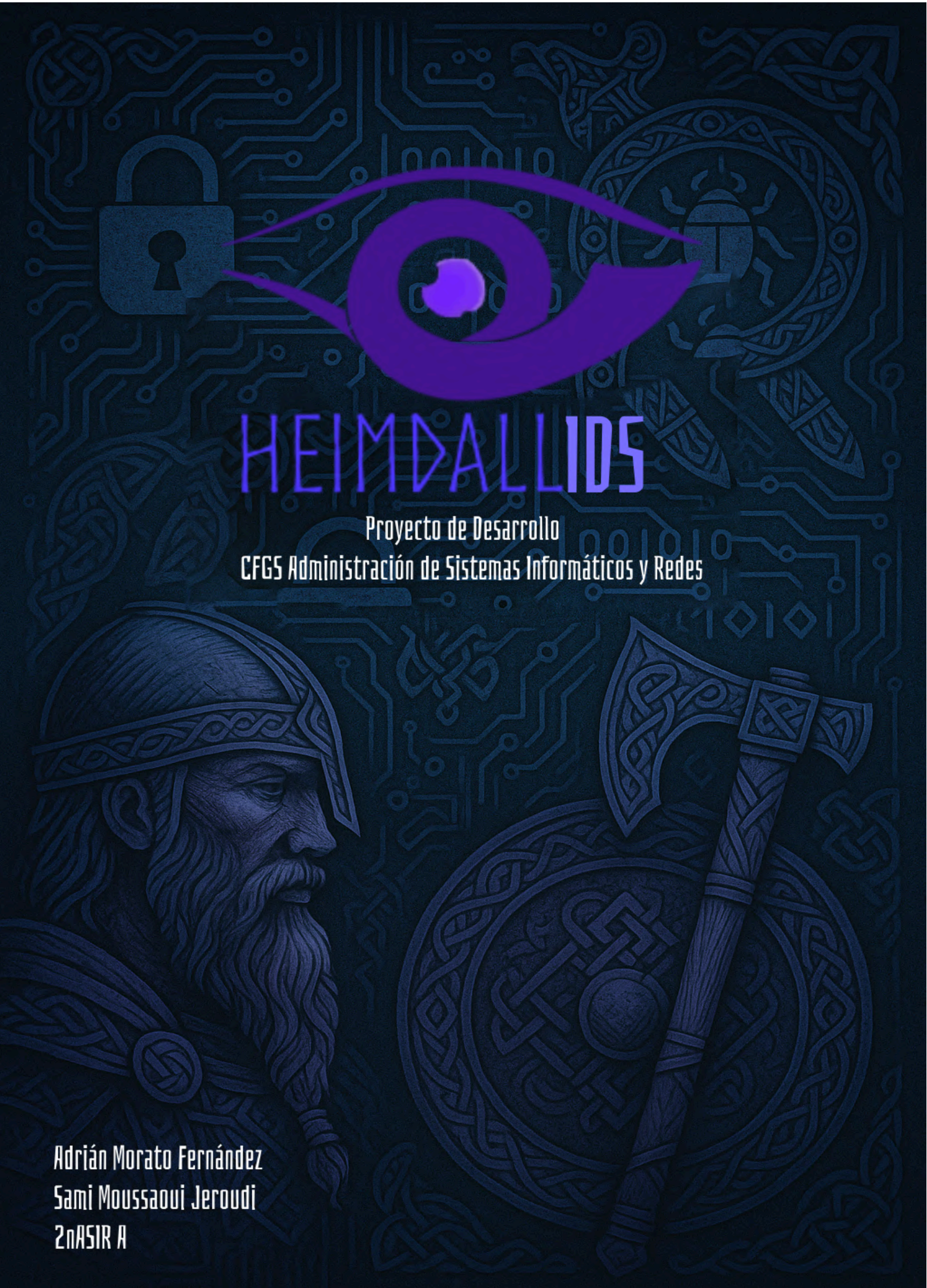


HEIMDALLIDS

Proyecto de Desarrollo

CFGS Administración de Sistemas Informáticos y Redes

Adrián Morato Fernández
Sami Moussaoui Jeroudi
2nASIR A

Título

Heimdall IDS

El nombre del proyecto hace referencia al dios nórdico de la vigilancia, conocido por su capacidad para percibir cualquier peligro. Esta simbología resulta adecuada para el funcionamiento del sistema, el cual monitorea de manera continua la red con el fin de detectar y responder a posibles amenazas.

Propuesta

El proyecto Heimdall IDS se orienta a la detección, el monitoreo y la respuesta ante amenazas de seguridad en redes informáticas. Para ello, el sistema combina diversas herramientas de ciberseguridad que permiten analizar el tráfico de la red, supervisar los eventos generados por los dispositivos y emitir alertas al identificar comportamientos sospechosos.

El diseño se fundamenta en la integración de Suricata, un sistema de detección y prevención de intrusiones (IDS/IPS), y Wazuh, una plataforma destinada al análisis de registros y la protección de endpoints. Ambos componentes operan conjuntamente con Elasticsearch y Kibana, los cuales facilitan el almacenamiento y la visualización estructurada de los datos recopilados.

Adicionalmente, Heimdall IDS incorpora un mecanismo de notificaciones mediante Telegram, de modo que los administradores reciben alertas en tiempo real para proceder con las decisiones de seguridad desde cualquier ubicación. Asimismo, se contempla la aplicación de técnicas de Machine Learning con el objetivo de identificar patrones anómalos de comportamiento y optimizar la capacidad de detección de posibles ataques.

Logo

El logo de Heimdall IDS ha sido diseñado para reflejar tanto su propósito como su inspiración mitológica. La forma principal del logo es un ojo estilizado, símbolo de vigilancia, protección y alerta constante, representa la función del IDS: vigilar el sistema de forma continua para detectar amenazas invisibles. La forma del ojo acaba transformándose en un cuerno vikingo, el Gjallarhorn. Este objeto lo usaba Heimdall en la mitología nórdica para avisar a los habitantes de Asgard cuando sus enemigos, los gigantes, se acercaban en el Ragnarök, el fin del mundo de los dioses y hombres.

Se han elegido unos colores morados y rosáceos para diferenciar de los típicos rojos o azules, además de que le da una estética mística. La tipología del logo simula ser runas vikingas.

Tecnologías

Se utilizarán diversas tecnologías orientadas a la detección, monitoreo y respuesta a incidentes de ciberseguridad. Suricata se encargará del análisis del tráfico de red como sistema de detección y prevención de intrusiones (IDS/IPS), mientras que Wazuh permitirá monitorear registros y proteger los endpoints. Estos componentes se complementarán con Elasticsearch y Kibana, que facilitarán el almacenamiento y la visualización de datos mediante paneles de control dinámicos.

Para automatizar respuestas, se implementará un bot de Telegram, que enviará notificaciones en tiempo real y permitirá ejecutar acciones de seguridad, como el bloqueo de IPs mediante Nftables. Además, se explorará el uso de Machine Learning con Scikit-learn para detectar anomalías en los registros de seguridad. Todo el sistema se desplegará en tres máquinas, con un servidor central, un cliente y una máquina atacante.

Objetivos

- Monitorear la seguridad de la red y de los sistemas mediante la recopilación y análisis de registros en tiempo real.
- Detectar intrusiones y comportamientos anómalos con Suricata (IDS/IPS) y Wazuh (monitoreo de endpoints).
- Automatizar alertas y respuesta a incidentes a través de un bot de Telegram que informe sobre posibles amenazas y permita acciones remotas.
- Utilizar Elasticsearch y Kibana para el almacenamiento, consulta y visualización de datos de seguridad de manera eficiente.
- Implementar Machine Learning para identificar patrones de ataque y mejorar la detección de amenazas emergentes.
- Realizar pruebas en un entorno controlado con tres máquinas para simular una infraestructura real y validar el funcionamiento del sistema.

ÍNDICE

Memoria del Proyecto Heimdall IDS
Autores: Adrián Morato Fernández y Sami Moussaoui Jeroudi

Capítulo 1: Introducción

El proyecto Heimdall IDS tiene como objetivo el desarrollo de un Sistema de Gestión de Eventos e Información de Seguridad (SIEM) basado en tecnologías como Wazuh y Suricata, con la capacidad de detectar, analizar y responder a amenazas en tiempo real. Implementaremos un sistema que recopile registros de seguridad y genere alertas automatizadas, integrándose con Telegram para la interacción con los administradores de seguridad.

1.1 Contexto

En un entorno digital cada vez más expuesto a ciberamenazas, la monitorización proactiva de redes y sistemas se ha convertido en un pilar fundamental para garantizar la seguridad informática. Los sistemas tradicionales de detección de intrusiones (IDS) y las soluciones de análisis de logs suelen operar de forma aislada, lo que limita su eficacia. Heimdall IDS surge como un proyecto integrador que combina tecnologías de vanguardia para ofrecer una respuesta unificada ante amenazas, aprovechando herramientas como Suricata y Wazuh. Este enfoque transversal busca superar las limitaciones de las soluciones actuales.

1.2 Justificación

La creciente sofisticación de los ciberataques exige sistemas capaces de correlacionar datos en tiempo real y automatizar respuestas. Heimdall IDS aborda esta necesidad al integrar detección de intrusiones, monitorización de endpoints, visualización de datos y alertas automatizadas en una única plataforma. Su desarrollo no solo demuestra la aplicación práctica de conocimientos adquiridos en el ciclo formativo (ciberseguridad, redes, programación) sino que también aporta un modelo escalable para entornos empresariales.

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar un sistema integral de detección, monitorización y respuesta ante amenazas de seguridad en redes informáticas, combinando tecnologías de IDS/IPS y análisis de logs.

1.3.2 Objetivos Específicos

1. Implementar Suricata para analizar tráfico de red y detectar intrusiones.
2. Configurar Wazuh para monitorizar endpoints y recopilar logs de seguridad.
3. Integrar Elasticsearch y Kibana para almacenar y visualizar datos.
4. Automatizar alertas mediante un bot de Telegram.
5. Explorar modelos de machine learning para identificar patrones anómalos y que aconsejen soluciones.
6. Validar el sistema en un entorno controlado con tres máquinas (servidor, cliente, atacante).

1.4 Estrategia y Planificación del Proyecto

Se ha optado por la estrategia de adaptar herramientas de código abierto para construir una solución integrada y personalizada. Esta estrategia permite aprovechar tecnologías probadas y reducir tiempos de desarrollo, enfocándose en la personalización y mejora del sistema.

- **Fase 1**: Investigación y selección de tecnologías.
- **Fase 2**: Configuración inicial de Suricata y Wazuh.
- **Fase 3**: Integración con Elasticsearch/Kibana y desarrollo del bot de Telegram.
- **Fase 4**: Implementación de modelos de machine learning y pruebas.
- **Fase 5**: Documentación y optimización.

1.5 Metodología de Trabajo:

Se seguirá una metodología ágil basada en Scrum, organizando el trabajo en sprints semanales. Se utilizarán herramientas como Trello para la gestión de tareas y este diagrama de Gantt para visualizar el progreso del proyecto:

También se utilizará GitHub para el control de versiones del código y documentación y organizaremos reuniones periódicas para evaluar avances y realizar los ajustes necesarios.

1.6 Estudio Económico y Presupuestario

Dado que el proyecto se basa en tecnologías de código abierto, los costos se centrarán en hardware y horas de desarrollo. Se usarán dos portátiles para la implementación y pruebas. El costo estimado incluye:

- **Infraestructura**: 0€ (máquinas virtuales en VirtualBox).
- **Software**: 0€ (tecnologías de código abierto).
- **Formación y documentación**: 0€ (recursos en línea como Google Drive y GitHub). El principal costo será el tiempo de desarrollo, estimado en unas 200 horas.

Capítulo 2: Descripción del Proyecto

2.1 Análisis de Requisitos

Requisitos Funcionales:

- Monitorizar tráfico de red en tiempo real (Suricata).
- Recopilar y analizar logs de endpoints (Wazuh).
- Visualizar datos mediante dashboards interactivos (Kibana).
- Enviar alertas automáticas vía Telegram.
- Detectar anomalías mediante Machine Learning.

Requisitos No Funcionales:

- **Robustez**: Tolerancia a fallos en componentes críticos.
- **Escalabilidad**: Capacidad de añadir nuevos nodos.
- **Seguridad**: Configuración de reglas de firewall (NFTABLES).

Requisitos Técnicos:

- **Hardware**: 2 portátiles con al menos 8GB de RAM y 100GB de almacenamiento. Mínimo 4GB de RAM por máquina virtual.
- **Software**: VirtualBox, ISOs de Ubuntu 22.04 Server y de Kali Linux.

2.2 Tecnologías

2.2.1 Comparativa de Tecnologías

Para la implementación de Heimdall IDS, se han evaluado diversas tecnologías que podrían cumplir con los requerimientos del sistema. A continuación, se presenta una comparativa de las soluciones consideradas para cada componente clave del proyecto.

1. Sistema de Detección de Intrusos (IDS/IPS)

Tecnología	Tipo	Ventajas	Desventajas
Suricata	IDS/IPS basado en firmas y anomalías	Detección en tiempo real, alto rendimiento, análisis profundo de paquetes, reglas personalizables	Requiere ajuste manual de reglas para evitar falsos positivos
Snort	IDS/IPS basado en firmas	Comunidad activa, reglas bien documentadas, bajo consumo de recursos	Menos eficiente con tráfico de alta velocidad en comparación con Suricata
Zeek (Bro)	IDS basado en análisis de tráfico	Análisis detallado de protocolos, scripting avanzado	No actúa como IPS, requiere configuraciones avanzadas

2. Gestión de Eventos de Seguridad (SIEM)

Tecnología	Ventajas	Desventajas
Wazuh	Integración con Elasticsearch, análisis de logs, respuesta automatizada, cumplimiento normativo	Puede generar un gran volumen de logs, requiere afinación de reglas
Splunk	Potente análisis de datos, dashboards avanzados	Software propietario, alto costo
Graylog	Gestión eficiente de logs, soporte para múltiples formatos	Menos integración con herramientas de detección activa

3. Almacenamiento y Visualización de Datos

Tecnología	Tipo	Ventajas	Desventajas
Elasticsearch/Kibana	Base de datos + visualización	Alta escalabilidad, indexación rápida, dashboards personalizables	Alto consumo de memoria, requiere tuning para grandes volúmenes de datos
Grafana	Visualización de datos	Interfaz intuitiva, compatible con múltiples fuentes de datos	Requiere integración con una base de datos externa
Splunk	Plataforma SIEM completa	Poderoso procesamiento de logs	Costoso y de código cerrado

4. **Inteligencia Artificial / Machine Learning**

Tecnología	Enfoque	Ventajas	Desventajas
Scikit-learn	Modelos de machine learning	Amplia variedad de algoritmos, fácil implementación	No está optimizado para Big Data en tiempo real
TensorFlow	Deep Learning	Gran capacidad para modelos complejos	Mayor consumo de recursos, requiere GPUs para alto rendimiento
PyTorch	Redes neuronales	Flexibilidad en experimentación	Menos maduro que TensorFlow en entornos de producción

2.2.2 Tecnologías Escogidas

Suricata

Suricata ha sido elegido como IDS/IPS debido a su capacidad de análisis en tiempo real, su compatibilidad con reglas personalizadas y su alto rendimiento en el análisis de tráfico de red. Permite detectar patrones maliciosos en paquetes de datos y generar alertas en caso de detectar actividad sospechosa.

Wazuh

Wazuh ha sido seleccionado como SIEM principal debido a su integración nativa con Elasticsearch, su capacidad de correlación de eventos y su funcionalidad auditoría de seguridad. Además, ofrece monitorización de logs y respuesta automatizada ante incidentes, facilitando la gestión centralizada de alertas.

Elasticsearch/Kibana

Elasticsearch y Kibana han sido elegidos por su potente capacidad de almacenamiento y visualización de datos en tiempo real. Elasticsearch permite almacenar grandes volúmenes de logs de eventos de seguridad, mientras que Kibana facilita la visualización mediante dashboards intuitivos. Esta combinación permitirá analizar tendencias y correlacionar alertas de manera eficiente.

Scikit-learn

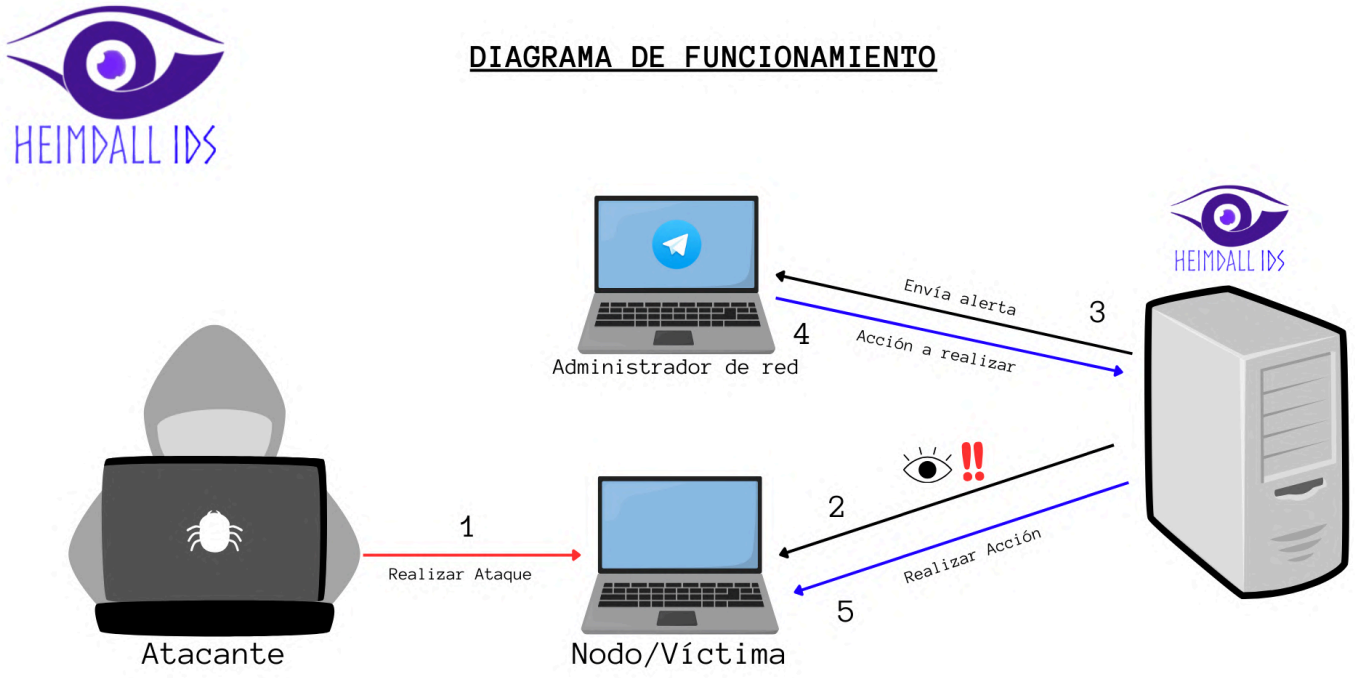
Para dotar a Heimdall de capacidades avanzadas de detección de amenazas, se usará Scikit-learn, una de las librerías más robustas de machine learning. Esto permitirá aplicar algoritmos de detección de anomalías y clasificar patrones sospechosos en los eventos registrados. Esto mejorará la capacidad del sistema para aprender y adaptarse a nuevas

amenazas con el tiempo. Puede que no se disponga de tiempo o del conocimiento necesario para añadir esta funcionalidad de detección de patrones, pero en un futuro sí que sería muy interesante implementarla.

2.3 Estructura del Proyecto

El sistema Heimdall IDS se estructura en tres nodos principales, cada uno con una función específica dentro de la detección, análisis y respuesta ante amenazas cibernéticas:

1. **Nodo Servidor (Heimdall IDS)**: Es el núcleo del sistema y se encarga de la monitorización, detección de intrusiones y gestión de alertas. Incluye los siguientes componentes:
    - **Suricata**
    - **Wazuh Manager**
    - **Elasticsearch/Kibana**
    - **Bot de Telegram**
    - **Machine Learning**
2. **Nodo Cliente**: Máquina donde se instala el agente Wazuh, que envía información al nodo servidor para su análisis. Esta máquina también hace de servidor de algún servicio como Apache, simulando ser un servidor real de una empresa o usuario.
3. **Nodo Atacante**: Se utiliza para pruebas de penetración y evaluación de la seguridad del sistema. Contará con herramientas como Metasploit y Nmap para simular ataques reales al nodo cliente.
4. **Portátil/PC/dispositivo móvil del administrador de red**: Aquí se reciben las notificaciones de Telegram y el administrador puede darle la orden al servidor de Wazuh para que este actúe (cierre puertos, bloquee IPs…).



Fases:

1. El atacante realiza el ataque a la víctima (escaneo de puertos, intento de sesión SSH…).
2. El servidor con Heimdall IDS está constantemente monitorizando a la víctima y detecta que la están atacando mediante unas reglas de Suricata previamente definidas.
3. El servidor con Heimdall IDS envía una alerta al ordenador del administrador de red a través del bot de Telegram.
4. El administrador de red decide que acción realizar a través del bot de Telegram.
5. El servidor con Heimdall IDS realiza la acción sobre la víctima (cierra puertos, bloquea IPs…).

2.4 Descripción de Componentes

- **Suricata**: Monitorea el tráfico de red en tiempo real, detecta y bloquea ataques basándose en reglas predefinidas.
- **Wazuh**: Recolecta y analiza logs del sistema, detectando posibles vulnerabilidades y configuraciones inseguras.
- **Elasticsearch/Kibana**: Almacena y visualiza datos de eventos en tiempo real, facilitando la identificación de patrones sospechosos.
- **Bot de Telegram**: Recibe alertas y permite interactuar con Heimdall IDS para gestionar respuestas a incidentes.
- **Machine Learning (Scikit-learn)**: Implementa algoritmos de aprendizaje automático para mejorar la detección de anomalías en la red y responder a amenazas.
- **Hydra/Nmap**: Utilizados en el nodo atacante para realizar pruebas de penetración y evaluar la efectividad de las medidas de seguridad.

2.5 Definición de las Tareas

Para la implementación de Heimdall IDS, se definirán las siguientes tareas principales:

1. Instalación y configuración de Suricata y Wazuh en el nodo servidor.
2. Integración de Elasticsearch/Kibana para la visualización de eventos.
3. Desarrollo del bot de Telegram y su vinculación con el sistema de alertas.
4. Entrenamiento del modelo de Machine Learning con datos de tráfico real y ataques simulados.
5. Implementación del agente Wazuh en el nodo cliente para la recolección de logs.
6. Simulación de ataques desde el nodo atacante para probar la efectividad del sistema.
7. Evaluación de rendimiento y ajuste de reglas de detección para mejorar la precisión del sistema.

2.5 Definición de Funcionalidades

- **Detección de Intrusiones**: Suricata analiza el tráfico y bloquea amenazas según las reglas definidas.
- **Alertas Automáticas**: El bot de Telegram envía notificaciones con detalles del incidente (dirección IP atacante, tipo de ataque detectado, medidas recomendadas).
- **Dashboards en Kibana**: Se crean paneles visuales para monitorizar eventos críticos, patrones de tráfico anómalo y tendencias de ataques.
- **Respuesta Activa**: A través del bot de Telegram, el usuario puede activar acciones como bloqueo de IPs, cierre de puertos o reconfiguración de reglas de seguridad.
- **Análisis Predictivo con Machine Learning**: Se aplican modelos de detección de anomalías para identificar ataques desconocidos o patrones sospechosos en la red.

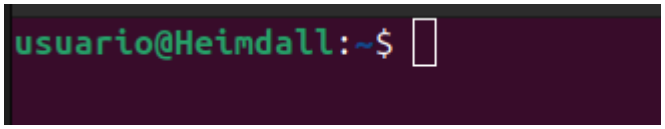
Capítulo 3: Creación del entorno y configuración básica

3.1 Creación de máquinas virtuales

1) Máquina **Ubuntu 22.04 Server** con las siguientes características:

Procesador: 2 cores.
RAM: 4GB.
Red: Adaptador puente.
Disco: 25GB (donde estará alojado el sistema operativo y el Suricata) y otro de 60GB vacío (donde tendremos Wazuh instalado).

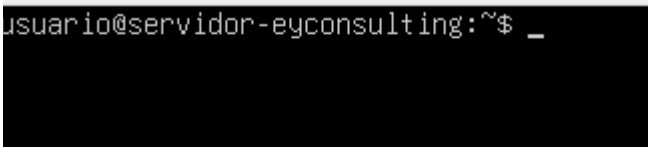
Se optó por utilizar **Ubuntu 22.04 Server** en lugar de versiones más recientes por varias razones. En primer lugar, la mayoría de la documentación y guías disponibles para Wazuh y Suricata están basadas en esta versión de Ubuntu. Además, existía incertidumbre respecto a posibles problemas de compatibilidad con dependencias y paquetes importantes en la versión 24.04. Se tenía la certeza de que todas las herramientas a emplear estaban completamente validadas y probadas en Ubuntu 22.04, lo que motivó la elección de dicha versión.



2) Máquina **Ubuntu 24.04 Server** que hará de víctima con las siguientes características:

Procesador: 2 cores
RAM: 1GB
Red: Adaptador puente
Disco: 20GB (donde estará alojado el sistema operativo y todos los servicios)

Dispondrá de servicios como **openssh-server**, **vsftpd, apache2** y **PHP**.



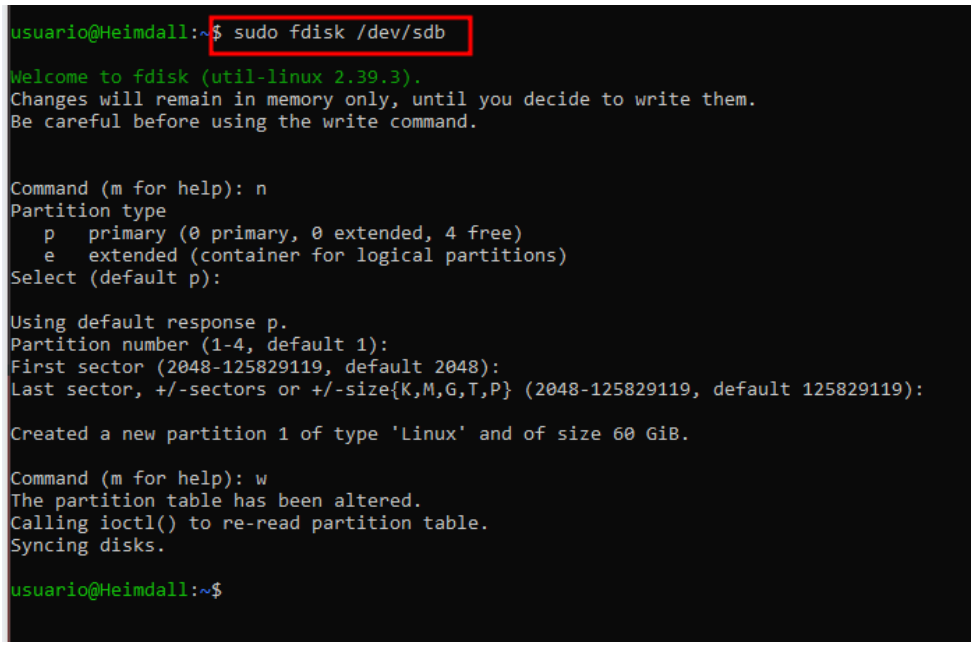
3) Máquina **Kali Linux** con las siguientes características:

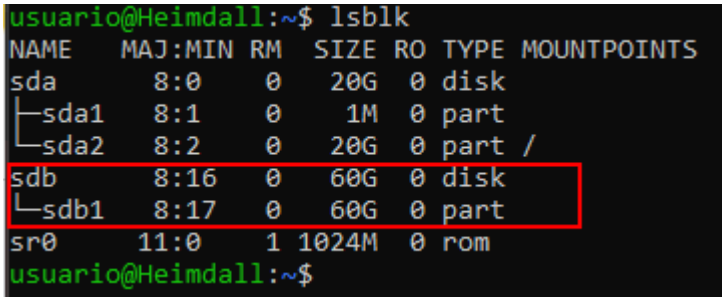
Procesador: 2 cores
RAM: 1G
Red: Adaptador puente
Disco: 20G (donde se instalará el sistema operativo Kali Linux)

Esta máquina será la encargada de realizar ataques sobre la víctima.

3.2 Instalación de programas y dependencias

En la máquina **Ubuntu Server 22.04** que hará de servidor de **Heimdall IDS** se tiene que instalar **Wazuh** y **Suricata**. Antes de instalar Wazuh se **particiona** y se **monta** en una carpeta el disco que se ha añadido de 60G para poder instalar ahí Wazuh.





Una vez montada la partición en la carpeta **`/mnt/heimdall`** se instala Wazuh.

1. Se descarga el archivo **`wazuh-install`** y se ejecuta para instalar Wazuh:

```
usuario@Heimdall:~$ cd /mnt/heimdall/
usuario@Heimdall:/mnt/heimdall$ sudo curl -sO https://packages.wazuh.com/4.10/wazuh-install.sh && sudo bash ./wazuh-i
nstall.sh -a
16/04/2025 18:12:04 INFO: Starting Wazuh installation assistant. Wazuh version: 4.10.1
16/04/2025 18:12:04 INFO: Verbose logging redirected to /var/log/wazuh-install.log
```

2. Se comprueba que está todo completamente instalado con los siguientes comandos:

```
usuario@Heimdall:/mnt/heimdall$ systemctl status wazuh-manager
● wazuh-manager.service - Wazuh manager
     Loaded: loaded (/lib/systemd/system/wazuh-manager.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2025-04-16 19:25:49 CEST; 2min 14s ago
      Tasks: 153 (limit: 4579)
     Memory: 1.3G
        CPU: 1min 7.807s
     CGroup: /system.slice/wazuh-manager.service
             ├─58486 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─58489 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─58493 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─58497 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─58535 /var/ossec/bin/wazuh-authd
             ├─58551 /var/ossec/bin/wazuh-db
```

```
usuario@Heimdall:/mnt/heimdall$ systemctl status wazuh-indexer
● wazuh-indexer.service - wazuh-indexer
     Loaded: loaded (/lib/systemd/system/wazuh-indexer.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2025-04-16 19:07:49 CEST; 20min ago
       Docs: https://documentation.wazuh.com
   Main PID: 5776 (java)
```

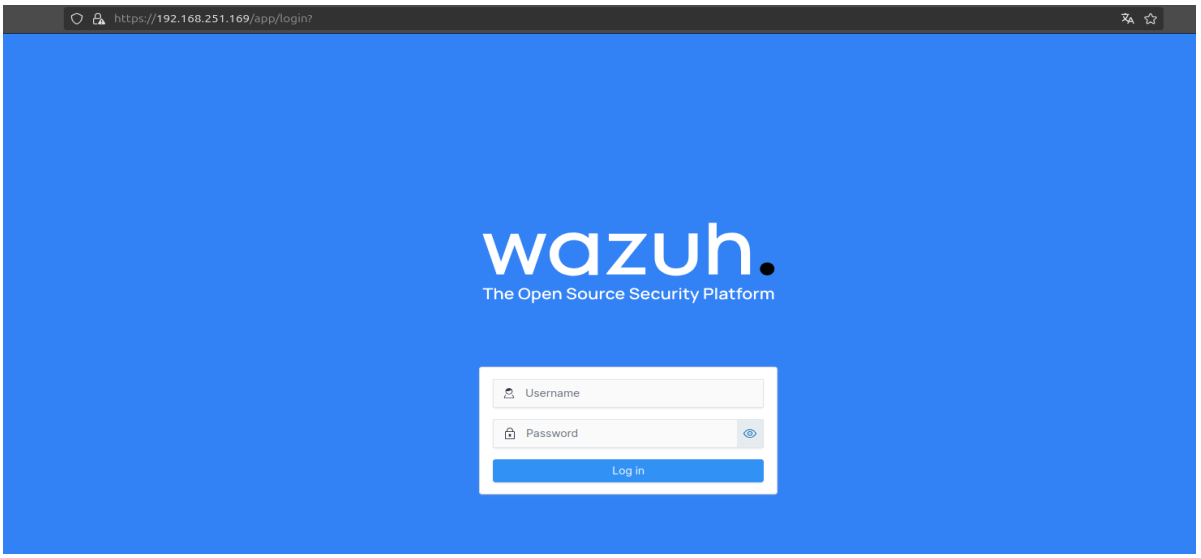
```
usuario@Heimdall:/mnt/heimdall$ systemctl status wazuh-dashboard
● wazuh-dashboard.service - wazuh-dashboard
     Loaded: loaded (/etc/systemd/system/wazuh-dashboard.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2025-04-16 19:25:56 CEST; 2min 15s ago
   Main PID: 59480 (node)
      Tasks: 11 (limit: 4579)
     Memory: 195.3M
        CPU: 24.782s
     CGroup: /system.slice/wazuh-dashboard.service
             └─59480 /usr/share/wazuh-dashboard/node/bin/node --no-warnings --max-http-header-size=65536

abr 16 19:26:29 Heimdall opensearch-dashboards[59480]: {"type":"log","@timestamp":"2025-04-16T17:26:29Z",
```

Wazuh Manager: Procesa, analiza y correlaciona los datos de seguridad enviados por los agentes.

Wazuh Indexer: Almacena y permite realizar búsquedas rápidas sobre los datos recopilados por el sistema.

Wazuh Dashboard: Proporciona una interfaz web para visualizar y gestionar la información de seguridad.

3. Si todo está correctamente se puede acceder al entorno web desde el navegador escribiendo la IP del propio servidor:



Suricata

Se procede con la instalación de Suricata.

```
usuario@Heimdall:/mnt/heimdall$ sudo apt update && sudo apt install suricata -y
Obj:1 https://packages.wazuh.com/4.x/apt stable InRelease
Obj:2 http://es.archive.ubuntu.com/ubuntu jammy InRelease
Obj:3 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease
Obj:4 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease
Obj:5 http://es.archive.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Se pueden actualizar 295 paquetes. Ejecute «apt list --upgradable» para verlos.
```

Se edita el fichero **`suricata.yaml`**, ubicado en **`/etc/suricata/`** para cambiar la interfaz de red que pone por defecto por la que se está usando (la cual tiene que estar en la misma red que la del servidor cliente y el administrador). De esta manera:

```
af-packet:
  - interface: enp0s3
    # Number of receive threads. "auto" uses the number of cores
    #threads: auto
    # Default clusterid. AF_PACKET will load balance packets based on
```

Con el comando **`sudo systemctl restart suricata`** se reinicia el servicio y se comprueba el estado de servicio, si está todo bien se procede a la configuración.

```
usuario@Heimdall:~$ sudo systemctl status suricata
● suricata.service - Suricata IDS/IDP daemon
     Loaded: loaded (/lib/systemd/system/suricata.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2025-04-16 19:40:22 CEST; 2s ago
       Docs: man:suricata(8)
             man:suricatasc(8)
             https://suricata-ids.org/docs/
    Process: 64246 ExecStart=/usr/bin/suricata -D --af-packet -c /etc/suricata/suricata.yaml --pidfile /run/suricata
   Main PID: 64247 (Suricata-Main)
```

Con el siguiente comando se comprueba que Heimdall puede ver qué tráfico pasa por la IP del cliente servidor:

```
usuario@Heimdall:~$ sudo tcpdump -i enp0s3 host 192.168.252.238
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), snapshot length 262144 bytes
17:25:43.839030 ARP, Request who-has 192.168.250.212 tell 192.168.252.238, length 46
17:25:44.766784 ARP, Request who-has 192.168.250.212 tell 192.168.252.238, length 46
17:25:45.702174 ARP, Request who-has 192.168.250.212 tell 192.168.252.238, length 46
```

Se ha capturado la paquetería que entra y sale por la IP del cliente servidor.

Se vuelve a editar el archivo **`/etc/suricata/suricata.yaml`** y se edita la siguiente sección para indicar cuál será el archivo que recopila los logs (en este caso el archivo **`eve-json`**):

```
# Extensible Event Format (nicknamed EVE) event log in JSON format
- eve-log:
    enabled: yes
    filetype: regular #regular|syslog|unix_dgram|unix_stream|redis
    filename: /var/log/suricata/eve.json
    # Enable for multi-threaded eve.json output; output files are amended wit
    # with an identifier, e.g., eve.9.json
    #threaded: false
    #prefix: "@cee: " # prefix to prepend to each log entry
    # the following are valid when type: syslog above
```

También es necesario editar el ***path*** del archivo de las reglas y escribir **`heimdall.rules`**. Este archivo será el que tendrá todas las reglas personalizadas para detectar posibles amenazas.

```
##

default-rule-path: /etc/suricata/rules

rule-files:
  - heimdall.rules
```

3.3 Creación de reglas personalizadas de Suricata

Las reglas de Suricata son instrucciones que le dicen al sistema qué tipo de tráfico de red debe detectar, cómo debe identificarlo y qué hacer cuando lo ve.

El objetivo es que Heimdall sea capaz de detectar amenazas específicas y alertar o actuar automáticamente. Las reglas permiten:

1. **Detectar actividad maliciosa**: como escaneos de puertos, fuerza bruta en SSH, SQL injection, etc.

2. **Filtrar y priorizar alertas**: por ejemplo, un intento de XSS puede ser prioridad media, pero un ataque de fuerza bruta es alto.

3. **Automatizar respuestas**: en combinación con Wazuh, Suricata y el bot de Telegram, el servidor podrá responder a esas alertas (bloquear IPs, cerrar puertos, etc).

Estas son las reglas que se han creado:

```
# 1. Detección de ping (ICMP) — prioridad media (3)

alert icmp any any -> 192.168.253.153 any (msg:"[HEIMDALL] Ping flood detectado"; sid:2000010; rev:1; threshold:type threshold, track by_src, count 100, seconds 10;)
```

Detecta si **una misma IP origen** envía más de **100 paquetes ICMP (ping)** en **10 segundos** hacia el host **192.168.253.153**.

- **`alert icmp any any -> 192.168.253.153 any`**: Aplica a cualquier tráfico ICMP (como *ping*) desde cualquier IP/puerto hacia el host 192.168.253.153 en cualquier puerto.
- **`msg:"[HEIMDALL] Ping flood detectado"`**: Mensaje que se mostrará en los logs o alertas.
- **`sid:2000010;`**: ID único de esta regla (Snort ID).
- **`rev:1;`**: Revisión de la regla.
- **`threshold:type threshold, track by_src, count 100, seconds 10;`**: Solo se activa si una misma IP origen manda más de 100 ICMP en 10 segundos (protege contra ping flood o ataques de denegación de servicio ICMP).

```
# 2. Posible intento de Cross-Site Scripting (XSS) – prioridad alta (2)

alert http any any -> 192.168.253.153 any (msg:"[HEIMDALL] Posible intento de XSS detectado"; content:"<script>"; nocase; sid:2000003; rev:1; priority:2;)
```

Esta regla detecta intentos de **Cross-Site Scripting (XSS)** en tráfico HTTP que contenga la etiqueta **`<script>`**.

- **`http`**: aplica al tráfico HTTP.
- **`content:"<script>"`**: busca el uso de scripts maliciosos (clásico en XSS).
- **`nocase`**: ignora mayúsculas/minúsculas.
- **`priority:2`**: alta prioridad, pero no crítica.
- **`sid/rev`**: identificador y versión de la regla.

```
# 3. Escaneo de puertos – prioridad media (3)

alert tcp any any -> 192.168.253.153 any (msg:"[HEIMDALL] Posible escaneo de puertos detectado"; flags:S; threshold:type threshold, track by_src, count 10, seconds 60; sid:2000004; rev:2; priority:3;)
```

Esta regla detecta **posibles escaneos de puertos** al observar múltiples paquetes TCP con flag SYN en poco tiempo.

- **`tcp`**: aplica solo al tráfico TCP.
- **`flags:S`**: busca paquetes con flag SYN, típicos de inicio de conexión.
- **`threshold`**: si una IP fuente envía ≥ 10 SYN en 60 segundos, se activa.
- **`track by_src`**: seguimiento por IP origen.
- **`priority:3`**: prioridad media, es común pero puede indicar reconocimiento previo a un ataque.
- **`sid/rev`**: identificador y versión.

```
# 4. Fuerza bruta en FTP – prioridad alta (2)

alert tcp any any -> 192.168.253.153 21 (msg:"[HEIMDALL] Posible ataque de fuerza bruta en FTP detectado"; flow:to_server,established; content:"USER"; nocase; threshold:type threshold, track by_src, count 5, seconds 60; sid:2000006; rev:1; priority:2;)
```

Esta regla detecta un posible **ataque de fuerza bruta en FTP**, es decir, intentos repetidos de autenticación con comandos USER para adivinar credenciales.

- **`tcp`**: se aplica a tráfico TCP.
- **`any any -> 192.168.253.153 21`**: cualquier IP/puerto de origen hacia puerto 21 (que es el de FTP) del cliente.
- **`flow:to_server,established`**: el paquete va hacia el servidor y está en una conexión establecida.
- **`content`**:"USER": busca el comando USER, típico de inicio de sesión en FTP.
- **`nocase`**: no distingue entre mayúsculas y minúsculas.
- **`threshold`**: si el mismo IP origen manda 5 intentos en 60 segundos, lanza la alerta.
- **`priority:2`**: se considera alerta importante.

```
# 5. Fuerza bruta en SSH – prioridad alta (2)

alert tcp any any -> 192.168.253.153 22 (msg:"[HEIMDALL] Posible ataque de Fuerza Bruta en SSH detectado"; flow:to_server,established; content:"SSH-"; nocase; threshold:type threshold, track by_src, count 5, seconds 60; sid:2000007; rev:1; priority:2;)
```

Esta regla detecta un posible **ataque de fuerza bruta contra el servicio SSH**, identificando múltiples intentos de conexión desde la misma IP en un corto período de tiempo.

- **`tcp`**: protocolo TCP, usado por SSH.
- **`any any -> 192.168.253.153 22`**: cualquier IP/puerto hacia puerto 22 (SSH) del cliente.
- **`flow:to_server,established`**: tráfico hacia el servidor con conexión establecida.
- **`content:"SSH-"`**: busca el inicio típico del banner SSH (SSH-2.0...), indicativo de conexión.

- **`nocase`**: sin distinguir mayúsculas.
- **`threshold`**: si una IP hace 5 conexiones en 60 segundos, lanza alerta.
- **`sid:2000007`**: identificador único.
- **`priority:2`**: alerta alta, pero no crítica.

Ataques que podría detectar Heimdall:

- **Escaneo de puertos**: Un atacante intenta descubrir puertos abiertos en un servidor para identificar vulnerabilidades explotables, usando herramientas como Nmap para mapear el sistema.
- **XSS (Cross-Site Scripting)**: Un atacante inyecta scripts maliciosos en sitios web, aprovechando vulnerabilidades en formularios o entradas de usuario, con el fin de robar información o ejecutar código en el navegador de las víctimas.
- **Ataques de fuerza bruta (SSH y FTP)**: Intentos automatizados de adivinar credenciales de acceso a servicios como SSH o FTP mediante repetidos intentos de inicio de sesión con múltiples combinaciones de usuario y contraseña.

Se puede proceder a validar la configuración:

```
sudo suricata -T -c /etc/suricata/suricata.yaml -i enp0s3
```

Suricata guarda los eventos en el archivo **`eve.json`**, para comprobar si está escribiendo ahí podemos ejecutar:

```
sudo tail -f /var/log/suricata/eve.json
```

Se puede hacer un ping o una prueba de escaneo (por ejemplo con **`nmap`**) para ver si Suricata lo detecta.

3.4 Editar la configuración de Wazuh para leer los logs de Suricata

Se abre el archivo de configuración:

```
nano /var/ossec/etc/ossec.conf
```

Añadir esta sección dentro del bloque principal **`<ossec_config>`**:

```
  <localfile>
    <log_format>json</log_format>
    <location>/var/log/suricata/eve.json</location>
  </localfile>

</ossec_config>
```

3.5 Configurar Suricata para que genere solo alertas compatibles con Wazuh

Editar el archivo de configuración de Suricata:

```
sudo nano /etc/suricata/suricata.yaml
```

Se busca la sección **`eve-log`** y dejamos únicamente lo siguiente:

```
# Extensible Event Format (nicknamed EVE) event log in JSON format
- eve-log:
    enabled: yes
    filetype: regular
    filename: /var/log/suricata/eve.json
    types:
      - alert:
          metadata: yes
          tagged-packets: no
```

Reiniciar los servicios para aplicar los cambios.

```
root@Heimdall:/mnt/heimdall# root@Heimdall:/mnt/heimdall# sudo systemctl restart suricata
root@Heimdall:/mnt/heimdall# sudo systemctl restart wazuh-manager
```

Verificar que todo funcione correctamente:

```
sudo tail -f /var/ossec/logs/ossec.log
```

```
2025/04/18 10:48:19 wazuh-modulesd:syscollector: INFO: Module started.
2025/04/18 10:48:19 wazuh-modulesd:syscollector: INFO: Starting evaluation.
2025/04/18 10:48:20 sca: INFO: Starting evaluation of policy: '/var/ossec/ruleset/sca/cis_ubuntu22-04.yml'
2025/04/18 10:48:20 wazuh-modulesd:syscollector: INFO: Evaluation finished.
2025/04/18 10:48:21 indexer-connector: INFO: IndexerConnector initialized successfully for index: wazuh-states-vulner
abilities-heimdall.
2025/04/18 10:48:23 wazuh-modulesd:vulnerability-scanner: INFO: Vulnerability scanner module started.
2025/04/18 10:48:39 wazuh-syscheckd: INFO: (6009): File integrity monitoring scan ended.
2025/04/18 10:48:39 wazuh-syscheckd: INFO: FIM sync module started.
2025/04/18 10:48:52 sca: INFO: Evaluation finished for policy '/var/ossec/ruleset/sca/cis_ubuntu22-04.yml'
2025/04/18 10:48:52 sca: INFO: Security Configuration Assessment scan finished. Duration: 33 seconds.
2025/04/18 10:49:39 wazuh-modulesd:vulnerability-scanner: INFO: Initiating update feed process.
2025/04/18 10:49:42 rootcheck: INFO: Ending rootcheck scan.
```

Instalar Elasticsearch:

```
root@Heimdall:/mnt/heimdall# sudo apt install elasticsearch
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

```
root@Heimdall:/mnt/heimdall# sudo systemctl enable --now elasticsearch
Created symlink /etc/systemd/system/multi-user.target.wants/elasticsearch.service → /lib/systemd/system/elasticsearc
.service.



root@Heimdall:/mnt/heimdall# sudo systemctl status elasticsearch
● elasticsearch.service - Elasticsearch
     Loaded: loaded (/lib/systemd/system/elasticsearch.service; enabled; vendor preset: enabled)
     Active: active (running) since Fri 2025-04-18 11:40:28 CEST; 4s ago
       Docs: https://www.elastic.co
   Main PID: 61240 (java)
      Tasks: 91 (limit: 4579)
     Memory: 1.8G
        CPU: 1min 34.691s
     CGroup: /system.slice/elasticsearch.service
             ├─61240 /usr/share/elasticsearch/jdk/bin/java -Xms4m -Xmx64m -XX:+UseSerialGC -Dcli.name=server -Dcli.s
             ├─61303 /usr/share/elasticsearch/jdk/bin/java -Des.networkaddress.cache.ttl=60 -Des.networkaddress.cach
             └─61323 /usr/share/elasticsearch/modules/x-pack-ml/platform/linux-x86_64/bin/controller

abr 18 11:38:11 Heimdall systemd[1]: Starting Elasticsearch...
abr 18 11:40:28 Heimdall systemd[1]: Started Elasticsearch.
lines 1-15/15 (END)
```

Para integrar Wazuh Manager con Elasticsearch en vez de con Wazuh Indexer se debe editar el archivo **`/var/ossec/etc/ossec.conf`** y dentro de la sección **`<ossec_config>`** se edita el bloque **`<indexer>`** para apuntar al endpoint de Elasticsearch:

```
<indexer>
  <enabled>yes</enabled>
  <hosts>
    <host>https://127.0.0.1:9200</host>
  </hosts>
  <ssl>
  <certificate_authorities>
      <ca>/etc/filebeat/certs/root-ca.pem</ca>
    </certificate_authorities>
    <certificate>/etc/filebeat/certs/filebeat.pem</certificate>
    <key>/etc/filebeat/certs/filebeat-key.pem</key>
  </ssl>
</indexer>
```

3.5 Conectar cliente con Heimdall IDS

Primero se descargará la clave pública GPG del repositorio de Wazuh, esta clave sirve para verificar que los paquetes que se descargan son legítimos. Luego se convertirá la clave a un formato compatible con APT y se guardará en **`/usr/share/keyrings/wazuh.gpg`**:

```
root@servidor-eyconsulting:/home/usuario# curl -s https://packages.wazuh.com/key/GPG-KEY-WA
ZUH | sudo gpg --dearmor -o /usr/share/keyrings/wazuh.gpg
```

El siguiente comando crea una línea con la URL del repositorio oficial de Wazuh y guarda esa línea dentro de una archivo **.list**, que es donde APT busca nuevos repositorios:

```
root@servidor-eyconsulting:/home/usuario# echo "deb [signed-by=/usr/share/keyrings/wazuh.gpg] https://packages.wazuh.com/4.x/apt/ stable main" | sudo tee /etc/apt/sources.list.d/wazuh.list
```

Finalmente, se actualiza el índice de paquetes del sistema para que APT reconozca el nuevo repositorio de Wazuh y detecte los paquetes disponibles. Seguidamente, se instala el paquete **wazuh-agent**, agente encargado de ejecutarse en el cliente y transmitir información al Wazuh Manager del servidor Heimdall.

```
root@servidor-eyconsulting:/home/usuario# apt update && apt install wazuh-agent
```

Una vez ya se ha instalado **wazuh-agent** se configura el cliente para que se conecte al servidor Heimdall IDS. Para hacerlo se edita el siguiente archivo de configuración:

```
root@servidor-eyconsulting:/home/usuario# nano /var/ossec/etc/ossec.conf
```

Se busca la sección **<client>** y dentro de **<server>** se pone la IP del servidor Heimdall:

```
<ossec_config>
  <client>
    <server>
      <address>192.168.252.79</address>
      <port>1514</port>
      <protocol>tcp</protocol>
    </server>
```

Ahora en el servidor Heimdall se ejecutará el siguiente comando para obtener la clave del cliente:

```
usuario@Heimdall:~$ sudo /var/ossec/bin/manage_agents
[sudo] password for usuario:


****************************************
* Wazuh v4.11.2 Agent manager.         *
* The following options are available: *
****************************************
   (A)dd an agent (A).
   (E)xtract key for an agent (E).
   (L)ist already added agents (L).
   (R)emove an agent (R).
   (Q)uit.
Choose your action: A,E,L,R or Q: E

Available agents:
   ID: 001, Name: cliente1, IP: any
   ID: 002, Name: servidor-eyconsulting, IP: any
Provide the ID of the agent to extract the key (or '\q' to quit): 002

Agent key information for '002' is:
MDAyIHNlcnZpZG9yLWV5Y29uc3VsdGluZyBhbnkgODc4NzUyMjQzMGRlOTQ1ZDAxYmM5NWIwM2I3M2MwNjQyYWZjZTZ
lOGY0ZmZkYTQ2ZGE5MmIwMTY0M2RmMzNhYw==

** Press ENTER to return to the main menu.
```

Se pueden crear nuevos agentes, eliminar los existentes, listarlos o extraer la clave de alguno de ellos. En este caso, se opta por extraer la clave del cliente (tecla E).

En el equipo cliente, se ejecuta el mismo comando utilizado en el servidor Heimdall ey se importa la clave (tecla I):

```
root@servidor-eyconsulting:/home/usuario# /var/ossec/bin/manage_agents


****************************************
* Wazuh v4.11.2 Agent manager.         *
* The following options are available: *
****************************************
   (I)mport key from the server (I).
   (Q)uit.
Choose your action: I or Q: I

* Provide the Key generated by the server.
* The best approach is to cut and paste it.
*** OBS: Do not include spaces or new lines.

Paste it here (or '\q' to quit): MDAyIHNlcnZpZG9yLWV5Y29uc3VsdGluZyBhbnkgODc4NzUyMjQzMGRlOT
Q1ZDAxYmM5NWIwM2I3M2MwNjQyYWZjZTZlOGY0ZmZkYTQ2ZGE5MmIwMTY0M2RmMzNhYw==

Agent information:
   ID:002
   Name:servidor-eyconsulting
   IP Address:any

Confirm adding it?(y/n): y
Added.


****************************************
* Wazuh v4.11.2 Agent manager.         *
* The following options are available: *
****************************************
   (I)mport key from the server (I).
   (Q)uit.
Choose your action: I or Q: q

manage_agents: Exiting.
```

Por último, se activa el servicio **wazuh-agent**:

```
root@servidor-eyconsulting:/home/usuario# systemctl daemon-reexec
root@servidor-eyconsulting:/home/usuario# systemctl enable --now wazuh-agent
root@servidor-eyconsulting:/home/usuario# systemctl status wazuh-agent
● wazuh-agent.service - Wazuh agent
     Loaded: loaded (/usr/lib/systemd/system/wazuh-agent.service; enabled; preset: enabled)
     Active: active (running) since Wed 2025-04-23 15:16:32 CEST; 32min ago
      Tasks: 31 (limit: 1068)
     Memory: 19.0M (peak: 33.9M)
        CPU: 13.433s
     CGroup: /system.slice/wazuh-agent.service
             ├─1544 /var/ossec/bin/wazuh-execd
             ├─1555 /var/ossec/bin/wazuh-agentd
             ├─1567 /var/ossec/bin/wazuh-syscheckd
             ├─1579 /var/ossec/bin/wazuh-logcollector
             └─1593 /var/ossec/bin/wazuh-modulesd
```

3.6 Heimdall IDS detecta las alertas del cliente siendo atacado

En un entorno real de red empresarial, un sistema de detección de intrusos (IDS) como Suricata se ubicaría en una de las dos ubicaciones siguientes:

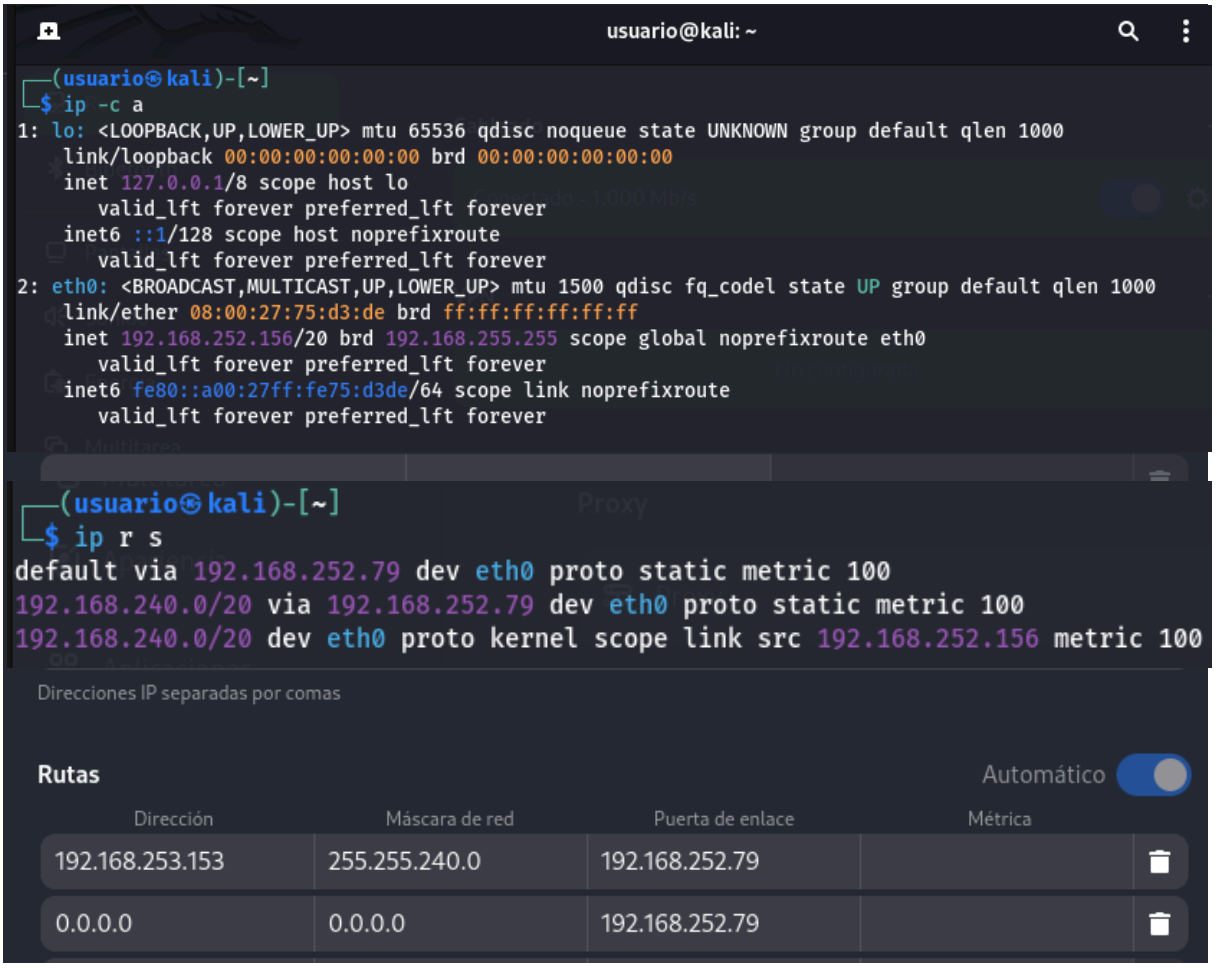
- En el router, analizando la totalidad del tráfico entrante y saliente de la red.
- En un switch gestionable con port mirroring (SPAN port), duplicando el tráfico hacia el IDS.

De este modo, todo el tráfico intercambiado entre usuarios, servicios y posibles atacantes resulta copiado o enrutado al IDS, sin necesidad de desplegar agentes en cada dispositivo.

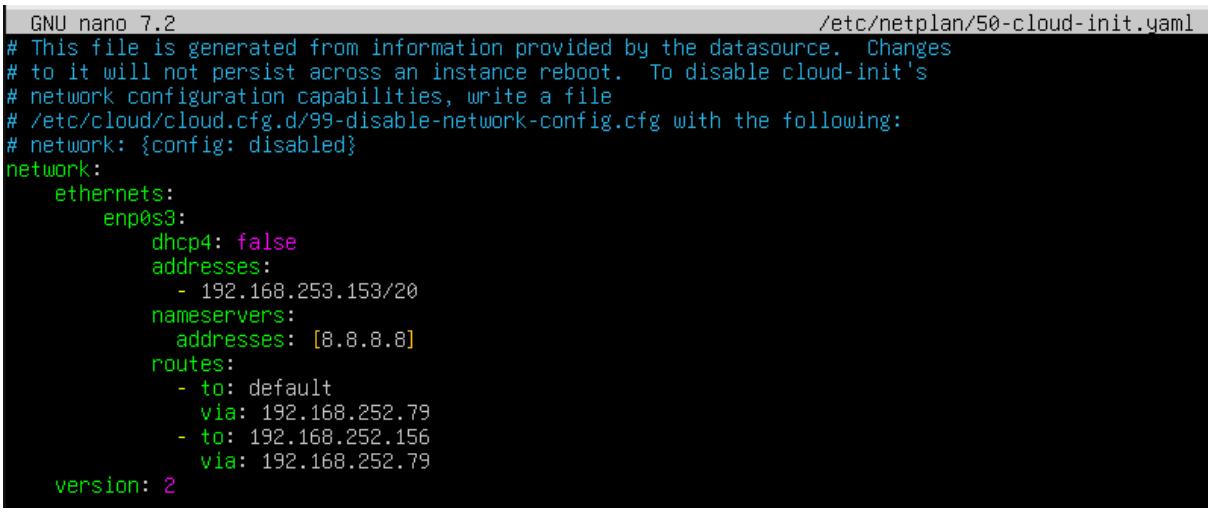
Dado que el entorno de pruebas utiliza máquinas virtuales y carece de router físico o switches gestionables, se optó por simular la posición intermedia del IDS mediante la modificación de las rutas de red. El objetivo consistió en forzar que el tráfico procedente del atacante hacia la víctima transitara por el servidor Heimdall, de modo que Suricata pudiera analizarlo.

En ambas máquinas (atacante y víctima), se ajustó la ruta por defecto para enviar el tráfico a través del servidor Heimdall, en lugar de dirigirlo directamente al gateway real de la red.

Atacante:



Cliente:



```
usuario@servidor-eyconsulting:~$ ip -c a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:da:0a:f1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.253.153/20 brd 192.168.255.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feda:af1/64 scope link
       valid_lft forever preferred_lft forever
usuario@servidor-eyconsulting:~$
```

```
usuario@servidor-eyconsulting:~$ ip r s
default via 192.168.252.79 dev enp0s3 proto static
192.168.240.0/20 dev enp0s3 proto kernel scope link src 192.168.253.153
192.168.252.156 via 192.168.252.79 dev enp0s3 proto static
```

⚠️ Consideraciones de Seguridad

- Este método modifica las rutas de red, lo cual no sería recomendable en un entorno de producción salvo en entornos controlados.

- En un entorno profesional, lo ideal sería un switch o router gestionable o ubicar el IDS directamente entre la red interna y la red externa.

- En un entorno profesional, podríamos usar herramientas como pfSense, OpenWRT, VLANs o firewalls físicos para enrutar el tráfico hacia el IDS de forma más robusta y automatizada.

Como resultado se puede ver que el servidor Heimdall IDS detecta las alertas del cliente.

Por ejemplo, si el atacante realiza un escaneo de puertos de la víctima:

```
┌──(usuario㉿kali)-[~]
└─$ nmap -sS -p 1-20 192.168.253.153
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-25 18:28 CEST
Nmap scan report for 192.168.253.153
Host is up (0.062s latency).

PORT   STATE  SERVICE
1/tcp  closed tcpmux
2/tcp  closed compressnet
3/tcp  closed compressnet
4/tcp  closed unknown
5/tcp  closed rje
6/tcp  closed unknown
7/tcp  closed echo
8/tcp  closed unknown
9/tcp  closed discard
10/tcp closed unknown
11/tcp closed systat
12/tcp closed unknown
13/tcp closed daytime
14/tcp closed unknown
15/tcp closed netstat
16/tcp closed unknown
17/tcp closed qotd
18/tcp closed msp
19/tcp closed chargen
20/tcp closed ftp-data

Nmap done: 1 IP address (1 host up) scanned in 1.07 seconds
```

En Heimdall IDS se observa que la alerta ha sido detectada, proporcionando información sobre la IP de destino (la víctima), la IP de origen (el atacante), la prioridad y el mensaje generado. En este caso, el mensaje registrado corresponde a "[HEIMDALL] Posible escaneo de puertos detectado", tal como se definió en la regla.


```
{"timestamp":"2025-04-25T18:29:48.033996+0200","flow_id":1191448497915084,"in_iface":"enp0s
3","event_type":"alert","src_ip":"192.168.252.156","src_port":55977,"dest_ip":"192.168.253.
153","dest_port":15,"proto":"TCP","alert":{"action":"allowed","gid":1,"signature_id":200000
4,"rev":1,"signature":"[HEIMDALL] Posible escaneo de puertos detectado","category":"","seve
rity":3},"flow":{"pkts_toserver":1,"pkts_toclient":0,"bytes_toserver":60,"bytes_toclient":0
,"start":"2025-04-25T18:29:48.033996+0200"}}
```

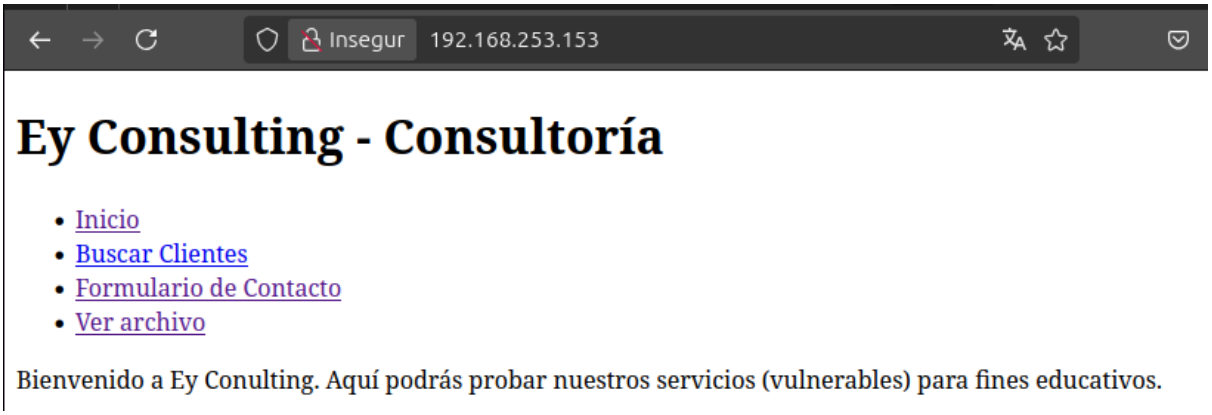

3.7 Web del cliente

La máquina cliente, configurada para simular el servidor de la empresa, dispone de una página web básica desarrollada en PHP. Dicha página incluye funcionalidades habituales, como subida de archivos y formularios, que permiten realizar pruebas de ataques y verificar la capacidad de Heimdall para detectarlos y responder. Además, en este equipo se habilitan los servicios SSH y FTP, facilitando la ejecución de ataques de fuerza bruta y escaneos de puertos.

La descripción detallada del proceso de creación de la web y de la configuración de los servicios se encuentra recopilada en el anexo correspondiente:

Configurar página web y servicios

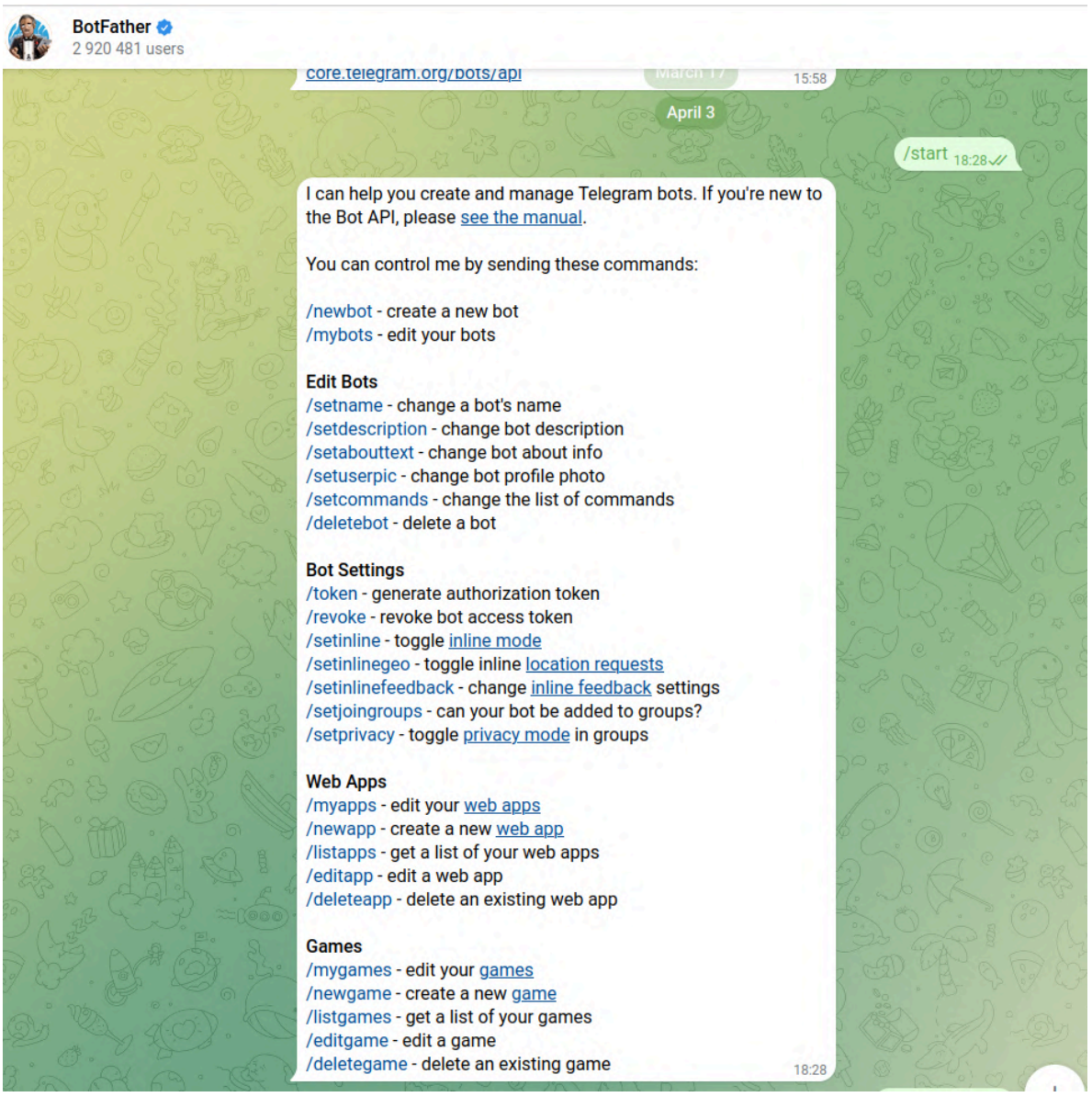
Esta es la web que se ha creado:



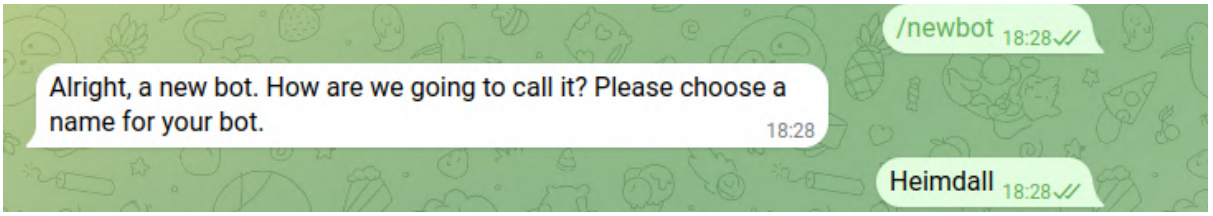
Es una web muy pero que muy básica pero que sirve para realizar ataques XSS (ya que se pueden subir archivos).

3.8 Crear bot en Telegram

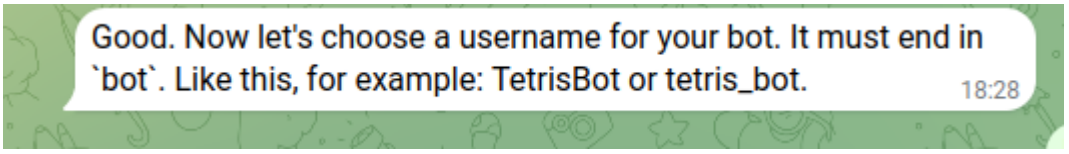
Se accede a la aplicación de Telegram, se busca el bot denominado **@BotFather** y se envía el comando **`/start`**.

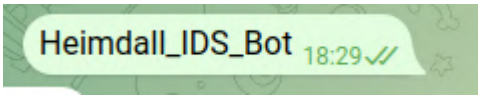


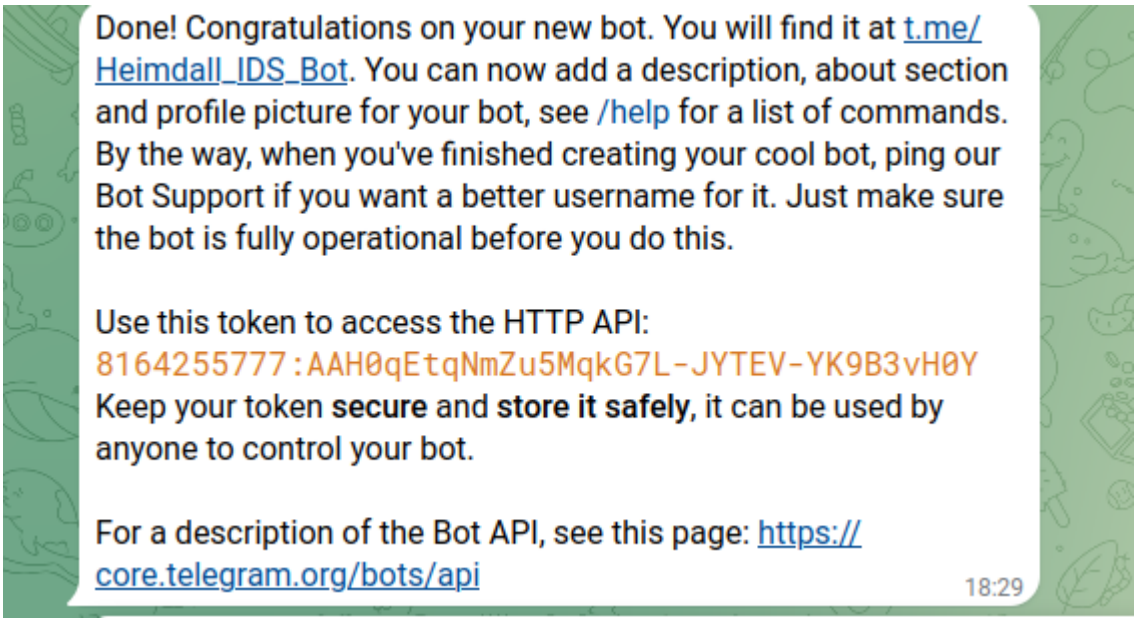
Se utiliza el comando **`/newbot`** para crear uno nuevo y solicita el nombre del bot.



A continuación, se introduce el nombre de usuario, el cual debe terminar necesariamente con la palabra **bot**:







El token generado debe conservarse, ya que resultará necesario para la conexión desde el script desarrollado en Python.

Asimismo, se requiere el token de acceso a la API RESTful de Wazuh, la cual permite interactuar con el sistema mediante scripts o aplicaciones externas (por ejemplo, el bot creado) para visualizar las alertas de la víctima, ejecutar acciones como bloqueo de IPs, búsqueda de patrones sospechosos y envío de notificaciones a través de Telegram.

Para obtener este token, es preciso autenticarse con un usuario válido. Las credenciales de dicho usuario se encuentran en el archivo siguiente:

```
$ sudo nano /usr/share/wazuh-dashboard/data/wazuh/config/wazuh.yml
```

```
hosts:
  - default:
      url: https://127.0.0.1
      port: 55000
      username: wazuh-wui
      password: "0cT91aeFF5?YEl5eF4RZjOwumC6312bj"
      run_as: false
```

Una vez obtenidas las credenciales, se ejecuta un comando **curl** a la API de Wazuh para autenticarse con el usuario **wazuh-ui** y almacenar el resultado (el token) en una variable denominada **TOKEN**. Al ejecutar **echo $TOKEN**, se muestra el contenido de dicha variable, que corresponde al token necesario.

```
usuario@Heimdall:~$ TOKEN=$(curl -u wazuh-wui:0cT91aeFF5?YEl5eF4RZjOwumC6312bj -k -X POST "
https://localhost:55000/security/user/authenticate?raw=true")
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   404  100   404    0     0   1576      0 --:--:-- --:--:-- --:--:--  1571
usuario@Heimdall:~$ echo $TOKEN
eyJhbGciOiJFUzUxMiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ3YXp1aCIsImF1ZCI6IldhenVoIEFQSSBSRVNUIiwibm
JmIjoxNzQ1NTk5NjMyLCJleHAiOjE3NDU2MDA1MzIsInN1YiI6IndhenVoLXd1aSIsInJ1bl9hcyI6ZmFsc2UsInJiY
WNfcm9sZXMiOlsxXSwicmJhY19tb2RlIjoid2hpdGUifQ.AcTT-DCOkNd6BNbawRzt9TfBY5YE2DfcEQEBP7-o1mVeI
PSdZxb661ixuw7UA9zQksMIUoFvI_-0riKct7zxJ4zVASd3Qixpk9_LhYAbug1hNidsFCUtYqU4bmwiOC-MkpIDh2jN
RJjXAtCD5UYP7NTk9y7kD-apFTGuk9APj97oW7Qx
usuario@Heimdall:~$
```

Para comprobar la capacidad de realizar consultas a la API de Wazuh, se elabora un script en Python basado en el ejemplo proporcionado en la documentación oficial: https://documentation.wazuh.com/current/user-manual/api/getting-started.html

Script de prueba

Básicamente realiza consultas a la API para saber información sobre el servidor, si hace una consulta exitosa se observa algo como esto:

```
usuario@Heimdall:~$ python3 prueba.py

Login request ...

eyJhbGciOiJFUzUxMiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ3YXp1aCIsImF1ZCI6IldhenVoIEFQSSBSRVNUIiwibm
JmIjoxNzQ1NjAyMjU3LCJleHAiOjE3NDU2MDMxNTcsInN1YiI6IndhenVoLXd1aSIsInJ1bl9hcyI6ZmFsc2UsInJiY
WNfcm9sZXMiOlsxXSwicmJhY19tb2RlIjoid2hpdGUifQ.ARqIkWUuYeFDVh4wO4xx6O2UnTBBospmVlnGJ-oe9yv5N
6ZdoE43grZ61EodZglcf--6d357cVWpNZZ29wwMHCktAQg14YRDroKtsyVXZTHJgeL0H2Qb0915ZttIBzGO2yP9YqxU
PUoRyvLnsBlyQg2hHnVWiclxQSAXxroYBREoDJJ5

- API calls with TOKEN environment variable ...

Getting API information:
{
   "data": {
      "title": "Wazuh API REST",
      "api_version": "4.11.2",
      "revision": 41122,
      "license_name": "GPL 2.0",
      "license_url": "https://github.com/wazuh/wazuh/blob/v4.11.2/LICENSE",
      "hostname": "Heimdall",
      "timestamp": "2025-04-25T17:30:57Z"
   },
   "error": 0
}

Getting agents status summary:
{
   "data": {
      "connection": {
         "active": 1,
         "disconnected": 1,
         "never_connected": 0,
         "pending": 0,
         "total": 2
      },
      "configuration": {
         "synced": 2,
         "total": 2,
         "not_synced": 0
      }
   },
   "error": 0
}

End of the script.
```

En este caso, se ve la información de los agentes de Wazuh: existen dos agentes, aunque únicamente uno permanece conectado (el configurado para simular el servidor de la empresa).

Una vez verificado su funcionamiento, se procede al desarrollo del bot definitivo, al que se denominará **`heimdall_bot.py`**.

3.9 Bot de Telegram

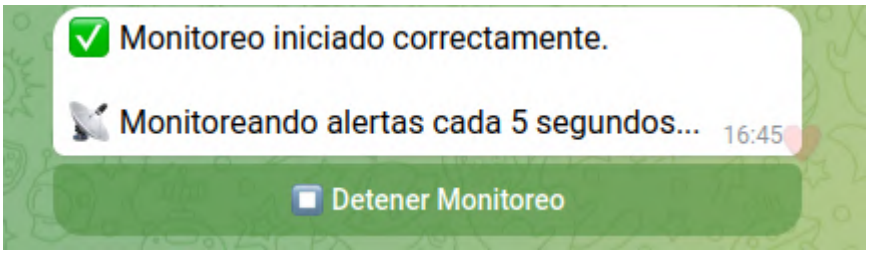
Toda la parte de programación y código, al ser más extensa, está en el siguiente apartado del anexo:

Código del bot

Al iniciar el bot mediante el comando **`/start`**, se solicita la contraseña para su uso, en caso de introducirla incorrectamente se retorna un error y se requiere reingresar la credencial. Al proporcionar la contraseña de forma adecuada, se despliega el menú principal con varios botones disponibles. A continuación, se describen dichos botones:

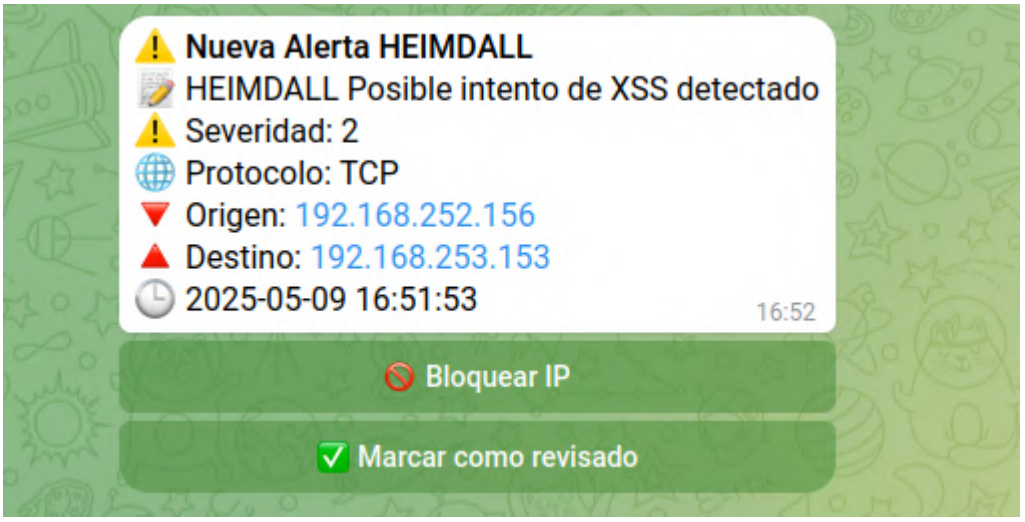
- Iniciar Monitoreo
- Desbloquear IP
- Ver IPs bloqueadas
- Gestionar Blacklist
- Estado de Agentes
- Estadísticas

Iniciar Monitoreo:



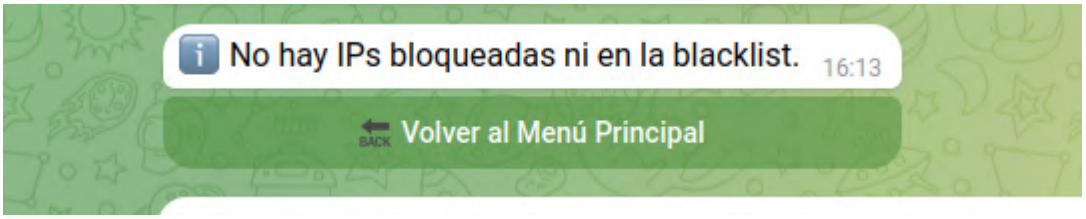
Se inicia la monitorización que, cada cinco segundos, analiza los registros en busca de nuevas alertas. Al detectarse una, se formatea separando las secciones y se muestra en pantalla junto con diversas opciones. Para detener la monitorización, puede pulsarse el botón "**Detener Monitoreo**" o ejecutar el comando **`/stop_monitoring`.**

Cuando salta una alerta aparece un mensaje indicando el tipo de ataque que se está produciendo, el nivel de severidad, el protocolo, la dirección IP de origen (el atacante) y de destino (el cliente) y la fecha.



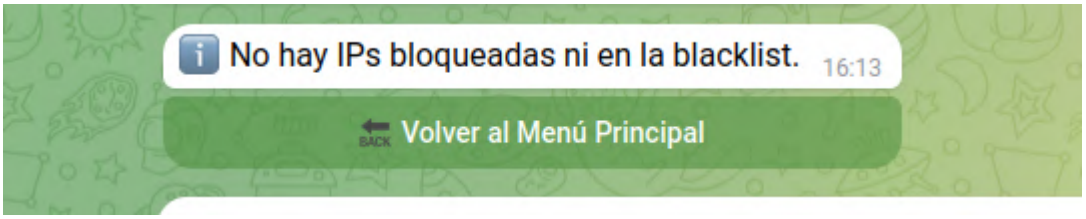
En este apartado se dispone de dos botones principales. El primero, **"Bloquear IP"**, permite aplicar reglas de **NFTABLES** para bloquear la dirección IP de origen (correspondiente al atacante), impidiendo que pueda seguir ejecutando ataques contra la víctima. El segundo botón, **"Marcar como revisado"**, resulta útil una vez se ha revisado una alerta, ya que permite indicar que esta ya ha sido gestionada.

Desbloquear IP:



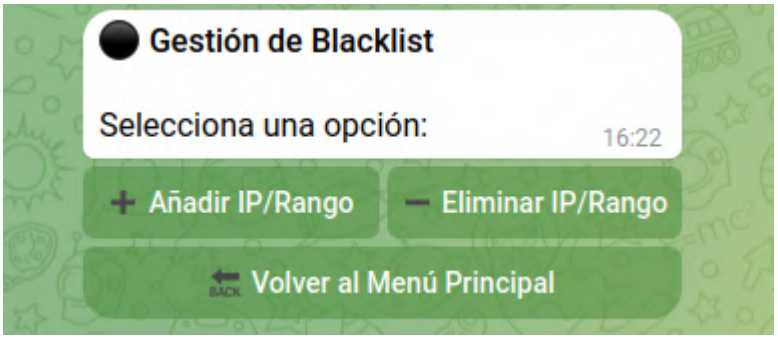
Con este botón, tal y como su nombre indica, desbloquea direcciones IPs previamente bloqueadas en el menú de monitorización. ¿Por qué sería de interés desbloquear una IP? Puede ser útil poder desbloquear una IP si se ha bloqueado por error o, si una vez bloqueada y analizada, se entiende que no hay peligro y se quiere que esa máquina pueda comunicarse con el cliente.

Ver IPs Bloqueadas:



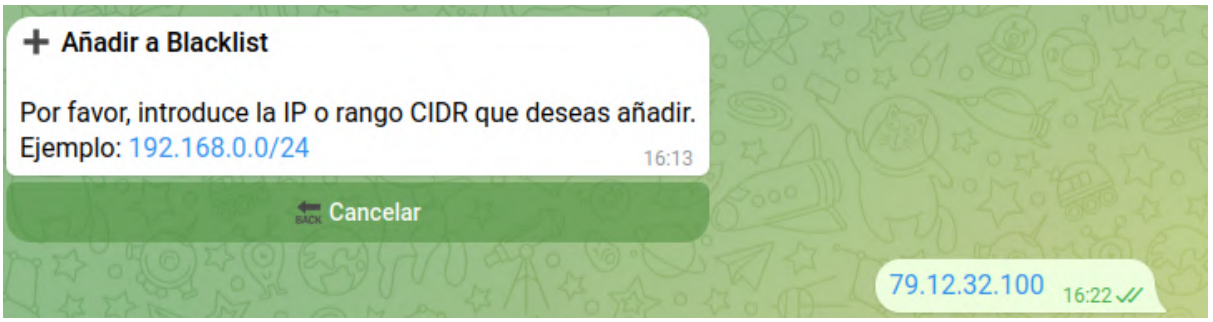
En este apartado está la lista de las direcciones IPs bloqueadas, ya sean direcciones IPs bloqueadas en el menú de monitoreo o añadidas manualmente. Este botón se puede usar para tener un control sobre las direcciones que se han bloqueado.

Gestión de Blacklist:

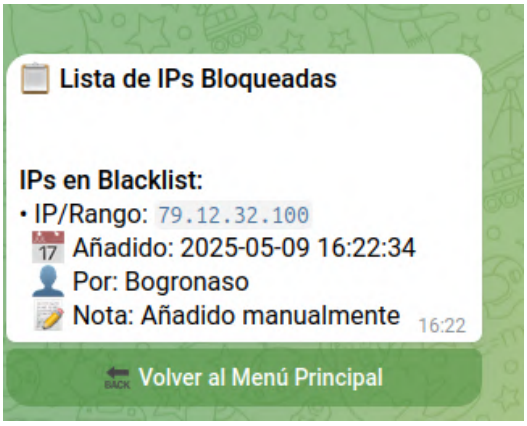


En este menú se pueden añadir direcciones IPs manualmente a la lista de IPs bloqueadas o eliminar las que añadimos manualmente. Esta lista de direcciones bloqueadas se denomina **Blacklist**.

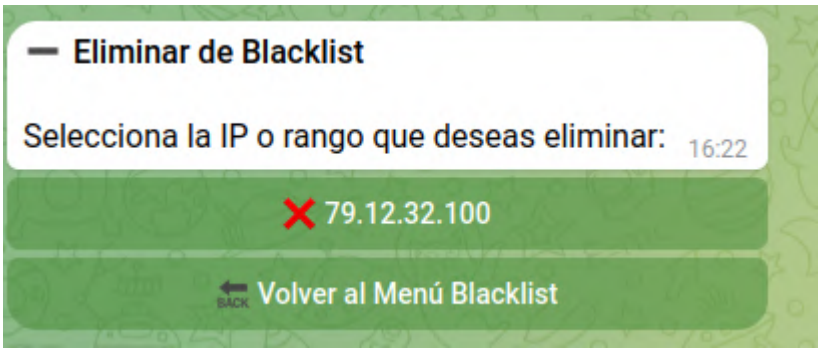
Si se pulsa el botón de **"Añadir IP/Rango"** aparece el siguiente enunciado:



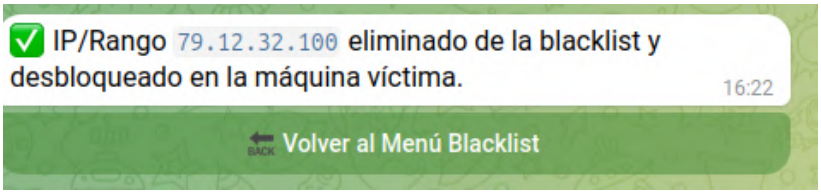
Se escribe la dirección que se quiere bloquear, entonces ejecuta las reglas de NFTABLES para bloquearla y si se vuelve a mirar la lista de IPs bloqueadas sale la IP bloqueada, la fecha en que se añadió, el usuario y una nota:



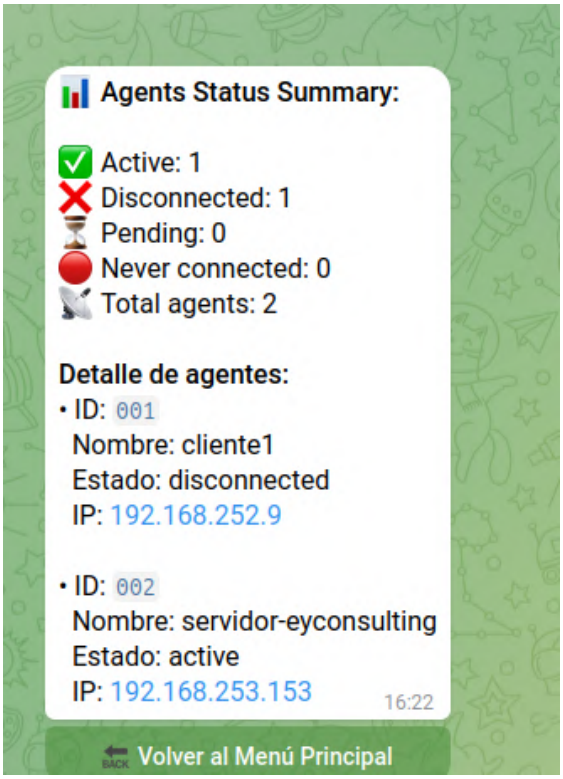
En la función de **"Eliminar de Blacklist"** salen las direcciones IP previamente bloqueadas y se pueden eliminar para que ya no formen parte de la blacklist.



Para eliminar una IP de la blacklist es tan simple como clicar encima de la IP que se desea borrar:



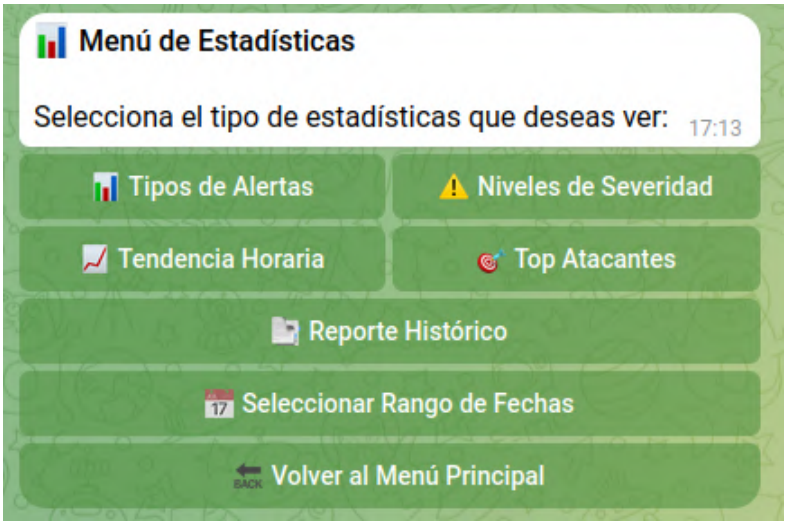
Estado de agentes



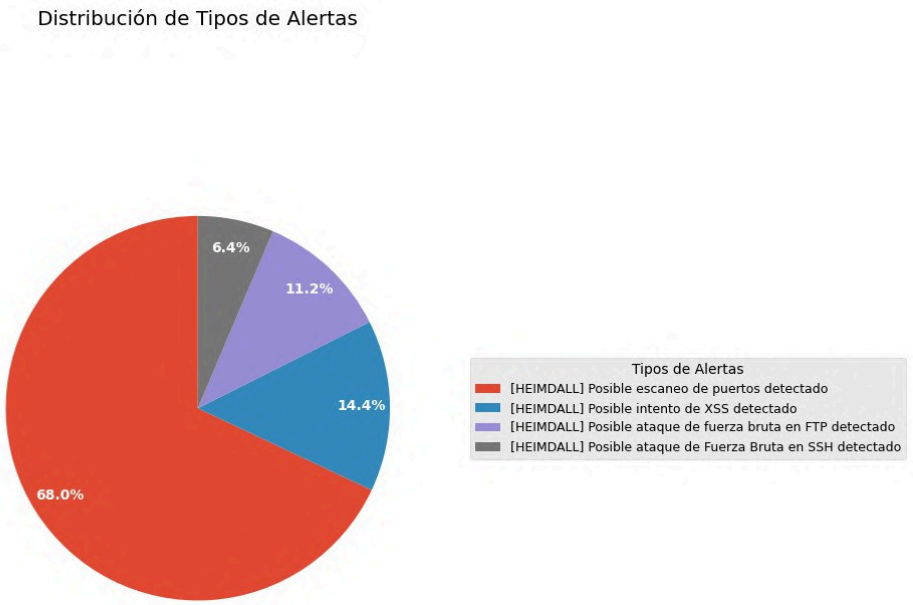
Con el botón de **"Estado de agentes"** aparece cuántos agentes hay en total, los que están activos, los desconectados, los que están pendientes de vincularse, los que nunca estuvieron activos y los detalles de cada uno (como el ID, el nombre, su estado y su dirección IP).

Estadísticas

Con el botón de "**Estadísticas**" aparecen una serie de botones para poder ver cierto tipo de estadísticas, también permite ver un informe e incluso seleccionar un rango de fechas concreto para ver las estadísticas:

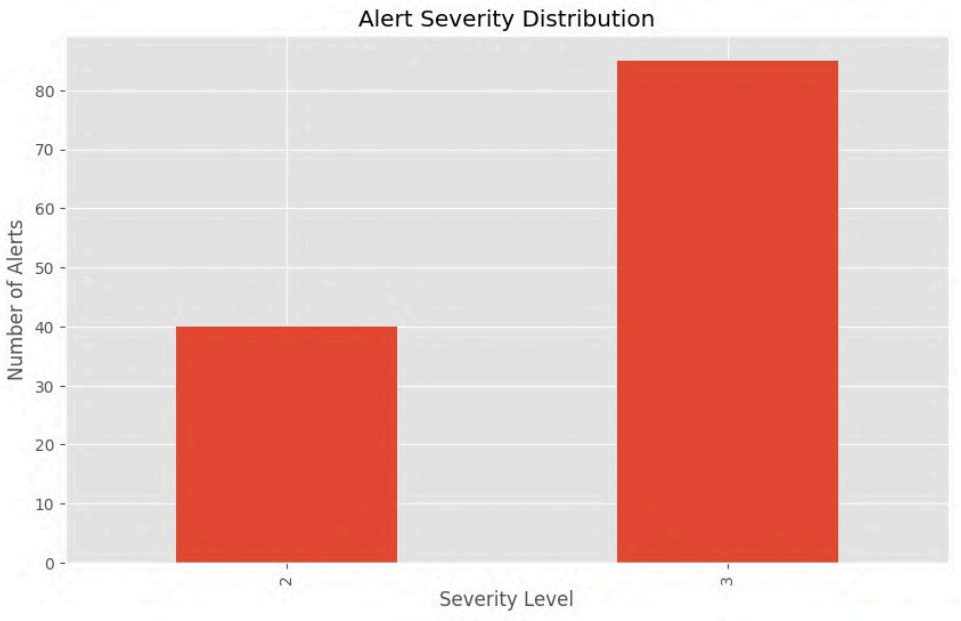


Tipos de Alertas



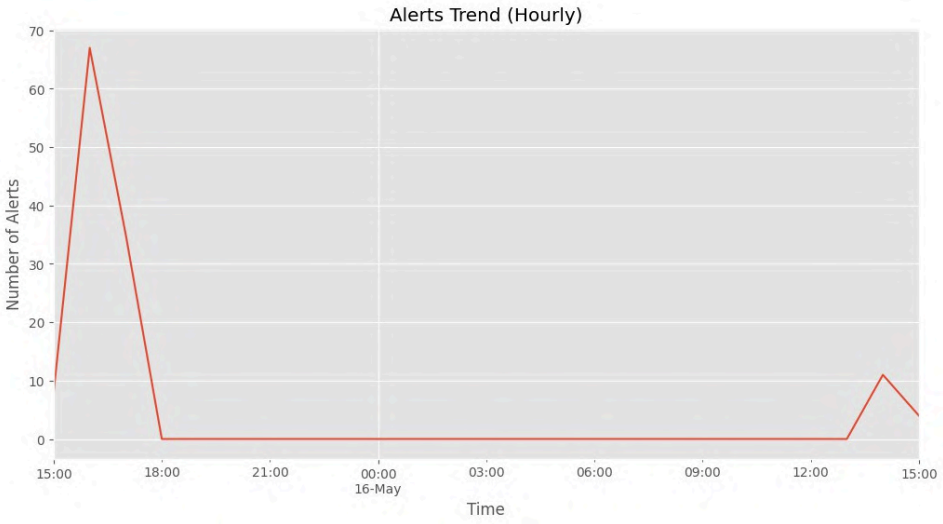
En este menú sale un gráfico en forma de tarta en el que sale el porcentaje de cada alerta. Se puede ver qué tipo de ataque se ha realizado más.

Niveles de Severidad



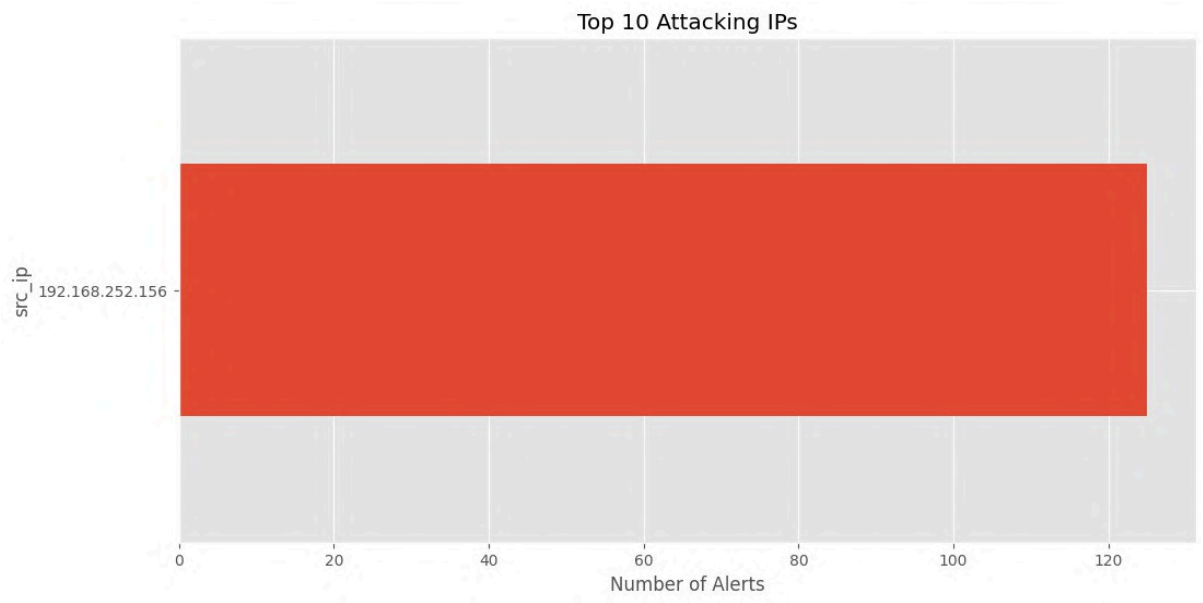
En "**Niveles de Severidad**" aparecen el número de alertas y si son de severidad 2 o 3, como la mayoría son escaneos de puertos y estos están clasificados como categoría 3, salen más alertas de ese nivel de severidad.

Tendencia Horaria



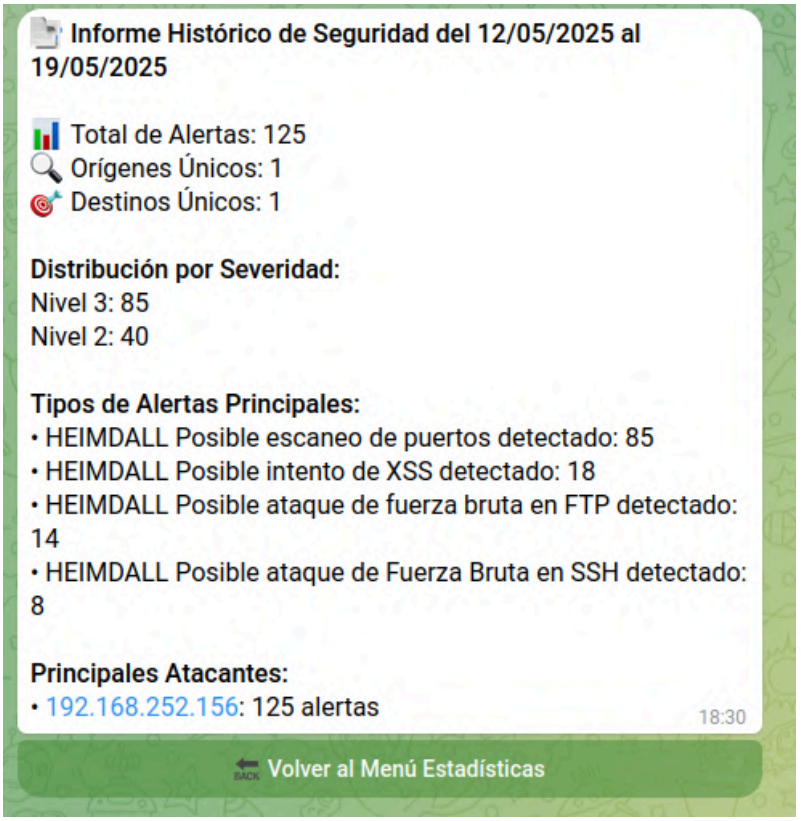
En "**Tendencia Horaria**" aparece un gráfico con las horas en que hay más actividad de alertas.

Top Atacantes



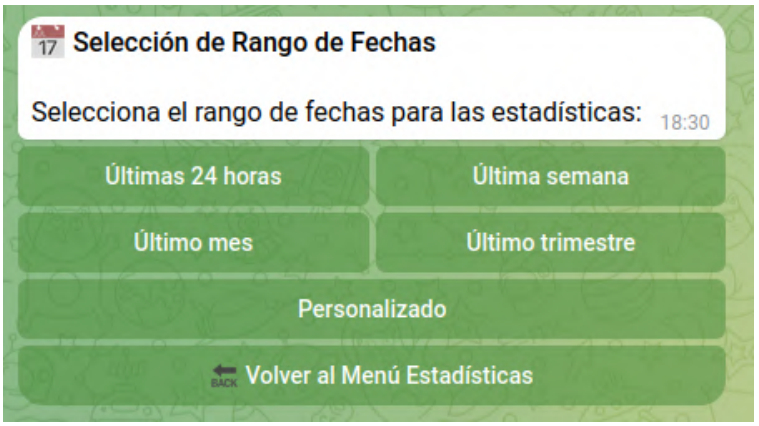
En "**Top Atacantes**" se puede ver quienes han sido los dispositivos que han realizado más ataques, como únicamente se está utilizando un atacante aparece su dirección IP.

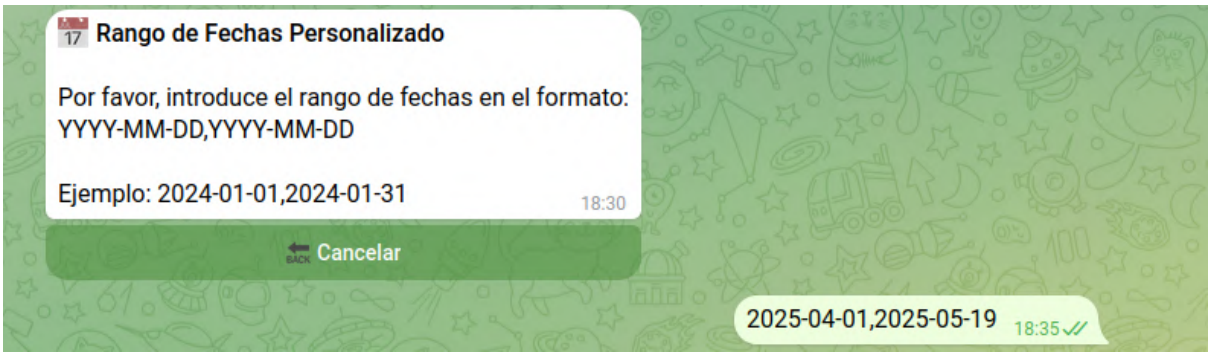
Reporte Histórico



En "**Reporte Histórico**" aparece un pequeño informe sobre el total de alertas, los tipos de alertas principales, los atacantes principales y más información relevante.

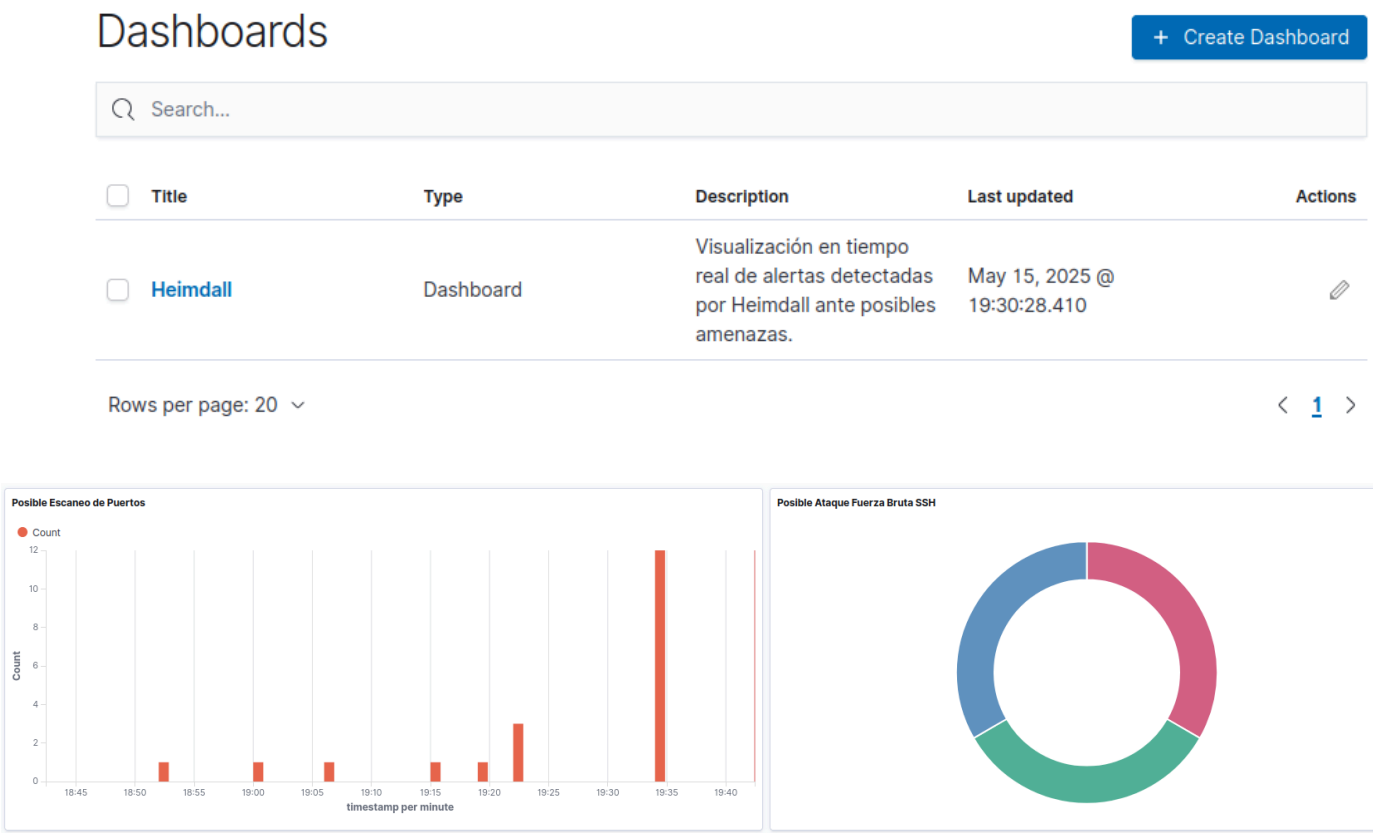
Seleccionar Rango de Fechas

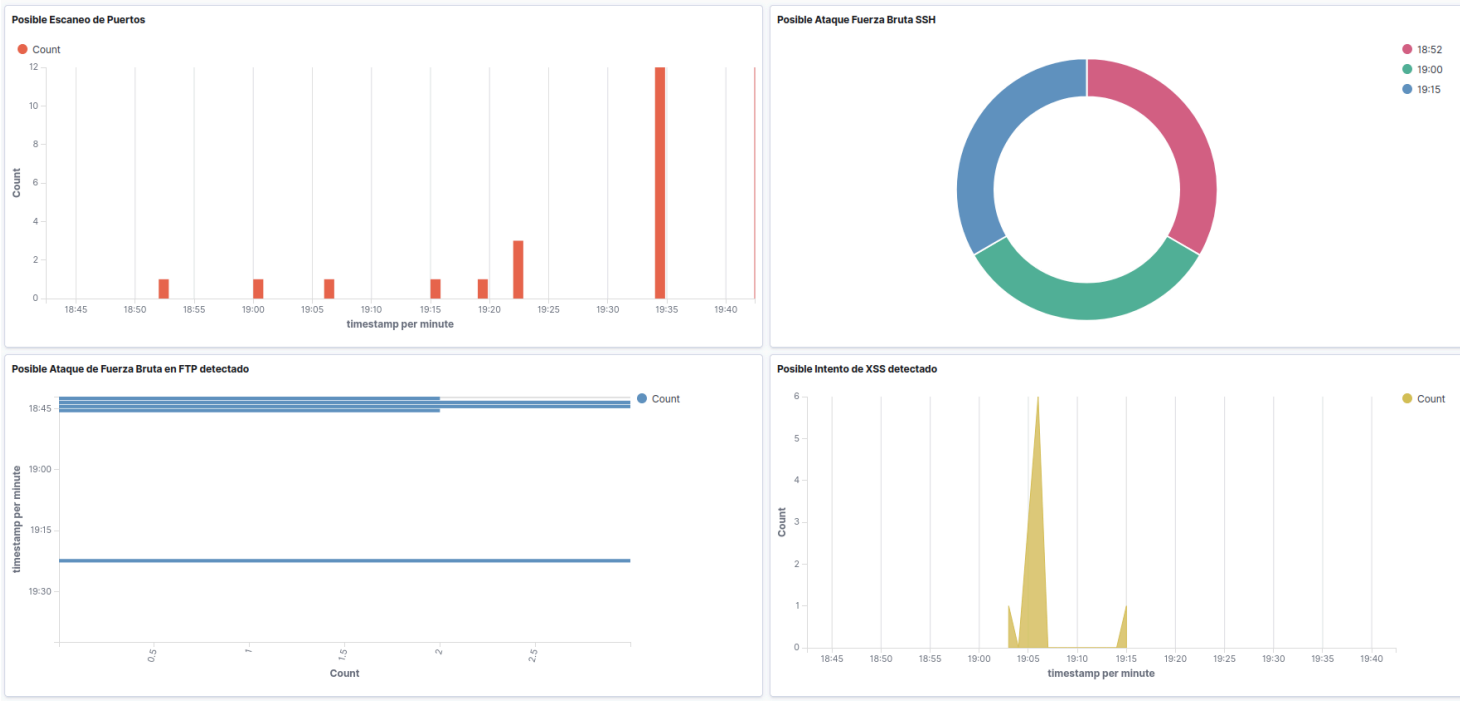




En "**Seleccionar de Rango de Fechas**" se puede seleccionar un rango de fechas para ver las estadísticas. Permite escoger entre las alertas de hace 24 horas, de hace un mes, de una semana, de hace tres meses e incluso permite escoger la fecha pulsando en "**Personalizado**".

3.10 Dashboards en Wazuh





Se ha creado un dashboard en el que salen varios gráficos sobre la cantidad de alertas que van saliendo de cada tipo de ataque. A parte de usar el menú de estadísticas del bot Heimdall, se pueden observar estos gráficos para tener una idea visual de la cantidad de alertas por cada tipo de ataques recibidos directamente en la página de Wazuh del servidor.

3.10 Problemas encontrados durante el proceso de creación del proyecto Heimdall

1. **Incompatibilidad de versiones entre componentes de Wazuh**
   Uno de los primeros problemas críticos fue que la versión del dashboard no coincidía con la del Wazuh Indexer. Esta desincronización impedía el uso correcto de la API, ya que el dashboard exigía versiones iguales como requisito. Se tuvo que reinstalar versiones compatibles para poder avanzar.

2. **Dificultades iniciales con las llamadas a la API**
   Las primeras pruebas con la API de Wazuh resultaron fallidas debido al desconocimiento de su estructura interna. Fue necesario estudiar cómo autenticar correctamente las solicitudes, gestionar los tokens y comprender la estructura de los endpoints para obtener respuestas válidas.

3. **Errores en la implementación de reglas personalizadas en Suricata**
   La creación de reglas propias para Suricata presentó desafíos tanto en la sintaxis como en la generación del tráfico adecuado para probarlas. En varios casos, las reglas no se activaban debido a errores menores que exigieron una revisión minuciosa y repetidas pruebas.

4. **Falta de experiencia en el desarrollo de bots con integración a APIs**
   La creación del bot de Telegram fue un proceso de aprendizaje, ya que no se contaba con experiencia previa en este tipo de desarrollos. Fue necesario aprender a gestionar eventos, manejar la autenticación de usuarios, diseñar menús dinámicos e integrar funcionalidades con la API de Wazuh.

5. **Falta de visibilidad en los dashboards iniciales**
   Al momento de crear visualizaciones en el dashboard de Wazuh, no se lograba mostrar información útil debido al desconocimiento de los campos adecuados para el filtrado. Fue necesario realizar varias pruebas hasta identificar correctamente el atributo de filtrado para que las gráficas funcionaran como se esperaba.

6. **No implementación del componente de Machine Learning**
   Si bien inicialmente se valoró la inclusión de un módulo de Machine Learning para la detección automática de patrones de ataque, no fue posible su desarrollo en esta fase del proyecto. Las razones principales fueron la falta de tiempo y de conocimientos avanzados en modelos predictivos aplicados a ciberseguridad. No obstante, se considera una mejora futura con alto potencial.

Capítulo 4: Conclusiones

A lo largo del desarrollo del proyecto se han consolidado y ampliado de forma significativa los conocimientos en herramientas de seguridad de infraestructura. Se ha profundizado en el uso de Wazuh, aprovechando sus capacidades avanzadas para el análisis de logs y la gestión de alertas, funcionalidades que previamente sólo se conocían de manera superficial. Asimismo, se ha integrado Suricata para llevar a cabo la inspección en tiempo real del tráfico de red y la detección de intrusiones.

Por primera vez, se ha implementado un bot capaz de interactuar directamente con la API de Wazuh, gestionando peticiones, enviando notificaciones y ejecutando controles remotos a través de Telegram. También se han aplicado conceptos clave de ciberseguridad mediante el diseño y ajuste de reglas específicas orientadas a la detección de escaneos de puertos, ataques de fuerza bruta mediante SSH o ataques XSS, visualizando estos eventos de manera clara y estructurada mediante dashboards.

La combinación de tareas de despliegue, automatización y resolución de problemas ha proporcionado una experiencia de aprendizaje completa y bien estructurada.

Con vistas al futuro, se plantea enriquecer el bot con funcionalidades adicionales como la gestión dinámica de listas negras y blancas, la generación de informes detallados, la implementación de respuestas automatizadas y la incorporación de técnicas de Machine Learning para identificar patrones de ataque emergentes y activar contramedidas proactivas.

Este proceso ha servido no solo para reforzar la competencia técnica, sino también para desarrollar la capacidad de coordinar soluciones integrales de ciberseguridad.

Capítulo 5: Glosario

- **IDS (Intrusion Detection System)**: Sistema de Detección de Intrusos. Monitorea y analiza el tráfico de red o actividades del sistema para detectar comportamientos maliciosos.
- **Heimdall**: Figura mitológica nórdica, guardián de Asgard.
- **Blacklist**: Lista de direcciones IP o dominios bloqueados por considerarse maliciosos o no deseados.
- **ICMP**: Protocolo de control de mensajes de Internet. Utilizado para diagnosticar problemas de red, como en los comandos ping.
- **nmap**: Herramienta de escaneo de red utilizada por administradores y atacantes para descubrir servicios y puertos abiertos.
- **Firewall (Cortafuegos)**: Sistema que regula el tráfico de red entrante y saliente basado en reglas de seguridad establecidas. Puede ser físico o software.
- **SIEM**: (Security Information and Event Management) Sistema que recopila, analiza y correlaciona logs de múltiples fuentes para detección de amenazas.
- **Machine Learning**: Rama de la inteligencia artificial que permite a los sistemas aprender automáticamente a partir de datos, identificar patrones y tomar decisiones sin intervención humana explícita. En un IDS, puede usarse para detectar anomalías en el tráfico que no coinciden con firmas conocidas.
- **Logs**: Registros generados automáticamente por sistemas, aplicaciones o dispositivos que contienen información detallada sobre eventos, accesos, errores o transacciones. Son fundamentales para el análisis forense y la detección de amenazas en Heimdall IDS.
- **Endpoint**: Cualquier dispositivo final conectado a una red, como estaciones de trabajo, servidores, móviles o IoT. Son objetivos frecuentes de ataques y deben ser monitorizados por soluciones como Heimdall.
- **Dashboard**: Interfaz gráfica que centraliza la visualización de datos clave del sistema, como alertas, eventos de seguridad, estado del sistema y estadísticas. Permite un monitoreo rápido y eficaz.
- **ISO**: Imagen de disco que contiene una copia exacta de un sistema de archivos. Comúnmente usada para instalaciones de sistemas operativos o aplicaciones completas como Heimdall IDS en entornos dedicados.
- **Partición**: División lógica de un disco duro que permite organizar datos y sistemas operativos de forma separada. Útil para separar logs, sistema operativo y archivos críticos por seguridad.
- **Repositorio**: Lugar donde se almacenan, mantienen y versionan archivos o paquetes, como código fuente, reglas de detección o configuraciones. Puede ser local o remoto (por ejemplo, GitHub).

- **API (Application Programming Interface)**: Conjunto de reglas y definiciones que permiten la comunicación entre aplicaciones. Heimdall puede usar APIs para integrar con otros sistemas como Wazuh, SIEMs, o herramientas de visualización.

Capítulo 6: Bibliografía

- **API de Wazuh:**
https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://documentation.wazuh.com/current/user-manual/api/reference.html&ved=2ahUKEwjG4eeXjZ6NAxWsS_EDHZG7L-oQFnoECAkQAQ&usg=AOvVaw3HIZ6vEOitahQqUi2_SMtz

- **Integración de Suricata (Reddit):**
https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.reddit.com/r/Wazuh/comments/1caeb9r/suricata_integration/%3Ftl%3Des-es&ved=2ahUKEwiyg7mqjZ6NAxUaRvEDHcAvI2kQFnoECB4QAQ&usg=AOvVaw1Wzx8tKGznb10_y4dlKChb

- **ChatGPT:**
https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://chatgpt.com/&ved=2ahUKEwiPp42zjZ6NAxWvcvEDHSpbEFsQFnoECAoQAQ&usg=AOvVaw29vbCnS_7PDD4xupasoOfg

- **Integración de Wazuh con Suricata (vídeo):**
https://youtu.be/NB_u9m-MMcY?si=N97wJ1SHcY0vMvcC

- **Wazuh Dashboards (vídeo):**
https://youtu.be/3CaG2GI1kn0?si=TBBQaUoujk5WdhEB

Capítulo 7: Anexos

7.1 Script de prueba

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

import json
import requests
import urllib3
from base64 import b64encode

Disable insecure https warnings (for self-signed SSL certificates)
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

Configuration
protocol = 'https'
host = 'localhost'
port = 55000
user = '<WAZUH_API_USER>'
password = '<WAZUH_API_PASSWORD>'
login_endpoint = 'security/user/authenticate'

login_url = f"{protocol}://{host}:{port}/{login_endpoint}"
basic_auth = f"{user}:{password}".encode()
login_headers = {'Content-Type': 'application/json',
               'Authorization': f'Basic {b64encode(basic_auth).decode()}'}

print("\nLogin request ...\n")
response = requests.post(login_url, headers=login_headers, verify=False)
token = json.loads(response.content.decode())['data']['token']
print(token)

New authorization header with the JWT we got
requests_headers = {'Content-Type': 'application/json',
                     'Authorization': f'Bearer {token}'}

print("\n- API calls with TOKEN environment variable ...\n")

print("Getting API information:")

response = requests.get(f"{protocol}://{host}:{port}/?pretty=true",
headers=requests_headers, verify=False)
print(response.text)

print("\nGetting agents status summary:")

response = requests.get(f"{protocol}://{host}:{port}/agents/summary/status?pretty=true",
headers=requests_headers, verify=False)
print(response.text)

print("\nEnd of the script.\n")
```

7.2 Bloques de código de cada botón

Primero se tienen que realizar las llamadas correspondientes a la API para obtener la información que se desea. Se necesita obtener un token y realizar un ***login request*** para "iniciar sesión" en la API para que posteriormente se puedan realizar las llamadas deseadas.

```
class WazuhAPI:
   def __init__(self):
       self.base_url = CONFIG['INDEXER_URL']
       self.session = requests.Session()
       self.session.verify = False
       self.session.auth = (CONFIG['INDEXER_USER'], CONFIG['INDEXER_PASSWORD'])
       self.session.headers.update({'Content-Type': 'application/json'})
       self.last_alert_timestamp = None
       logger.info(f"Initializing WazuhAPI with Indexer URL: {self.base_url}")
```

En la siguiente función lo que se hace es que una vez el bot está conectado a la API de Wazuh recoge e imprime por pantalla la información sobre el estado de los agentes, básicamente ver los agentes que están conectados y los que no y un poco de información.

```
async def get_agents_status():
   """Obtiene el estado de los agentes de Wazuh"""
   try:
       # Autenticación para obtener el token JWT
       login_endpoint = f"https://{CONFIG['IDS_HOST']}:55000/security/user/authenticate"
       basic_auth = f"{CONFIG['IDS_USER']}:{CONFIG['IDS_PASSWORD']}".encode()
       login_headers = {
           'Content-Type': 'application/json',
           'Authorization': f'Basic {b64encode(basic_auth).decode()}'
       }

       login_response = requests.post(login_endpoint, headers=login_headers, verify=False)
       login_response.raise_for_status()
       token = login_response.json()['data']['token']

       # Obtener la lista completa de agentes
       agents_endpoint = f"https://{CONFIG['IDS_HOST']}:55000/agents"
       agents_headers = {
           'Content-Type': 'application/json',
           'Authorization': f'Bearer {token}'
       }

       agents_response = requests.get(agents_endpoint, headers=agents_headers,
verify=False)
       agents_response.raise_for_status()

       agents_data = agents_response.json()
       all_agents = agents_data.get('data', {}).get('affected_items', [])

       # Filtrar el manager (agent 000)
       agents = [agent for agent in all_agents if agent.get('id') != '000']
```

```
        # Contar agentes por estado
        active = sum(1 for agent in agents if agent.get('status') == 'active')
        disconnected = sum(1 for agent in agents if agent.get('status') == 'disconnected')
        pending = sum(1 for agent in agents if agent.get('status') == 'pending')
        never_connected = sum(1 for agent in agents if agent.get('status') ==
'never_connected')
        total = len(agents)

        # Traducir estados de los agentes
        status_translations = {
            'active': 'activo',
            'disconnected': 'desconectado',
            'pending': 'pendiente',
            'never_connected': 'nunca conectado'
        }

        # Actualizar estados en la lista de agentes
        for agent in agents:
            agent['status'] = status_translations.get(agent.get('status'),
agent.get('status'))

        # Crear mensaje detallado para logging
        logger.info("Detalle de agentes:")
        for agent in agents:
            logger.info(f"ID: {agent.get('id')}, Nombre: {agent.get('name')}, Estado:
{agent.get('status')}")

        return {
            'active': active,
            'disconnected': disconnected,
            'pending': pending,
            'never_connected': never_connected,
            'total': total,
            'agents': agents  # Incluir la lista completa de agentes para debugging
        }
    except Exception as e:
        logger.error(f"Error getting agents status: {str(e)}")
        return None
```

Para recoger las alertas e imprimirlas, la API de Wazuh busca en el archivo **`eve.json`** (el archivo donde se guardan las alertas) y filtra por **“`data.alert.signature: “*[HEIMDALL]*”`**, solo saldrán alertas que contengan **`[HEIMDALL].`** El bot monitoriza cada 5 segundos las alertas y las imprime por pantalla.

```
   def get_alerts(self, limit: int = 5):
       """Fetch latest alerts from Wazuh Indexer"""
       try:
           logger.info(f"Fetching last {limit} alerts from Indexer")
           endpoint = f"{self.base_url}/wazuh-alerts-*/_search"

           # Si es la primera vez, guardamos el timestamp actual
           if self.last_alert_timestamp is None:
               self.last_alert_timestamp =
datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%S.%fZ")
               logger.info(f"Setting initial timestamp: {self.last_alert_timestamp}")
               return []

           payload = {
               "query": {
                "bool": {
                       "must": [
                           {"match": {"location": "/var/log/suricata/eve.json"}},
                           {"wildcard": {"data.alert.signature": "*[HEIMDALL]*"}},
                           {"range": {
                               "@timestamp": {
                                   "gt": self.last_alert_timestamp
                               }
                           }}
                       ]
                   }
               },
               "size": limit,
               "sort": [{"@timestamp": {"order": "desc"}}]
           }

           response = self.session.get(endpoint, json=payload)
           response.raise_for_status()

           data = response.json()
           hits = data.get('hits', {}).get('hits', [])
           alerts = [hit['_source'] for hit in hits]

           # Actualizamos el timestamp con la alerta más reciente si hay nuevas
           if alerts:
               self.last_alert_timestamp = alerts[0].get('@timestamp')
               logger.info(f"Updated last alert timestamp to: {self.last_alert_timestamp}")

           logger.info(f"Successfully retrieved {len(alerts)} alerts")
           return alerts

       except Exception as e:
           logger.error(f"Error getting alerts: {str(e)}")
           return []
```

Autenticación

Para darle más seguridad a Heimdall, se ha implementado un sistema en el que al ejecutar el bot (con **`/start`**) se solicita la contraseña de acceso. Si se escribe bien sale un mensaje de éxito y se puede usar el bot sin ningún problema. Si se escribe mal saldrá un mensaje de error y solicitará la contraseña de nuevo, también se puede utilizar el comando **`/cancel`** para salir.

```
async def authenticate(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
        """Maneja la autenticación del usuario"""
        if update.message.text == CONFIG['BOT_PASSWORD']:
        context.user_data['authenticated'] = True
        await update.message.reply_text(
                "✅ Autenticación exitosa\n"
                "🤖 Bot de monitoreo HEIMDALL iniciado\n"
                "📡 Usa el menú para iniciar el monitoreo..."
        )
        await show_main_menu(update, context)
        return AUTHENTICATED
        else:
        await update.message.reply_text(
                "❌ Contraseña incorrecta. Por favor, intenta de nuevo o usa /cancel para
salir."
        )
        return UNAUTHENTICATED
```

Menú principal

Este fragmento de código muestra el **menú principal** al usuario con botones interactivos. Primero, verifica si el sistema de monitoreo está activo consultando si hay algún trabajo y luego construye un teclado con varios botones como son ***Desbloquear IPs***, ***Ver IPs bloqueadas***, ***Gestionar la Blacklist***, ***Ver el estado de los agentes*** y ***Estadísticas***. Finalmente, según el estado del monitoreo, se puede iniciar o detener.

```
async def show_main_menu(update: Update, context: ContextTypes.DEFAULT_TYPE):
   """Muestra el menú principal con botones"""
   # Verificar si el monitoreo está activo
   is_monitoring = bool(context.job_queue.get_jobs_by_name('check_alerts'))

   keyboard = [
       [
           InlineKeyboardButton("🔓 Desbloquear IP", callback_data="unblock_ip_menu"),
           InlineKeyboardButton("📋 Ver IPs Bloqueadas", callback_data="show_blocked")
       ],
       [
           InlineKeyboardButton("⚫ Gestionar Blacklist",
callback_data="manage_blacklist"),
           InlineKeyboardButton("📊 Estadísticas", callback_data="show_stats")
       ],
       [
           InlineKeyboardButton("📊 Estado de Agentes", callback_data="show_agents_status")
       ]
   ]
```

```
    # Añadir el botón de monitoreo según el estado actual
    if is_monitoring:
        keyboard.append([InlineKeyboardButton("🔲 Detener Monitoreo",
callback_data="stop_monitoring")])
    else:
        keyboard.append([InlineKeyboardButton("▶️ Iniciar Monitoreo",
callback_data="start_monitoring")])

    reply_markup = InlineKeyboardMarkup(keyboard)

    if update.callback_query:
        await update.callback_query.edit_message_text(
            text="🤖 *Menú Principal HEIMDALL*\n\nSelecciona una opción:",
            reply_markup=reply_markup,
            parse_mode='Markdown'
        )
    else:
        await update.message.reply_text(
            "🤖 *Menú Principal HEIMDALL*\n\nSelecciona una opción:",
            reply_markup=reply_markup,
            parse_mode='Markdown'
        )
```

Gestionar Blacklist

Esta función define y despliega el menú de gestión de la blacklist en el bot de Telegram. Cuando el usuario accede a esta sección, se le presentan tres opciones interactivas:

- **Añadir IP/Rango:** permite incorporar una nueva IP o rango de IPs a la lista negra.
- **Eliminar IP/Rango:** elimina direcciones IP previamente añadidas.
- **Volver al Menú Principal:** regresa al menú principal del bot.

El objetivo es facilitar la administración de IPs potencialmente peligrosas directamente desde la interfaz del bot, sin necesidad de intervenir manualmente en configuraciones del sistema. El texto del mensaje se envía utilizando formato Markdown para una presentación más clara y ordenada.

```
async def show_blacklist_menu(update: Update, context: ContextTypes.DEFAULT_TYPE):
        """Muestra el menú de gestión de blacklist"""
        keyboard = [
        [
                InlineKeyboardButton("➕ Añadir IP/Rango",
callback_data="add_to_blacklist"),
                InlineKeyboardButton("➖ Eliminar IP/Rango",
callback_data="remove_from_blacklist")
        ],
        [
                InlineKeyboardButton("🔙 Volver al Menú Principal",
callback_data="main_menu")
        ]
        ]
        reply_markup = InlineKeyboardMarkup(keyboard)

        await update.callback_query.edit_message_text(
        text="⚫ *Gestión de Blacklist*\n\nSelecciona una opción:",
```

```
        reply_markup=reply_markup,
        parse_mode='Markdown'
        )
```

Iniciar Monitoreo

Esta función se encarga de activar el monitoreo de alertas desde el bot de Telegram. Primero comprueba si el usuario está autenticado y si no lo está le pide que lo haga. Luego revisa si el monitoreo ya está en marcha, si no lo está lo inicia y muestra un mensaje confirmando que ahora se están revisando las alertas de forma automática cada pocos segundos, junto con un botón para detenerlo cuando se desee.

```
async def start_monitoring(update: Update, context: ContextTypes.DEFAULT_TYPE):
        """Inicia el monitoreo de alertas"""
        if not context.user_data.get('authenticated', False):
        if update.callback_query:
                await update.callback_query.edit_message_text(
                text="❌ No estás autenticado. Usa /start para iniciar el proceso de
autenticación.",
                parse_mode='Markdown'
                )
        else:
                await update.message.reply_text(
                "❌ No estás autenticado. Usa /start para iniciar el proceso de
autenticación."
                )
        return

        # Verificar si ya está monitoreando
        if context.job_queue.get_jobs_by_name('check_alerts'):
        if update.callback_query:
                await update.callback_query.edit_message_text(
                text="ℹ️ El monitoreo ya está activo.",
                parse_mode='Markdown'
                )
        else:
                await update.message.reply_text("ℹ️ El monitoreo ya está activo.")
        return

        # Iniciar el monitoreo
        context.job_queue.run_repeating(
        check_alerts,
        interval=CONFIG['ALERT_CHECK_INTERVAL'],
        name='check_alerts'
        )

        # Crear el botón para detener el monitoreo
        keyboard = [[InlineKeyboardButton("🔽 Detener Monitoreo",
callback_data="stop_monitoring")]]
        reply_markup = InlineKeyboardMarkup(keyboard)

        if update.callback_query:
        await update.callback_query.edit_message_text(
                text="✅ Monitoreo iniciado correctamente.\n\n📡 Monitoreando alertas cada 5
segundos...",
                parse_mode='Markdown',
                reply_markup=reply_markup
        )
```

```
        else:
        await update.message.reply_text(
                "✅ Monitoreo iniciado correctamente.\n\n📡 Monitoreando alertas cada 5
segundos...",
                reply_markup=reply_markup
        )
```

Además de las funcionalidades explicadas en este documento, el sistema cuenta con muchas más funciones que complementan y amplían sus capacidades, como la visualización de estadísticas y el manejo avanzado de listas de bloqueo. Para consultar el código completo y en funcionamiento, puedes acceder a nuestro repositorio oficial en GitHub mediante el siguiente enlace:
https://github.com/Bogronas0/HEIMDALL/blob/main/HEIMDALL_BOT.PY

7.3 Configurar página web y servicios en el cliente

Se empezará instalando varios servicios y paquetes, como **Apache**, **PHP** y **libapache2-mod-php**, que es un módulo de Apache que permite ejecutar scripts PHP.

```
root@servidor-eyconsulting:/home/usuario# apt install apache2
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  apache2-bin apache2-data apache2-utils libapr1t64 libaprutil1-dbd-sqlite3
```

```
root@servidor-eyconsulting:/home/usuario# apt install php libapache2-mod-php
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  libapache2-mod-php8.3 php-common php8.3 php8.3-cli php8.3-common php8.3-opcache
```

También se instala el servicio FTP, que se utilizará para poder realizar ataques de fuerza bruta para adivinar el usuario y contraseña y poder acceder al FTP.

```
root@servidor-eyconsulting:/home/usuario# apt install vsftpd
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  vsftpd
```

Se configura:

```
root@servidor-eyconsulting:/home/usuario# nano /etc/vsftpd.conf
```

```
# Uncomment this to allow local users to log in.
local_enable=YES
#
# Uncomment this to enable any form of FTP write command.
write_enable=YES
```

Se crea el archivo **`eyconsulting.conf`** que servirá para añadir nuestro VirtualHost:

```
usuario@servidor-eyconsulting:~$ sudo nano /etc/apache2/sites-available/eyconsulting.conf
[sudo] password for usuario:
```

```
 GNU nano 7.2          /etc/apache2/sites-available/eyconsulting.conf
<VirtualHost *:80>
    ServerName eyconsulting.local
    DocumentRoot /var/www/html/eyconsulting

    <Directory /var/www/html/eyconsulting>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/eyconsulting_error.log
    CustomLog ${APACHE_LOG_DIR}/eyconsulting_access.log combined
</VirtualHost>
```

Se crea la carpeta donde se encontrarán los archivos PHP, que serán las diferentes secciones de la página web. Se le otorgan permisos al usuario **www-data** sobre la carpeta.

```
usuario@servidor-eyconsulting:~$ sudo mkdir -p /var/www/html/eyconsulting
usuario@servidor-eyconsulting:~$ sudo chown -R www-data:www-data /var/www/html/eyconsulting
```

```
usuario@servidor-eyconsulting:~$ sudo chmod -R 755 /var/www/html/eyconsulting
```

El primer archivo que se crea es **index.php**, que es la página principal de la web y la que tiene los enlaces a las diferentes secciones:

```
 GNU nano 7.2                  /var/www/html/eyconsulting/index.php
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ey Conulting</title>
</head>
<body>
  <h1>Ey Conulting - Consultoría</h1>
  <nav>
    <ul>
      <li><a href="index.php">Inicio</a></li>
      <li><a href="search.php">Buscar Clientes</a></li>
      <li><a href="xss.php">Formulario de Contacto</a></li>
      <li><a href="view.php">Ver archivo</a></li>
    </ul>
  </nav>
  <p>Bienvenido a Ey Conulting. Aquí podrás probar nuestros servicios (vulnerables) para f>
</body>
</html>
```

El archivo **search.php** permite realizar consultas SQL, interesante si se quiere probar a hacer ataques SQL Injection.

```
 GNU nano 7.2                  /var/www/html/eyconsulting/search.php
<?php
header('Content-Type: text/html; charset=UTF-8');
$term = isset($_GET['q']) ? $_GET['q'] : '';
// Sin sanitización para demonstar SQLi
echo "<h2>Resultados de búsqueda para: " . htmlspecialchars($term) . "</h2>";
echo "<p>Mostrando resultados simulados...</p>";
?>
<form method="GET">
  <input name="q" placeholder="Buscar cliente">
  <button>Buscar</button>
</form>
```

En el archivo **xss.php** se ha puesto una pequeña sección para escribir y subir archivos, muy útil para poder hacer ataques XSS.

```
 GNU nano 7.2                    /var/www/html/eyconsulting/xss.php
<!DOCTYPE html>
<html lang="es">
<head><meta charset="UTF-8"><title>Contacto</title></head>
<body>
  <h2>Formulario de Contacto</h2>
  <form method="GET">
    <label>Mensaje: <input name="msg"></label>
    <button>Enviar</button>
  </form>
  <?php
    if (isset($_GET['msg'])) {
      // Refleja sin escape para XSS
      echo "<div><strong>Tu mensaje:</strong> " . $_GET['msg'] . "</div>";
    }
  ?>
</body>
</html>
```

Por último, el archivo **view.php** permite ejecutar comandos en la web sobre el servidor, hecho con la intención de que Heimdall detecte que desde la web que se están ejecutando comandos como **cat /etc/passwd** o comandos que no debería intentar ejecutar ningún usuario.

```
 GNU nano 7.2                    /var/www/html/eyconsulting/view.php
<!DOCTYPE html>
<html lang="es">
<head><meta charset="UTF-8"><title>Ver Archivo</title></head>
<body>
  <h2>Leer archivo del sistema</h2>
  <form method="POST">
    <label>Comando (ej: cat /etc/passwd): <input name="cmd"></label>
    <button>Ejecutar</button>
  </form>
  <?php
    if ($_SERVER['REQUEST_METHOD'] === 'POST' && !empty($_POST['cmd'])) {
      echo "<pre>";
      // Ejecuta directamente para simular RCE
      system($_POST['cmd']);
      echo "</pre>";
    }
  ?>
</body>
</html>
```

La página web se ve así:

Ey Consulting - Consultoría

- Inicio
- Buscar Clientes
- Formulario de Contacto
- Ver archivo

Bienvenido a Ey Conulting. Aquí podrás probar nuestros servicios (vulnerables) para fines educativos.

Resultados de búsqueda para:

Mostrando resultados simulados...

Buscar cliente | Buscar

Formulario de Contacto

Mensaje: | Enviar

Leer archivo del sistema

Comando (ej: cat /etc/passwd): ls | Ejecutar

```
index.php
search.php
view.php
xss.php
```