], [27].
- **Vulnerabilidad ante ataques adversariales:** los sistemas basados en IA pueden ser manipulados mediante pequeños cambios en los datos de entrada, diseñados para provocar errores en la clasificación. Esta técnica, conocida como *evasion attack*, puede permitir que un atacante modifique mínimamente un paquete o flujo de red para que sea identificado como benigno. Esta vulnerabilidad plantea la necesidad de desarrollar modelos más robustos y mecanismos de defensa adicionales [26], [27], [28].
- **Elevados requerimientos computacionales:** el entrenamiento y despliegue de modelos de aprendizaje profundo exige recursos significativos de cálculo, especialmente cuando se requiere procesar tráfico en tiempo real. Esto puede representar una barrera para organizaciones con infraestructuras limitadas, ya que la operación eficiente de estos modelos suele requerir GPUs o hardware especializado.
- ***Concept drift* o evolución del comportamiento de la red:** el comportamiento normal del tráfico de red cambia con el tiempo debido a nuevas aplicaciones, dispositivos, servicios y patrones de uso. Si un modelo no se actualiza de forma periódica o no incorpora mecanismos de aprendizaje continuo, su precisión se degrada progresivamente, generando falsos positivos o negativos [28], [29].
- **Integración operativa y compatibilidad con sistemas existentes:** para que la inteligencia artificial tenga un impacto real en la seguridad, debe integrarse correctamente con herramientas SIEM, centros de operaciones de seguridad (SOC) y pipelines DevSecOps. Sin esta integración, aunque los modelos sean técnicamente potentes, su utilidad práctica se ve limitada [25], [29].

Estos desafíos evidencian que el diseño de sistemas de detección basados en inteligencia artificial requiere encontrar un equilibrio entre rendimiento, interpretabilidad y eficiencia. La combinación de modelos inteligentes con mecanismos tradicionales de seguridad, junto con procesos continuos de validación, actualización y auditoría, constituye una estrategia fundamental para garantizar la confiabilidad del sistema en entornos reales.

3.3.5. Ventajas de la IA en ciberseguridad

La incorporación de la IA en los sistemas de defensa digital aporta ventajas sustanciales frente a los métodos tradicionales:

- Mejora de la capacidad de detección gracias al aprendizaje continuo de nuevos patrones.
- Reducción del tiempo de respuesta y de la carga manual sobre los analistas de seguridad.

- Disminución de falsos positivos mediante el ajuste dinámico y la optimización de los modelos.
- Procesamiento eficiente de flujos de datos en tiempo real, incluso en entornos de alta latencia o gran volumen.

La aplicación de la inteligencia artificial en la ciberdefensa no debe entenderse como un reemplazo de los sistemas existentes, más bien como una ampliación estratégica o complemento que potencia la capacidad de anticipación, reacción y adaptación frente a amenazas en constante evolución.

3.3.6. Revisión de soluciones existentes y proyectos relacionados

En los últimos años, diferentes investigaciones y proyectos tecnológicos han explorado el uso de la inteligencia artificial y el aprendizaje automático para mejorar la detección de intrusiones, con resultados alentadores en términos de precisión, escalabilidad y capacidad de adaptación de los modelos. Sin embargo, a pesar de los avances logrados, aún existen limitaciones en la integración de estas soluciones dentro de entornos automatizados y orientados a la práctica operativa, lo que motiva la propuesta desarrollada en este trabajo.

Entre los estudios más relevantes destacan los siguientes:

- Córdovez Machado, Cruz Garzón e Inca Balseca (2025) realizaron un análisis de aprendizaje supervisado aplicado a la predicción de brechas de ciberseguridad en sistemas empresariales. En su investigación concluyen que los modelos de Random Forest y Support Vector Machines (SVM) ofrecen un equilibrio óptimo entre precisión y velocidad de inferencia, resultandos apropiados para entornos corporativos donde la eficiencia computacional es esencial [2].
- Pinto, Herrera, Donoso y Gutiérrez (2023) llevaron a cabo una revisión exhaustiva sobre sistemas de detección de intrusiones basados en aprendizaje automático orientados a infraestructuras críticas. Los autores subrayan que los modelos basados en redes neuronales profundas (*Deep Learning*), especialmente las Convolutional Neural Networks (CNN) y las Recurrent Neural Networks (RNN), logran una detección más precisa de patrones temporales complejos en el tráfico de red [3].
- Quirumbay Yagual, Castillo Yagual y Coronel Suárez (2022) presentan una revisión sistemática sobre el uso del aprendizaje profundo en la ciberseguridad, destacando la eficacia de las arquitecturas secuenciales y de *autoencoders* para la detección de anomalías y ataques desconocidos (*zero-day*). Asimismo, resaltan la necesidad de contar con conjuntos de datos realistas y actualizados como CICIDS2017 o UNSW-NB15 para garantizar la validez de los modelos [9].
- Castro Sánchez (2020) aborda la aplicación del enfoque DevSecOps mediante herramientas *open source*, destacando la importancia de incorporar la seguridad desde las etapas iniciales del desarrollo. Su trabajo demuestra que la integración de análisis automatizados de vulnerabilidades en los *pipelines* de CI/CD mejora significativamente la detección temprana y la trazabilidad de los incidentes [6].
- Lázaro, Moreno y Cervantes (2023), en su *Guía de iniciación en la seguridad aplicada al DevOps*, publicada por ISMS Forum, remarcan la relevancia de la automatización y la cultura colaborativa en la implementación de prácticas DevSecOps. Estas prácticas favorecen la integración continua de controles de seguridad y la supervisión constante de los sistemas, contribuyendo a una seguridad más preventiva y adaptativa [11].

Por otro lado, en el ámbito de los proyectos de código abierto, destacan soluciones como Wazuh y Zeek, que combinan la monitorización avanzada con módulos de análisis de comportamiento y visualización mediante la pila ELK (Elasticsearch, Logstash y Kibana). Estas herramientas ofrecen capacidades de correlación de eventos y detección en tiempo real, aunque su efectividad depende de configuraciones manuales y de la calidad de las reglas implementadas [1].

En general, los estudios revisados coinciden en que los modelos basados en inteligencia artificial han demostrado una notable mejora en la detección de intrusiones.

3.4. *Datasets* y fuentes de datos en la detección de intrusiones

El rendimiento de los sistemas de detección de intrusiones basados en inteligencia artificial depende en gran medida de la calidad, variedad y representatividad del conjunto de datos empleado durante el entrenamiento y la validación. Un *dataset* adecuado debe reflejar la complejidad del tráfico real, incluir tanto actividades legítimas como maliciosas y contemplar múltiples vectores de ataque, topologías de red y escenarios temporales. Tal como señala la literatura especializada, la ausencia de datos diversos y actualizados limita significativamente la capacidad de generalización de los modelos [3].

La obtención de registros reales de ciberseguridad constituye uno de los mayores desafíos del campo. Las organizaciones suelen restringir el acceso a sus datos por motivos de confidencialidad, cumplimiento normativo o protección reputacional, lo que dificulta la generación de *datasets* fieles a las condiciones reales. Por esto mismo, la comunidad científica ha desarrollado conjuntos de datos estandarizados que simulan entornos representativos y que permiten comparar de forma homogénea el rendimiento de distintos modelos de aprendizaje automático y profundo. No obstante, muchos de estos *datasets* presentan limitaciones habituales, como distribuciones artificiales, ausencia de tráfico cifrado o falta de interacción humana realista.

Entre los conjuntos de datos más empleados, junto con sus características técnicas y su relevancia en el aprendizaje automático:

- **KDD Cup 99: *dataset* histórico y base de la investigación clásica**
  El *dataset* KDD Cup 99, ampliamente utilizado en los inicios de la investigación en detección de intrusiones, contiene conexiones de red etiquetadas como normales o maliciosas. Estas últimas se agrupan en cuatro categorías clásicas definidas: DoS (Denial of Service), Probe (exploración/escaneo), U2R (User-to-Root) o R2L (Remote-to-Local).
  Este *dataset* incluye 41 características por conexión, tales como duración, tipo de protocolo, servicio solicitado, número de bytes enviados/recibidos, estado de la conexión o número de intentos fallidos. Estudios que se llevaron a cabo, en uno de los análisis más exhaustivos sobre sus limitaciones, destacando su redundancia, desequilibrio y la falta de realismo en relación con los ataques modernos.
  Las 41 características se normalizan o codifican, y posteriormente se entrenan modelos supervisados como SVM, Random Forest o redes neuronales artificiales para clasificar conexiones en las distintas categorías de ataque.
  Aunque actualmente se considera obsoleto, sigue siendo útil para fines didácticos y para comparaciones entre algoritmos clásicos

.

- **NSL-KDD: versión depurada y equilibrada del *dataset* original**
  El *dataset* surge como una mejora del KDD'99, eliminando duplicados y corrigiendo desequilibrios significativos en la distribución de clases. Mantiene las mismas 41 características originales, pero proporciona un conjunto más equilibrado y adecuado para comparar el rendimiento de distintos algoritmos sin el sesgo inherente al *dataset* anterior.
  Revisiones como las de Blanco & Bringas destacan que, aunque NSL-KDD reduce el ruido, continúa sin representar tráfico moderno y carece de ataques recientes y de tráfico cifrado.
  En el entrenamiento típico, los modelos aprenden a clasificar conexiones basándose en las mismas 41 *features*, pero con un nivel mucho menor de redundancia que el *dataset* original.

- **UNSW-NB15: realismo moderado y diversidad de ataques**
  Este *dataset*, desarrollado por Moustafa y Slay, combina tráfico sintético y real capturado en un entorno experimental moderno. Incluye 49 características, organizadas en cuatro grandes grupos: características de flujo (duración, número de paquetes, bytes), características basadas en contenido (*payload*, *flags*), características de comportamiento y estadísticas de red.
  Contiene nueve categorías de ataque, entre ellas: DoS, Exploits, Fuzzers, Reconnaissance, Shellcode, Generic, Backdoor y Worms.
  Su equilibrio entre realismo y diversidad lo ha convertido en uno de los datasets más citados en investigaciones publicadas entre 2017 y 2024, especialmente en experimentos de aprendizaje supervisado. No obstante, presenta limitaciones como la escasez de tráfico cifrado y la representación reducida de comportamientos humanos complejos.
- **CICIDS2017: *dataset* moderno orientado a secuencias temporales**:
  El *dataset* CICIDS2017, creado por el Canadian Institute for Cybersecurity, es actualmente uno de los más completos y utilizados en la evaluación de sistemas de detección de intrusiones. Incluye tráfico real capturado durante varios días en un entorno corporativo simulado, incorporando una amplia variedad de ataques: DDoS, Web Attacks, Infiltration, Botnet, PortScan, ataques de fuerza bruta SSH/FTP, entre otros.
  Este *dataset* contiene más de 80 características por flujo, generadas mediante CICFlowMeter, que incluyen: estadísticas temporales, tamaños de paquetes, *flags* TCP, tasas de envío, variaciones temporales, número de paquetes entrantes y salientes, etc.
  Su organización cronológica permite analizar secuencias de eventos, lo que lo convierte en un recurso ideal para entrenar modelos orientados al análisis temporal, como LSTM, GRU y arquitecturas híbridas CNN–LSTM. Entre sus limitaciones, diversas revisiones señalan la ausencia de tráfico cifrado real y el carácter parcialmente guiado de algunos ataques, lo que puede afectar al realismo del conjunto [30], [31].

En la práctica, el proceso de preprocesamiento de los datos es tan relevante como la selección del propio *dataset*. Este proceso incluye la limpieza de datos, la normalización de variables numéricas, la codificación de atributos categóricos, la eliminación de redundancias y el balanceo de clases, especialmente cuando el tráfico malicioso está subrepresentado. Un tratamiento inadecuado de los datos puede conducir a modelos sesgados o sobreajustados, reduciendo la capacidad de generalización y la fiabilidad de los resultados [9], [32].

3.5. DevSecOps: integración de seguridad en el ciclo de vida del software

El paradigma DevSecOps surge como una evolución del modelo DevOps, motivada por la creciente necesidad de integrar la seguridad de forma nativa en todas las fases del ciclo de vida del desarrollo y despliegue del software. En contraste con los enfoques tradicionales, donde las medidas de seguridad se incorporaban únicamente en las etapas finales (durante las pruebas o auditorías previas a la entrega), DevSecOps promueve el principio de *shift-left security*, es decir, desplazar las prácticas de seguridad hacia las primeras fases del proceso de desarrollo [6], [11].

Esta integración temprana de la seguridad implica un cambio cultural y metodológico dentro de las organizaciones, donde los equipos de desarrollo, operaciones y seguridad trabajan de forma colaborativa, compartiendo la responsabilidad sobre la protección del software y la infraestructura. El objetivo es garantizar que la seguridad no sea un obstáculo, sino un componente continuo y automatizado del ciclo de vida del producto [6], [11].

La implementación práctica de DevSecOps se apoya en la creación de *pipelines* de Integración y Despliegue Continuos (CI/CD), que automatizan las etapas de compilación, validación, prueba, empaquetado y despliegue del software. Dentro de estos pipelines, se incorporan controles automáticos de seguridad que verifican la calidad del código, detectan vulnerabilidades y supervisan el comportamiento del sistema en producción. Estas medidas permiten identificar y corregir fallos de seguridad desde el inicio del proceso, reduciendo el coste y el tiempo de respuesta ante incidentes.

Entre las herramientas más utilizadas en entornos DevSecOps destacan:

- Jenkins, GitLab CI y GitHub Actions, empleadas para la orquestación de *pipelines* y la ejecución automatizada de pruebas de seguridad.
- Docker y Kubernetes, que facilitan el uso de contenedores, el aislamiento de entornos y el despliegue reproducible de aplicaciones y modelos de IA.
- SonarQube, OWASP Dependency-Check y Snyk, orientadas al análisis de vulnerabilidades, auditoría de dependencias y revisión estática del código fuente.
- Prometheus, Grafana y ELK Stack (Elasticsearch, Logstash y Kibana), esenciales para la *observability*, trazabilidad y monitorización continua del sistema en entornos de producción.

La adopción del enfoque DevSecOps ofrece ventajas sustanciales para los proyectos de detección de amenazas basados en inteligencia artificial, ya que incrementa la agilidad, resiliencia y capacidad de adaptación del sistema frente a nuevas vulnerabilidades. Además, favorece la automatización de tareas repetitivas, la reducción de errores humanos y la mejora de la respuesta ante incidentes, configurándose, así, como un marco metodológico idóneo para el desarrollo y despliegue de soluciones de ciberseguridad inteligentes [1], [6], [11].

3.5.1. Monitorización y observabilidad en entornos DevSecOps

No debemos confundir monitorización con observabilidad: el primero se enfoca en detectar problemas conocidos mediante la recopilación de métricas y el establecimiento de umbrales predefinidos; y, el segundo, es un enfoque más amplio que permite entender el 'porqué' de un problema, conocido o no, al recopilar y correlacionar datos como métricas, logs o trazas para inferir el estado interno del sistema. Dicho esto, otro componente clave de este enfoque es la observabilidad (*observability*), que permite supervisar en tiempo real el estado del sistema y la eficacia de las medidas de seguridad implementadas. Herramientas como Prometheus, Grafana o ELK Stack (Elasticsearch, Logstash y Kibana) proporcionan métricas detalladas sobre el rendimiento, las alertas y los incidentes de seguridad, lo que contribuye a la toma de decisiones basadas en datos y a la mejora continua del sistema [11].

3.6. Sistemas de detección en tiempo real

La detección en tiempo real constituye un elemento esencial de las estrategias modernas de ciberdefensa inteligente, ya que permite identificar, analizar y responder a incidentes a medida que se producen, reduciendo el tiempo de exposición y el impacto potencial de los ataques. En aplicaciones de seguridad, se considera detección en tiempo real aquella cuya latencia oscila entre 10 y 100 milisegundos, dependiendo del volumen de tráfico y de la criticidad del entorno [7], [8]. En escenarios de alta demanda, como redes corporativas de gran tamaño o servicios expuestos a Internet, la latencia máxima recomendada suele situarse por debajo de 50 ms, de modo que la respuesta sea prácticamente inmediata.

Para alcanzar estos niveles de rapidez, los sistemas deben procesar flujos continuos de datos y ser capaces de extraer características, ejecutar el modelo de detección y generar alertas dentro de ese margen temporal. Esta exigencia ha motivado el desarrollo de modelos ligeros y optimizados, así como arquitecturas diseñadas específicamente para minimizar la latencia de inferencia.

Diversas investigaciones recientes han evaluado métodos de detección en tiempo real basados en inteligencia artificial:

- Laghrissi, Douzi y Hssina (2021) implementaron un modelo basado en LSTM, obteniendo tiempos de inferencia inferiores a 30 milisegundos por flujo, lo que permite su uso en entornos de red de alta velocidad [17].
- Bmaberm Katkuri, Sharma y Angurala (2025) desarrollaron un sistema híbrido CNN–LSTM capaz de detectar intrusiones en tráfico complejo con latencias promedio cercanas a 40–60 milisegundos, manteniendo una alta precisión incluso con grandes volúmenes de datos [18].
- Fabián, Ramos, Larriva y Ponente (2021) evaluaron distintos modelos supervisados en tráfico real y observaron que Random Forest y SVM pueden operar por debajo de los 20 ms en fases de inferencia cuando el conjunto de características se reduce adecuadamente [8].
- Choras y Pawlicki (2021) demostraron que redes neuronales profundas optimizadas con técnicas de poda y cuantización pueden alcanzar tiempos de ejecución inferiores a 15 ms, lo que las hace adecuadas para despliegues en tiempo real [22].

Estos estudios coinciden en que el equilibrio entre precisión y latencia depende tanto del modelo como del proceso de extracción de características. En algunos casos, la propia generación de atributos, por ejemplo, estadísticas de flujo, puede consumir más tiempo que la inferencia del modelo, lo que obliga a implementar técnicas de preprocesamiento incremental o extracción paralela.

El diseño de sistemas de detección en tiempo real también debe abordar diversos desafíos técnicos:

- Minimización de la latencia de inferencia, asegurando que el modelo genere una decisión en milisegundos sin sacrificar precisión.
- Estabilidad en entornos no estacionarios, donde el comportamiento del tráfico cambia con el tiempo (*concept drift*).
- Escalabilidad horizontal, necesaria para manejar picos de tráfico o ataques masivos como DDoS sin degradar el rendimiento.

Finalmente, la integración de modelos de inteligencia artificial con plataformas SIEM, como Wazuh, Splunk o Elastic Security, permite automatizar la correlación de eventos y la respuesta temprana, reduciendo aún más el tiempo de detección y contención. Gracias a esta convergencia entre IA, analítica en *streaming* y automatización, los sistemas de detección en tiempo real se consolidan como una pieza clave para una ciberseguridad proactiva, escalable y adaptativa [11].

3.7. Implicaciones del marco teórico y del estudio del arte

El análisis teórico y bibliográfico desarrollado a lo largo de este capítulo permite extraer una serie de conclusiones fundamentales que sustentan la base conceptual del sistema propuesto:

Limitaciones de los métodos tradicionales: los sistemas de detección de intrusiones basados en firmas o reglas, aunque han sido eficaces frente a amenazas conocidas, resultan insuficientes ante la creciente complejidad y velocidad de los ataques modernos. Su dependencia de patrones predefinidos impide una respuesta eficaz ante amenazas nuevas o desconocidas (*zero-day*), lo que refuerza la necesidad de adoptar modelos más adaptativos [1], [2].

Aportaciones de la Inteligencia artificial: las técnicas de aprendizaje automático y profundo constituyen una alternativa sólida para la detección temprana de intrusiones. Permiten analizar grandes volúmenes de datos, identificar comportamientos anómalos y anticiparse a incidentes con una precisión superior a los métodos tradicionales. Estudios recientes, como los de Pinto, Herrera, Donoso y Gutiérrez (2023) y Quirumbay Yagual, Castillo Yagual y Coronel Suárez (2022), demuestran la eficacia de las redes neuronales y de los *autoencoders* en la detección de patrones complejos y ataques desconocidos.

Relevancia de los *datasets* representativos: el uso de conjuntos de datos actualizados y realistas, como *CICIDS2017* o *UNSW-NB15*, sigue siendo esencial para el entrenamiento y validación de modelos robustos. Tal y como destacan Quirumbay Yagual, Castillo Yagual y Coronel Suárez (2022), la calidad de los datos influye directamente en la capacidad de generalización de los modelos y, por tanto, en la fiabilidad de los sistemas de detección en entornos reales.

Aportes del enfoque DevSecOps: el paradigma DevSecOps se consolida como un marco metodológico clave para integrar la seguridad a lo largo de todo el ciclo de vida del software. Trabajos como los de Castro Sánchez (2020) y Lázaro, Moreno y Cervantes (2023) evidencian que la automatización de las pruebas, el análisis de vulnerabilidades y la monitorización continua incrementan la resiliencia, la trazabilidad y la agilidad operativa de los sistemas de seguridad.

Necesidad de integración completa: aunque la investigación ha avanzado significativamente en la mejora de los modelos de detección, sigue existiendo una brecha entre el desarrollo teórico y la implementación práctica en entornos automatizados. La integración efectiva entre inteligencia artificial, automatización y monitorización continua, enmarcada dentro de un ecosistema DevSecOps, es un desafío pendiente que condiciona la adopción real de estos sistemas en contextos empresariales [1], [11].

El marco teórico evidencia la necesidad de desarrollar sistemas inteligentes, escalables y observables, capaces de combinar la potencia analítica de la inteligencia artificial con la eficiencia y automatización del enfoque DevSecOps. Esta convergencia representa un paso decisivo hacia una ciberseguridad proactiva, autónoma y adaptativa, alineada con las exigencias de la protección digital en entornos empresariales modernos.

En este contexto, la propuesta desarrollada en el presente trabajo de fin de grado pretende contribuir a cerrar dicha brecha, integrando modelos de inteligencia artificial para la detección de intrusiones dentro de un entorno DevSecOps automatizado, capaz de gestionar el entrenamiento, despliegue y supervisión continua en tiempo real. De este modo, se busca ofrecer un enfoque integral, escalable y adaptable a los requerimientos de la ciberseguridad moderna.

Capítulo 4. Diseño del sistema propuesto

4.1. Arquitectura general del sistema

El sistema propuesto tiene como objetivo principal la detección de amenazas en tiempo real mediante la aplicación de técnicas de inteligencia artificial integradas en un entorno automatizado, escalable y monitorizable, conforme a los principios del paradigma DevSecOps. La arquitectura ha sido diseñada para garantizar modularidad, trazabilidad, flexibilidad y automatización, permitiendo un flujo de trabajo continuo desde la adquisición y procesamiento de datos hasta la generación de alertas y la retroalimentación del modelo de aprendizaje [8].

Con el fin de poder llegar a alcanzar estos objetivos, la arquitectura se organiza en cuatro capas funcionales interconectadas, que conforman el núcleo operativo del sistema, como se muestra en la figura 2:

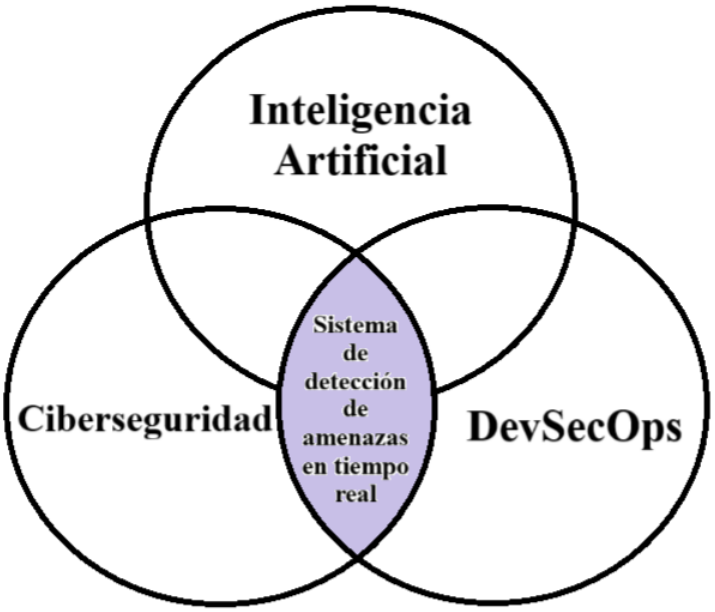

Figura 2. Arquitectura general del sistema de detección de amenazas en tiempo real basado en Inteligencia artificial, Ciberseguridad y DevSecOps. Fuente: Elaboración propia.

La Figura 2 ilustra de forma conceptual la intersección entre los tres pilares fundamentales sobre los que se construye el sistema propuesto: la inteligencia artificial, la ciberseguridad y el paradigma DevSecOps. En el centro de esta intersección se sitúa el sistema de detección de amenazas en tiempo real, que combina capacidades analíticas avanzadas, mecanismos de protección y una infraestructura automatizada y segura. La inteligencia artificial aporta la capacidad de aprendizaje, generalización y detección de comportamientos anómalos; la ciberseguridad define los objetivos de protección, monitorización y respuesta frente a amenazas; mientras que DevSecOps proporciona el marco metodológico que permite integrar de forma continua el desarrollo, la operación y la seguridad del sistema. Esta convergencia garantiza que la detección de amenazas no se realice de manera aislada, sino como parte de un proceso automatizado, trazable y alineado con los requisitos de los entornos de producción modernos.

1. Capa de adquisición de datos (*Data Ingestion Layer*): Se encarga de la captura, filtrado y almacenamiento temporal de los flujos de red y eventos de seguridad procedentes de múltiples fuentes, como son los logs de servidores, tráfico de red, sensores IDS, o *datasets* simulados como CICIDS2017 o UNSW-NB15. Esta capa constituye el punto de entrada del sistema y asegura la integridad y disponibilidad de la información antes de su procesamiento.
2. Capa de procesamiento y análisis (*Processing Layer*): En esta etapa se lleva a cabo el preprocesamiento, transformación y normalización de los datos, garantizando su adecuación al formato requerido por los modelos de aprendizaje. Se incluyen tareas como la limpieza de registros, la codificación de variables y la reducción de dimensionalidad. Su correcta implementación resulta esencial para minimizar errores y optimizar la precisión de los modelos [3].
3. Capa de inteligencia y detección (*AI Layer*): Aquí se implementan los modelos de aprendizaje automático y profundo, entrenados previamente con datos representativos. Su función consiste en identificar comportamientos anómalos o maliciosos en tiempo real, utilizando arquitecturas como, de las que ya he hablado previamente, Random

Forest, Support Vector Machines (SVM) o Redes Neuronales Convolucionales (CNN), según el tipo de análisis requerido. Esta capa actúa como el componente central del sistema de detección.

4. Capa de respuesta y monitorización (*Response & Monitoring Layer*): Gestiona la generación de alertas, la visualización de métricas y la integración con sistemas externos, como plataformas SIEM (*Security Information and Event Management*) o paneles de control basados en Grafana, además de permitir notificaciones automatizadas ante eventos críticos. Su propósito es proporcionar visibilidad en tiempo real y soporte para la toma de decisiones en entornos operativos.

Cada una de estas capas se implementa de manera independiente, pero orquestada mediante un *pipeline* CI/CD, lo que permite automatizar las tareas de entrenamiento, validación, despliegue y actualización de modelos. Este enfoque modular, alineado con los principios de la arquitectura de microservicios y la contenerización, facilita la sustitución o mejora de componentes sin afectar el funcionamiento global del sistema [6], [11].

Desde una perspectiva técnica, el prototipo se apoya en contenedores Docker para encapsular los distintos servicios, garantizando portabilidad, aislamiento y reproducibilidad de los entornos. Los procesos de entrenamiento y predicción se ejecutan en entornos controlados, mientras que la orquestación de despliegues se realiza mediante Jenkins y scripts declarativos en YAML, que automatizan las tareas del pipeline. La comunicación entre módulos se implementa a través de APIs RESTful y colas de mensajería (*message queues*), lo que permite gestionar flujos de información asíncronos y de baja latencia, optimizando el rendimiento del sistema en escenarios de alta concurrencia [1], [11].

Esta arquitectura integra de manera coherente los tres pilares fundamentales del proyecto:

- Inteligencia artificial: detección proactiva mediante modelos entrenados con *datasets* actualizados y representativos.
- Ciberseguridad: monitorización continua, respuesta automatizada y análisis de comportamiento anómalo.
- DevSecOps: automatización del ciclo de vida del software con despliegue seguro, trazabilidad completa y control continuo.

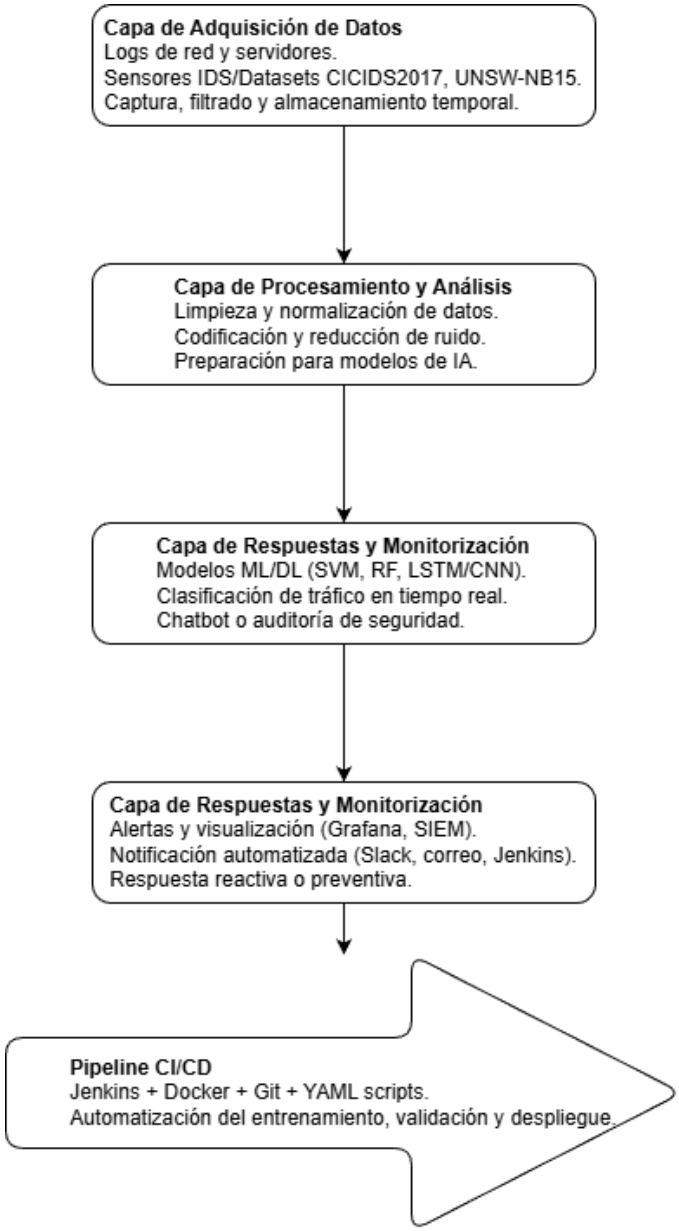

Figura 3. Arquitectura general del sistema de detección de amenazas en tiempo real. Fuente: Elaboración propia.

La Figura 3 representa el flujo operativo completo del sistema de detección de amenazas en tiempo real, mostrando la interacción entre las distintas capas funcionales y su integración dentro del pipeline DevSecOps. El proceso se inicia con la captura de eventos de red y registros de actividad en la capa de adquisición de datos, donde se recopilan flujos de tráfico, logs de servidores y eventos procedentes de sensores IDS o fuentes simuladas. Esta información es filtrada y almacenada de forma temporal, garantizando su integridad antes de ser procesada.

A continuación, los datos capturados se transfieren a la capa de procesamiento y análisis, donde se aplican técnicas de limpieza, normalización y transformación con el objetivo de preparar la información para su análisis por los modelos de inteligencia artificial. En esta fase se eliminan inconsistencias, se codifican variables y se reducen dimensiones innecesarias, asegurando que los datos de entrada sean adecuados y homogéneos.

Una vez procesados, los datos son enviados a la capa de inteligencia y detección, que constituye el núcleo del sistema. En esta capa, los modelos de aprendizaje automático y profundo analizan los flujos de información en tiempo real para identificar patrones anómalos o comportamientos potencialmente maliciosos. El resultado de este análisis es una clasificación de los eventos como legítimos o sospechosos, junto con métricas asociadas al nivel de riesgo detectado.

Cuando se identifica una amenaza, el sistema activa la capa de respuesta y monitorización, donde se generan alertas, se visualizan métricas en paneles de control y se integran los resultados con sistemas externos como plataformas SIEM. Además, esta capa permite la notificación automatizada de incidentes y la ejecución de acciones reactivas o preventivas, proporcionando soporte a la toma de decisiones en entornos operativos.

Todo este flujo se encuentra orquestado por el pipeline de integración y despliegue continuo, que automatiza las tareas de entrenamiento, validación, despliegue y actualización de los modelos de detección. De este modo, cualquier mejora en el código, en los datos o en los modelos puede ser incorporada al sistema de forma controlada y segura, cerrando el ciclo DevSecOps y garantizando una detección de amenazas continua, actualizada y alineada con las necesidades del entorno.

4.2. Componentes principales: adquisición, análisis y respuesta

El sistema propuesto se compone de un conjunto de componentes funcionales interdependientes, diseñados para trabajar de manera sincronizada y garantizar la ejecución eficiente de las tres tareas principales del proceso de detección: adquisición de datos, análisis mediante inteligencia artificial y respuesta automatizada. Cada módulo desempeña una función específica dentro del flujo operativo, lo que permite mantener la modularidad, escalabilidad y continuidad del servicio bajo los principios del enfoque DevSecOps [7], [8].

4.2.1. Módulo de adquisición de datos

El módulo de adquisición de datos constituye el punto de entrada del sistema y tiene como finalidad recopilar, filtrar y gestionar la información de red que servirá de base tanto para el entrenamiento de los modelos como para su operación en tiempo real. Este proceso de ingesta de datos debe garantizar la integridad, coherencia y disponibilidad de la información procedente de múltiples fuentes heterogéneas [3].

Las principales fuentes consideradas en esta arquitectura son:

- *Datasets* simulados, como CICIDS2017 o UNSW-NB15, empleados durante la fase experimental para el entrenamiento y validación inicial de los modelos.
- Flujos de red en tiempo real, capturados mediante herramientas de análisis como Zeek o Wireshark, conectadas a interfaces de red virtualizadas dentro del entorno de pruebas.
- Registros de sistemas IDS, como Snort o Suricata, que proporcionan información contextual sobre eventos de seguridad relevantes.

Para la transmisión de datos en tiempo real se empleará Kafka, seleccionado por su capacidad para manejar flujos continuos con baja latencia y por su integración nativa con arquitecturas distribuidas. De forma complementaria, el sistema utilizará Elasticsearch como

almacenamiento intermedio, dado que permite consultas rápidas, agregaciones en milisegundos y facilita la visualización operativa a través de Kibana. La elección de estas tecnologías evita cuellos de botella durante la fase de adquisición y permite mantener un flujo continuo incluso ante incrementos súbitos del tráfico.

El diseño modular del sistema permite independizar el flujo de adquisición del flujo de inferencia, posibilitando la actualización o sustitución de modelos sin interrumpir la captación de datos. Esta arquitectura se alinea con los principios de alta disponibilidad (*High Availability*, HA) y despliegue continuo (*Continuous Deployment*) promovidos por DevSecOps, garantizando la resiliencia y continuidad del servicio.

4.2.2. Módulo de análisis y detección

El módulo de análisis y detección constituye el núcleo central del sistema, encargado de procesar los flujos de datos para identificar patrones anómalos o maliciosos. Su funcionamiento se basa en la aplicación de modelos de aprendizaje automático y aprendizaje profundo, entrenados previamente con datos representativos y etiquetados.

Durante la fase experimental, se consideran tres modelos principales con el objetivo de comparar su rendimiento en términos de precisión, latencia y eficiencia:

- Random Forest (RF): destaca por su robustez y capacidad de generalización, siendo adecuado para manejar variables de distinta naturaleza y mitigar el sobreajuste.
- Support Vector Machine (SVM): especialmente útil en tareas de clasificación binaria y multiclase en entornos de alta dimensionalidad.
- Redes Neuronales LSTM (Long Short-Term Memory): adecuadas para modelar dependencias temporales dentro de secuencias de tráfico de red, facilitando la detección de comportamientos persistentes o coordinados [9].

El módulo se desarrolla en Python, utilizando *frameworks* como TensorFlow y Scikit-learn, que permiten realizar tanto el entrenamiento local como la inferencia en tiempo real. En el caso del modelo LSTM, se configuran ventanas temporales deslizantes que representan intervalos de tráfico, permitiendo detectar desviaciones respecto al comportamiento considerado normal.

Las métricas de evaluación empleadas incluyen precisión, *recall*, *F1-score* y tiempo medio de inferencia. Estas métricas se calculan automáticamente tras cada iteración del modelo y se registran en Prometheus, lo que posibilita su visualización y análisis en tiempo real mediante paneles de Grafana. Este proceso proporciona una retroalimentación constante que contribuye a la mejora continua de los modelos implementados.

4.2.3. Módulo de respuesta y monitorización

Una vez que el sistema identifica una posible intrusión o comportamiento anómalo, se activa el módulo de respuesta y monitorización, encargado de notificar, registrar y, en su caso, ejecutar medidas automáticas de contención. Este módulo se compone de tres subcomponentes interrelacionados:

1. Generación de alertas: se crean registros de eventos en formato JSON o Syslog, que pueden enviarse a sistemas SIEM (por ejemplo, Wazuh o Elastic Security) o mostrarse directamente en *dashboards* de Grafana para su análisis visual.
2. Notificación automatizada: mediante scripts integrados en Jenkins o a través de servicios REST, el sistema puede enviar alertas por correo electrónico, mensajería instantánea (como Slack) o paneles de control internos, garantizando la comunicación inmediata con los administradores.
3. Respuesta reactiva o preventiva: en configuraciones avanzadas, el módulo puede ejecutar acciones automáticas de contención, como el bloqueo de direcciones IP, el aislamiento de nodos comprometidos o la aplicación dinámica de políticas de cortafuegos.

La integración de este módulo con herramientas de observabilidad como Prometheus y Grafana permite obtener una visión integral del sistema, monitorizando métricas como latencia de detección, rendimiento de los modelos y comportamiento del tráfico. Además, la trazabilidad de eventos y decisiones facilita la mejora iterativa del sistema, ya que los datos detectados pueden reutilizarse en procesos de reentrenamiento automático para optimizar la eficacia de los modelos a lo largo del tiempo [1].

4.3. Pipeline de integración y despliegue continuo (CI/CD)

El desarrollo de un sistema de detección de amenazas basado en inteligencia artificial no se limita únicamente a la creación de modelos de aprendizaje eficaces, sino que requiere de un mecanismo automatizado y trazable que garantice la reproducibilidad, seguridad y eficiencia en todas las fases del ciclo de vida del modelo. En este contexto, el proyecto incorpora un pipeline de Integración y Despliegue Continuo (CI/CD) fundamentado en los principios del paradigma DevSecOps, que permite automatizar las etapas de construcción, prueba, validación, despliegue y monitorización del sistema.

Este *pipeline* constituye la columna vertebral de la arquitectura propuesta, al conectar el desarrollo del código con su despliegue operativo en un flujo continuo e ininterrumpido. Su implementación persigue tres objetivos esenciales: garantizar la coherencia de versiones, mejorar la seguridad del código y reducir los tiempos de entrega mediante la automatización completa del ciclo de vida.

El *pipeline* CI/CD se estructura en cinco etapas principales, que establecen una transición ordenada desde el código fuente hasta la ejecución del sistema en producción.

La elección de un *pipeline* de integración y despliegue continuo como eje central del sistema responde a la necesidad de garantizar un funcionamiento fiable, seguro y escalable en un entorno de detección de amenazas en tiempo real. Frente a enfoques tradicionales basados en despliegues manuales o procesos semi-automatizados, el uso de un pipeline CI/CD permite reducir de forma significativa los errores humanos, asegurar la coherencia entre versiones y acelerar la incorporación de mejoras o correcciones sin interrumpir el servicio. Estas características resultan especialmente relevantes en sistemas de ciberseguridad, donde la rapidez de respuesta y la estabilidad operativa son factores críticos.

Desde el punto de vista de la inteligencia artificial, el pipeline CI/CD facilita la gestión completa del ciclo de vida de los modelos, incluyendo su entrenamiento, validación, versionado

y despliegue controlado. A diferencia de arquitecturas estáticas, este enfoque permite actualizar los modelos de detección de forma continua a medida que se dispone de nuevos datos o se identifican patrones de ataque emergentes, manteniendo así la eficacia del sistema frente a un entorno de amenazas en constante evolución. Esta capacidad de actualización continua resulta fundamental para evitar la obsolescencia de los modelos y minimizar el impacto de ataques desconocidos o de tipo *zero-day*.

La integración del *pipeline* dentro del paradigma DevSecOps permite, además, incorporar controles de seguridad de forma temprana y automatizada en cada una de las fases del proceso. Mediante la ejecución continua de pruebas, análisis de vulnerabilidades y verificaciones de configuración segura, es posible detectar fallos o debilidades antes de que el sistema sea desplegado en producción. Frente a modelos tradicionales donde la seguridad se evalúa de manera reactiva, este enfoque proactivo refuerza la robustez del sistema y reduce el riesgo de exposición en entornos operativos críticos.

Asimismo, el uso de un *pipeline* CI/CD proporciona una trazabilidad completa de los cambios realizados sobre el código, los modelos y la infraestructura, lo que facilita la auditoría, el diagnóstico de incidentes y la recuperación ante fallos. Esta trazabilidad resulta especialmente valiosa en sistemas de detección de amenazas, donde es necesario correlacionar versiones del modelo, configuraciones del sistema y resultados de detección para analizar el comportamiento del sistema ante incidentes concretos.

La adopción de un *pipeline* de integración y despliegue continuo no solo mejora la eficiencia del proceso de desarrollo, sino que constituye un elemento clave para garantizar la fiabilidad, seguridad y capacidad de adaptación del sistema propuesto. Su integración permite alinear el desarrollo del modelo de inteligencia artificial con los principios de automatización, seguridad continua y escalabilidad que exige un sistema moderno de detección de amenazas en tiempo real.

4.3.1. Etapa 1: Control de versiones y gestión del código

El proceso se inicia con la gestión del código fuente y los scripts asociados al entrenamiento, configuración y despliegue, los cuales se almacenan en un repositorio Git (por ejemplo, el de GitHub o GitLab). Cada *commit* o actualización en el repositorio activa automáticamente la ejecución del *pipeline* a través de herramientas de automatización como Jenkins o GitHub Actions.

Este enfoque garantiza la trazabilidad de los cambios, la gestión de ramas de desarrollo y la revisión por pares (*peer review*) antes de la integración en la rama principal (*main branch*). De esta manera, se asegura la calidad del código, la transparencia de los cambios y la detección temprana de vulnerabilidades, en línea con las buenas prácticas de DevSecOps [11].

4.3.2. Etapa 2: Construcción y empaquetamiento del sistema

Una vez confirmados los cambios en el repositorio, se inicia la fase de construcción del entorno de ejecución. Esta etapa utiliza archivos Dockerfile para crear imágenes Docker independientes correspondientes a los distintos módulos del sistema:

- Módulo de adquisición de datos, encargado de la captura y almacenamiento temporal del tráfico de red.
- Módulo de análisis y detección,que contiene las dependencias de TensorFlow, Scikit-learn y bibliotecas de procesamiento en Python.
- Módulo de visualización y monitorización, integrado con Prometheus y Grafana.
- Orquestador y *scripts* de automatización, gestionados mediante Jenkins.

Estas imágenes se almacenan en un registro privado de contenedores (por ejemplo, Docker Hub, AWS Elastic Container Registry o GitLab Container Registry), lo que permite su reutilización y despliegue reproducible en diferentes entornos sin conflictos de dependencias [6], [11].

4.3.3. Etapa 3: Pruebas automáticas y validación de seguridad

Antes del despliegue, el pipeline ejecuta una batería de pruebas automatizadas destinadas a validar la integridad, seguridad y rendimiento del sistema. Estas pruebas incluyen:

- Tests unitarios: verifican el correcto funcionamiento de las funciones críticas, como la lectura de datos, la inferencia del modelo o la conexión con las APIS.
- Tests de integración: comprueban la comunicación entre los distintos servicios y contenedores (por ejemplo, Kafka, Jenkins o Prometheus).
- Análisis estático y de vulnerabilidades: ejecutado mediante herramientas como SonarQube, OWASP Dependency-Check o Snyk, que detectan dependencias inseguras, código vulnerable o malas prácticas de programación.

A mayores, se realizan pruebas de rendimiento del modelo de IA, evaluando la latencia de inferencia y el uso de recursos. Si alguna validación no cumple los criterios predefinidos, el pipeline se detiene automáticamente, evitando el despliegue de versiones defectuosas o inseguras [1], [11].

4.3.4. Etapa 4: Despliegue automatizado

Una vez verificados los componentes, Jenkins ejecuta los pipelines de despliegue definidos mediante archivos YAML o Jenkinsfile, según el entorno de ejecución. El sistema contempla tres entornos diferenciados:

- Entorno de desarrollo (DEV): permite realizar pruebas unitarias e integraciones iniciales.
- Entorno de pruebas (STAGING): evalúa el rendimiento y la seguridad bajo condiciones de carga simulada.
- Entorno de producción (PROD): despliega el sistema en modo operativo con monitorización activa y recolección de métricas.

El despliegue se gestiona mediante Docker Compose o Kubernetes, que facilitan la escalabilidad horizontal, el aislamiento de servicios y la recuperación ante fallos. Este enfoque garantiza alta disponibilidad (*High Availability*) y resiliencia operativa, aspectos fundamentales en sistemas de detección en tiempo real [6], [11].

4.3.5. Etapa 5: Monitorización, retroalimentación y reentrenamiento

La etapa final del *pipeline* contempla la monitorización continua del rendimiento tanto del sistema como de los modelos de IA desplegados. Las métricas generadas por Prometheus, como el uso de CPU, latencia, precisión o tasa de detección, se visualizan en paneles de Grafana, proporcionando una visión global y actualizada del estado del sistema.

Además, se incluye un mecanismo de retroalimentación automática, que recopila los casos de falsos positivos y falsos negativos detectados durante la operación. Estos datos se utilizan posteriormente en el proceso de reentrenamiento del modelo, asegurando que la IA mantenga su precisión y capacidad de adaptación en entornos dinámicos y no estacionarios, mitigando el efecto del *concept drift* [9].

En síntesis de estas etapas, el pipeline CI/CD actúa como el eje vertebrador del enfoque DevSecOps adoptado en este trabajo, al combinar automatización, seguridad y monitorización continua dentro de un ciclo de mejora iterativa. Gracias a este mecanismo, el sistema garantiza un proceso de desarrollo repetible, auditable y seguro, minimizando la intervención manual y reduciendo significativamente el riesgo de errores en entornos de producción [1].

4.4. Infraestructura de contenedorización y orquestación

El sistema propuesto se apoya en una infraestructura basada en contenedores con el objetivo de garantizar la portabilidad, escalabilidad y aislamiento de los distintos componentes que lo integran. Para ello, se emplea la tecnología de contenedorización Docker, que permite encapsular cada módulo del sistema, incluidos los servicios de adquisición de datos, preprocesamiento, modelos de inteligencia artificial y generación de alertas, junto con sus dependencias y configuraciones específicas. Este enfoque facilita la reproducibilidad de los entornos de ejecución y evita inconsistencias entre los entornos de desarrollo, pruebas y producción.

La elección de Docker frente a alternativas tradicionales como máquinas virtuales completas responde principalmente a criterios de eficiencia y rendimiento. Los contenedores presentan un menor consumo de recursos y un tiempo de arranque significativamente inferior, lo que resulta especialmente relevante en sistemas de detección de amenazas en tiempo real, donde la latencia y la capacidad de respuesta son factores críticos. Además, el uso de imágenes versionadas permite controlar de forma precisa los cambios introducidos en cada componente del sistema, facilitando la gestión de actualizaciones y la reversión a estados anteriores en caso de fallo.

Sobre esta base de contenedorización, el sistema integra un mecanismo de orquestación mediante Kubernetes, encargado de gestionar el despliegue, escalado y supervisión de los distintos servicios. Kubernetes permite distribuir la carga de trabajo, reiniciar automáticamente contenedores en caso de fallo y escalar horizontalmente los módulos más exigentes, como el motor de detección basado en inteligencia artificial, en función de la carga de tráfico o del volumen de eventos procesados. Esta capacidad de escalado dinámico resulta fundamental en entornos corporativos modernos, donde el tráfico de red puede variar de forma significativa y no siempre predecible.

Asimismo, la orquestación facilita la separación lógica de los distintos componentes del sistema, mejorando el aislamiento y reduciendo el impacto de posibles fallos o compromisos de seguridad. La combinación de Docker y Kubernetes proporciona, por tanto, una infraestructura robusta y flexible, alineada con los principios de DevSecOps y adecuada para el despliegue de sistemas de detección de amenazas que requieren alta disponibilidad, resiliencia y capacidad de adaptación a entornos cambiantes.

4.5. Integración del diseño dentro del paradigma DevSecOps

El diseño del sistema propuesto se concibe desde una perspectiva DevSecOps, integrando la seguridad como un elemento transversal y continuo a lo largo de todo el ciclo de vida del desarrollo. Esta integración no se limita a la fase de despliegue, sino que abarca desde el diseño inicial de la arquitectura hasta la operación y monitorización del sistema en producción, garantizando un enfoque de seguridad proactivo y automatizado.

Dentro de este paradigma, cada uno de los componentes descritos en los apartados anteriores se integra en un flujo continuo gestionado por el pipeline CI/CD. El código fuente, los modelos de inteligencia artificial y las configuraciones de infraestructura son sometidos de forma automática a procesos de validación, pruebas y análisis de seguridad antes de su despliegue. Este enfoque permite detectar errores, vulnerabilidades o configuraciones inseguras en fases tempranas, reduciendo el riesgo de exposición y los costes asociados a correcciones tardías.

La infraestructura basada en contenedores y orquestación refuerza este modelo al facilitar la aplicación de controles de seguridad consistentes y repetibles. El uso de imágenes versionadas, entornos aislados y despliegues declarativos permite auditar cada cambio introducido en el sistema y garantizar que las versiones desplegadas cumplen con los requisitos de seguridad establecidos. Además, la monitorización continua de los servicios desplegados proporciona información en tiempo real sobre el estado del sistema, el rendimiento de los modelos de detección y la aparición de posibles incidentes.

Desde el punto de vista operativo, la integración de DevSecOps permite mantener los modelos de inteligencia artificial actualizados sin interrumpir el servicio, incorporando nuevas versiones de forma controlada y reversible. Esta capacidad resulta especialmente relevante en sistemas de detección de amenazas, donde la adaptación a nuevas tipologías de ataque y la mejora continua de los modelos son factores clave para mantener la eficacia del sistema.

El diseño propuesto combina inteligencia artificial, automatización y prácticas DevSecOps para ofrecer una solución de detección de amenazas en tiempo real que es segura por diseño, escalable y preparada para entornos de producción exigentes. Este enfoque integral garantiza la coherencia entre las decisiones de diseño adoptadas y los objetivos de seguridad, rendimiento y fiabilidad establecidos para el sistema.

Capítulo 5: Detalles de la implementación

5.1. Selección de herramientas

La selección de herramientas tecnológicas constituye un aspecto esencial en el desarrollo del sistema, ya que determina su rendimiento, compatibilidad, mantenibilidad y capacidad de automatización**.** En este trabajo de fin de grado, la elección se ha basado, en lo que yo pienso, en criterios de eficiencia técnica, soporte comunitario, flexibilidad y alineación con el paradigma DevSecOps, asegurando la integración fluida entre las fases de desarrollo, despliegue y monitorización.

Cada tecnología seleccionada desempeña un rol específico dentro de la arquitectura modular del sistema**,** contribuyendo al cumplimiento de tres objetivos fundamentales: la automatización de procesos, la implementación de inteligencia artificial y la observabilidad continua del entorno [6], [11].

La selección de estas herramientas no responde únicamente a su popularidad o disponibilidad, sino a su adecuación específica para un sistema de detección de amenazas en tiempo real. Python permite una integración directa entre el procesamiento de datos, el entrenamiento de modelos y su despliegue en producción; Docker y Kubernetes garantizan aislamiento, portabilidad y escalabilidad; mientras que Jenkins y las herramientas de monitorización permiten automatizar y supervisar el comportamiento del sistema de forma continua. Frente a alternativas más monolíticas o dependientes de infraestructuras propietarias, esta combinación proporciona un mayor control, flexibilidad y alineación con los principios del paradigma DevSecOps, aspectos clave para la fiabilidad y evolución del sistema propuesto.

La siguiente Figura 4 resume las principales tecnologías para desarrollo, despliegue, monitorización y análisis de seguridad del sistema.

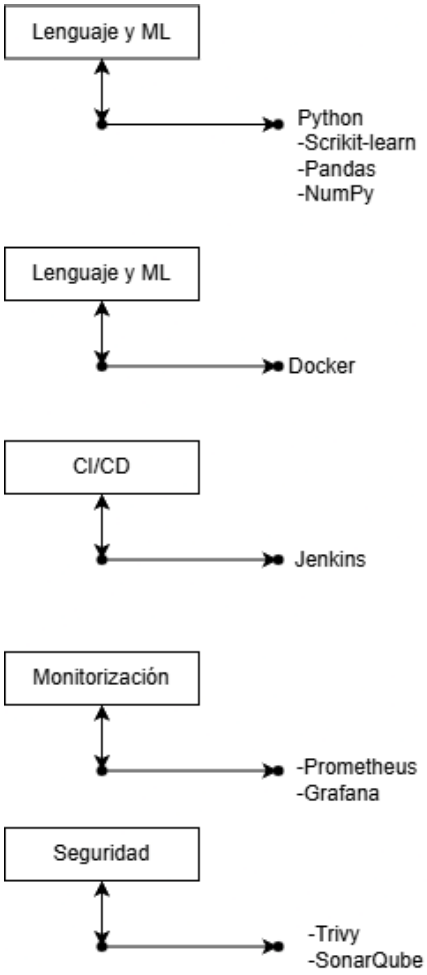

Figura 4. Herramientas empleadas en el entorno experimental del sistema. Elaboración propia.

5.1.1. Lenguaje de programación y entorno de desarrollo

El lenguaje principal empleado es Python, debido a su versatilidad, sintaxis accesible y amplio ecosistema de librerías orientadas a la ciencia de datos y la inteligencia artificial. Su compatibilidad con herramientas DevOps y su integración con APIs y contenedores lo convierten en una opción idónea para el desarrollo de soluciones híbridas de ciberseguridad e inteligencia artificial [3].

Entre los principales entornos y librerías utilizadas destacan:

- TensorFlow y Keras: para la implementación y entrenamiento de redes neuronales profundas, incluidas arquitecturas LSTM orientadas a la detección de secuencias anómalas en el tráfico de red.
- Scikit-learn: utilizada para algoritmos clásicos de clasificación, como Random Forest o SVM, que proporcionan una base comparativa frente a los modelos de aprendizaje profundo.

- Pandas y NumPy: orientadas a la manipulación, limpieza y análisis de grandes volúmenes de datos estructurados, permitiendo preparar eficientemente los conjuntos de entrenamiento.
- Flask o FastAPI: frameworks ligeros empleados para la exposición de *endpoints* REST, lo que posibilita la comunicación entre el modelo y otros componentes del sistema o servicios externos.

Dicho esto, la elección de Python ofrece la flexibilidad necesaria para desarrollar tanto las fases de entrenamiento como las de inferencia en producción, integrándose de forma natural con los pipelines CI/CD definidos bajo la metodología DevSecOps [9].

5.1.2. Contenedorización y orquestación

El sistema se implementa en entornos contenedorizados mediante Docker, lo que proporciona portabilidad, aislamiento y control de dependencias entre los diferentes módulos. Cada componente (adquisición, análisis, detección y monitorización) se despliega como un contenedor independiente, comunicándose con los demás a través de APIs REST o colas de mensajería (Kafka o RabbitMQ).

Este enfoque modular facilita la actualización o sustitución de componentes sin afectar al sistema global, mejorando la mantenibilidad y la resiliencia. En entornos de mayor complejidad o escalabilidad, la orquestación de contenedores puede gestionarse mediante Kubernetes, lo que permite ajustar dinámicamente el número de instancias activas en función de la carga del sistema y garantizar una alta disponibilidad (HA) [6], [11].

5.1.3. Integración y despliegue continuo

Para la gestión del *pipeline* CI/CD, se ha seleccionado Jenkins, una de las herramientas más consolidadas en el ámbito DevOps por su capacidad de automatizar tareas repetitivas, coordinar procesos y ejecutar *pipelines* definidos de forma declarativa.

Jenkins permite:

- Automatizar la construcción, validación y despliegue del código, garantizando consistencia entre entornos.
- Integrarse con sistemas de control de versiones (Git) y registros de contenedores (Docker Hub, GitLab Registry o AWS ECR).
- Configurar pipelines complejos mediante scripts YAML o Jenkinsfile, asegurando trazabilidad y reproducibilidad en el proceso de despliegue.

De manera alternativa, en entornos de nube, pueden utilizarse GitLab CI/CD o GitHub Actions, que ofrecen funcionalidades equivalentes y una integración nativa con sus repositorios, reduciendo la necesidad de infraestructura adicional [6].

5.1.4. Monitorización y observabilidad

La monitorización continua constituye un componente crítico en el diseño del sistema, permitiendo evaluar tanto el rendimiento de los modelos como la salud operativa de los

servicios. Para este fin se ha optado por el conjunto de herramientas Prometheus + Grafana, ampliamente reconocido en entornos de producción de alta disponibilidad.

- Prometheus se encarga de recopilar y almacenar métricas del sistema y del modelo de IA, tales como uso de CPU, memoria, tasa de detección, precisión o latencia media de inferencia.
- Grafana proporciona una interfaz visual interactiva para la visualización de métricas en tiempo real, mediante paneles personalizados que facilitan la supervisión y la toma de decisiones.

Adicionalmente, se contempla la integración con el *stack* ELK (Elasticsearch, Logstash y Kibana), que permite indexar, analizar y visualizar registros de eventos. Esta combinación de herramientas mejora la observabilidad y trazabilidad del sistema, aspectos clave dentro del enfoque DevSecOps [1], [11].

5.1.5. Seguridad y análisis de vulnerabilidades

De acuerdo con los principios del DevSecOps, la seguridad debe incorporarse de manera continua y automatizada en todo el ciclo de vida del *software*. Por este mismo motivo, para ello se integran herramientas que permiten detectar vulnerabilidades, evaluar dependencias y auditar el código fuente antes de su despliegue:

- SonarQube: realiza análisis estático del código fuente, detectando malas prácticas, vulnerabilidades y posibles defectos lógicos.
- Trivy o Clair: ejecutan escaneos de seguridad sobre las imágenes Docker, identificando paquetes desactualizados o dependencias con vulnerabilidades conocidas.
- OWASP Dependency-Check: analiza las bibliotecas utilizadas en el proyecto, verificando su correspondencia con bases de datos de vulnerabilidades (CVE).

Estas herramientas se ejecutan automáticamente en la etapa de validación del pipeline CI/CD, garantizando que solo los artefactos verificados, seguros y auditables lleguen al entorno de producción [1], [11].

5.2. Descripción de los *datasets*

El correcto funcionamiento de un sistema de detección de intrusiones basado en inteligencia artificial depende de manera directa de la calidad, representatividad y equilibrio del conjunto de datos utilizado durante las fases de entrenamiento, validación y prueba. La capacidad de un modelo para generalizar y reconocer comportamientos anómalos en entornos reales está determinada, en gran medida, por la riqueza y diversidad de los datos que conforman su base de aprendizaje [9].

En el contexto del presente proyecto, se han seleccionado dos conjuntos de datos de referencia internacional ampliamente utilizados en la investigación académica y aplicada en ciberseguridad: CICIDS2017 y UNSW-NB15. Ambos *datasets* destacan por su estructura detallada, amplitud de escenarios y relevancia en la detección de ataques modernos, lo que los convierte en recursos idóneos para el entrenamiento de modelos de aprendizaje automático y profundo orientados a la detección de intrusiones.

5.2.1. *Dataset* CICIDS2017

El conjunto de datos CICIDS2017, desarrollado por el Canadian Institute for Cybersecurity, representa una de las fuentes más completas y realistas para el estudio de flujos de tráfico de red corporativo y la identificación de ciberataques contemporáneos [30].

Este *dataset* fue diseñado a partir de un entorno de red simulado que emula un ecosistema empresarial, incluyendo múltiples usuarios, servidores, servicios web y aplicaciones. Se registran tanto actividades legítimas (como lo son la navegación web, la transferencia de archivos o las conexiones seguras) como comportamientos maliciosos generados por *scripts* y herramientas de ataque reales.

Entre las categorías de ataque incluidas se encuentran:

- Denial of Service (DoS) y Distributed DoS (DDoS), destinados a saturar los recursos de red.
- Brute Force sobre protocolos SSH y FTP, que buscan el acceso no autorizado.
- Infiltration y Botnet, orientados al control remoto de sistemas comprometidos.
- Web Attacks, que incluyen Cross-Site Scripting (XSS), SQL Injection **y** Command Injection.
- Port Scanning y Reconnaissance, empleados para identificar servicios vulnerables.

El conjunto CICIDS2017 contiene más de 80 características (*features*) por flujo de conexión, entre las que se incluyen métricas estadísticas de tráfico, duración de sesiones, número de paquetes transmitidos, volumen de bytes enviados y recibidos, tasas de conexión, banderas TCP y otros atributos derivados. Estas variables permiten representar el comportamiento del tráfico tanto a nivel de paquete como de sesión, lo que resulta especialmente adecuado para el entrenamiento de modelos de aprendizaje profundo, como las redes LSTM (Long Short-Term Memory) o las redes convolucionales (CNN), que analizan dependencias temporales y espaciales del tráfico.

Además, la organización temporal de los datos permite reproducir secuencias cronológicas de eventos, lo cual es esencial para la detección de ataques persistentes o coordinados. Por estas razones, CICIDS2017 es considerado uno de los *datasets* más completos para el estudio de detección en tiempo real y para la validación de arquitecturas híbridas de IA en ciberseguridad [1], [31].

5.2.2. *Dataset* UNSW-NB15

Como complemento al anterior, se ha empleado también el conjunto de datos UNSW-NB15, desarrollado por Moustafa y Slay (2015) en el Australian Centre for Cyber Security**.** Este *dataset* fue concebido con el propósito de superar las limitaciones de los conjuntos tradicionales como KDD Cup 99 o NSL-KDD, incorporando tráfico real y sintético capturado en un entorno mixto de red, donde se combinan escenarios normales con múltiples tipos de ataque.

El conjunto incluye nueve categorías principales de amenazas, entre ellas:

- Exploits (aprovechamiento de vulnerabilidades),

- Fuzzers (ataques basados en inputs aleatorios),
- Generic (ataques de fuerza bruta generalizados),
- Denial of Service (DoS),
- Reconnaissance (escaneo de red y reconocimiento),
- Shellcode y Worms.

El *dataset* contiene 49 características, clasificadas en tres grandes grupos:

- Atributos básicos del flujo: dirección IP de origen y destino, puertos, protocolo y duración de la conexión.
- Atributos derivados: tasas de paquetes, tiempos de respuesta y medidas estadísticas de rendimiento.
- Atributos basados en contenido: número de bytes enviados, flags, fragmentación y conteo de errores.

Esta estructura tabular, junto con la presencia de etiquetas binarias ("normal" / "malicioso"), facilita el entrenamiento de modelos supervisados como Random Forest o SVM, así como la comparación directa entre distintos algoritmos de clasificación. Además, su equilibrio entre clases y su riqueza de atributos hacen de UNSW-NB15 un complemento ideal para la validación cruzada con otros *datasets*, contribuyendo a reducir el riesgo de sobreajuste (*overfitting*) y mejorando la capacidad de generalización del modelo [30], [33].

5.2.3. División de los datos

Con el fin de garantizar la robustez y capacidad de generalización de los modelos de inteligencia artificial, los datos se dividen en tres subconjuntos siguiendo las proporciones más aceptadas en la literatura:

- 70% del total para entrenamiento, permitiendo que el modelo aprenda patrones relevantes.
- 15% para validación, empleada en el ajuste de hiperparámetros y la detección de sobreajuste.
- 15% para pruebas, destinada a la evaluación final del modelo sobre datos no vistos.

Esta división asegura una evaluación equilibrada y reproducible del rendimiento, evitando que el modelo dependa excesivamente de un subconjunto específico de datos.

Los *datasets* se almacenan en un sistema de archivos estructurado dentro del entorno de desarrollo y se cargan dinámicamente en los contenedores Docker durante la fase de entrenamiento. Este procedimiento garantiza la trazabilidad, portabilidad y reproducibilidad del proceso dentro del *pipeline* CI/CD, lo que resulta coherente con los principios del paradigma DevSecOps [6], [11].

La utilización combinada de CICIDS2017 y UNSW-NB15 proporciona una base sólida para entrenar y validar modelos capaces de detectar comportamientos anómalos de distinta naturaleza y complejidad, asegurando una visión integral de las amenazas y una mayor fiabilidad en los resultados experimentales.

5.3. Estrategia de preprocesamiento y normalización de datos

El preprocesamiento de los datos constituye una etapa crítica en cualquier proyecto de detección de intrusiones mediante inteligencia artificial, ya que de esta fase depende en gran medida la calidad del modelo, la estabilidad del entrenamiento y la fiabilidad de las predicciones. Los datos de tráfico de red suelen presentar valores faltantes, duplicados, ruido, redundancia, desequilibrio entre clases y escalas heterogéneas, lo que puede comprometer la capacidad de generalización de los algoritmos si no se trata adecuadamente [9].

El proceso de preprocesamiento implementado en este proyecto se ha diseñado para maximizar la calidad del *dataset* sin alterar su representatividad estadística, preservando las relaciones entre atributos y garantizando la reproducibilidad dentro del *pipeline* CI/CD. Este procedimiento se estructura en cinco fases principales, cada una orientada a resolver un tipo específico de problema de los datos.

5.3.1. Limpieza y depuración de datos

La primera fase consiste en la limpieza y depuración del *dataset*, eliminando registros incompletos, duplicados o inconsistentes que puedan introducir sesgos en el proceso de entrenamiento. Los valores faltantes (NaN) se imputan utilizando estrategias estadísticas simples, en función del tipo de variable:

- Para atributos numéricos, se reemplazan por la media o mediana del conjunto de datos.
- Para atributos categóricos, se sustituyen por la moda o categoría más frecuente.

Estas operaciones se ejecutan con las librerías Pandas y NumPy, garantizando la integridad estructural del *dataset* sin alterar su distribución original. Además, se aplican verificaciones automáticas de consistencia mediante *scripts* definidos en el *pipeline* CI/CD, lo que asegura que el proceso sea repetible y auditable.

5.3.2. Selección de características (*Feature Selection*)

El *dataset* CICIDS2017 contiene más de 80 características, muchas de las cuales pueden ser irrelevantes o redundantes para la tarea de detección. El exceso de atributos no solo aumenta el tiempo de cómputo, sino que también puede introducir ruido y sobreajuste (*overfitting*) en los modelos.

Por esto mismo, se aplica una reducción sistemática de características mediante tres técnicas complementarias:

- Correlación de Pearson: elimina variables altamente correlacionadas entre sí, reteniendo solo una de cada grupo redundante.
- Análisis de Varianza (ANOVA): permite determinar qué características presentan una relación estadísticamente significativa con la variable objetivo (etiqueta de clase).
- Importancia de características con Random Forest: se evalúa la contribución individual de cada atributo al proceso de clasificación, seleccionando aquellas variables con mayor peso en la decisión del modelo.

Esta combinación de métodos proporciona un conjunto óptimo de características, reduciendo la dimensionalidad y mejorando tanto la eficiencia computacional como la interpretabilidad de los resultados [3].

5.3.3. Codificación de atributos categóricos

En los *datasets* de tráfico de red, ciertos atributos (por ejemplo, como el protocolo de comunicación, el tipo de servicio o las banderas TCP) se presentan en formato categórico. Dado que los algoritmos de aprendizaje automático requieren entradas numéricas, estas variables se transforman mediante técnicas de codificación (*encoding*).

- En la mayoría de los casos, se aplica One-Hot Encoding, que genera una representación binaria independiente para cada categoría, evitando la introducción de relaciones ordinales inexistentes.
- Cuando el número de categorías es elevado, se utiliza Label Encoding, que asigna un identificador numérico único a cada valor categórico, optimizando el consumo de memoria y el tiempo de procesamiento.

Estas transformaciones se realizan con las funciones nativas de Scikit-learn, garantizando compatibilidad con los modelos implementados y homogeneidad en los formatos de entrada.

5.3.4. Normalización y escalado

Una vez completada la codificación, se procede a la normalización y escalado de los atributos numéricos. Este paso es esencial, ya que muchos algoritmos, especialmente los basados en distancia o gradiente, son sensibles a la magnitud de las variables.

Para ello, se aplican dos métodos principales:

- Min -Max Scaling, que ajusta los valores al rango [0, 1], preservando las proporciones relativas entre observaciones.
- StandardScaler, que transforma los datos para que tengan media 0 y desviación típica 1, mejorando la convergencia del entrenamiento en modelos de redes neuronales y SVM.

La normalización contribuye a una distribución homogénea de los datos, reduce el sesgo hacia atributos de gran escala y mejora la estabilidad numérica de los cálculos durante la fase de aprendizaje [9].

5.3.5. Balanceo de clases

Una característica común en los *datasets* de detección de intrusiones es el desequilibrio entre clases, donde los registros de tráfico normal superan ampliamente a los maliciosos. Este fenómeno puede inducir a los modelos a favorecer las predicciones mayoritarias, reduciendo su capacidad para identificar ataques reales [1].

Para mitigar este problema, se aplican técnicas de *resampling* que equilibran la proporción entre clases:

- SMOTE (Synthetic Minority Over-sampling Technique): genera nuevas muestras sintéticas de la clase minoritaria mediante interpolación, mejorando la sensibilidad del modelo hacia los ataques poco frecuentes.
- Random Under-Sampling: elimina aleatoriamente una parte de las instancias de la clase mayoritaria para equilibrar la distribución sin incrementar el volumen total de datos.

El uso combinado de ambas técnicas garantiza un equilibrio adecuado entre precisión y *recall*, reduciendo los falsos negativos y mejorando la capacidad de generalización del modelo.

Por último, los datos preprocesados se exportan en formato CSV o Parquet y se versionan dentro del repositorio del proyecto, manteniendo una trazabilidad completa. Durante la ejecución del pipeline CI/CD, estos archivos se cargan automáticamente en los contenedores Docker que ejecutan las fases de entrenamiento y validación, asegurando un flujo reproducible y auditado conforme a los principios del enfoque DevSecOps [1], [11].

5.4. Diseño del flujo de detección en tiempo real

El flujo de detección en tiempo real constituye el núcleo operativo del sistema propuesto, donde confluyen todos los módulos previamente descritos para materializar un proceso continuo de adquisición, análisis e interpretación de eventos de red. Este flujo representa la integración efectiva entre la inteligencia artificial, la automatización DevSecOps y las tecnologías de monitorización, garantizando una respuesta proactiva y escalable ante posibles intrusiones [9].

El diseño de este flujo se fundamenta en una arquitectura distribuida y basada en *streaming*, que permite el tratamiento de grandes volúmenes de datos con baja latencia. El sistema no solo detecta anomalías en tiempo real, sino que también aprende de sus errores gracias a su mecanismo de retroalimentación continua, manteniendo así su precisión y relevancia a lo largo del tiempo.

5.4.1. Descripción general del flujo

El proceso completo se estructura en cuatro fases principales, que permiten la transición ordenada de los datos desde su origen hasta la visualización y notificación final.

1. Captura y transmisión de datos: los flujos de red se capturan en tiempo real mediante herramientas como Zeek o Wireshark, que monitorizan el tráfico en las interfaces de red virtualizadas. Estos datos se transmiten hacia el módulo de análisis a través de Apache Kafka, que actúa como *message broker*, gestionando la entrega garantizada, la persistencia temporal y la tolerancia a fallos en la transmisión. Kafka desempeña un papel crucial en el desacoplamiento entre productores y consumidores de datos, asegurando la escalabilidad del sistema y permitiendo que los distintos módulos funcionen de manera asíncrona sin depender unos de otros [3].
2. Preprocesamiento en *streaming*: antes de que los datos sean analizados, se ejecuta un preprocesamiento en tiempo real mediante scripts en Python integrados con Apache Spark Streaming. Este paso normaliza las características entrantes, manteniendo su coherencia con las empleadas durante el entrenamiento del modelo. Se aplican operaciones de limpieza, escalado y transformación en micro-lotes, lo que permite reducir la latencia total del sistema sin sacrificar la integridad estadística del flujo.

Gracias a esta capa, el sistema puede procesar datos de forma continua y paralela, preservando la consistencia entre el entorno experimental y el entorno productivo.

3. Inferencia del modelo de inteligencia artificial: en esta etapa, el modelo entrenado (por ejemplo, una red neuronal LSTM) se despliega en un contenedor Docker independiente, configurado para recibir flujos de datos a través de Kafka o una API REST. Los datos son procesados en micro-lotes o *streams* y clasificados como "normales" o "anómalos" en función de las probabilidades generadas por el modelo. El resultado de cada predicción se transmite al módulo de respuesta, acompañado de metadatos como la marca temporal, el identificador de flujo y el nivel de criticidad. Este enfoque modular permite escalar el número de instancias del modelo de inferencia según la carga del sistema y facilita el procesamiento paralelo en tiempo real [9].
4. Gestión de alertas y visualización: los resultados del análisis se envían al sistema de monitorización Prometheus + Grafana, donde se registran métricas como tasa de detección, latencia promedio, precisión y número de eventos por categoría. Las alertas de alta criticidad generan notificaciones automáticas mediante integraciones con Jenkins, Slack, correo electrónico o APIs REST conectadas a plataformas SIEM como Wazuh o Elastic Security. Esta integración garantiza una respuesta inmediata y centralizada ante incidentes de seguridad, en consonancia con los principios de automatización y observabilidad del paradigma DevSecOps [1].

5.4.2. Mecanismo de retroalimentación

Uno de los componentes más relevantes del sistema es su mecanismo de retroalimentación automática, diseñado para mejorar continuamente el modelo de IA a partir de los resultados operativos. Los registros clasificados erróneamente, tanto falsos positivos como falsos negativos, se almacenan en una base de datos dedicada que sirve como fuente de datos para el *pipeline* de reentrenamiento.

Durante las ejecuciones programadas del *pipeline* CI/CD, estos registros se integran con la base de datos de entrenamiento, generando un nuevo conjunto de datos ampliado y actualizado.Este proceso permite reentrenar automáticamente los modelos y ajustar sus parámetros sin intervención manual, asegurando que el sistema se adapte progresivamente a nuevos patrones de ataque y a los cambios en la dinámica del tráfico de red [1], [11].

De esta forma, el modelo evoluciona de manera adaptativa y autosuficiente, mejorando su capacidad predictiva y reduciendo los errores a lo largo del tiempo, un principio esencial dentro del enfoque de seguridad continúa promovido por DevSecOps.

5.4.3. Trazabilidad y auditoría

La trazabilidad del proceso y la auditoría de decisiones son elementos esenciales para garantizar la transparencia y la confiabilidad del sistema. Cada decisión, desde la adquisición de un paquete, se registra en logs estructurados en formato JSON, que son indexados y almacenados en Elasticsearch.

Estos registros incluyen información contextual, como:

- El identificador del modelo y su versión.
- Los hashes de las imágenes Docker empleadas.

- Los metadatos del *pipeline* CI/CD (fecha, *commit*, versión de dependencias).

Este nivel de trazabilidad permite reconstruir el estado del sistema ante un incidente de seguridad, facilitando auditorías posteriores, validación de resultados y análisis forense. Además, proporciona un marco de cumplimiento normativo y control de versiones, aspectos fundamentales en entornos de producción sensibles [1].

5.4.4. Descripción conceptual del diagrama

El diagrama de flujo general, como se puede ver en la Figura 5, representa visualmente la interacción entre los componentes que integran el sistema de detección en tiempo real. El flujo se organiza en dos canales principales:

- Canal operativo: Fuente de datos → Preprocesamiento → Modelo de IA → Sistema de alertas → Panel de observabilidad. Este circuito constituye el flujo principal de procesamiento, desde la captura del tráfico hasta la visualización y gestión de alertas.
- Canal de retroalimentación: Eventos clasificados → Almacenamiento → Pipeline CI/CD → Reentrenamiento del modelo.

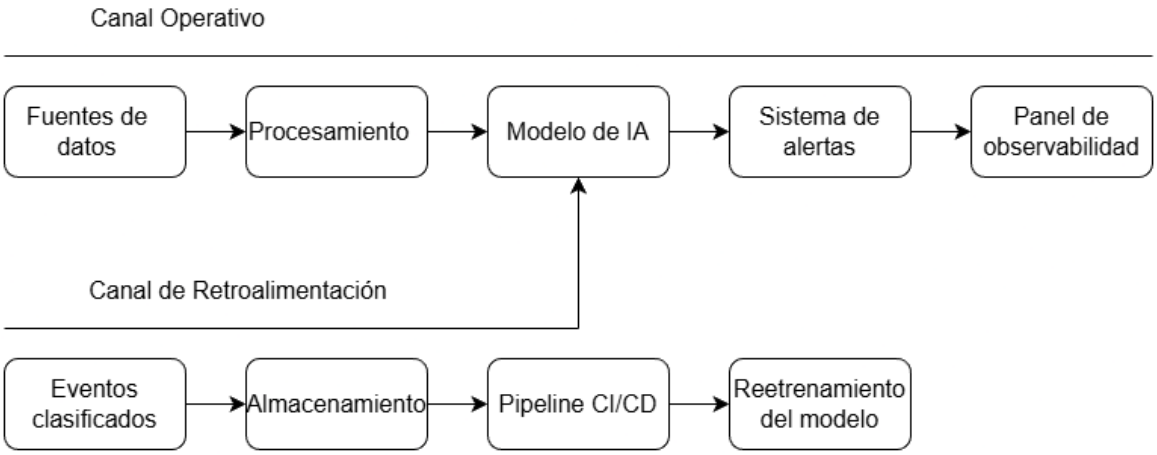

Figura 5. Diagrama de flujo general del sistema de detección de amenazas en tiempo real. Fuente: Elaboración propia.

Este canal complementario, representado en la Figura 5, garantiza la mejora continua del sistema mediante la integración automatizada de nuevos datos en los ciclos de entrenamiento.

El flujo de detección diseñado combina procesamiento en *streaming*, análisis inteligente y automatización DevSecOps, ofreciendo un entorno modular, escalable y resiliente, capaz de responder de manera adaptativa a amenazas dinámicas y evolucionar de forma autónoma a lo largo del tiempo.

5.5. Configuración y selección de los modelos de inteligencia artificial

La configuración adecuada de los modelos de aprendizaje automático y profundo constituye un factor determinante en el rendimiento final del sistema de detección. En este proyecto se han evaluado distintos enfoques de clasificación con el objetivo de identificar un equilibrio óptimo entre precisión, capacidad de generalización y latencia de inferencia, aspectos críticos en un sistema de detección en tiempo real.

Para los modelos supervisados clásicos, como Random Forest y Support Vector Machines (SVM), se han ajustado hiperparámetros clave que influyen directamente en su comportamiento. En el caso de Random Forest, se ha configurado un número de árboles comprendido entre 100 y 300, evaluando su impacto en la estabilidad del modelo y en la reducción del sobreajuste. Asimismo, se ha limitado la profundidad máxima de los árboles para evitar una especialización excesiva en los datos de entrenamiento. Para SVM, se han probado distintos valores del parámetro de regularización (C) y diferentes funciones kernel, priorizando configuraciones que ofrecieran un buen compromiso entre precisión y coste computacional.

En el caso de los modelos de aprendizaje profundo, especialmente las redes LSTM orientadas al análisis de secuencias temporales de tráfico, se ha definido una arquitectura compuesta por una o dos capas recurrentes con un número de unidades comprendido entre 64 y 128 neuronas. El entrenamiento se ha realizado utilizando funciones de activación ReLU y una capa de salida sigmoide para clasificación binaria. Como optimizador se ha empleado Adam, con tasas de aprendizaje ajustadas experimentalmente, y se han utilizado tamaños de lote (*batch size*) moderados para garantizar la estabilidad del entrenamiento. Además, se han aplicado técnicas de regularización, como *dropout*, para reducir el riesgo de sobreajuste.

La selección del modelo final se ha realizado mediante pruebas comparativas sobre el conjunto de validación, evaluando métricas como precisión, *recall*, *F1-score* y latencia media de inferencia. Aunque los modelos de aprendizaje profundo ofrecen una mayor capacidad para capturar dependencias temporales complejas, los modelos clásicos presentan menores tiempos de inferencia. Por este motivo, la arquitectura propuesta permite la combinación de ambos enfoques, utilizando modelos más ligeros para detección inmediata y reservando modelos más complejos para análisis en profundidad o reentrenamiento.

Las decisiones de implementación descritas en este capítulo influyen de manera directa en los resultados experimentales presentados en el Capítulo 6. Aspectos como la estrategia de preprocesamiento y normalización de los datos, el balanceo de clases mediante técnicas de *resampling*, la selección de características y la configuración de los hiperparámetros de los modelos tienen un impacto significativo en las métricas de rendimiento obtenidas, especialmente en términos de precisión, *recall*, tasa de falsos positivos y latencia de detección.

En el capítulo siguiente se analizan de forma cuantitativa y cualitativa los resultados obtenidos, evaluando cómo estas decisiones técnicas afectan al comportamiento del sistema en escenarios reales de detección de amenazas y justificando la idoneidad del enfoque adoptado.

Capítulo 6. Implementación y experimentación

6.1. Entorno experimental

El entorno experimental empleado para el desarrollo y validación de este sistema de detección de amenazas en tiempo real se implementó sobre una máquina virtual, con el propósito de disponer de un entorno de trabajo aislado, estable y fácilmente reproducible. Esta decisión permitió controlar de forma precisa tanto la configuración del *software* como la asignación de recursos, reduciendo posibles interferencias con el sistema anfitrión y garantizando la consistencia de los resultados obtenidos durante las distintas fases del proyecto.

La máquina virtual se ejecutó mediante el hipervisor Oracle VirtualBox [34] y se utiliza sobre ella como sistema operativo Ubuntu Server 22.04.3 LTS (Jammy Jellyfish) [35]. Se optó por una versión de soporte extendido (LTS) debido a su probada estabilidad, su amplio respaldo por parte de la comunidad y la disponibilidad de actualizaciones de seguridad a largo plazo, aspectos especialmente relevantes en un entorno orientado a la experimentación en ciberseguridad.

La instalación del sistema se realizó sin entorno gráfico, priorizando el uso exclusivo de la línea de comandos. Esta configuración permite reducir el consumo de recursos y aproximar el entorno de pruebas a escenarios reales de producción, donde este tipo de sistemas suele desplegarse en servidores sin interfaz gráfica para mejorar el rendimiento y la seguridad.

En cuanto a los recursos hardware asignados, la máquina virtual se configuró con cuatro núcleos de CPU y aproximadamente 6,5 GB de memoria RAM, suficientes para soportar la ejecución simultánea de tareas de preprocesamiento de datos, entrenamiento de modelos de aprendizaje automático y despliegue de servicios contenerizados. El almacenamiento se estructuró en dos volúmenes independientes: un primer disco de 30 GB destinado exclusivamente al sistema operativo y un segundo volumen de 70 GB dedicado al almacenamiento de *datasets*, modelos entrenados, resultados experimentales y otros artefactos generados durante el desarrollo.

El volumen adicional se formateó con el sistema de archivos ext4 y se montó en el directorio */data*. Para garantizar su disponibilidad tras cada reinicio, se configuró su montaje automático mediante el fichero */etc/fstab*. Esta separación entre el sistema y los datos facilitó una gestión más ordenada del entorno experimental, además de simplificar tareas de mantenimiento y respaldo.

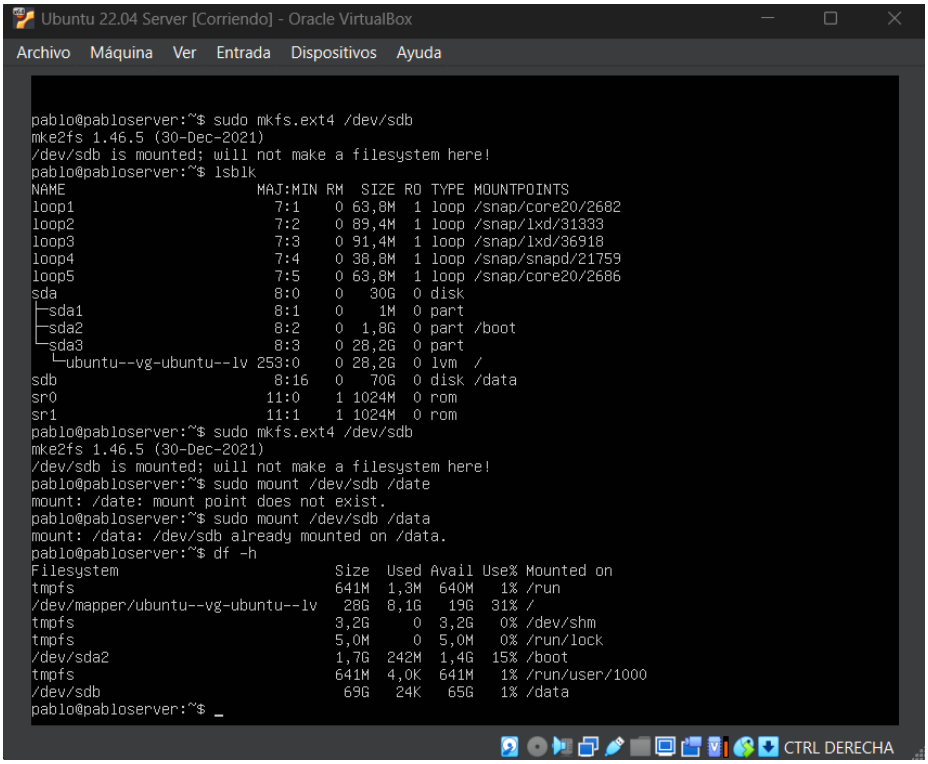

Figura 6. Configuración del montaje del volumen de datos en el sistema Ubuntu 22.04 Server. Fuente: Elaboración propia.

Esta configuración, mostrada en la Figura 6, proporciona un entorno suficientemente robusto y flexible para la ejecución de los experimentos planteados, permitiendo trabajar de forma controlada y reproducible. Este entorno sirvió como base para la implementación y evaluación del sistema propuesto, cuyos detalles se desarrollan en las secciones posteriores.

6.2. Configuración experimental

6.2.1. Objetivo de la configuración experimental

La configuración experimental se concibió con el objetivo de establecer un entorno de ejecución controlado y reproducible que permitiera abordar, de forma integral, el desarrollo, entrenamiento, despliegue y evaluación de un sistema de detección de amenazas en tiempo real basado en técnicas de aprendizaje automático.

Antes de proceder a la instalación y puesta en marcha de las distintas herramientas, fue necesario definir un marco experimental coherente que garantizase la consistencia de los resultados obtenidos a lo largo del trabajo. Para ello, se seleccionó un conjunto de tecnologías *software* orientadas a la contenerización de servicios, el procesamiento y análisis de datos, el entrenamiento de modelos y la monitorización del sistema en ejecución.

Esta aproximación permitió asegurar la alineación entre el entorno de desarrollo y el entorno de experimentación, reduciendo posibles discrepancias y facilitando la repetibilidad de los experimentos. De este modo, la configuración experimental no solo actúa como soporte técnico del sistema propuesto, sino que constituye un elemento clave para la validez y fiabilidad de los resultados presentados en los apartados posteriores [6], [11], [19].

6.2.2. Configuración del entorno base y gestión de dependencias

Durante la fase inicial de configuración del entorno experimental se realizó la actualización del sistema operativo Ubuntu Server 22.04 LTS [35], junto con la instalación de las dependencias necesarias para la posterior contenedorización del sistema. Este paso resultó fundamental para garantizar la compatibilidad de las herramientas empleadas y disponer de un entorno actualizado desde el punto de vista funcional y de seguridad.

Durante el proceso de actualización se produjo una interrupción inesperada que provocó que el gestor de paquetes *dpkg* quedase en un estado inconsistente, impidiendo temporalmente la instalación correcta de nuevos paquetes. La situación fue solventada mediante la ejecución manual de los comandos estándar de recuperación del sistema de paquetes (*dpkg --configure -a y apt --fix-broken install*), lo que permitió restaurar la coherencia del gestor de paquetes y continuar con el proceso de instalación sin pérdida de información ni necesidad de reinstalar el sistema operativo.

Una vez restablecido el estado correcto del sistema, se procedió a la instalación de las dependencias requeridas y del motor de contenedores Docker, que constituye la base del despliegue de los distintos componentes del sistema propuesto. La correcta instalación del entorno de contenedorización se verificó mediante la ejecución de contenedores de prueba, confirmando su funcionamiento adecuado [6], [11].

Esta fase permitió disponer de un entorno estable, controlado y reproducible, preparado para el despliegue de servicios multi-contenedor y la ejecución de los experimentos descritos en las secciones posteriores del documento.

6.2.3. Contenedorización del sistema

Con el objetivo de garantizar la reproducibilidad del entorno de ejecución y aislar las dependencias del sistema, se adoptó una estrategia de contenedorización basada en Docker. Esta aproximación permite empaquetar el servicio de detección junto con su runtime y librerías necesarias, asegurando un comportamiento consistente independientemente del sistema anfitrión [6], [11].

El servicio principal se encapsuló mediante un fichero *Dockerfile*, que define una imagen base de Python y automatiza la instalación de dependencias a partir del fichero *requirements.txt*. Asimismo, se incluyó una estructura inicial del proyecto (directorios *src/* y *models/*) para validar el proceso de construcción de la imagen y preparar el despliegue incremental del sistema.

La orquestación del servicio se realizó mediante *Docker Compose*, utilizando un fichero *docker-compose.yml* donde se especifican la construcción de la imagen, el mapeo de puertos y el uso de volúmenes para persistir los modelos entrenados. Finalmente, se verificó la correcta construcción de la imagen mediante el comando *docker compose build*, confirmando la operatividad del entorno de contenedorización y dejando preparado el sistema para incorporar el código funcional y los experimentos en fases posteriores. En la siguiente imagen, la Figura 7, se puede ver la construcción satisfactoria de la imagen Docker [6].

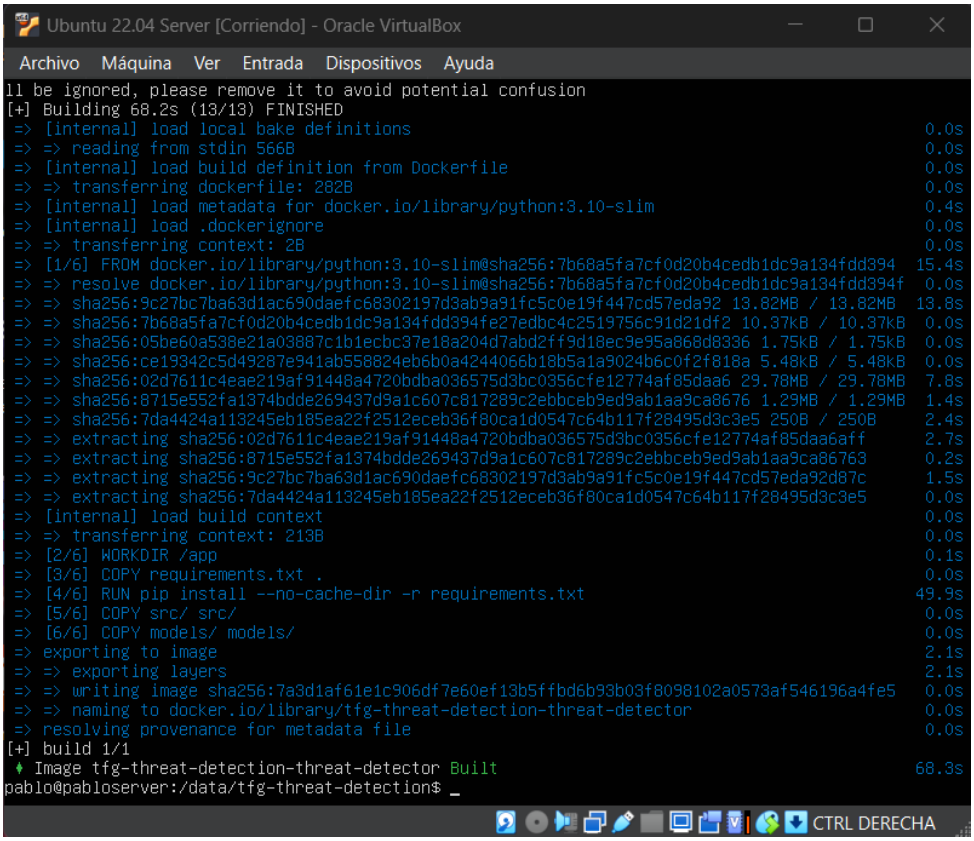

Figura 7. Construcción satisfactoria de la imagen Docker del servicio de detección mediante docker compose build. Fuente: Elaboración propia.

6.2.4. Preparación y selección de los *datasets*

Tal y como se ha descrito en el apartado 5.2, para la fase experimental del sistema se emplearon los conjuntos de datos CICIDS2017 y UNSW-NB15. En este apartado se detalla el modo en que dichos *datasets* fueron integrados y preparados dentro del entorno experimental.

El uso conjunto de ambos conjuntos de datos permite evaluar el comportamiento del sistema bajo condiciones de tráfico con características diferenciadas, reforzando el análisis de la capacidad de generalización de los modelos entrenados. Mientras CICIDS2017 aporta un escenario de tráfico moderno y realista, UNSW-NB15 facilita una evaluación complementaria gracias a su estructura tabular y etiquetado cuidadosamente definido.

Desde el punto de vista experimental, los *datasets* fueron tratados de forma independiente durante las fases de preparación y preprocesamiento, manteniendo una estructura organizada que distingue entre datos originales y datos procesados. Esta aproximación permitió garantizar la trazabilidad de los experimentos y facilitar la comparación posterior de resultados entre ambos conjuntos de datos.

6.2.4.1 Organización y almacenamiento de los *datasets*

Con el objetivo de facilitar la gestión de los datos y garantizar la reproducibilidad de los experimentos, los conjuntos de datos fueron organizados dentro del volumen de datos */data* siguiendo una estructura jerárquica clara. Se establece una separación explícita entre los datos originales y los datos procesados, almacenando los ficheros sin modificar en un directorio específico y reservando directorios independientes para los resultados del preprocesamiento. Esta organización permite preservar la integridad de los datos originales y facilita la repetición de los experimentos bajo distintas configuraciones. La estructura adoptada distingue además entre los *datasets* CICIDS2017 y UNSW-NB15, permitiendo así un tratamiento independiente de cada conjunto de datos durante las fases posteriores del proceso experimental.

6.2.4.2 Selección de archivos relevantes y preparación inicial

Para cada uno de los conjuntos de datos se llevó a cabo una selección de los ficheros más representativos, con el objetivo de reducir la complejidad del procesamiento sin comprometer la diversidad de escenarios de tráfico y ataques incluidos en el experimento.

En el caso del *dataset* CICIDS2017 [36], se seleccionaron varios ficheros correspondientes a distintos días y franjas horarias, abarcando tráfico normal y múltiples tipologías de ataque, tales como denegación de servicio, infiltración y actividades de reconocimiento. Esta selección permite disponer de un conjunto de datos equilibrado y representativo, manteniendo al mismo tiempo un volumen de información manejable para la fase experimental [31].

Para el *dataset* UNSW-NB15 [37], [38], [39], [40], [41] se emplearon los ficheros oficiales de entrenamiento y prueba proporcionados por los autores del conjunto de datos. Ambos ficheros fueron conservados de forma independiente, respetando la partición original del *dataset* y facilitando así la evaluación posterior de los modelos entrenados [33].

Los archivos seleccionados fueron almacenados en directorios temporales específicos, como se puede ver en la Figura 8, manteniendo los conjuntos de datos separados y preparados para su posterior preprocesamiento. Esta estrategia permite aplicar técnicas de limpieza, transformación y normalización de manera controlada en las siguientes fases del desarrollo experimental.

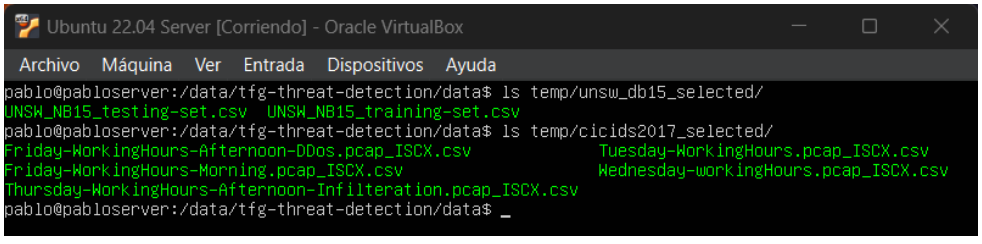

Figura 8. *Datasets* descargados y almacenados en sus respectivos directorios. Fuente: Elaboración propia.

6.2.5. Preprocesamiento de los datos

6.2.5.1 Exploración inicial y análisis de características

Durante la fase de exploración inicial se observaron diferencias significativas entre ambos conjuntos de datos, como se puede ver en las figuras 9, 10 y 11. Mientras que UNSW-NB15 presenta una estructura homogénea con una columna de etiquetado claramente definida, en el caso de CICIDS2017 la información relativa a la clase no se encuentra normalizada bajo un nombre único, lo que requiere un tratamiento adicional previo a la fase de entrenamiento. Asimismo, se detectaron valores nulos en determinadas variables del tráfico, justificando la necesidad de una etapa específica de limpieza y preprocesamiento de los datos.

Como se observa en la Figura 9 y la Figura 10, correspondientes a la exploración inicial del *dataset* CICIDS2017, el conjunto de datos presenta un elevado número de características (79 columnas) y un volumen considerable de registros, lo que evidencia su riqueza descriptiva para el análisis de tráfico de red. No obstante, durante esta fase se detectó la ausencia de una columna de etiquetado claramente identificada bajo el nombre estándar *Label*, lo que pone de manifiesto la heterogeneidad estructural del *dataset* y la necesidad de realizar una normalización previa antes de su uso en modelos supervisados.

En la Figura 10 y la Figura 11, correspondientes al *dataset* UNSW-NB15, se aprecia una estructura más homogénea y compacta, con un total de 45 atributos y una columna de etiquetado binaria claramente definida. La distribución de las clases muestra un cierto desbalance entre tráfico normal y malicioso, aspecto que será tenido en cuenta en la fase de entrenamiento. Asimismo, no se detectaron valores nulos relevantes, lo que confirma la calidad inicial del conjunto de datos y facilita su integración directa en el *pipeline* de aprendizaje automático. Para la exploración inicial de los conjuntos de datos, se realizó mediante un *script* desarrollado en Python. El *script* se puede ver en el Anexo 1, subapartado A.1.

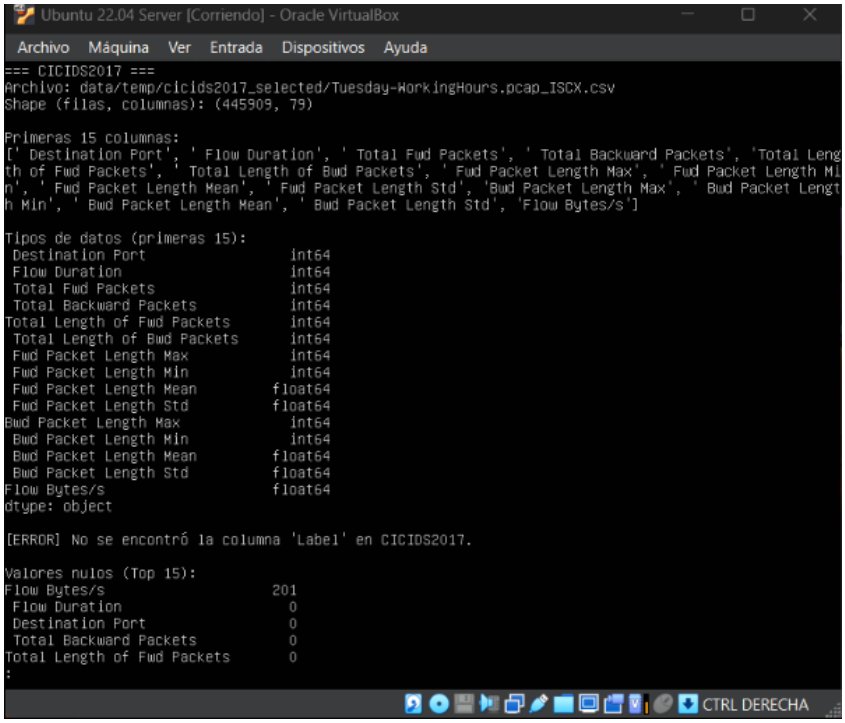

Figura 9. Exploración inicial del *dataset* CICIDS2017. Fuente: Elaboración propia.

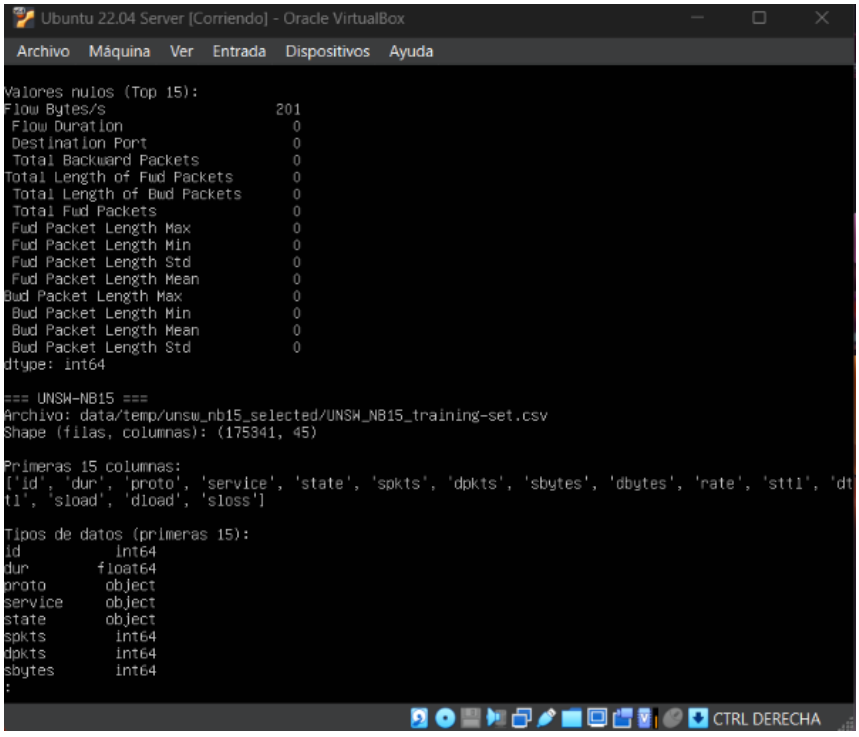

Figura 10. Exploración inicial del *dataset* CICIDS2017 y del *dataset* UNSW-NB15. Fuente: Elaboración propia.

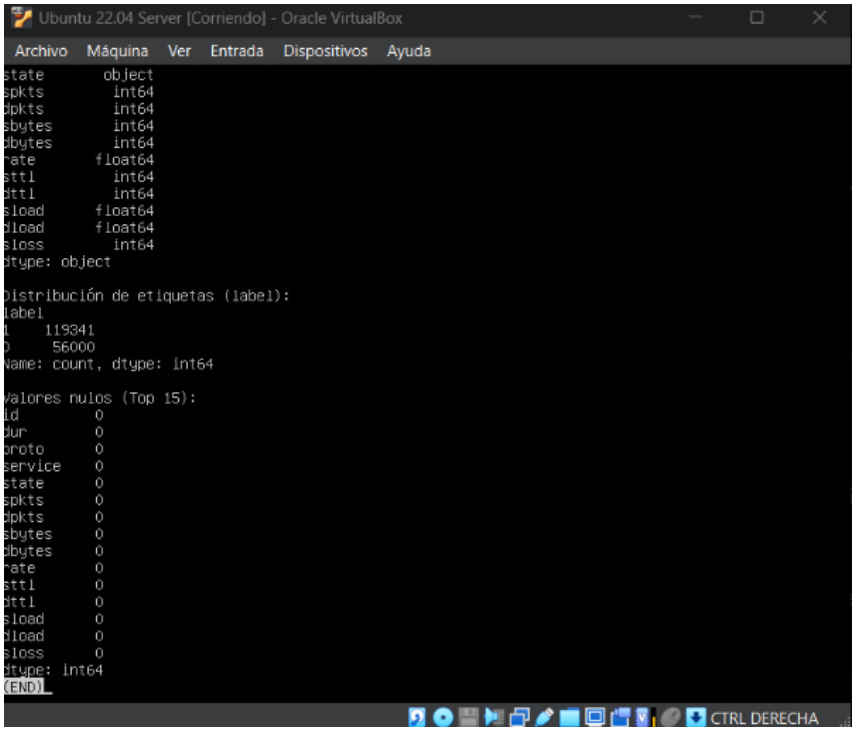

Figura 11. Exploración inicial del *dataset* UNSW-NB15

El análisis comparativo de ambos conjuntos de datos, reflejado en las tres figuras anteriores, pone de manifiesto diferencias estructurales significativas. Mientras CICIDS2017 destaca por su riqueza de atributos y realismo en la simulación del tráfico, también introduce una mayor complejidad en las tareas de limpieza y etiquetado. Por otro lado, UNSW-NB15 presenta una estructura más controlada y uniforme, lo que permite una validación más directa de los modelos supervisados. La combinación de ambos *datasets* refuerza la evaluación de la capacidad de generalización del sistema propuesto.

6.2.5.2 Limpieza y preprocesamiento de los datos

Una vez identificadas las inconsistencias estructurales y la presencia de valores nulos durante la fase de exploración inicial, se procedió a la implementación de un proceso de limpieza y preprocesamiento automatizado para ambos conjuntos de datos. Dicho proceso se lleva a cabo mediante *scripts* específicos desarrollados en Python, ejecutados dentro de un entorno virtual controlado, garantizando la reproducibilidad de los resultados.

En el caso del *dataset* CICIDS2017, se normalizaron los nombres de las columnas, se identificó y transformó la variable de etiquetado en una representación binaria (tráfico benigno frente a tráfico malicioso) y se eliminaron los registros que contenían valores nulos o infinitos. Como resultado de este proceso, se obtuvo un conjunto de datos limpio compuesto por 445.645 registros y 79 características, tal y como se observa en la Figura 12.

Para el *dataset* UNSW-NB15, cuya estructura inicial es más homogénea, el proceso de limpieza se centró en la eliminación de atributos categóricos, el tratamiento de posibles valores inconsistentes y la preservación de la variable de etiquetado original. El *dataset* resultante consta de 175.341 registros y 41 características, como se muestra en la Figura 12.

Ambos conjuntos de datos fueron almacenados en formato CSV dentro del directorio de datos procesados, quedando listos para su uso en las fases posteriores de selección de características, normalización y entrenamiento de modelos de aprendizaje automático.

Este proceso de limpieza y preprocesamiento fue automatizado a través de dos *scripts* específicos para cada *dataset*, como son lo que se muestran en el Anexo 1, subapartados A.2 y A.3.

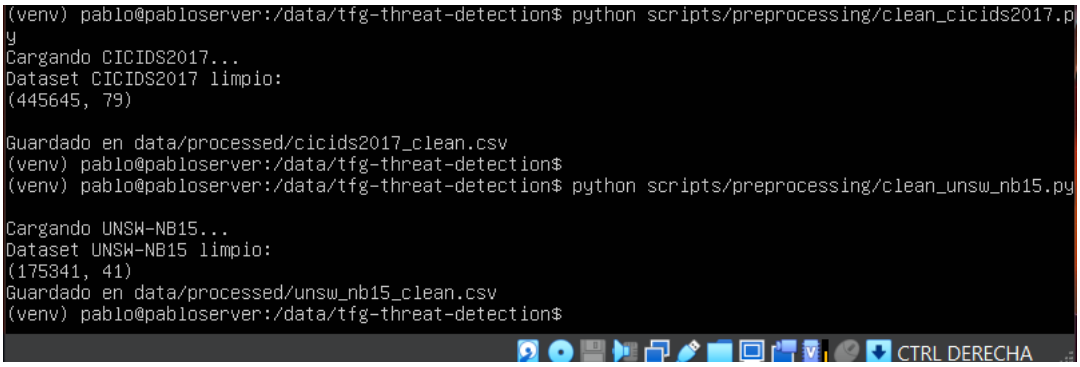

Figura 12. Limpieza del CICIDS2017 y del UNSW-NB15. Fuente: Elaboración propia.

6.2.5.3 Selección de características y normalización

Tras la anterior fase de limpieza, en este apartado se procede a la selección de características y a la normalización de los datos con el objetivo de preparar los conjuntos de datos para el entrenamiento de modelos de aprendizaje automático. En esta etapa se separaron explícitamente las variables independientes (features) de la variable objetivo (label), permitiendo un tratamiento diferenciado de ambas.

Como se observa en la Figura 13, en el caso del *dataset* CICIDS2017, el conjunto final quedó compuesto por 445.645 registros y 79 características, manteniendo la totalidad de los atributos numéricos relevantes tras el proceso de limpieza. Para el *dataset* UNSW-NB15, el conjunto resultante consta de 175.341 registros y 41 características, una vez eliminados los atributos categóricos no directamente utilizables por los modelos supervisados empleados.

Con el fin de evitar sesgos derivados de diferencias de escala entre las variables, se aplicó una técnica de estandarización basada en Standard Scaling, transformando cada característica para que presentase media cero y desviación estándar unitaria. Este procedimiento garantiza que todas las variables contribuyan de forma equilibrada al proceso de aprendizaje, mejorando la estabilidad y la convergencia de los modelos entrenados.

Los conjuntos de datos resultantes fueron almacenados en formato CSV dentro del directorio de datos procesados, quedando preparados para su uso directo en la fase de experimentación y evaluación de modelos.

Esta sección de características y la normalización de los datos se implementaros mediant un script dedicado para este fin. Este script se puede encontrar en el Anexo 1, subapartado A.4.

```
(venv) pablo@pabloserver:/data/tfg-threat-detection$ python scripts/preprocessing/feature_selection.
py
Procesando CICIDS2017...
CICIDS2017 listo para entrenamiento: (445645, 79)
Procesando UNSW-NB15...
UNSW-NB15 listo para entrenamiento: (175341, 41)
```

Figura 13. Resultado de los procesos de selección de características y normalización de ambos *datasets*.

6.3. Métricas de evaluación

La evaluación del rendimiento de un sistema de detección de intrusiones basado en técnicas de aprendizaje automático exige el empleo de métricas que permitan analizar de forma rigurosa el comportamiento del modelo. No resulta suficiente considerar únicamente la tasa global de aciertos, sino que es necesario examinar cómo el sistema gestiona los distintos tipos de errores, especialmente en contextos donde la distribución de las clases no es equilibrada.

Los conjuntos de datos utilizados en este trabajo de fin de grado presentan una proporción desigual entre tráfico legítimo y tráfico malicioso, una situación habitual en entornos reales de ciberseguridad. Por este motivo, se han seleccionado métricas que permiten evaluar con mayor precisión el impacto de los falsos positivos y los falsos negativos, aspectos clave para determinar la fiabilidad operativa del sistema.

La exactitud, *Accuracy*, se emplea como una medida general del porcentaje de predicciones correctas sobre el total de instancias analizadas. Debido a su limitada representatividad en escenarios desbalanceados, su interpretación se realiza con cautela y siempre en combinación con otras métricas [7], [19].

La precisión, *Precision*, refleja la proporción de eventos identificados como maliciosos que realmente corresponden a ataques. Esta métrica resulta especialmente relevante en sistemas de monitorización, ya que un elevado número de falsos positivos puede generar alertas innecesarias y aumentar la carga de trabajo de los equipos de seguridad.

Por otra parte, la sensibilidad o tasa de detección, *Recall*, mide la capacidad del modelo para identificar correctamente los ataques reales presentes en el tráfico analizado. Un valor alto de esta métrica es fundamental en sistemas de detección de intrusiones, dado que los falsos negativos suponen la omisión de amenazas potencialmente críticas.

El *F1-score* combina *precision* y *recall* mediante su media armónica, proporcionando un indicador equilibrado del rendimiento del modelo cuando existe un compromiso entre ambos tipos de error. Esta métrica resulta especialmente útil para comparar modelos en escenarios donde ninguna de las dos medidas, por separado, ofrece una visión completa [7], [31].

Finalmente, el área bajo la curva ROC, *ROC-AUC,* se utiliza para evaluar la capacidad discriminativa del modelo frente a distintos umbrales de decisión. Al tratarse de una métrica independiente del umbral de clasificación, permite comparar de forma objetiva el rendimiento de distintos modelos [19].

Como complemento al análisis cuantitativo, se emplea la matriz de confusión, que facilita una visión detallada de la distribución de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

El uso conjunto de estas métricas proporciona una evaluación sólida y completa del sistema propuesto, permitiendo analizar su comportamiento en condiciones realistas y comparar de manera fundamentada el rendimiento de los modelos entrenados para su posible aplicación en entornos de detección de amenazas en tiempo real.

6.4. Ejecución de los experimentos

6.4.1. Diseño experimental

La evaluación de los modelos propuestos se realizó siguiendo un diseño experimental controlado, orientado a analizar de manera objetiva su rendimiento en tareas de detección de intrusiones. Para cada conjunto de datos empleado, se llevó a cabo una partición del total de instancias en subconjuntos de entrenamiento y prueba, garantizando la separación entre ambas fases y evitando posibles sesgos durante la evaluación.

Los experimentos se desarrollaron de forma independiente sobre los *datasets* CICIDS2017 y UNSW-NB15, lo que permitió estudiar el comportamiento de los modelos frente a diferentes patrones de tráfico y distribuciones de datos. Esta aproximación facilitó la comparación del rendimiento de los algoritmos en escenarios con características heterogéneas [7], [19].

Todos los modelos fueron evaluados empleando el mismo conjunto de métricas descritas en el apartado 6.3, lo que aseguró la consistencia metodológica y la comparabilidad directa de los resultados obtenidos entre los distintos experimentos.

6.4.2. Entrenamiento y validación de modelos

Para la fase experimental se entrenó un modelo basado en el algoritmo Random Forest utilizando los conjuntos de datos CICIDS2017 y UNSW-NB15. En ambos casos, los datos se dividieron en un 70 % destinado al entrenamiento y un 30 % reservado para la fase de prueba. La partición se realizó de forma estratificada, con el fin de conservar la distribución original de las clases y garantizar una evaluación representativa del modelo.

En el caso del *dataset* CICIDS2017, el modelo alcanzó un rendimiento sobresaliente en todas las métricas consideradas. Se obtuvo una exactitud del 100 % y un valor de área bajo la curva ROC igual a 1.00, lo que evidencia una capacidad prácticamente perfecta para discriminar entre tráfico legítimo y tráfico malicioso. La matriz de confusión asociada, representada en la Figura 14, muestra un número residual de clasificaciones incorrectas.

Para el *dataset* UNSW-NB15, el modelo presentó igualmente un rendimiento elevado, con una exactitud próxima al 99 % y un valor de ROC-AUC de 0.9992. A diferencia de los resultados obtenidos con CICIDS2017, en este caso se identifican algunos falsos positivos y falsos negativos, lo que pone de manifiesto una mayor complejidad intrínseca del conjunto de datos. Este comportamiento puede observarse en la matriz de confusión mostrada en la Figura 14.

Aunque los resultados obtenidos durante esta fase experimental se presentan de manera detallada y se analizan en profundidad en el Capítulo 7, este rendimiento excepcional debe interpretarse con cautela, ya que diversos estudios han señado que determinados subconjuntos

del *dataset* CICIDS2017 pueden facilitar resultados cercanos a la clasificación perfecta cuando se emplean algoritmos como el Random Forest [31].

El entrenamiento del modelo Random Forest se llevó mediante un *script* desarrollado en Python, el cual se puede ver en el Anexo 1, subapartado A.5.

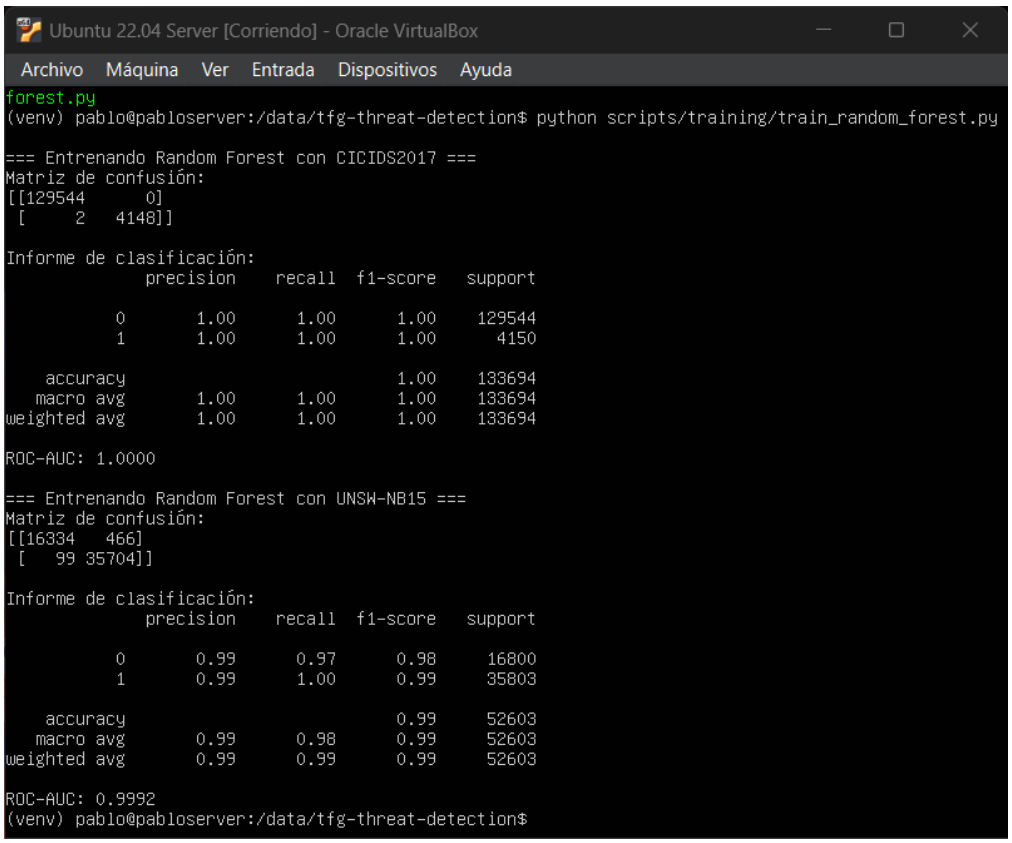

Figura 14. Resultados del modelo Random Forest entrenado sobre ambos *datasets*. Fuente: Elaboración propia.

6.4.3. Procedimiento de evaluación

La evaluación de los modelos entrenados se realizó de manera sistemática empleando las métricas definidas en el apartado 6.3. Para cada conjunto de datos, las predicciones generadas a partir del conjunto de prueba se compararon con las etiquetas reales, lo que permitió calcular los valores de exactitud, precisión, *recall*, *F1-score* y el área bajo la curva ROC.

Como complemento al análisis cuantitativo, se utilizaron matrices de confusión con el fin de examinar de forma detallada el comportamiento de los modelos ante los distintos tipos de error. Este análisis permitió identificar la presencia de falsos positivos y falsos negativos, aportando una visión más precisa del desempeño del sistema en contextos de detección de intrusiones.

La aplicación uniforme de este procedimiento de evaluación garantiza la coherencia metodológica y la comparabilidad de los resultados obtenidos entre los diferentes conjuntos de datos y modelos considerados. Los resultados derivados de este proceso se presentan y analizan de manera detallada en el Capítulo 7.

Capítulo 7. Resultados y análisis

7.1. Resultados cuantitativos

En este apartado se presentan los resultados cuantitativos obtenidos durante la fase experimental del sistema de detección de amenazas. Los resultados corresponden a la evaluación del modelo Random Forest entrenado sobre los *datasets* CICIDS2017 y UNSW-NB15, empleando las métricas definidas en el apartado 6.3. El objetivo de este análisis es ofrecer una visión objetiva del rendimiento del sistema en términos de capacidad de detección y clasificación del tráfico de red.

7.1.1. Resultados sobre CICIDS2017

El modelo entrenado sobre este *dataset* se obtuvo un rendimiento sobresaliente en todas las métricas evaluadas. Tal y como se muestra en la Tabla 1, la exactitud alcanzada fue del 100% con valores igualmente máximos en precisión, *recall* y *F1-score*. La matriz de confusión correspondiente, representada en la Tabla 1, evidencia una separación prácticamente perfecta entre tráfico benigno y malicioso, registrándose únicamente un número residual de clasificaciones erróneas.

Métrica	Valor
Accuracy	1.00
Precision	1.00
Recall	1.00
F1-score	1.00
ROC-AUC	1.00

Tabla 1. Resultados de evaluación del modelo Random Forest sobre el *dataset* CICIDS2017.

7.1.2. Resultados sobre UNSW-NB15

En el caso del dataset UNSW-NB15, el modelo Random Forest obtuvo también un rendimiento elevado, aunque ligeramente inferior al observado en CICIDS2017. Como se recoge en la Tabla 2, la exactitud alcanzada fue cercana al 99 %, con valores de precisión y recall equilibrados. La matriz de confusión asociada, mostrada en la Figura 12, refleja la presencia de falsos positivos y falsos negativos, lo que pone de manifiesto una mayor complejidad intrínseca del conjunto de datos.

Métrica	Valor
Accuracy	0.99
Precision	0.99
Recall	0.97-1.00
F1-score	0.99
ROC-AUC	0.9992

Tabla 2. Resultados de evaluación del modelo Random Forest sobre el *dataset* UNSW-NB15.

7.2. Análisis cualitativo de los resultaos

El análisis de los resultados obtenidos pone de relieve diferencias relevantes en el comportamiento del modelo Random Forest en función del conjunto de datos empleado. Estas variaciones están estrechamente vinculadas a las características propias de cada *dataset*, en particular a su estructura interna, nivel de complejidad y grado de realismo del tráfico representado.

En el caso del conjunto de datos CICIDS2017, el modelo alcanzó valores prácticamente óptimos en todas las métricas evaluadas. Este resultado puede explicarse por la riqueza y calidad de las características estadísticas incluidas, así como por la elevada separabilidad entre tráfico legítimo y malicioso tras la aplicación del preprocesamiento. La combinación de atributos derivados de los flujos de red y el carácter controlado del entorno en el que se generaron los datos facilita la identificación de patrones discriminativos claros, favoreciendo un rendimiento elevado del modelo.

Por el contrario, el *dataset* UNSW-NB15 presenta un escenario más heterogéneo y cercano a condiciones reales de operación, lo que se traduce en la aparición de falsos positivos y falsos negativos durante el proceso de clasificación. No obstante, el modelo mantiene un rendimiento notablemente alto, lo que evidencia una buena capacidad de generalización y una adaptación adecuada a conjuntos de datos con mayor variabilidad y presencia de ruido.

La comparación entre ambos conjuntos de datos pone de manifiesto que el rendimiento de un sistema de detección de intrusiones no depende exclusivamente del algoritmo empleado, sino también de la calidad, estructura y representatividad de los datos utilizados durante las fases de entrenamiento y evaluación. En este contexto, el uso combinado de CICIDS2017 y UNSW-NB15 permite obtener una visión más completa y realista del comportamiento del sistema propuesto, reforzando la validez y solidez de los resultados alcanzados.

7.3. Discusión y limitaciones

Los resultados obtenidos durante la fase experimental evidencian un rendimiento muy elevado del modelo Random Forest en ambos conjuntos de datos analizados, especialmente en el caso de CICIDS2017, donde se alcanzaron valores prácticamente óptimos en todas las métricas evaluadas. Si bien estos resultados ponen de manifiesto la eficacia del enfoque propuesto, resulta necesario abordarlos desde una perspectiva crítica, teniendo en cuenta las limitaciones asociadas tanto al diseño experimental como a las características de los *datasets* empleados.

En el caso de CICIDS2017, el rendimiento excepcional del modelo puede verse condicionado por la naturaleza controlada del entorno en el que se generaron los datos y por la elevada separabilidad entre tráfico legítimo y malicioso tras el proceso de preprocesamiento. Este fenómeno ha sido ampliamente señalado en la literatura, donde se indica que dicho conjunto de datos tiende a favorecer resultados optimistas cuando se emplean algoritmos supervisados con acceso a un conjunto amplio de características estadísticas. En este sentido, no puede descartarse la existencia de cierto grado de sobreajuste, especialmente si el modelo fuese evaluado sobre tráfico con características significativamente diferentes a las presentes en el conjunto de entrenamiento.

El comportamiento especialmente favorable del modelo Random Forest sobre el conjunto de datos CICIDS2017 puede explicarse, además, por las propias características del algoritmo. Random Forest es un método de aprendizaje basado en la combinación de múltiples árboles de decisión entrenados sobre subconjuntos aleatorios de datos y características, lo que le permite capturar relaciones no lineales complejas y reducir la varianza del modelo. Este enfoque resulta particularmente eficaz en escenarios con alta dimensionalidad y atributos estadísticos bien definidos, como es el caso de CICIDS2017, donde muchas de las características del tráfico de red presentan patrones claramente diferenciables entre clases.

En comparación con otros enfoques de aprendizaje automático, como los clasificadores lineales o los métodos basados en distancia, Random Forest presenta una mayor robustez frente a ruido y correlaciones entre variables, aspectos frecuentes en datos de tráfico de red. Asimismo, frente a modelos de aprendizaje profundo, Random Forest ofrece un equilibrio adecuado entre rendimiento y coste computacional, lo que facilita su aplicación en entornos donde los recursos son limitados o donde se requiere una fase de entrenamiento relativamente rápida.

Desde el punto de vista de su aplicación en entornos reales, los resultados obtenidos sugieren que Random Forest puede constituir una solución eficaz para sistemas de detección de intrusiones en escenarios controlados o semi-controlados, así como en tareas de análisis offline o monitorización periódica del tráfico. No obstante, su despliegue en sistemas de detección en tiempo real requiere considerar aspectos adicionales, como la latencia de inferencia, la actualización del modelo ante cambios en los patrones de tráfico y la capacidad para detectar ataques previamente desconocidos. En este contexto, el uso de Random Forest podría complementarse con técnicas adicionales de detección de anomalías o aprendizaje continuo.

Por el contrario, el *dataset* UNSW-NB15 presenta un escenario más exigente, caracterizado por una mayor heterogeneidad y la presencia de ruido en los datos. Este aspecto se refleja en un rendimiento ligeramente inferior, aunque todavía elevado, lo que sugiere una evaluación más cercana a condiciones reales de operación. Los resultados obtenidos con este conjunto de datos aportan, por tanto, una visión más realista de la capacidad de generalización del sistema propuesto en entornos de detección de intrusiones.

Entre las principales limitaciones del presente trabajo destaca el uso de conjuntos de datos estáticos, lo que impide evaluar el comportamiento del sistema frente a tráfico de red en tiempo real o ante la aparición de ataques previamente desconocidos. Asimismo, el estudio se ha centrado en un único algoritmo de clasificación, lo que limita la comparación directa con otros enfoques basados en aprendizaje automático o aprendizaje profundo.

A pesar de estas limitaciones, la utilización combinada de conjuntos de datos con características diferenciadas, junto con la aplicación de un proceso de preprocesamiento riguroso y una evaluación sistemática, permite extraer conclusiones fundamentadas sobre la viabilidad del enfoque propuesto como sistema de detección de amenazas basado en inteligencia artificial.

Capítulo 8. Conclusiones y líneas futuras

8.1. Conclusiones

Este Trabajo Fin de Grado ha abordado el diseño, la implementación y la evaluación de un sistema de detección de amenazas en redes basado en técnicas de aprendizaje automático. A lo largo del desarrollo del proyecto se ha definido y aplicado un *pipeline* completo que abarca desde la preparación y el preprocesamiento de los datos hasta la ejecución experimental y el análisis de los resultados obtenidos.

Los resultados experimentales ponen de manifiesto que el enfoque propuesto permite identificar de forma eficaz tráfico malicioso en distintos escenarios, alcanzando un rendimiento elevado en los conjuntos de datos analizados. En particular, el modelo basado en Random Forest ha demostrado una notable capacidad para discriminar entre tráfico benigno y malicioso, especialmente cuando se aplica un proceso adecuado de limpieza, normalización y preparación de los datos.

La utilización conjunta de los *datasets* CICIDS2017 y UNSW-NB15 ha permitido evaluar el comportamiento del sistema bajo condiciones de distinta complejidad, proporcionando una visión más completa de su capacidad de generalización. Mientras que CICIDS2017 presenta una separabilidad clara entre las clases, el conjunto UNSW-NB15 ha permitido validar el rendimiento del modelo en un escenario más exigente y próximo a situaciones reales de detección de intrusiones.

Asimismo, los resultados obtenidos evidencian la relevancia de las fases de preprocesamiento y selección de características en los sistemas de detección de intrusiones basados en inteligencia artificial. Un tratamiento inadecuado de los datos puede afectar de manera significativa al rendimiento de los modelos, independientemente del algoritmo de clasificación empleado.

Los objetivos planteados al inicio del proyecto han sido alcanzados de forma satisfactoria, demostrando la viabilidad del uso de técnicas de aprendizaje automático como herramienta de apoyo a la detección de amenazas en redes de comunicación.

8.2. Líneas futuras

Como líneas de trabajo futuro, se plantean diversas mejoras y ampliaciones orientadas a incrementar el alcance, la robustez y la aplicabilidad del sistema propuesto. En primer lugar, resultaría de especial interés evaluar el rendimiento del sistema mediante la incorporación de otros algoritmos de aprendizaje automático y aprendizaje profundo, tales como redes neuronales profundas, modelos basados en arquitecturas LSTM o enfoques híbridos. Esta comparación permitiría analizar el comportamiento de distintas técnicas frente al modelo Random Forest empleado en el presente trabajo.

Otra línea de mejora relevante consiste en la integración del sistema dentro de un entorno de detección en tiempo real, posibilitando el análisis continuo del tráfico de red y la generación

dinámica de alertas ante la identificación de comportamientos anómalos. La materialización de este enfoque requeriría la adopción de técnicas de procesamiento de datos en streaming y la adaptación del pipeline actual a flujos de información en tiempo real.

De este mismo modo, sería conveniente ampliar el estudio a otros conjuntos de datos o a tráfico de red capturado en entornos reales, con el objetivo de evaluar la robustez del sistema frente a ataques desconocidos y escenarios no controlados. En este contexto, la incorporación de técnicas de aprendizaje no supervisado o semisupervisado podría contribuir a mejorar la capacidad de detección de amenazas emergentes.

Otra línea futura del trabajo, podría ser que el trabajo se integrase dentro de un *pipeline CI/CD* que automatice el entrenamiento, la validación y despliegue del modelo ante cambios en los datos o en el código, alineándose al 100% con un enfoque DevSecOps.

Por último, se plantea como línea futura profundizar en el análisis del desbalance de clases y en la aplicación de técnicas avanzadas de balanceo y selección de características, con el fin de optimizar la capacidad de generalización del sistema y mejorar su rendimiento en escenarios de detección complejos.

Referencias bibliográficas

[1] J. Fantl, «Cost of a Data Breach Report», 2023. Accedido: 9 de noviembre de 2025. [En línea]. Disponible en: https://d110erj175o600.cloudfront.net/wp-content/uploads/2023/07/25111651/Cost-of-a-Data-Breach-Report-2023.pdf

[2] S. P. Córdovez Machado, J. J. Cruz Garzón, y C. L. Inca Balseca, «¿Puede el Aprendizaje Automático Predecir Brechas de Ciberseguridad en los Sistemas de Información Empresariales? Un análisis de aprendizaje supervisado», *ASCE*, vol. 4, n.º 3, pp. 333-352, jul. 2025, doi: 10.70577/asce/333.352/2025.

[3] A. Pinto, L. C. Herrera, Y. Donoso, y J. A. Gutierrez, «Survey on Intrusion Detection Systems Based on Machine Learning Techniques for the Protection of Critical Infrastructure», 1 de marzo de 2023, *MDPI*. doi: 10.3390/s23052415.

[4] Ifigeneia. Lella *et al.*, *ENISA threat landscape 2023 : July 2022 to June 2023*. ENISA, 2023. doi: 10.2824/782573.

[5] José Manuel Rodríguez Rama y Enric Hernández Jiménez, «Aplicación de técnicas de Machine Learning a la detección de ataques», jun. 2018. Accedido: 28 de noviembre de 2025. [En línea]. Disponible en: https://openaccess.uoc.edu/server/api/core/bitstreams/2c8ad70f-5d63-4502-a9c1-dc2717f1c74e/content

[6] J. E. Castro Sánchez, «DevSecOps: Implementación de seguridad en DevOps a través de herramientas open source», Barcelona, dic. 2020. Accedido: 9 de noviembre de 2025. [En línea]. Disponible en: https://openaccess.uoc.edu/server/api/core/bitstreams/f82fc345-2686-4600-8e13-cf85b5f447ce/content

[7] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, y E. Vázquez, «Anomaly-based network intrusion detection: Techniques, systems and challenges», *Comput. Secur.*, vol. 28, n.º 1-2, pp. 18-28, feb. 2009, doi: 10.1016/J.COSE.2008.08.003.

[8] D. Fabián, M. Ramos, D. X. Larriva, y N. Ponente, «Diseño de un Sistema de Detección de Intrusiones basado en Machine Learning para tráfico de red real», Madrid, 2021. Accedido: 28 de noviembre de 2025. [En línea]. Disponible en: https://oa.upm.es/69370/1/TFG_FabianMartinRamos.pdf

[9] D. I. Quirumbay Yagual, C. Castillo Yagual, y I. Coronel Suárez, «Una revisión del Aprendizaje profundo aplicado a la ciberseguridad», *Revista Científica y Tecnológica UPSE*, vol. 9, n.º 1, pp. 57-65, jun. 2022, doi: 10.26423/rctu.v9i1.671.

[10] C. Flores Siñani, «Inteligencia Artificial, Machine Learning, Deep Learning aplicados a la Ciberseguridad», nov. 2021. Accedido: 28 de noviembre de 2025. [En línea]. Disponible en: https://ojs.umsa.bo/index.php/inf_fcpn_pgi/article/view/96

[11] F. Lázaro, D. Moreno, y E. Cervantes, «Guía de iniciación en la Seguridad aplicada al DevOps DevSecOps», sep. 2023. Accedido: 9 de noviembre de 2025. [En línea]. Disponible en: https://www.ismsforum.es/noticias/2325/isms-forum-presenta-la-gu-a-devsecops-que-implementa-la-seguridad-desde-el-dise-o-de-cualquier-pol-tica-de-ciberseguridad/

[12] «Ataque DDoS a Activision Blizzard», *Incibe*, may 2022, Accedido: 30 de noviembre de 2025. [En línea]. Disponible en: https://www.incibe.es/incibe-cert/publicaciones/bitacora-de-seguridad/ataque-ddos-activision-blizzard

[13] Jonathan Desrosiers, «WordPress 5.8.3 Security Release», *WordPress*, ene. 2022, Accedido: 30 de noviembre de 2025. [En línea]. Disponible en: https://wordpress.org/news/2022/01/wordpress-5-8-3-security-release/

[14] «Ransomware EKANS afecta a todas las versiones de productos Proficy de GE», *Incibe*, ene. 2022, Accedido: 30 de noviembre de 2025. [En línea]. Disponible en: https://www.incibe.es/incibe-cert/alerta-temprana/avisos-sci/ransomware-ekans-afecta-todas-las-versiones-productos-proficy-ge

[15] «Orangeworm, nueva APT orientada al sector sanitario», *Incibe*, abr. 2018, Accedido: 30 de noviembre de 2025. [En línea]. Disponible en: https://www.incibe.es/incibe-cert/publicaciones/bitacora-de-seguridad/orangeworm-nueva-apt-orientada-al-sector-sanitario

[16] Ainhoa Veramendi Pérez y José Miguel Alonso, «Evaluación de sistemas de detección de amenazas: Snort y Suricata», oct. 2021, Accedido: 30 de noviembre de 2025. [En línea]. Disponible en: http://hdl.handle.net/10810/53353

[17] F. E. Laghrissi, S. Douzi, K. Douzi, y B. Hssina, «Intrusion detection systems using long short-term memory (LSTM)», *J. Big Data*, vol. 8, n.º 1, dic. 2021, doi: 10.1186/s40537-021-00448-4.

[18] S. S. Bamber, A. V. R. Katkuri, S. Sharma, y M. Angurala, «A hybrid CNN-LSTM approach for intelligent cyber intrusion detection system», *Comput. Secur.*, vol. 148, p. 104146, ene. 2025, doi: 10.1016/J.COSE.2024.104146.

[19] G. Apruzzese *et al.*, «The Role of Machine Learning in Cybersecurity», *Digital Threats: Research and Practice*, vol. 4, n.º 1, mar. 2023, doi: 10.1145/3545574.

[20] by Prateek Verma, H.-C. Yang, F. Gebali, y D. Member, «A Machine Learning Approach To Network Security Anomaly Detection», 2025. Accedido: 30 de noviembre de 2025. [En línea]. Disponible en: https://dspace.library.uvic.ca/server/api/core/bitstreams/f50befc8-a9a8-4943-8cce-dc8d64cea8d7/content

[21] O. Almomani, «A Hybrid Model Using Bio-Inspired Metaheuristic Algorithms for Network Intrusion Detection System», *Computers, Materials and Continua*, vol. 68, n.º 1, pp. 409-429, mar. 2021, doi: 10.32604/cmc.2021.016113.

[22] M. Choraś y M. Pawlicki, «Intrusion detection approach based on optimised artificial neural network», *Neurocomputing*, vol. 452, pp. 705-715, sep. 2021, doi: 10.1016/J.NEUCOM.2020.07.138.

[23] A. Gómez Carrera, «Estudio de algoritmos de reconstrucción de vértices usando información del detector "MIPs Timing Detector" del Solenoide Compacto de Muones», jul. 2023, Accedido: 30 de noviembre de 2025. [En línea]. Disponible en: https://hdl.handle.net/10902/30288

[24] David Ameijeiras Sánchez, Héctor González Diez, y Odeynis Valdés Suárez, «Algoritmos de detección de anomalías con redes profundas. Revisión para detección de fraudes bancarios», *Revista Cubana de Ciencias Informáticas*, vol. 15, pp. 244-264, jun. 2021, Accedido: 30 de noviembre de 2025. [En línea]. Disponible en: http://scielo.sld.cu/pdf/rcci/v15n4s1/2227-1899-rcci-15-04-s1-244.pdf

[25] K. Noor, A. L. Imoize, C. T. Li, y C. Y. Weng, «A Review of Machine Learning and Transfer Learning Strategies for Intrusion Detection Systems in 5G and Beyond», 1 de abril de 2025, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/math13071088.

[26] A. Alotaibi y M. A. Rassam, «Adversarial Machine Learning Attacks against Intrusion Detection Systems: A Survey on Strategies and Defense», 1 de febrero de 2023, *MDPI*. doi: 10.3390/fi15020062.

[27] H. Khazane, M. Ridouani, F. Salahdine, y N. Kaabouch, «A Holistic Review of Machine Learning Adversarial Attacks in IoT Networks», 19 de enero de 2024, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/fi16010032.

[28] G. Apruzzese, A. Fass, y F. Pierazzi, «When Adversarial Perturbations meet Concept Drift: an Exploratory Analysis on ML-NIDS», en *AISec 2024 - Proceedings of the 2024 Workshop on Artificial Intelligence and Security, Co-Located with: CCS 2024*, Association for Computing Machinery, Inc, nov. 2024, pp. 149-160. doi: 10.1145/3689932.3694757.

[29] Z. Deng, «Developing machine learning-based intrusion detection systems for IoT environments», 2025. Accedido: 30 de noviembre de 2025. [En línea]. Disponible en: https://ceur-ws.org/Vol-4034/paper68.pdf

[30] O. Castellanos Leyva y M. García Borroto, «Análisis y caracterización de conjuntos de datos para detección de intrusiones Analysis and characterization of data sets for intrusion detection», *Serie Científica de la Universidad de las Ciencias Informáticas*, vol. 13, n.º 4, abr. 2020, Accedido: 9 de noviembre de 2025. [En línea]. Disponible en: https://dialnet.unirioja.es/descarga/articulo/8590270.pdf

[31] Kurniabudi, D. Stiawan, Darmawijoyo, M. Y. Bin Bin Idris, A. M. Bamhdi, y R. Budiarto, «CICIDS-2017 Dataset Feature Analysis with Information Gain for Anomaly Detection», *IEEE Access*, vol. 8, pp. 132911-132921, jul. 2020, doi: 10.1109/ACCESS.2020.3009843.

[32] O. Castellanos Leyva y M. García Borroto, «Análisis y caracterización de conjuntos de datos para detección de intrusiones Analysis and characterization of data sets for intrusion detection», *Serie Científica De La Universidad De Las Ciencias Informáticas*, vol. 13, n.º 4, pp. 39-52, abr. 2020, Accedido: 30 de noviembre de 2025. [En línea]. Disponible en: https://publicaciones.uci.cu/index.php/serie/article/view/558

[33] S. Meftah, T. Rachidi, y N. Assem, «Network based intrusion detection using the UNSW-NB15 dataset», *International Journal of Computing and Digital Systems*, vol. 8, n.º 5, pp. 477-487, sep. 2019, doi: 10.12785/ijcds/080505.

[34] «Oracle VirtualBox». Accedido: 4 de enero de 2026. [En línea]. Disponible en: https://www.virtualbox.org/

[35] «Ubuntu 22.04 LTS (Jammy Jellyfish)». Accedido: 4 de enero de 2026. [En línea]. Disponible en: https://www.stackscale.com/es/blog/ubuntu-22-04-lts/

[36] Iman Sharafaldin, Arash Habibi Lashkari, y Ali A. Ghorbani, «Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization», *4th International Conference on Information Systems Security and Privacy (ICISSP)*, ene. 2018, Accedido: 5 de enero de 2026. [En línea]. Disponible en: https://www.unb.ca/cic/datasets/ids-2017.html

[37] N. Moustafa y J. Slay, «UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)», en *2015 Military Communications and Information Systems Conference (MilCIS)*, Camberra: IEEE, nov. 2015, pp. 1-6. doi: 10.1109/MilCIS.2015.7348942.

[38] N. Moustafa y J. Slay, «The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set», *Information Security Journal: A Global Perspective*, vol. 25, n.º 1-3, pp. 18-31, abr. 2016, doi: 10.1080/19393555.2015.1125974.

[39] N. Moustafa, J. Slay, y G. Creech, «Novel Geometric Area Analysis Technique for Anomaly Detection Using Trapezoidal Area Estimation on Large-Scale Networks», *IEEE Trans. Big Data*, vol. 5, n.º 4, pp. 481-494, dic. 2019, doi: 10.1109/TBDATA.2017.2715166.

[40] N. Moustafa, G. Creech, y J. Slay, «Big Data Analytics for Intrusion Detection System: Statistical Decision-Making Using Finite Dirichlet Mixture Models», 2017, pp. 127-156. doi: 10.1007/978-3-319-59439-2_5.

[41] M. Sarhan, S. Layeghy, N. Moustafa, y M. Portmann, «NetFlow Datasets for Machine Learning-Based Network Intrusion Detection Systems», 2021, pp. 117-135. doi: 10.1007/978-3-030-72802-1_9.

Anexo 1

En este anexo se adjuntan los *scripts* desarrollados.

A.1 *Script* de exploración inicial de los datasets.

Este *script* implementa la fase de exploración inicial de los conjuntos de datos CICIDS2017 y UNSW-NB15, permitiendo analizar su estructura, número de atributos, presencia de valores nulos y distribución de las clases.

Los resultados obtenidos mediante este script se describen en el apartado 6.2.5.1 Exploración inicial y análisis de características del documento.

Fuente: Elaboración propia.

```
import pandas as pd

#CICIDS2017

def explore_cicids():

    cicids_file = "data/temp/cicids2017_selected/Tuesday-WorkingHours.pcap_ISCX.csv"

    print("=== CICIDS2017 ===")

    df = pd.read_csv(cicids_file)

    print("Archivo:", cicids_file)

    print("Shape (filas, columnas):", df.shape)


    #Columnas y tipos

    print("\nPrimeras 15 columnas:")

    print(list(df.columns[:15]))

    print("\nTipos de datos (primeras 15):")

    print(df.dtypes.head(15))
```

```
    #Etiquetas
    if "Label" in df.columns:
        print("\nDistribución de etiquetas (Label):")
        print(df["Label"].value_counts().head(20))
    else:
        print("\n[ERROR] No se encontró la columna 'Label' en CICIDS2017.")

    #Valores perdidos
    print("\nValores nulos (Top 15):")
    print(df.isna().sum().sort_values(ascending=False).head(15))
#UNSW-NB15
def explore_unsw():
    unsw_file = "data/temp/unsw_nb15_selected/UNSW_NB15_training-set.csv"
    print("\n=== UNSW-NB15 ===")
    df = pd.read_csv(unsw_file)
    print("Archivo:", unsw_file)
    print("Shape (filas, columnas):", df.shape)

    #Columnas y tipos
    print("\nPrimeras 15 columnas:")
    print(list(df.columns[:15]))
    print("\nTipos de datos (primeras 15):")
    print(df.dtypes.head(15))

    #Etiquetas
```

```
    if "label" in df.columns:
        print("\nDistribución de etiquetas (label):")
        print(df["label"].value_counts())
    else:
        print("\n[ERROR] No se encontró la columna 'label' en UNSW-NB15.")

    #Valores perdidos
    print("\nValores nulos (Top 15):")
    print(df.isna().sum().sort_values(ascending=False).head(15))
if __name__ == "__main__":
    explore_cicids()
    explore_unsw()
```

A2. *Script* de limpieza y preprocesamiento de *dataset* CICIDS2017

Este *script* implementa el proceso de limpieza y preprocesamiento aplicado al conjunto de datos CICIDS2017, incluyendo la normalización de nombres de columnas, la identificación y transformación de la variable de etiquetado a un formato binario (tráfico benigno frente a tráfico malicioso) y la eliminación de registros con valores nulos o infinitos. Asimismo, el *script* genera un conjunto de datos depurado y coherente para su uso posterior en las fases de selección de características, normalización y entrenamiento de modelos supervisados.

Los resultados y efectos de este proceso se describen en el apartado 6.2.5.2 Limpieza y preprocesamiento de los datos del documento.

Fuente: Elaboración propia.

```
import pandas as pd
import numpy as np

INPUT_FILE = "data/temp/cicids2017_selected/Tuesday-WorkingHours.pcap_ISCX.csv"
OUTPUT_FILE = "data/processed/cicids2017_clean.csv"
```

```
print("Cargando CICIDS2017...")
df = pd.read_csv(INPUT_FILE)

#Normalizar nombres de columnas
df.columns = df.columns.str.strip()

#identificar columna de etiqueta
if "Label" not in df.columns:
    raise ValueError("No se encontró la columna de etiqueta en CICIDS2017")

#convertir etiquetas a binario
df["Label"] = df["Label"].apply(lambda x: 0 if x == "BENIGN" else 1)

#Reemplazar infinitos por NaN
df.replace([np.inf, -np.inf], np.nan, inplace=True)

#Eliminar filas con valores nulos
df.dropna(inplace=True)

#Eliminar columnas no numéricas irrelevantes
non_numeric = df.select_dtypes(include=["object"]).columns
df.drop(columns=non_numeric, inplace=True)

print("Dataset CICIDS2017 limpio:")
print(df.shape)
```

```
df.to_csv(OUTPUT_FILE, index=False)

print(f"Guardado en {OUTPUT_FILE}")
```

A.3 *Script* de limpieza y preprocesamiento del *dataset* UNSW-NB15

Este *script* automatiza la limpieza y preprocesamiento del conjunto de datos UNSW-NB15, aplicando un tratamiento orientado a obtener un *dataset* consistente y directamente utilizable por modelos de aprendizaje automático.

El proceso incluye la depuración de valores inconsistentes, el tratamiento de posibles registros anómalos y la eliminación o transformación de atributos categóricos que no pueden ser empleados directamente por modelos supervisados basados en variables numéricas, preservando en todo momento la variable objetivo (label) para su posterior evaluación.

Este procedimiento se corresponde con lo descrito en el apartado 6.2.5.2 Limpieza y preprocesamiento de los datos del documento.

Fuente: Elaboración propia.

```
import pandas as pd

import numpy as np

INPUT_FILE = "data/temp/unsw_nb15_selected/UNSW_NB15_training-set.csv"

OUTPUT_FILE = "data/processed/unsw_nb15_clean.csv"

print("Cargando UNSW-NB15...")

df = pd.read_csv(INPUT_FILE)


#Normalizar nombres

df.columns = df.columns.str.strip()


#Etiqueta ya binaria

if "label" not in df.columns:

    raise ValueError("No se encontró la columna label")


#Eliminar columnas categóricas
```

```
categorical = df.select_dtypes(include=["object"]).columns

df.drop(columns=categorical, inplace=True)


#Reemplazar infinitos y nulos

df.replace([np.inf, -np.inf], np.nan, inplace=True)

df.dropna(inplace=True)


print("Dataset UNSW-NB15 limpio:")

print(df.shape)


df.to_csv(OUTPUT_FILE, index=False)

print(f"Guardado en {OUTPUT_FILE}")
```

A.4 *Script* de selección de características y normalización

Este *script* implementa la etapa de preparación final de los datos previa al entrenamiento, aplicando la separación entre variables independientes (features) y la variable objetivo (label). Además, realiza el proceso de selección/depuración de características (manteniendo únicamente los atributos relevantes para el entrenamiento) y aplica una estandarización mediante Standard Scaling, transformando las variables para que presenten media cero y desviación estándar unitaria.

Como salida, el script genera los ficheros finales normalizados en formato CSV, listos para ser utilizados directamente en el entrenamiento y evaluación de modelos.

Este proceso se corresponde con el apartado 6.2.5.3 Selección de características y normalización del documento.

Fuente: Elaboración propia.

```
import pandas as pd

from sklearn.preprocessing import StandardScaler

#---------- CICIDS2017 ----------

print("Procesando CICIDS2017...")
```

```
cicids = pd.read_csv("data/processed/cicids2017_clean.csv")

X_cicids = cicids.drop(columns=["Label"])
y_cicids = cicids["Label"]

scaler_cicids = StandardScaler()
X_cicids_scaled = scaler_cicids.fit_transform(X_cicids)

cicids_ml = pd.DataFrame(X_cicids_scaled, columns=X_cicids.columns)
cicids_ml["Label"] = y_cicids.values

cicids_ml.to_csv("data/processed/cicids2017_ml.csv", index=False)

print("CICIDS2017 listo para entrenamiento:", cicids_ml.shape)
#---------- UNSW-NB15 ----------
print("Procesando UNSW-NB15...")
unsw = pd.read_csv("data/processed/unsw_nb15_clean.csv")

X_unsw = unsw.drop(columns=["label"])
y_unsw = unsw["label"]

scaler_unsw = StandardScaler()
X_unsw_scaled = scaler_unsw.fit_transform(X_unsw)

unsw_ml = pd.DataFrame(X_unsw_scaled, columns=X_unsw.columns)
unsw_ml["label"] = y_unsw.values
```

```
unsw_ml.to_csv("data/processed/unsw_nb15_ml.csv", index=False)


print("UNSW-NB15 listo para entrenamiento:", unsw_ml.shape)
```

A.5 *Script* de entrenamiento del modelo Random Forest

Este *script* ejecuta el entrenamiento y la validación del modelo supervisado basado en Random Forest, utilizando los conjuntos de datos previamente preprocesados.
El procedimiento incluye la carga de los datos finales, la partición en subconjuntos de entrenamiento y prueba siguiendo una proporción 70/30 (de forma estratificada para preservar la distribución de clases), el entrenamiento del modelo y la generación de predicciones sobre el conjunto de prueba.

Como resultado, el *script* calcula las métricas de evaluación definidas en el apartado 6.3 (Accuracy, Precision, Recall, F1-score y ROC-AUC) y genera la matriz de confusión asociada, que se emplea posteriormente para el análisis comparativo de resultados.

Los resultados derivados de este script se presentan en el apartado 6.4.2 Entrenamiento y validación de modelos y se analizan en el Capítulo 7.

Fuente: Elaboración propia.

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix


def run_experiment(dataset_path, label_col, name):

    print(f"\n=== Entrenando Random Forest con {name} ===")


    df = pd.read_csv(dataset_path)

    X = df.drop(columns=[label_col])

    y = df[label_col]
```

```
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42, stratify=y
    )

    model = RandomForestClassifier(
        n_estimators=100, random_state=42, n_jobs=-1
    )

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    print("Matriz de confusión:")
    print(confusion_matrix(y_test, y_pred))

    print("\nInforme de clasificación:")
    print(classification_report(y_test, y_pred))

    roc = roc_auc_score(y_test, y_prob)
    print(f"ROC-AUC: {roc:.4f}")

# CICIDS2017
run_experiment(
    "data/processed/cicids2017_ml.csv", "Label", "CICIDS2017"
```

```
)

#UNSW-NB15

run_experiment(

    "data/processed/unsw_nb15_ml.csv", "label", "UNSW-NB15
```