GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

SISTEMA VISUAL DE ALERTA TEMPRANA EN RED SIMULADA MEDIANTE PATRONES SIMPLES DE TRÁFICO

Autor: Paula Sánchez Bodas

Director: Alejandro García San Luis

Madrid

Declaración de originalidad

Declaro bajo mi responsabilidad que el Proyecto presentado con el título **Sistema visual de alerta temprana en red simulada mediante patrones simples de tráfico** en la ETS de Ingeniería – ICAI de la Universidad Pontificia Comillas en el curso académico 2025-2026 es de mi autoría y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Uso de Inteligencia Artificial[1]

Declaro bajo mi responsabilidad que (indicar la opción correcta):

☐ No he utilizado Inteligencia Artificial en la elaboración del presente documento.

☒ He utilizado Inteligencia Artificial en la elaboración del presente documento y/o del Anexo B siempre en las condiciones permitidas por la Universidad Pontificia Comillas, es decir, aplicando el Nivel 2 de la Escala de Evaluación de Perkins et al. (2024): "*La IA puede utilizarse para actividades previas a la tarea, como la lluvia de ideas, la descripción y la investigación inicial. Este nivel se centra en el uso de la IA para la planificación, las síntesis y la generación de ideas, pero las evaluaciones deben hacer hincapié en la capacidad de desarrollar y refinar estas ideas de forma independiente*". En concreto, la Inteligencia Artificial ha sido empleada para:

La inteligencia artificial ha sido utilizada como herramienta de apoyo para la estructuración del documento, mejora de redacción y generación de borradores, siendo posteriormente revisados, modificados y adaptados de forma crítica por la autora.


Firmado (alumno): Paula Sánchez Bodas
Fecha: 18/06/2026

[1] Esta declaración se refiere al uso de la Inteligencia Artificial generativa para realizar los documentos del Proyecto (Anexo B y Memoria). No aplica a Proyectos donde, por su naturaleza, deban emplear inteligencia artificial como parte de los mismos (aplicación de técnicas de aprendizaje automático, redes neuronales, análisis de datos…)

Autorización para la entrega del Proyecto

El Director del Proyecto	El co-Director del Proyecto (si aplica)
Firmado digitalmente por GARCIA SAN LUIS ALEJANDRO - 05266691J Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-05266691J, givenName=ALEJANDRO, sn=GARCIA SAN LUIS, cn=GARCIA SAN LUIS ALEJANDRO - 05266691J Fecha: 2026.06.19 08:42:05 +02'00'	
Fdo: Alejandro García San Luis	Fdo:
Fecha: 18/06/2026	Fecha:

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

SISTEMA VISUAL DE ALERTA TEMPRANA EN RED SIMULADA MEDIANTE PATRONES SIMPLES DE TRÁFICO

Autor: Paula Sánchez Bodas

Director: Alejandro García San Luis

Madrid

Agradecimientos

A mi tutor, por su orientación, seguimiento y apoyo constante durante todo el desarrollo de este Trabajo Fin de Grado. Su experiencia y sus recomendaciones han sido fundamentales para encauzar correctamente el proyecto, ayudándome a tomar decisiones técnicas adecuadas y a mejorar la calidad del trabajo en cada una de sus fases. Asimismo, agradezco el tiempo dedicado a la revisión de los avances y las indicaciones proporcionadas, que han contribuido de manera decisiva a la consecución de los objetivos planteados.

De igual manera, quiero expresar mi agradecimiento a mis compañeros, por su ayuda, sus consejos y por compartir conocimientos a lo largo del desarrollo del proyecto, lo que ha facilitado la resolución de diversas dificultades encontradas durante el proceso.

Finalmente, agradezco profundamente a mi familia y personas cercanas por su apoyo incondicional durante todo el periodo de realización de este Trabajo Fin de Grado. Su comprensión, ánimo y motivación han sido un pilar fundamental para poder llevar a cabo este proyecto hasta su finalización.

SISTEMA VISUAL DE ALERTA TEMPRANA EN RED SIMULADA MEDIANTE PATRONES SIMPLES DE TRÁFICO

Autor: Sánchez Bodas, Paula.
Director: García San Luis, Alejandro.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

El presente Trabajo Fin de Grado propone el diseño e implementación de un sistema de alerta temprana para la detección de comportamientos maliciosos en redes de comunicaciones mediante el análisis de patrones simples de tráfico. La solución desarrollada integra un entorno de red simulado utilizando Mininet, un sistema de detección de intrusos basado en firmas mediante Snort, un módulo de monitorización en Python y una interfaz gráfica que permite visualizar alertas en tiempo real.

La arquitectura del sistema reproduce un entorno real mediante el uso de Open vSwitch con técnicas de port mirroring, permitiendo capturar tráfico de red sin interferir en la comunicación entre nodos. Sobre este entorno, el sistema IDS analiza los paquetes utilizando reglas estándar y personalizadas, siendo capaz de detectar distintos tipos de ataques como SYN flood, escaneos de puertos, tráfico ICMP y tráfico UDP anómalo.

Adicionalmente, se ha incorporado un módulo basado en aprendizaje automático entrenado con el dataset CIC-IDS-2017, con el objetivo de evaluar la capacidad de un modelo de clasificación para distinguir entre tráfico benigno y malicioso. Tras un proceso de preprocesamiento y selección de características, se ha implementado un modelo Random Forest que alcanza una precisión aproximada del 68%, con una tasa de detección de ataques cercana al 67%.

Los resultados obtenidos validan el correcto funcionamiento de la plataforma desarrollada para la simulación, detección y visualización de eventos de seguridad en redes de comunicaciones. Asimismo, permiten evaluar la viabilidad de complementar los sistemas IDS tradicionales basados en firmas con técnicas de aprendizaje automático orientadas al análisis de tráfico de red.

Palabras clave: IDS, Ciberseguridad, Snort, Mininet, Machine Learning, Detección de intrusos

1. Introducción

En los últimos años, el crecimiento exponencial de las redes de comunicaciones y la digitalización de los servicios han incrementado significativamente la superficie de ataque de los sistemas informáticos. Como consecuencia, la detección temprana de amenazas se ha convertido en un elemento clave dentro de la ciberseguridad.

Los sistemas de detección de intrusos (IDS) permiten monitorizar el tráfico de red y generar alertas ante posibles comportamientos maliciosos. Sin embargo, los enfoques tradicionales basados en firmas presentan limitaciones frente a ataques desconocidos o variantes de ataques existentes.

En este contexto, el presente trabajo aborda el desarrollo de un sistema capaz de simular un entorno de red, detectar eventos sospechosos y visualizar alertas en tiempo real, combinando técnicas clásicas de detección con enfoques modernos basados en aprendizaje automático.

2. Definición del Proyecto

El objetivo del proyecto es diseñar e implementar un sistema de alerta temprana que permita identificar comportamientos maliciosos en una red simulada mediante el análisis de patrones de tráfico.

Para ello, se ha desarrollado una solución completa que integra:

- Un entorno de red virtual basado en Mininet
- Un sistema IDS basado en Snort
- Un módulo de monitorización en Python
- Una interfaz gráfica de visualización
- Un modelo de Machine Learning para clasificación de tráfico

El sistema permite generar tráfico de red, simular ataques y analizar su detección en tiempo real, proporcionando una plataforma experimental para el estudio de técnicas de ciberseguridad.

3. Descripción del modelo/sistema/herramienta

El sistema desarrollado se estructura en dos bloques principales:

Por un lado, el entorno de red simulado, compuesto por tres hosts y un switch virtual, permite generar tráfico y ejecutar el sistema de detección basado en firmas. Mediante el uso de Open vSwitch y técnicas de port mirroring, el tráfico es replicado hacia un host sensor donde se ejecuta Snort.

El sistema IDS analiza el tráfico mediante reglas predefinidas y personalizadas, detectando distintos tipos de ataques como SYN flood, escaneos de puertos o tráfico anómalo.

Por otro lado, se ha desarrollado un módulo de monitorización en Python que procesa las alertas generadas y una interfaz gráfica que permite visualizar en tiempo real el estado del sistema y los eventos detectados.

Adicionalmente, el sistema incorpora un módulo de aprendizaje automático basado en Random Forest, entrenado con el dataset CIC-IDS-2017, que permite evaluar la detección de tráfico malicioso mediante técnicas de clasificación.

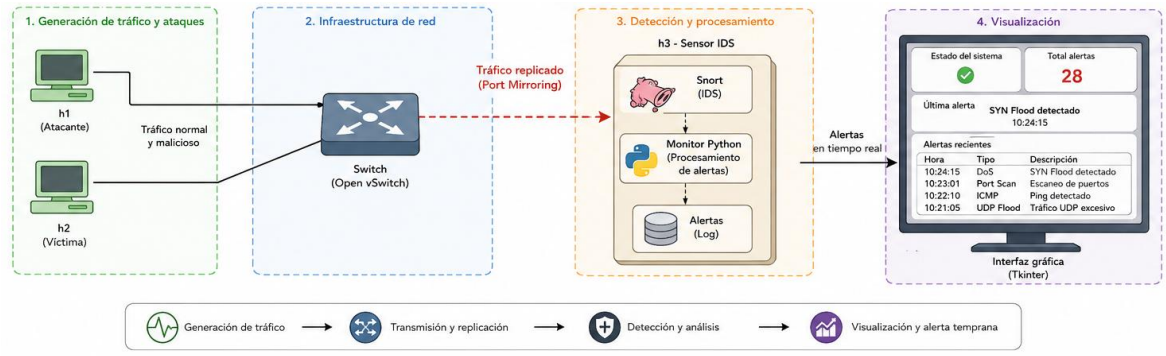

Ilustración 1 - Esquema de la arquitectura general del sistema

4. Resultados

Los resultados obtenidos muestran que el sistema es capaz de detectar correctamente distintos tipos de ataques en el entorno simulado mediante el uso de reglas en Snort, generando alertas en tiempo real que son visualizadas a través de la interfaz gráfica.

En cuanto al módulo de aprendizaje automático, el modelo Random Forest alcanza una precisión aproximada del 68%, siendo capaz de identificar alrededor del 67% de los ataques presentes en el conjunto de datos. No obstante, se observa una tasa de falsas alarmas cercana al 31%, lo que indica margen de mejora en la clasificación.

Estos resultados validan la funcionalidad del sistema y demuestran la viabilidad de combinar enfoques basados en firmas y aprendizaje automático para la detección de intrusos.

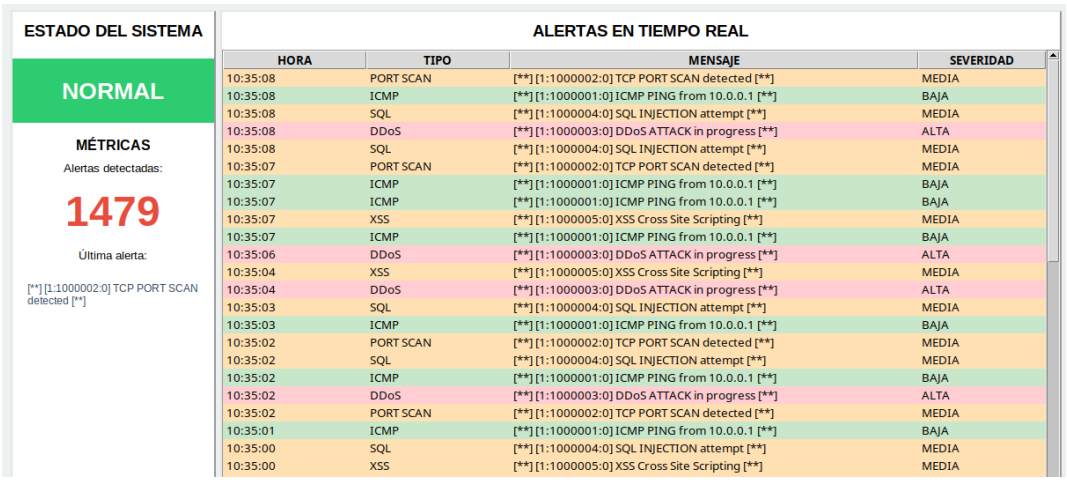

Ilustración 2 - Simulación del sistema de detección de alertas

5. Conclusiones

El trabajo desarrollado demuestra la viabilidad de implementar un sistema completo de detección de intrusos utilizando herramientas de código abierto y técnicas modernas de análisis de datos.

La solución propuesta permite simular una red, generar ataques, detectarlos y visualizar las alertas en tiempo real, constituyendo una plataforma útil para el estudio de mecanismos de ciberseguridad.

Asimismo, la incorporación de un modelo de aprendizaje automático pone de manifiesto el potencial de estas técnicas para complementar los sistemas tradicionales, aunque también evidencia la necesidad de mejorar su precisión y reducir las falsas alarmas.

6. Referencias

[1] Roesch, M. "Snort - Lightweight Intrusion Detection for Networks", 1999.

[2] Sharafaldin, I., Lashkari, A. H., Ghorbani, A. A. "Toward Generating a New Intrusion Detection Dataset", 2018.

[3] Breiman, L. "Random Forests", Machine Learning, 2001.

[4] Mininet Documentation.

[5] Open vSwitch Documentation.

EARLY WARNING VISUAL SYSTEM IN A SIMULATED NETWORK USING SIMPLE TRAFFIC PATTERNS

Author: Sánchez Bodas, Paula.
Supervisor: García San Luis, Alejandro
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This Final Degree Project presents the design and implementation of an early warning system for intrusion detection in network environments using simple traffic pattern analysis. The proposed solution integrates a simulated network environment using Mininet, a signature-based Intrusion Detection System (IDS) using Snort, a real-time monitoring module developed in Python, and a graphical interface for visualization of alerts.

The system architecture reproduces real-world scenarios by leveraging Open vSwitch with port mirroring techniques, allowing traffic to be captured without interfering with network communications. The IDS analyzes packets using both standard and custom rules, detecting various types of attacks such as SYN flood, port scanning, ICMP traffic, and abnormal UDP traffic.

Additionally, a machine learning module has been developed using the CIC-IDS-2017 dataset. After preprocessing and feature selection, a Random Forest classifier was trained to distinguish between benign and malicious traffic, achieving approximately 68% accuracy and a detection rate close to 67%.

The results obtained validate the correct operation of the platform developed for the simulation, detection, and visualization of security events in communication networks. Furthermore, they allow the feasibility of complementing traditional signature-based IDS systems with machine learning techniques aimed at network traffic analysis to be evaluated.

Keywords: Intrusion Detection System, Cybersecurity, Machine Learning, Network Security, Mininet

1. Introduction

In recent years, the rapid growth of communication networks and the digitalization of services have significantly increased the attack surface of information systems. As a result, early detection of cybersecurity threats has become a critical requirement in modern network infrastructures.

Intrusion Detection Systems (IDS) play a key role in this context, as they allow continuous monitoring of network traffic and the identification of potentially malicious activities. However, traditional signature-based approaches present limitations when dealing with unknown or evolving threats.

In this context, this project focuses on the development of a system capable of simulating a network environment, detecting suspicious events, and visualizing alerts in real time, combining classical detection techniques with modern machine learning approaches.

2. Project Definition

The main objective of this project is to design and implement an early warning system capable of identifying malicious behavior in a simulated network environment through the analysis of simple traffic patterns.

To achieve this goal, a complete solution has been developed, integrating the following components:

- A virtual network environment based on Mininet
- A signature-based Intrusion Detection System using Snort
- A real-time monitoring module developed in Python
- A graphical interface for visualization
- A Machine Learning model for traffic classification

The system enables the generation of network traffic, simulation of attacks, and real-time analysis of detection results, providing an experimental platform for cybersecurity research and validation.

3. Model Description/System/Tool

The developed system is structured into two main components.

On the one hand, the simulated network environment, composed of three hosts and a virtual switch, allows the generation of network traffic and the execution of a signature-based intrusion detection system. Using Open vSwitch and port mirroring techniques, network traffic is replicated to a dedicated sensor host where Snort is deployed.

The IDS analyzes the captured traffic using both predefined and custom rules, enabling the detection of various types of attacks such as SYN flood, port scanning, ICMP traffic, and anomalous UDP traffic.

On the other hand, a monitoring module developed in Python processes the alerts generated by the IDS in real time. These alerts are displayed through a graphical interface, providing a clear and structured visualization of detected events.

Additionally, the system includes a Machine Learning module based on a Random Forest classifier, trained using the CIC-IDS-2017 dataset. This module evaluates the ability of classification models to distinguish between benign and malicious traffic.

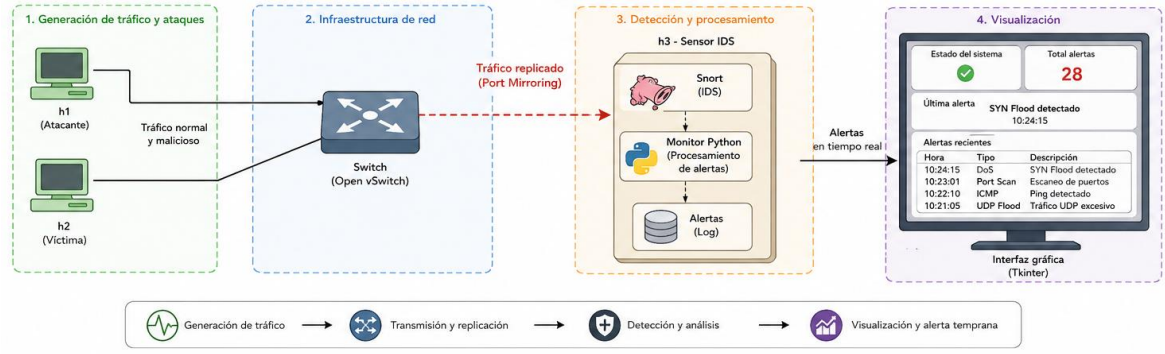

Illustration 1 - General system architecture diagram

4. Results

The results obtained demonstrate that the system is capable of accurately detecting different types of attacks within the simulated environment using the rule-based approach implemented in Snort. Alerts are generated in real time and successfully visualized through the graphical interface.

Regarding the Machine Learning module, the Random Forest classifier achieves an overall accuracy of approximately 68%, with a detection rate of around 67% for malicious traffic. However, the system presents a false positive rate of approximately 31%, indicating that further improvements are required.

These results validate the functionality of the proposed system and highlight the feasibility of combining signature-based detection techniques with machine learning approaches for intrusion detection.

Illustration 2 - Alert Detection System Simulation

5. Conclusions

The work carried out demonstrates the feasibility of implementing a complete intrusion detection system using open-source tools and modern data analysis techniques.

The proposed solution allows the simulation of a network environment, the generation of attack scenarios, the detection of malicious activities, and the real-time visualization of alerts, providing a comprehensive platform for cybersecurity experimentation.

Furthermore, the inclusion of a Machine Learning module highlights the potential of data-driven approaches to complement traditional detection systems, although it also reveals the need for further optimization to reduce false positives and improve overall performance.

6. References

[1] Roesch, M. “Snort – Lightweight Intrusion Detection for Networks”, 1999.

[2] Sharafaldin, I., Lashkari, A. H., Ghorbani, A. A. “Toward Generating a New Intrusion Detection Dataset”, 2018.

[3] Breiman, L. “Random Forests”, Machine Learning, 2001.

[4] Mininet Documentation.

[5] Open vSwitch Documentation.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

En la actualidad, las redes de comunicación desempeñan un papel esencial en el funcionamiento de los sistemas digitales, lo que ha convertido a la ciberseguridad en un elemento imprescindible para garantizar la protección y estabilidad de las infraestructuras tecnológicas modernas. El crecimiento exponencial del tráfico de datos y la sofisticación progresiva de las amenazas informáticas han generado la necesidad de implementar mecanismos capaces de detectar de manera temprana comportamientos anómalos que puedan derivar en incidentes de seguridad relevantes. En este escenario, la capacidad de supervisar, analizar y representar visualmente el tráfico de red en tiempo real resulta fundamental para mantener entornos de comunicación seguros y eficientes.

El presente Trabajo Fin de Grado se centra en el diseño e implementación de un sistema visual de alerta temprana aplicado a una red simulada, orientado especialmente a la identificación de patrones básicos de tráfico anómalo. Mediante la creación de un entorno virtualizado y la utilización de herramientas de análisis y detección, se busca mostrar de forma clara, comprensible y didáctica el proceso de identificación de actividades sospechosas, como escaneos de puertos, ataques de flooding o conexiones con comportamientos inusuales.

Además de la detección de anomalías, el proyecto incorpora un componente interactivo y visual destinado a facilitar la interpretación de los eventos generados dentro de la red. Para ello, se plantea el desarrollo de un panel de monitorización dinámico desde el cual el sistema pueda emitir alertas, simulando así el funcionamiento básico de un Centro de Operaciones de Seguridad (SOC). Este enfoque permitirá que el usuario comprenda de manera más intuitiva el comportamiento y la utilidad de los sistemas de alerta temprana utilizados en entornos reales de ciberseguridad.

Por tanto, el principal problema que pretende abordar este proyecto es la escasez de herramientas educativas que integren de manera conjunta la simulación de redes, la

detección de tráfico anómalo y la visualización de alertas en tiempo real. Esta carencia dificulta que los estudiantes puedan entender de forma práctica e intuitiva cómo se comporta una red frente a actividades maliciosas y cuál es el impacto de este tipo de tráfico sobre los sistemas de comunicación. Asimismo, el trabajo busca reducir la distancia existente entre el ámbito académico y el profesional mediante la creación de una solución funcional y realista adaptada a fines formativos. Aunque las herramientas profesionales disponibles ofrecen amplias capacidades de análisis y protección, su complejidad y dificultad de configuración hacen que, en muchos casos, no resulten adecuadas para entornos educativos.

1.1 MOTIVACIÓN DEL PROYECTO

El crecimiento continuo de los ciberataques dirigidos tanto a entornos empresariales como domésticos ha evidenciado la necesidad de contar con sistemas capaces de detectar amenazas de forma anticipada y responder rápidamente ante posibles incidentes. En la actualidad, muchos ataques no se basan únicamente en la explotación directa de vulnerabilidades, sino también en patrones de tráfico anómalos difíciles de identificar sin herramientas adecuadas de supervisión. Por ello, disponer de mecanismos de monitorización, análisis y representación de datos en tiempo real resulta fundamental para preservar la seguridad, integridad y disponibilidad de los sistemas de comunicación.

No obstante, dentro del ámbito académico, los sistemas de detección de intrusiones y las plataformas de alerta temprana suelen abordarse desde una perspectiva excesivamente teórica o mediante prácticas poco dinámicas, lo que dificulta que los estudiantes comprendan de manera clara las consecuencias reales de un ciberataque o del tráfico malicioso sobre una red. Esta separación entre los conceptos teóricos y su aplicación práctica reduce significativamente la capacidad de asimilación de los riesgos asociados a la ciberseguridad.

En este contexto, la motivación principal de este proyecto surge de la necesidad de desarrollar una herramienta didáctica, interactiva y visual que permita comprender de forma práctica el funcionamiento de un sistema de alerta temprana orientado a la ciberseguridad. A diferencia de las soluciones comerciales o profesionales, diseñadas para proteger

infraestructuras críticas complejas, el presente trabajo pone el foco en el aprendizaje y la comprensión de los principios fundamentales, priorizando el valor educativo frente a la complejidad técnica.

El sistema propuesto pretende acercar al estudiante a un entorno lo más realista posible, donde pueda observar el comportamiento del tráfico dentro de una red simulada, identificar cómo se producen las anomalías y analizar la reacción del sistema ante dichas situaciones. La incorporación de mecanismos de alerta en tiempo real, como notificaciones emergentes, mensajes SMS o correos electrónicos, transforma el proceso de aprendizaje en una experiencia más dinámica e inmersiva, favoreciendo la interiorización de los conceptos mediante la experimentación directa.

Además, este trabajo fomenta un enfoque multidisciplinar al integrar conocimientos relacionados con redes de comunicaciones, programación, herramientas de monitorización, representación visual de datos y seguridad informática. Desde el punto de vista formativo, el proyecto también contribuye al desarrollo de capacidades analíticas, pensamiento crítico y toma de decisiones en escenarios vinculados a la protección de redes y sistemas.

En conclusión, la motivación de este trabajo reside en la necesidad de ofrecer una solución accesible, práctica y visual que facilite el aprendizaje de los fundamentos de los sistemas de alerta temprana en ciberseguridad, al tiempo que permita adquirir competencias técnicas de especial relevancia en el contexto tecnológico y digital actual.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

2.1 INTRODUCCIÓN

El desarrollo de sistemas modernos de detección de intrusos requiere la integración de múltiples tecnologías relacionadas con la simulación de redes, el análisis de tráfico, la monitorización de eventos de seguridad y el aprendizaje automático. En el presente Trabajo Fin de Grado se emplean diversas herramientas y protocolos ampliamente utilizados tanto en investigación como en entornos profesionales de ciberseguridad [1].

En la actualidad, el crecimiento exponencial del tráfico de red, la complejidad de las arquitecturas distribuidas y la aparición constante de nuevas amenazas han hecho necesario el desarrollo de sistemas de seguridad más avanzados y adaptativos. Por este motivo, los sistemas de detección de intrusos ya no se limitan únicamente a la identificación de patrones conocidos, sino que se integran con técnicas de análisis inteligente de datos y modelos de aprendizaje automático capaces de detectar comportamientos anómalos.

La finalidad de este capítulo es describir en profundidad las principales tecnologías utilizadas durante el desarrollo del sistema propuesto, proporcionando el contexto técnico necesario para comprender la arquitectura implementada y las decisiones de diseño adoptadas a lo largo del proyecto. Además, se busca justificar la elección de cada tecnología en función de su relevancia dentro del ámbito de la ciberseguridad moderna.

El sistema desarrollado integra herramientas de simulación de redes, mecanismos de monitorización pasiva, sistemas IDS basados en firmas y modelos de Machine Learning orientados a la clasificación de tráfico malicioso. La combinación de estas tecnologías permite construir una plataforma experimental completa para el análisis y detección de amenazas en redes de comunicaciones, proporcionando un entorno controlado pero representativo de escenarios reales.

2.2 SISTEMAS DE DETECCIÓN DE INTRUSOS (IDS)

Los Sistemas de Detección de Intrusos (Intrusion Detection Systems, IDS) son herramientas de seguridad diseñadas para monitorizar el tráfico de red o la actividad de un sistema con el objetivo de identificar comportamientos sospechosos, accesos no autorizados o posibles ataques informáticos [2].

Estos sistemas desempeñan un papel fundamental dentro de las infraestructuras de ciberseguridad modernas, ya que actúan como una capa de defensa capaz de detectar amenazas que han conseguido superar otros mecanismos de protección como firewalls o sistemas de autenticación. Su principal función no es necesariamente bloquear el tráfico, sino alertar sobre actividades potencialmente peligrosas para que puedan ser analizadas por un administrador o por sistemas automatizados de respuesta.

El funcionamiento general de un IDS se basa en la captura continua de eventos o tráfico de red, seguido de su análisis mediante un conjunto de reglas, modelos estadísticos o técnicas de aprendizaje automático. Dependiendo del enfoque utilizado, los IDS pueden clasificarse en varias categorías, siendo las más importantes los sistemas basados en firmas y los sistemas basados en anomalías.

2.2.1 IDS BASADOS EN FIRMAS

Los IDS basados en firmas identifican ataques comparando el tráfico observado con un conjunto de patrones o reglas previamente definidos. Estos patrones representan comportamientos conocidos asociados a ataques concretos, como escaneos de puertos, denegaciones de servicio o explotación de vulnerabilidades.

Cuando un paquete o flujo de red coincide con una firma existente, el sistema genera una alerta inmediata. Este enfoque es altamente eficiente en la detección de amenazas conocidas y presenta un nivel muy bajo de falsos positivos cuando está correctamente configurado.

Sin embargo, su principal limitación radica en la dependencia de bases de conocimiento actualizadas. Esto significa que cualquier ataque nuevo o variante modificada de un ataque existente puede pasar desapercibido si no ha sido previamente incorporada al sistema de

reglas. Por este motivo, los IDS basados en firmas requieren mantenimiento constante y actualización frecuente [3].

2.2.2 IDS BASADOS EN ANOMALÍAS

Los sistemas basados en anomalías adoptan un enfoque diferente, basado en la identificación de comportamientos que se desvían del patrón normal de la red. En lugar de buscar ataques conocidos, estos sistemas construyen un modelo del comportamiento habitual del sistema y detectan cualquier desviación significativa respecto a dicho modelo.

Este enfoque permite detectar ataques desconocidos o de día cero, lo que supone una ventaja importante frente a los sistemas basados en firmas. Sin embargo, su principal inconveniente es la dificultad de definir con precisión qué se considera comportamiento normal, especialmente en redes complejas o con tráfico muy variable.

Como consecuencia, estos sistemas tienden a generar un mayor número de falsos positivos, lo que puede dificultar su uso en entornos productivos sin un ajuste fino previo [4].

En este proyecto se adopta un enfoque híbrido, combinando un IDS basado en firmas mediante Snort con un modelo de Machine Learning supervisado, con el objetivo de aprovechar las ventajas de ambos enfoques y mitigar sus limitaciones.

2.3 SNORT

Snort es uno de los sistemas IDS de código abierto más utilizados en el ámbito de la ciberseguridad. Fue desarrollado originalmente por Martin Roesch y actualmente es mantenido por Cisco Systems [5].

Su relevancia dentro del ecosistema de seguridad se debe a su flexibilidad, su capacidad de análisis en tiempo real y su amplia comunidad de soporte. Snort puede funcionar como sniffer de paquetes, como sistema de registro o como sistema completo de detección de intrusos, siendo este último el modo utilizado en este proyecto.

El funcionamiento de Snort se basa en la captura de paquetes de red y su comparación con un conjunto de reglas definidas por el usuario o incluidas por defecto. Estas reglas permiten

identificar patrones específicos en el tráfico, como direcciones IP concretas, puertos, protocolos, flags TCP o incluso contenido dentro de los paquetes.

Cuando una regla coincide con un paquete analizado, Snort genera una alerta que puede ser almacenada en archivos de log o enviada a sistemas externos de monitorización. Esta capacidad lo convierte en una herramienta especialmente útil para entornos de análisis forense y monitorización de redes.

En el contexto de este proyecto, Snort se utiliza para analizar tráfico replicado dentro de una red virtual, permitiendo detectar ataques simulados como ICMP flood, SYN flood, escaneos de puertos y tráfico UDP anómalo asociado a protocolos modernos como HTTP3/QUIC.

2.4 MININET

Mininet es una herramienta de virtualización ligera utilizada para la simulación de redes de comunicaciones y entornos SDN (Software Defined Networking) [6].
A diferencia de otros simuladores más pesados, Mininet permite ejecutar redes completas dentro de un único sistema operativo real utilizando mecanismos de virtualización a nivel de kernel. Esto lo convierte en una herramienta extremadamente eficiente para la experimentación en entornos académicos y de investigación.
Mininet permite la creación de topologías compuestas por hosts, switches, routers y controladores SDN, todo ello mediante interfaces virtuales y espacios de nombres de red. Esta flexibilidad facilita la simulación de escenarios realistas sin necesidad de hardware adicional.
En este proyecto, Mininet se utiliza para construir una topología controlada que permite generar tráfico entre hosts y simular ataques en un entorno aislado. Esta característica resulta fundamental para garantizar la reproducibilidad de los experimentos.

2.5 OPEN vSWITCH

Open vSwitch (OVS) es un switch virtual multicapa diseñado para entornos virtualizados y arquitecturas SDN [7].

Su principal objetivo es proporcionar funcionalidades avanzadas de conmutación de red dentro de entornos virtuales, permitiendo la gestión eficiente del tráfico entre máquinas virtuales o contenedores.

Open vSwitch soporta múltiples características avanzadas como VLANs, QoS, túneles de red y, especialmente relevante para este proyecto, port mirroring. Esta funcionalidad permite duplicar el tráfico de determinados puertos hacia un puerto de monitorización, lo que resulta esencial para la captura pasiva de tráfico en sistemas IDS.

En el sistema desarrollado, Open vSwitch actúa como el núcleo de la infraestructura de red, permitiendo la comunicación entre hosts y la replicación del tráfico hacia el sistema de monitorización.

2.6 PORT MIRRORING

El port mirroring es una técnica utilizada en redes de comunicaciones para replicar el tráfico de uno o varios puertos hacia un puerto específico destinado al análisis [8].

Esta técnica es ampliamente utilizada en sistemas IDS y sistemas de monitorización de red, ya que permite capturar tráfico sin interferir en la comunicación original ni modificar los paquetes.

En este proyecto, el port mirroring se implementa mediante Open vSwitch, replicando todo el tráfico entre los hosts *h1* y *h2* hacia el nodo sensor *h3* donde se ejecuta Snort. Esto permite realizar un análisis completo del tráfico en tiempo real sin afectar al rendimiento de la red virtual.

2.7 MACHINE LEARNING APLICADO A CIBERSEGURIDAD

El aprendizaje automático es una rama de la inteligencia artificial que permite desarrollar modelos capaces de aprender patrones a partir de datos y realizar predicciones o clasificaciones [9].

En ciberseguridad, estas técnicas se utilizan para detectar patrones complejos que no pueden ser fácilmente identificados mediante reglas definidas. Su aplicación en sistemas IDS permite mejorar la detección de amenazas desconocidas y reducir la dependencia de firmas predefinidas.

Sin embargo, su uso también implica desafíos importantes, como la necesidad de grandes volúmenes de datos, la sensibilidad a datos desbalanceados y la dificultad de interpretación de los modelos.

2.8 DATASET CIC-IDS-2017

El dataset CIC-IDS-2017 es uno de los conjuntos de datos más utilizados en investigación sobre detección de intrusiones [10].

Este dataset contiene tráfico de red real generado en entornos controlados, etiquetado con múltiples tipos de ataques y tráfico benigno. Su principal ventaja es que permite entrenar modelos de Machine Learning en condiciones cercanas a escenarios reales.

Incluye diferentes tipos de ataques como DDoS, fuerza bruta, escaneos de puertos, botnets y ataques web. En este proyecto se utiliza un subconjunto reducido del dataset original centrado en tráfico benigno y escaneos de puertos, con el objetivo de simplificar el problema de clasificación.

2.9 RANDOM FOREST

Random Forest es un algoritmo de aprendizaje supervisado basado en conjuntos de árboles de decisión propuesto por Leo Breiman [11].

Este algoritmo combina múltiples árboles de decisión entrenados con subconjuntos aleatorios de datos y características, obteniendo una predicción final mediante votación o promedio.

Su principal ventaja es la robustez frente al sobreajuste y su capacidad para manejar datasets con alta dimensionalidad y ruido. Además, ofrece buenos resultados sin necesidad de una parametrización excesivamente compleja.

En este proyecto se utiliza Random Forest como modelo principal para clasificar tráfico de red en benigno o malicioso.

2.10 CONCLUSIÓN

Las tecnologías descritas en este capítulo constituyen la base del sistema desarrollado a lo largo del proyecto. La integración de herramientas de simulación de redes, sistemas IDS y técnicas de aprendizaje automático permite construir una plataforma experimental completa que analiza y detecta amenazas en redes de comunicaciones.

La combinación de estos elementos proporciona además un entorno flexible, reproducible y escalable para el estudio de distintos mecanismos de detección de intrusiones, permitiendo evaluar tanto enfoques tradicionales basados en firmas como técnicas modernas de análisis de datos aplicadas a la ciberseguridad.

Capítulo 3. ESTADO DE LA CUESTIÓN

El estado de la cuestión en detección de tráfico anómalo y sistemas de alerta temprana ha evolucionado significativamente en los últimos años. Los sistemas de detección de intrusiones (IDS) se clasifican principalmente según su método de análisis, pudiendo ser basados en firmas o basados en anomalías.

Los IDS basados en firmas, como Snort y Suricata, funcionan identificando ataques a partir de reglas predefinidas que describen patrones característicos de tráfico malicioso [5][12]. Estos sistemas destacan por su efectividad al ser precisos frente a amenazas conocidas, pero presentan dificultades frente a ataques nuevos o desconocidos. Por otro lado, los IDS basados en anomalías, como Zeek, se focalizan en establecer un comportamiento "normal" de tráfico para la red o el equipo y así detectan desviaciones respecto a ese patrón [13]. Este enfoque tiene la capacidad de identificar ataques desconocidos, pero también puede generar falsos positivos.

Desde el punto de vista académico, el uso de plataformas como Mininet ha permitido la creación de entornos virtualizados en los que se dispone de la posibilidad de probar y comparar estos sistemas sin necesidad de hardware dedicado [14]. Mininet ofrece una simulación realista de redes complejas sobre un solo equipo físico, lo que lo convierte en una herramienta de especial utilidad para el aprendizaje experimental.

En los últimos años, diversos estudios han explorado la combinación de IDS con sistemas de visualización para facilitar la interpretación de los resultados. Por ejemplo, Chen et al. desarrollaron una interfaz gráfica basada en dashboards para representar eventos detectados por Snort, mientras que Gómez et al. [15] propusieron un entorno educativo interactivo para enseñar ciberseguridad mediante simulación y visualización de alertas.

A pesar de estos avances, todavía existe una carencia de herramientas que integren de manera completa la detección, simulación y visualización, y que además sean sencillas y

reproducibles en entornos académicos. En este contexto, el proyecto aporta una maqueta práctica que combina tecnologías consolidadas con una parte visual e intuitiva orientada al aprendizaje, contribuyendo así a la formación en el ámbito de la ciberseguridad.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

La creciente digitalización de los servicios y la expansión continua de las infraestructuras de comunicaciones han provocado en los últimos años un incremento significativo tanto del volumen de tráfico de red como de la diversidad de amenazas informáticas que afectan a sistemas conectados. En la actualidad, el tráfico que circula por redes corporativas y por Internet no está compuesto únicamente por comunicaciones legítimas entre usuarios y servicios, sino que también incluye de forma constante actividades maliciosas automatizadas, tales como escaneos masivos de puertos, intentos de explotación de vulnerabilidades, campañas de denegación de servicio distribuido, ataques de fuerza bruta o técnicas de reconocimiento previo a intrusiones más complejas. Este tipo de comportamientos forma ya parte del "ruido normal" presente en cualquier entorno conectado, lo que incrementa considerablemente la dificultad de su identificación y análisis [1].

En este contexto, los mecanismos de detección temprana de amenazas adquieren un papel fundamental dentro de las arquitecturas modernas de ciberseguridad. Los sistemas IDS tradicionales siguen siendo ampliamente utilizados en entornos reales debido a su capacidad para identificar amenazas conocidas mediante reglas y firmas previamente definidas, lo que les proporciona una alta fiabilidad en escenarios controlados y ante ataques ya catalogados [2]. Sin embargo, a pesar de su eficacia, estas soluciones presentan limitaciones importantes cuando se enfrentan a entornos dinámicos y altamente cambiantes, donde aparecen nuevas variantes de ataques o comportamientos que no han sido previamente modelados. Entre estas limitaciones destacan la dificultad para detectar amenazas desconocidas, la dependencia constante de la actualización manual de reglas y la generación de un volumen elevado de alertas que, en muchos casos, pueden resultar redundantes o incluso falsas, dificultando la labor de los analistas de seguridad.

A esta problemática se suma otro factor relevante en el ámbito actual de la ciberseguridad, que es la complejidad creciente de los sistemas de monitorización y análisis de eventos. Muchas de las plataformas comerciales utilizadas en centros de operaciones de seguridad (SOC) presentan elevados costes económicos, requieren infraestructuras complejas y dependen de licencias propietarias, lo que limita su accesibilidad en entornos académicos, laboratorios de investigación o proyectos formativos. Esta situación genera una barrera importante para el aprendizaje práctico de técnicas de detección de intrusos y análisis de tráfico de red en contextos educativos.

A partir de esta problemática surge la motivación principal del presente Trabajo Fin de Grado, que consiste en el diseño y desarrollo de una plataforma experimental de detección de intrusos capaz de integrar en un único sistema la simulación de redes, la detección basada en firmas, el análisis inteligente mediante técnicas de aprendizaje automático y la visualización de eventos en tiempo real. El objetivo no es replicar la complejidad de soluciones comerciales de gran escala, sino construir un entorno funcional, reproducible y flexible que permita estudiar el comportamiento de distintos mecanismos de detección en condiciones controladas.

El sistema propuesto permite, en este sentido, simular tráfico de red benigno y malicioso, analizar dicho tráfico mediante reglas IDS, visualizar las alertas generadas de forma comprensible para el usuario y complementar este análisis con modelos de Machine Learning capaces de identificar patrones adicionales en los datos. Esta combinación de enfoques tradicionales y modernos proporciona una visión más completa del problema de la detección de intrusiones y permite evaluar sus ventajas y limitaciones dentro de un mismo entorno experimental.

Desde una perspectiva global, la principal aportación del sistema desarrollado reside en la integración de múltiples tecnologías de código abierto dentro de una única arquitectura funcional. Esta integración permite construir una plataforma accesible, extensible y adecuada tanto para fines académicos como para investigación, facilitando el estudio de técnicas de detección de amenazas sin necesidad de infraestructuras costosas o entornos

propietarios. Además, el sistema ofrece un entorno controlado que puede ser utilizado para reproducir escenarios de ataque de forma segura, lo que resulta especialmente útil en contextos educativos.

Asimismo, el proyecto presenta un interés multidisciplinar, ya que combina aspectos académicos, técnicos, investigadores y educativos. Desde el punto de vista académico, permite aplicar conocimientos adquiridos en áreas como redes de comunicaciones, seguridad informática y programación. Desde el punto de vista técnico, integra herramientas reales utilizadas en entornos profesionales de ciberseguridad. Desde la perspectiva investigadora, facilita la experimentación con tráfico malicioso en entornos controlados. Y desde el punto de vista educativo, proporciona una plataforma visual e intuitiva que ayuda a comprender el funcionamiento interno de los sistemas de detección de intrusos.

En conjunto, la incorporación de un módulo basado en Machine Learning añade un componente adicional de innovación al sistema, ya que permite explorar enfoques modernos basados en el análisis de datos y la identificación de patrones complejos dentro del tráfico de red. Aunque estos modelos no sustituyen a los sistemas tradicionales basados en firmas, sí ofrecen una capa complementaria de análisis que puede mejorar la capacidad general de detección en determinados escenarios [4][16]. Los resultados obtenidos en este trabajo, aunque limitados en precisión, demuestran la viabilidad de aplicar técnicas de aprendizaje automático sobre conjuntos de datos reales como CIC-IDS-2017, ampliamente utilizado en investigación en ciberseguridad [10].

En consecuencia, el presente trabajo queda justificado por la necesidad de disponer de plataformas experimentales accesibles, reproducibles y extensibles que permitan combinar técnicas tradicionales y modernas de detección de intrusiones dentro de un entorno controlado, facilitando tanto la investigación como la formación en el ámbito de la ciberseguridad.

4.1.1 NECESIDAD DE SISTEMAS DE DETECCIÓN TEMPRANA

Uno de los principales retos en el ámbito de la ciberseguridad moderna consiste en la detección temprana de amenazas antes de que estas logren comprometer de manera significativa la infraestructura afectada. Muchos ataques no comienzan directamente con una acción destructiva, sino que se desarrollan de forma progresiva mediante fases de reconocimiento, exploración de servicios o generación de tráfico anómalo que, en un primer momento, puede pasar desapercibido dentro del flujo habitual de la red.

La detección temprana de este tipo de comportamientos resulta fundamental, ya que permite reducir los tiempos de respuesta ante incidentes de seguridad, minimizar el impacto potencial de un ataque, facilitar el análisis forense posterior y mejorar de forma general la capacidad de supervisión de la infraestructura. En este sentido, los sistemas IDS continúan siendo un componente esencial dentro de las arquitecturas de defensa en profundidad, ya que permiten identificar actividad sospechosa en tiempo real y generar alertas que pueden ser analizadas por personal especializado.

4.1.2 LIMITACIONES DE LOS SISTEMAS IDS TRADICIONALES

Los sistemas IDS basados exclusivamente en firmas presentan una serie de limitaciones que se hacen especialmente evidentes en entornos modernos caracterizados por la alta variabilidad del tráfico y la aparición constante de nuevas amenazas. Aunque su precisión es elevada frente a ataques conocidos, su dependencia de reglas predefinidas implica una incapacidad inherente para detectar amenazas nuevas o variantes no contempladas previamente.

Además, estos sistemas requieren un mantenimiento constante para actualizar las bases de reglas, lo que incrementa la carga operativa en entornos reales. En redes de gran tamaño, el número de reglas puede crecer de forma considerable, dificultando su gestión y aumentando la probabilidad de generar falsas alarmas o alertas redundantes. Este fenómeno complica significativamente la labor de los analistas de seguridad, que deben filtrar grandes volúmenes de información para identificar eventos realmente relevantes.

Estas limitaciones justifican la necesidad de complementar los enfoques tradicionales con técnicas más avanzadas basadas en análisis de comportamiento y aprendizaje automático, capaces de identificar patrones no explícitamente definidos mediante reglas.

4.1.3 APLICACIÓN DE MACHINE LEARNING A CIBERSEGURIDAD

Las técnicas de aprendizaje automático han adquirido una gran relevancia en el ámbito de la detección de intrusiones debido a su capacidad para identificar patrones complejos dentro de grandes volúmenes de datos. A diferencia de los sistemas basados exclusivamente en reglas, los modelos de Machine Learning no dependen de firmas predefinidas, sino que aprenden directamente del comportamiento observado en los datos, lo que les permite detectar anomalías o comportamientos sospechosos incluso en ausencia de conocimiento previo sobre el ataque.

En este proyecto se implementa un modelo de clasificación supervisada basado en Random Forest, entrenado con datos reales procedentes del dataset CIC-IDS-2017, con el objetivo de evaluar su capacidad para diferenciar entre tráfico benigno y tráfico malicioso. Esta aproximación permite analizar hasta qué punto los modelos de aprendizaje automático pueden complementar a los sistemas IDS tradicionales en la detección de intrusiones.

4.1.4 IMPORTANCIA DE LA VISUALIZACIÓN EN TIEMPO REAL

En los sistemas de ciberseguridad modernos, la capacidad de visualización en tiempo real de los eventos de red constituye un elemento clave para la correcta interpretación de la información generada por los sistemas de detección. La monitorización visual permite a los analistas identificar de forma rápida patrones de comportamiento, detectar anomalías y reaccionar ante posibles incidentes de seguridad de manera más eficiente.

Por este motivo, el proyecto incorpora una interfaz gráfica desarrollada en Python utilizando Tkinter, diseñada para mostrar de forma clara y estructurada las alertas generadas por Snort. Esta interfaz reproduce, a escala reducida, el funcionamiento de los paneles de monitorización utilizados en centros de operaciones de seguridad (SOC), donde la presentación visual de los eventos resulta fundamental para la toma de decisiones.

4.1.5 VIABILIDAD TÉCNICA DEL PROYECTO

Desde el punto de vista técnico, el proyecto presenta una elevada viabilidad debido al uso exclusivo de herramientas de código abierto ampliamente consolidadas en el ámbito de la ciberseguridad y la investigación en redes. Tecnologías como Mininet, Open vSwitch, Snort, Python y Scikit-learn permiten construir una plataforma completa sin necesidad de infraestructura física adicional ni licencias comerciales, lo que reduce considerablemente la complejidad de implementación.

Además, la arquitectura diseñada es completamente reproducible y modular, lo que facilita su ampliación futura mediante la incorporación de nuevos ataques, algoritmos de aprendizaje automático o mecanismos adicionales de monitorización. Esta característica convierte al sistema en una base sólida para trabajos posteriores o extensiones del proyecto.

4.1.6 VALOR ACADÉMICO Y FORMATIVO

El sistema desarrollado posee un importante valor académico y formativo, ya que integra conocimientos pertenecientes a múltiples áreas de la ingeniería y la ciberseguridad. En particular, el proyecto permite aplicar conceptos relacionados con redes de comunicaciones, seguridad informática, virtualización de redes, programación, análisis de datos y aprendizaje automático dentro de un mismo entorno práctico.

Esta integración multidisciplinar convierte la plataforma en una herramienta especialmente útil tanto para el aprendizaje como para la investigación, ya que permite comprender de forma global el funcionamiento de los sistemas de detección de intrusiones y experimentar con ellos en un entorno controlado. De este modo, el proyecto no solo cumple una función técnica, sino también educativa, al facilitar la comprensión de conceptos complejos mediante su implementación práctica.

4.2 OBJETIVOS

El proyecto tiene como finalidad la construcción de un sistema funcional y visualmente intuitivo que simule el comportamiento de una red real cuando es sometida a diferentes condiciones de tráfico, con especial énfasis en la detección de patrones anómalos. Para lograr este propósito general, se establecen una serie de objetivos específicos que guían las distintas fases del desarrollo.

4.2.1 OBJETIVOS ESPECÍFICOS

1. **Diseñar una topología de red simulada funcional y escalable.** Crear un entorno de red virtualizada que represente de manera realista los elementos fundamentales de una infraestructura real de comunicaciones (hosts y switch virtual). Este entorno funciona con tráfico normal y actúa como base para poder introducir patrones anómalos de tráfico en ella y así experimentar cómo responde la red ante estos sucesos.

2. **Implementar un sistema de detección de tráfico anómalo basado en IDS.** Integrar una herramienta de detección de intrusiones (como Snort o Suricata) configurada para identificar patrones de tráfico sospechoso, como son el escaneo de puertos, ataques de denegación de servicio o tráfico DNS inusual. Se definirán reglas personalizadas para adaptar el comportamiento del sistema a los objetivos específicos del proyecto.

3. **Desarrollar un sistema de visualización e interacción en tiempo real.** Implementar un panel de control que represente de forma clara y visual los eventos detectados. Este sistema deberá incluir la generación de SMS y/o correos electrónicos que reflejen la presencia y gravedad del incidente.

4. **Simular escenarios de ataque y validar la eficacia del sistema.** Generar distintos escenarios controlados que permitan poner a prueba el sistema base, analizando su capacidad para detectar, clasificar y visualizar los eventos de

tráfico anómalo. Se realizarán pruebas de rendimiento en las que se medirán los tiempos de detección, tasas de falsos positivos y capacidad de respuesta visual.

5. **Documentar el proceso.**
   Elaborar la memoria técnica que detalle la arquitectura, configuración y funcionamiento del sistema, junto con una evaluación de su aplicabilidad como herramienta educativa.

4.2.2 OBJETIVO GENERAL

El objetivo general de este proyecto se corresponde con el diseño y desarrollo de un sistema visual de alerta temprana que, mediante la simulación de una red virtual y el análisis de patrones simples de tráfico, permita detectar y representar visualmente comportamientos anómalos en tiempo real, contribuyendo a la comprensión didáctica de los mecanismos de detección y monitorización de seguridad en redes.

4.3 METODOLOGÍA

El desarrollo del sistema se estructura en distintas fases que abarcan desde la planificación inicial hasta la validación del prototipo final. Este proyecto requiere la combinación de principios de ingeniería de software, experimentación en redes virtualizadas y prototipado iterativo, con el fin de garantizar la funcionalidad, estabilidad y valor didáctico del resultado final.

4.3.1 FASE DE ANÁLISIS Y DISEÑO CONCEPTUAL

En esta primera etapa queda definido cuál es el alcance del sistema y los requisitos funcionales y técnicos. Se diseñará la topología de red que servirá de base para la simulación, especificando el número de nodos, los tipos de dispositivos (hosts y switch virtual) y los servicios que estarán activos (DNS, HTTP, etc.).
Se establecerán también los escenarios de tráfico normal y anómalo, determinando qué comportamientos serán clasificados como patrones sospechosos. Asimismo, se evaluarán distintas herramientas de detección, comparando la funcionalidad de Snort y Suricata en

función de su facilidad de integración con el entorno de simulación y su capacidad de conseguir los objetivos de esta etapa.

4.3.2 FASE DE IMPLEMENTACIÓN DEL ENTORNO DE RED

Esta fase consistirá en la creación de una infraestructura de red virtual utilizando la herramienta Mininet, que permitirá emular el comportamiento de una red real dentro de un único equipo físico. Se configurarán distintos nodos con direcciones IP y roles. Posteriormente, se incorporará el sistema de detección de intrusiones (IDS) en el flujo de tráfico para analizar los paquetes en tiempo real. Las alertas generadas se enviarán a un servidor de registro o a un archivo de logs que servirá como fuente de datos para la visualización.

4.3.3 FASE DE DESARROLLO DEL SISTEMA DE VISUALIZACIÓN

El objetivo de esta fase es transformar los datos de detección en información visual comprensible. Se desarrollará una interfaz gráfica preferiblemente en Python utilizando librerías como Tkinter, Dash o Plotly, que muestre en tiempo real los eventos registrados. El panel podrá incluir indicadores de estado, gráficos dinámicos de tráfico, así como notificaciones emergentes como SMS o correos electrónicos. Se buscará que la interfaz sea intuitiva, pedagógica y fácilmente extensible, de modo que futuros usuarios tengan la posibilidad de modificarla o añadir nuevas funciones.

4.3.4 FASE DE VALIDACIÓN Y PRUEBAS

Una vez implementado el sistema, se realizarán pruebas con distintos escenarios de ataque simulados, tales como:

- **Escaneo de puertos (Nmap)** para comprobar la detección de actividad exploratoria.
- **Ataques de denegación de servicio (DoS o Flooding)** para evaluar la capacidad de respuesta del IDS.
- **Tráfico DNS anómalo** para verificar el reconocimiento de peticiones inusuales.
- **Conexiones persistentes no autorizadas** que simulen comportamientos de intrusión o exfiltración de datos.

Durante esta fase se analizarán métricas como la tasa de detección, el tiempo de reacción y la estabilidad del sistema. Los resultados se documentarán y se compararán con los objetivos iniciales, evaluando la eficacia del sistema como herramienta de detección.

4.3.5 FASE DE DOCUMENTACIÓN Y PRESENTACIÓN

Finalmente, se elaborará la documentación técnica del proyecto. Esta incluirá el análisis del problema, el desarrollo del proyecto, una explicación detallada de las configuraciones utilizadas y un análisis de los resultados y conclusiones a las que se han llegado tras observar la respuesta del sistema a los distintos escenarios de tráfico anómalo. Además, incluirá los scripts utilizados al igual que las capturas de la interfaz.

4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

La planificación de las actividades se muestra a continuación:

Fase Actividad principal	**Duración estimada**	**Resultados esperados**
1. Análisis y diseño conceptual	Tres semanas	Definición de requisitos, topología y escenarios de tráfico.
2. Implementación del entorno de red	Tres semanas	Red virtual operativa en Mininet con IDS integrado.
3. Desarrollo del sistema de visualización	Tres semanas	Panel funcional con gráficos y envío de SMS y/o correos.
4. Validación y pruebas	Dos semanas	Resultados de rendimiento y efectividad de detección.
5. Documentación y presentación	Dos semanas	Memoria final.

Tabla 1. Cronograma del trabajo

La elaboración del proyecto depende en gran medida de la correcta selección y disponibilidad de los recursos técnicos necesarios para la implementación, validación y demostración del sistema propuesto. En esta sección se describen y justifican los recursos software y hardware empleados, así como los recursos institucionales y de apoyo implicados en el desarrollo del proyecto.

4.4.1 RECURSOS SOFTWARE

El proyecto se apoya principalmente en herramientas de código abierto y libre distribución, lo que permite por un lado minimizar los costes de implementación y por otro facilitar la reproducibilidad del sistema en distintos entornos académicos.

4.4.1.1 ENTORNO DE SIMULACIÓN DE RED

- **Mininet**

  Es una herramienta de creación y emulación de redes ampliamente utilizada en entornos de investigación y docencia. Permite crear topologías virtuales que simulan el comportamiento de redes físicas con múltiples nodos, conmutadores y controladores de forma instantánea en un solo equipo.
  La elección de Mininet se debe a la facilidad que resulta su uso, el bajo consumo de recursos y capacidad para ejecutar software de red real dentro de los nodos simulados. Gracias a su compatibilidad con Linux, permite integrar de manera directa herramientas de análisis de tráfico y sistemas IDS/IPS.

4.4.1.2 SISTEMA DE DETECCIÓN DE INTRUSIONES (IDS)

- **Snort o Suricata**

  Ambas herramientas son opciones viables y complementarias. Snort destaca por su estabilidad a largo plazo, pero requiere más recursos y en sus primeras versiones utilizaba una arquitectura monohilo, mientras que Suricata ofrece soporte multihilo y mayor rendimiento en procesadores multinúcleo donde existe un alto tráfico. Se seleccionará una de ellas en función de los resultados de las pruebas preliminares

y de las necesidades específicas como son la integración, rendimiento y funciones avanzadas. En cualquier caso, el IDS se configurará para generar alertas basadas en patrones específicos, que posteriormente serán procesadas por el sistema visual.

4.4.1.3 HERRAMIENTAS DE APOYO AL ANÁLISIS

- **Wireshark:** permite la captura, visualización y análisis detallado del tráfico de red tanto en tiempo real como desde un archivo de captura. Esta herramienta es útil para el análisis de paquetes de datos con un alto nivel de detalle y para crear filtros personalizados del sistema y supervisar el comportamiento de las sesiones de red.
- **Nmap:** se creó como herramienta para mapear y escanear redes de gran tamaño, pero funciona también para hosts individuales. Permitiendo descubrir hosts activos y puertos y servicios abiertos.
- **Hping3 y nping:** permite el envío de paquetes TCP/IP personalizados por parte del usuario y analizar en detalle las respuestas de la red. Estas herramientas permiten simular diferentes tipos de ataques, como inundaciones UDP, para poner a prueba las defensas de la red. Es posible probar reglas de firewall, realizar escaneos de puertos o para suplantar paquetes.

4.4.1.4 DESARROLLO DEL SISTEMA DE VISUALIZACIÓN

- **Python** será el lenguaje principal de desarrollo por su versatilidad y abundancia de librerías de análisis y visualización.
    - **Tkinter**: permitirá crear una interfaz gráfica de usuario (GUI) sencilla y nativa.
    - **Dash o Plotly**: posibilitarán la creación de visualizaciones de datos. La librería Plotly presenta una funcionalidad de generación de gráficos dinámicos. Mientras que la librería Dash se utiliza para construir paneles web interactivos que utilizan los gráficos de Plotly.

- **Matplotlib / Pandas**: se usarán para el tratamiento y representación de los datos generados por el IDS.

Todas estas herramientas ofrecen una combinación de robustez, compatibilidad multiplataforma y potencial educativo, lo que las convierte en opciones idóneas para un proyecto orientado al aprendizaje práctico.

4.4.2 RECURSOS HARDWARE

El sistema se desarrollará sobre entornos virtualizados por lo que los requisitos de hardware son moderados, pero deben asegurar un rendimiento estable durante la simulación y el análisis en tiempo real.

- **Equipo principal de desarrollo:**
  - Procesador multinúcleo (mínimo 4 núcleos, preferiblemente 8).
  - Memoria RAM: mínimo 16 GB.
  - Almacenamiento: SSD ≥ 512 GB.
  - Sistema operativo: Ubuntu 22.04 LTS (Linux).
- **Infraestructura de red local:**
  Se requerirá una conexión Ethernet o Wi-Fi estable para permitir la comunicación entre nodos virtuales o con otros dispositivos físicos en caso de ampliaciones futuras del sistema.

4.4.3 RECURSOS INSTITUCIONALES Y DE APOYO

El proyecto se desarrollará con el soporte de los recursos proporcionados por la universidad, tales como:

- Laboratorios de informática con capacidad de virtualización.
- Licencias institucionales de software (cuando sean necesarias).

Capítulo 5. SISTEMA/MODELO DESARROLLADO

5.1 ANÁLISIS DEL SISTEMA

El sistema desarrollado en este Trabajo Fin de Grado se ha diseñado con el propósito de construir una plataforma funcional de detección temprana de amenazas en redes de comunicaciones, capaz de reproducir de forma simplificada el funcionamiento de sistemas de monitorización utilizados en entornos reales de ciberseguridad. La idea principal no es únicamente detectar ataques de forma aislada, sino construir una arquitectura completa en la que convivan la generación de tráfico, la captura de paquetes, la detección basada en reglas y el análisis posterior mediante técnicas de aprendizaje automático.

Desde el inicio del diseño se buscó que el sistema fuese modular, ya que esto permite separar claramente las responsabilidades de cada componente y facilita tanto la depuración como la ampliación futura. Por este motivo, la arquitectura final se divide en dos grandes bloques funcionales. Por un lado, se encuentra el entorno de simulación y detección basado en Snort, que actúa como sistema de detección de intrusos basado en firmas. Por otro lado, se encuentra el módulo de análisis inteligente del tráfico, que utiliza técnicas de Machine Learning para complementar la detección tradicional.

El entorno de simulación se construye utilizando Mininet, una herramienta que permite emular redes completas dentro de un sistema Linux mediante virtualización ligera. Esta elección resulta especialmente adecuada porque no requiere hardware adicional y permite reproducir escenarios complejos de red en un entorno controlado. La topología utilizada en este proyecto es intencionadamente sencilla, ya que el objetivo no es la complejidad de la red en sí, sino la validación del sistema de detección. Por ello, se utilizan tres hosts virtuales conectados a un switch Open vSwitch. Los hosts h1 y h2 funcionan como emisores de tráfico, mientras que el host h3 actúa como sensor encargado de analizar todo el tráfico replicado.

Uno de los aspectos clave del sistema es la utilización de port mirroring, una técnica que permite duplicar el tráfico que circula entre dos puntos de la red hacia un tercer dispositivo. En este caso, el switch Open vSwitch se configura para replicar todo el tráfico entre *h1* y *h2* hacia *h3*. Esto permite que el sistema de detección trabaje de forma pasiva, es decir, sin interferir en la comunicación original. Esta característica es fundamental, ya que reproduce el comportamiento de muchos entornos reales donde los sistemas IDS operan fuera de la ruta directa del tráfico para evitar afectar al rendimiento de la red.

Sobre este entorno se despliega Snort, que actúa como motor principal de detección de intrusos. Snort analiza el tráfico capturado en tiempo real y lo compara con un conjunto de reglas previamente definidas. Cuando un paquete coincide con alguna de estas reglas, se genera una alerta que posteriormente es registrada en un archivo de logs. Estas alertas constituyen la base del análisis posterior realizado por el sistema de monitorización desarrollado en Python.

Finalmente, el sistema incorpora un módulo de Machine Learning basado en Random Forest, entrenado con datos del dataset CIC-IDS-2017. Este módulo no sustituye a Snort, sino que lo complementa, ya que su objetivo es detectar patrones más complejos que no pueden ser fácilmente expresados mediante reglas estáticas.

5.2 DISEÑO DEL SISTEMA

5.2.1 DISEÑO DE LA ARQUITECTURA DE RED

El diseño de la arquitectura de red se realizó con una idea clara: mantener la simplicidad estructural, pero garantizar que el flujo de datos reprodujera de forma realista el comportamiento de una red monitorizada. La red se compone de tres elementos principales: dos hosts emisores, un switch virtual y un host sensor.

Los hosts *h1* y *h2* representan nodos convencionales dentro de una red local. Entre ellos se genera el tráfico que posteriormente será analizado. Este tráfico puede ser benigno o

malicioso dependiendo de las pruebas realizadas, lo que permite simular diferentes escenarios de ataque sin necesidad de infraestructura externa.

El switch Open vSwitch actúa como núcleo de interconexión. A diferencia de un switch tradicional físico, este componente es completamente software y permite una configuración mucho más flexible. Gracias a su capacidad de programación mediante comandos *ovs-vsctl*, es posible modificar el comportamiento del tráfico en tiempo real, lo cual resulta esencial para la implementación del port mirroring.

El host *h3* funciona exclusivamente como sensor IDS. Este nodo no participa activamente en las comunicaciones entre *h1* y *h2*, sino que recibe una copia del tráfico. Para poder analizar correctamente los paquetes, la interfaz del sensor se configura en modo promiscuo, lo que permite capturar todo el tráfico que llega independientemente de su dirección MAC.

El funcionamiento global de esta arquitectura puede entenderse como un flujo continuo donde los datos se generan en los hosts emisores, son transportados a través del switch, duplicados mediante port mirroring y finalmente analizados en el nodo sensor. Este diseño reproduce de manera bastante fiel la estructura de muchas redes reales en las que la monitorización se realiza de forma externa a la comunicación principal.

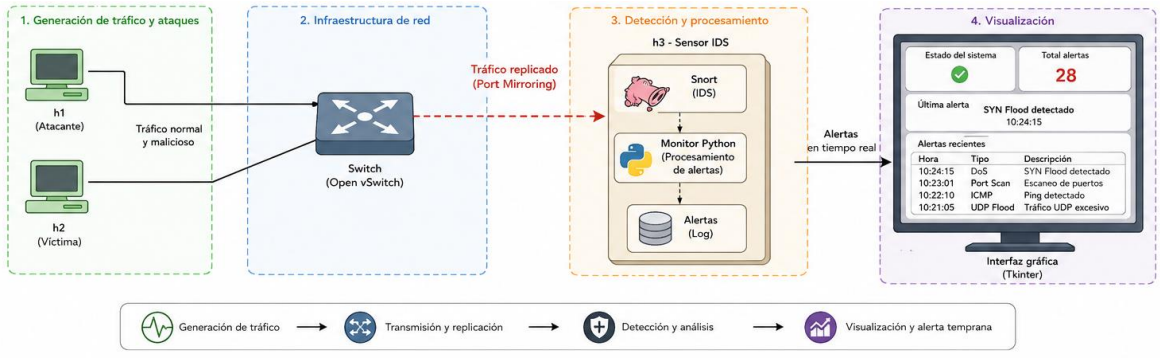

Figura 1. Esquema de la arquitectura general del sistema

La Figura 1 muestra el flujo completo del sistema, desde la generación de tráfico (normal y malicioso) en los hosts, pasando por la infraestructura de red basada en un switch Open vSwitch con duplicación de tráfico mediante port mirroring, hasta el módulo de detección y

procesamiento con Snort y un sistema de monitorización en Python. Finalmente, las alertas se envían a una interfaz gráfica en tiempo real.

5.2.2 DISEÑO DEL SISTEMA DE MONITORIZACIÓN

El sistema de monitorización se diseñó con el objetivo de transformar las alertas generadas por Snort en información comprensible y útil en tiempo real. En lugar de trabajar directamente con los logs de forma manual, se implementa un proceso automatizado que actúa como un observador continuo del archivo de alertas.

El principio de funcionamiento es relativamente sencillo, pero eficaz. El sistema mantiene abierto el archivo de logs generado por Snort y realiza una lectura incremental, lo que significa que solo procesa las nuevas líneas añadidas y no vuelve a leer eventos antiguos. Esto permite que el sistema funcione de manera eficiente incluso cuando el volumen de alertas es elevado.

Cada vez que Snort genera una nueva alerta, esta es escrita en el archivo */var/log/snort/alert*. El monitor detecta esta nueva entrada, extrae la información relevante y la procesa inmediatamente. A partir de ahí, el sistema añade una marca temporal propia, lo que permite contextualizar el evento dentro del flujo general de actividad del sistema.

Este enfoque es muy similar al utilizado en sistemas SIEM reales, donde los eventos de seguridad se recopilan en tiempo real desde múltiples fuentes y se presentan de forma centralizada al analista. Aunque en este proyecto la implementación es más sencilla, el principio operativo es el mismo: transformar datos brutos en información accionable.

5.2.3 DISEÑO DEL SISTEMA DE MACHINE LEARNING

El módulo de Machine Learning se diseñó como un componente complementario al sistema basado en reglas. Mientras que Snort detecta ataques conocidos mediante patrones predefinidos, el modelo de aprendizaje automático intenta identificar comportamientos anómalos basándose en los datos.

Para ello se utilizó el dataset CIC-IDS-2017, que contiene tráfico de red etiquetado con distintos tipos de ataques. En este proyecto se seleccionó un subconjunto reducido compuesto por tráfico benigno y ataques de escaneo de puertos, ya que este tipo de ataque representa una fase inicial muy común en la cadena de ataque.

El diseño del pipeline de Machine Learning incluye varias fases consecutivas. En primer lugar, se realiza una limpieza de los datos para eliminar columnas que no aportan información relevante, como identificadores únicos o direcciones IP, ya que estos podrían sesgar el modelo. Posteriormente se tratan los valores nulos e infinitos, que son habituales en este tipo de datasets debido a errores de captura o cálculos numéricos.

A continuación, se lleva a cabo la selección de características, con el objetivo de reducir la dimensionalidad del problema y mejorar la capacidad de generalización del modelo. Después se aplica un escalado robusto de los datos, lo que permite reducir el impacto de valores extremos típicos del tráfico de red malicioso.

Finalmente se entrena un modelo Random Forest, que ha sido elegido por su capacidad para manejar datos tabulares complejos y su resistencia al sobreajuste. El objetivo del modelo no es únicamente alcanzar una alta precisión, sino mantener un equilibrio razonable entre detección de ataques y número de falsas alarmas.

5.3 IMPLEMENTACIÓN

5.3.1 IMPLEMENTACIÓN DEL ENTORNO DE RED VIRTUAL

La implementación del entorno de red virtual se realizó utilizando Mininet sobre un sistema operativo Linux Ubuntu. Esta herramienta permite crear redes completas mediante la virtualización de hosts, switches y enlaces dentro de un mismo sistema, lo que facilita enormemente el proceso de pruebas.

La topología implementada se construye automáticamente mediante el script red.py, que actúa como punto de entrada del sistema. Este script se encarga de crear los hosts, definir sus direcciones IP, establecer los enlaces entre ellos y configurar el switch Open vSwitch.

Durante la ejecución del script, también se inicializa el mecanismo de port mirroring, que permite replicar el tráfico entre *h1* y *h2* hacia el sensor *h3*. Esta configuración es fundamental para el funcionamiento del IDS, ya que sin ella el host sensor no tendría visibilidad del tráfico de la red.

Además, se activa el modo promiscuo en la interfaz del sensor. Este detalle es importante porque garantiza que el sistema operativo no filtre los paquetes en función de la dirección MAC, permitiendo así que Snort pueda analizar todo el tráfico recibido.

5.3.2 IMPLEMENTACIÓN DEL SISTEMA IDS (SNORT)

El sistema de detección de intrusos se implementó utilizando Snort, que actúa como motor principal de análisis del tráfico de red. Snort trabaja comparando cada paquete capturado con un conjunto de reglas definidas por el usuario o incluidas por defecto.

En este proyecto se desarrollaron varias reglas personalizadas adaptadas al entorno de pruebas. Estas reglas permiten detectar distintos tipos de ataques simulados, desde tráfico ICMP hasta ataques de denegación de servicio o escaneos de puertos.

Cuando Snort detecta un paquete que coincide con alguna regla, genera automáticamente una alerta que se almacena en un archivo de log. Este archivo es posteriormente leído por el sistema de monitorización en tiempo real.

A continuación, se explican en detalle las reglas implementadas.

La primera regla corresponde a la detección de tráfico ICMP. ICMP es el protocolo utilizado por herramientas como ping para comprobar la conectividad entre dos hosts. La regla utilizada es:

alert icmp any any -> any any (msg:"[TFG-ICMP] ICMP detectado"; sid:10000001; rev:1;)

Esta regla es muy sencilla, ya que simplemente detecta cualquier paquete ICMP independientemente de su origen o destino. Su función principal no es identificar un ataque en sí, sino verificar que el sistema IDS es capaz de capturar tráfico básico de red.

La segunda regla está orientada a detectar ataques SYN flood, que son un tipo de denegación de servicio que explota el proceso de establecimiento de conexiones TCP. La regla es:

alert tcp any any -> any any (flags:S; msg:"[TFG-DDOS] SYN Flood detectado"; threshold:type threshold, track by_dst, count 5, seconds 2; sid:10000002; rev:1;)

En este caso, la regla no se limita a detectar un único paquete, sino un patrón de comportamiento. El flag S indica que solo se consideran paquetes con el bit SYN activado, es decir, intentos de inicio de conexión. Además, el sistema establece un umbral que activa la alerta cuando un destino recibe más de 5 paquetes SYN 5 paquetes SYN en un intervalo de 2 segundos. Esto permite identificar situaciones en las que un servidor está siendo inundado con solicitudes de conexión.

La tercera regla detecta escaneos de puertos horizontales, donde un atacante intenta identificar múltiples hosts activos dentro de una red. La regla es:

alert tcp any any -> any any (flags:S; msg:"[TFG-SCAN] Port Scan horizontal"; threshold:type threshold, track by_src, count 10, seconds 10; sid:10000003; rev:1;)

Aquí el análisis se centra en el origen del tráfico. Si una misma dirección IP inicia múltiples conexiones SYN hacia distintos destinos en poco tiempo, se considera un comportamiento sospechoso típico de herramientas como nmap.

La cuarta regla detecta escaneos verticales, donde el atacante se centra en un único host intentando descubrir qué puertos están abiertos. La lógica es similar a la anterior, pero en este caso se analiza el destino del tráfico.

alert tcp any any -> any any (flags:S; msg:"[TFG-SCAN] Port Scan vertical"; threshold:type threshold, track by_dst, count 8, seconds 5; sid:10000004; rev:1;)

Este tipo de ataque suele ser más sigiloso, por lo que el umbral es más bajo, ya que incluso pocas conexiones pueden ser indicativas de un escaneo.

La quinta regla detecta inundaciones UDP, que son ataques de denegación de servicio basados en el envío masivo de datagramas UDP. La regla es:

alert udp any any -> any any (msg:"[TFG-DDOS] UDP Flood detectado"; threshold:type threshold, track by_dst, count 10, seconds 2; sid:10000005; rev:1;)

En este caso no se analiza el contenido del paquete, sino la frecuencia de aparición, ya que UDP no mantiene estado de conexión.

Finalmente, la sexta regla detecta tráfico asociado a HTTP/3 o QUIC, que utiliza UDP en el puerto 443. La regla es:

alert udp any any -> any 443 (msg:"[TFG-HTTP3] Tráfico HTTP/3 (QUIC) detectado"; threshold:type threshold, track by_src, count 2, seconds 10; sid:10000006; rev:1;)

Esta regla no identifica necesariamente un ataque, sino la presencia de un protocolo moderno cifrado que puede ser relevante para análisis de seguridad.

5.3.3 IMPLEMENTACIÓN DEL MONITOR DE ALERTAS

El monitor de alertas se implementó en Python y tiene como objetivo principal actuar como un puente entre el sistema de detección de intrusos (Snort) y el usuario final, permitiendo que la información generada por el IDS pueda ser interpretada en tiempo real sin necesidad de acceder directamente a los archivos de registro. Este componente resulta especialmente importante dentro de la arquitectura global del sistema, ya que Snort genera un flujo continuo de eventos que, sin un mecanismo de procesamiento adicional, sería difícil de analizar de forma eficiente.

La implementación se basa en la lectura continua del archivo de alertas generado por Snort, manteniéndolo abierto durante toda la ejecución del sistema. A diferencia de un procesamiento tradicional en el que se analiza el archivo completo cada cierto tiempo, en

este caso se utiliza un enfoque incremental similar al funcionamiento del comando tail -f en sistemas Linux. Esto permite que el sistema únicamente procese la información nueva que va siendo añadida al fichero, evitando así la sobrecarga asociada a la relectura constante de datos ya procesados.

El funcionamiento interno del monitor se basa en un bucle continuo que permanece a la espera de nuevas entradas en el archivo. Cuando Snort genera una nueva alerta, esta es escrita automáticamente en el log, y el monitor detecta inmediatamente la aparición de una nueva línea. En ese momento, el sistema realiza una lectura de la entrada, extrae la información relevante del evento y la muestra en pantalla o la envía directamente al módulo de visualización gráfica.

Este diseño permite que el sistema funcione prácticamente en tiempo real, con una latencia muy reducida entre la detección del evento y su visualización. Además, el consumo de recursos es bajo, ya que el sistema evita operaciones de lectura masiva o procesamiento innecesario del archivo completo. Este aspecto resulta especialmente importante en entornos donde el volumen de eventos puede aumentar considerablemente.

En conjunto, este módulo reproduce el comportamiento básico de los sistemas de monitorización utilizados en entornos profesionales, donde los eventos de seguridad deben ser capturados y procesados de forma continua para permitir una respuesta rápida ante posibles incidentes.

5.3.4 IMPLEMENTACIÓN DE LA INTERFAZ GRÁFICA

La interfaz gráfica del sistema se desarrolló utilizando la biblioteca Tkinter de Python, seleccionada por su simplicidad, su integración nativa con el lenguaje y su capacidad suficiente para construir paneles de monitorización básicos pero funcionales. El objetivo principal de esta interfaz no es ofrecer una visualización compleja o avanzada, sino proporcionar una representación clara e intuitiva del estado del sistema IDS y de los eventos de seguridad detectados.

La interfaz actúa como la capa final de presentación del sistema, permitiendo al usuario observar en tiempo real la actividad generada por Snort a través del monitor de alertas. A pesar de su aparente sencillez, su papel dentro de la arquitectura es fundamental, ya que transforma datos en bruto en información comprensible, facilitando la interpretación del comportamiento de la red.

El panel muestra información esencial como el número total de alertas generadas desde el inicio de la ejecución del sistema, el tipo de ataque detectado en cada evento, la última alerta registrada y un historial cronológico de eventos de seguridad. Esta estructura permite al usuario tener una visión global del estado de la red sin necesidad de consultar directamente los logs del sistema o analizar manualmente la salida de Snort.

Uno de los aspectos más relevantes de la implementación es la actualización dinámica de la interfaz. A medida que el monitor de alertas detecta nuevos eventos, estos son enviados a la interfaz gráfica, que se actualiza automáticamente sin necesidad de intervención del usuario. Este comportamiento simula parcialmente el funcionamiento de los paneles utilizados en centros de operaciones de seguridad (SOC), donde la información debe actualizarse constantemente en tiempo real.

Además, la interfaz incorpora un sistema básico de clasificación de eventos, lo que permite diferenciar entre distintos tipos de ataques o actividades sospechosas. Aunque esta clasificación es sencilla, resulta útil para priorizar la atención sobre ciertos eventos frente a otros, especialmente en escenarios donde el volumen de alertas puede aumentar.

En conjunto, esta interfaz cumple una función de visualización esencial dentro del sistema, proporcionando una capa de abstracción que mejora la usabilidad y facilita la interpretación de los resultados obtenidos por el IDS.

5.3.5 IMPLEMENTACIÓN DEL SISTEMA MACHINE LEARNING

El módulo de Machine Learning se implementó utilizando Python junto con la biblioteca Scikit-learn, una de las herramientas más utilizadas en el ámbito del aprendizaje automático aplicado a problemas de clasificación. El objetivo principal de este módulo es complementar

el sistema basado en firmas mediante un enfoque adicional basado en comportamiento, permitiendo analizar patrones de tráfico desde una perspectiva estadística.

El modelo se entrenó utilizando el dataset CIC-IDS-2017, ampliamente reconocido en el ámbito de la ciberseguridad por contener tráfico de red real etiquetado con distintos tipos de ataques y tráfico benigno. Para este proyecto se seleccionó un subconjunto específico del dataset con el objetivo de simplificar el problema a una clasificación binaria entre tráfico normal y tráfico malicioso, centrándose especialmente en escenarios de port scanning.

Antes del entrenamiento del modelo se llevó a cabo un proceso exhaustivo de preprocesamiento de los datos, ya que el dataset original contiene una gran cantidad de variables heterogéneas, valores nulos e incluso valores infinitos derivados del procesamiento del tráfico de red. Durante esta fase se eliminaron aquellas columnas que no aportaban información relevante o que podían introducir sesgos en el modelo, como identificadores únicos o campos relacionados con direcciones IP.

Posteriormente se realizó el tratamiento de valores nulos e infinitos, sustituyéndolos por valores estadísticos representativos o eliminándolos en función de su impacto. También se aplicaron técnicas de selección de características con el objetivo de reducir la dimensionalidad del problema y conservar únicamente aquellas variables con mayor capacidad discriminativa. Finalmente, se aplicó un escalado robusto para normalizar las variables y evitar que los valores extremos propios del tráfico de red afectaran negativamente al proceso de aprendizaje.

El modelo seleccionado fue Random Forest debido a su capacidad para manejar conjuntos de datos complejos, su resistencia al sobreajuste y su buen rendimiento en problemas de clasificación tabular. Durante el entrenamiento se ajustaron distintos parámetros, como el número de árboles o el peso asignado a cada clase, con el objetivo de mejorar el equilibrio entre la detección de ataques y la reducción de falsas alarmas.

En términos generales, este módulo permite añadir una capa adicional de análisis al sistema, aportando una perspectiva complementaria a la detección basada en reglas.

5.3.6 INTEGRACIÓN FINAL DEL SISTEMA

La integración final del sistema representa la fase en la que todos los componentes desarrollados se unen para formar una plataforma completa de detección de intrusos. En este punto, cada módulo cumple una función específica dentro de un flujo operativo continuo que reproduce de manera simplificada el funcionamiento de sistemas de seguridad reales.

El proceso comienza con la generación de tráfico dentro de la red virtual implementada con Mininet, donde los hosts simulan tanto tráfico legítimo como tráfico malicioso. Este tráfico es capturado de forma pasiva por el switch Open vSwitch, que lo replica hacia el host sensor mediante la técnica de port mirroring. A partir de este momento, Snort analiza los paquetes recibidos y los compara con las reglas definidas en su configuración.

Cuando se detecta un patrón coincidente con alguna regla de seguridad, Snort genera automáticamente una alerta que es almacenada en su archivo de registro. A continuación, el monitor de alertas entra en acción, detectando la aparición de nuevos eventos en el log y procesándolos de forma incremental. Estos eventos son enviados posteriormente a la interfaz gráfica, que los muestra en tiempo real al usuario.

En paralelo a este flujo principal, el módulo de Machine Learning actúa como una capa adicional de análisis. Aunque no interviene directamente en la generación de alertas de Snort, permite estudiar el tráfico desde una perspectiva más global, identificando patrones que pueden no estar contemplados en las reglas estáticas del sistema. Esta dualidad entre detección basada en firmas y análisis basado en aprendizaje automático refuerza la capacidad general del sistema para identificar comportamientos anómalos.

En conjunto, la integración de todos estos componentes da lugar a una plataforma funcional, coherente y reproducible que combina distintas técnicas de análisis de tráfico dentro de un único entorno experimental de ciberseguridad.

Capítulo 6. ANÁLISIS DE RESULTADOS

6.1 INTRODUCCIÓN

En este capítulo se presentan de forma detallada los resultados obtenidos tras la implementación y evaluación del sistema de detección de intrusos desarrollado a lo largo del proyecto. Esta fase representa uno de los puntos más importantes del trabajo, ya que es donde se comprueba si todo el diseño previo, tanto a nivel de arquitectura de red como de lógica de detección, cumple realmente con los objetivos planteados inicialmente.

El propósito principal del análisis no es únicamente comprobar si el sistema "funciona", sino entender hasta qué punto es capaz de identificar comportamientos maliciosos en un entorno de red simulado, cómo responde ante distintos tipos de ataques y qué limitaciones aparecen cuando se somete a condiciones más realistas o exigentes. En este sentido, el análisis de resultados se plantea como una evaluación integral del sistema, teniendo en cuenta tanto la parte de red como la parte de seguridad, la capa de monitorización y el módulo de aprendizaje automático.

A lo largo del capítulo se estudia el comportamiento del sistema desde varias perspectivas complementarias. En primer lugar, se valida el funcionamiento de la infraestructura de red creada con Mininet, ya que sin una base de red estable no tendría sentido evaluar el resto de los componentes. Posteriormente, se analizan los resultados obtenidos por el sistema IDS basado en Snort, observando su capacidad de detección frente a distintos tipos de ataques reales simulados en el entorno virtual. A continuación, se estudia el sistema de monitorización en tiempo real desarrollado en Python, que actúa como capa intermedia entre el motor de detección y el usuario final. Finalmente, se evalúa el módulo de Machine Learning, que introduce una capa adicional de análisis basada en comportamiento y no únicamente en firmas.

Además de la presentación de resultados, este capítulo incluye una interpretación crítica de los mismos. No se trata solo de describir qué ocurre, sino de explicar por qué ocurre, qué implicaciones tiene y qué limitaciones se han detectado durante el proceso. Este enfoque

permite obtener una visión más realista del sistema y de su aplicabilidad en escenarios más cercanos a entornos de producción.

6.2 VALIDACIÓN DEL ENTORNO DE RED

Antes de analizar el comportamiento del sistema de detección, fue necesario asegurar que la infraestructura de red sobre la que se apoya todo el proyecto funcionaba correctamente. Esta fase de validación es fundamental, ya que cualquier error en la configuración de la red virtual puede afectar directamente a la fiabilidad de los resultados posteriores, especialmente en lo relativo a la captura de tráfico y a la generación de alertas.

La red implementada mediante Mininet reproduce un escenario bastante sencillo pero suficiente para los objetivos del proyecto. En este entorno se interconectan varios hosts virtuales a través de un switch basado en Open vSwitch, lo que permite simular una red local real con capacidades de conmutación a nivel 2. Aunque la topología es reducida, su comportamiento es suficientemente representativo como para evaluar mecanismos de detección de intrusos.

La primera verificación realizada consistió en comprobar la conectividad básica entre los distintos nodos. Para ello se utilizaron herramientas estándar del sistema operativo, como el comando ping, que permite enviar paquetes ICMP entre máquinas y medir la respuesta. Los resultados obtenidos mostraron que los hosts podían comunicarse entre sí sin pérdida de paquetes significativa y con tiempos de respuesta coherentes con un entorno local virtualizado. Este comportamiento era esperado, pero aun así resulta importante validarlo, ya que confirma que las interfaces virtuales están correctamente configuradas dentro de sus respectivos espacios de red.

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=3.64 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.058 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.053 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.054 ms
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6091ms
rtt min/avg/max/mdev = 0.049/0.569/3.636/1.251 ms
```

Figura 2. Verificación de conectividad en la red simulada

Una vez verificada la conectividad básica, el siguiente paso consistió en comprobar el funcionamiento del switch virtual y, en particular, de la capacidad de replicación de tráfico mediante port mirroring. Este aspecto es clave dentro de la arquitectura del sistema, ya que el sensor IDS no se encuentra en línea con el tráfico, sino que recibe una copia del mismo. Esto implica que el switch debe duplicar correctamente los paquetes sin alterar el flujo original entre los hosts.

Durante las pruebas se observó que el tráfico generado entre los nodos principales era efectivamente replicado hacia el nodo sensor, lo que confirmaba que la configuración del espejo de puertos estaba funcionando según lo esperado. Este comportamiento permitió validar uno de los principios fundamentales del diseño del sistema, que es la monitorización pasiva del tráfico. Es importante destacar que en este punto no se produjo ninguna interferencia con las comunicaciones originales, lo cual refuerza la validez del enfoque elegido, ya que el sistema de detección no introduce latencia ni afecta al rendimiento de la red simulada.

Adicionalmente, se observó que el nodo sensor era capaz de recibir una copia completa del tráfico incluso en situaciones de mayor carga, lo que indica que la configuración del switch virtual es estable dentro del entorno de pruebas utilizado. Este aspecto es relevante porque en sistemas IDS reales uno de los principales problemas suele ser precisamente la pérdida de paquetes cuando el volumen de tráfico aumenta.

En conjunto, los resultados obtenidos en esta fase permiten afirmar que la infraestructura de red cumple correctamente su función como base del sistema de detección, proporcionando un entorno estable, reproducible y adecuado para la generación y análisis de tráfico.

```
mininet> h3 tcpdump -i any host 10.0.0.1 or host 10.0.0.2
[2026-05-29 10:32:41] ALERTA DETECTADA
05/29-10:32:40.871294  [**] [1:10000001:1] [TFG-ICMP] ICMP detectado [**] [Priority: 0] {IPV6-ICMP} fe80::843c:d9ff:fec6
:359a -> ff02::2
------------------------------------------------------------
[2026-05-29 10:32:41] ALERTA DETECTADA
05/29-10:32:40.871294  [**] [1:1000001:1] TFG-ICMP: Ping detectado [**] [Priority: 0] {IPV6-ICMP} fe80::843c:d9ff:fec6:3
59a -> ff02::2
------------------------------------------------------------
[2026-05-29 10:32:41] ALERTA DETECTADA
05/29-10:32:40.934965  [**] [1:10000001:1] [TFG-ICMP] ICMP detectado [**] [Priority: 0] {IPV6-ICMP} fe80::74af:27ff:fe28
:f39d -> ff02::2
------------------------------------------------------------
[2026-05-29 10:32:41] ALERTA DETECTADA
05/29-10:32:40.934965  [**] [1:1000001:1] TFG-ICMP: Ping detectado [**] [Priority: 0] {IPV6-ICMP} fe80::74af:27ff:fe28:f
39d -> ff02::2
------------------------------------------------------------
[2026-05-29 10:32:41] ALERTA DETECTADA
05/29-10:32:41.128079  [**] [1:10000001:1] [TFG-ICMP] ICMP detectado [**] [Priority: 0] {IPV6-ICMP} fe80::c867:2dff:fea3
:bf39 -> ff02::2
------------------------------------------------------------
[2026-05-29 10:32:41] ALERTA DETECTADA
05/29-10:32:41.128079  [**] [1:1000001:1] TFG-ICMP: Ping detectado [**] [Priority: 0] {IPV6-ICMP} fe80::c867:2dff:fea3:b
f39 -> ff02::2
```

Figura 3. Captura de tráfico en la red simulada.

6.3 RESULTADOS DEL SISTEMA IDS

Una vez validada la infraestructura de red, se procedió a evaluar el funcionamiento del sistema de detección de intrusos basado en Snort. Esta fase constituye el núcleo del análisis de resultados, ya que es donde se comprueba la capacidad real del sistema para identificar comportamientos maliciosos mediante reglas previamente definidas.

El enfoque utilizado en este proyecto se basa en detección por firmas, lo que significa que el sistema identifica patrones concretos en los paquetes de red que coinciden con reglas establecidas manualmente. Este enfoque tiene la ventaja de ser muy preciso cuando el patrón es conocido, aunque presenta limitaciones frente a ataques nuevos o variantes no contempladas.

Durante las pruebas se generaron diferentes tipos de tráfico malicioso controlado desde el nodo atacante hacia el nodo objetivo, con el objetivo de evaluar la respuesta del sistema IDS ante distintos escenarios.

6.3.1 DETECCIÓN DE TRÁFICO ICMP

La primera serie de pruebas se centró en el tráfico ICMP, que es uno de los protocolos más simples dentro del modelo TCP/IP. Este tipo de tráfico se utiliza habitualmente para diagnósticos de red mediante comandos como ping, pero también puede ser empleado para reconocimiento de sistemas activos dentro de una red.

En este caso, se generó tráfico ICMP de forma controlada para simular un proceso de descubrimiento de hosts. El sistema IDS fue capaz de identificar correctamente la presencia de este tipo de tráfico y generar las alertas correspondientes en el sistema de logs.

El comportamiento observado indica que el motor de Snort está capturando correctamente los paquetes ICMP replicados por el switch y aplicando las reglas definidas sin problemas. Aunque el tráfico ICMP no siempre se considera malicioso por sí mismo, su detección es importante dentro del contexto de un sistema de seguridad, ya que puede ser el primer indicio de una fase de reconocimiento.

Estos resultados confirman que tanto la captura de paquetes como el procesamiento de reglas básicas están funcionando de manera correcta dentro del sistema implementado.

6.3.2 DETECCIÓN DE ATAQUES SYN FLOOD

Una de las pruebas más relevantes dentro del análisis consistió en la simulación de un ataque de denegación de servicio mediante inundación de paquetes SYN. Este tipo de ataque explota el mecanismo de establecimiento de conexiones TCP, conocido como three-way handshake.

En condiciones normales, cuando un cliente desea establecer una conexión TCP, envía un paquete SYN al servidor, que responde con un SYN-ACK, quedando a la espera del ACK final para completar la conexión. En un ataque SYN Flood, este proceso se interrumpe deliberadamente en la fase inicial, enviando una gran cantidad de paquetes SYN sin completar el establecimiento de conexión. Esto provoca que el servidor mantenga múltiples

conexiones semiabiertas, consumiendo recursos de memoria y eventualmente saturando su capacidad de respuesta.

Durante las pruebas realizadas, el sistema IDS fue capaz de detectar patrones anómalos en la frecuencia de paquetes SYN recibidos en el nodo objetivo. Cuando el número de paquetes superaba los umbrales definidos en las reglas de Snort, se generaban alertas de forma automática.

El comportamiento observado confirma que el sistema es capaz de identificar no solo la presencia de tráfico TCP, sino también su comportamiento temporal, lo cual es fundamental para detectar este tipo de ataques. Sin embargo, también se observó que la sensibilidad de las reglas juega un papel importante, ya que umbrales demasiado bajos pueden generar un número elevado de falsas alarmas en situaciones de tráfico legítimo elevado.

6.3.3 DETECCIÓN DE ESCANEO DE PUERTOS

Dentro del conjunto de pruebas realizadas, uno de los escenarios más relevantes desde el punto de vista de seguridad fue la detección de actividades de escaneo de puertos. Este tipo de comportamiento no representa en sí mismo un ataque directo, pero sí constituye una de las fases más habituales dentro de cualquier proceso de intrusión. En un entorno real, un atacante raramente intenta explotar una vulnerabilidad sin antes haber realizado un reconocimiento previo de la superficie de ataque, y precisamente el escaneo de puertos es una de las herramientas más utilizadas para este propósito.

Durante las pruebas se simularon dos tipos de comportamiento distintos. Por un lado, se realizaron escaneos horizontales, en los que se intenta identificar múltiples máquinas activas dentro de la red comprobando si responden a un determinado puerto concreto. Por otro lado, se ejecutaron escaneos verticales, en los que el objetivo es un único host, pero se analizan múltiples puertos con el fin de descubrir servicios expuestos.

El sistema IDS basado en Snort mostró un comportamiento adecuado en ambos casos, siendo capaz de identificar patrones repetitivos de conexiones TCP en estado SYN que no se corresponden con el comportamiento habitual de una comunicación legítima. En particular, lo que permitió activar las alertas no fue únicamente la existencia de intentos de conexión, sino la frecuencia y la estructura del tráfico generado en un intervalo corto de tiempo.

En el caso del escaneo horizontal, el sistema detectó correctamente que un mismo origen estaba intentando establecer conexiones hacia múltiples destinos dentro de la red, lo que encaja claramente con un patrón de reconocimiento automatizado. Este tipo de comportamiento es especialmente relevante porque suele ser una fase previa a ataques más dirigidos, en los que el atacante ya dispone de información sobre qué máquinas están activas y qué servicios podrían ser vulnerables.

En el escaneo vertical, en cambio, se observó un patrón diferente, ya que el tráfico se concentraba en un único host, pero con un elevado número de intentos sobre distintos puertos. Este comportamiento también fue identificado correctamente por el sistema, lo que demuestra que las reglas implementadas no dependen únicamente de la dirección de destino, sino del análisis global del comportamiento del flujo de red.

En términos generales, los resultados obtenidos en esta fase permiten confirmar que el sistema es capaz de detectar actividades de reconocimiento de forma efectiva, lo cual es especialmente importante porque este tipo de tráfico suele pasar desapercibido en sistemas de seguridad más básicos. Sin embargo, también se observó que en entornos con tráfico más dinámico o con múltiples usuarios simultáneos, este tipo de reglas podría requerir ajustes finos para evitar interpretaciones erróneas de comportamientos legítimos como potencialmente maliciosos.

6.3.4 DETECCIÓN DE TRÁFICO UDP Y QUIC

Otra parte importante del análisis se centró en la evaluación del sistema frente a tráfico basado en UDP, así como en protocolos más modernos como QUIC, que se utiliza como base para HTTP/3. Este escenario es especialmente interesante porque introduce una complejidad adicional respecto a los protocolos tradicionales, ya que QUIC incorpora cifrado desde las primeras fases de la comunicación y utiliza UDP como transporte, lo que dificulta su inspección profunda mediante técnicas clásicas.

En las pruebas realizadas se generó tráfico UDP de forma intensiva con el objetivo de simular un comportamiento anómalo similar a un ataque de inundación. A diferencia de TCP, UDP no establece conexión previa ni mantiene estado, lo que lo convierte en un protocolo especialmente vulnerable a abusos en forma de envío masivo de paquetes. El

sistema IDS fue capaz de detectar este incremento anómalo de tráfico hacia un mismo destino en un intervalo de tiempo reducido, lo que permitió generar alertas basadas en umbrales de volumen.

Este resultado es relevante porque demuestra que, incluso en ausencia de información de estado a nivel de transporte, el sistema sigue siendo capaz de identificar patrones de comportamiento sospechoso basándose en métricas temporales y volumétricas.

En el caso del tráfico QUIC, la situación es más compleja. Dado que este protocolo cifra la mayor parte de su comunicación desde el inicio, el análisis de contenido deja de ser viable, y la detección debe basarse en características indirectas como el puerto de destino, el uso de UDP y la frecuencia de los paquetes. En este proyecto se optó por una aproximación basada en la identificación del tráfico asociado al puerto 443 sobre UDP, lo cual permite inferir la posible presencia de QUIC.

Los resultados obtenidos muestran que el sistema es capaz de identificar este tipo de tráfico de forma aproximada, aunque no con el mismo nivel de precisión que en protocolos más tradicionales. Esto pone de manifiesto una de las limitaciones inherentes a los sistemas IDS basados en firmas cuando se enfrentan a tecnologías de cifrado modernas, donde la visibilidad del contenido se reduce considerablemente.

Aun así, el sistema mantiene su utilidad al permitir al menos la detección de patrones indirectos asociados a este tipo de tráfico, lo que puede ser suficiente en muchos escenarios de monitorización de red.

6.3.5 SÍNTESIS DE RESULTADOS DEL IDS

Si se analizan en conjunto todas las pruebas realizadas sobre el sistema IDS, se puede observar un patrón bastante consistente. El sistema se comporta de manera fiable en la detección de tráfico claramente estructurado y asociado a ataques conocidos, especialmente aquellos que generan patrones repetitivos o que dependen de la saturación de recursos.

El uso de reglas basadas en Snort demuestra ser eficaz en entornos controlados, donde el comportamiento del tráfico es relativamente predecible y los ataques siguen patrones clásicos. En este sentido, el sistema cumple correctamente su función como mecanismo de detección basado en firmas.

Sin embargo, también se observa que el rendimiento del sistema depende en gran medida de la calidad de las reglas definidas. Reglas demasiado sensibles pueden generar un número elevado de falsas alarmas, mientras que reglas demasiado permisivas pueden dejar pasar comportamientos maliciosos. Este equilibrio entre sensibilidad y precisión es uno de los problemas clásicos en la configuración de sistemas IDS y ha quedado claramente reflejado en las pruebas realizadas.

Los resultados obtenidos se resumen en la Tabla 2, donde se observa la relación entre el tipo de ataque simulado, el comando utilizado para su generación y la respuesta del sistema en términos de generación de alertas. En todos los casos evaluados, el sistema fue capaz de detectar correctamente los patrones de tráfico definidos en las reglas de Snort, generando las alertas correspondientes en el sistema de monitorización.

TIPO DE REGLA	COMANDO LINUX	NÚMERO DE ALERTAS GENERADAS	VENTANA TEMPORAL	UMBRAL DE LA REGLA	PRUEBA DE TRÁFICO BENIGNO
ICMP (PING)	h1 ping h2	1 alerta por paquete ICMP	No aplica	Sin umbral	No
SYN flood	h1 hping3 -sS 10.0.0.2 -p 80 -c 20	1 alerta	2 segundos	5 paquetes SYN hacia el mismo destino	Sí, conexiones TCP normales no generan alerta
Port Scan Horizontal	h1 nmap -sS 10.0.0.2	1 alerta	10 segundos	10 intentos desde el mismo origen	Sí, accesos normales a pocos puertos no generan alerta

Port Scan Vertical	h1 nmap -sS 10.0.0.2	1 alerta	5 segundos	8 intentos hacia el mismo host desde un origen	Sí, acceso a múltiples puertos de forma limitada no genera alerta
UDP	h1 hping3 --udp 10.0.0.2 -c 10	1 alerta	2 segundos	10 paquetes UDP hacia el mismo destino	Sí, tráfico UDP ocasional no genera alerta
HTTP3/QUIC	h1 hping3 --udp -p 443 10.0.0.2 -c 10	1 alerta	10 segundos	2 paquetes UDP dirigidos al puerto 443	No, tráfico QUIC es detectado como sospechoso

Tabla 2. Resultados de detección de ataques mediante Snort

6.4 RESULTADOS DEL SISTEMA DE MONITORIZACIÓN

El sistema de monitorización desarrollado en Python constituye la capa intermedia entre el motor de detección y el usuario final. Su objetivo principal es transformar los eventos generados por Snort, que en su forma original se presentan como registros de texto sin procesar, en información estructurada y comprensible en tiempo real.

Durante las pruebas realizadas, el sistema mostró un funcionamiento estable y continuo, sin interrupciones ni pérdidas de información relevantes. Uno de los aspectos más importantes

que se validó en esta fase fue la capacidad del sistema para leer el archivo de logs de Snort de forma incremental, simulando un comportamiento similar al comando *tail -f* de sistemas Unix. Esto permitió que las alertas fueran procesadas prácticamente en el mismo instante en que se generaban.

La interfaz gráfica desarrollada permitió visualizar de forma clara la evolución del sistema en tiempo real. A medida que se generaban nuevas alertas, estas se incorporaban automáticamente al panel, mostrando información como el tipo de evento detectado, el momento exacto de su ocurrencia y un identificador asociado al tipo de ataque.

Uno de los aspectos más relevantes observados durante las pruebas es que la visualización en tiempo real mejora significativamente la comprensión del comportamiento del sistema. Mientras que los logs tradicionales requieren un análisis posterior y manual, la interfaz permite identificar patrones de ataque de forma inmediata, lo que resulta especialmente útil en escenarios de monitorización continua.

Además, el sistema fue capaz de mantener un rendimiento estable incluso cuando el número de alertas aumentaba en un corto intervalo de tiempo, lo que indica que el diseño basado en colas y procesamiento asíncrono es adecuado para este tipo de aplicaciones.

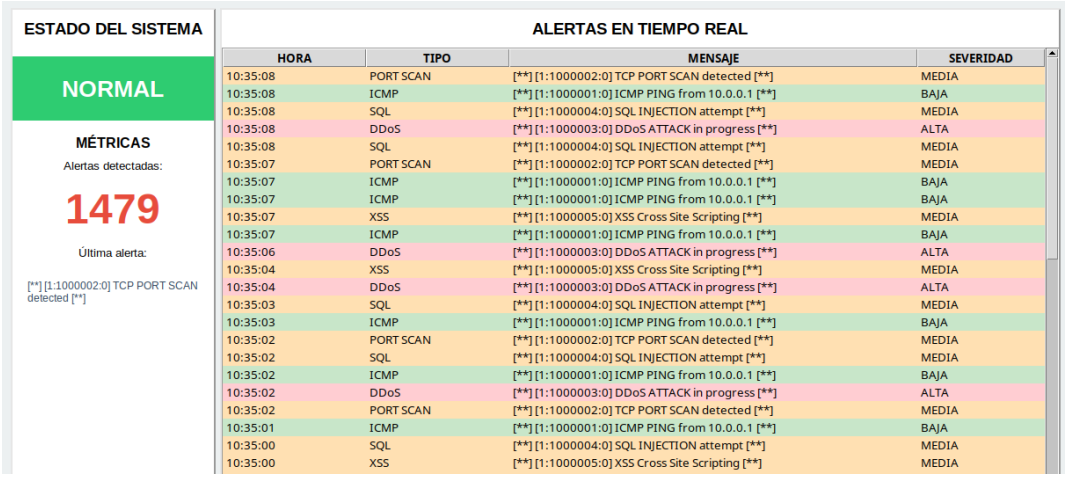

Figura 4. Interfaz gráfica mostrando alertas detectadas en tiempo real.

La Figura 4 muestra el panel principal del sistema, donde se indica el estado general del sistema, el número total de alertas detectadas y el listado de eventos en tiempo real. Cada alerta incluye la hora, el tipo de ataque detectado (por ejemplo, escaneo de puertos, ICMP, SQL injection o DDoS), un mensaje descriptivo y su nivel de severidad.

6.5 RESULTADOS DEL MODELO DE MACHINE LEARNING

El módulo de Machine Learning desarrollado en este proyecto representa una aproximación complementaria al sistema de detección basado en firmas implementado con Snort. Mientras que el IDS tradicional se centra en identificar patrones concretos previamente definidos, el modelo de aprendizaje automático busca capturar comportamientos estadísticos dentro del tráfico de red, intentando generalizar a partir de ejemplos conocidos. Esta diferencia conceptual es importante, ya que condiciona tanto la forma en la que se entrenan los datos como la manera en la que deben interpretarse los resultados obtenidos.

El dataset utilizado para esta parte del sistema, CIC-IDS-2017, es un conjunto de datos ampliamente empleado en investigación en ciberseguridad, ya que incluye tráfico realista con diferentes tipos de ataques y tráfico benigno. Sin embargo, uno de los principales problemas al trabajar con este tipo de datasets es el fuerte desequilibrio entre clases, ya que el tráfico normal suele ser mucho más abundante que el tráfico malicioso. Este desequilibrio puede provocar que un modelo de clasificación aprenda a "acertar" simplemente prediciendo siempre la clase mayoritaria, lo cual daría una falsa sensación de buen rendimiento.

Para evitar este problema, se realizó un proceso de submuestreo controlado, equilibrando el número de muestras de tráfico benigno y de tráfico malicioso. Esta decisión, aunque reduce la cantidad total de datos disponibles para el entrenamiento, permite que el modelo aprenda patrones relevantes de ambas clases de forma más equilibrada. Aun así, este tipo de técnicas siempre implica una cierta pérdida de información, ya que no se está utilizando la totalidad del comportamiento del tráfico benigno original.

6.5.1 INTERPRETACIÓN DEL PREPROCESAMIENTO DEL DATASET

Antes del entrenamiento del modelo, fue necesario realizar un proceso de limpieza y transformación de los datos. Este paso es fundamental en cualquier sistema de Machine Learning aplicado a redes, ya que los datos de tráfico suelen contener valores inconsistentes, campos irrelevantes o características que pueden introducir sesgos no deseados en el modelo. Una de las decisiones más importantes fue la eliminación de columnas identificativas como direcciones IP, identificadores de flujo o marcas temporales. Aunque a primera vista estas variables podrían parecer útiles, en realidad tienden a provocar sobreajuste, ya que el modelo podría aprender a reconocer casos concretos del dataset en lugar de patrones generales de comportamiento. En un entorno real, esto se traduciría en una pérdida de capacidad de generalización.

También se aplicó un tratamiento de valores nulos e infinitos, que aparecen con frecuencia en datos de red debido a errores de cálculo o a características de protocolos específicos. Estos valores fueron sustituidos o eliminados dependiendo de su impacto en la distribución global de los datos. Este proceso, aunque puede parecer puramente técnico, tiene un impacto directo en la estabilidad del modelo final.

Posteriormente se realizó una selección de características basada en varianza y correlación. Este paso permite reducir la dimensionalidad del problema eliminando variables que aportan poca información o que están altamente correlacionadas con otras. La reducción de dimensionalidad no solo mejora el rendimiento computacional, sino que también ayuda a evitar ruido en el proceso de aprendizaje.

Finalmente, se aplicó un escalado robusto de los datos. Este punto es especialmente importante en el contexto de tráfico de red, ya que muchas variables presentan distribuciones con valores extremos debido a ataques o picos de tráfico. El uso de un escalado basado en mediana e intervalo intercuartílico permite reducir la influencia de estos valores extremos, haciendo que el modelo sea más estable.

6.5.2 ANÁLISIS DEL ENTRENAMIENTO DEL MODELO

El modelo seleccionado fue Random Forest, debido a su buen rendimiento en problemas de clasificación y su capacidad para manejar conjuntos de datos con múltiples variables

heterogéneas. El entrenamiento se realizó utilizando tanto tráfico benigno como tráfico asociado a ataques de escaneo, lo que permitió construir un modelo capaz de diferenciar entre comportamiento normal y actividad sospechosa.

Durante el proceso se probaron distintas configuraciones de parámetros, ajustando el peso de las clases con el objetivo de equilibrar la detección de ataques con la reducción de falsas alarmas. Este equilibrio es uno de los principales retos en sistemas de detección basados en aprendizaje automático.

6.5.3 INTERPRETACIÓN DE LOS RESULTADOS OBTENIDOS

Los resultados obtenidos muestran una precisión global aproximada del 68%, junto con una tasa de detección de ataques cercana al 67% y una tasa de falsos positivos en torno al 31%. A primera vista, estos valores pueden parecer moderados en comparación con otros problemas de clasificación más tradicionales, pero en el contexto de tráfico de red deben interpretarse con mayor cautela.

En primer lugar, es importante tener en cuenta que el tráfico de red real es extremadamente variable y contiene patrones que pueden solaparse entre comportamiento legítimo y malicioso. Esto hace que la frontera de decisión entre clases no sea clara, lo que limita el rendimiento de cualquier modelo supervisado.

En segundo lugar, el hecho de haber reducido el problema a una clasificación binaria simplifica artificialmente la realidad del tráfico, ya que en entornos reales existen múltiples tipos de ataques con características muy diferentes entre sí. Esta simplificación facilita el entrenamiento, pero también limita la capacidad del modelo para capturar matices más complejos.

Otro factor importante es la calidad del dataset. Aunque CIC-IDS-2017 es un conjunto de datos ampliamente utilizado, sigue siendo una representación simulada de tráfico real, lo que implica que ciertos comportamientos pueden no reflejar completamente lo que ocurre en redes de producción.

En este contexto, el valor del modelo no debe medirse únicamente por su precisión numérica, sino por su capacidad para complementar otros sistemas de detección, como el IDS basado en firmas.

```
==========================================================================
RESULTADOS ÓPTIMOS
==========================================================================
              precision    recall  f1-score   support

      Benign       0.67      0.69      0.68      7500
      Attack       0.68      0.67      0.67      7500

    accuracy                           0.68     15000
   macro avg       0.68      0.68      0.68     15000
weighted avg       0.68      0.68      0.68     15000


Matriz de confusión:
[[5171 2329]
 [2493 5007]]
```

Figura 5. Matriz de confusión obtenida durante la evaluación del modelo Random Forest.

6.5.4 ANÁLISIS CRÍTICO DEL MODELO

Desde un punto de vista crítico, el modelo de Machine Learning desarrollado presenta tanto fortalezas como limitaciones importantes. Por un lado, demuestra que es posible construir un sistema capaz de identificar patrones de tráfico malicioso sin necesidad de reglas explícitas, lo cual aporta una capa adicional de flexibilidad al sistema global.

Sin embargo, también se observa que el modelo es sensible a la calidad de los datos de entrenamiento y a las decisiones tomadas durante el preprocesamiento. Pequeñas variaciones en la selección de características o en el balanceo de clases pueden producir cambios significativos en el rendimiento final.

Otro aspecto relevante es la tasa de falsos positivos, que en este caso sigue siendo relativamente elevada. Esto supone una limitación importante en escenarios reales, ya que un sistema de seguridad que genera demasiadas alertas puede perder efectividad operativa, incluso si técnicamente es capaz de detectar la mayoría de los ataques.

También es importante señalar que el modelo no está integrado directamente con el sistema Snort, lo que significa que ambos componentes funcionan de forma independiente. Esta

separación simplifica el diseño, pero también limita la capacidad del sistema para correlacionar eventos entre la detección por firmas y la detección basada en comportamiento.

6.6 EVALUACIÓN GLOBAL DEL SISTEMA

La integración de todos los componentes desarrollados permite obtener una visión global del sistema como una plataforma de detección de intrusos híbrida. Por un lado, se dispone de un sistema basado en firmas capaz de identificar ataques conocidos con alta precisión en entornos controlados. Por otro lado, se incorpora un modelo de Machine Learning que introduce la capacidad de análisis estadístico del tráfico.

Esta combinación permite abordar el problema de la detección de intrusos desde dos perspectivas complementarias. Mientras que el sistema basado en reglas es más preciso en la identificación de ataques conocidos, el modelo de aprendizaje automático tiene la capacidad de detectar patrones potencialmente anómalos que no están explícitamente definidos.

La visualización en tiempo real añade una capa adicional de valor al sistema, ya que permite observar de forma inmediata la evolución de los eventos en la red. Esto resulta especialmente útil en escenarios de análisis operativo, donde la rapidez en la detección puede ser un factor crítico.

En conjunto, los resultados validan la arquitectura propuesta como una solución funcional y coherente para entornos de monitorización de red a pequeña escala.

6.6.1 ANÁLISIS VISUAL INTEGRADO DE LOS RESULTADOS

La evaluación experimental del sistema puede reforzarse incorporando una lectura visual y cuantitativa de las evidencias que genera cada componente. La Tabla 2 confirma si cada regla de Snort produce o no una alerta, pero una evaluación más completa debe explicar también qué condición activa la detección, qué información aporta la alerta y cuáles son sus principales limitaciones. De acuerdo con la guía NIST SP 800-94, un sistema IDPS no solo identifica posibles incidentes, sino que debe registrar información, notificar eventos relevantes y facilitar su análisis posterior [2].

Por este motivo, los resultados se organizan a continuación en tres niveles complementarios: detección por reglas, monitorización visual y clasificación estadística mediante Machine Learning. Esta separación permite distinguir entre evidencias operativas, que proceden directamente de Snort y de la interfaz, y evidencias analíticas, que proceden del modelo Random Forest entrenado sobre CIC-IDS-2017. La finalidad de esta ampliación no es cambiar la arquitectura desarrollada, sino presentar los resultados de forma más clara, medible y defendible desde el punto de vista experimental.

Escenario evaluado	Criterio de activación	Evidencia generada	Interpretación técnica
ICMP (ping)	Cualquier paquete ICMP capturado por el sensor.	Alerta inmediata en log e interfaz.	Valida captura pasiva; no implica ataque por sí mismo.
SYN flood	5 paquetes SYN hacia un mismo destino en 2 segundos.	Alerta de denegación de servicio.	Patrón volumétrico de saturación de conexiones TCP.
Port scan horizontal/vertical	Repetición de SYN desde un origen o hacia un destino en ventanas cortas.	Alerta de reconocimiento de red o servicios.	Detecta exploración previa a intrusiones posteriores.
UDP flood y HTTP/3/QUIC	Volumen UDP anómalo o UDP hacia puerto 443 con umbral bajo.	Alerta de tráfico UDP o presencia de QUIC.	QUIC debe leerse como indicador de protocolo, no como ataque confirmado.
Random Forest	Clasificación estadística de flujos benignos o maliciosos.	Métricas de exactitud, detección y falsos positivos.	Complementa a Snort con una lectura basada en comportamiento.

Tabla 3. Matriz ampliada de evidencias de detección y análisis del sistema

La Tabla 3 transforma la validación binaria en una matriz de evidencias. Esta lectura resulta especialmente útil porque distingue entre detecciones directas, como SYN flood o UDP flood, y detecciones interpretativas, como HTTP/3/QUIC. En el segundo caso, la alerta no debe entenderse como una intrusión confirmada, sino como un indicador de tráfico que requiere contexto adicional. Esta cautela es importante porque QUIC es un transporte multiplexado sobre UDP y cifra la mayor parte de la comunicación, lo que reduce la visibilidad disponible para un IDS basado en inspección de paquetes [17].

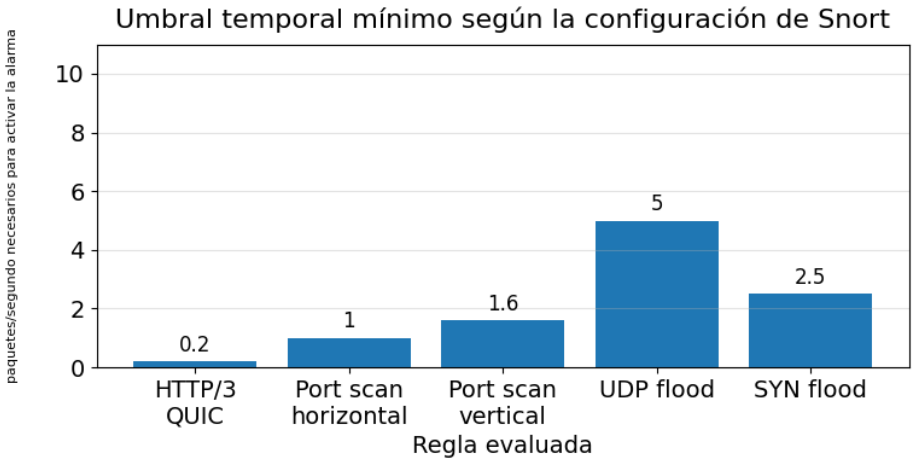

Figura 6. Umbral mínimo de activación de las reglas Snort expresado en paquetes por segundo.

La Figura 6 representa los umbrales temporales definidos en las reglas del sistema, expresados como paquetes por segundo necesarios para activar cada alerta. La regla de UDP flood presenta la tasa más alta, con 5 paquetes/s, mientras que SYN flood se activa con 2,5 paquetes SYN/s hacia el mismo destino. Las reglas de escaneo de puertos requieren tasas menores, ya que su detección depende más del patrón repetitivo de intentos de conexión que del volumen total de tráfico observado. En el caso de HTTP/3/QUIC, el umbral es especialmente bajo, por lo que debe interpretarse como un mecanismo de observación del protocolo y no como criterio suficiente para clasificar el tráfico como malicioso. Esta distinción evita confundir la detección de tráfico específico con la identificación de un ataque.

6.6.2 INDICADORES CUANTITATIVOS PARA REFORZAR LA VALIDACIÓN

Para que la evaluación sea reproducible, las alertas deberían analizarse mediante un conjunto mínimo de indicadores. Estos indicadores pueden obtenerse sin modificar la arquitectura principal: basta con ejecutar cada escenario varias veces, registrar la hora de inicio del ataque, la hora de la primera alerta, el número total de alertas generadas y el tráfico benigno utilizado como control. Con esta información, el proyecto pasa de una comprobación funcional a una validación experimental más sólida.

Indicador	Fórmula o criterio	Cómo medirlo en el sistema	Utilidad en la interpretación
Tasa de detección	TP / (TP + FN)	Comparar ataques ejecutados con ataques alertados.	Mide la capacidad para detectar eventos maliciosos.
Tasa de falsos positivos	FP / (FP + TN)	Ejecutar tráfico benigno y contar alertas indebidas.	Evalúa el ruido generado por reglas demasiado sensibles.
Precisión de alertas	TP / (TP + FP)	Relacionar alertas correctas con alertas totales.	Indica cuánto puede confiar el usuario en cada alerta.
Latencia de alerta	t_alerta - t_inicio	Comparar timestamp del ataque y timestamp del log.	Mide la rapidez del sistema de alerta temprana.
Densidad de alertas	Alertas por minuto	Contar alertas en ventanas temporales homogéneas.	Permite detectar ráfagas y saturación visual.
Pérdida de visibilidad	Paquetes no observados / paquetes emitidos	Comparar captura en h3 con tráfico generado desde h1.	Comprueba si el port mirroring mantiene cobertura suficiente.

Tabla 4. Indicadores recomendados para una evaluación experimental reproducible

La incorporación de estos indicadores permite detectar problemas que no aparecen en una tabla de tipo "sí/no". Por ejemplo, dos reglas pueden activar correctamente una alerta, pero una puede hacerlo con latencia mínima y otra generar decenas de alertas redundantes que dificulten la lectura del panel. Del mismo modo, una regla de escaneo puede ser adecuada en una red pequeña, pero producir falsos positivos en una red con muchos usuarios legítimos abriendo conexiones simultáneas. Por tanto, el valor del sistema no depende solo de detectar, sino de detectar con una relación razonable entre sensibilidad, precisión y coste operativo.

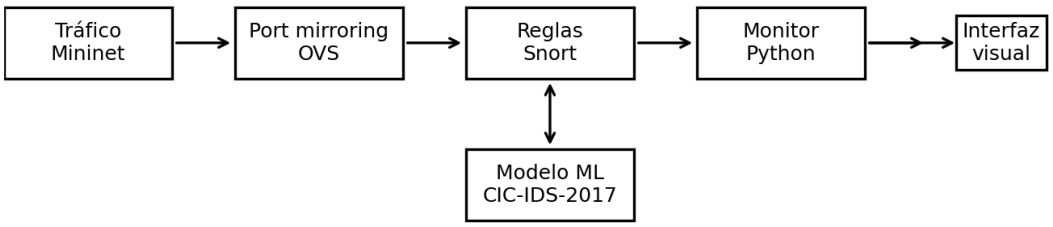

Figura 7. Flujo de evidencias utilizado para interpretar de forma conjunta Snort, monitorización y Machine Learning.

La Figura 7 resume el flujo de evidencias del sistema. Snort proporciona la detección basada en reglas y genera eventos de bajo nivel; el monitor de Python transforma esos eventos en información estructurada; la interfaz visual facilita su interpretación; y el módulo de Machine Learning aporta una evaluación estadística independiente. Esta separación es relevante porque evita presentar el sistema como una única caja negra. Cada módulo aporta una evidencia distinta y, por tanto, también una fuente de incertidumbre diferente.

6.6.3 LECTURA COMPARADA DEL MODELO DE MACHINE LEARNING

El modelo Random Forest debe interpretarse como una capa complementaria y no como sustituto del IDS basado en firmas. Los resultados reportados en la sección 6.5 indican una exactitud aproximada del 68 %, una tasa de detección de ataques cercana al 67 % y una tasa de falsos positivos próxima al 31 %. Estos valores muestran que el modelo aprende patrones útiles, pero todavía genera un volumen relevante de alertas incorrectas. En un entorno docente esto es aceptable como demostración del problema, aunque en un entorno operativo exigiría una optimización adicional del preprocesamiento, del umbral de decisión y de la validación sobre datos no vistos.

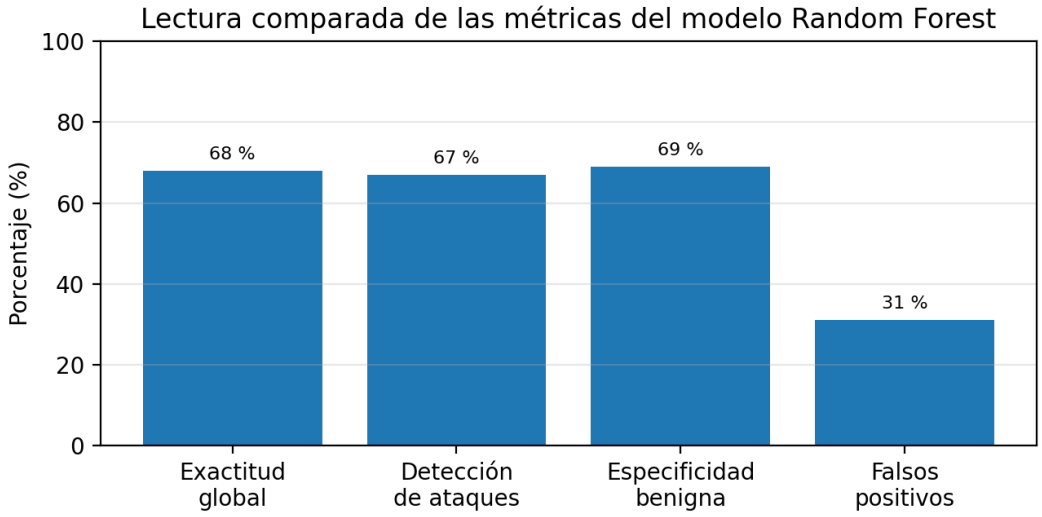

Figura 8. Resumen visual de las métricas principales del modelo Random Forest.

La Figura 8 permite observar el principal compromiso del modelo: la detección de ataques y la identificación de tráfico benigno se mantienen en valores similares, pero la tasa de falsos positivos sigue siendo elevada. Esto significa que el sistema tiende a confundir una parte relevante del tráfico benigno con tráfico malicioso. En ciberseguridad, este comportamiento puede ser problemático porque un exceso de alertas reduce la atención del analista y puede ocultar incidentes realmente importantes. Por ello, la métrica de exactitud no debe analizarse

de forma aislada; debe acompañarse siempre de sensibilidad, especificidad y falsos positivos.

6.6.4 SÍNTESIS DE LA APORTACIÓN VISUAL

La ampliación visual de los resultados permite extraer tres conclusiones principales. En primer lugar, el sistema Snort funciona adecuadamente como detector de patrones simples, especialmente cuando el comportamiento malicioso se expresa mediante umbrales temporales o volumétricos claros. En segundo lugar, la interfaz gráfica añade valor porque convierte el log técnico en una representación comprensible y útil para el aprendizaje. En tercer lugar, el módulo de Machine Learning demuestra potencial como complemento analítico, pero sus métricas evidencian que todavía requiere ajustes antes de poder considerarse una capa de decisión plenamente fiable.

En conjunto, esta lectura integrada refuerza el carácter formativo del proyecto. La plataforma no solo permite lanzar ataques y observar alertas, sino también discutir por qué se genera cada alerta, qué evidencias la respaldan, qué incertidumbre introduce cada técnica y qué métricas deberían emplearse para validar una solución de detección de intrusos. Esta aproximación aproxima el trabajo a una metodología experimental más cercana a la empleada en laboratorios de ciberseguridad y centros de operaciones de seguridad.

6.7 LIMITACIONES DEL SISTEMA

A pesar de que los resultados obtenidos en las fases anteriores permiten validar el correcto funcionamiento general del sistema, es importante realizar un análisis crítico de sus limitaciones, ya que estas condicionan directamente su aplicabilidad en entornos reales y sirven como punto de partida para posibles mejoras futuras.

Una de las limitaciones más evidentes es el tamaño y la simplicidad de la topología de red utilizada. El entorno implementado en Mininet está compuesto por un número reducido de nodos y una estructura de red plana, lo que facilita enormemente tanto la generación de tráfico como la monitorización de este. Sin embargo, esta simplificación también implica que el sistema no ha sido sometido a escenarios de alta complejidad, como redes

segmentadas, múltiples dominios de colisión o entornos con tráfico concurrente elevado. En un entorno real, donde pueden coexistir cientos o miles de dispositivos generando tráfico simultáneamente, el comportamiento del sistema podría variar significativamente.

Otra limitación importante está relacionada con el número de tipos de ataques evaluados. Aunque se han simulado escenarios representativos como ICMP, SYN Flood, escaneo de puertos y tráfico UDP, la realidad es que el espectro de amenazas en redes modernas es mucho más amplio y dinámico. Ataques más avanzados, como técnicas de evasión, ataques cifrados más sofisticados o comportamientos distribuidos a gran escala, no han sido contemplados en este proyecto. Esto implica que el sistema, tal y como está diseñado, está más orientado a la detección de patrones clásicos que a la identificación de amenazas emergentes.

En relación con el módulo de Machine Learning, otra limitación relevante es la dependencia del dataset utilizado. Aunque CIC-IDS-2017 es una referencia ampliamente aceptada en el ámbito académico y uno de los conjuntos de datos más utilizados en investigación sobre detección de intrusiones, sigue representando un entorno controlado que no necesariamente refleja toda la diversidad y complejidad del tráfico presente en redes reales. Esto introduce un posible riesgo de sesgo, ya que el modelo aprende patrones asociados a un escenario concreto que podría no generalizar correctamente ante otros entornos o tipos de tráfico.

Asimismo, deben señalarse ciertas limitaciones metodológicas relacionadas con el proceso de entrenamiento y evaluación del modelo. Durante la fase de preprocesamiento se realizaron operaciones como el tratamiento de valores faltantes, la selección de características y el escalado de variables antes de efectuar la división definitiva entre los conjuntos de entrenamiento y prueba. Desde un punto de vista experimental, este procedimiento puede introducir una fuga de información (data leakage), ya que determinadas características estadísticas del conjunto de prueba quedan indirectamente incorporadas durante la fase de entrenamiento. Como consecuencia, las métricas obtenidas podrían presentar un ligero sesgo optimista respecto al rendimiento real del modelo frente a datos completamente desconocidos.

Una metodología más rigurosa consistiría en realizar inicialmente la partición entre entrenamiento y prueba y aplicar posteriormente todas las transformaciones utilizando

únicamente la información disponible en el conjunto de entrenamiento, trasladando después dichas transformaciones al conjunto de prueba. Aunque esta cuestión no invalida los resultados obtenidos, sí constituye una posible línea de mejora para futuras versiones del sistema.

De forma similar, durante la fase experimental se analizaron distintos umbrales de decisión con el objetivo de optimizar el equilibrio entre capacidad de detección y tasa de falsas alarmas. Sin embargo, la selección del umbral óptimo se realizó utilizando el mismo conjunto de prueba empleado posteriormente para calcular las métricas finales de rendimiento. Desde un punto de vista metodológico, esta aproximación hace que el conjunto de test deje de ser una evaluación completamente independiente, pudiendo introducir un cierto sesgo optimista en los resultados.

En un escenario más riguroso sería recomendable disponer de un conjunto de validación adicional o emplear técnicas de validación cruzada para la selección de hiperparámetros y umbrales de decisión, reservando el conjunto de prueba exclusivamente para la evaluación final del modelo. No obstante, dado que el objetivo principal de este Trabajo Fin de Grado consiste en analizar la viabilidad de aplicar técnicas de aprendizaje automático en un entorno experimental de detección de intrusiones, estas simplificaciones metodológicas se han considerado aceptables para los fines exploratorios del proyecto, aunque constituyen una línea clara de mejora futura.

Además, el problema se ha planteado como una clasificación binaria, lo que simplifica el análisis global, pero limita la riqueza de la información obtenida. En un sistema más avanzado sería deseable distinguir entre múltiples tipos de ataques en lugar de agrupar todo el tráfico malicioso en una única categoría, permitiendo así una respuesta más precisa y detallada por parte del sistema de seguridad.

También es importante destacar que el módulo de Machine Learning se ha evaluado como un componente complementario y en ejecución paralela respecto al sistema basado en Snort, sin integración operativa directa en la generación de alertas en tiempo real. En este sentido, su función se limita a analizar el tráfico de forma independiente, lo que impide explotar sinergias entre detección basada en firmas y detección basada en comportamiento. Una

integración futura permitiría mejorar la capacidad global del sistema combinando ambos enfoques dentro de una arquitectura híbrida.

Otra limitación relevante se encuentra en la gestión de falsos positivos. A pesar de los ajustes realizados sobre los umbrales de decisión y la configuración del modelo, el sistema sigue generando un número apreciable de clasificaciones incorrectas. Este comportamiento es habitual en sistemas IDS, especialmente cuando se utilizan reglas genéricas o modelos entrenados con conjuntos de datos limitados.

Finalmente, el sistema no ha sido evaluado bajo condiciones de carga elevada ni en entornos distribuidos de gran escala, lo que limita la capacidad de extrapolar su rendimiento a escenarios de producción reales donde coexisten numerosos dispositivos generando tráfico simultáneamente. La evaluación de la escalabilidad y el comportamiento en entornos más complejos constituye una línea de trabajo futura especialmente relevante.

6.8 CONCLUSIÓN

El desarrollo y análisis del sistema de detección de intrusos permite extraer una serie de conclusiones relevantes tanto a nivel técnico como conceptual. En primer lugar, se ha demostrado que es posible construir una arquitectura funcional de detección de amenazas utilizando herramientas de código abierto y entornos virtualizados, lo que reduce significativamente la complejidad y el coste de implementación.

El sistema basado en Snort ha mostrado un comportamiento sólido en la detección de ataques conocidos y bien definidos. Su capacidad para identificar tráfico ICMP, ataques de denegación de servicio mediante SYN Flood, escaneos de puertos y patrones de tráfico UDP confirma que las reglas implementadas son efectivas dentro de los escenarios planteados. Este componente representa la base más estable del sistema, ya que permite una detección rápida y determinista de amenazas conocidas.

Por otro lado, el módulo de Machine Learning introduce una capa adicional de análisis que, aunque menos precisa en términos absolutos, aporta valor al sistema al permitir la detección de patrones basados en comportamiento. Esta dualidad entre detección por firmas y detección por comportamiento constituye uno de los aspectos más interesantes del proyecto,

ya que refleja la tendencia actual en ciberseguridad hacia sistemas híbridos que combinan múltiples enfoques.

La capa de monitorización en tiempo real también ha demostrado ser un componente clave dentro del sistema global. La capacidad de visualizar eventos de forma inmediata facilita enormemente la interpretación del estado de la red y permite una reacción más rápida ante posibles incidentes. Este aspecto es especialmente relevante en entornos donde el tiempo de respuesta es crítico.

En conjunto, los resultados obtenidos validan la arquitectura propuesta como una solución coherente y funcional dentro del contexto de un entorno académico y experimental. Aunque el sistema presenta limitaciones evidentes, especialmente en términos de escalabilidad, generalización y reducción de falsos positivos, estas limitaciones no invalidan el enfoque, sino que más bien abren la puerta a posibles líneas de mejora.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

7.1 CONCLUSIONES

El presente Trabajo Fin de Grado ha permitido el diseño, desarrollo e integración de una plataforma experimental orientada a la detección temprana de amenazas en redes de comunicaciones. A lo largo del proyecto se han combinado distintas tecnologías pertenecientes a ámbitos complementarios como la simulación de redes, la ciberseguridad aplicada, los sistemas de detección de intrusos y el aprendizaje automático, con el objetivo de construir un entorno funcional que permitiera analizar el comportamiento del tráfico de red desde múltiples perspectivas.

Uno de los principales objetivos planteados al inicio del proyecto consistía en la creación de un sistema capaz de simular una red de comunicaciones realista, generar tráfico entre distintos nodos y, a partir de dicho tráfico, detectar comportamientos potencialmente maliciosos de forma automatizada. A esto se añadía la necesidad de visualizar los eventos de seguridad en tiempo real, con el fin de facilitar su análisis y comprensión. Tras el desarrollo e implementación de todos los módulos del sistema, se puede afirmar que estos objetivos han sido alcanzados de manera satisfactoria dentro del contexto del entorno experimental diseñado.

En primer lugar, la construcción de la infraestructura de red virtual mediante Mininet y Open vSwitch ha permitido reproducir un entorno suficientemente cercano a una red real como para validar los distintos mecanismos de detección implementados. Aunque se trata de una simulación, el uso de namespaces de red y de conmutadores virtuales ha hecho posible trabajar con conceptos reales de redes de comunicaciones, como el encaminamiento de paquetes, la conmutación a nivel de enlace o la replicación de tráfico mediante técnicas de port mirroring. Esta base ha sido fundamental para el resto del proyecto, ya que sin una infraestructura estable y coherente no habría sido posible analizar de forma fiable el comportamiento del sistema de detección.

Sobre esta infraestructura se desplegó Snort como sistema de detección de intrusos basado en firmas. A través de la definición de reglas personalizadas, el sistema fue capaz de identificar distintos tipos de tráfico y patrones asociados a comportamientos maliciosos. Durante las pruebas realizadas se comprobó su eficacia en la detección de tráfico ICMP utilizado para reconocimiento de red, ataques de denegación de servicio basados en inundación SYN, escaneos de puertos tanto horizontales como verticales, así como tráfico UDP con patrones anómalos y tráfico relacionado con protocolos modernos como QUIC. En todos estos casos, el sistema generó alertas correctamente y en tiempo real, lo que valida su funcionamiento dentro del entorno simulado.

A nivel de procesamiento de eventos, se desarrolló un módulo adicional en Python encargado de actuar como sistema de monitorización. Este componente permitió interpretar las alertas generadas por Snort y transformarlas en información visual comprensible mediante una interfaz gráfica desarrollada con Tkinter. Este sistema de visualización facilitó enormemente el seguimiento de los eventos de seguridad, permitiendo observar de forma inmediata qué tipo de ataque se estaba produciendo, en qué momento y con qué frecuencia. Aunque no se trata de un entorno profesional completo, su comportamiento se asemeja conceptualmente a los paneles utilizados en centros de operaciones de seguridad, lo que refuerza el valor práctico del sistema desarrollado.

Por otro lado, el proyecto ha incorporado un módulo de aprendizaje automático basado en el dataset CIC-IDS-2017, con el objetivo de complementar el enfoque tradicional basado en firmas con una aproximación basada en comportamiento. Tras un proceso de preprocesamiento que incluyó limpieza de datos, selección de características y normalización, se entrenó un modelo Random Forest orientado a distinguir entre tráfico benigno y tráfico malicioso. Los resultados obtenidos, con una precisión aproximada del 68 % y una tasa de detección cercana al 67 %, reflejan que el modelo es capaz de identificar patrones relevantes dentro del tráfico de red, aunque también evidencian las limitaciones inherentes a este tipo de aproximaciones cuando se aplican en escenarios complejos.

Uno de los aspectos más destacables del trabajo ha sido la integración de todos estos componentes dentro de una única plataforma funcional. La combinación de simulación de red, detección basada en reglas, monitorización visual y análisis mediante Machine Learning

ha permitido construir un sistema híbrido que reproduce, a pequeña escala, la estructura de soluciones de ciberseguridad utilizadas en entornos reales. Esta integración no solo aporta valor desde el punto de vista técnico, sino que también demuestra la viabilidad de combinar distintas tecnologías para abordar problemas complejos como la detección de intrusiones en redes.

Desde una perspectiva académica, el desarrollo de este proyecto ha permitido aplicar de forma conjunta conocimientos adquiridos a lo largo del grado en áreas como redes de comunicaciones, seguridad informática, programación en Python, virtualización de sistemas y análisis de datos. Esta combinación multidisciplinar ha sido clave para poder comprender la complejidad real de los sistemas de detección de intrusos y su funcionamiento en entornos operativos.

En conclusión, el trabajo desarrollado demuestra que es posible construir una plataforma de detección temprana de amenazas completamente funcional utilizando herramientas de código abierto y tecnologías ampliamente utilizadas tanto en entornos académicos como profesionales. Aunque el sistema no pretende ser una solución definitiva ni desplegable directamente en producción, sí constituye una base sólida sobre la que se pueden desarrollar soluciones más avanzadas en el futuro.

7.2 TRABAJOS FUTUROS

A pesar de que el sistema desarrollado cumple con los objetivos planteados inicialmente, el propio análisis de los resultados obtenidos permite identificar diversas líneas de mejora que podrían ser abordadas en trabajos futuros con el fin de aumentar la robustez, escalabilidad y precisión del sistema.

Una de las principales líneas de evolución consistiría en ampliar el conjunto de escenarios de ataque considerados en el sistema. En su estado actual, el proyecto se centra principalmente en ataques clásicos como escaneos de puertos, inundaciones de tráfico o reconocimiento mediante ICMP. Sin embargo, en entornos reales de ciberseguridad existen amenazas mucho más complejas y dinámicas, como ataques distribuidos de denegación de servicio, técnicas avanzadas de evasión, malware que utiliza canales cifrados o incluso

ataques dirigidos a aplicaciones web. La incorporación de este tipo de escenarios permitiría acercar el sistema a condiciones más realistas y mejorar su capacidad de detección en situaciones menos controladas.

Otra posible mejora relevante estaría relacionada con el módulo de aprendizaje automático. Aunque el modelo implementado ha permitido obtener resultados aceptables dentro del contexto del proyecto, su rendimiento podría mejorarse significativamente ampliando la cantidad de datos utilizados para el entrenamiento y explorando otros algoritmos de clasificación. Modelos como XGBoost, Support Vector Machines o incluso redes neuronales profundas podrían ofrecer un mejor rendimiento en la detección de patrones complejos dentro del tráfico de red. Además, sería interesante evaluar técnicas de aprendizaje no supervisado para la detección de anomalías, lo que permitiría identificar comportamientos desconocidos sin necesidad de etiquetas previas.

También sería especialmente interesante trabajar en la integración directa entre el sistema basado en Snort y el módulo de Machine Learning. En la implementación actual, ambos sistemas funcionan de manera independiente, lo que limita la capacidad de correlación entre eventos. Una integración más profunda permitiría construir un sistema híbrido real, en el que las alertas generadas por Snort alimenten al modelo de aprendizaje automático y, a su vez, las predicciones del modelo puedan influir en la generación de alertas o en la priorización de eventos. Este tipo de enfoque combinado es cada vez más habitual en sistemas modernos de detección de intrusiones.

Desde el punto de vista de la visualización, otra línea de mejora consistiría en ampliar las capacidades del panel desarrollado. La incorporación de estadísticas en tiempo real, gráficos de evolución del tráfico, almacenamiento histórico de eventos o integración con bases de datos permitiría transformar la herramienta en una plataforma mucho más completa y cercana a soluciones profesionales utilizadas en centros de operaciones de seguridad. Además, la inclusión de sistemas de filtrado y búsqueda avanzada de eventos facilitaría el análisis forense posterior a incidentes de seguridad.

Finalmente, una evolución natural del sistema consistiría en pasar de un modelo puramente de detección a un sistema de prevención activa, incorporando capacidades propias de un Intrusion Prevention System. Esto implicaría no solo detectar ataques, sino también

reaccionar automáticamente ante ellos, por ejemplo, bloqueando tráfico sospechoso, aislando nodos comprometidos o modificando dinámicamente reglas de firewall en función de los eventos detectados. Este tipo de automatización representa uno de los objetivos actuales en el ámbito de la ciberseguridad, especialmente en entornos donde el tiempo de respuesta es crítico.

En definitiva, el trabajo realizado no solo cumple con los objetivos iniciales del proyecto, sino que también abre múltiples líneas de investigación y desarrollo futuro. La combinación de redes virtualizadas, sistemas IDS y técnicas de inteligencia artificial constituye una base sólida sobre la que es posible construir sistemas de seguridad más avanzados, adaptativos y cercanos a los desafíos reales que presentan las redes modernas.

Aunque el sistema desarrollado cumple correctamente los objetivos planteados, existen diversas posibilidades de ampliación y mejora que permitirían aumentar las capacidades de la plataforma.

Capítulo 8. BIBLIOGRAFÍA

[1] Stallings, W. *Network Security Essentials: Applications and Standards*. Pearson, 2017.

[2] Scarfone, K.; Mell, P. *Guide to Intrusion Detection and Prevention Systems (IDPS)*. NIST Special Publication 800-94, 2007.

[3] Axelsson, S. "Intrusion Detection Systems: A Survey and Taxonomy", Technical Report, Chalmers University, 2000.

[4] Sommer, R.; Paxson, V. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection", IEEE Symposium on Security and Privacy, 2010.

[5] Roesch, M. "Snort – Lightweight Intrusion Detection for Networks", Proceedings of the 13th USENIX Conference on System Administration, 1999.

[6] Lantz, B.; Heller, B.; McKeown, N. "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks", ACM SIGCOMM, 2010.

[7] Pfaff, B. et al. "The Design and Implementation of Open vSwitch", USENIX NSDI, 2015.

[8] Cisco Systems. *SPAN (Switched Port Analyzer) Configuration Guide*, Cisco Documentation.

[9] Bishop, C. *Pattern Recognition and Machine Learning*. Springer, 2006.

[10] Sharafaldin, I.; Lashkari, A.; Ghorbani, A. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization", ICISSP, 2018.

[11] Breiman, L. "Random Forests", Machine Learning Journal, vol. 45, no. 1, pp. 5–32, 2001.

[12] Open Information Security Foundation (OISF), *Suricata IDS/IPS/NSM*, 2024.

[13] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks*, vol. 31, no. 23–24, 1999.

[14] J. Wang et al., "Teaching Network Security with Mininet," *IEEE Frontiers in Education Conference*, 2018.

[15] A. Gómez, P. López, "Entorno de simulación interactivo para la docencia en ciberseguridad," *Revista Iberoamericana de Ingeniería Informática*, vol. 10, no. 2, 2022.

[16] Buczak, A.; Guven, E. "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection", *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[17] J. Iyengar y M. Thomson, QUIC: A UDP-Based Multiplexed and Secure Transport, RFC 9000, IETF, 2021.

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

El presente Trabajo Fin de Grado presenta una alineación directa y significativa con varios de los Objetivos de Desarrollo Sostenible (ODS) definidos por la Agenda 2030 de las Naciones Unidas. En particular, se alinea con los ODS 4, 9 y 16, aportando desde el ámbito tecnológico y educativo al fortalecimiento de la formación en ciberseguridad, la innovación y la resiliencia digital.

ODS4 – EDUCACIÓN DE CALIDAD

El desarrollo de este sistema promueve un aprendizaje activo y práctico en el ámbito de las redes y la ciberseguridad. Gracias a la maqueta visual e interactiva, los estudiantes pueden experimentar con escenarios de tráfico realista y comprender los principios fundamentales de los sistemas de detección de intrusiones.
La herramienta contribuye así al fomento de competencias tanto digitales como técnicas, al tiempo que favorece la formación integral en un campo de alta demanda profesional. Además, al estar basada en software de libre acceso, se garantiza la accesibilidad y posible replicación en diversos contextos educativos sin costes adicionales.

ODS 9 – INDUSTRIA, INNOVACIÓN E INFRAESTRUCTURA

Este proyecto impulsa la innovación tecnológica mediante la creación de un entorno virtual de análisis y visualización de tráfico que emula procesos de monitorización y alerta propios de la industria. Al aplicar principios propios de los sistemas de operación de seguridad (SOC) en un entorno académico simplificado, se promueve la transferencia de conocimiento desde la práctica profesional hacia la formación universitaria.

El sistema representa una infraestructura experimental que fomenta la investigación y el desarrollo en ciberseguridad, pudiendo servir como base para futuros proyectos relacionados con redes inteligentes, detección avanzada de amenazas o automatización de respuestas.

ODS 16 – PAZ, JUSTICIA E INSTITUCIONES SÓLIDAS

En un mundo donde los sistemas digitales forman parte de nuestro día a día, la ciberseguridad se ha convertido en un elemento clave para la estabilidad y la confianza en las instituciones. Al proporcionar un entorno en el que resultan observables los efectos de tráfico malicioso y las estrategias para hacerle frente, se fomenta la concienciación y una cultura de la seguridad digital.

Desde esta perspectiva, el proyecto contribuye a fortalecer la resiliencia tecnológica y a formar profesionales con capacidades de diseño de infraestructuras seguras, alineándose con el objetivo de promover instituciones y sistemas más sólidos y confiables.

En conjunto, el proyecto integra la educación, la innovación y la responsabilidad social tecnológica, reforzando los valores de sostenibilidad y desarrollo responsable en el contexto digital actual.

ANEXO II: CÓDIGOS

Script de creación de red (red.py)

```python
from mininet.net import Mininet
from mininet.node import OVSSwitch, OVSController
from mininet.cli import CLI
from mininet.log import setLogLevel
import os
import time

# Ruta absoluta del monitor de alertas
MONITOR_SCRIPT = "/home/paula/TFG_IDS_Visual/scripts/monitor_alertas.py"

def create_topology():
    net = Mininet(controller=None, switch=OVSSwitch)

    # Añadir hosts
    h1 = net.addHost('h1', ip='10.0.0.1/24')
    h2 = net.addHost('h2', ip='10.0.0.2/24')
    h3 = net.addHost('h3', ip='10.0.0.3/24')

    # Añadir switch
    s1 = net.addSwitch('s1')

    # Conectar hosts al switch
    net.addLink(h1, s1)
    net.addLink(h2, s1)
    net.addLink(h3, s1)

    # Iniciar la red
    net.start()

    print("\n" + "=" * 60)
    print("CONFIGURANDO SISTEMA IDS PARA TFG")
    print("=" * 60)

    # ================ MIRRORING ================
    print("[INFO] Configurando port mirroring hacia h3...")
```

```
    s1.cmd("""
    ovs-vsctl -- --id=@p1 get Port s1-eth1 \
              -- --id=@p2 get Port s1-eth2 \
              -- --id=@p3 get Port s1-eth3 \
              -- --id=@m create Mirror name=mirror-h3 \
                 select-src-port=@p1,@p2 \
                 select-dst-port=@p1,@p2 \
                 output-port=@p3 \
              -- set Bridge s1 mirrors=@m
    """)

    # Preparar logs de Snort
    os.system("sudo mkdir -p /var/log/snort 2>/dev/null")
    os.system("sudo touch /var/log/snort/alert")
    os.system("sudo chmod 666 /var/log/snort/alert")

    # Poner h3 en modo promiscuo
    #h3.cmd("ifconfig h3-eth0 promisc")
    h3.cmd("ip link set h3-eth0 promisc on")

    # === TRÁFICO NORMAL ===
    s1.cmd('ovs-ofctl add-flow s1 "actions=normal"')
    print("[INFO] Switch configurado con forwarding normal")

    # Iniciar Snort en h3
    print("[INFO] Iniciando Snort en h3...")
    h3.cmd(
        "snort -i h3-eth0 -A fast -c /etc/snort/snort.conf "
        "-l /var/log/snort/ -K ascii > /dev/null 2>&1 &"
    )
    time.sleep(2)

    # Iniciar monitor de alertas
    print("[INFO] Iniciando monitor de alertas en h3...")
    h3.cmd(f"python3 {MONITOR_SCRIPT} &")

    # Generar tráfico de prueba
    print("[INFO] Generando tráfico de prueba...")
    h1.cmd("ping -c 3 10.0.0.2 > /dev/null")
    time.sleep(1)

    # Verificar alertas
    alertas = h3.cmd("wc -l /var/log/snort/alert 2>/dev/null").strip()
```

```
    print(f"[INFO] Alertas iniciales en /var/log/snort/alert: {alertas}")

    print("\n" + "=" * 60)
    print("LISTO! Comandos útiles:")
    print("  h3 tail -f /var/log/snort/alert    # Ver alertas en tiempo
real")
    print("  h1 ping h2                         # Generar tráfico")
    print("  exit                               # Salir de Mininet")
    print("=" * 60)

    CLI(net)
    net.stop()

if __name__ == "__main__":
    setLogLevel("info")
    create_topology()
```

REGLAS SNORT

```
# 1. ICMP - Ping de prueba
alert icmp any any -> any any (msg: "[TFG-ICMP] ICMP detectado"; sid: 10000001; rev: 1;)
# 2. SYN Flood
alert tcp any any -> any any (flags:S; msg: "[TFG-DDOS] SYN Flood detectado"; threshold: type threshold, track by_dst, count 5, seconds 2; sid:10000002; rev:1;)
# 3. Port Scan horizontal
alert tcp any any -> any any (flags:S; msg: "[TFG-SCAN] Port Scan horizontal"; threshold: type threshold, track by_src, count 10, seconds 10; sid:10000003; rev:1;)
# 4. Port Scan vertical
alert tcp any any -> any any (flags:S; msg: "[TFG-SCAN] Port Scan vertical"; threshold: type threshold, track by_dst, count 8, seconds 5; sid: 10000004; rev:1;)
# 5. UDP Flood
alert udp any any -> any any (msg: "[TFG-DDOS] UDP Flood detectado"; threshold: type threshold, track by_dst, count 10, seconds 2; sid: 10000005; rev:1;)
# 6. HTTP/3 / QUIC traffic
alert udp any any -> any 443 (msg: "[TFG-HTTP3] Tráfico HTTP/3 (QUIC) detectado"; threshold: type threshold, track by_src, count 2, seconds 10; sid:10000006; rev:1;)
```

Script de creación de monitor de alertas (monitor_alertas.py)

```
#!/usr/bin/env python3
import time
from datetime import datetime

ALERT_FILE = "/var/log/snort/alert"

print("="*60)
print(" MONITOR IDS - SNORT")
print("="*60)
print("[INFO] Monitorizando alertas...\n")

with open(ALERT_FILE, "r") as f:
    f.seek(0, 2)

    while True:
        line = f.readline()
        if not line:
            time.sleep(0.3)
            continue

        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        print(f"[{timestamp}] ALERTA DETECTADA")
        print(line.strip())
        print("-"*60)
```

Script ML (cic_ids.py)

```
import pandas as pd
import numpy as np
import os
import time
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import RobustScaler

# ============================================================
# CONFIGURACIÓN
# ============================================================

BENIGN_PATH = "Benign-Monday-no-metadata.csv"
ATTACK_PATH = "Portscan-Friday-no-metadata.csv"
RESULTS_DIR = "resultados_optimos"

os.makedirs(RESULTS_DIR, exist_ok=True)

print("=" * 70)
print("DETECCIÓN DE ATAQUES - VERSIÓN ÓPTIMA")
print("=" * 70)

# ============================================================
# CARGA DE DATOS
# ============================================================

print("[INFO] Cargando datos...")

# Usar 60k muestras para mejor representación
n_samples = 30000  # 30k de cada clase

df_benign = pd.read_csv(BENIGN_PATH)
df_benign["label"] = 0
df_benign = df_benign.sample(n=n_samples, random_state=42)

df_attack = pd.read_csv(ATTACK_PATH)
df_attack["label"] = 1
df_attack = df_attack.sample(n=n_samples, random_state=42)
```

```
df = pd.concat([df_benign, df_attack], ignore_index=True)
df = df.sample(frac=1, random_state=42)

print(f"[INFO] Dataset total: {len(df)} muestras")

# ============================================================
# PREPROCESAMIENTO OPTIMIZADO
# ============================================================

print("[INFO] Preprocesando datos...")

# Eliminar columnas obvias
cols_to_drop = ["Flow ID", "Source IP", "Destination IP", "Timestamp", "Fwd
Header Length.1"]
for col in cols_to_drop:
    if col in df.columns:
        df.drop(columns=col, inplace=True)

# Convertir a numérico
for col in df.columns:
    if col != 'label':
        df[col] = pd.to_numeric(df[col], errors='coerce')

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

# Eliminar columnas con demasiados NaN
nan_threshold = 0.4
cols_with_too_many_nan = [col for col in df.columns
                          if col != 'label' and df[col].isnull().mean() >
nan_threshold]
if cols_with_too_many_nan:
    print(f"[INFO] Eliminando {len(cols_with_too_many_nan)} columnas con
>40% NaN")
    df.drop(columns=cols_with_too_many_nan, inplace=True)

# Llenar NaN con la mediana
for col in df.select_dtypes(include=[np.number]).columns:
    if col != 'label':
        df[col].fillna(df[col].median(), inplace=True)

print(f"[INFO] Dataset después de limpieza: {df.shape}")
```

```
# ============================================================
# SELECCIÓN DE CARACTERÍSTICAS MEJORADA
# ============================================================

print("[INFO] Seleccionando características...")

X = df.drop(columns=["label"])
y = df["label"]

# Eliminar características con varianza muy baja
variances = X.var()
low_var_cols = variances[variances < 0.0001].index.tolist()
if low_var_cols:
    print(f"[INFO] Eliminando {len(low_var_cols)} características con
varianza <0.0001")
    X = X.drop(columns=low_var_cols)

# Calcular correlación con la etiqueta
print("[INFO] Calculando correlaciones...")
correlations = []
feature_names = X.columns.tolist()

for col in feature_names:
    try:
        corr = abs(X[col].corr(y))
        if not np.isnan(corr):
            correlations.append((col, corr))
    except:
        pass

# Ordenar por correlación
correlations.sort(key=lambda x: x[1], reverse=True)

# Tomar top características (más que antes)
max_features = 40
top_features = [col for col, _ in correlations[:max_features]]
X = X[top_features]
print(f"[INFO] Seleccionadas {len(top_features)} características")

# ============================================================
# ESCALADO
# ============================================================
```

```
print("[INFO] Escalando características...")
scaler = RobustScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

# ============================================================
# DIVISIÓN TRAIN/TEST
# ============================================================

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.25, random_state=42, stratify=y
)

print(f"[INFO] Train: {len(X_train):,} | Test: {len(X_test):,}")

# ============================================================
# ENTRENAMIENTO CON MÚLTIPLES PESOS
# ============================================================

print("[INFO] Probando diferentes configuraciones...")
start_time = time.time()

# Configuración 1: Balance equilibrado
model1 = RandomForestClassifier(
    n_estimators=300,
    max_depth=20,
    min_samples_split=10,
    min_samples_leaf=5,
    max_features='sqrt',
    class_weight={0: 1.5, 1: 1.0},  # Balance medio
    n_jobs=-1,
    random_state=42
)

# Configuración 2: Menos peso a benignos (menos falsas alarmas)
model2 = RandomForestClassifier(
    n_estimators=300,
    max_depth=20,
    min_samples_split=10,
    min_samples_leaf=5,
    max_features='sqrt',
    class_weight={0: 1.3, 1: 1.0},  # Balance más ajustado
    n_jobs=-1,
```

```
    random_state=42
)

# Configuración 3: Con pesos basados en la proporción real
n_benign = sum(y_train == 0)
n_attack = sum(y_train == 1)
model3 = RandomForestClassifier(
    n_estimators=300,
    max_depth=20,
    min_samples_split=10,
    min_samples_leaf=5,
    max_features='sqrt',
    class_weight={0: n_attack/n_benign * 1.2, 1: 1.0},  # Peso proporcional
    n_jobs=-1,
    random_state=42
)

# Entrenar modelos
model1.fit(X_train, y_train)
model2.fit(X_train, y_train)
model3.fit(X_train, y_train)

training_time = time.time() - start_time
print(f"[OK] Modelos entrenados en {training_time:.2f} segundos")

# ============================================================
# EVALUACIÓN Y SELECCIÓN DEL MEJOR MODELO
# ============================================================

print("[INFO] Evaluando modelos...")

models = [model1, model2, model3]
model_names = ["Balance 1.5:1", "Balance 1.3:1", "Balance Proporcional"]
best_score = 0
best_model = None
best_model_name = ""
best_threshold = 0.5
best_predictions = None
best_metrics = {}

for idx, (model, name) in enumerate(zip(models, model_names)):
    y_pred_proba = model.predict_proba(X_test)[:, 1]
```

```
    # Probar diferentes umbrales
    thresholds = np.arange(0.35, 0.65, 0.05)
    model_best_score = 0
    model_best_threshold = 0.5
    model_best_pred = None
    model_best_metrics = {}

    for threshold in thresholds:
        y_pred_threshold = (y_pred_proba >= threshold).astype(int)
        tn, fp, fn, tp = confusion_matrix(y_test, y_pred_threshold).ravel()

        precision = tp / (tp + fp) if (tp + fp) > 0 else 0
        recall = tp / (tp + fn) if (tp + fn) > 0 else 0
        f1 = 2 * (precision * recall) / (precision + recall) if (precision +
recall) > 0 else 0
        false_alarm_rate = fp / (fp + tn) if (fp + tn) > 0 else 0

        # Score compuesto: F1 * (1 - false_alarm_rate/2) para penalizar
falsas alarmas
        composite_score = f1 * (1 - false_alarm_rate/2)

        if composite_score > model_best_score:
            model_best_score = composite_score
            model_best_threshold = threshold
            model_best_pred = y_pred_threshold
            model_best_metrics = {
                'precision': precision,
                'recall': recall,
                'f1': f1,
                'false_alarm_rate': false_alarm_rate,
                'tn': tn, 'fp': fp, 'fn': fn, 'tp': tp
            }

    print(f"\n{name}:")
    print(f"  F1: {model_best_metrics['f1']:.4f}")
    print(f"  Precision: {model_best_metrics['precision']:.4f}")
    print(f"  Recall: {model_best_metrics['recall']:.4f}")
    print(f"  Falsas alarmas: {model_best_metrics['false_alarm_rate']:.2%}")
    print(f"  Umbral: {model_best_threshold:.2f}")

    if model_best_metrics['f1'] > best_score:
        best_score = model_best_metrics['f1']
        best_model = model
```

```
        best_model_name = name
        best_threshold = model_best_threshold
        best_predictions = model_best_pred
        best_metrics = model_best_metrics

print(f"\n[INFO] Mejor modelo: {best_model_name}")
print(f"    F1: {best_metrics['f1']:.4f}")
print(f"    Precision: {best_metrics['precision']:.4f}")
print(f"    Recall: {best_metrics['recall']:.4f}")
print(f"    Falsas alarmas: {best_metrics['false_alarm_rate']:.2%}")

# ============================================================
# RESULTADOS FINALES
# ============================================================

report = classification_report(y_test, best_predictions,
target_names=["Benign", "Attack"])
matrix = confusion_matrix(y_test, best_predictions)

print("\n" + "=" * 70)
print("RESULTADOS ÓPTIMOS")
print("=" * 70)
print(report)
print("\nMatriz de confusión:")
print(matrix)

# ============================================================
# ANÁLISIS DETALLADO
# ============================================================

tn, fp, fn, tp = matrix.ravel()

print("\n" + "-" * 40)
print("ANÁLISIS DE RENDIMIENTO:")
print(f"Ataques totales: {tp+fn}")
print(f"Benignos totales: {tn+fp}")
print(f"\nAtaques detectados: {tp} ({tp/(tp+fn):.2%})")
print(f"Ataques perdidos: {fn} ({fn/(tp+fn):.2%})")
print(f"\nBenignos correctos: {tn} ({tn/(tn+fp):.2%})")
print(f"Falsas alarmas: {fp} ({fp/(tn+fp):.2%})")
print(f"\nPrecisión global: {(tp+tn)/(tn+fp+fn+tp):.2%}")

# ============================================================
```

```
# GUARDAR RESULTADOS
# ============================================================

with open(os.path.join(RESULTS_DIR, "resultados_optimos.txt"), "w") as f:
    f.write("=" * 50 + "\n")
    f.write("RESULTADOS ÓPTIMOS - DETECCIÓN DE ATAQUES\n")
    f.write("=" * 50 + "\n\n")
    f.write(f"Modelo: {best_model_name}\n")
    f.write(f"Umbral: {best_threshold:.2f}\n")
    f.write(f"Muestras entrenamiento: {len(X_train):,}\n")
    f.write(f"Muestras prueba: {len(X_test):,}\n")
    f.write(f"Características: {len(top_features)}\n\n")
    f.write(report)
    f.write("\n" + "-" * 30 + "\n")
    f.write("MÉTRICAS DETALLADAS:\n")
    f.write(f"Ataques detectados: {tp} ({tp/(tp+fn):.2%})\n")
    f.write(f"Ataques perdidos: {fn} ({fn/(tp+fn):.2%})\n")
    f.write(f"Benignos correctos: {tn} ({tn/(tn+fp):.2%})\n")
    f.write(f"Falsas alarmas: {fp} ({fp/(tn+fp):.2%})\n")

# Guardar características importantes
feature_importance = pd.DataFrame({
    'feature': top_features,
    'importance': best_model.feature_importances_
}).sort_values('importance', ascending=False)
feature_importance.to_csv(os.path.join(RESULTS_DIR,
"feature_importance.csv"), index=False)

print(f"\n[OK] Resultados guardados en '{RESULTS_DIR}/'")
print("=" * 70)
```

Script de creación del panel visual (panel_visual.py)

```
#!/usr/bin/env python3

import tkinter as tk
from tkinter import ttk
import threading
import time
import os
import sys
from datetime import datetime
import queue
import random


class PanelAlertasTFG:

    def __init__(self, root):

        self.root = root
        self.root.title("TFG - Sistema de Detección de Intrusos")
        self.root.geometry("1000x600")

        self.archivo_alertas = "/var/log/snort/alert"
        self.ultima_posicion = 0

        self.alertas_detectadas = 0

        self.queue_alertas = queue.Queue()

        self.colores = {
            'normal': '#2ecc71',
            'alerta': '#e74c3c',
            'fondo': '#ecf0f1',
            'panel': '#ffffff'
        }

        self.configurar_archivo_alertas()
        self.crear_interfaz()
        self.iniciar_monitoreo()
        self.procesar_cola_alertas()
```

```
# ==========================

    def configurar_archivo_alertas(self):

        os.makedirs("/var/log/snort", exist_ok=True)

        if not os.path.exists(self.archivo_alertas):
            open(self.archivo_alertas, "w").close()

        self.ultima_posicion = os.path.getsize(self.archivo_alertas)


# ==========================
# INTERFAZ
# ==========================

    def crear_interfaz(self):

        self.root.configure(bg=self.colores['fondo'])

        frame_top = tk.Frame(self.root, bg='#2c3e50', height=80)
        frame_top.pack(fill=tk.X)

        tk.Label(
            frame_top,
            text="SISTEMA DE ALERTA TEMPRANA",
            font=("Arial", 18, "bold"),
            bg='#2c3e50',
            fg='white',
            pady=20
        ).pack()

        frame_center = tk.Frame(self.root, bg=self.colores['fondo'])
        frame_center.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

# ---------- IZQUIERDA ----------

        frame_left = tk.Frame(
            frame_center,
            bg=self.colores['panel'],
            relief=tk.RAISED,
            bd=2,
```

```
            width=220
        )

        frame_left.pack(side=tk.LEFT, fill=tk.Y, padx=(0, 5))
        frame_left.pack_propagate(False)

        tk.Label(
            frame_left,
            text="ESTADO",
            font=("Arial", 14, "bold"),
            bg=self.colores['panel']
        ).pack()

        self.indicador_estado = tk.Label(
            frame_left,
            text="NORMAL",
            font=("Arial", 18, "bold"),
            bg=self.colores['normal'],
            fg='white',
            width=12,
            pady=15
        )

        self.indicador_estado.pack(pady=10)

        tk.Label(frame_left, text="Alertas:",
bg=self.colores['panel']).pack()

        self.contador_alertas = tk.Label(
            frame_left,
            text="0",
            font=("Arial", 28, "bold"),
            fg='#e74c3c',
            bg=self.colores['panel']
        )

        self.contador_alertas.pack(pady=10)

        self.label_ultima_alerta = tk.Label(
            frame_left,
            text="",
            bg=self.colores['panel'],
            wraplength=200,
```

```
            justify=tk.LEFT
        )

        self.label_ultima_alerta.pack()


# ---------- DERECHA ----------

        frame_right = tk.Frame(
            frame_center,
            bg=self.colores['panel'],
            relief=tk.RAISED,
            bd=2
        )

        frame_right.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

        columns = ('tipo', 'mensaje', 'severidad')

        self.tree_alertas = ttk.Treeview(
            frame_right,
            columns=columns,
            show='headings'
        )

        self.tree_alertas.heading('tipo', text='TIPO')
        self.tree_alertas.heading('mensaje', text='MENSAJE')
        self.tree_alertas.heading('severidad', text='SEVERIDAD')

        self.tree_alertas.column('tipo', width=120)
        self.tree_alertas.column('mensaje', width=650)
        self.tree_alertas.column('severidad', width=100)

        scrollbar = ttk.Scrollbar(
            frame_right,
            orient=tk.VERTICAL,
            command=self.tree_alertas.yview
        )

        self.tree_alertas.configure(yscrollcommand=scrollbar.set)

        self.tree_alertas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
```

```
        self.tree_alertas.tag_configure('ALTA', background='#ffb3b3')
        self.tree_alertas.tag_configure('MEDIA', background='#ffe0b3')
        self.tree_alertas.tag_configure('BAJA', background='#ccffcc')


# ---------- BOTONES ----------

        frame_bottom = tk.Frame(self.root, bg=self.colores['fondo'])
        frame_bottom.pack(fill=tk.X, pady=10)

        tk.Button(
            frame_bottom,
            text="Simular alerta",
            command=self.simular_alerta,
            width=15
        ).pack(side=tk.LEFT, padx=10)

        tk.Button(
            frame_bottom,
            text="Limpiar",
            command=self.limpiar_alertas,
            width=15
        ).pack(side=tk.LEFT, padx=10)

        tk.Button(
            frame_bottom,
            text="Salir",
            command=self.root.destroy,
            width=15
        ).pack(side=tk.RIGHT, padx=10)


# ========================

    def iniciar_monitoreo(self):

        threading.Thread(
            target=self.monitorear_archivo,
            daemon=True
        ).start()


    def monitorear_archivo(self):
```

```
        while True:

            with open(
                    self.archivo_alertas,
                    "r",
                    encoding="utf-8",
                    errors="ignore"
            ) as f:

                f.seek(self.ultima_posicion)

                nuevas = f.readlines()

                if nuevas:

                    for linea in nuevas:

                        linea = linea.strip()

                        if linea:
                            self.queue_alertas.put(linea)

                    self.ultima_posicion = f.tell()

            time.sleep(0.2)


    def procesar_cola_alertas(self):

        while not self.queue_alertas.empty():

            alerta = self.queue_alertas.get()

            self.procesar_alerta(alerta)

        self.root.after(100, self.procesar_cola_alertas)


    def determinar_severidad(self, texto):

        t = texto.upper()

        if "FLOOD" in t or "DDOS" in t:
```

```
            return "ALTA"

        if "SCAN" in t or "ATTACK" in t:
            return "MEDIA"

        return "BAJA"


    def procesar_alerta(self, texto):

        self.alertas_detectadas += 1

        self.contador_alertas.config(
            text=str(self.alertas_detectadas)
        )

        self.indicador_estado.config(
            text="ALERTA",
            bg=self.colores['alerta']
        )

        if len(texto) > 95:
            texto = texto[:95]

        self.label_ultima_alerta.config(text=texto)

        sev = self.determinar_severidad(texto)

        self.tree_alertas.insert(
            "",
            0,
            values=("SNORT", texto, sev),
            tags=(sev,)
        )


    def simular_alerta(self):

        a = random.choice([
            "TCP PORT SCAN detected",
            "ICMP PING detected",
            "DDoS attack",
            "SQL injection"
        ])
```

```
        with open(self.archivo_alertas, "a") as f:
            f.write(a + "\n")


    def limpiar_alertas(self):

        self.tree_alertas.delete(
            *self.tree_alertas.get_children()
        )


def main():

    root = tk.Tk()

    app = PanelAlertasTFG(root)

    root.mainloop()


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