Aleix Carrillo Solà

DISEÑO E IMPLEMENTACIÓN DE UN IDS

TRABAJO DE FIN DE GRADO

Dirigido por Xavier Palomo Teruel

Grado en Ingeniería Informática

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2025

Resum.

Aquest projecte consisteix a dissenyar i implementar un sistema IDS (Sistema de Detecció d'Intrusos), desenvolupat en un entorn real format per quatre màquines que tenen Ubuntu Server 20.04.

La necessitat del projecte sorgeix a causa de l'augment de ciberatacs a l'època digital que vivim, on protegir-nos davant d'aquestes amenaces és fonamental.

S'ha creat una arquitectura distribuïda on cada màquina compleix una funció específica: les màquines víctimes monitoren el trànsit i detecten els atacs, la màquina atacant genera trànsit maliciós per a les proves controlades, i la màquina recol·lectora centralitza, emmagatzema i exposa les alertes rebudes.

El sistema ha estat desenvolupat a Python a causa de comptar amb biblioteques com Scapy per a l'anàlisi de paquets i Folium per a la visualització geogràfica. Els resultats obtinguts mostren una detecció eficaç dels atacs simulats, validant la utilitat del sistema per a futures investigacions en ciberseguretat.

Resumen.

Este proyecto consiste en diseñar e implementar un sistema IDS (Sistema de Detección de Intrusos), desarrollado en un entorno real formado por cuatro máquinas que disponen de Ubuntu Server 20.04.

La necesidad del proyecto surge debido al aumento de ciberataques en la época digital que vivimos, donde protegernos frente a estas amenazas es fundamental.

Se ha creado una arquitectura distribuida donde cada máquina cumple una función específica: las máquinas víctimas monitorizan el tráfico y detectan los ataques, la máquina atacante genera tráfico malicioso para las pruebas controladas, y la máquina recolectora centraliza, almacena y expone las alertas recibidas.

El sistema ha sido desarrollado en Python debido a contar con bibliotecas como Scapy para el análisis de paquetes y Folium para la visualización geográfica. Los resultados obtenidos muestran una detección eficaz de los ataques simulados, validando la utilidad del sistema para futuras investigaciones en ciberseguridad.

Abstract.

This project consists of designing and implementing an IDS (Intrusion Detection System) system, developed in a real environment consisting of four machines running Ubuntu Server 20.04.

The need for this project arose due to the increase in cyberattacks in our current digital age, where protecting ourselves against these threats is essential.

A distributed architecture was created where each machine fulfills a specific function: the victim machines monitor traffic and detect attacks, the attacking machine generates malicious traffic for controlled testing, and the collecting machine centralizes, stores, and displays the received alerts.

The system was developed in Python due to its libraries such as Scapy for packet analysis and Folium for geographic visualization. The results obtained demonstrate effective detection of simulated attacks, validating the system's usefulness for future cybersecurity research.

Índice

1 INTRODUCCIÓN 6
1.1 CONTEXTO ACTUAL DE LA CIBERSEGURIDAD 6
1.2 OBJETIVOS DEL PROYECTO 6
1.2.1 Objetivos técnicos 6
1.2.2 Objetivos formativos 7
1.3 ALCANCE 7
1.4 ¿PORQUE ES IMPORTANTE UN IDS? 7

2 FUNDAMENTOS TEÓRICOS 9
2.1 ANÁLISIS FORENSE 9
2.2 TIPOS DE IDS 9
2.3 TÉCNICAS DE DETECCIÓN 10
2.3.1 Detección basada en firmas (Signature-based) 10
2.3.2 Detección basada en anomalías (Anomaly-based) 10
2.3.3 Detección basada en políticas (Policy-based) 10
2.4 SISTEMAS DE PREVENCIÓN DE INTRUSOS 11
2.5 TIPOS DE ATAQUES 11
2.5.1 Ataques de Red 11
2.5.2 Ataques a protocolos 13

3 DESCRIPCIÓN GENERAL DEL PROYECTO 15
3.1 ENTORNO TÉCNICO 15
3.1.1 Arquitectura general del sistema IDS 16
3.1.2 Maquina Recolectora 17
3.1.3 Maquina Atacante 17
3.1.4 Maquinas Victimas 18
3.1.5 Asignación de IPs 18

4 REQUISITOS DEL SISTEMA 19
4.1 REPRESENTACIÓN DE CLASES DEL SISTEMA 20
4.2 REQUISITOS FUNCIONALES 21
4.3 REQUISITOS NO FUNCIONALES 26

5 DISEÑO DEL SISTEMA 27
5.1 ARQUITECTURA DE LA APLICACIÓN 27
5.2 DISEÑO DE LA PERSISTENCIA DE DATOS 29
5.2.1 Estructura de la base de datos 29
5.2.2 Índices de la base de datos 30

6 IMPLEMENTACIÓN DEL SISTEMA 31
6.1 DETECCIÓN DE ESCANEO NMAP – NMAP.PY 31
6.2 DETECCIÓN DE ATAQUE DOS – DOS.PY 34
6.3 DETECCIÓN DE ATAQUE DOS HTTP – HTTP_DOS.PY 37
6.4 DETECCIÓN DE INYECCIONES SQL – SQLI.PY 40
6.5 ENVIÓ DE LOS REGISTROS – TRANSFERIR.SH 41
6.6 PROCESAMIENTO Y ALMACENAMIENTO DE LOS REGISTROS – DB_ATAQUES.PY 42
6.7 IMPLEMENTACIÓN DEL SERVIDOR WEB, MAPA Y GRÁFICOS – APP.PY 45
6.8 INTERFAZ WEB HTML – INDEX.HTML 49

7 CENTRALIZACIÓN E INTEGRACIÓN DEL SISTEMA IDS 51
7.1 TRANSFERENCIA DE REGISTROS AUTOMÁTICOS 53

8 SIMULACIÓN DE ATAQUES 56
8.1 ESCANEO DE PUERTOS TCP SYN 56
8.2 DOS TCP FLOOD 57

8.3 DoS ICMP Flood 57
8.4 DoS UDP Flood 58
8.5 DoS HTTP Flood 59
8.6 Escaneo Stealth TCP FIN 60
8.7 Escaneo Stealth TCP XMAS 61
8.8 Escaneo TCP NULL 61
8.9 Escaneo TCP ACK 62
8.10 Ataque SQLi 63

9 EVALUACIÓN 64
9.1 Consideraciones sobre ataques ICMP y UDP Flood 65
9.2 Consideraciones sobre ataque HTTP Flood 66

10 CONCLUSIONES 67

11 ANEXOS 68
11.1 Read.me 68
11.2 Nmap.py 69
11.3 Dos.py 71
11.4 Http_DoS.py 74
11.5 SQLi.py 76
11.6 Transferir.sh 77
11.7 Db_ataques.py 78
11.8 App.py 81
11.9 Index.html 84
11.10 Styles.css 88
11.11 DoS_test.py 95
11.12 SQLi_test.py 96
11.13 main_ids.py 96

REFERENCIAS 99

Índice de tablas

TABLA 1. COMPARACIÓN TIPOS DE IDS 9
TABLA 2. COMPARACIONES TÉCNICAS DE DETECCIÓN 10
TABLA 3. ARQUITECTURA DEL SISTEMA IDS 16
TABLA 4. ASIGNACIÓN DE IPS PÚBLICAS 18
TABLA 5. DESCRIPCIÓN DE LOS CAMPOS BD 29
TABLA 6. REPRESENTACIÓN FLAGS CABECERA TCP 31
TABLA 7. DESCRIPCIÓN COMANDO TCPDUMP DOS 35
TABLA 8. DESCRIPCIÓN COMANDO TCPDUMP HTTP DOS 38
TABLA 9. DEFINICIÓN ESTRUCTURA CARPETA SSH 55
TABLA 10. DESCRIPCIÓN COMANDO NMAP SYN 56
TABLA 11. DESCRIPCIÓN COMANDO HPING3 ICMP 58
TABLA 12. DESCRIPCIÓN COMANDO HPING3 UDP 58
TABLA 13. DESCRIPCIÓN COMANDO APACHEBENCH 60
TABLA 14. DESCRIPCIÓN COMANDO NMAP FIN 60
TABLA 15. DESCRIPCIÓN COMANDO NMAP XMAS 61
TABLA 16. DESCRIPCIÓN COMANDO NMAP NULL 62
TABLA 17. DESCRIPCIÓN COMANDO NMAP ACK 62
TABLA 18. VERIFICACIÓN FIREWALL Y REGLAS DE IPTABLES 65

Índice de figuras

Figura 1. Representación ataques modelo OSI 14
Figura 2. Gestión remota de las máquinas virtuales 15
Figura 3. Flujo del sistema IDS 16
Figura 4. Estructura diagramas de secuencias 19
Figura 5. Representación diagrama de clases 20
Figura 6. Diagrama de secuencia escaneo de puertos 21
Figura 7. Diagrama de secuencia SQLi 22
Figura 8. Diagrama de secuencia DoS 23
Figura 9. Diagrama de secuencia gestión de alertas 24
Figura 10. Diagrama de secuencia geolocalización 25
Figura 11. Representación en capas del sistema IDS 28
Figura 12. Creación de la base de datos 29
Figura 13. Creación índices BD 30
Figura 14. Consulta para la recuperación de ataques 30
Figura 15. Configuración inicial detección Nmap 32
Figura 16. Comparación bits cabecera TCP 32
Figura 17. Registro detalles del ataque Nmap 33
Figura 18. Main del detector Nmap 33
Figura 19. Log detector Nmap 34
Figura 20. Captura de paquetes DoS 34
Figura 21. Registro generado comando tcpdump DoS 35
Figura 22. Configuración inicial detección DoS 36
Figura 23. Patrón detector DoS 36
Figura 24. Identificación ataque DoS 36
Figura 25. Log generado DoS 37
Figura 26. Captura de paquetes HTTP DoS 37
Figura 27. Registro generado comando tcpdump HTTP DoS 38
Figura 28. Patrón detector HTTP DoS 38
Figura 29. Identificación ataque HTTP DoS 39
Figura 30. Log generado HTTP DoS 39
Figura 31. Definición patrones SQLi 40
Figura 32. Identificación ataque SQLi 40
Figura 33. Log generado SQLi 41
Figura 34. Configuración inicial transferencia 41
Figura 35. Lógica principal transferencia 42
Figura 36. Configuración inicial base de datos 42
Figura 37. Lógica geolocalización 43
Figura 38. Lógica creación BD 43
Figura 39. Almacenamiento e importación logs 44
Figura 40. Identificación del tipo de ataque 45
Figura 41. Configuración servidor Flask 45
Figura 42. Lógica principal servidor web 46
Figura 43. Lógica generación del mapa 47
Figura 44. Lógica descarga archivo CSV 48
Figura 45. Ejecución servidor Flask 48
Figura 46. Definición tabla HTML 49
Figura 47. Lógica de inserción mapa y gráficos 50
Figura 48. Representación dashboard 50
Figura 49. Estructura Sistema IDS 51
Figura 50. Ejecución captura del tráfico 51
Figura 51. Definiciones de los detectores 52
Figura 52. Paralelización de las funciones 53
Figura 53. Generaciones de las claves SSH 54
Figura 54. Estructura carpeta SSH 54
Figura 55. Instalación clave SSH en maquina recolectora 55
Figura 56. Escaneo de puertos SYN 56
Figura 57. Script ataque DoS 57

FIGURA 58. EJECUCIÓN ICMP FLOOD 57
FIGURA 59. EJECUCIÓN UDP FLOOD 58
FIGURA 60. EJECUCIÓN HTTP FLOOD 59
FIGURA 61. ESCANEO DE PUERTOS FIN 60
FIGURA 62. ESCANEO DE PUERTOS XMAS 61
FIGURA 63. ESCANEO DE PUERTOS NULL 61
FIGURA 64. ESCANEO DE PUERTOS ACK 62
FIGURA 65. SCRIPT ATAQUE INYECCIÓN SQL 63
FIGURA 66. INICIO SISTEMA IDS 64
FIGURA 67. EJECUCIÓN DEL SISTEMA IDS 64
FIGURA 68. RESULTADO GENERADO DE LA COMPROBACIÓN 65
FIGURA 69. PACKET LOSS ICMP FLOOD 65
FIGURA 70. SERVIDOR WEB APACHE 66

1 Introducción

1.1 Contexto actual de la Ciberseguridad

En esta época digital en la que vivimos, los ciberataques han evolucionado en complejidad y frecuencia, afectando tanto a empresas como a usuarios individuales.

Si observamos los informes anuales de la **ENISA**[1] [2], el número de ciberataques a la red aumentan año tras año respecto al anterior y destacan la importancia de adoptar medidas urgentes de prevención.

Según el ministerio de interior, en su informe[3] sobre la cibercriminalidad en España enfatiza que "Un aspecto que es necesario resaltar es el previsible aumento de la ciberdelincuencia". Además, en palabras de las propias instituciones europeas: "Los ciberataques y la ciberdelincuencia están aumentando en toda Europa, y cada vez son más sofisticados. Esta tendencia seguirá agravándose en el futuro, ya que se espera que 22300 millones de dispositivos en todo el mundo estén conectados a la internet de las cosas".

Este escenario hace esencial un **Sistema de Detección de Intrusos**, capaz de analizar el tráfico de red en tiempo real con la finalidad de identificar y alertar sobre comportamientos maliciosos antes de que comprometan la integridad de los sistemas.

1.2 Objetivos del proyecto

Este proyecto tiene como propósito diseñar e implementar un sistema IDS[4] tipo Sniffer desde cero, utilizando herramientas accesibles para un usuario.

1.2.1 Objetivos técnicos

1. Captura de los paquetes de la red en tiempo real.
2. Análisis de protocolos para detectar posibles ciberataques
3. Generación y envío de alertas para una integración con posibles sistemas SIEMs[5].
4. Geolocalización de atacantes utilizando IPs[6] públicas.

---

1 European Union Agency for Cybersecurity

2 ENISA. Report on the State of the Cybersecurity in the Union https://www.enisa.europa.eu/publications/2024-report-on-the-state-of-the-cybersecurity-in-the-union

3 Ministerio del interior. Informe sobre la cibercriminalidad en España. https://www.interior.gob.es/opencms/export/sites/default/.galleries/galeria-de-prensa/documentos-y-multimedia/balances-e-informes/2023/Informe-Cibercriminalidad_2023.pdf

4 Sistema de Detección de Intrusos

5 Security Information and Event Management

6 Protocolo de Internet

1.2.2 Objetivos formativos

1. Comprender los fundamentos de la detección de intrusos y las técnicas de ataque y defensa en entornos reales.
2. Desarrollo de las habilidades en Python para el análisis de redes y paquetes.
3. Adquirir una experiencia inicial en la monitorización de sistemas.

1.3 Alcance

El proyecto incluye:

- Sniffing sobre servidores accesibles desde Internet
- Detección basada en firmas de ataques conocidos
- Generación de alertas en tiempo real
- Visual de los ataques mediante un mapa interactivo con marcadores de geolocalización.

El proyecto no incluye:

- Análisis de tráfico cifrado como HTTPS[7]/TLS[8].
- Soporte para IPv6.
- Prevención activa de ataques.

En entornos profesionales, herramientas como pueden ser Suricata o Snort se utilizan ampliamente, este proyecto pretende demostrar la posibilidad de construir un IDS funcional desde un diseño inicial con desarrollo propio.

1.4 ¿Porque es importante un IDS?

Actualmente, estamos en una época donde todo está absolutamente conectado, por tanto, la frecuencia en la que se producen los ciberataques cada vez va aumento, además, el impacto que estas tienen en los sistemas es cada vez mayor. Por eso mismo la seguridad en las redes es crítico tanto para los usuarios como para las organizaciones.

Los Sistemas de Detección de Intrusos son soluciones esenciales para poder identificar y mitigar las posibles amenazas en tiempo real. Un IDS actúa como un “sistema de alarma” que comprueba/monitoriza el tráfico en la red con el objetivo de encontrar comportamientos maliciosos, capaces de identificar patrones sospechosos en la red, permitiendo una respuesta rápida antes de que se agrave la situación.

---

7 Hypertext Transfer Protocol Secure

8 Transport Layer Security

Podemos considerar algunos factores que remarcan la importancia de un IDS:

1. Incremento de los ciberataques
2. Monitorizar continuamente el tráfico de red en tiempo real
3. Apoyar al análisis forense, gracias a los registros de ataques detectados.
4. Dar conciencia a la sociedad de la importancia de la ciberseguridad.

2 Fundamentos teóricos

2.1 Análisis forense

El análisis forense en red es una rama que se enfoca en investigar, recolectar y analizar evidencia digital relacionada con incidentes de seguridad.

Su objetivo es identificar ataques, rastrear amenazas, determinar el origen de un incidente y proporcionar pruebas para acciones legales o correctivas.

1. Detectar intrusiones
2. Reconstruir eventos para entender cómo ocurrió un ataque.
3. Identificar el origen (IPs atacantes, usuarios).
4. Generar evidencia válida en procesos legales.
5. Prevenir futuros ataques mediante lecciones aprendidas.

2.2 Tipos de IDS

Dentro de los Sistemas de Detección de Intrusos podemos diferenciar dos tipos:

1. *Network-based IDS*

   Un NIDS[9] tiene la capacidad de monitorizar el tráfico de red en tiempo real. Capaz de analizar paquetes en busca de patrones de ataque.

2. *Host-based IDS*

   Un HIDS[10] tiene la capacidad de detectar cambios que no han sido autorizados en el sistema operativo. Se instalan en diversos dispositivos o equipos individuales, como pueden ser servidores.

Características	NIDS	HIDS
Ubicación	Red	Sistema operativo
Alcance	Múltiples dispositivos	Equipo en el que está instalado
Implementación	Switches, routers	Servidores o endpoints
Amenazas	Escaneos, DoS[11]	Rootkits,
Ejemplos	Snort, Suricata	Wazuh, OSSEC

Tabla 1. Comparación tipos de IDS

9 Network Intrusion Detection System
10 Host-based Intrusion Detection System
11 Denial-of-Service

2.3 Técnicas de Detección

Un IDS puede tener diversos enfoques para poder identificar actividades maliciosas. Las dos técnicas más comúnmente usadas son:

2.3.1 Detección basada en firmas (Signature-based)

Este enfoque lo que hace es monitoriza el tráfico de red y compara esta actividad del sistema con una base de datos que contiene patrones de ataques conocidos o como también se le llama firmas. Si el patrón coincide, se genera una alerta.

Es muy efectivo si se utiliza contra amenazas ya conocidas, su principal desventaja es que es inútil contra ataques ***zero-day***, es decir, no existe su firma en la base de datos.

En un ataque SQLi[12] compara patrones ya conocidos como `"OR 1=1",` una ventaja principal es el bajo porcentaje de falsos positivos.

2.3.2 Detección basada en anomalías (Anomaly-based)

Este tipo de enfoque usa modelos estadísticos/probabilidad o de machine learning. Estos modelos establecen en una red o sistema un tipo de comportamiento que se puede considerar aceptable. Cualquier acción que se desvié de este comportamiento se consideraría sospechosa.

A diferencia de la técnica basada en firmas, esta tiene la capacidad de detectar ataques zero-day, su principal desventaja es su alto costo de procesamiento.

2.3.3 Detección basada en políticas (Policy-based)

Este enfoque utiliza reglas predefinidas como las políticas de acceso, uso de puertos o horarios concretos y si se detectan conexiones fuera de esos parámetros se considerará como una anomalía y se generará una alerta.

Técnica	Zero-day	Falsos positivos	Ideal
Signature-based	No	Bajo	Ataques comunes
Anomaly-based	Si	Alto	Entorno dinámico
Policy-based	Parcialmente	Parcialmente	Entorno restringido

Tabla 2. Comparaciones técnicas de detección

[12] SQL Injection

2.4 Sistemas de Prevención de Intrusos

Este proyecto está enfocado en el desarrollo de un IDS, pero también cabe mencionar la importancia y existencia de los **IPS**[13]. Este tipo de sistemas comparten muchos aspectos con los IDS, pero además de detectar ataques, tienen la capacidad de bloquear o prevenir el tráfico.

Las herramientas comentadas anteriormente pueden realizar la función de un IDS o IPS en función de las necesidades del sistema.

2.5 Tipos de Ataques

Los IDS tienen la capacidad de detectar dos categorías diferentes de ataques.

1. *Ataques de Red*

   Tienen la finalidad de explotar las vulnerabilidades en la infraestructura de la red, como por ejemplo suplantando la identidad o manipulando los paquetes. Buscan los fallos en la configuración o el diseño de la red.

2. *Ataques a protocolos*

   Buscan abusar de las vulnerabilidades en la implementación de protocolos específicos.

2.5.1 Ataques de Red

Denegación de servicio

Un ataque de Denegación de Servicio tiene como objetivo quitar la disponibilidad que tiene un sistema o servicio al sobrecargar los recursos disponibles, como el ancho de banda o la memoria. La diferencia principal entre un DoS y un DDoS[14] es que, en el segundo caso, el ataque está coordinado desde múltiples orígenes aumentando su efectividad.

Un DoS presenta diversos tipos en clasificados según el sistema atacado y las técnicas utilizadas:

- **SYN Flood:** Este tipo vulnera el protocolo TCP[15] enviando múltiples paquetes SYN para sobrecargar los recursos del sistema al nunca completar el 3-way handshake. Se agota la capacidad del servidor, debido a que este guarda recursos para conexiones que realmente no se han establecido.
- **ICMP[16] Flood:** Busca saturar el ancho de banda enviando una gran cantidad de paquetes ICMP echo request, saturando la capacidad del sistema

13 Intrusion Prevention System
14 Distributed Denial-of-Service
15 Transmission Control Protocol
16 Internet Control Message Protocol

- **UDP[17] Flood:** Busca inundar los puertos enviando paquetes UDP completamente aleatorios a puertos abiertos o cerrados. Normalmente se utilizan servicios como DNS[18] o NTP[19] para aumentar su efectividad.
- **HTTP Flood:** Se envían muchas solicitudes HTTP/HTTPS masivas como pueden ser GET o POST a un servidor web en concreto.

Unas de las estrategias más efectivas para poder evitar este tipo de ataques es establecer un umbral del tráfico de la red y alertar en caso de que se superara este límite impuesto.

Man-in-the-Middle

Un ataque MITM[20] busca interceptar o redirigir la comunicación entre dos partes, pero sin que estas tengan conocimiento. Este tipo de ataques tiene como objetivo robar credenciales o poder llegar a escuchar tráfico sensible.

Un MITM presenta diversos tipos en función de a qué sistema se ataque y las técnicas utilizadas:

- **ARP[21] Spoofing/Poisoning:** El atacante asocia su MAC[22] con la IP de otro dispositivo enviando respuestas ARP que no son correctas. Por tanto, el tráfico se redirige primero al atacante antes de llegar a su destino real.
- **DNS Spoofing:** Consiste en que el atacante da respuestas a consultas DNS, pero con direcciones IP falsas con el objetivo de redirigir el tráfico a un servidor falso.
- **SSL[23] Stripping:** Es un ataque comúnmente usado en redes Wi-Fi públicas donde se fuerza a la víctima a eliminar el cifrado de HTTPS haciéndoles usar HTTP, exponiendo la información que se transmite.
- **Rogue AP[24]:** Se crea un punto de acceso Wi-Fi que parece legítimo, pero pertenece al atacante, por tanto, el tráfico siempre pasa por él.

17 User Datagram Protocol
18 Domain Name System
19 Network Time Protocol
20 Man-in-the-Middle
21 Address Resolution Protocol
22 Medium Access Control
23 Secure Sockets Layer
24 Access Point

Escaneo de puertos

El escaneo de puertos busca enumerar los puertos abiertos o servicios de un sistema para poder identificar posibles vulnerabilidades.

En función de las técnicas de escaneo podemos encontrar diversos tipos:

- **Escaneo TCP:** Se envían paquetes SYN y se analizan las respuestas, si se encuentra un SYS-ACK el puerto está abierto, en caso contrario, si se encuentra un RST está cerrado.
- **Escaneo UDP:** El atacante envía paquetes UDP a diversos puertos y comprueba si recibe respuesta. Si no recibe una respuesta el puerto probablemente está abierto.
- **Escaneo Stealth:** Son técnicas de ataques tienen la finalidad de evitar ser detectados por un IDS. Tenemos el FIN Scan el cual el atacante envía paquetes FIN. Por otro lado, tenemos el XMAS Scan el cual envía paquetes FIN, PSH, URG.

2.5.2 Ataques a protocolos

Podemos encontrar diversos ataques a los protocolos HTTP y HTTPS (el cifrado que usa puede ser TLS/SSL). Un IDS trata de analizar el tráfico para detectar las posibles vulnerabilidades.

Inyección SQL[25]: Se inserta código SQL malicioso en los formularios web, con el objetivo de extraer información de la base de datos o hacer un ***bypass*** de la autenticación.

Cross-Site Scripting: Un ataque XSS[26] tiene como objetivo insertar scripts maliciosos, realizados normalmente en JavaScript, en web legitimas. Cuando un usuario accede a la página, los scripts se ejecutan en su navegador permitiendo robar información sensible.

Cross-Site Request Forgery : En un ataque CSFR[27], el atacante engaña a un usuario previamente autenticado para que realice acciones indeseadas en una aplicación web.

25 Structured Query Language
26 Cross-site scripting
27 Cross-Site Request Forgery

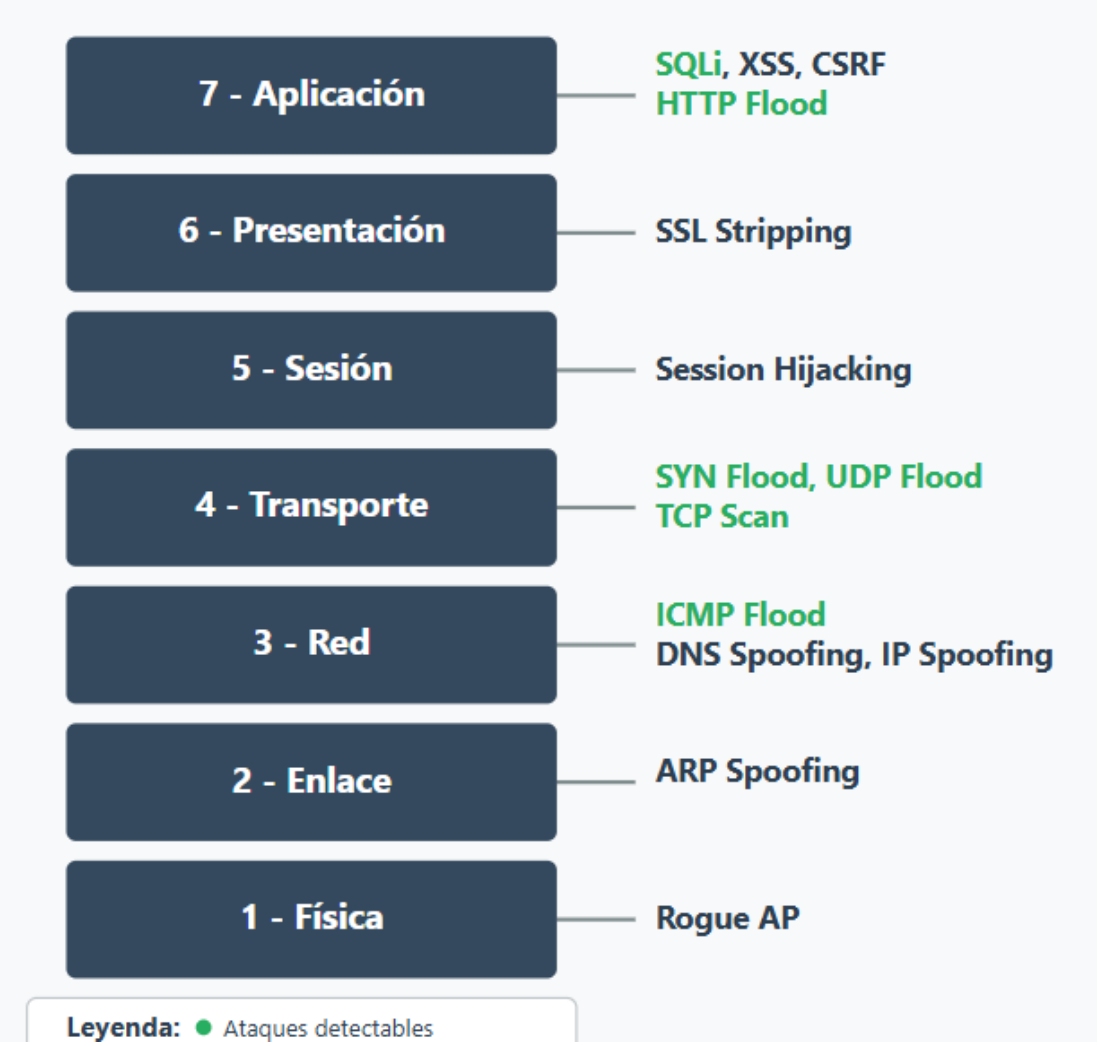

Figura 1. Representación ataques modelo OSI

3 Descripción General del Proyecto

3.1 Entorno Técnico

El proyecto se desarrolla en un entorno virtualizado distribuido, con **cuatro máquinas virtuales,** cada una ejecutando Ubuntu Server 20.04 y cumpliendo con un tipo de rol especifico dentro del sistema IDS. Para la creación y gestión de las máquinas virtuales se ha decidido utilizar **Piensa Solutions**[28]**,** empresa destinada al hosting y registro de dominios, permitiendo configurar las IPs publicas convirtiendo el proyecto en un entorno real pero controlado.

Para la administración de las máquinas virtuales del sistema se ha utilizado **mRemoteNG**, es una herramienta de gestión de conexiones remotas, lo que me ha permitido controlar las múltiples sesiones desde una misma interfaz centralizada para gestionar las conexiones SSH[29].

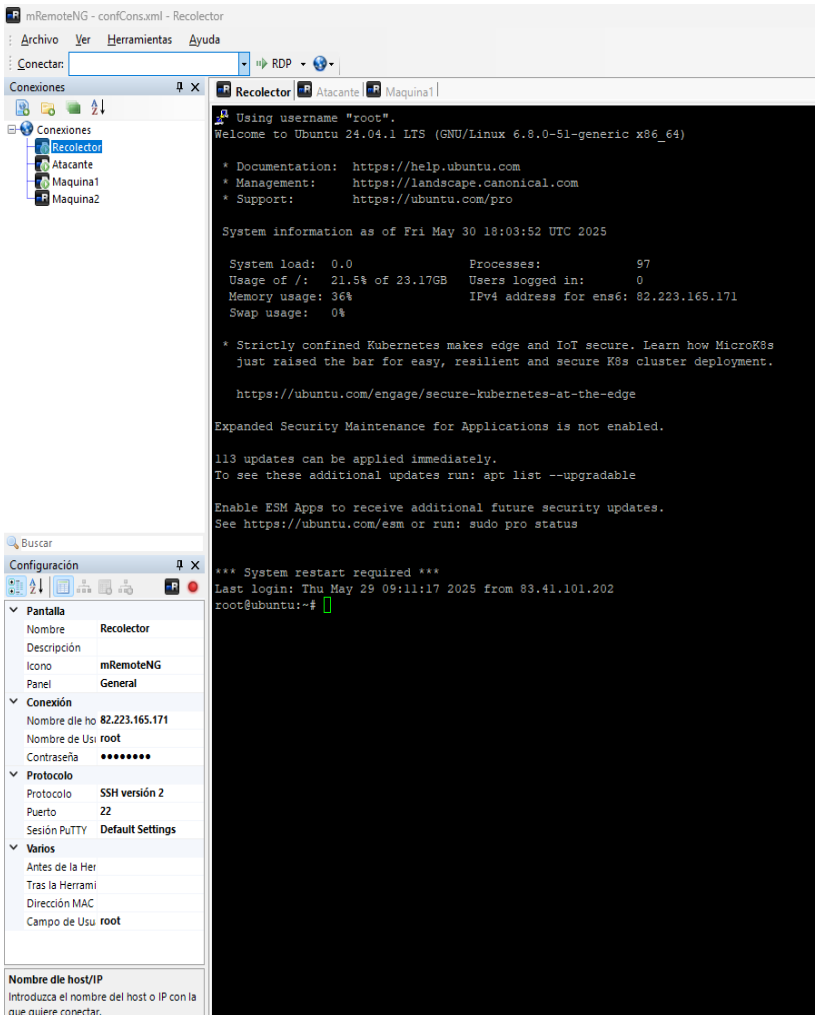

Figura 2. Gestión remota de las máquinas virtuales

28 https://www.piensasolutions.com

29 Secure Shell

El desarrollo del sistema IDS está realizado mediante el lenguaje de programación **Python**. Este lenguaje dispone de multitud de librerías especializadas para el análisis de paquetes y redes o para la creación de APIs[30] web. Podemos ver la diferencia con lenguajes como C++ donde necesitaría una gestión exhaustiva de la memoria y la creación de estructuras más complejas.

Además, Python es uno de los lenguajes más adoptados en el ámbito de la ciberseguridad. Podemos observar múltiples herramientas ya previamente mencionadas, como Wazuh o Volatility para el análisis forense de memoria.

Cada máquina del sistema cumple un rol especifico dentro del sistema IDS distribuido.

3.1.1 Arquitectura general del sistema IDS

La arquitectura del sistema está formada por:

Máquina	Función principal
Recolectora	Almacenar los registros y visualización de la API web
Atacante	Ejecutar ataques controlados para probar el sistema
Victimas	Monitorizan el tráfico y detectan posibles ataques

Tabla 3. Arquitectura del sistema IDS

Figura 3. Flujo del sistema IDS

30 Application programming interface

3.1.2 Maquina Recolectora

La función principal de esta máquina virtual es la de recoger y guardar todas las alertas generadas por las maquinas víctimas, para posteriormente registrar todos los detalles, como la fecha, IP del atacante, tipo de ataque e IP de la víctima, en una base de datos.

Proporciona una interfaz API para poder consultas los datos-

Además, genera un mapa interactivo que muestra geográficamente los ataques recibidos e incluyendo información detallada sobre los incidentes.

Herramientas:

- **SQLite**: Base de datos ligera, ha sido ideal para este proyecto por su simplicidad y no necesitar configuración extra.
- **Flask**: Framework utilizado para la web en Python.
- **Folium**: Es la biblioteca de Python que me ha permitido realizar mapas interactivos con Leaflet.js.
- **Ip-api**: Es un servicio gratuito de geolocalización de IPs. Se ha utilizado juntamente con Folium para para la creación de mapas con las geolocalizaciones de otras maquinas atacantes.

3.1.3 Maquina Atacante

Es la encargada de realizar la simulación de los ataques tales como escaneo de puertos, ataques de denegación de servicio e intentos de inyección SQL.

Los ataques son simulados utilizando herramientas de gestión remota y scripts programados, lo que me ha permitido generar tráfico malicioso de manera controlada.

Herramientas:

- **Nmap**: Herramienta de análisis de red que permite realizar escaneos de puertos. Se ha utilizado para identificar puertos vulnerables en las maquinas víctimas.
- **Hping3**: Herramienta que permite generar paquetes TCP/IP. Me ha permitido hacer simulaciones de ataques de denegación de servicio.
- **Nmap – Zenmap GUI**[31]: Interfaz gráfica de Nmap que me ha facilitado la visualización y configuración de las diversas simulaciones de ataques
- **Scripts personales**: Son un conjunto de archivos desarrollados en Python que me han permitido realizar ataques programados para evaluar el sistema IDS.

31 Graphical user interface

3.1.4 Maquinas Victimas

Encargadas principalmente de la detección de los intrusos, va escaneando el tráfico de red con el objetivo de patrones de ataque que puedan llegar a resultar maliciosos.

Al recibir comportamiento malicioso, realiza un registro de logs que posteriormente enviaran a la maquina recolectora para su registro detallado en la base de datos

Herramientas:

- **Scapy**: Corresponde a una biblioteca de Python para el análisis de paquetes. Se ha utilizado principalmente en las maquinas victimas para la identificación de los diversos ataques simulados en este proyecto.

3.1.5 Asignación de IPs

Las máquinas están conectadas a través de Internet mediante sus direcciones IP públicas. Esta configuración convierte el proyecto en un entorno real controlado. La comunicación entre las maquinas se realiza mediante:

- **SSH**: Se utiliza para el acceso remoto de las máquinas virtuales.
- **SCP**[32]: Transferencia segura de archivos entre las diferentes maquinas.

Máquina	Dirección Ipv4
Recolectora	82.223.165.171
Atacante	70.35.205.202
Victima 1	217.160.225.16
Victima 2	194.164.166.225

Tabla 4. Asignación de IPs públicas

[32] Secure copy protocol

4 Requisitos del Sistema

Se presentan diagramas **UML**[33] para una vista completa y coherente de la arquitectura del sistema IDS, los diagramas presentados en este análisis han sido realizados utilizando la herramienta **Enterprise Architect**, una herramienta para el modelado y diseño de sistemas.

La estructura del modelado en EA[34] incluye todos los diagramas de secuencia organizados por casos de uso, además del diagrama de clases general del sistema.

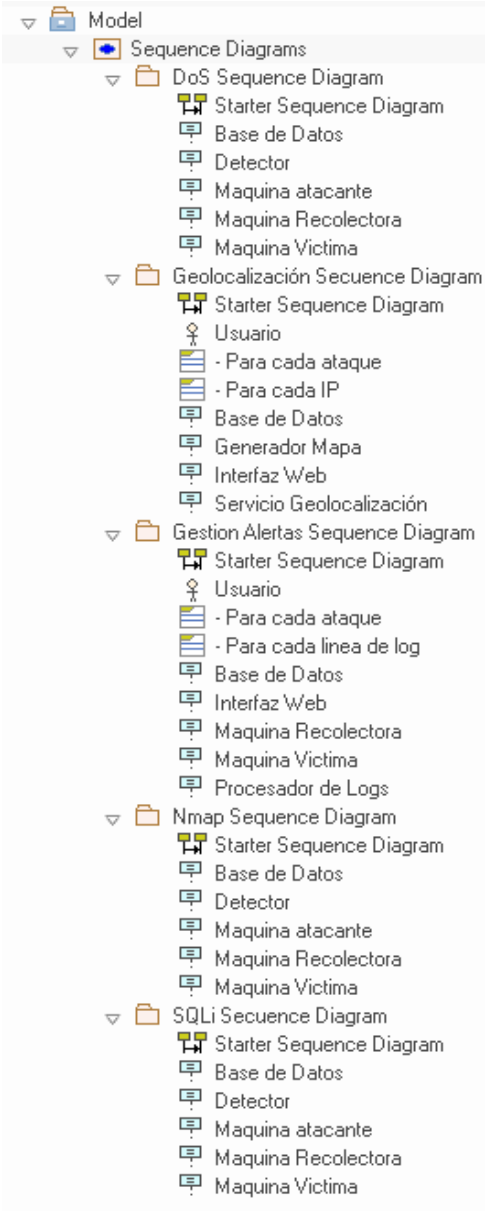

Figura 4. Estructura diagramas de secuencias

33 Unified Modeling Language

34 Enterprise Architect

4.1 Representación de Clases del Sistema

El diagrama de clases representa la estructura del sistema **IDS**, mostrando sus principales componentes. Además, proporciona una base para poder entender los casos de uso descritos más adelante.

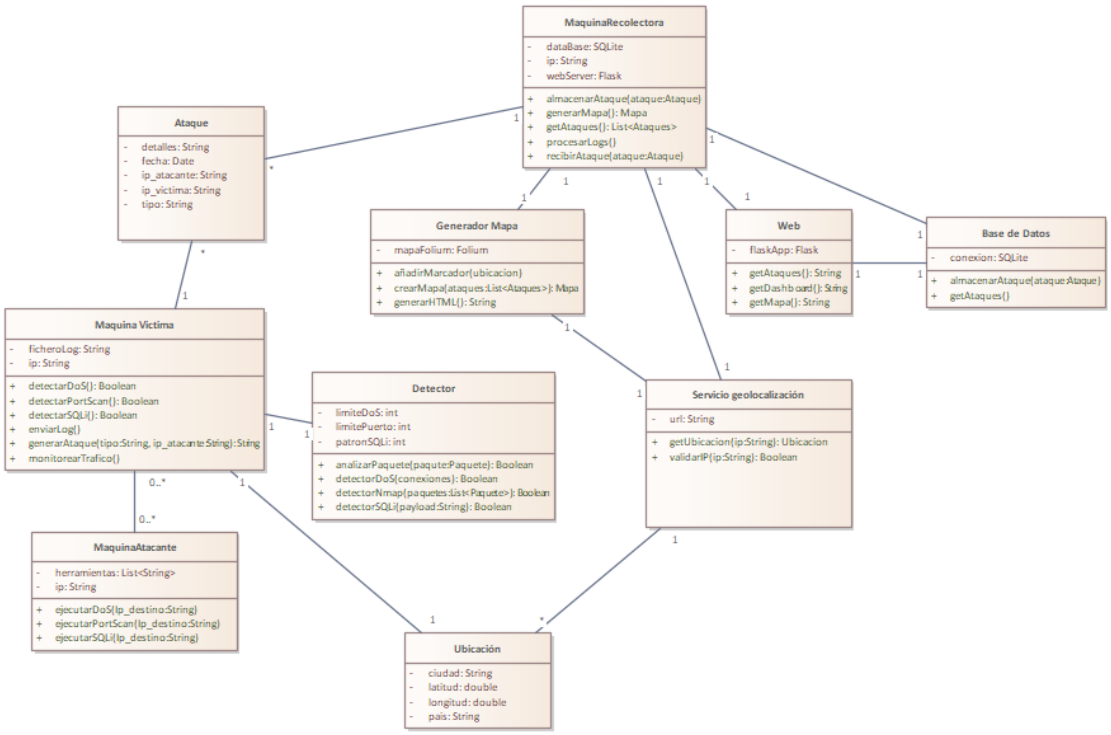

Figura 5. Representación diagrama de clases

4.2 Requisitos Funcionales

1- Detección de un escaneo de puertos

El sistema debe ser capaz de detectar múltiples técnicas de escaneo de puertos, como son: **SYN, FIN, NULL, XMAS y ACK scan**.

Caso de uso:

1. El atacante ejecuta un escaneo de puertos en su máquina virtual mediante comandos específicos hacia la IP de la víctima.
2. La máquina de la víctima monitoriza la red en busca de comportamientos maliciosos utilizando **Scapy**.
3. La máquina de la víctima identifica múltiples paquetes a sus puertos.
4. Se registra en un fichero **.log** el ataque con sus detalles: el IP del atacante, que tipo de ataque se ha realizado, fecha y la IP de la víctima.

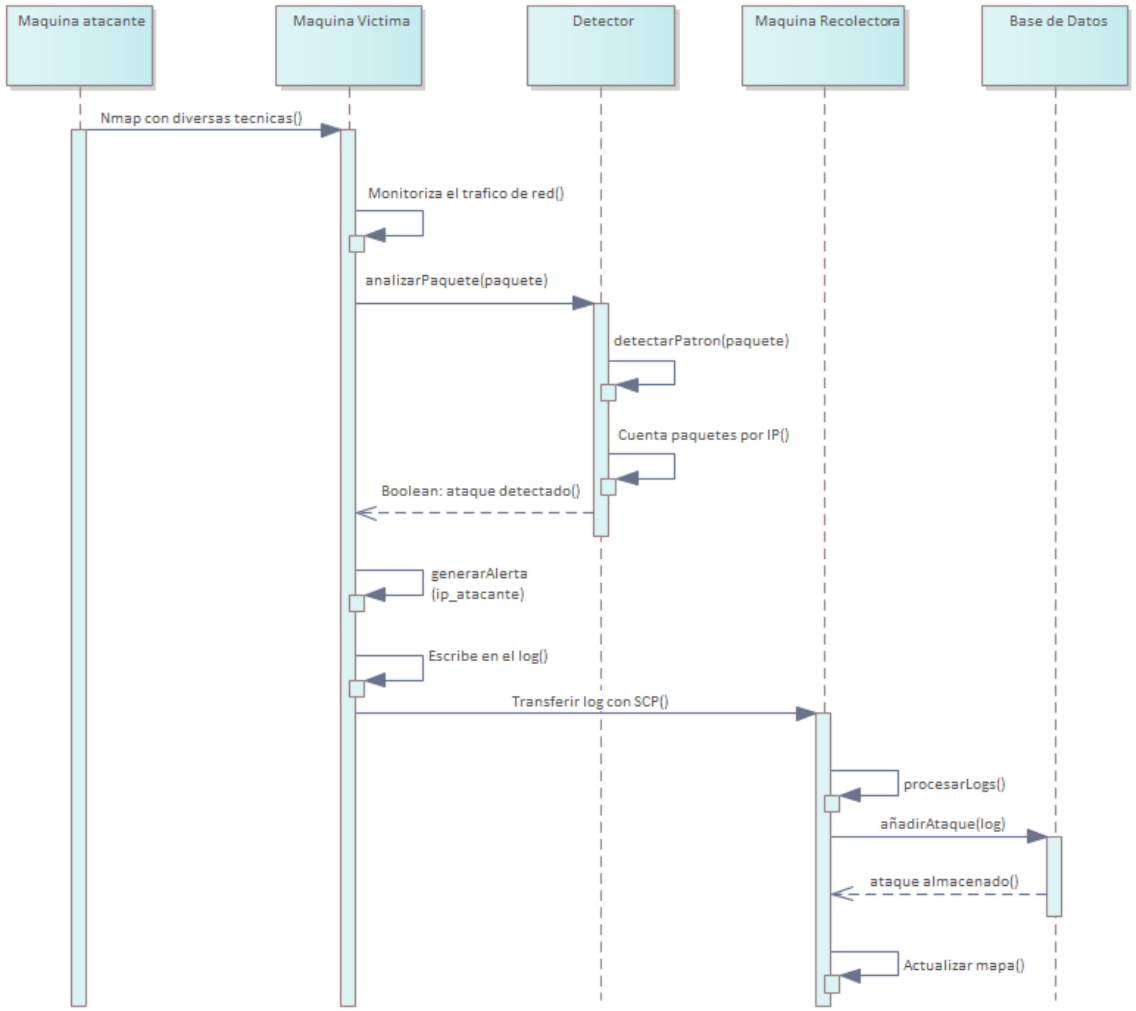

Figura 6. Diagrama de secuencia escaneo de puertos

2- Detección de ataques SQLi

El sistema debe identificar intentos de inyección SQL, simulando peticiones HTTP con payloads maliciosos.

Caso de uso:

1. El atacante ejecuta en su máquina un script que simula peticiones HTTP manipuladas con payloads como **'OR'1'='1, UNION SELECT**.
2. La máquina de la víctima captura los paquetes en tiempo real utilizando **Scapy**.
3. Se analiza el contenido de los payloads buscando coincidencias con los patrones de ataque SQLi definidos previamente.
4. Si se detecta un patrón malicioso, se registra en un fichero **.log**.

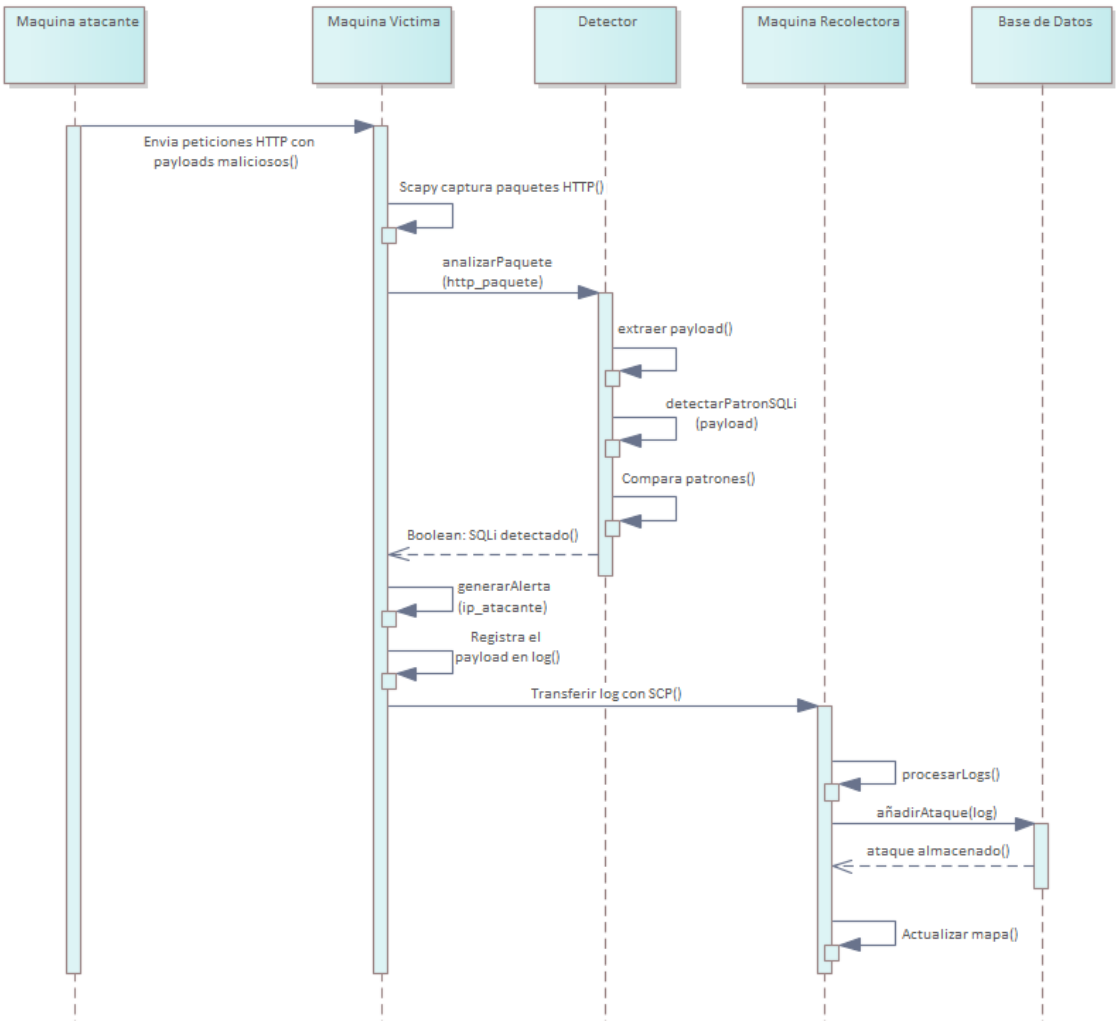

Figura 7. Diagrama de secuencia SQLi

3- Detección de ataques DoS

El sistema debe identificar ataques de Denegación de Servicio si se realizan múltiples conexiones desde una misma **IP** en un rango de tiempo muy corto.

Caso de uso:

1. El atacante lanza múltiples conexiones TCP hacia un puerto especifico de la víctima en un rango muy pequeño de tiempo.
2. La máquina víctima captura el tráfico entrante con la herramienta **tcpdump** y genera un archivo de log.
3. La máquina de la víctima analiza el log buscando algún patrón que sea sospechoso.
4. Si se supera el umbral de conexiones, se registran los detalles del incidente.

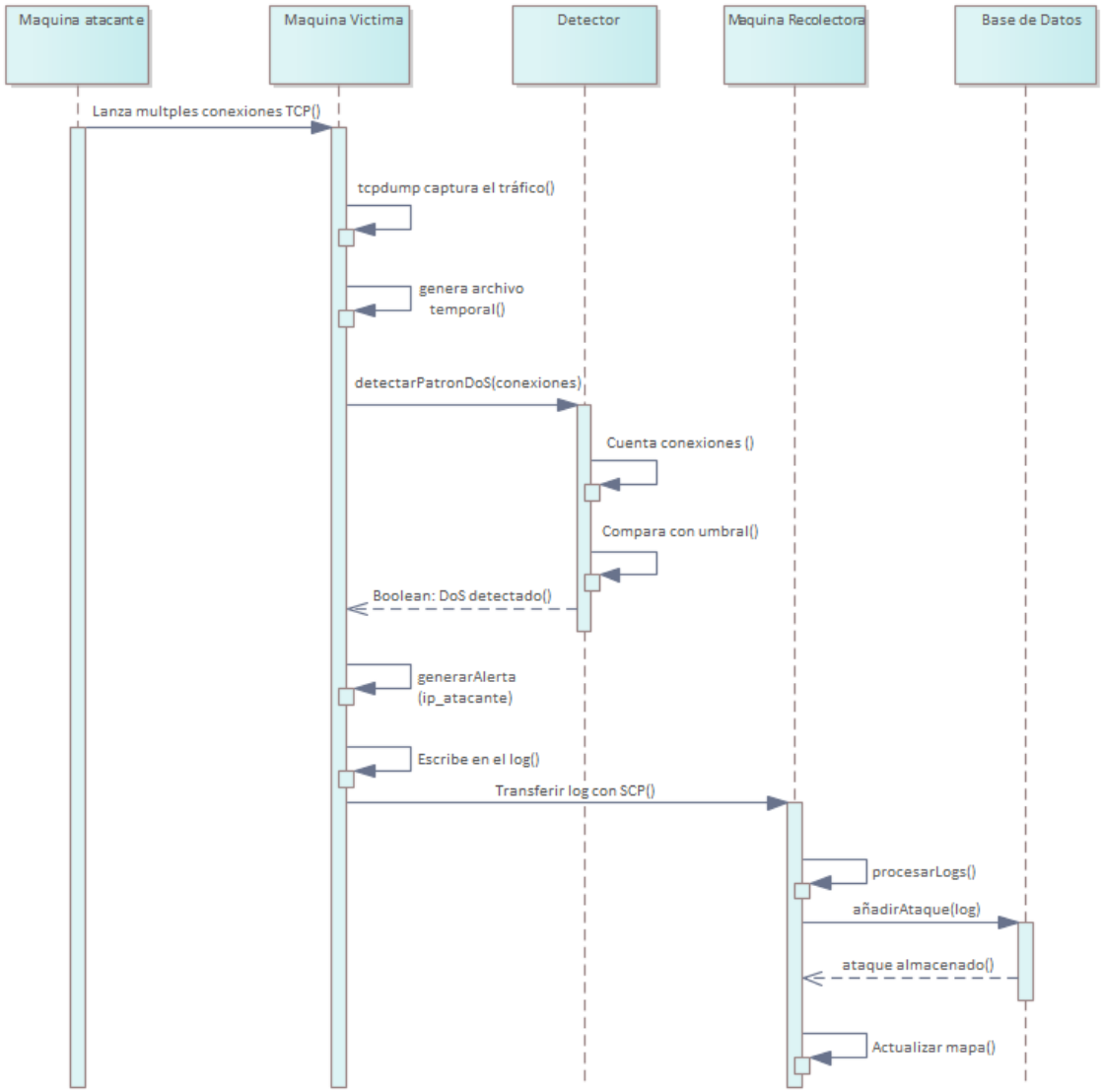

Figura 8. Diagrama de secuencia DoS

4- Gestión de las alertas

El sistema almacena, consulta y visualiza todas las alertas generadas por las maquinas víctima.

Caso de uso:

1. Cuando se detecta alguno de los posibles ataques en una maquina víctima se genera un archivo **.log** con los detalles del ataque.
2. La máquina victima mediante SCP transfiere estos logs a la maquina recolectora.
3. Mediante la ejecución de un script en la maquina recolectora, se analizan estos ficheros **.log**, se extrae la información más relevante del log y se almacena en una base de datos para su posterior consulta.

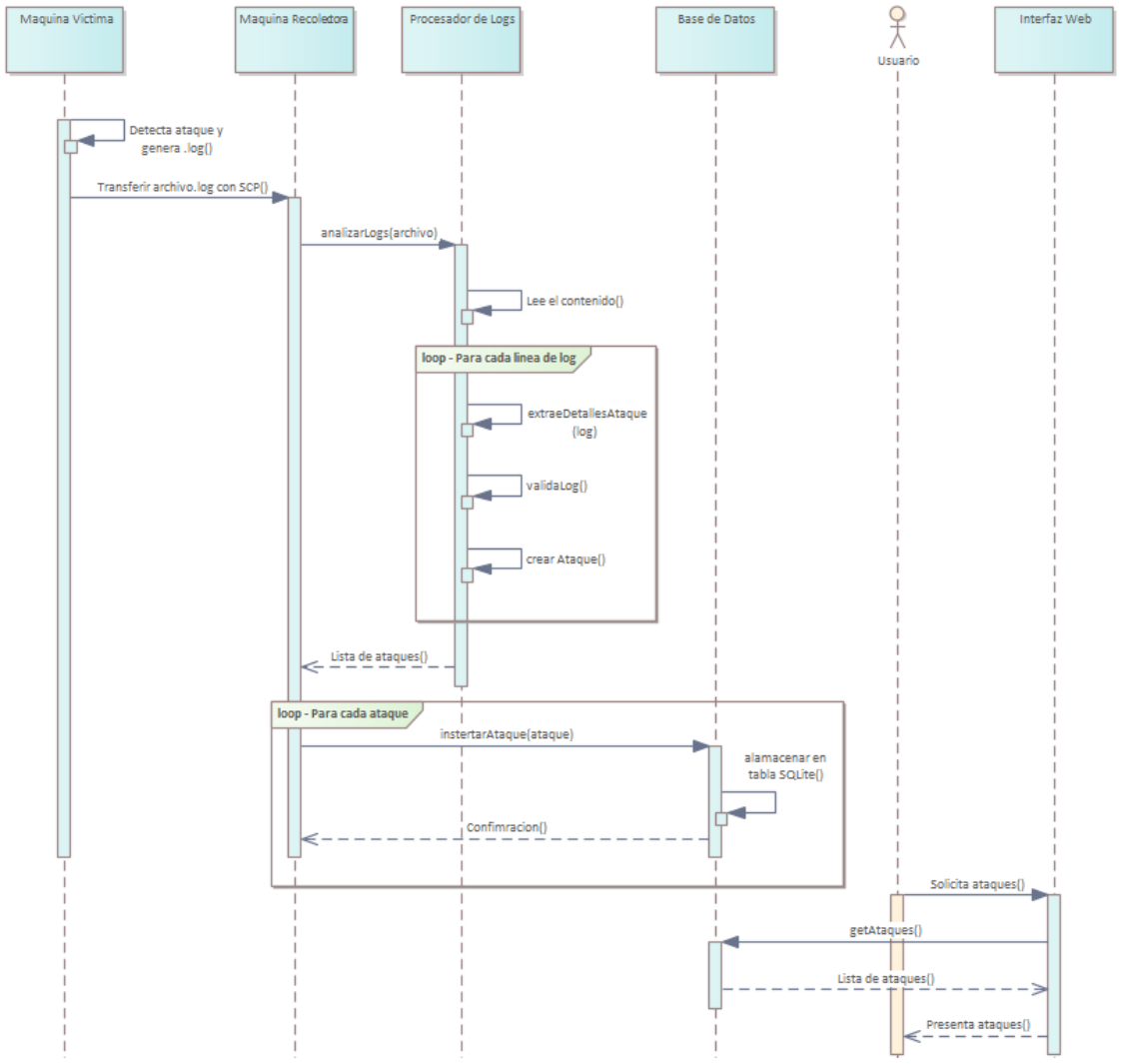

Figura 9. Diagrama de secuencia gestión de alertas

5- Geolocalización de los atacantes

El sistema muestra un mapa interactivo con la ubicación aproximada de los atacantes, utilizando servicios de geolocalización.

Caso de uso:

1. Una vez detectado algunos de los posibles ataques definidos, el sistema registra la IP de origen en la base de datos.
2. El servidor web Flask consulta al servicio de geolocalización externo **ip-api.com** con la finalidad de obtener la ubicación aproximada del atacante.
3. Cuando obtiene la información se genera un mapa dinámico utilizando la librería folium.
4. El usuario que accede al servidor web puede consultar el mapa con los ataques geolocalizados.

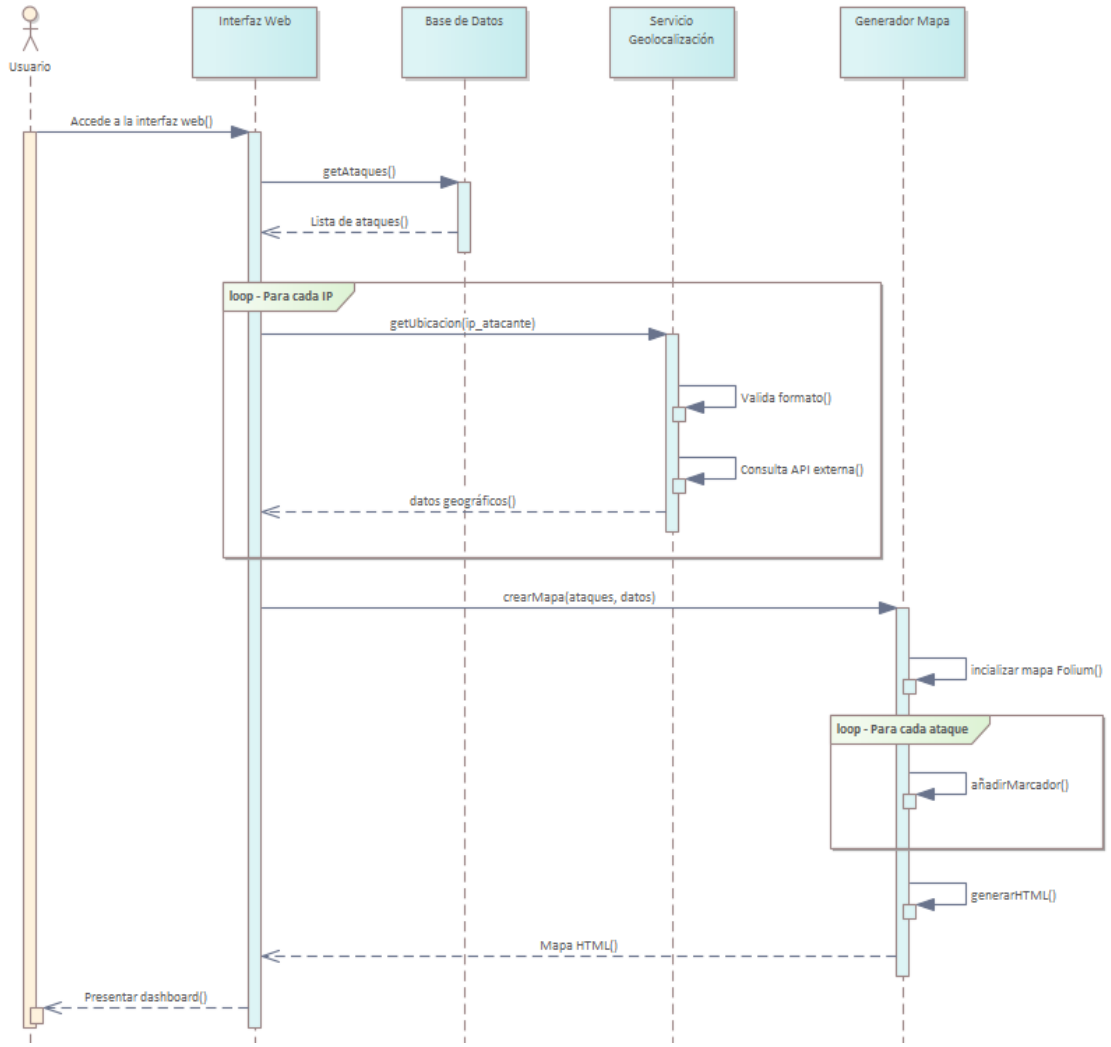

Figura 10. Diagrama de secuencia geolocalización

4.3 Requisitos No Funcionales

1- Interfaz clara

El sistema proporciona una interfaz web para la visualización de los ataques de manera clara, permitiendo al usuario consultar ataques recientes, el tipo y su origen. Además, puede observar gráficos históricos.

2- Transmisión segura

Mediante SCP nos aseguramos de que la comunicación entre las diversas máquinas virtuales sea cifrada con SSH. Además, para la transferencia de los ficheros garantizamos seguridad al tener autenticación mediante clave SSH.

3- Escalabilidad del sistema

El diseño del sistema permite que sea fácilmente escalable, pudiendo adaptarse a más cantidad de máquinas o nuevos ataques.

4- Alta disponibilidad

El sistema está diseñado para ser capaz de operar con una disponibilidad superior al 99% sin que se presenten fallos o caídas, utilizando procesos autónomos.

5 Diseño del Sistema

5.1 Arquitectura de la Aplicación

El sistema IDS sigue una arquitectura distribuida en tres niveles, está pensado para poder realizar pruebas en un entorno controlado con máquinas virtuales.

1. Capa de Captura

Este nivel está representado por las maquinas víctimas, cuyo objetivo es analizar el tráfico de red en tiempo real y detectar posibles comportamientos sospechosos.

Utilizando la biblioteca **Scapy**, se han desarrollado diversos sniffers para poder detectar patrones comunes de ataques.

En esta capa los diversos tipos de ataques que se pueden detectar son:

- ***Escaneo de puertos TCP***: Mediante el análisis de paquetes TCP.
- ***SQLi***: A través del análisis del payload HTTP.
- ***DoS***: Mediante el análisis de múltiples conexiones en ventanas de tiempo muy pequeñas.

Cada ataque que se detecta a través de estas máquinas se guarda automáticamente en archivos .log que contienen la IP del atacante, IP de la víctima, tipo de ataque realizado y fecha.

2. Capa de Procesamiento

La lógica de esta capa se centraliza en la maquina recolectora que actúa como backend para la generación de la web

Mediante scripts en **Python**, se analizan los registros que se reciben de las víctimas y se insertan en una base de datos. Además, se ha tenido en cuenta la posible duplicidad de los registros, esto se controla con `UNIQUE` y `INSERT OR IGNORE`.

Utilizando la API publica ip-api.com se hace una búsqueda de la información geográfica de la IP del atacante para posteriormente generar el mapa con su ubicación y proveedor si este último es conocido.

3. Capa de Presentación

Esta capa es implementada con el framework **Flask**, en la cual me permite exponer una interfaz web interactiva para poder consultar los ataques detectados por el sistema IDS.

Esta interfaz está compuesta por diversos elementos:

- **Tabla HTML**[35] que muestra todos los ataques obtenidos de la base de datos. Cada fila muestra información importante sobre el ataque.

[35] HyperText Markup Language

- Mapa interactivo generado con Folium, cada ataque se representa con un marcador según la IP del atacante. Este mapa nos permite tener una representación visual de los diversos ataques
- Gráficos estadísticos utilizando la librería Chart.js. Se pueden observar dos tipos de gráficos, el grafico de barras en función del tipo de ataque detectado y una grafico de líneas que muestra el número de ataques por fecha.
- Descarga desde la interfaz web de todos los registros almacenados en la base de datos en formato **CSV**[36].

Esta capa nos permite visualizar los ataques con una perspectiva global y al mismo tiempo resulta una herramienta útil para las revisiones en caso de brechas en la seguridad o incluso para integrarlos en sistemas SIEM para un posterior análisis.

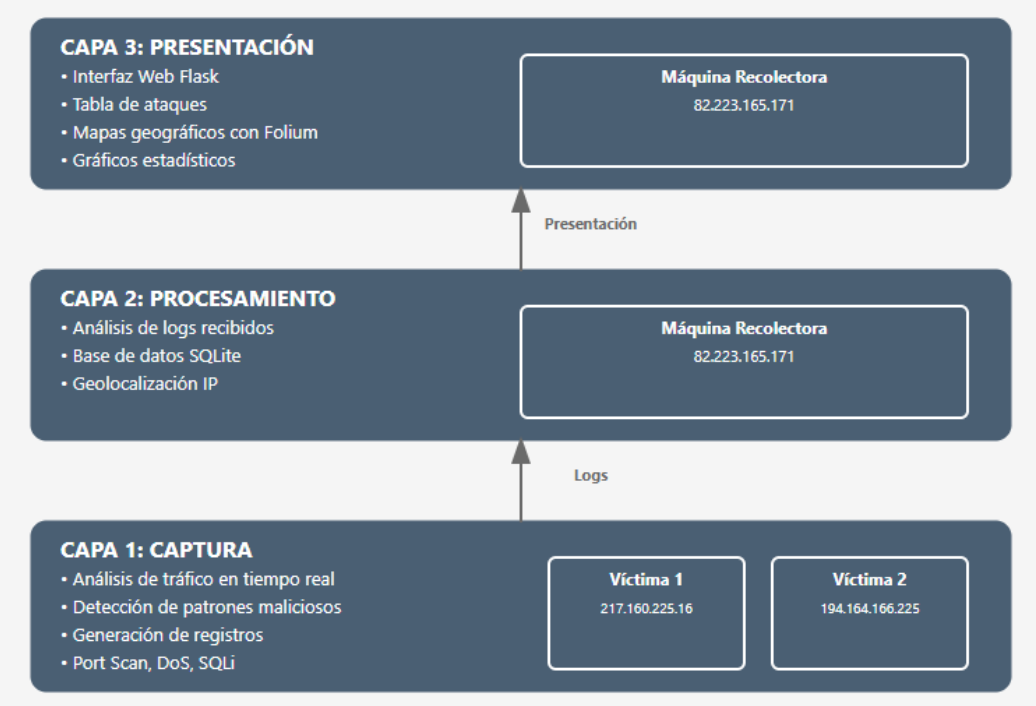

Figura 11. Representación en capas del sistema IDS

Flujo del sistema:

1. La máquina atacante realiza tráfico malicioso hacia las víctimas.
2. Utilizando los scripts de detección, analizan le tráfico en busca de posibles patrones y generan los logs detallados.
3. Envían el registro de este tráfico a la maquina recolectora.
4. Se procesan los logs recibidos y se insertan en la base de datos, por tanto, se actualiza el mapa generado.

[36] Comma-separated values

5.2 Diseño de la Persistencia de Datos

La persistencia de los datos se realiza mediante SQLite, este tipo de base de datos es ideal para este sistema controlado al no requerir la instalación de servidores y por su facilidad de configuración.

Para almacenar los diversos ataques detectados por las maquinas víctimas se define una única tabla principal **ataques.**

5.2.1 Estructura de la base de datos

La creación de la tabla ataques se hace mediante el siguiente código:

```
    CREATE TABLE IF NOT EXISTS ataques (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        fecha TEXT NOT NULL,
        ip_victima TEXT NOT NULL,
        ip_atacante TEXT NOT NULL,
        tipo TEXT,
        pais TEXT,
        detalle TEXT,
        UNIQUE(fecha, ip_victima, ip_atacante, tipo, detalle)
    )
'''\
```

Figura 12. Creación de la base de datos

Campo	Tipo	Descripción
Id	INT	Identificador único del ataque
Fecha	TEXT	Fecha del ataque
Ip_atacante	TEXT	Dirección IPv4 de la maquina atacante
Ip_victima	TEXT	Dirección IPv4 de la maquina victima
Tipo	TEXT	Tipo de ataque detectado
País	TEXT	Información geográfica del atacante
Detalle	TEXT	Información adicional (payload, puertos, ventana de tiempo)

Tabla 5. Descripción de los campos BD

Para evitar la duplicación de los registros si se procesa el mismo archivo de log múltiples veces, he definido la restricción UNIQUE sobre los campos (fecha, ip_atacante, ip_victima, tipo, detalle).

5.2.2 Índices de la base de datos

Para mejorar el rendimiento de las consultas si hay un gran volumen de registros se ha creado los diversos índices:

```
cursor.execute('CREATE INDEX IF NOT EXISTS index_fecha ON ataques(fecha)')
cursor.execute('CREATE INDEX IF NOT EXISTS index_tipo ON ataques(tipo)')
```

Figura 13. Creación índices BD

Esto ayuda a optimizar el filtrado de los ataques o la generación de los gráficos de la web. Además, desde el backend Flask, se realiza una consulta para recuperar todos los ataques, para posteriormente mostrarlos en la interfaz gráfica.

```
conn = sqlite3.connect('/root/logs_ataques.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM ataques ORDER BY fecha DESC")
ataques = cursor.fetchall()
conn.close()
```

Figura 14. Consulta para la recuperación de ataques

6 Implementación del Sistema

La implementación del sistema IDS se ha realizado mediante un conjunto de scripts en Python y Bash que se ejecutan en las diversas máquinas virtuales según el rol que tienen asignado.

En esta sección se describen los scripts más importantes que forman el sistema. Para cada código se detallará su funcionalidad principal y destacare las partes claves.

6.1 Detección de escaneo Nmap – Nmap.py

Este script implementa un detector de escaneo de puertos capaz de detectar las diversas técnicas explicadas anteriormente:

- SYN Scan
- FIN Scan
- XMAS Scan
- NULL Scan
- ACK Scan

Se utiliza Scapy para hacer un análisis en tiempo real del tráfico TCP que llega a las maquinas víctimas. Se pretende detectar los valores de los flags que están presentes en la cabecera del paquete TCP.

1. Estructura de la cabecera TCP

La cabecera del paquete TCP contiene un campo **flags** de 6 bits principales. Estos flags serán clave para poder detectar los diversos escaneos de puertos.

Flag	Posición	Binario	Hexadecimal
FIN	Bit 0	00000001	0x01
SYN	Bit 1	00000010	0x02
RST	Bit 2	00000100	0x04
PSH	Bit 3	00001000	0x08
ACK	Bit 4	00010000	0x10
URG	Bit 5	00100000	0x20

Tabla 6. Representación flags cabecera TCP

2. *Inicializaciones*

```
def __init__(self, alertas="/root/registros/registresNmap.log"):
    self.LIMIT = 0
    self.TEMPS_LIMIT = 10
    self.contador = defaultdict(int)
    self.iniciTemps = time.time()
    self.alertas = alertas
    self.ip_victima = self.ipLocal()
    self.detecciones_totales = 0
    self.tipos_detecciones = defaultdict(int)
    if not os.path.exists(self.alertas):
        with open(self.alertas, 'w'): pass
    atexit.register(self.mostrar_estadisticas)
```

Figura 15. Configuración inicial detección Nmap

- **LIMIT_SYN**: Representa el número mínimo de paquetes desde una misma dirección IP para que se considere un ataque.
- **TEMPS_LIMIT**: Rango de tiempo que se define para considerar que los paquetes vienen agrupados.
- **contSyn**: Contador de cuantos paquetes se reciben por IP.
- **iniciTemps**: Representa el inicio del rango de tiempo.

3. *Captura de los paquetes*

```
def analizarPaquete(self, paquet):
    if paquet.haslayer(TCP):
        flags = paquet[TCP].flags
        ip_origen = paquet[IP].src
        portDesti = paquet[TCP].dport

        tipo = None
        if flags == 0x02:
            tipo = "SYN scan"
        elif flags == 0x01:
            tipo = "FIN scan"
        elif flags == 0x00:
            tipo = "NULL scan"
        elif flags == 0x29:
            tipo = "XMAS scan"
```

Figura 16. Comparación bits cabecera TCP

En este bloque de código se verifican los paquetes TCP y se detecta el tipo de escaneo en función del flag activado. Se extrae la IP de origen y el puerto de destino que está comprobando y se incrementa el contador de los paquetes recibidos por esa IP.

4. Registro del ataques

```
if tipo:
    self.contador[(ip_origen, tipo)] += 1
    if self.contador[(ip_origen, tipo)] > self.LIMIT:
        self.detecciones_totales += 1
        self.tipos_detecciones[tipo] += 1
        alerta = (
            f"    Maquina victima: {self.ip_victima}\n"
            f"    Nmap ip: {ip_origen}\n"
            f"    Tipo: {tipo}\n"
            f"    Paquete: {self.contador[(ip_origen, tipo)]}\n"
            f"    Puerto: {portDesti}\n"
            "----------------------------------------"
        )
        self.guardarFitxer(alerta)
        self.contador[(ip_origen, tipo)] = 0

if time.time() - self.iniciTemps > self.TEMPS_LIMIT:
    self.contador.clear()
    self.iniciTemps = time.time()
```

Figura 17. Registro detalles del ataque Nmap

Cuando se supera el umbral definido anteriormente, se genera una alerta con la información del ataque y se guarda el registro en el archivo **registresNmap.log**

5. Ejecución del script

```
if __name__ == "__main__":
    detector = IdsNmap()
    try:
        sniff(prn=detector.analizarPaquete, filter="tcp", store=0)
    except KeyboardInterrupt:
        print("\nDetenido")
    finally:
        pass
```

Figura 18. Main del detector Nmap

Se inicia la función ***sniff()*** de Scapy, captura el tráfico TCP que llega a la máquina virtual y llama a la función ***analizarPaquete()*** para cada paquete que capture.

6. Registro generado

```
  GNU nano 7.2
[2025-06-03 19:10:13]
    Maquina victima: 217.160.225.16
    Nmap ip: 70.35.205.202
    Tipo: XMAS scan
    Paquete: 1
    Puerto: 80
----------------------------------------
[2025-06-03 19:10:13]
    Maquina victima: 217.160.225.16
    Nmap ip: 70.35.205.202
    Tipo: XMAS scan
    Paquete: 1
    Puerto: 22
```

Figura 19. Log detector Nmap

Este log será enviado posteriormente al recolector que luego almacenará en la base de datos.

El código completo de este script está disponible en el Anexo 11.2

6.2 Detección de ataque DoS – dos.py

Este script implementa un detector de posibles ataques de Denegación de Servicio realizando un análisis del archivo de tráfico ***dos.log.*** Su funcionamiento es observar si una misma dirección IP realiza un número muy elevado de conexiones en rango de tiempo corto.

1. Captura con tcpdump

```
def iniciar_tcpdump():
    comando = [
        "tcpdump", "-tttt", "-n", "-A", "-i", "ens6",
        "(icmp or udp or tcp)", "-l"
    ]
    with open("/root/trafico/dos.log", "w") as f:
        return subprocess.Popen(comando, stdout=f, stderr=subprocess.DEVNULL)
```

Figura 20. Captura de paquetes DoS

Primeramente, para poder detectar ataques de tipo DoS, necesita un registro detallado de las conexiones que entran. Se captura de forma conjunta el tráfico de los protocolos TCP, UDP, ICMP. esto nos permite detectar múltiples variantes de ataques DoS en una misma captura. Para generar este registro se ha utilizado la herramienta ***tcpdump***.

Parámetro	Descripción
-tttt	Muestra el timestamp
-n	Muestra directamente la dirección IP
-A	Muestra el contenido en ASCII[37] para el protocolo HTTP
-i ens6	Nos indica la interfaz
-l	Permite que la salida se escriba en tiempo real
'tcp or udp or icmp'	Filtro para poder capturar múltiples protocolos.
stdout=f	Redirige toda la salida al archivo dos.log que posteriormente analizara el script

Tabla 7. Descripción comando tcpdump DoS

2. Archivo dos.log

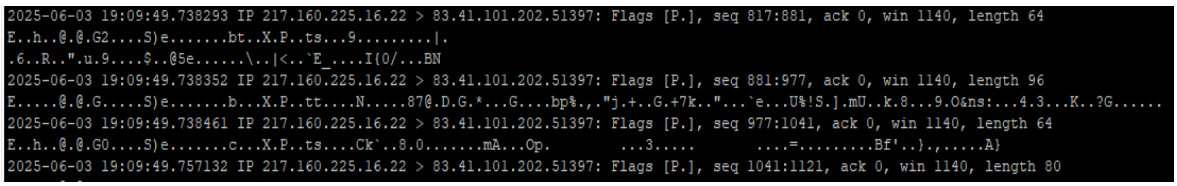

Figura 21. Registro generado comando tcpdump DoS

Utilizando la herramienta **tcpdump** se genera este archivo dos.log, cada línea del archivo contiene la siguiente información que nos interesa:

- Fecha y hora del paquete
- IP origen del posible atacante
- IP destino que representa la maquina victima
- El protocolo utilizado, para poder clasificar el tipo de ataque.

[37] American Standard Code for Information Interchange

3. *Inicializaciones DoS*

```
LOG_PATH = "/root/trafico/dos.log"
OUTPUT_PATH = "/root/registros/registresDoS.log"
THRESHOLD = 50
WINDOW = 15
ataques_totales = 0
ips_atacantes = set()
```

Figura 22. Configuración inicial detección DoS

- THRESHOLD: Corresponde el número mínimo de conexiones desde una misma IP.
- WINDOW: Representa el rango de tiempo.

4. *Expresión regular*

```
global ataques_totales, ips_atacantes
conexiones = defaultdict(list)
ataques_detectados = []
ip_victima = obtener_ip_local()
patron = re.compile(
    r'(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}\.\d+)\s+IP\s+(\d+\.\d+\.\d+\.\d+)(?:\.\d+)?\s+>\s+(\d+\.\d+\.\d+\.\d+)(?:\.\d+)?:'
)
```

Figura 23. Patrón detector DoS

Se define un patrón para poder extraer el timestamp y la dirección IP del posible atacante.

5. *Análisis de registros*

```
with open(LOG_PATH, 'r') as f:
    for linea in f:
        match = patron.search(linea)
        if match:
            timestamp_str, ip_origen, ip_destino = match.groups()
            timestamp = datetime.strptime(timestamp_str, '%Y-%m-%d %H:%M:%S.%f')
            conexiones[ip_origen].append(timestamp)

for ip, tiempos in conexiones.items():
    tiempos.sort()
    for i in range(len(tiempos)):
        ventana = [t for t in tiempos if 0 <= (t - tiempos[i]).total_seconds() <= WINDOW]
        if len(ventana) >= THRESHOLD:
            ataques_totales += 1
            ips_atacantes.add(ip)
            registro = {
                "fecha": tiempos[i].strftime('%Y-%m-%d %H:%M:%S'),
                "ip_victima": ip_victima,
                "ip_atacante": ip,
                "tipo": "DoS",
                "conexiones": len(ventana),
                "ventana_tiempo": WINDOW
            }
            ataques_detectados.append(registro)
            break

return ataques_detectados
```

Figura 24. Identificación ataque DoS

Primeramente, se aplica la expresión regular sobre cada línea para poder extraer la información relevante del ataque. Seguidamente, se identifica si desde una misma IP se realizan múltiples conexiones en un tiempo corto. Si el umbral es superado, se identifica como un posible ataque DoS.

6. Registro generado

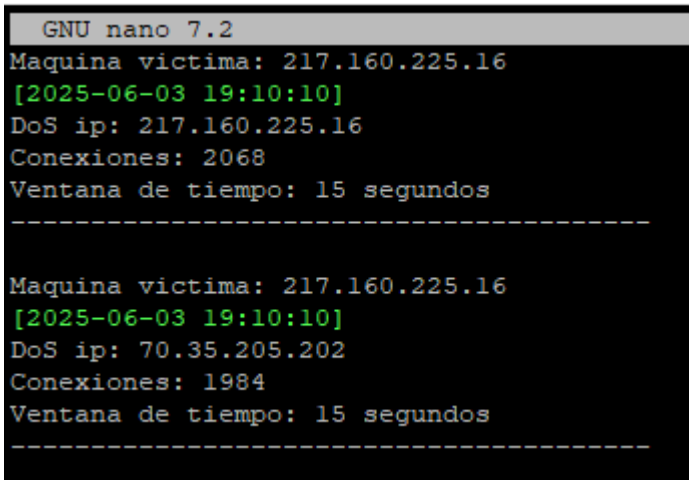

Figura 25. Log generado DoS

El log será enviado posteriormente al recolector que luego almacenará en la base de datos.

El código completo de este script está disponible en el Anexo 11.3

6.3 Detección de ataque DoS HTTP – Http_DoS.py

Este script está diseñado para detectar ataques DoS específicamente del protocolo HTTP, nos podemos encontrar con floods de peticiones GET.

El funcionamiento del Http_DoS.py es parecido al de DoS.py, los dos detectan muchas conexiones de una IP en poco tiempo, pero en este caso he decidido separar los códigos porque el patrón de detección en este caso es diferente, ahora no me baso en el protocolo IP, sino en encontrar métodos de petición HTTP en el contenido del paquete

1. Captura con tcpdump

```
def iniciar_tcpdump_http():
    comando = [
        "tcpdump", "-i", "ens6", "-nn",
        "tcp port 80", "-l"
    ]
    with open("/root/trafico/http_traffic.log", "w") as f:
        return subprocess.Popen(comando, stdout=f, stderr=subprocess.DEVNULL)
```

Figura 26. Captura de paquetes HTTP DoS

Para la detección de ataques HTTP DoS, capturamos el tráfico en tiempo real sobre el puerto 80 el cual corresponde al protocolo HTTP. Se logra con otra captura de tcpdump, permitiéndome registrar el tráfico de red que cumple con el filtro.

Parámetro	Descripción
-i ens6	Nos especifica la interfaz
-nn	Muestra los datos numéricos, evitando la resolución de nombres de host
-l	Permite que la salida se escriba en tiempo real
'tcp port 80	Filtro para poder capturar tráfico TCP que va dirigido al puerto 80
stdout=f	Redirige toda la salida al archivo http_traffic.log que posteriormente analizara el script

Tabla 8. Descripción comando tcpdump HTTP DoS

2. Archivo http_traffic.log

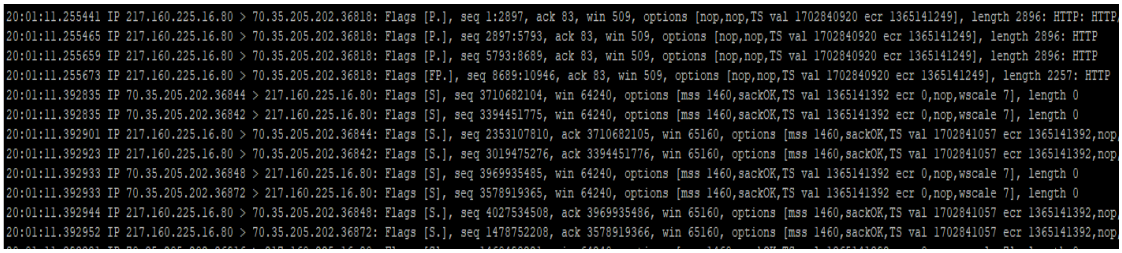

Figura 27. Registro generado comando tcpdump HTTP DoS

Este log es el resultante de la captura con tcpdump y es utilizado por el sistema IDS para poder detectar comportamientos anómalos de este tipo de ataque en específico.

3. Expresión regular HTTP DoS

```
def detectar_http_dos():
    global ataques_totales, ips_atacantes
    peticiones = defaultdict(list)
    ataques_detectados = []
    ip_victima = obtener_ip_local()
    patron = re.compile(
        r'^(\d{2}:\d{2}:\d{2}\.\d+)\s+IP\s+(\d+\.\d+\.\d+\.\d+)\.\d+\s+>\s+(\d+\.\d+\.\d+\.\d+)\.80:'
```

Figura 28. Patrón detector HTTP DoS

Para este caso, se define este patrón para extraer timestamp y la dirección IP del posible atacante si la línea contiene GET o POST.

4. Análisis de registros

```
with open(LOG_PATH, 'r') as f:
    for linea in f:
        match = patron.search(linea)
        if match:
            timestamp_str, ip_origen, ip_destino = match.groups()
            if ip_destino == ip_victima:
                try:
                    hoy = datetime.now().strftime('%Y-%m-%d')
                    timestamp = datetime.strptime(f"{hoy} {timestamp_str}", '%Y-%m-%d %H:%M:%S.%f')
                    peticiones[ip_origen].append(timestamp)
                except ValueError as e:
                    continue

for ip, tiempos in peticiones.items():
    tiempos.sort()
    for i in range(len(tiempos)):
        ventana = [t for t in tiempos if 0 <= (t - tiempos[i]).total_seconds() <= WINDOW]
        if len(ventana) >= THRESHOLD:
            ataques_totales += 1
            ips_atacantes.add(ip)
            registro = {
                "fecha": tiempos[i].strftime('%Y-%m-%d %H:%M:%S'),
                "ip_victima": ip_victima,
                "ip_atacante": ip,
                "tipo": "HTTP DoS",
                "peticiones": len(ventana),
                "ventana_tiempo": WINDOW,
                "detalles": {
                    "puerto_destino": 80                    }
            }
            ataques_detectados.append(registro)
            break

return ataques_detectados
```

Figura 29. Identificación ataque HTTP DoS

Podemos observar cómo los registros del archivo generado previamente. Primeramente, se aplica la expresión regular sobre cada línea para poder extraer la información relevante del ataque. Seguidamente, se identifica si desde una misma IP se realizan múltiples conexiones en un tiempo corto. Si el umbral es superado, se identifica como un posible ataque HTTP DoS.

5. Registro generado

```
Maquina victima: 217.160.225.16
[2025-06-03 19:10:16]
HTTP DoS ip: 70.35.205.202
Peticiones HTTP: 8175
Ventana de tiempo: 10 segundos
Puerto destino: 80
----------------------------------------

Maquina victima: 217.160.225.16
[2025-06-03 19:10:16]
HTTP DoS ip: 70.35.205.202
Peticiones HTTP: 8175
Ventana de tiempo: 10 segundos
Puerto destino: 80
----------------------------------------
```

Figura 30. Log generado HTTP DoS

El archivo que se genera contiene el tráfico de la red. Para este caso analizamos las líneas de los métodos HTTP.

El código completo de este script está disponible en el Anexo 11.4

6.4 Detección de inyecciones SQL – SQLi.py

Este script implementa un detector de posibles intentos de inyección SQL. Analiza el tráfico TCP con el objetivo de identificar patrones que son comunes en los payloads de las solicitudes HTTP.

1. Patrones SQLi comunes

```
class SQLiDetector:
    def __init__(self):
        self.patrones_sqli = [
            r"union.*select",
            r"1=['\"]?1['\"]?",
            r"sleep\(\d+\)",
            r"drop\s+table",
            r"select.*from",
            r"or\s+1=1",
            r"--\s*$"
        ]
```

Figura 31. Definición patrones SQLi

En esta lista de expresiones regulares se han definido las firmas de inyección SQL más comúnmente utilizadas en este tipo de ataque en concreto.

2. Captura y análisis del tráfico TCP

```
def guardar_log(self, ip_victima, ip_atacante, payload):
    global detecciones_totales, ips_atacantes, payloads_detectados
    detecciones_totales += 1
    ips_atacantes.add(ip_atacante)
    payloads_detectados.append(payload)
    timestamp = datetime.now().strftime("[%Y-%m-%d %H:%M:%S]")
    with open(LOG_FILE, 'a') as f:
        f.write(
            f"{timestamp}\n"
            f"    Maquina victima: {ip_victima}\n"
            f"    SQLi ip: {ip_atacante}\n"
            f"    Payload: {payload[:200]}"
            "----------------------------------------\n"
        )

def analizar_paquete(self, paquete):
    if paquete.haslayer(TCP) and paquete.haslayer(Raw):
        try:
            payload = paquete[Raw].load.decode('utf-8', errors='ignore')
            for patron in self.patrones_sqli:
                if re.search(patron, payload, re.IGNORECASE):
                    self.guardar_log(
                        ip_victima=paquete[IP].dst,
                        ip_atacante=paquete[IP].src,
                        payload=payload
                    )
                    break
        except:
            pass
```

Figura 32. Identificación ataque SQLi

Podemos observar cómo se filtran los datos Raw de TCP. Seguidamente se descodifica el contenido como un texto utf-8 para ser tratado como HTTP y busca en los patrones definidos anteriormente si hay alguna semejanza, en ese caso se considera un ataque y se guarda en el registro. Esta lógica implementa una detección por firma expuesta en el punto ***5.2.1.***

3. ***Registro generado***

```
[2025-06-03 19:10:36]
    Maquina victima: 217.160.225.16
    SQLi ip: 70.35.205.202
    Payload: GET /search?q=1' UNION SELECT 1,2,3-- HTTP/1.1
Host: 217.160.225.16

----------------------------------------
[2025-06-03 19:58:34]
    Maquina victima: 217.160.225.16
    SQLi ip: 70.35.205.202
    Payload: GET /search?q=1' UNION SELECT 1,2,3-- HTTP/1.1
Host: 217.160.225.16
```

Figura 33. Log generado SQLi

El código completo de este script está disponible en el Anexo 11.5

6.5 Envió de los registros – transferir.sh

Este script implementado en Bash tiene como objetivo enviar todos los archivos de log que generen en las maquinas victimas hacia la maquina recolectora, donde allí posteriormente los analizara y almacenara en la base de datos.

1. ***Configuración inicial***

```
directorioLocal="/root/registros"
patronArchivos="registres*"
recolector="root"
IpRecolector="82.223.165.171"
directorioRecolector="/root"
```

Figura 34. Configuración inicial transferencia

Se definen los diversos parámetros:

- Ruta de los archivos generados por las victimas (**/root**).
- El patrón que se sigue para los nombres de los registros: registres*
- El usuario y la dirección IPv4 de la maquina recolectora.
- El directorio donde serán enviados los diversos ficheros.

2. Bucle principal para él envío de archivos

```
for archivo in $directorioLocal/$patronArchivos; do
    if [ -f "$archivo" ]; then
        scp "$archivo" "$recolector@$IpRecolector:$directorioRecolector"
        if [ $? -eq 0 ]; then
            echo "Archivo $archivo transferido"
        fi
    fi
done
```

Figura 35. Lógica principal transferencia

La función de este código es recorrer todos los archivos que coincidan con el patrón visto anteriormente y verificando si son archivos válidos. Seguidamente se utiliza SCP con la finalidad de enviarlos de manera segura a la otra máquina.

El código completo de este script está disponible en el Anexo 11.6

6.6 Procesamiento y almacenamiento de los registros – db_ataques.py

El objetivo principal de esta script es leer todos los archivos de log que hayan generado las maquinas víctimas, extraer toda la información de cada ataque detectado y almacenar los datos en la base de datos **logs_ataques.db**. Además, se geolocaliza la IP del atacante.

1. Configuración inicial

```
DB_FILE = "logs_ataques.db"
LOG_FILES = "/root/registres*.log"
API_URL = "http://ip-api.com/json/{}?fields=country"
```

Figura 36. Configuración inicial base de datos

En la figura se puede apreciar cómo se definen los diversos parámetros, como son el nombre del archivo de la base de datos, el patrón a seguir de los logs a leer (registres*.log) , este patrón abarca todos los ataques que pueden llegar a ser detectados. Además, se define la URL[38] del servicio externo ip-api que nos permite geolocalizar la IP del atacante.

[38] Uniform Resource Locator

2. Geolocalización del atacante

```
def obtener_pais(ip):
    if ip not in pais_cache:
        try:
            response = requests.get(API_URL.format(ip), timeout=3)
            if response.status_code == 200:
                pais_cache[ip] = response.json().get('country', 'Desconocido')
            else:
                pais_cache[ip] = 'Desconocido'
        except:
            pais_cache[ip] = 'Desconocido'
    return pais_cache[ip]
```

Figura 37. Lógica geolocalización

En esta lógica se puede apreciar la petición a la URL previamente definida para obtener la ubicación de la IPv4 del atacante. Además, se puede observar cómo se utiliza un diccionario para poder almacenar las IPv4s ya geolocalizadas. Esta implementación supone una mejora del rendimiento en caso de tener mucho volumen de registros.

3. Creación de la tabla SQLite

```
def crear_tabla():
    conn = sqlite3.connect(DB_FILE)
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS ataques (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            fecha TEXT NOT NULL,
            ip_victima TEXT NOT NULL,
            ip_atacante TEXT NOT NULL,
            tipo TEXT,
            pais TEXT,
            detalle TEXT,
            UNIQUE(fecha, ip_victima, ip_atacante, tipo, detalle)
        )
    ''')

    cursor.execute('CREATE INDEX IF NOT EXISTS index_fecha ON ataques(fecha)')
    cursor.execute('CREATE INDEX IF NOT EXISTS index_tipo ON ataques(tipo)')
    conn.commit()
    conn.close()
```

Figura 38. Lógica creación BD

Nos aseguramos con la restricción `IF NOT EXISTS` que la tabla exista antes de insertar los logs. Además, con la restricción `UNIQUE` prevenimos los duplicados en el caso excepcional que se importen los mismos logs. Los índices nos permiten realizar búsquedas más rápidas en función del campo que se requiera.

4. Procesamiento de los bloques

```
def importar_logs():
    crear_tabla()
    conn = sqlite3.connect(DB_FILE)
    cursor = conn.cursor()
    for file in glob.glob(LOG_FILES):
        with open(file, 'r') as f:
            log_block = ""
            for line in f:
                if line.startswith("----------------------------------------"):
                    if log_block:
                        datos = procesar_log(log_block)
                        if datos:
                            pais = obtener_pais(datos['ip_atacante'])
                            cursor.execute('''
                                INSERT OR IGNORE INTO ataques
                                (fecha, ip_victima, ip_atacante, tipo, pais, detalle)
                                VALUES (?, ?, ?, ?, ?, ?)
                            ''', (
                                datos['fecha'],
                                datos['ip_victima'],
                                datos['ip_atacante'],
                                datos['tipo'],
                                pais,
                                datos['detalle']
                            ))
                        log_block = ""
                    else:
                        log_block += line
    conn.commit()
    conn.close()
    print(f"Ataques en {DB_FILE}")
```

Figura 39. Almacenamiento e importación logs

En este código se puede apreciar cómo se separan los bloques de los ataques por una línea de guiones. Cada bloque representa un ataque individual de los previamente mencionados. Además, en este código se observa la llamada a la función que extrae los datos de cada ataque y también la inserción de los ataques en la base de datos.

5. Extracción de la información de los bloques

```
def procesar_log(log_block):
    try:
        ip_victima = re.search(r'Maquina victima: (.*?)\n', log_block).group(1).strip()
        fecha = re.search(r'\[(.*?)\]', log_block).group(1)
        if 'SQLi ip:' in log_block:
            return {
                'fecha': fecha,
                'ip_victima': ip_victima,
                'ip_atacante': re.search(r'SQLi ip: (.*?)\n', log_block).group(1).strip(),
                'tipo': 'SQLi',
                'detalle': re.search(r'Payload: (.*?)\n', log_block).group(1)
            }
        elif 'Nmap ip:' in log_block:
            return {
                'fecha': fecha,
                'ip_victima': ip_victima,
                'ip_atacante': re.search(r'Nmap ip: (.*?)\n', log_block).group(1).strip(),
                'tipo': 'Nmap',
                'detalle': log_block.strip()
            }

        elif 'HTTP DoS ip:' in log_block:
            return {
                'fecha': fecha,
                'ip_victima': ip_victima,
                'ip_atacante': re.search(r'HTTP DoS ip: (.*?)\n', log_block).group(1).strip(),
                'tipo': 'HTTP DoS',
                'detalle': (
                      f"Peticiones HTTP: {re.search(r'Peticiones HTTP: (.*?)\n', log_block).group(1)} en "
                      f"{re.search(r'Ventana de tiempo: (.*?) segundos', log_block).group(1)}s, "
                      f"Puerto destino: {re.search(r'Puerto destino: (.*?)\n', log_block).group(1)}, "
                )
            }
        elif 'DoS ip:' in log_block:
            return {
                'fecha': fecha,
                'ip_victima': ip_victima,
                'ip_atacante': re.search(r'DoS ip: (.*?)\n', log_block).group(1).strip(),
                'tipo': 'DoS',
                'detalle': f"Conexiones: {re.search(r'Conexiones: (.*?)\n', log_block).group(1)} en {re.search(r'Ventana de tiempo: (.*?) segundos', log_block).group(1)}s"
            }
    except Exception as e:
        print(f"Error bloque: {e}")
    return None
```

Figura 40. Identificación del tipo de ataque

Seguidamente de la separación del archivo de los logs, se extraen los datos con mayor importancia de cada ataque, para obtener estos datos se utilizan diversas expresiones regulares.

El código completo de este script está disponible en el Anexo 11.7

6.7 Implementación del servidor web, mapa y gráficos – app.py

Este script tiene como función principal implementar un servidor web utilizando Flash, este servidor web nos proporciona una interfaz gráfica con el listado de todos los ataques detectados y además muestra un mapa interactivo generado con Folium, el cual nos permite ver la geolocalización de los atacantes.

1. Inicializacion del servidor Flask

```
app = Flask(__name__,
            template_folder='/root/web/html',
            static_folder='/root/web/css')

MAPS_DIR = '/root/web/html'
STATIC_DIR = '/root/web/css'
os.makedirs(MAPS_DIR, exist_ok=True)
os.makedirs(os.path.join(STATIC_DIR, 'css'), exist_ok=True)
```

Figura 41. Configuración servidor Flask

Se puede apreciar como en el código se indican las dos diversas carpetas, una para la carpeta de la plantilla HTML y otra para los elementos estáticos como CSS[39] e iconos.

2. *Funcionalidad principal*

```
@app.route('/')
def inicio():
    conn = sqlite3.connect('/root/logs_ataques.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM ataques ORDER BY fecha DESC")
    ataques = cursor.fetchall()
    conn.close()
    generar_mapa(ataques)
    tipos = [a[4] for a in ataques]
    conteo_tipos = Counter(tipos)

    fechas = [datetime.strptime(a[1], "%Y-%m-%d %H:%M:%S").date().isoformat() for a in ataques]
    conteo_fechas = Counter(fechas)

    return render_template(
        'index.html',
        ataques=ataques,
        tipos_data=json.dumps(conteo_tipos),
        fechas_data=json.dumps(conteo_fechas)
    )
```

Figura 42. Lógica principal servidor web

Se implementa la lógica de la conexión a la base de datos y se recuperan todos los ataques producidos a esta. Además, se llama a la función ***generar_mapa()*** para actualizar el mapa dinámico con los nuevos logs que reciba la maquina recolectora. También muestra la plantilla HTML con la tabla de los diversos ataques.

A partir de la consulta SQL que extrae los registros de ataques, se preparan los datos para posteriormente generar los gráficos. Se generan dos conjuntos de datos:

- Contador por tipo de ataque: Utilizando ***Counter()*** de Python, se hace un conteo de cuantas veces aparece cada tipo de ataque.
- Contador por fecha: Se hace un conteo del campo fecha de la base de datos.

Los datos son convertidos a JSON[40] con la función ***json.dumps()*** y se envían al archivo de la interfaz web index.html

39 Cascading Style Sheets

40 JavaScript Object Notation

3. Marcador geográfico en el mapa

Para cada IP única del atacante se añade un marcador rojo. Además, se puede apreciar un ***pop-up*** con la siguiente información:

- La IPv4 del atacante
- Tipo de ataque realizado
- Ubicación y el ISP[41] siempre y cuando este sea conocido públicamente.

```python
def generar_mapa(ataques):
    mapa = folium.Map(location=[40, -3], zoom_start=2)

    for ataque in ataques:
        ip = ataque[3]
        tipo = ataque[4]

        try:
            response = requests.get(f"http://ip-api.com/json/{ip}", timeout=3)
            data = response.json()
            if data['status'] == 'success':
                lat, lon = data['lat'], data['lon']
                ciudad = data.get('city', 'Desconocida')
                pais = data.get('country', 'Desconocido')
                org = data.get('org', 'Desconocida')
            else:
                lat, lon = 40, -3
                ciudad = pais = org = "Desconocido"
        except Exception as e:
            print(f"Error geolocalizando {ip}: {str(e)}")
            lat, lon = 40, -3
            ciudad = pais = org = "Desconocido"

        popup_content = f"""
        <b>IP:</b> {ip}<br>
        <b>Tipo:</b> {tipo}<br>
        <b>Ubicación:</b> {ciudad}, {pais}<br>
        <b>Proveedor:</b> {org}
        """

        folium.Marker(
            location=[lat, lon],
            popup=folium.Popup(popup_content, max_width=300),
            icon=folium.Icon(color='red', icon='warning-sign')
        ).add_to(mapa)

    mapa.save(os.path.join(MAPS_DIR, 'mapa.html'))
```

Figura 43. Lógica generación del mapa

41 Internet service provider

4. Generación archivo CSV

```python
@app.route('/descargar')
def descargar_csv():
    conn = sqlite3.connect('/root/logs_ataques.db')
    cursor = conn.cursor()
    cursor.execute("SELECT fecha, tipo, ip_atacante, pais FROM ataques ORDER BY fecha DESC")
    ataques = cursor.fetchall()
    conn.close()

    salida = StringIO()
    writer = csv.writer(salida)
    writer.writerow(['Fecha', 'Tipo', 'IP Atacante', 'País'])

    for ataque in ataques:
        writer.writerow(ataque)

    salida.seek(0)
    return Response(
        salida.getvalue(),
        mimetype='text/csv',
        headers={'Content-Disposition': 'attachment; filename=reportes_ataques.csv'}
    )
```

Figura 44. Lógica descarga archivo CSV

Se ha añadido la ruta **/descargar** que realiza una consulta SQL para extraer todos los ataques de la base de datos y se genera un archivo CSV en memoria con los siguientes campos.

- Fecha de la detección del ataque
- Tipo de ataque detectado
- IPv4 del atacante
- País de origen

```
root@ubuntu:~# python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://82.223.165.171:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 444-377-640
```

Figura 45. Ejecución servidor Flask

El código completo de este script está disponible en el Anexo 11.8

6.8 Interfaz web HTML – index.html

La función principal este archivo es definir una estructura visual para el sistema IDS previamente explicado que el usuario observara al acceder al servidor web. En esta interfaz se muestran todos los ataques detectados y se pueden visualizar en un mapa interactivo. Además, se presentan los diversos gráficos.

1. Tabla de ataques

```
<div class="dashboard-panel">
        <div class="card">
            <div class="card-header">Ataques detectados</div>
            <div class="card-body">
                <a href="/descargar" class="btn-descarga">Descargar reportes CSV</a>
                <div class="tabla-scroll">
                  <table>
                    <thead>
                        <tr>
                            <th>Fecha</th>
                            <th>Tipo</th>
                            <th>IP Atacante</th>
                            <th>País</th>
                        </tr>
                    </thead>
                    <tbody>
                        {% for ataque in ataques %}
                        <tr>
                            <td>{{ ataque[1] }}</td>
                            <td><span class="badge badge-danger">{{ ataque[4] }}</span></td>
                            <td>{{ ataque[3] }}</td>
                            <td>{{ ataque[5] | truncate(30) }}</td>
                        </tr>
                        {% endfor %}
                    </tbody>
                  </table>
                </div>
            </div>
        </div>
    </div>
```

Figura 46. Definición tabla HTML

En esta sección se puede apreciar el uso de la sintaxis de la plantilla **Jinja2** para ir iterando sobre la lista de ataques y se observa la definición del botón para descargar los reportes CSV. Además, se muestra la siguiente información:

- Fecha de la detección del ataque
- Tipo de ataque detectado
- IPv4 del atacante
- País de origen

2. *Inserción del mapa y gráficos*

```
<div class="dashboard-panel" id="mapa-panel">
    <div class="card">
        <div class="card-header">
            Mapa de Ataques en Tiempo Real
            <button onclick="toggleMapa()" class="btn-toggle-mapa">Ampliar</button>
        </div>
        <div class="card-body">
            <iframe id="mapa-frame" src="/mapa"></iframe>
        </div>
    </div>
</div>


    <div class="dashboard-panel">
        <div class="card">
            <div class="card-header">Estadísticas de Ataques</div>
            <div class="card-body">
                <div class="chart-container">
                    <canvas id="graficoTipos"></canvas>
                </div>
                <div class="chart-container">
                    <canvas id="graficoFechas"></canvas>
                </div>
            </div>
        </div>
    </div>
```

Figura 47. Lógica de inserción mapa y gráficos

Se inserta el archivo **mapa.html** generado por Folium con el código app.py previamente mencionado y se integran los gráficos generados con Chart.js. Además, cabe recalcar el archivo styles.css que proporciona a la interfaz web un diseño más profesional.

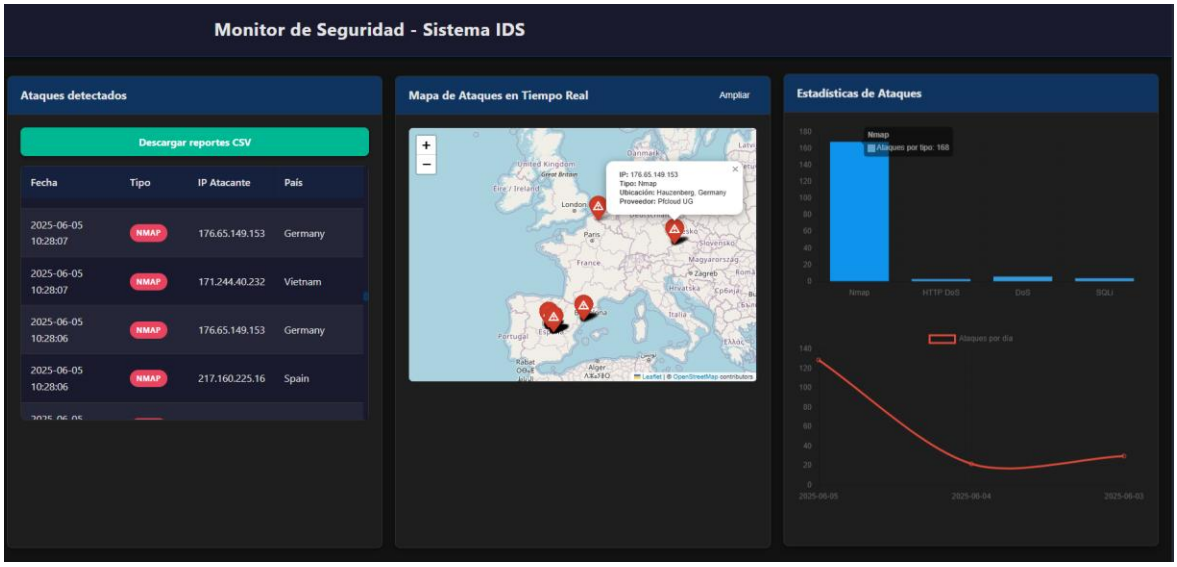

Figura 48. Representación dashboard

El monitor gráfico muestra el resultado del análisis forense en red que permite rastrear el origen del atacante y entender la línea de tiempo en que se produce un ciberataque, crucial para la respuesta a incidentes.

El código completo de este script está disponible en el Anexo 11.9

7 Centralización e integración del sistema IDS

Para realmente simular el comportamiento real de un sistema de detección de intrusos, se ha creado un script que actúa como "núcleo" del sistema llamado main_ids.py y tiene diversas funciones.

- Ejecuta los scripts de detección de ataques
- Captura el tráfico utilizando tcpdump
- Realiza la transferencia automática de los registros a la maquina recolectora.

Además, se ha estructurado el sistema para tener una distribución de los archivos más visual para el usuario, a la vez que esta estructura permite una futura escalabilidad del sistema con la implementación de más ataques.

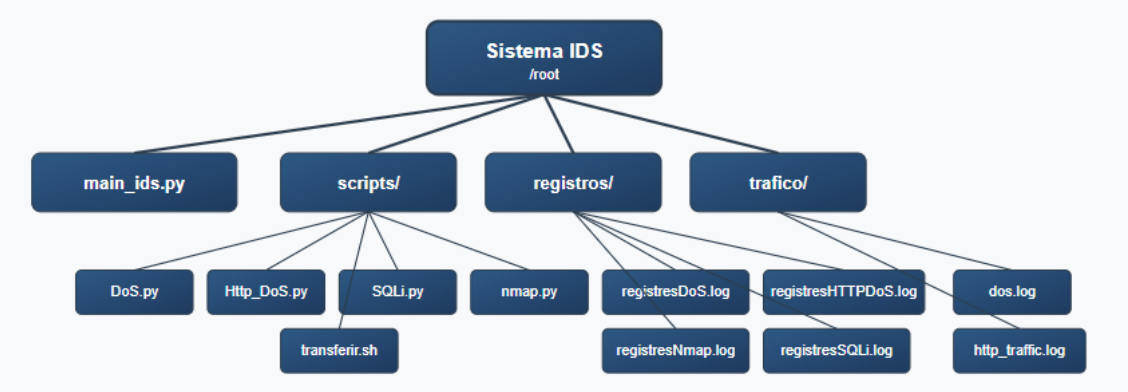

Figura 49. Estructura Sistema IDS

Lógica general – main_ids.py

```
def iniciar_tcpdump():
    comando = [
        "tcpdump", "-tttt", "-n", "-A", "-i", "ens6",
        "(icmp or udp or tcp)", "-l"
    ]
    with open("/root/trafico/dos.log", "w") as f:
        return subprocess.Popen(comando, stdout=f, stderr=subprocess.DEVNULL)

def iniciar_tcpdump_http():
    comando = [
        "tcpdump", "-i", "ens6", "-nn",
        "tcp port 80", "-l"
    ]
    with open("/root/trafico/http_traffic.log", "w") as f:
        return subprocess.Popen(comando, stdout=f, stderr=subprocess.DEVNULL)
```

Figura 50. Ejecución captura del tráfico

Se inicializa el IDS con la ejecución de los tcpdump, que capturaran el tráfico, para posteriormente hacer el análisis de los logs y detectar posibles comportamientos que me indicaran si la maquina está siendo atacada o no.

```
def ejecutar_sql_ids():
    print("Detector SQLi")
    subprocess.run(["python3", "/root/scripts/SQLi.py"])

def ejecutar_nmap_ids():
    print("Detector Nmap - SYN - FIN - NULL - XMAS")
    subprocess.run(["python3", "/root/scripts/nmap.py"])

def ejecutar_dos_detector():
    print("Detector TCP, UDP, ICMP DoS")
    while not detener_ejecucion:
        subprocess.run(["python3", "/root/scripts/DoS.py"])
        time.sleep(10)

def ejecutar_http_dos():
    print("Detector HTTP DoS")
    while not detener_ejecucion:
        subprocess.run(["python3", "/root/scripts/Http_DoS.py"])
        time.sleep(10)

def transferir_registros():
    print("Transfiriendo archivos")
    subprocess.run(["/bin/bash", "/root/scripts/transferir.sh"])
```

Figura 51. Definiciones de los detectores

Se definen distintas funciones que permiten ejecutar paralelamente los scripts desarrollados para la detección de los ciberataques. La ejecución de cada script se realiza utilizando la biblioteca **subprocess,** permitiéndome lanzar procesos externos desde Python. En el caso de las funciones de tipo DoS (TCP/UDP/ICMP y HTTP), he establecido un bucle que las ejecuta cada 10 segundos, garantizando que se ejecuten continuamente. Además, se incluye la función que automatiza la transferencia de registros.

```
if __name__ == "__main__":
    signal.signal(signal.SIGINT, handler_shutdown)
    signal.signal(signal.SIGTERM, handler_shutdown)
    print("[+] IDS iniciado")
    tcpdump_proc = iniciar_tcpdump()
    http_tcpdump_proc = iniciar_tcpdump_http()
    time.sleep(2)

    t1 = threading.Thread(target=ejecutar_sql_ids)
    t2 = threading.Thread(target=ejecutar_nmap_ids)
    t3 = threading.Thread(target=ejecutar_dos_detector)
    t4 = threading.Thread(target=ejecutar_http_dos)

    t1.start()
    t2.start()
    t3.start()
    t4.start()
    try:
        t1.join()
        t2.join()
        t3.join()
        t4.join()
    except KeyboardInterrupt:
        pass
    finally:
        transferir_registros()
        tcpdump_proc.terminate()
        http_tcpdump_proc.terminate()
        tcpdump_proc.wait()
        http_tcpdump_proc.wait()
        print("[+] IDS detenido")
        os._exit(0)
```

Figura 52. Paralelización de las funciones

Finalmente, se paralelizan las llamadas a los archivos de detección en hilos independientes utilizando ***threading.Thread*.** Cuando se interrumpe el sistema, se detiene la captura del tráfico y se transfieren los archivos a la maquina recolectora.

Al realizar esta integración pasamos de tener detectores de ataques individuales a una arquitectura centralizada, acercándonos a IDS profesionales como **Snort**.

El código completo de este script está disponible en el Anexo 11.13

7.1 Transferencia de registros automáticos

Como se ha comentado anteriormente, se utiliza el script **Bash** transferir.sh con el protocolo SCP para hacer la transferencia de los archivos. Durante la ejecución del sistema me encontré con la "problemática" de tener que introducir la contraseña de la máquina virtual del recolector para cada archivo de registros que quisiera enviar.

Para solucionar y garantizar que el sistema IDS se completamente autónomo, he desarrollado una función adicional que me permite transferir los registros de manera automática. Para lograr esto, he utilizado la autenticación SSH basada en claves.

1. ***Generación de las claves en la máquina de la víctima:***

```
root@ubuntu:~# ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:r3fqW/Dh6FPieLm9s28CYbkTH/ZU/CpSNUSAmAH8ULs root@ubuntu
The key's randomart image is:
+---[RSA 4096]----+
|     ..oo+ ..+o. |
|      o o..   o o|
|       o.  . . o.|
|        ..= + . .|
|        E..B.+ . |
|         .*=+.o  |
|         ooB+.   |
|        .o*o+ .  |
|        .+**+B.  |
+----[SHA256]-----+
```

Figura 53. Generaciones de las claves SSH

Esta comanda genera una pareja de claves RSA[42] de 4096 bits y se almacenan en la carpeta .ssh. El tamaño de la clave ayuda en caso de ataques de fuerza bruta proporcionando mayor seguridad.

```
root@ubuntu:~# cd .ssh/
root@ubuntu:~/.ssh# ls -l
total 16
-rw------- 1 root root    0 Feb 25 15:47 authorized_keys
-rw------- 1 root root 3369 May 28 12:54 id_rsa
-rw-r--r-- 1 root root  737 May 28 12:54 id_rsa.pub
-rw------- 1 root root  978 Mar 31 17:46 known_hosts
-rw-r--r-- 1 root root  142 Mar 31 17:45 known_hosts.old
```

Figura 54. Estructura carpeta SSH

[42] Rivest–Shamir–Adleman

Archivo	Descripción
authorized_keys	Claves públicas que están autorizadas para hacer la conexión sin contraseña
id_rsa	Clave privada
id_rsa.pub	Clave publica que se comparte
known_hosts	Lista de servidores SSH que se conocen

Tabla 9. Definición estructura carpeta SSH

2. Copia de la clave publica

```
root@ubuntu:~/scripts# ssh-copy-id root@82.223.165.171
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@82.223.165.171's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@82.223.165.171'"
and check to make sure that only the key(s) you wanted were added.
```

Figura 55. Instalación clave SSH en maquina recolectora

Esta comanda copia la clave publica que se ha generado en la maquina victima al archivo **autorized_keys** de la maquina recolectora. Haciendo estos pasos conseguimos automatizar el proceso de transferencia de registros simulando un comportamiento autónomo por parte del sistema.

8 Simulación de ataques

Para validar el correcto funcionamiento del sistema IDS, se han diseñado y ejecutado diferentes tipos de ataques simulados en este entorno controlado. Estas simulaciones se han realizado desde la maquina atacante.

8.1 Escaneo de puertos TCP SYN

El primer ataque es el ataque de escaneo de puertos utilizando la técnica SYN Scan. El ataque se ha simulado desde la maquina atacante utilizando la herramienta Nmap, con el objetivo de comprobar el correcto funcionamiento del sistema IDS.

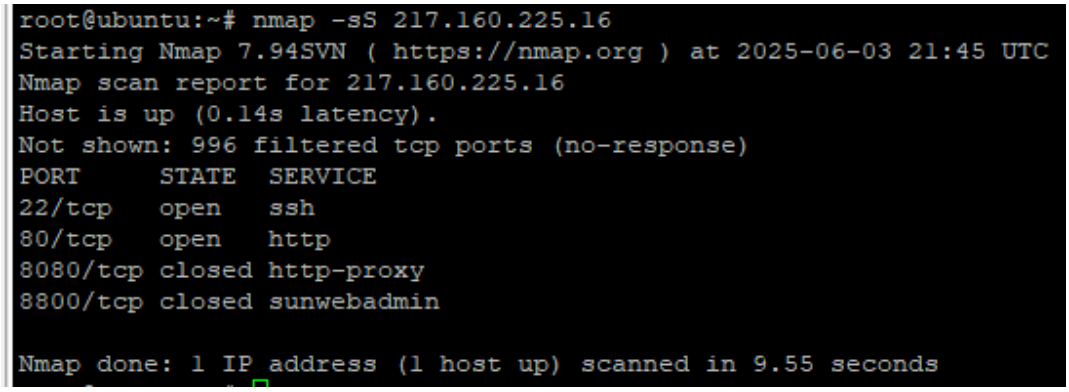

Figura 56. Escaneo de puertos SYN

Parámetro	Descripción
-sS	Hace un escaneo TCP SYN. Envía paquetes con la bandera SYN sin llegar completar el handshake
217.160.225.16	IP de la maquina victima

Tabla 10. Descripción comando Nmap SYN

Funcionamiento de un SYN Scan

Este tipo de escaneo se basa principalmente en el **3-way handshake**:

1. El atacante envía el paquete TCP con el flag SYN activado al puerto de destino.
2. Si ese puerto está abierto, la victima responde con un SYN-ACK.
3. El atacante en lugar de completar el handshake, no responde, por tanto, evita establecer una conexión completa.

Aunque no dañe directamente al sistema, este tipo de ciberataque se puede considerar como una fase de reconocimiento que permite al atacante enumerar servicios o posibles vulnerabilidades. Esto habilita a que se puedan planificar ataques más avanzados como exploits.

8.2 DoS TCP Flood

Este script simula un ataque DoS enviando repetidamente conexiones TCP completas a un puerto en específico, en este caso para la realización de las pruebas contra el puerto 22 (SSH) en las maquinas víctimas.

Este tipo de ataque tiene como objetivo agotar los recursos de red del sistema objetivo mediante el envío de conexiones TCP en un periodo de tiempo muy corto.

```
import socket
import time

IP_VICTIMA = "217.160.225.16"
PUERTO =22
REPETICIONES = 1000

for _ in range(REPETICIONES):
    try:
        s = socket.socket()
        s.settimeout(0.5)
        s.connect((IP_VICTIMA, PUERTO))
    except:
        pass
    finally:
        s.close()
```

Figura 57. Script ataque DoS

El código completo de este script está disponible en el Anexo 11.11

8.3 DoS ICMP Flood

Simulado un ataque DoS haciendo un envío masivo de paquetes ICMP del tipo echo request, con el objetivo de saturar el ancho de banda de la víctima. Se utiliza la herramienta hping3 que tiene la finalidad de crear y analizar paquetes personalizados de red, con el objetivo de comprobar el correcto funcionamiento del sistema IDS.

```
root@ubuntu:~# hping3 -1 --flood -d 1400 217.160.225.16
HPING 217.160.225.16 (ens6 217.160.225.16): icmp mode set, 28 headers + 1400 data bytes
hping in flood mode, no replies will be shown
^C
--- 217.160.225.16 hping statistic ---
346543 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Figura 58. Ejecución ICMP Flood

Parámetro	Descripción
-1	Se activa el modo ICMP
--flood	Se envían los paquetes tan rápido como pueda
-d 1400	Se especifica el tamaño del payload en bytes
217.160.225.16	IP de la maquina victima

Tabla 11. Descripción comando Hping3 ICMP

8.4 DoS UDP Flood

Simulado un ataque DoS haciendo un envío masivo de paquetes UDP, este caso apunta al puerto 53 que es el comúnmente utilizado para el servicio DNS. Se utiliza la herramienta hping3 que tiene la finalidad de crear y analizar paquetes personalizados de red, con el objetivo de comprobar el correcto funcionamiento del sistema IDS.

```
root@ubuntu:~# hping3 -2 --flood -p 53 -d 1400 217.160.225.16
HPING 217.160.225.16 (ens6 217.160.225.16): udp mode set, 28 headers + 1400 data bytes
hping in flood mode, no replies will be shown
^C
--- 217.160.225.16 hping statistic ---
106088 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Figura 59. Ejecución UDP Flood

Parámetro	Descripción
-2	Se activa el modo UDP
--flood	Se envían los paquetes tan rápido como pueda
-p 53	Puerto destino 53
-d 1400	Se especifica el tamaño del payload en bytes
217.160.225.16	IP de la maquina victima

Tabla 12. Descripción comando Hping3 UDP

8.5 DoS HTTP Flood

El ataque HTTP Flood consiste en enviar una gran cantidad de peticiones HTTP hacia un servidor web, con la finalidad de saturar los recursos, volviéndolo extremadamente lento. A diferencia de los otros ataques DoS previamente mencionados, este se realiza en el nivel de aplicación que corresponde a L7 del modelo OSI.

Para la simulación de este ataque se ha utilizado la herramienta **ApacheBench**, que está diseñada principalmente para realizar pruebas de rendimiento a servidores HTTP. Para realmente poder realizar la detección, se han utilizado valores muy elevados contra el servidor web de la maquina victima que posteriormente comentare.

```
root@ubuntu:~# ab -n 5000 -c 1000 http://217.160.225.16/
This is ApacheBench, Version 2.3 <$Revision: 1903618 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 217.160.225.16 (be patient)
Completed 500 requests
Completed 1000 requests
Completed 1500 requests
Completed 2000 requests
Completed 2500 requests
Completed 3000 requests
Completed 3500 requests
Completed 4000 requests
Completed 4500 requests
Completed 5000 requests
Finished 5000 requests


Server Software:        Apache/2.4.58
Server Hostname:        217.160.225.16
Server Port:            80

Document Path:          /
Document Length:        10671 bytes

Concurrency Level:      1000
Time taken for tests:   3.861 seconds
Complete requests:      5000
Failed requests:        0
Total transferred:      54725000 bytes
HTML transferred:       53355000 bytes
Requests per second:    1295.05 [#/sec] (mean)
Time per request:       772.172 [ms] (mean)
Time per request:       0.772 [ms] (mean, across all concurrent requests)
Transfer rate:          13842.09 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:      137  191 161.0    161    2293
Processing:   138  266 275.0    197    2291
Waiting:      138  229 175.3    188    2291
Total:        275  457 317.7    360    2487

Percentage of the requests served within a certain time (ms)
  50%    360
  66%    369
  75%    378
  80%    392
  90%    654
  95%   1286
  98%   1820
  99%   1976
 100%   2487 (longest request)
```

Figura 60. Ejecución HTTP Flood

Parámetro	Descripción
-n 5000	Representa el número total de peticiones que se van a realizar
-c 1000	Número de peticiones simultaneas
http://217.160.225.16/	URL del servidor

Tabla 13. Descripción comando ApacheBench

8.6 Escaneo Stealth TCP FIN

Simulado un ataque Stealth de escaneo de puertos utilizando paquetes TCP con el flag FIN activado, este tipo de ataque a diferencia de SYN Scan se utiliza para intentar evadir firewalls y nos es útil para validar si nuestro sistema IDS también detecta otro tipo de escaneos.

El ataque se ha simulado desde la maquina atacante utilizando la herramienta **Nmap**, con el objetivo de comprobar el correcto funcionamiento del sistema IDS.

```
root@ubuntu:~# nmap -sF 217.160.225.16
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-03 21:44 UTC
Nmap scan report for 217.160.225.16
Host is up (0.14s latency).
All 1000 scanned ports on 217.160.225.16 are in ignored states.
Not shown: 1000 open|filtered tcp ports (no-response)

Nmap done: 1 IP address (1 host up) scanned in 11.94 seconds
```

Figura 61. Escaneo de puertos FIN

Parámetro	Descripción
-sF	Activa el flag FIN
217.160.225.16	IP de la maquina victima

Tabla 14. Descripción comando Nmap FIN

Aunque no dañe directamente al sistema, este tipo de ciberataque se puede considerar como una fase de reconocimiento que permite al atacante enumerar servicios o posibles vulnerabilidades. Esto habilita a que se puedan planificar ataques más avanzados como exploits. A diferencia del TCP SYN Scan, permite hacer el reconocimiento sin levantar tantas sospechas por ser más difícil de registrar en logs si no hay un IDS.

8.7 Escaneo Stealth TCP XMAS

Se ha realizado un escaneo XMAS o Xmas Tree, es una técnica de escaneo de puertos considerada como stealth. El nombre lo recibe por el hecho de que activa múltiples flags del protocolo TCP al mismo tiempo: FIN, URG y PSH, iluminando el paquete como un árbol de Navidad.

El ataque se ha simulado desde la maquina atacante utilizando la herramienta **Nmap**, con el objetivo de comprobar el correcto funcionamiento del sistema IDS.

```
root@ubuntu:~# nmap -sX 217.160.225.16
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-04 11:39 UTC
Nmap scan report for 217.160.225.16
Host is up (0.13s latency).
All 1000 scanned ports on 217.160.225.16 are in ignored states.
Not shown: 1000 open|filtered tcp ports (no-response)
```

Figura 62. Escaneo de puertos XMAS

Parámetro	Descripción
-sX	Activa los flags FIN, URG, PUSH
217.160.225.16	IP de la maquina victima

Tabla 15. Descripción comando Nmap XMAS

8.8 Escaneo TCP NULL

Se ha realizado la simulación del ataque NULL Scan, es una técnica de escaneo de puertos en la que se envían paquetes TCP, pero sin ningún flag activado. Es un ataque más avanzado porque intenta determinar el estado de los puertos, pero sin iniciar una conexión.

El ataque se ha simulado desde la maquina atacante utilizando la herramienta **Nmap**, con el objetivo de comprobar el correcto funcionamiento del sistema IDS.

```
root@ubuntu:~# nmap -sN 217.160.225.16
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-04 12:04 UTC
Nmap scan report for 217.160.225.16
Host is up (0.13s latency).
All 1000 scanned ports on 217.160.225.16 are in ignored states.
Not shown: 1000 open|filtered tcp ports (no-response)

Nmap done: 1 IP address (1 host up) scanned in 10.96 seconds
```

Figura 63. Escaneo de puertos NULL

Parámetro	Descripción
-sN	No se activa ninguna bandera
217.160.225.16	IP de la maquina victima

Tabla 16. Descripción comando Nmap NULL

8.9 Escaneo TCP ACK

Simulado un ataque de escaneo de puertos utilizando paquetes TCP, pero en este caso con el flag ACK activado, realmente esta técnica no busca detectar los puertos abiertos de la víctima, sino pretende mapear el comportamiento del firewall para comprobar si los paquetes ACK serán bloqueados o permitidos, lo que permite revelar reglas de filtrado o rutas permitidas.

```
root@ubuntu:~# nmap -sA 217.160.225.16
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-04 12:07 UTC
Nmap scan report for 217.160.225.16
Host is up (0.13s latency).
Not shown: 996 filtered tcp ports (no-response)
PORT     STATE      SERVICE
22/tcp   unfiltered ssh
80/tcp   unfiltered http
8080/tcp unfiltered http-proxy
8800/tcp unfiltered sunwebadmin
```

Figura 64. Escaneo de puertos ACK

Parámetro	Descripción
-sA	Activa el flag ACK
217.160.225.16	IP de la maquina victima

Tabla 17. Descripción comando Nmap ACK

Se puede observar el estado de **unfiltered**, por tanto, el puerto de la maquina victima ha respondido con un RST, por tanto, el paquete ACK ha pasado, lo que significa que el puerto no está protegido por un firewall.

8.10 Ataque SQLi

Se ha realizado la simulación de un ataque SQLi, este ataque simula el envio de peticiones HTTP maliciosas a un servidor web. Estas peticiones contienen patrones que comúnmente se utilizan en ataques de inyección SQL, como `'OR '1'='1, UNION SELECT`, con la finalidad de manipular las consultas a base de datos.

```
RECOLECTOR_IP ="217.160.225.16"
PORT = 80

payloads = [
    "GET /login.php?user=admin' OR '1'='1'-- HTTP/1.1\r\nHost: {}\r\n\r\n",
    "POST /check_login.php HTTP/1.1\r\nHost: {}\r\nContent-Length: 28\r\n\r\nusername=admin'--&password=1234",
    "GET /search?q=1' UNION SELECT 1,2,3-- HTTP/1.1\r\nHost: {}\r\n\r\n"
]

def enviar_ataques():
    for i, payload in enumerate(payloads):
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.connect((RECOLECTOR_IP, PORT))
            s.sendall(payload.format(RECOLECTOR_IP).encode())
            print(f"Payload {i+1} enviado")
            s.close()
        except Exception as e:
            print(f"Error: {str(e)}")
        time.sleep(random.uniform(0.5, 2))

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

Figura 65. Script ataque inyección SQL

La lógica del script es crear conexiones TCP hacia el servidor de la maquina victima en el puerto 80. Mediante ***sendall***, se transmiten strings que están diseñados para imitar peticiones HHTP con inyecciones SQL en los parámetros.

El servidor web que se comenta de la máquina víctima es un servidor web Apache específicamente diseñado para hacer las simulaciones del ataque HTTP Flood, se comenta su implementación y funcionalidad dentro del proyecto en el punto ***12***.

El código completo de este script está disponible en el Anexo 11.12

9 Evaluación

Este apartado tiene como objetivo validar el correcto funcionamiento del sistema IDS frente a diversos escenarios, para poder demostrar su capacidad de detección, visualización y registro de los ataques en tiempo real.

Primeramente, se han ejecutados los códigos de detección del sistema IDS mediante el script principal main.py previamente mencionado.

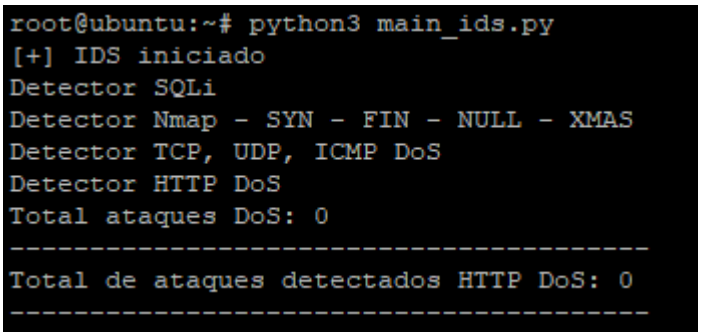

Figura 66. Inicio sistema IDS

Seguidamente, se ha procedido a ejecutar de manera consecutiva todos los ataques diseñados en este entorno controlado. Desde la maquina atacante se han ejecutado los escaneos de puertos, ataques de denegación de servicios e inyecciones de SQL. Por otro lado, las maquinas victimas capturan y analizan el tráfico.

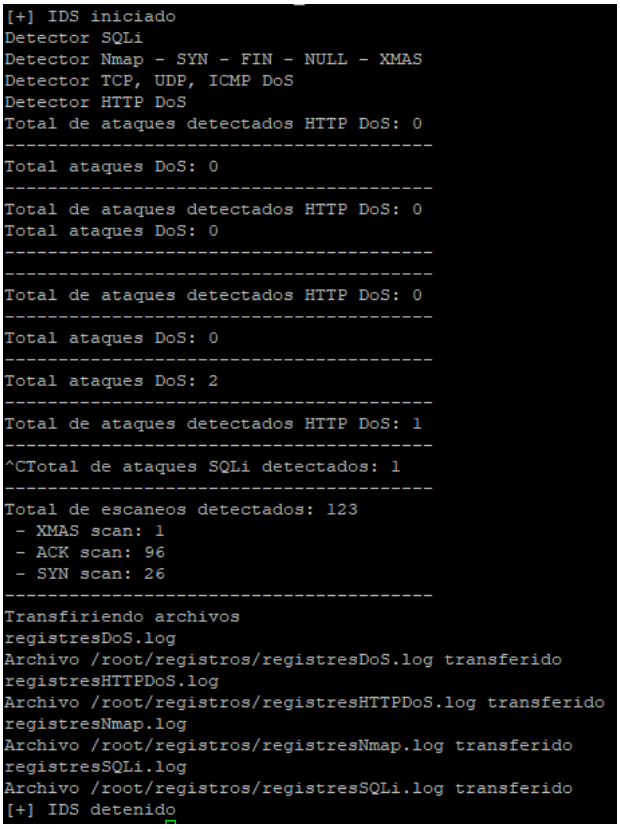

Figura 67. Ejecución del sistema IDS

9.1 Consideraciones sobre ataques ICMP y UDP Flood

Durante la simulación de ataques DoS basados en los protocolos ICMP y UDP, he observado un comportamiento anómalo. A pesar de la ejecución de los ataques con la herramienta hping3, el IDS no era capaz de detectar todos los paquetes enviados hacia la máquina. Con la finalidad de encontrar el origen de casuística, he realizado diversas comprobaciones:

```
root@ubuntu:~# sudo ufw status
Status: inactive
root@ubuntu:~# sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Figura 68. Resultado generado de la comprobación

Comando	Finalidad	Resultado
sudo ufw status	Verificamos el estado del firewall	inactive
sudo iptables -L	Visualizamos las reglas del iptables	Política ACCET

Tabla 18. Verificación firewall y reglas de iptables

Se aprecia como las maquinas víctimas no tienen los firewalls activados. Además, cuando se realizan las simulaciones de ICMP y UDP Flood se ha obtenido un 100% de ***packet loss***.

```
root@ubuntu:~# hping3 -1 --flood -d 1400 217.160.225.16
HPING 217.160.225.16 (ens6 217.160.225.16): icmp mode set, 28 headers + 1400 data bytes
hping in flood mode, no replies will be shown
^X^C
--- 217.160.225.16 hping statistic ---
142115 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Figura 69. Packet loss ICMP Flood

Este comportamiento me sugiere que el tráfico de ICMP y UDP está siendo filtrado directamente por el proveedor de la infraestructura, que en mi caso es Piensa Solutions.

9.2 Consideraciones sobre ataque HTTP Flood

En comparación a otros ataques de denegación de servicios que se basan en patrones TCP, el ataque HTTP Flood requiere que la víctima disponga de un servidor web respondiendo a las solicitudes HTTP en el puerto 80. Si tenemos en cuenta el diseño del sistema IDS, las maquinas víctimas no disponen de un servidor web, ya que el único servidor web se encuentra en la maquina recolectora. Para poder simular de forma realista el ataque DoS a nivel de aplicación y a la vez no mezclar componentes de las diferentes maquinas manteniendo así las funcionalidades únicas, he tomado la decisión de instalar un servidor Apache en las maquinas víctimas.

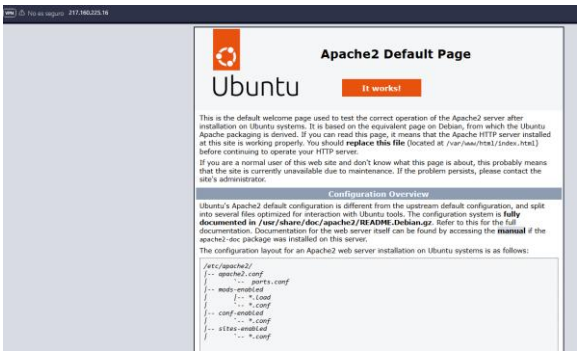

Figura 70. Servidor web Apache

Este cambio me ha permitido simular correctamente el ataque y conseguir que el IDS capturara peticiones HTTP reales. Consiguiendo que analizara los payloads y detectara patrones irregulares en el tráfico.

10 Conclusiones

El desarrollo del sistema IDS me ha permitido lograr con satisfacción los objetivos que se establecieron al comienzo del proyecto. Desde un enfoque más técnico, se ha conseguido desarrollar un sistema completamente funcional con la capacidad de capturar paquetes de red en tiempo real, examinar multitud de protocolos para reconocer patrones anómalos, crear alertas de una forma organizada y enviar los registros a una máquina centralizada de forma autónoma. Además, se ha integrado un sistema de geolocalización basado en IPs públicas, añadiendo un valor a la contextualización de los ataques.

Las pruebas realizadas han sido claves para demostrar la eficacia del sistema frente a diversos tipos de amenazas, abarcando escaneos de puertos, ataques de denegación de servicio y ataques de inyección SQL. Además, durante estas simulaciones, llegaron a surgir limitaciones del entorno -como la transferencia autónoma de los archivos o el filtrado del tráfico ICMP y UDP por parte del proveedor de la infraestructura-, casuísticas que me llevaron a investigar y tomar decisiones técnicas específicas como la instalación de servidores web con la finalidad de ajustar el entorno de pruebas para simular ataques HTTP Flood correctamente.

Por otro lado, desde una perspectiva formativa, el proyecto me ha permitido tener la oportunidad de comprender las bases de la detección de intrusos, adquirir habilidades prácticas en Python para el análisis de redes, y ha supuesto una primera toma de contacto con el enfoque Red Team / Blue Team que tanto me interesa, permitiéndome simular ataques como un atacante y diseñar estrategias de detección y análisis, mejorando mi comprensión sobre la ciberseguridad desde ambas perspectivas. Considero que, para ser capaz de defender adecuadamente un sistema, es fundamental comprender como piensa y actúa un atacante. Asimismo, un proyecto con estas características me ha hecho incrementar la conciencia sobre las amenazas a las que estamos expuestos día a día.

Finalmente, este proyecto ha sido capaz de demostrar que es posible el desarrollar un sistema IDS eficiente utilizando herramientas accesibles y disponibles actualmente, al mismo tiempo que ha sido una gran experiencia de aprendizaje y significativa. Si en un futuro decido orientar mi carrera profesional hacia el ámbito de la ciberseguridad, este trabajo habrá sido, sin lugar a duda, un primer paso muy valioso.

11 Anexos

11.1 Read.me

El proyecto consiste en el diseño e implementación de un Sistema IDS desarrollado en un entorno virtual. Tiene como objetivo la detección de ciberataques simulados en un entorno controlado.

Maquina victima

Sisitema operativo: Ubuntu Server 20.04

Comandos utilizados para su posterior ejecución:

```
sudo apt update && sudo apt upgrade
sudo apt install python3 python3-pip
sudo apt install python3-scapy
sudo apt install apache2 -y
sudo systemctl start apache2
sudo systemctl enable apache2
ssh-keygen -t rsa -b 4096
ssh-copy-id root@<IP_MAQUINA_RECOLECTORA>
```

/root/

```
├── main_ids.py
├── scripts/
│   ├── DoS.py
│   ├── Http_DoS.py
│   ├── SQLi.py
│   ├── nmap.py
│   └── transferir.sh
├── registros/
│   ├── registresDoS.log
│   ├── registresHTTPDoS.log
│   ├── registresNmap.log
│   └── registresSQLi.log
└── trafico/
    ├── dos.log
    └── http_traffic.log
```

Maquina atacante

Sistema operativo: Ubuntu Server 20.04

Comandos utilizados para su posterior ejecución:

```
sudo apt update && sudo apt upgrade
sudo apt install python3 python3-pip
pip3 install scapy
sudo apt install nmap
```

Maquina recolectora

Sistema operativo: Ubuntu Server 20.04

Comandos utilizados para su posterior ejecución:

```
sudo apt update && sudo apt upgrade
sudo apt install python3 python3-pip
sudo apt install python3-scapy
sudo apt install sqlite3
```

Desarrollado por: Aleix Carrillo Solà

Email: aleix.carso@gmail.com

Fecha: Junio 2024

11.2 Nmap.py

```
from scapy.all import sniff, IP, TCP
from collections import defaultdict
import time
import os
import socket
import atexit

class IdsNmap:
    def __init__(self, alertas="/root/registros/registresNmap.log"):
        self.LIMIT = 5
        self.TEMPS_LIMIT = 10
        self.contador = defaultdict(int)
        self.iniciTemps = time.time()
        self.alertas = alertas
        self.ip_victima = self.ipLocal()
        self.detecciones_totales = 0
        self.tipos_detecciones = defaultdict(int)
        if not os.path.exists(self.alertas):
```

```
                with open(self.alertas, 'w'): pass
            atexit.register(self.mostrar_estadisticas)

        def ipLocal(self):
            try:
                s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                s.connect(("8.8.8.8", 80))
                ip_local = s.getsockname()[0]
                s.close()
                return ip_local
            except:
                return "IP_DESCONEGUDA"

        def guardarFitxer(self, missatge):
            timestamp      =      time.strftime("%Y-%m-%d      %H:%M:%S",
time.localtime())
            entrada_log = f"[{timestamp}]\n{missatge}\n"
            with open(self.alertas, 'a') as fitxer:
                fitxer.write(entrada_log)

        def mostrar_estadisticas(self):
            print(f"Total         de         escaneos         detectados:
{self.detecciones_totales}")
            for tipo, cantidad in self.tipos_detecciones.items():
                print(f" - {tipo}: {cantidad}")
            print("----------------------------------------")

        def analizarPaquete(self, paquet):
            if paquet.haslayer(TCP):
                flags = paquet[TCP].flags
                ip_origen = paquet[IP].src
                portDesti = paquet[TCP].dport

                tipo = None
                if flags == 0x02:
                    tipo = "SYN scan"
              elif flags == 0x01:
                    tipo = "FIN scan"
                elif flags == 0x00:
                    tipo = "NULL scan"
                elif flags == 0x29:
```

```
                    tipo = "XMAS scan"
                elif flags == 0x10:
                    tipo = "ACK scan"

                if tipo:
                    self.contador[(ip_origen, tipo)] += 1
                    if self.contador[(ip_origen, tipo)] > self.LIMIT:
                        self.detecciones_totales += 1
                        self.tipos_detecciones[tipo] += 1
                        alerta = (
                            f"    Maquina victima: {self.ip_victima}\n"
                            f"    Nmap ip: {ip_origen}\n"
                            f"    Tipo: {tipo}\n"
                            f"       Paquete:  {self.contador[(ip_origen,
tipo)]}\n"
                            f"    Puerto: {portDesti}\n"
                            "----------------------------------------"
                        )
                        self.guardarFitxer(alerta)
                        self.contador[(ip_origen, tipo)] = 0

                if time.time() - self.iniciTemps > self.TEMPS_LIMIT:
                    self.contador.clear()
                    self.iniciTemps = time.time()

    if __name__ == "__main__":
        detector = IdsNmap()
        try:
            sniff(prn=detector.analizarPaquete, filter="tcp", store=0)
        except KeyboardInterrupt:
            print("\nDetenido")
        finally:
            pass
```

11.3 Dos.py

```
import re
import socket
from collections import defaultdict
from datetime import datetime
```

```
import atexit

LOG_PATH = "/root/trafico/dos.log"
OUTPUT_PATH = "/root/registros/registresDoS.log"
THRESHOLD=200
WINDOW = 15
ataques_totales = 0
ips_atacantes = set()

def mostrar_estadisticas():
    print(f"Total ataques DoS: {ataques_totales}")
    print("----------------------------------------")

atexit.register(mostrar_estadisticas)

def obtener_ip_local():
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.connect(("8.8.8.8", 80))
        ip_local = s.getsockname()[0]
        s.close()
        return ip_local
    except:
        return "127.0.0.1"

def detectar_dos():
    global ataques_totales, ips_atacantes
    conexiones = defaultdict(list)
    ataques_detectados = []
    ip_victima = obtener_ip_local()
    patron = re.compile(
        r'(\d{4}-\d{2}-\d{2}
\d{2}:\d{2}:\d{2}\.\d+)\s+IP\s+(\d+\.\d+\.\d+\.\d+)(?:\.\d+)?\s+>\s+(\d+\
.\d+\.\d+\.\d+)(?:\.\d+)?:'
    )

    with open(LOG_PATH, 'r') as f:
        for linea in f:
            match = patron.search(linea)
            if match:
                timestamp_str, ip_origen, ip_destino = match.groups()
```

```
                    timestamp = datetime.strptime(timestamp_str, '%Y-%m-
%d %H:%M:%S.%f')
                    conexiones[ip_origen].append(timestamp)

        for ip, tiempos in conexiones.items():
            tiempos.sort()
            for i in range(len(tiempos)):
                ventana = [t for t in tiempos if 0 <= (t -
tiempos[i]).total_seconds() <= WINDOW]
                if len(ventana) >= THRESHOLD:
                    ataques_totales += 1
                    ips_atacantes.add(ip)
                    registro = {
                        "fecha":           tiempos[i].strftime('%Y-%m-%d
%H:%M:%S'),
                        "ip_victima": ip_victima,
                        "ip_atacante": ip,
                        "tipo": "DoS",
                        "conexiones": len(ventana),
                        "ventana_tiempo": WINDOW
                    }
                    ataques_detectados.append(registro)
                    break

        return ataques_detectados

    def guardar_registros(ataques):
        with open(OUTPUT_PATH, 'a') as f:
            for ataque in ataques:
                f.write(f"Maquina victima: {ataque['ip_victima']}\n")
                f.write(f"[{ataque['fecha']}]\n")
                f.write(f"{ataque['tipo']}                              ip:
{ataque['ip_atacante']}\n")
                f.write(f"Conexiones: {ataque['conexiones']}\n")
                f.write(f"Ventana de tiempo: {ataque['ventana_tiempo']}
segundos\n")
                f.write("----------------------------------------\n\n")

    if __name__ == "__main__":
        ataques = detectar_dos()
        guardar_registros(ataques)
```

11.4 Http_DoS.py

```
import re
import socket
from collections import defaultdict
from datetime import datetime, timedelta
import atexit

LOG_PATH = "/root/trafico/http_traffic.log"
OUTPUT_PATH = "/root/registros/registresHTTPDoS.log"
THRESHOLD = 150
WINDOW = 10
ataques_totales = 0
ips_atacantes = set()

def mostrar_estadisticas():
    print(f"Total de ataques detectados HTTP DoS: {ataques_totales}")
    print("----------------------------------------")

atexit.register(mostrar_estadisticas)

def obtener_ip_local():
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.connect(("8.8.8.8", 80))
        ip_local = s.getsockname()[0]
        s.close()
        return ip_local
    except:
        return "127.0.0.1"

def detectar_http_dos():
    global ataques_totales, ips_atacantes
    peticiones = defaultdict(list)
    ataques_detectados = []
    ip_victima = obtener_ip_local()
    patron = re.compile(

r'^(\d{2}:\d{2}:\d{2}\.\d+)\s+IP\s+(\d+\.\d+\.\d+\.\d+)\.\d+\s+>\s+(\d+\.
\d+\.\d+\.\d+)\.80:'
    )
```

```
        with open(LOG_PATH, 'r') as f:
            for linea in f:
                match = patron.search(linea)
                if match:
                    timestamp_str, ip_origen, ip_destino = match.groups()
                    if ip_destino == ip_victima:
                        try:
                            hoy = datetime.now().strftime('%Y-%m-%d')
                            timestamp     =     datetime.strptime(f"{hoy}
{timestamp_str}", '%Y-%m-%d %H:%M:%S.%f')
                            peticiones[ip_origen].append(timestamp)
                        except ValueError as e:
                            continue


        for ip, tiempos in peticiones.items():
             tiempos.sort()
            for i in range(len(tiempos)):
                ventana  =  [t  for  t  in  tiempos  if  0  <=  (t  -
tiempos[i]).total_seconds() <= WINDOW]
                if len(ventana) >= THRESHOLD:
                    ataques_totales += 1
                    ips_atacantes.add(ip)
                    registro = {
                        "fecha":           tiempos[i].strftime('%Y-%m-%d
%H:%M:%S'),
                        "ip_victima": ip_victima,
                        "ip_atacante": ip,
                        "tipo": "HTTP DoS",
                        "peticiones": len(ventana),
                        "ventana_tiempo": WINDOW,
                        "detalles": {
                            "puerto_destino": 80                    }
                    }
                    ataques_detectados.append(registro)
                    break


        return ataques_detectados


    def guardar_registros(ataques):
        with open(OUTPUT_PATH, 'a') as f:
            for ataque in ataques:
```

```
                f.write(f"Maquina victima: {ataque['ip_victima']}\n")
                f.write(f"[{ataque['fecha']}]\n")
                f.write(f"{ataque['tipo']}                              ip:
{ataque['ip_atacante']}\n")
                f.write(f"Peticiones HTTP: {ataque['peticiones']}\n")
                f.write(f"Ventana de tiempo: {ataque['ventana_tiempo']}
segundos\n")
                f.write(f"Puerto                                    destino:
{ataque['detalles']['puerto_destino']}\n")
                f.write("----------------------------------------\n\n")

    if __name__ == "__main__":
        ataques = detectar_http_dos()
        guardar_registros(ataques)
```

11.5 SQLi.py

```
from scapy.all import sniff, IP, Raw, TCP
import re
from datetime import datetime
import atexit

LOG_FILE = "/root/registros/registresSQLi.log"
detecciones_totales = 0
ips_atacantes = set()
payloads_detectados = []

def mostrar_estadisticas():
    print(f"Total de ataques SQLi detectados: {detecciones_totales}")
    print("----------------------------------------")
atexit.register(mostrar_estadisticas)

class SQLiDetector:
    def __init__(self):
        self.patrones_sqli = [
            r"union.*select",
            r"1=['\"]?1['\"]?",
            r"sleep\(\d+\)",
            r"drop\s+table",
            r"select.*from",
            r"or\s+1=1",
```

```
                r"--\s*$"
            ]

        def guardar_log(self, ip_victima, ip_atacante, payload):
            global          detecciones_totales,          ips_atacantes,
payloads_detectados
            detecciones_totales += 1
            ips_atacantes.add(ip_atacante)
            payloads_detectados.append(payload)
            timestamp = datetime.now().strftime("[%Y-%m-%d %H:%M:%S]")
            with open(LOG_FILE, 'a') as f:
                f.write(
                    f"{timestamp}\n"
                    f"    Maquina victima: {ip_victima}\n"
                    f"    SQLi ip: {ip_atacante}\n"
                    f"    Payload: {payload[:200]}"
                    "----------------------------------------\n"
                )

        def analizar_paquete(self, paquete):
            if paquete.haslayer(TCP) and paquete.haslayer(Raw):
                try:
                    payload     =     paquete[Raw].load.decode('utf-8',
errors='ignore')
                    for patron in self.patrones_sqli:
                        if re.search(patron, payload, re.IGNORECASE):
                            self.guardar_log(
                                ip_victima=paquete[IP].dst,
                                ip_atacante=paquete[IP].src,
                                payload=payload
                            )
                            break
                except:
                    pass

    if __name__ == "__main__":
        detector = SQLiDetector()
        sniff(prn=detector.analizar_paquete, store=0, filter="tcp")
```

11.6 Transferir.sh

```
#!/bin/bash
```

```
directorioLocal="/root/registros"
patronArchivos="registres*"
recolector="root"
IpRecolector="82.223.165.171"
directorioRecolector="/root"

for archivo in $directorioLocal/$patronArchivos; do
    if [ -f "$archivo" ]; then
        scp                                                    "$archivo"
"$recolector@$IpRecolector:$directorioRecolector"
        if [ $? -eq 0 ]; then
            echo "Archivo $archivo transferido"
        fi
    fi
done
```

11.7 Db_ataques.py

```
import sqlite3
import glob
import re
import requests
from datetime import datetime

DB_FILE = "logs_ataques.db"
LOG_FILES = "/root/registres*.log"
API_URL = "http://ip-api.com/json/{}?fields=country"

pais_cache = {}

def obtener_pais(ip):
    if ip not in pais_cache:
        try:
            response = requests.get(API_URL.format(ip), timeout=3)
            if response.status_code == 200:
                pais_cache[ip]    =    response.json().get('country',
'Desconocido')
            else:
                pais_cache[ip] = 'Desconocido'
        except:
```

```
                pais_cache[ip] = 'Desconocido'
        return pais_cache[ip]


    def crear_tabla():
        conn = sqlite3.connect(DB_FILE)
        cursor = conn.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS ataques (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                fecha TEXT NOT NULL,
                ip_victima TEXT NOT NULL,
                ip_atacante TEXT NOT NULL,
                tipo TEXT,
                pais TEXT,
                detalle TEXT,
                UNIQUE(fecha, ip_victima, ip_atacante, tipo, detalle)
            )
        ''')

        cursor.execute('CREATE INDEX IF NOT EXISTS index_fecha ON
ataques(fecha)')
        cursor.execute('CREATE INDEX IF NOT EXISTS index_tipo ON
ataques(tipo)')
        conn.commit()
        conn.close()


    def procesar_log(log_block):
        try:
            ip_victima = re.search(r'Maquina victima: (.*?)\n',
log_block).group(1).strip()
            fecha = re.search(r'\[(.*?)\]', log_block).group(1)
            if 'SQLi ip:' in log_block:
                return {
                    'fecha': fecha,
                    'ip_victima': ip_victima,
                    'ip_atacante': re.search(r'SQLi ip: (.*?)\n',
log_block).group(1).strip(),
                    'tipo': 'SQLi',
                    'detalle': re.search(r'Payload: (.*?)\n',
log_block).group(1)
                }
            elif 'Nmap ip:' in log_block:
```

```
                return {
                    'fecha': fecha,
                    'ip_victima': ip_victima,
                    'ip_atacante':   re.search(r'Nmap   ip:   (.*?)\n',
log_block).group(1).strip(),
                    'tipo': 'Nmap',
                    'detalle': log_block.strip()
                }

            elif 'HTTP DoS ip:' in log_block:
                return {
                    'fecha': fecha,
                    'ip_victima': ip_victima,
                    'ip_atacante':  re.search(r'HTTP  DoS  ip:  (.*?)\n',
log_block).group(1).strip(),
                    'tipo': 'HTTP DoS',
                    'detalle': (
                        f"Peticiones   HTTP:   {re.search(r'Peticiones
HTTP: (.*?)\n', log_block).group(1)} en "
                        f"{re.search(r'Ventana   de   tiempo:   (.*?)
segundos', log_block).group(1)}s, "
                        f"Puerto destino: {re.search(r'Puerto destino:
(.*?)\n', log_block).group(1)}, "
                    )
                }
            elif 'DoS ip:' in log_block:
                return {
                    'fecha': fecha,
                    'ip_victima': ip_victima,
                    'ip_atacante':   re.search(r'DoS   ip:   (.*?)\n',
log_block).group(1).strip(),
                    'tipo': 'DoS',
                    'detalle':  f"Conexiones:  {re.search(r'Conexiones:
(.*?)\n', log_block).group(1)} en {re.search(r'Ventana de tiempo: (.*?)
segundos', log_block).group(1)}s"
                }
        except Exception as e:
            print(f"Error bloque: {e}")
        return None


    def importar_logs():
        crear_tabla()
        conn = sqlite3.connect(DB_FILE)
```

```
        cursor = conn.cursor()
        for file in glob.glob(LOG_FILES):
            with open(file, 'r') as f:
                log_block = ""
                for line in f:
                    if  line.startswith("----------------------------------------"):
                        if log_block:
                            datos = procesar_log(log_block)
                            if datos:
                                pais = obtener_pais(datos['ip_atacante'])
                                cursor.execute('''
                                        INSERT OR IGNORE INTO ataques
                                    (fecha, ip_victima, ip_atacante, tipo, pais, detalle)
                                    VALUES (?, ?, ?, ?, ?, ?)
                                ''', (
                                    datos['fecha'],
                                    datos['ip_victima'],
                                    datos['ip_atacante'],
                                    datos['tipo'],
                                    pais,
                                    datos['detalle']
                                ))
                            log_block = ""
                    else:
                        log_block += line
        conn.commit()
        conn.close()
        print(f"Ataques en {DB_FILE}")

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

11.8 App.py

```
    from flask import Flask, render_template, send_from_directory, Response
    import csv
    import sqlite3
```

```
import folium
import os
import requests
from collections import Counter
import json
from datetime import datetime
from io import StringIO

app = Flask(__name__,
            template_folder='/root/web/html',
            static_folder='/root/web/css')

MAPS_DIR = '/root/web/html'
STATIC_DIR = '/root/web/css'
os.makedirs(MAPS_DIR, exist_ok=True)
os.makedirs(os.path.join(STATIC_DIR, 'css'), exist_ok=True)

@app.route('/')
def inicio():
    conn = sqlite3.connect('/root/logs_ataques.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM ataques ORDER BY fecha DESC")
    ataques = cursor.fetchall()
    conn.close()
    generar_mapa(ataques)
    tipos = [a[4] for a in ataques]
    conteo_tipos = Counter(tipos)

    fechas = [datetime.strptime(a[1], "%Y-%m-%d
%H:%M:%S").date().isoformat() for a in ataques]
    conteo_fechas = Counter(fechas)

    return render_template(
        'index.html',
        ataques=ataques,
        tipos_data=json.dumps(conteo_tipos),
        fechas_data=json.dumps(conteo_fechas)
    )

@app.route('/descargar')
def descargar_csv():
```

```
        conn = sqlite3.connect('/root/logs_ataques.db')
        cursor = conn.cursor()
        cursor.execute("SELECT fecha, tipo, ip_atacante, pais FROM
ataques ORDER BY fecha DESC")
        ataques = cursor.fetchall()
        conn.close()

        salida = StringIO()
        writer = csv.writer(salida)
        writer.writerow(['Fecha', 'Tipo', 'IP Atacante', 'País'])

        for ataque in ataques:
            writer.writerow(ataque)

        salida.seek(0)
        return Response(
            salida.getvalue(),
            mimetype='text/csv',
            headers={'Content-Disposition': 'attachment;
filename=reportes_ataques.csv'}
        )

    @app.route('/mapa')
    def mostrar_mapa():
        return send_from_directory(MAPS_DIR, 'mapa.html')

    def generar_mapa(ataques):
        mapa = folium.Map(location=[40, -3], zoom_start=2)

        for ataque in ataques:
            ip = ataque[3]
            tipo = ataque[4]

            try:
                response = requests.get(f"http://ip-api.com/json/{ip}",
timeout=3)
                data = response.json()
                if data['status'] == 'success':
                    lat, lon = data['lat'], data['lon']
                    ciudad = data.get('city', 'Desconocida')
                    pais = data.get('country', 'Desconocido')
```

```
                org = data.get('org', 'Desconocida')
            else:
                lat, lon = 40, -3
                ciudad = pais = org = "Desconocido"
        except Exception as e:
            print(f"Error geolocalizando {ip}: {str(e)}")
            lat, lon = 40, -3
            ciudad = pais = org = "Desconocido"

        popup_content = f"""
        <b>IP:</b> {ip}<br>
        <b>Tipo:</b> {tipo}<br>
        <b>Ubicación:</b> {ciudad}, {pais}<br>
        <b>Proveedor:</b> {org}
        """
        folium.Marker(
            location=[lat, lon],
            popup=folium.Popup(popup_content, max_width=300),
            icon=folium.Icon(color='red', icon='warning-sign')
        ).add_to(mapa)

    mapa.save(os.path.join(MAPS_DIR, 'mapa.html'))
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

11.9 Index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Monitor de Ataques</title>
    <link rel="stylesheet" href="/css/styles.css">
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
    <header>
        <div class="container">
            <h1>Monitor de Seguridad - Sistema IDS</h1>
        </div>
    </header>

    <div class="dashboard-container">
```

```
    <div class="dashboard-panel">
            <div class="card">
                <div class="card-header">Ataques detectados</div>
                <div class="card-body">
                    <a  href="/descargar"  class="btn-descarga">Descargar
reportes CSV</a>
                    <div class="tabla-scroll">
                      <table>
                        <thead>
                            <tr>
                                <th>Fecha</th>
                                <th>Tipo</th>
                                <th>IP Atacante</th>
                                <th>País</th>
                            </tr>
                        </thead>
                        <tbody>
                            {% for ataque in ataques %}
                            <tr>
                                <td>{{ ataque[1] }}</td>
                                <td><span  class="badge  badge-danger">{{
ataque[4] }}</span></td>
                                <td>{{ ataque[3] }}</td>
                                <td>{{ ataque[5] | truncate(30) }}</td>
                            </tr>
                            {% endfor %}
                        </tbody>
                      </table>
                    </div>
                </div>
            </div>
        </div>

    <div class="dashboard-panel" id="mapa-panel">
        <div class="card">
            <div class="card-header">
                Mapa de Ataques en Tiempo Real
                <button     onclick="toggleMapa()"     class="btn-toggle-
mapa">Ampliar</button>
            </div>
```

```
        <div class="card-body">
            <iframe id="mapa-frame" src="/mapa"></iframe>
        </div>
    </div>
</div>


    <div class="dashboard-panel">
        <div class="card">
            <div class="card-header">Estadísticas de Ataques</div>
            <div class="card-body">
                <div class="chart-container">
                    <canvas id="graficoTipos"></canvas>
                </div>
                <div class="chart-container">
                    <canvas id="graficoFechas"></canvas>
                </div>
            </div>
        </div>
    </div>


    </div>

    <div class="footer">
        Sistema IDS &copy; 2025 | Sistema de seguridad en tiempo real
    </div>

    <script>
        const tiposData = {{ tipos_data|safe }};
        const fechasData = {{ fechas_data|safe }};

        new Chart(document.getElementById("graficoTipos"), {
            type: 'bar',
            data: {
                labels: Object.keys(tiposData),
                datasets: [{
                    label: 'Ataques por tipo',
                    data: Object.values(tiposData),
                    backgroundColor: '#3498db'
```

```
                    }]
                },
                options: {
                    responsive: true,
                    plugins: { legend: { display: false } },
                    scales: { y: { beginAtZero: true } }
                 }
            });

            new Chart(document.getElementById("graficoFechas"), {
                type: 'line',
                data: {
                    labels: Object.keys(fechasData),
                    datasets: [{
                        label: 'Ataques por día',
                        data: Object.values(fechasData),
                        borderColor: '#e74c3c',
                        fill: false,
                        tension: 0.3
                    }]
                },
                options: {
                    responsive: true,
                    scales: { y: { beginAtZero: true } }
                }
            });
         </script>
        <script>
         function toggleMapa() {
            const panel = document.getElementById("mapa-panel");
            panel.classList.toggle("mapa-ampliado");

            const btn = document.querySelector(".btn-toggle-mapa");
            btn.textContent = panel.classList.contains("mapa-ampliado") ?
"Restaurar vista" : "Ampliar";
         }
    </script>

    </body>
    </html>
```

11.10 Styles.css

```
:root {
    --primary-color: #1a1a2e;
    --secondary-color: #16213e;
    --accent-color: #0f3460;
    --danger-color: #e94560;
    --success-color: #00b894;
    --warning-color: #f39c12;
    --info-color: #3498db;
    --text-color: #e6e6e6;
    --text-muted: #b3b3b3;
    --bg-color: #121212;
    --card-bg: #1e1e1e;
    --border-color: #333;
    --border-radius: 8px;
    --box-shadow: 0 4px 6px rgba(0, 0, 0, 0.3);
    --transition: all 0.3s ease;
}

* {
    box-sizing: border-box;
}

body {
    font-family: 'Roboto', 'Segoe UI', Tahoma, Geneva, Verdana, sans-
serif;
    margin: 0;
    padding: 0;
    background-color: var(--bg-color);
    color: var(--text-color);
    line-height: 1.6;
    min-height: 100vh;
    display: flex;
    flex-direction: column;
}

.container {
    max-width: 1200px;
    margin: 0 auto;
```

```
    padding: 0 20px;
    width: 100%;
}

header {
    background-color: var(--primary-color);
    color: var(--text-color);
    padding: 20px 0;
    margin-bottom: 30px;
    box-shadow: var(--box-shadow);
    border-bottom: 1px solid var(--accent-color);
}

header h1 {
    margin: 0;
    padding: 0;
    font-size: 28px;
    text-align: left;
    letter-spacing: 0.5px;
}

.dashboard-container {
    display: flex;
    justify-content: space-between;
    gap: 20px;
    padding: 0 20px;
    flex-wrap: wrap;
    margin-bottom: 30px;
}

.dashboard-panel {
    flex: 1 1 32%;
    min-width: 300px;
    display: flex;
    flex-direction: column;
}

.card {
    background: var(--card-bg);
    border-radius: var(--border-radius);
```

```
    box-shadow: var(--box-shadow);
    margin-bottom: 30px;
    overflow: hidden;
    display: flex;
    flex-direction: column;
    flex-grow: 1;
    border: 1px solid var(--border-color);
    transition: var(--transition);
}

.card:hover {
    transform: translateY(-5px);
    box-shadow: 0 10px 20px rgba(0, 0, 0, 0.3);
}

.card-header {
    background-color: var(--accent-color);
    color: var(--text-color);
    padding: 15px 20px;
    font-size: 18px;
    font-weight: bold;
    border-bottom: 1px solid var(--border-color);
    display: flex;
    justify-content: space-between;
    align-items: center;
}

.card-body {
    padding: 20px;
    flex-grow: 1;
    display: flex;
    flex-direction: column;
}

.tabla-scroll {
    max-height: 400px;
    overflow-y: auto;
    border: 1px solid var(--border-color);
    background-color: var(--primary-color);
    border-radius: var(--border-radius);
```

```
    flex-grow: 1;
}

table {
    width: 100%;
    border-collapse: collapse;
    margin: 0;
}

th, td {
    padding: 12px 15px;
    text-align: left;
    border-bottom: 1px solid var(--border-color);
}

th {
    background-color: var(--secondary-color);
    color: var(--text-color);
    font-weight: 500;
    position: sticky;
    top: 0;
}

tr:nth-child(even) {
    background-color: rgba(255, 255, 255, 0.05);
}

tr:hover {
    background-color: rgba(255, 255, 255, 0.1);
}

.badge {
    display: inline-block;
    padding: 4px 10px;
    border-radius: 20px;
    font-size: 12px;
    font-weight: bold;
    text-transform: uppercase;
    letter-spacing: 0.5px;
}
```

```
.badge-danger {
    background-color: var(--danger-color);
    color: white;
}

.badge-warning {
    background-color: var(--warning-color);
    color: white;
}

.badge-info {
    background-color: var(--info-color);
    color: white;
}

.btn-descarga {
    display: inline-block;
    margin-bottom: 15px;
    background-color: var(--success-color);
    color: white;
    padding: 10px 15px;
    border-radius: var(--border-radius);
    text-decoration: none;
    font-weight: bold;
    box-shadow: var(--box-shadow);
    transition: var(--transition);
    border: none;
    cursor: pointer;
    text-align: center;
}

.btn-descarga:hover {
    background-color: #00a884;
    transform: translateY(-2px);
    box-shadow: 0 6px 12px rgba(0, 0, 0, 0.2);
}

.btn-toggle-mapa {
    background-color: var(--accent-color);
```

```
    color: white;
    border: none;
    padding: 6px 12px;
    border-radius: 4px;
    cursor: pointer;
    transition: var(--transition);
    font-size: 14px;
}

.btn-toggle-mapa:hover {
    background-color: #0d2b52;
    transform: translateY(-1px);
}

.mapa-ampliado {
    flex: 1 1 100% !important;
    z-index: 100;
}

.mapa-ampliado iframe {
    height: 600px !important;
}

iframe {
    border: none;
    border-radius: var(--border-radius);
    width: 100%;
    height: 400px;
    background-color: var(--primary-color);
    border: 1px solid var(--border-color);
}

.chart-container {
    position: relative;
    margin-bottom: 20px;
    flex-grow: 1;
    min-height: 300px;
}

.footer {
```

```
    text-align: center;
    padding: 20px;
    margin-top: auto;
    background-color: var(--primary-color);
    color: var(--text-muted);
    font-size: 14px;
    border-top: 1px solid var(--border-color);
}

.tabla-scroll::-webkit-scrollbar {
    width: 8px;
    height: 8px;
}

.tabla-scroll::-webkit-scrollbar-track {
    background: var(--primary-color);
}

.tabla-scroll::-webkit-scrollbar-thumb {
    background-color: var(--accent-color);
    border-radius: 4px;
}

@media (max-width: 1024px) {
    .dashboard-panel {
        flex: 1 1 48%;
    }
}

@media (max-width: 768px) {
    .dashboard-panel {
        flex: 1 1 100%;
    }

    th, td {
        padding: 10px 12px;
        font-size: 14px;
    }

    .card-header {
```

```
        padding: 12px 15px;
        font-size: 16px;
    }

    header h1 {
        font-size: 24px;
    }
}

@media (max-width: 480px) {
    .dashboard-container {
        padding: 0 10px;
        gap: 15px;
    }

    .btn-descarga, .btn-toggle-mapa {
        padding: 8px 12px;
        font-size: 13px;
    }
}
```

11.11 DoS_test.py

```
import socket
import time

IP_VICTIMA = "217.160.225.16"
PUERTO =22
REPETICIONES = 1000

for _ in range(REPETICIONES):
    try:
        s = socket.socket()
        s.settimeout(0.5)
        s.connect((IP_VICTIMA, PUERTO))
    except:
        pass
    finally:
        s.close()
```

11.12 SQLi_test.py

```
import socket
import random
import time

RECOLECTOR_IP ="217.160.225.16"
PORT = 80

payloads = [
    "GET /login.php?user=admin' OR '1'='1'-- HTTP/1.1\r\nHost: {}\r\n\r\n",
    "POST /check_login.php HTTP/1.1\r\nHost: {}\r\nContent-Length: 28\r\n\r\nusername=admin'--&password=1234",
    "GET /search?q=1' UNION SELECT 1,2,3-- HTTP/1.1\r\nHost: {}\r\n\r\n"
]

def enviar_ataques():
    for i, payload in enumerate(payloads):
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.connect((RECOLECTOR_IP, PORT))
            s.sendall(payload.format(RECOLECTOR_IP).encode())
            print(f"Payload {i+1} enviado")
            s.close()
        except Exception as e:
            print(f"Error: {str(e)}")
        time.sleep(random.uniform(0.5, 2))

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

11.13 main_ids.py

```
import threading
import subprocess
import time
import os
import signal
detener_ejecucion = False
```

```
    def iniciar_tcpdump():
        comando = [
            "tcpdump", "-tttt", "-n", "-A", "-i", "ens6",
            "(icmp or udp or tcp)", "-l"
        ]
        with open("/root/trafico/dos.log", "w") as f:
            return          subprocess.Popen(comando,          stdout=f,
stderr=subprocess.DEVNULL)

    def iniciar_tcpdump_http():
        comando = [
            "tcpdump", "-i", "ens6", "-nn",
            "tcp port 80", "-l"
        ]
        with open("/root/trafico/http_traffic.log", "w") as f:
            return          subprocess.Popen(comando,          stdout=f,
stderr=subprocess.DEVNULL)

    def ejecutar_sql_ids():
        print("Detector SQLi")
        subprocess.run(["python3", "/root/scripts/SQLi.py"])

    def ejecutar_nmap_ids():
        print("Detector Nmap - SYN - FIN - NULL - XMAS")
        subprocess.run(["python3", "/root/scripts/nmap.py"])

    def ejecutar_dos_detector():
        print("Detector TCP, UDP, ICMP DoS")
        while not detener_ejecucion:
            subprocess.run(["python3", "/root/scripts/DoS.py"])
            time.sleep(10)

    def ejecutar_http_dos():
        print("Detector HTTP DoS")
        while not detener_ejecucion:
            subprocess.run(["python3", "/root/scripts/Http_DoS.py"])
            time.sleep(10)

    def transferir_registros():
        print("Transfiriendo archivos")
        subprocess.run(["/bin/bash", "/root/scripts/transferir.sh"])
```

```
def handler_shutdown(signum, frame):
    global detener_ejecucion
    detener_ejecucion = True
if __name__ == "__main__":
    signal.signal(signal.SIGINT, handler_shutdown)
    signal.signal(signal.SIGTERM, handler_shutdown)
    print("[+] IDS iniciado")
    tcpdump_proc = iniciar_tcpdump()
    http_tcpdump_proc = iniciar_tcpdump_http()
    time.sleep(2)
    t1 = threading.Thread(target=ejecutar_sql_ids)
    t2 = threading.Thread(target=ejecutar_nmap_ids)
    t3 = threading.Thread(target=ejecutar_dos_detector)
    t4 = threading.Thread(target=ejecutar_http_dos)
    t1.start()
    t2.start()
    t3.start()
    t4.start()
    try:
        t1.join()
        t2.join()
        t3.join()
        t4.join()
    except KeyboardInterrupt:
        pass
    finally:
        transferir_registros()
        tcpdump_proc.terminate()
        http_tcpdump_proc.terminate()
        tcpdump_proc.wait()
        http_tcpdump_proc.wait()
        print("[+] IDS detenido")
        os._exit(0)
```

Referencias

Herramientas y software:

[1] mRemoteNG [Software], versión 1.76.

[2] Python [Lenguaje de programación], versión 3.12.3

[3] SQLite [Base de datos], versión 3.45.1

[4] Flask [Marco web], versión 2.x, Pallets Projects.

[5] Folium [Biblioteca de Python], versión 0.12.

[6] Ip-api [Servicio web]. [En línea]. https://ip-api.com

[7] Bash [Intérprete de comandos],

[8] Nmap [Herramienta de red], versión 7.80. [En línea]. https://nmap.org

[9] Hping3 [Herramienta de red], versión 3.0.0.-alpha-2

[10] Zenmap GUI [Interfaz gráfica para Nmap]. https://nmap.org/zenmap

[11] SCP (Protocolo de Copia Segura) [Herramienta de Transferencia], OpenSSH.

[12] Enterprise Architect [Software de Modelado], Sparx Systems.

[13] HTML y CSS [Lenguajes Web], Consorcio World Wide Web (W3C).

Referencias Web:

[14] IBM. “Sistemas de Detección de Intrusiones (IDS)”. [10-03-2025]
https://www.ibm.com/es-es/topics/intrusion-detection-system

[15] INCIBE. «¿Qué son los SIEM, los IDS y los IPS y para qué sirven?», [10-03-2025] 2020
https://www.incibe.es/empresas/blog/son-y-sirven-los-siem-ids-e-ips

[16] Wikipedia. «Sistema de Detección de Intrusiones». [10-03-2025]
https://es.wikipedia.org/wiki/Sistema_de_detecci%C3%B3n_de_intrusos

[17] Fortinet. "Sistema de Detección de Intrusiones (IDS)". [13-03-2025]
https://www.fortinet.com/lat/resources/cyberglossary/intrusion-detection-system

[18] Proofpoint. "Sistema de Detección de Intrusiones (IDS)". [13-03-2025]
https://www.proofpoint.com/es/threat-reference/intrusion-detection-system-ids

[19] KeepCoding. "¿Qué son los NIDS?".[20-03-2025] 2024
https://keepcoding.io/blog/que-son-los-nids/

[20] ACRecenta. "Sistema de Detección de Intrusiones en la Red (NIDS)". [20-03-2025]
https://acrecenta.com/es/glosario-ciberseguridad/network-intrusion-detection-system-nids

[21] IBM CORPORATION. "Intrusion Prevention Systems (IPS).". [20-03-2025] 2023
https://www.ibm.com/es-es/topics/intrusion-prevention-system

[22] CEISTT TEAM. "Intrusion Prevention System (IPS).". [20-03-2025] 2023
https://ceistt.es/sistema-de-prevencion-de-intrusos-ips/

[23] FORTINET INC. "Types of Cyberattacks.". [20-03-2025] 2023
https://www.fortinet.com/lat/resources/cyberglossary/types-of-cyber-attacks

[24] PROOFPOINT INC. "Network Threats.". [20-03-2025]
https://www.proofpoint.com/threat-reference/network-delivered-threats

[25] REDESZONE TEAM. "Complete List of Network Attacks and How to Avoid Them.". [29-03-2025] 2023
https://www.redeszone.net/tutoriales/seguridad/listado-completo-ataques-redes-como-evitarlos/

[26] FORTINET INC. "What is a Port Scan?". [29-03-2025] 2023

https://www.fortinet.com/lat/resources/cyberglossary/what-is-port-scan

[27] VECTRA AI TEAM. "Port Scanning.". [03-04-2025] 2023

https://es.vectra.ai/attack-techniques/port-scan

[28] AKAMAI TECHNOLOGIES. "What is a DDoS Attack?". [03-04-2025] 2023

https://www.akamai.com/glossary/what-is-ddos

[29] SCAPY DEVELOPMENT TEAM. "Official Scapy Documentation.". [03-04-2025] 2023

https://www.akamai.com/glossary/what-is-ddos

[30] GARCÍA GRANDES, J. L. "Folium: utilizando Leaflet cono Python". [07-04-2025] 2022

https://mappinggis.com/2022/09/folium-utilizando-leaflet-con-python/

[31] DIEGO B. "Protocolo TCP: qué se y cómo funciona". [07-04-2025] 2023

https://www.hostinger.com/es/tutoriales/protocolo-tcp

[32] FORTINET INC. "Modelo TCP/IP". [07-04-2025] 2023

https://www.fortinet.com/lat/resources/cyberglossary/tcp-ip

[33] IMPERVA. "HTTP Flood DDoS Attack". [12-04-2025] 2023

https://www.imperva.com/learn/ddos/http-flood/

[34] CLOUDFLARE. "HTTP Flood DDoS Attack". [12-04-2025] 2023

https://www.cloudflare.com/es-es/learning/ddos/http-flood-ddos-attack/

[35] CLOUDFLARE. "UDP Flood DDoS Attack". [12-04-2025] 2023

https://www.cloudflare.com/es-es/learning/ddos/udp-flood-ddos-attack/

[36] AKAMAI TECHNOLOGIES. "UDP Flood DDoS Attack". [15-04-2025] 2023

https://www.akamai.com/glossary/what-is-udp-flood-ddos-attack

[37] CLOUDFLARE. "ICMP Flood (Ping Flood) DDoS Attack". [15-04-2025] 2023

https://www.cloudflare.com/es-es/learning/ddos/ping-icmp-flood-ddos-attack/

[38] VERGARA CARMONA, A. "Guía del comando Tcpdump". [16-04-2025] 2023

https://vergaracarmona.es/guia-del-comando-tcpdump/

[39] DE LUZ, S. "Tcpdump: capturar tráfico de red en Linux". [16-04-2025] 2023

https://www.redeszone.net/tutoriales/servidores/tcpdump-capturar-trafico-red-linux/

[40] BLANTON, S. "Qué sueño las clavas SSH". [16-04-2025] 2023

https://jumpcloud.com/es/blog/what-are-ssh-keys

[41] GITHUB. "Generar una nueva clavo SSH y añadirla al ssh-agente". [16-04-2025] 2023

https://docs.github.com/es/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent