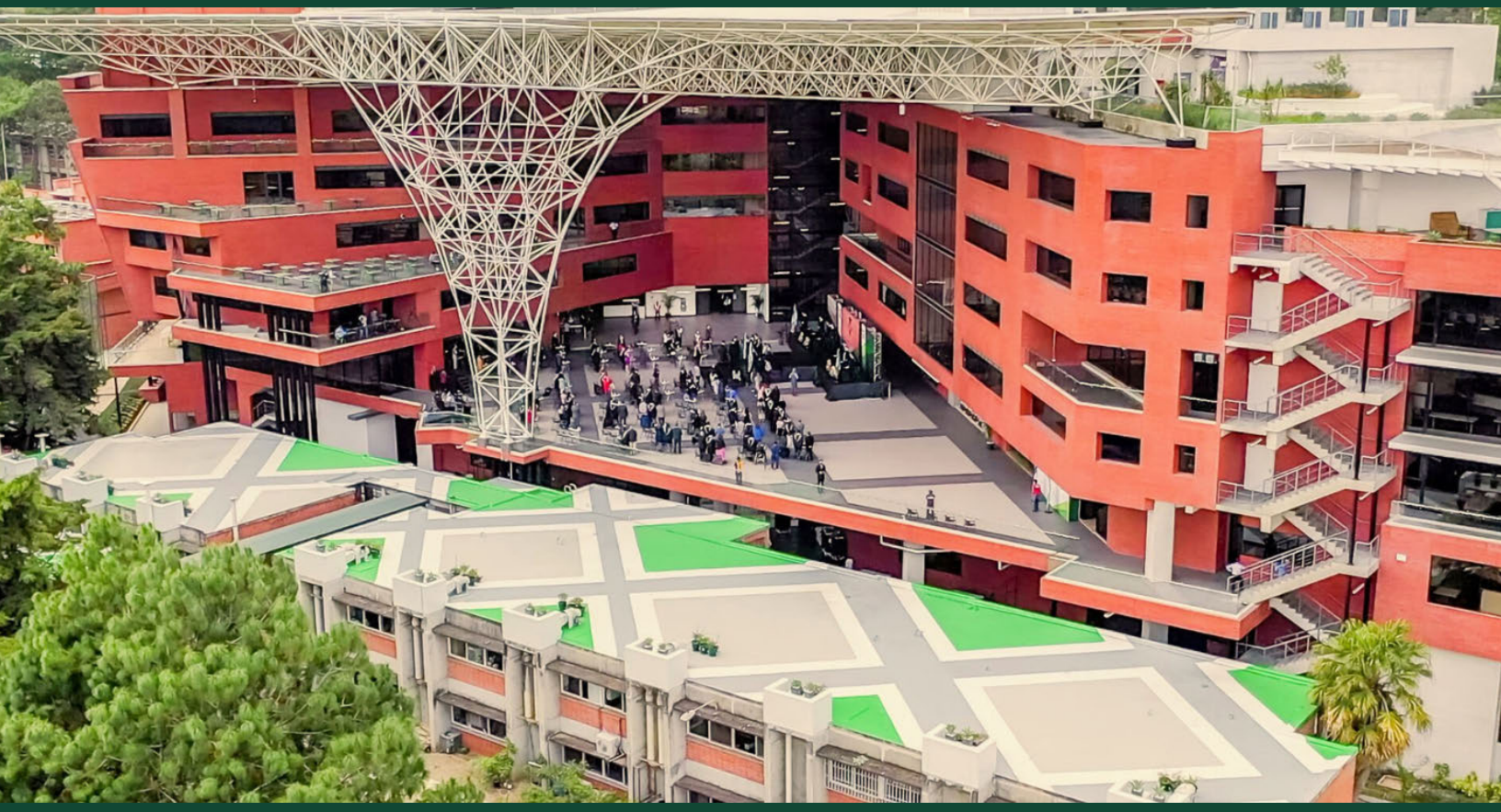
Sistema automatizado de detección y prevención de amenazas de red utilizando Snort

Andrés Estuardo Lemus Orozco



UVG | UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Sistema automatizado de detección y prevención de amenazas de red utilizando Snort

Trabajo de graduación presentado por Andrés Estuardo Lemus Orozco para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2025

Vo.Bo.:

(f) 

M.Sc. Jonathan de los Santos

(f) 

M.Sc. Carlos Esquit Hernández

Fecha de aprobación: Guatemala, 20 de noviembre de 2025.

Prefacio

Este trabajo de graduación es el resultado de años de aprendizaje, constancia y colaboración. Antes de presentar su contenido, deseo expresar mi sincero agradecimiento a quienes, con su apoyo y confianza, hicieron posible su realización.

A Dios, por la vida, la fortaleza y la claridad para perseverar en cada etapa de este proceso.

A mi familia, en especial a mis padres y hermana, por todo el amor y apoyo. Su ejemplo y respaldo han sido mi mayor motivación.

A mis amigos, por darme las risas y diversión necesarias para mantener el equilibrio durante este viaje.

A Spiky, que me acompañó durante todos estos años y fue mi compañía fiel en todo momento.

A mis compañeros, por el trabajo en equipo, las ideas compartidas y el ánimo constante.

A mis profesores y mentores, por compartir su conocimiento y experiencia, así como por inspirarme a buscar la excelencia en cada detalle.

Gracias.

Índice general

Índice de figuras

Índice de cuadros

Resumen

Ante el incremento y la complejidad de los ciberataques, este trabajo diseña e implementa un sistema automatizado de seguridad de red orientado a entornos pequeños y medianos, apoyado en tecnologías *open source* y validado en un laboratorio virtualizado. El sistema busca pasar de enfoques reactivos a una orquestación de respuesta en tiempo real que fortalezca la confidencialidad, integridad y disponibilidad de los servicios.

La propuesta presenta una arquitectura reproducible al integrar pfSense como cortafuegos y ruteador, Snort como IDS/IPS, Zabbix para observabilidad, Pi-hole para filtrado DNS y OpenVAS para evaluación de vulnerabilidades, con automatización de acciones de respuesta a partir de alertas de Snort.

El despliegue se realizó en una topología segmentada (LAN, DMZ y SOC & MOC), mientras que la simulación de ataques internos y externos, la elaboración de reglas personalizadas en Snort y el desarrollo de *scripts* se hizo en Python para el bloqueo y desbloqueo dinámico de direcciones IP en pfSense. Además, se incorporaron escaneos programados con OpenVAS y la verificación del filtrado por reputación vía Pi-hole.

El sistema mostró que pfSense garantiza conectividad y control de acceso y, en complemento con Snort, detecta actividades como escaneos y ataques de denegación de servicio de una manera efectiva. Esto permite que se añadan *scripts* en Python que automatizan y facilitan mitigar los ataques mediante el bloqueo y desbloqueo de direcciones IP maliciosas. Zabbix demostró que es una herramienta de monitoreo completa que ofrece un sistema de visibilidad y de alertas que ayuda a los administradores a mantener la infraestructura en óptimas condiciones. Todo esto en combinación con OpenVAS proporciona una base sistemática para identificar y gestionar vulnerabilidades; y Pi-hole añade una capa adicional de seguridad al filtrar amenazas basadas en DNS y mejorar la experiencia de navegación. Esto hace que la arquitectura propuesta eleve el umbral de protección de redes pequeñas y medianas con herramientas abiertas, costo-efectivas y replicables.

Palabras clave: seguridad de red, snort, pfsense, ids/ips, automatización.

Abstract

In response to the growing volume and sophistication of cyberattacks, this work designs and implements an automated network security system for small and medium-sized environments. The system is built on open-source technologies and validated in a virtualized laboratory. Its aim is to shift from reactive approaches to real-time, orchestrated response that strengthens the confidentiality, integrity, and availability of networked services.

The proposed architecture is reproducible and integrates pfSense as firewall and router, Snort as IDS/IPS, Zabbix for observability, Pi-hole for DNS filtering, and OpenVAS for vulnerability assessment, with automation that turns Snort alerts into concrete response actions.

The deployment was carried out on a segmented topology (LAN, DMZ, and SOC & MOC), with simulations of internal and external attacks, the creation of custom Snort rules, and Python scripts that dynamically block and unblock IP addresses in pfSense. In addition, scheduled OpenVAS scans were incorporated, as well as verification of reputation-based filtering through Pi-hole.

Results show that pfSense ensures connectivity and access control and, when complemented with Snort, effectively detects activities such as port scans and denial-of-service attacks. Python scripts automate and streamline mitigation by managing IP blocks. Zabbix proved to be a comprehensive monitoring platform that provides visibility and alerting to help administrators maintain infrastructure health. Together with OpenVAS, which offers a systematic basis for identifying and managing vulnerabilities, and Pi-hole, which adds a DNS-based protective layer and improves browsing experience, the proposed architecture raises the protection baseline for small and medium networks using open, cost-effective, and replicable tools.

Keywords: network security, snort, pfsense, ids/ips, automation.

CAPÍTULO 1

Introducción

La seguridad de redes comprende tecnologías, políticas y procedimientos que, en múltiples capas, resguardan la infraestructura frente a accesos no autorizados, pérdida de datos e intrusiones. Entre sus pilares se encuentran el *firewall* que proporciona la segmentación y los sistemas de detección y prevención de intrusiones (IDS/IPS), que combinan firmas y análisis de comportamiento para identificar actividad maliciosa.

En este contexto, el trabajo tiene como finalidad diseñar e implementar un sistema automatizado capaz de detectar, analizar y responder a amenazas en tiempo real. La solución se articula alrededor de Snort como IDS/IPS, con pfSense para enrutamiento y control de acceso, Pi-hole para filtrado DNS, Zabbix para monitoreo y OpenVAS para evaluación de vulnerabilidades, integrando automatización para convertir alertas en acciones de contención.

El sistema se delimita a un entorno de laboratorio virtualizado que reproduce una red segmentada con LAN de usuarios, DMZ con servicios expuestos y un segmento SOC & MOC para monitoreo y gestión. Esta configuración permite experimentar sin hardware adicional y evaluar la escalabilidad de la propuesta en escenarios típicos de redes pequeñas y medianas. El proyecto se enfoca en pruebas prácticas con ataques internos y externos, así como en la documentación necesaria para su reproducibilidad. El desarrollo se estima en seis meses, con énfasis en implementación, pruebas y documentación, buscando contribuir al conocimiento sobre seguridad informática en redes con recursos limitados.

La experiencia de diseño, despliegue y prueba demuestra que una arquitectura integrada y automatizada basada en herramientas abiertas puede elevar de manera tangible el nivel de protección en redes pequeñas y medianas. La combinación de detección temprana, respuesta orquestada y visibilidad operativa reduce la superficie de ataque, acorta los tiempos de contención y facilita la gestión diaria sin requerir infraestructura costosa. En conjunto, el sistema propuesto traslada la postura de seguridad desde lo reactivo hacia lo preventivo, con una solución reproducible y escalable que ofrece una ruta clara para su adopción en entornos reales.

CAPÍTULO 2

Antecedentes

En el artículo titulado *Implementation of Snort IPS Using PfSense as Network Forensic in Smk XYZ* [1] se explica cómo se implementó Snort sobre pfSense para proteger un servidor de *e-learning* en una institución educativa. El proyecto permitió la detección de ataques como escaneos de puertos, ARP *spoofing* y ataques de denegación de servicio (DoS); además, se integró un enfoque forense para registrar, analizar y responder automáticamente mediante el bloqueo de direcciones IP sospechosas. Esto nos indica la importancia de los sistemas de prevención basados en patrones y su valor para entornos educativos vulnerables.

En el artículo titulado *Analysis of a New Firewall Constructed on PfSense with Snort to Defend Against Common Internet Intrusions* [2] se propone un sistema basado en pfSense y Snort, donde se implementan algoritmos de detección avanzados con *machine learning*, como el modelo CNN-BiLSTM. También se sugiere que la dirección a futuro para fortalecer sistemas basados en Snort es mediante técnicas de inteligencia artificial, lo que permite considerar la evolución de la arquitectura en etapas posteriores.

En el artículo titulado *Snort IDS System Visualization Interface for Alert Analysis* [3], se desarrolló una interfaz de visualización de alertas generadas por Snort IDS. Su sistema está diseñado para representar gráficamente y en tiempo real las alertas clasificadas por tipo de ataque, prioridad y dirección IP de origen. Es un trabajo útil, ya que incorpora herramientas como Grafana para facilitar la interpretación visual de los datos recolectados por el IDS, mejorando así la capacidad de respuesta ante incidentes.

En la tesis titulada *Seguridad en Internet de las Cosas* se centra en la implementación de una arquitectura de seguridad utilizando Snort como IDS/IPS y el *firewall* UFW en una red local segmentada, incluyendo una DMZ. Se detalla la configuración de reglas, la generación de alertas y el análisis de logs para la detección de incidentes de seguridad. La implementación paso a paso proporciona una referencia útil del mundo real para configurar Snort, crear reglas de detección e integrarlas en un diseño de red accesible [4].

En la tesis titulada *Analysis of Security Configuration for IDS/IPS* se presenta un estudio detallado de los sistemas IDS/IPS, incluyendo tanto el desarrollo histórico como las

estrategias de implementación actuales, con especial énfasis en su integración en arquitecturas de red seguras. También ofrece una evaluación comparativa de soluciones como Snort, Suricata, Zeek y Security Onion. El trabajo presenta VEREFOO, un marco para la generación automática de reglas de seguridad, y destaca cómo la integración de dicho marco con herramientas IDS/IPS puede mejorar la automatización de la seguridad [5].

CAPÍTULO 3

Justificación

A medida que la infraestructura digital se vuelve la base de operación de organizaciones y personas, los ataques crecen en frecuencia y sofisticación, desbordando las contramedidas manuales y comprometiendo la confidencialidad, integridad y disponibilidad de los servicios. En este contexto, el trabajo busca mejorar el umbral de protección de redes pequeñas y medianas; al integrar una arquitectura reproducible con capacidades de detección, filtrado, monitoreo y respuesta en tiempo real; al apoyarse en tecnologías *open source* que reducen barreras económicas; y, al desarrollarse y validarse en un entorno virtualizado que evita depender de hardware adicional.

La razón para la realización de este trabajo es diseñar e implementar un sistema de seguridad de red automatizado capaz de detectar, analizar y responder a amenazas en tiempo real, utilizando como eje a Snort, complementado con filtrado de amenazas basado en DNS, con la gestión y actualización dinámica de reglas de *firewall* dentro de un entorno virtualizado y con el uso de herramientas de monitoreo y auditoría como Zabbix y OpenVAS.

Este trabajo aborda la insuficiencia de enfoques reactivos y manuales frente a la velocidad de propagación y complejidad de los ataques cibernéticos actuales. El proyecto responde consolidando, en una misma arquitectura, la detección/prevención a nivel de tráfico, el control de acceso y el filtrado por reputación de dominios, para pasar de la mera generación de alertas a una orquestación de respuesta automatizada que fortalezca la postura de seguridad de redes con recursos limitados.

Este trabajo se desplegará en un laboratorio virtualizado con cinco equipos físicos que representan los elementos clave de la topología: un *firewall* pfSense como núcleo de enrutamiento y segmentación; una LAN de usuarios; una DMZ con servicios expuestos; un SOC & MOC con herramientas de monitoreo y escaneo de vulnerabilidades; y un equipo externo de atacante controlado para pruebas. Sobre esta base se integrarán Snort operando como IDS/IPS en todos los segmentos de red, Pi-hole para filtrado DNS, Zabbix para observabilidad y OpenVAS para evaluación de vulnerabilidades. La configuración incluirá la automatización de respuestas a incidentes mediante scripts de Python basados en alertas de Snort, como el bloqueo dinámico de direcciones IP maliciosas en el *firewall* pfSense.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Diseñar e implementar un sistema de seguridad de red automatizado que sea capaz de detectar, analizar y responder a amenazas en tiempo real utilizando como herramienta principal Snort para la detección y prevención de intrusiones, filtrado de amenazas basado en DNS y la gestión y actualización dinámica de reglas de *firewall* en un entorno de red virtualizado.

4.2. Objetivos específicos

- Desplegar y configurar pfSense como *firewall* central y plataforma de enrutamiento, estableciendo la segmentación de la red, las políticas de control de acceso y asegurando la comunicación entre segmentos.
- Implementar y configurar Snort como módulo adicional a pfSense para la detección y posterior prevención de intrusiones en segmentos clave de la red, permitiendo el análisis en tiempo real del tráfico de red, la generación de alertas y la aplicación de reglas de bloqueo.
- Crear un conjunto de reglas personalizadas para Snort que se adapten a las necesidades específicas de la red que reconozcan patrones de ataque comunes que puedan afectar a los sistemas y servicios desplegados.
- Desarrollar un proceso automatizado utilizando Python que procese las alertas que procedan desde Snort y pueda cambiar dinámicamente las reglas del *firewall* pfSense.
- Integrar Zabbix para la recopilación y monitoreo en tiempo real de métricas de red, actividad de las posibles amenazas e indicadores de estado.

- Configurar OpenVAS para realizar evaluaciones periódicas de vulnerabilidades en los sistemas y servicios desplegados en la red, generando reportes que ayuden a identificar y mitigar riesgos.
- Implementar Pi-hole para el filtrado de amenazas basado en DNS, bloqueando dominios maliciosos y mejorando la seguridad del tráfico DNS.

CAPÍTULO 5

Alcance

El proyecto busca implementar un sistema automatizado de detección y prevención de amenazas de red, utilizando Snort como herramienta principal, en un entorno virtualizado utilizando el *firewall* pfSense como núcleo de enrutamiento y segmentación. El objetivo es proporcionar un entorno seguro para redes pequeñas y medianas, permitiendo la detección, análisis y respuesta automatizada ante patrones de ataque conocidos. Se pretende crear un entorno local de laboratorio que simule un entorno de producción, donde se puedan realizar pruebas y validaciones de las configuraciones y herramientas implementadas.

Este entorno estará compuesto por cinco computadoras físicas, donde cada una representa un componente clave de la topología. La primera computadora contendrá el *firewall* pfSense, que actuará como la puerta de enlace de la red, gestionando el tráfico entrante y saliente. Este será el componente central, que permitirá la segmentación efectiva de la red en diferentes zonas. En conjunto con pfSense, se integrará Snort para la detección y prevención de intrusiones como paquete mismo de pfSense, configurado para monitorear el tráfico de red en tiempo real y generar alertas ante actividades sospechosas. En la segunda computadora se encontrará el segmento de red LAN donde se alojarán los dispositivos de usuario final, como lo podrían ser computadoras y dispositivos móviles; además, en este segmento se implementará Pi-hole para el filtrado de amenazas basadas en DNS. La tercera computadora estará destinada a la red DMZ, que es donde se alojan servicios públicos como lo son servidores web, FTP, etc., permitiendo un acceso controlado desde el exterior. La cuarta computadora servirá como un *Security operations center* (SOC) & *Monitoring operations center* (MOC), donde se implementarán herramientas de seguridad y auditoría como OpenVAS para la evaluación de vulnerabilidades y scripts de Python para la automatización de respuestas ante las alertas generadas por Snort. También se utilizará una herramienta de monitoreo y análisis de métricas de red como lo es Zabbix para proporcionar una visión integral del estado de la red y el estado de los equipos en general. Finalmente, la quinta computadora actuará como un atacante que es externo a la red, simulando amenazas y ataques para evaluar la efectividad del sistema implementado.

El proyecto se enfocará en el desarrollo de pruebas prácticas en escenarios simulados

que incluirán ataques internos y externos, con el fin de evaluar la efectividad, capacidad de respuesta y escalabilidad del sistema implementado; y la documentación completa del proceso, configuraciones, resultados obtenidos, recomendaciones y conclusiones, facilitando la reproducibilidad del sistema para futuros proyectos o mejoras continuas.

La duración estimada del proyecto es de seis meses, con un enfoque en la implementación, pruebas y documentación. Se espera que al finalizar el proyecto se cuente con un sistema robusto y confiable que pueda ser utilizado como base para futuras investigaciones y desarrollos en el ámbito de la seguridad informática en redes pequeñas y medianas. Además, se busca contribuir al conocimiento general sobre la implementación de sistemas de detección y prevención de intrusiones en entornos virtualizados, proporcionando una guía práctica para profesionales y estudiantes interesados en el área.

CAPÍTULO 6

Marco teórico

6.1. *Network security*

Network security se refiere a todas las tecnologías, políticas, personas y procedimientos que defienden cualquier red de posibles ciberataques, accesos no autorizados y pérdida de datos. Consiste en crear un ambiente en el que todos los elementos de una red estén seguros. Implica protocolos criptográficos, de comunicación, de transporte y de intercambio. Por lo general, se organiza en varias capas de protección, tanto en el *edge* como en el *core* de la red. Cada capa establece reglas y controles que determinan quién puede acceder a los recursos de la red [6], [7].

6.1.1. Tipos de soluciones de *Network security*

- *Firewall*: un *firewall* es un elemento de red, ya sea hardware o software, que supervisa, filtra y controla el tráfico de red entrante y saliente, en función de reglas de seguridad definidas. Aplica políticas de seguridad establecidas por el administrador en uno o varios segmentos de red, comparando políticas, añadiendo módulos de protección y evaluando los paquetes de datos. Realiza decodificación y análisis de protocolos para permitir o denegar paquetes y/o conexiones dentro o fuera de la red. Por lo general, los *firewalls* se despliegan en el perímetro de la red y suelen constituir la capa más externa de defensa; sin embargo, también se pueden desplegar dentro de redes segmentadas para separar diversas secciones, como departamentos en una empresa [6], [8].
- Sistemas de Detección y Prevención de Intrusiones (IDS/IPS): un sistema IDS/IPS se compone de dos funciones: detección y prevención. Un *Intrusion detection system* (IDS) es un sistema que inspecciona y analiza el tráfico de red con el fin de identificar actividades maliciosas o intrusiones no autorizadas, basándose en patrones (*signatures*) conocidos y en comportamientos anómalos.

  Por su parte, un *Intrusion prevention system* (IPS) es una herramienta que, además de monitorear continuamente el tráfico, puede tomar acciones automáticas para bloquear

o prevenir las actividades maliciosas detectadas, como informar al administrador, bloquear el tráfico o eliminar la amenaza. Estos sistemas suelen integrarse con un *firewall* para proporcionar una protección más completa. Los IDS/IPS son especialmente eficaces para detectar y mitigar ataques de fuerza bruta, ataques de denegación de servicio (DoS) y de denegación de servicio distribuida (DDoS) [8], [9].

- Filtrado web y DNS: el filtrado web supervisa los sitios web a los que acceden los usuarios de una red, permitiendo o bloqueando el tráfico con el fin de proteger la red frente a posibles amenazas. Cuando un usuario solicita una página, el sistema de filtrado analiza el sitio web; si el sitio es considerado seguro, se autoriza el acceso, de lo contrario se bloquea.

  El filtrado de DNS actúa a nivel de resolución de dominios, bloqueando dominios completos en lugar de páginas individuales. Los resolutores DNS configurados como filtros pueden rechazar consultas para dominios maliciosos o sospechosos, generalmente a partir de listas de reputación previamente definidas [8].

- *Virtual Private Networks* (VPN): una red privada virtual es un túnel cifrado que permite a los usuarios establecer comunicaciones seguras a través de redes públicas como Internet. Cifra los datos que se transmiten entre el usuario remoto y la red corporativa. Existen dos tipos principales: VPN de acceso remoto (conexión de un usuario individual a la red corporativa) y VPN de sitio a sitio (conexión segura entre dos redes distintas) [8].

- *Network Access Control* (NAC): el propósito de un sistema NAC es evitar que dispositivos que no cumplen con los requisitos mínimos de seguridad se conecten a la red. Antes de autorizar el acceso, NAC verifica el estado de seguridad del dispositivo, comprobando por ejemplo si dispone de un antivirus actualizado o si el *firewall* está activado. Si el dispositivo no cumple con los requisitos, el acceso es denegado [10].

6.1.2. Tipos de amenazas por *malware*

Las amenazas a la seguridad de las redes son actividades maliciosas cuyo objetivo es interrumpir operaciones, robar datos o obtener acceso no autorizado a los sistemas [11].

- Virus: un virus en informática es un programa que, cuando se ejecuta, infecta el sistema y puede propagarse a otros componentes. Existen distintos tipos: *error-generating*, *program destroyers*, *system crushers*, *time theft*, *hardware destroyers*, *trojans*, *time bombs*, *trapdoors* y *hoaxes* [8].

- *Worm*: un *worm* es un programa malicioso que utiliza una red para replicarse. Un *worm* está diseñado para adentrarse a algún dispositivo a través de la red y después aprovecharse de una vulnerabilidad de dicho dispositivo. Una vez que el *worm* ha aprovechado alguna vulnerabilidad del sistema, buscará a otro ordenador en la red que tenga la misma vulnerabilidad [8], [11].

- *Spyware*: el *spyware* es un software que se instala en un dispositivo sin el consentimiento del usuario y que recopila información sobre dicho usuario y toda su actividad. Por lo general, un *spyware* busca recopilar datos como contraseñas, información de tarjetas

de crédito, historial de navegación y otra información personal. El *spyware* puede ser utilizado para robar identidad, realizar fraudes financieros o vender información personal a terceros [11].

- *Adware*: el *adware* es un software que muestra anuncios no deseados en el dispositivo del usuario. Estos anuncios pueden aparecer en forma de ventanas emergentes, *banners* o anuncios integrados en aplicaciones. El *adware* puede recopilar información sobre los hábitos de navegación del usuario y utilizarla para mostrar anuncios personalizados. Aunque el *adware* no es necesariamente malicioso, puede afectar el rendimiento del dispositivo y la experiencia del usuario [11].
- *Ransomware*: el *ransomware* es un tipo de malware que cifra los archivos del usuario y exige un rescate para restaurar el acceso a ellos. Este tipo de ataque puede ser devastador, ya que puede resultar en la pérdida permanente de datos si no se paga el rescate o si no se cuenta con copias de seguridad adecuadas. El *ransomware* a menudo se propaga a través de correos electrónicos de *phishing* o vulnerabilidades en el software [11].

6.2. Ataques de denegación de servicio (DoS)

Un ataque de denegación de servicio (DoS) es un intento malicioso de interrumpir el funcionamiento normal de un servicio, sistema o red, haciéndolo inaccesible para los usuarios legítimos. Los ataques DoS pueden adoptar diversas formas, pero su objetivo principal es agotar los recursos del sistema o red, como el ancho de banda, la memoria o la capacidad de procesamiento, para que no pueda atender las solicitudes legítimas [12].

6.2.1. *Teardrop attack*

Un ataque *teardrop* es un tipo de ataque DoS que explota una vulnerabilidad en la forma en que algunos sistemas operativos manejan la fragmentación de paquetes IP. En este ataque, el atacante envía paquetes IP fragmentados con valores de desplazamiento superpuestos o incorrectos. Cuando el sistema objetivo intenta reensamblar estos fragmentos, puede provocar un desbordamiento de búfer, lo que lleva a un bloqueo del sistema, reinicio o comportamiento inestable. Aunque la mayoría de los sistemas modernos están protegidos contra este tipo de ataque, sigue siendo una amenaza potencial para sistemas desactualizados o mal configurados [13].

6.2.2. *Volumetric attack*

Los ataques volumétricos son un tipo de ataque DoS que se centran en consumir el ancho de banda disponible de la red o del sistema objetivo. Estos ataques generan una gran cantidad de tráfico malicioso, como paquetes ICMP, UDP o TCP, con el fin de saturar la capacidad de la red y hacer que los servicios legítimos sean inaccesibles. Los ataques volumétricos pueden ser difíciles de mitigar, ya que el tráfico malicioso puede parecer similar al tráfico legítimo [13].

6.2.3. *Flooding attack*

Un ataque de inundación (*flooding attack*) es un tipo de ataque DoS que implica enviar una gran cantidad de tráfico o solicitudes a un sistema objetivo con el fin de abrumarlo y agotar sus recursos. Esto puede hacer que el sistema se vuelva lento, inestable o incluso inaccesible para los usuarios legítimos [13].

Existen varios tipos comunes de ataques de inundación:

- *Ping of death*: el *Ping of death* es un ataque DoS que implica enviar paquetes ICMP malformados o fragmentados a un sistema objetivo. Estos paquetes, al ser demasiado grandes o estar mal formados, pueden causar que el sistema se bloquee, se reinicie o se vuelva inestable al intentar procesarlos. Aunque la mayoría de los sistemas modernos están protegidos contra este tipo de ataque, sigue siendo una amenaza potencial para sistemas desactualizados o mal configurados [11].
- SYN *Flood*: un ataque de inundación SYN es un tipo de ataque DoS que explota el proceso de establecimiento de conexiones TCP. En este ataque, el atacante envía una gran cantidad de solicitudes SYN (solicitudes de conexión) a un servidor objetivo, pero no completa el proceso de conexión enviando el paquete ACK correspondiente. Esto provoca que el servidor mantenga abiertas múltiples conexiones semi-abiertas, agotando sus recursos y haciéndolo incapaz de atender nuevas solicitudes legítimas [11].
- UDP *Flood*: un ataque de inundación UDP es un tipo de ataque DoS que implica enviar una gran cantidad de paquetes UDP a un sistema objetivo. Estos paquetes suelen estar dirigidos a puertos aleatorios en el sistema, lo que provoca que el sistema intente responder con mensajes ICMP "Destination Unreachable". La sobrecarga de estos mensajes puede agotar los recursos del sistema y hacer que se vuelva inaccesible para los usuarios legítimos [11].
- HTTP *Flood*: un ataque de inundación HTTP es un tipo de ataque DoS que implica enviar una gran cantidad de solicitudes HTTP a un servidor web objetivo. Estas solicitudes pueden ser simples (como solicitudes *GET*) o más complejas (como solicitudes POST con datos). El objetivo del ataque es agotar los recursos del servidor web, como la CPU, la memoria o el ancho de banda, haciendo que el sitio web se vuelva lento o inaccesible para los usuarios legítimos [11].

6.3. Segmentación de la red

La segmentación de la red consiste en dividir una red en varios segmentos y que cada uno actúe como su propia red pequeña con el objetivo de tener mayor control y aumentar la seguridad [14]. Por lo general, cada segmento es aislado por un *firewall*, para controlar el flujo de tráfico. La segmentación se realiza por lo general usando VLANs, subredes o ACLs de *firewalls* que segregan el tráfico. La idea detrás de la segmentación de red es que diferentes tipos de dispositivos o usuarios se encuentren en diferentes segmentos, y solo se permita el tráfico necesario entre segmentos. Esto limita la capacidad de un atacante para

moverse lateralmente a través de la red si un segmento se ve comprometido. Realizar la segmentación de forma correcta evita que una brecha menor se convierta en un problema mayor. Existen dos tipos de segmentación:

6.3.1. Segmentación física

La segmentación física es dividir una red más grande en un conjunto de redes más pequeñas que, a menudo, se le llaman subredes. Un dispositivo físico de red actúa como puerta de entrada a dicha subred, por lo general es un *firewall*, controlando qué tráfico entra y sale [8].

6.3.2. Segmentación lógica

La segmentación lógica crea subredes a través de redes virtuales", es más flexible que la física porque no necesita cableado ni movimiento físico de dispositivos de red para crear la subred, y la automatización puede simplificar la configuración de las redes más pequeñas. Esto se logra separando los dispositivos en grupos lógicos, lo que se conoce como una LAN virtual (VLAN). Una VLAN permite agrupar lógicamente a usuarios dispersos, aunque estén conectados físicamente en distintos puntos de la red [8].

6.3.3. *Demilitarized zone* (DMZ)

Una *demilitarized zone* (DMZ) es un tipo de subred que se encuentra entre la red interna de una organización (segura) y la red externa (no segura), como Internet. La DMZ permite que dicha organización acceda a estas redes externas, garantizando al mismo tiempo la seguridad de su red privada o LAN y proteger la red interna de posibles amenazas. Los servidores públicos, como servidores web, servidores de correo electrónico y servidores DNS, suelen estar ubicados en la DMZ para que los usuarios externos puedan acceder a estos servicios y no poder infiltrarse y comprometer la seguridad de la red interna. Comúnmente se implementa mediante *firewalls* que controlan el tráfico entre la red interna, la DMZ y el exterior [15].

6.3.4. *Zero trust*

La arquitectura de *Zero Trust* es un enfoque de seguridad que asume que las amenazas pueden estar tanto dentro como fuera de la red; intenta eliminar el concepto de confianza de la arquitectura de red dentro de una organización. *Zero Trust* requiere autenticación y autorización para cada acceso a recursos, independientemente de la ubicación del usuario o del dispositivo mediante el cual accede. Esto implica que el usuario debe continuamente verificar su identidad y el estado de seguridad de los dispositivos antes de obtener acceso a los recursos [8].

6.4. *Intrusion detection and prevention systems* (IDS/IPS)

Aunque los *firewalls* controlan el acceso basándose en puertos, direcciones IP y protocolos, pueden no inspeccionar lo suficientemente a fondo el contenido del tráfico como para detectar ataques sofisticados. Un *Intrusion detection system* (IDS) es una herramienta de monitoreo de seguridad que inspecciona la actividad en redes o sistemas en busca de patrones maliciosos y violaciones de políticas, que alerta cuando se detectan ciertos eventos sospechosos. Los IDS generalmente se consideran pasivos; monitorean e informan, pero no bloquean el tráfico por sí mismos. En contraste, un *Intrusion prevention system* (IPS) es también una herramienta, pero a diferencia de un IDS, toma acciones automáticas para bloquear o prevenir la actividad maliciosa detectada. Los IPS son considerados activos, ya que pueden tomar medidas para prevenir ataques en tiempo real. Ambos sistemas pueden ser implementados en la red o en el *host*, y pueden utilizar diferentes métodos de detección basados en el comportamiento [8].

6.5. *Firewall* pfSense

El software pfSense nació como un *fork* del proyecto de código abierto *m0n0wall* en 2004. *m0n0wall* proporcionaba un *firewall/router* para dispositivos integrados y estaba dimensionado para cuando se tenían recursos limitados [16].

El proyecto pfSense es una distribución gratuita de FreeBSD, que es un sistema operativo de código abierto, adaptada para su uso como *firewall* y *router* gestionado por una interfaz web fácil de usar. pfSense ofrece una interfaz web intuitiva que permite configurar y gestionar el *firewall* sin necesidad de conocimientos avanzados de comandos. Es personalizable y contiene bastantes complementos y paquetes adicionales que pueden ampliar su funcionalidad, como VPN, IDS/IPS Snort, filtrado de contenido, etc. [16].

6.6. Snort

Snort es un sistema de detección y prevención de intrusiones (IDS/IPS) de código abierto, desarrollado por Cisco (anteriormente por *Sourcefire*), que se utiliza para monitorear el tráfico de red en tiempo real. Es capaz de realizar análisis de tráfico en tiempo real y registro de paquetes en redes IP y con ello realizar análisis de protocolos, búsqueda/comparación de contenidos; además, detecta diferentes tipos de ataques y sondas, como desbordamientos de búfer, escaneos de puertos sigilosos, ataques CGI, sondas SMB, etc. [17].

Snort utiliza un lenguaje de reglas para describir el tráfico a analizar, y dichas reglas son una parte importante de su implementación, ya que definen qué tipos de tráfico deben ser monitoreados y qué acciones deben tomarse en respuesta a eventos específicos. Estas reglas pueden ser personalizadas y actualizadas para adaptarse a nuevas amenazas y vulnerabilidades [18].

6.6.1. Arquitectura de Snort

La arquitectura modular de Snort permite insertar componentes para decodificar paquetes, normalizar tráfico, aplicar reglas de detección y generar salidas. A alto nivel, el *pipeline* de procesamiento es:

1. *Sniffer*: Snort utiliza la DAQ (*Data acquisition library*) para capturar paquetes de red. La DAQ sustituye las llamadas directas a las funciones libpcap por una capa de abstracción que facilita el funcionamiento en una variedad de interfaces de hardware y software sin necesidad de realizar cambios en Snort. Es posible seleccionar el tipo y el modo de DAQ al invocar Snort para realizar operaciones de lectura pcap o en línea, etc.

2. Decodificador: el decodificador tiene la función de determinar qué protocolos subyacentes se utilizan en el paquete (como Ethernet, IP, TCP, etc.) y guarda estos datos junto con la ubicación de los datos de carga útil/aplicación en el paquete (que no intenta decodificar) y el tamaño de esta carga útil para que los utilicen el preprocesador y los motores de detección.

3. Preprocesadores: permiten ampliar la funcionalidad de Snort al permitir a los usuarios y programadores añadir *plugins* modulares a Snort con bastante facilidad. El código del preprocesador se ejecuta antes de que se active el motor de detección, pero después de que se haya decodificado el paquete. El paquete se puede modificar o analizar fuera de banda utilizando este mecanismo.

4. Motor de detección: el motor de detección realiza una comparación binaria entre cada paquete y cada regla de un conjunto de reglas predeterminado. En caso de que los paquetes se correspondan con el contenido de la regla, o viceversa, se dirigen a la salida.

5. Módulos de salida: la salida registrará y/o iniciará notificaciones en respuesta a la acción de la regla. Los registros se pueden almacenar en muchos formatos, como el formato syslog y unified2, y se pueden dirigir a numerosos destinos, incluido el almacenamiento directo en un repositorio de base de datos.

En Snort 2.x el archivo de configuración principal es `snort.conf`, que contiene las directivas de configuración y las rutas a los archivos de reglas [19].

6.6.2. Modos de operación

- *Sniffer*: únicamente lee los paquetes de red y los muestra en un flujo continuo en la consola.
- *Packet logger*: los paquetes se registran en el almacenamiento para su análisis posterior.
- *Network intrusion detection/prevention system* (NIDS/NIPS): inspecciona y analiza tráfico en tiempo real y genera alertas o bloquea tráfico malicioso.

6.6.3. Variables y bloques fundamentales

Snort utiliza un lenguaje de descripción de reglas sencillo y ligero que es flexible y bastante potente. La mayoría de las reglas de Snort se escriben en una sola línea.

Las reglas de Snort se dividen en dos secciones lógicas: el encabezado de la regla y las opciones de la regla. El encabezado de la regla contiene la acción de la regla, el protocolo, las direcciones IP y las máscaras de red de origen y destino, y la información de los puertos de origen y destino. La sección de opciones de la regla contiene mensajes de alerta e información sobre qué partes del paquete deben inspeccionarse para determinar si se debe aplicar la acción de la regla [19].

La forma canónica de una regla es:

Cuadro 1. Estructura de una regla de Snort.

```
accion protocolo IP_origen puerto_origen -> IP_destino puerto_destino (
    opciones)
```

Nota. Forma canónica de una regla de Snort, donde se especifica la acción, protocolo, direcciones IP y puertos de origen y destino, junto con las opciones de detección. Elaboración propia.

6.6.4. Acciones y direccionalidad

El *rule header* contiene la información que define el quién, dónde y qué de un paquete, así como qué hacer en caso de que aparezca un paquete con todos los atributos indicados en la regla. El primer elemento de una regla es la *rule action*. La *rule action* le indica a Snort qué hacer cuando encuentra un paquete que coincide con los criterios de la regla. Hay 3 acciones predeterminadas disponibles en Snort: *alert*, *log*, *pass*. Además, si se ejecuta Snort en *inline mode*, tienes opciones adicionales que incluyen *drop*, *reject* y *sdrop* [19].

- *alert* — genera una alerta usando el método de alerta seleccionado y luego registra el paquete.
- *log* — registra el paquete.
- *pass* — ignora el paquete.
- *drop* — bloquea y registra el paquete.
- *reject* — bloquea el paquete, lo registra y luego envía un *TCP reset* si el protocolo es TCP, o un mensaje *ICMP port unreachable* si el protocolo es UDP.
- *sdrop* — bloquea el paquete pero no lo registra.

El *direction operator* `->` indica la orientación, o dirección, del tráfico al que aplica la regla. La dirección IP y los números de puerto a la izquierda del *direction operator* se consideran

como el tráfico proveniente del *host* de origen, y la información de dirección y puerto a la derecha del operador corresponde al *host* de destino. También existe un *bidirectional operator*, indicado con el símbolo <>. Esto le indica a Snort que considere los pares dirección/puerto en cualquiera de las orientaciones, ya sea como origen o como destino [19].

6.6.5. Opciones de reglas

Cuadro 2. Palabras clave de opciones de regla (Snort).

Keyword	Descripción
`msg`	La palabra clave `msg` indica al motor de registro y alertas el mensaje que se imprimirá junto con el volcado del paquete o la alerta.
`reference`	La palabra clave `reference` permite que las reglas incluyan referencias a sistemas externos de identificación de ataques.
`gid`	La palabra clave `gid` (*generator id*) se utiliza para identificar qué parte de Snort genera el evento cuando se dispara una regla en particular.
`sid`	La palabra clave `sid` se utiliza para identificar de forma única las reglas de Snort.
`rev`	La palabra clave `rev` se utiliza para identificar de forma única las revisiones de las reglas de Snort.
`classtype`	La palabra clave `classtype` se usa para categorizar una regla como la detección de un ataque que pertenece a una clase más general de ataques.
`priority`	La palabra clave `priority` asigna un nivel de severidad a las reglas.
`metadata`	La palabra clave `metadata` permite al autor de la regla incrustar información adicional sobre la regla, típicamente en formato *clave–valor*.

Nota. Resumen de las principales palabras clave disponibles en las opciones de regla de Snort, con su respectiva descripción funcional. Elaboración propia.

6.7. SNMP

Simple Network Management Protocol (SNMP) es un protocolo de la capa de aplicación del modelo OSI diseñado para supervisar y gestionar dispositivos en redes IP: *routers*, *switches*, servidores, impresoras, UPS, sensores, etc. Su modelo lógico separa un gestor de múltiples agentes. El gestor recopila datos (*polling*) y recibe notificaciones asíncronas (*traps*); los agentes exponen información estructurada mediante una base de información llamada MIB [20].

El transporte que utiliza SNMP es UDP (puertos 161 para consultas y 162 para *traps*). SNMP es ampliamente soportado por dispositivos de red y sistemas operativos, lo que lo convierte en un estándar de facto para la gestión de redes.

Los agentes SNMP exponen los datos de gestión de los sistemas gestionados como variables. El protocolo también permite tareas de gestión activa, como cambios de configuración, mediante la modificación remota de estas variables. Las variables accesibles a través de SNMP están organizadas en jerarquías. El SNMP en sí mismo no define qué variables debe ofrecer un sistema gestionado. Más bien, el SNMP utiliza un diseño extensible que permite a las aplicaciones definir sus propias jerarquías. Estas jerarquías se describen como una base de información de gestión (MIB). Las MIB describen la estructura de los datos de gestión de un subsistema de dispositivos; utilizan un espacio de nombres jerárquico que contiene identificadores de objetos (OID). Cada OID identifica una variable que se puede leer o configurar a través de SNMP. Los OID se representan como una secuencia de enteros separados por puntos, que reflejan la posición del objeto en la jerarquía de la MIB [21].

6.7.1. Versiones y seguridad

Cuadro 3. Comparativa de versiones de SNMP (seguridad, capacidades y uso recomendado).

Versión	Descripción
SNMPv1	Versión original, simple y limitada. Soporta solo operaciones básicas (*GET*, *SET*, *TRAP*). Utiliza *community strings* en claro para autenticación, lo que la hace insegura. Adecuada solo para entornos controlados y no sensibles.
SNMPv2c	Mejora el rendimiento y añade nuevas operaciones. Sin embargo, mantiene la misma seguridad débil basada en *community strings* en claro. Aún se usa en redes existentes, pero no es recomendable para nuevas implementaciones.
SNMPv3	Introduce mejoras significativas en seguridad, incluyendo autenticación (MD5, SHA) y cifrado (DES, AES). Soporta control de acceso basado en roles y ofrece integridad y confidencialidad de los datos. Recomendado para todas las nuevas implementaciones debido a sus robustas características de seguridad.

Nota. Comparativa de las tres versiones de SNMP, destacando las diferencias en autenticación, cifrado y casos de uso recomendados. Elaboración propia.

6.8. Zabbix

Zabbix es una solución de monitoreo distribuido de código abierto de clase empresarial que monitorea numerosos parámetros de una red y el estado e integridad de servidores, máquinas virtuales, aplicaciones, servicios, bases de datos, sitios web, y más. Zabbix utiliza un mecanismo de notificación flexible que permite a los usuarios configurar alertas prácticamente para cualquier evento, lo que posibilita una reacción rápida ante problemas en los servidores. Zabbix ofrece excelentes capacidades para realizar reportes y visualización de datos basadas en los datos almacenados [22].

Zabbix soporta tanto *polling* como *trapping*. Todos los reportes y estadísticas de Zabbix,

así como los parámetros de configuración, se acceden mediante la interfaz web. Zabbix puede desempeñar un papel importante en el monitoreo de la infraestructura de TI, tanto en organizaciones pequeñas con pocos servidores como en grandes empresas con multitud de servidores [22].

6.8.1. Arquitectura de Zabbix

Cuadro 4. Componentes de la arquitectura de Zabbix.

Componente	**Descripción**
Zabbix *Server*	Componente central que gestiona la configuración, recolección, almacenamiento y procesamiento de datos. Se comunica con *agents*, *proxies* y otros servicios para recolectar métricas y eventos. Maneja alertas, notificaciones y reportes. Requiere una base de datos para almacenar datos.
Base de datos	Almacena toda la configuración, datos históricos y tendencias. Soporta MySQL/MariaDB, PostgreSQL, Oracle y SQLite. La elección depende de la escala, rendimiento y características deseadas.
Zabbix *Frontend* (*Web interface*)	Interfaz web basada en PHP para administración, configuración y visualización de datos. Proporciona paneles personalizables, gráficos, mapas, informes y gestión de alertas. Requiere un servidor web (Apache/Nginx) y PHP.
Zabbix *Proxy*	Componente intermedio que recolecta datos de *agents* en redes remotas o segmentadas, y los envía al *Server*. Permite escalabilidad, reduce carga del *Server* y minimiza tráfico entre sitios. Puede operar en modo activo o pasivo.
Zabbix *Agents* (*Agent* / *Agent 2*)	Software ligero instalado en *hosts* monitoreados que recolecta métricas (CPU, memoria, disco, servicios) y reporta al *Server* o *Proxy*. Soporta modos activo y pasivo, y puede extenderse con *plugins* (*Agent 2*).

Nota. Descripción de los componentes principales de la arquitectura de Zabbix y su función dentro del ecosistema de monitoreo. Elaboración propia.

6.8.2. Modelo de objetos: *hosts*, ítems, *triggers*, plantillas

El modelo de objetos de Zabbix está diseñado para ser flexible y escalable, permitiendo a los administradores definir y organizar los elementos que se desean monitorear. Los principales objetos en Zabbix son:

6.8.2.1. *Host*

Un *host* representa cualquier dispositivo o sistema que se desea monitorear, como servidores, estaciones de trabajo, dispositivos de red, aplicaciones, etc. Cada *host* tiene varios

atributos clave:

- Nombre y dirección IP o nombre DNS para identificarlo en la red.
- Grupo de *hosts* para organizar y gestionar múltiples *hosts* de forma colectiva.
- Interfaces que definen cómo se comunica el *host* con el *Server* (Agente, SNMP, IPMI, JMX).
- Asociaciones con plantillas que contienen configuraciones predefinidas de ítems, triggers, gráficos, etc.

6.8.2.2. *Item* (métrica)

Un *item* es una métrica específica que se recolecta de un *host*. Cada *item* tiene varios atributos clave:

- Tipo de *item* que define el método de recolección.
- Clave que identifica de forma única el *item* y define qué dato se recolecta (por ejemplo, `system.cpu.load[percpu,avg1]` para la carga de CPU).
- Intervalo de actualización que especifica con qué frecuencia se recolecta el dato.
- Tipo de información (numérica, texto, log, etc.) y unidades (por ejemplo, porcentaje, bytes).
- Historial y retención de tendencias para definir cuánto tiempo se almacenan los datos históricos y las tendencias.

6.8.2.3. *Trigger* (alerta)

Un *trigger* define una condición lógica basada en uno o más *items* que, cuando se cumple, indica un problema potencial o real en el *host*. Cada *trigger* tiene varios atributos clave:

- Expresión que define la condición que activa el *trigger*.
- Severidad que clasifica la importancia del *trigger* (información, advertencia, alto, crítico).
- Dependencias que permiten encadenar *triggers* para evitar alertas redundantes.
- Mensaje de alerta que se envía cuando el *trigger* se activa.

6.8.3. *Templates*

Colección predefinida de ítems, triggers, gráficos, *discovery rules* y *web scenarios* que se pueden aplicar a múltiples *hosts*. Facilitan la estandarización y despliegue rápido de monitoreo para tecnologías comunes (Linux, Windows, MySQL, Apache, etc.). Las plantillas pueden heredar otras plantillas para reutilizar configuraciones.

6.8.4. Métodos de recolección

Zabbix soporta múltiples métodos para recolectar datos de los *hosts* monitoreados, cada uno con sus ventajas y casos de uso ideales.

6.8.4.1. Agente pasivo

El *agent* espera solicitudes del *Server* para enviar datos. Adecuado para redes confiables y de baja latencia, donde el *Server* puede alcanzar a los agentes fácilmente.

6.8.4.2. Agente activo

El *agent* inicia la conexión y envía valores al *Server*. Ideal en redes restringidas (salida permitida, entrada bloqueada) o entornos distribuidos, o en casos que no se quiera saturar al *Server* para realizar *polling* frecuente.

6.8.4.3. *Agent* 2 (*plugins*)

Agent 2 es una reescritura del agente original en Go, que ofrece mayor flexibilidad y extensibilidad mediante *plugins*. Los *plugins* permiten añadir nuevas funcionalidades sin recompilar el agente, facilitando el monitoreo de tecnologías emergentes o personalizadas.

6.8.5. SNMP

Zabbix puede actuar como un gestor SNMP, recolectando datos de dispositivos de red y sistemas que exponen información vía SNMP. Soporta SNMPv1, v2c y v3, permitiendo monitorear *routers*, *switches*, impresoras, UPS y otros equipos. Zabbix puede realizar *polling* de OID específicos y recibir *traps* SNMP para alertas asíncronas.

6.9. OpenVAS

OpenVAS (*Open vulnerability assessment scanner*) es un escáner de vulnerabilidades de código abierto usado para identificar y evaluar debilidades en sistemas, redes y aplicaciones. Forma parte del ecosistema *Greenbone vulnerability management* (GVM), que provee una

plataforma integral para auditorías de seguridad con pruebas autenticadas y no autenticadas, cobertura de protocolos de red (IP) y también de algunos protocolos industriales, afinado de rendimiento para exploraciones a gran escala, y un lenguaje interno para implementar pruebas de vulnerabilidad (*Network vulnerability tests*). Estas pruebas se actualizan de forma continua mediante los *feeds* de Greenbone para incorporar nuevas amenazas y vulnerabilidades conocidas [23].

6.9.1. *Greenbone community edition*

Greenbone community edition abarca las versiones actuales del marco de aplicaciones para el análisis y la gestión de vulnerabilidades, siendo el equivalente *open source* de la línea comercial *Greenbone enterprise*. Esta versión es adoptada por terceros externos; por ejemplo, si el marco de software es proporcionado por una distribución Linux, se compila a partir de la *Greenbone community edition*.

6.9.2. Arquitectura de *Greenbone community edition*

OpenVAS *Community edition* consiste en un marco con varios servicios.

Consta del *Greenbone vulnerability management daemon* (gvmd), el *Greenbone security assistant* (GSA) con el *Greenbone Security Assistant Daemon* (gsad) y la aplicación ejecutable de escaneo que realiza pruebas de vulnerabilidad (VT) contra los sistemas objetivo [24].

6.9.2.1. *Greenbone vulnerability management daemon* (gvmd)

El *Greenbone vulnerability management daemon* (gvmd), también llamado *Greenbone vulnerability manager*, es el servicio central que convierte el escaneo simple de vulnerabilidades en una solución completa de gestión de vulnerabilidades. Controla el OpenVAS *Scanner* mediante el *Open scanner protocol*, y está basado en XML, sin estado (*stateless*) y no requiere una conexión permanente para la comunicación. Este servicio también controla una base de datos SQL (PostgreSQL) donde se almacenan de forma centralizada toda la configuración y los resultados de los escaneos. Además, gestiona a los usuarios, incluyendo el control de permisos con grupos y roles. Por último, el servicio cuenta con un sistema interno de ejecución para tareas programadas y otros eventos [24].

6.9.2.2. *Greenbone security assistant* (GSA)

Greenbone security assistant (GSA) es la interfaz web con la que un usuario controla los escaneos y accede a la información de vulnerabilidades. Es el punto principal de contacto del usuario con el servicio. Se conecta a gvmd a través del servidor web *Greenbone security assistant daemon* para proporcionar una aplicación web completa de gestión de vulnerabilidades. La comunicación se realiza usando el *Greenbone management protocol*, con el cual el usuario también puede comunicarse directamente utilizando diferentes herramientas [24].

6.9.2.3. OpenVAS *Scanner*

El escáner principal, OpenVAS *Scanner*, es un motor de escaneo completo que ejecuta pruebas de vulnerabilidad contra sistemas objetivo. Para ello utiliza los *feeds* actualizados diariamente y completos: el OpenVAS *Enterprise feed*, completo y comercial, o el OpenVAS *Community feed*, disponible de forma gratuita. El escáner consta de los componentes *ospd-openvas* y *openvas-scanner*. El OpenVAS *Scanner* se controla mediante OSP. El *OSP daemon* para el OpenVAS *Scanner* se comunica con gvmd a través de OSP: se recopilan los datos de las pruebas de vulnerabilidad, se inician y detienen escaneos, y los resultados se transfieren a gvmd mediante ospd [24].

6.9.2.4. *openvasd*

Para acelerar las *local security checks* (LSC), *openvas-scanner* envía los paquetes recopilados a openvasd. La transmisión se realiza vía HTTP y el *listener* de *openvasd* está enlazado a *localhost*. Este servicio sustituye la lógica de potencialmente todas las *LSC* basadas en NASL. En lugar de ejecutar un VT *script* por cada *LSC*, el software instalado en un *host* se compara con una lista de software vulnerable conocido.

El OpenVAS Scanner regula la carga de cada NASL LSC de forma individual y lo ejecuta uno por uno para cada *host*. Luego se verifica una única vulnerabilidad conocida contra el software instalado. Esto se repite para todas las LSC. Con *openvasd*, la lista de software instalado se carga de la misma forma, pero se compara directamente con todo el software vulnerable conocido para el sistema operativo del *host* escaneado. Esto elimina la necesidad de ejecutar las *LSC*, ya que la información sobre el software vulnerable conocido se recopila en una sola lista, en lugar de estar distribuida en *scripts* NASL individuales.

6.9.3. *Vulnerability tests* (VT)

Vulnerability Tests (VT), también conocidos como *Network vulnerability tests* (NVT), son *scripts* escritos en el lenguaje de programación NASL (*Nessus attack scripting language*) para detectar vulnerabilidades en *hosts* remotos [25]. Cada VT está diseñado para identificar una vulnerabilidad específica, y puede incluir:

- Descripción de la vulnerabilidad.
- Métodos de detección (escaneo de puertos, análisis de servicios, pruebas de configuración).
- Referencias a bases de datos externas (CVE, CPE, CERT, etc.).
- Puntuación de severidad (CVSS).
- Recomendaciones para mitigación o remediación.

6.9.4. CPE: enumeración de plataformas y productos

Common platform enumeration (CPE) es un método estandarizado para describir e identificar clases de aplicaciones, sistemas operativos y dispositivos de hardware presentes entre los activos informáticos de una empresa.

6.9.5. CVE: identificadores de vulnerabilidades

Common vulnerabilities and exposures (CVE) es un diccionario de vulnerabilidades y exposiciones de seguridad de la información de conocimiento público. A cada vulnerabilidad se le asigna un identificador único, compuesto por el año de publicación y un número simple, y sirve como referencia central [25].

6.9.6. CERT y avisos de seguridad

El *Computer emergency response team* (CERT) es una organización que coordina respuestas a incidentes de seguridad informática. CERT publica avisos de seguridad que alertan sobre vulnerabilidades críticas, *exploits* y mejores prácticas para mitigar riesgos. Estos avisos son una fuente valiosa de información para administradores de sistemas y profesionales de seguridad. GVM integra referencias a avisos de CERT en sus NVT para proporcionar contexto adicional sobre las vulnerabilidades detectadas.

6.9.7. CVSS: puntuación de severidad

El *Common vulnerability scoring system* (CVSS) es un estándar de la industria para describir la gravedad de los riesgos de seguridad en sistemas informáticos. Los riesgos de seguridad se califican y comparan utilizando distintos criterios. Esto permite crear una lista priorizada de contramedidas [25].

6.10. Metasploitable 3

Metasploitable 3 es una máquina virtual intencionalmente vulnerable creada por Rapid7 para servir como objetivo de pruebas en formación, investigación y desarrollo de herramientas de seguridad ofensiva, especialmente con el *Metasploit framework*. A diferencia de un sistema endurecido, Metasploitable incorpora deliberadamente múltiples fallas explotables para practicar reconocimiento, explotación, post-explotación y técnicas de mitigación en un entorno controlado y legal [26].

6.10.1. Sistemas operativos

Metasploitable 3 está disponible en dos versiones principales:

- Metasploitable 3 Linux: basada en Ubuntu 14.04, incluye una variedad de servicios y aplicaciones vulnerables, como servidores web, bases de datos y servicios de red.
- Metasploitable 3 Windows: basada en Windows *Server* 2008 R2, también contiene múltiples vulnerabilidades en servicios y aplicaciones comunes.

6.10.2. Vulnerabilidades y servicios

Ambas versiones de Metasploitable 3 están configuradas con una amplia gama de vulnerabilidades conocidas, incluyendo:

- Servicios de red con configuraciones inseguras.
- Aplicaciones web vulnerables a ataques como SQL *Injection*, *Cross-Site scripting* (XSS) y *Remote code execution* (RCE).
- Servicios desactualizados con fallos conocidos.
- Configuraciones de seguridad débiles, como contraseñas por defecto y permisos excesivos.

6.11. Pi-hole

Pi-hole es un DNS *sinkhole* que protege sus dispositivos de contenidos no deseados, sin necesidad de instalar ningún software del lado del cliente [27].

Un DNS *sinkhole* es un servidor DNS que está configurado para distribuir direcciones no enrutables para un determinado conjunto de nombres de dominio o simplemente rechazar las peticiones de DNS [28].

Se utiliza para filtrar y bloquear anuncios, rastreadores y contenido no deseado en una red LAN. Al funcionar como un servidor DNS, Pi-hole intercepta las solicitudes de DNS de los dispositivos conectados a la red y bloquea las solicitudes a dominios conocidos. Esto significa que los anuncios no se cargan en los dispositivos, lo que mejora la velocidad de navegación y reduce el uso de ancho de banda [27].

6.12. nmap

Nmap (*Network Mapper*) es una herramienta de código abierto para la exploración y auditoría de seguridad de redes. Fue diseñada para escanear grandes redes, pero también funciona bien contra *hosts* individuales. Nmap utiliza paquetes IP sin procesar para determinar qué *hosts* están disponibles en la red, qué servicios (nombre y versión del programa) ofrecen esos *hosts*, qué sistemas operativos (y versiones del SO) están ejecutando, qué tipos de filtros/*firewalls* están en uso, y otras características [29].

Nmap es ampliamente utilizado por administradores de sistemas y profesionales de seguridad para tareas como inventario de red, gestión de actualizaciones de servicios y monitoreo de *host* o servicio [29].

CAPÍTULO 7

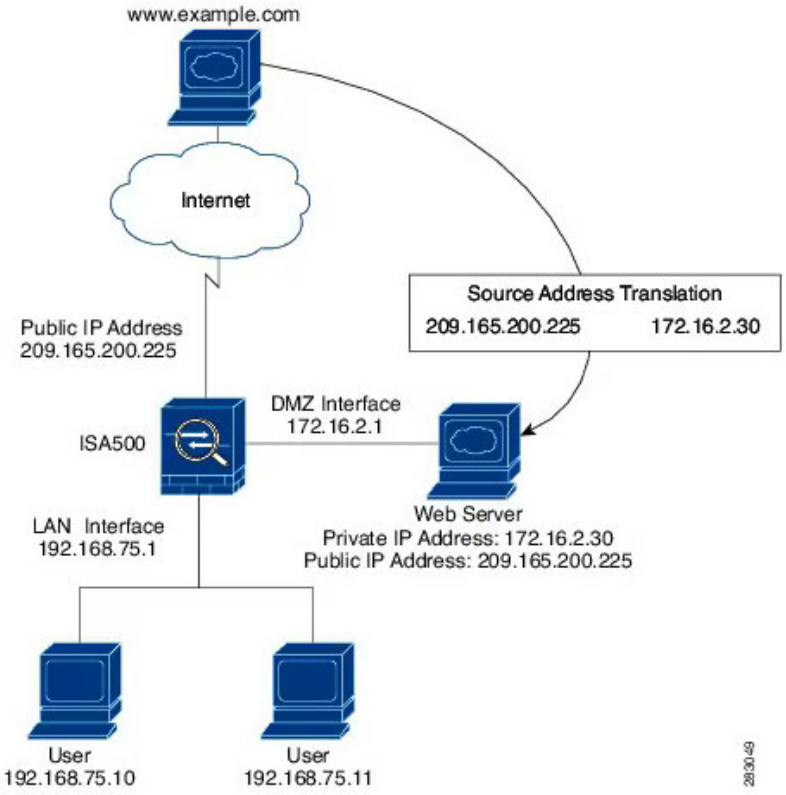
Instalación y configuración de los componentes a utilizar en el sistema propuesto

7.1. Estructura y funcionamiento del sistema

Previo a empezar la configuración de cualquier *software* es necesario definir la estructura del sistema. Se utiliza un sistema con cuatro interfaces de red: una para la red externa y las otras tres para la red interna que se encuentra segmentada. La interfaz de red externa (WAN) se conecta a la red local de la Universidad del Valle de Guatemala y da acceso a Internet, mientras que las interfaces de red internas se dividen entre segmentos que son: una red LAN, una red DMZ y una red que funciona como Centro de Operaciones de Seguridad (SOC) y Centro de Monitoreo (MOC).

El sistema propuesto se basa en la topología mostrada en la Figura 2, donde se tienen diferentes segmentos de red, emulando una red corporativa segmentada sencilla y segura como se muestra en la Figura 1; donde comúnmente se tiene una red LAN para los usuarios finales y una red DMZ para los servidores que deben ser accesibles desde fuera de la red. Cada segmento de red está alojado en una computadora física distinta y dentro de cada computadora se virtualizan varios sistemas operativos, con especificaciones de *hardware* distintos, correspondientes a su función.

Figura 1. Topología típica de red segmentada en una empresa.



Nota. La figura muestra una topología típica de red segmentada en una empresa, con los diferentes segmentos de red. Adaptada de [30].

En la topología a implementar [2] cada componente tiene una función específica dentro de la red. En la PC1 se instala pfSense, que actúa como *firewall* y *router* para controlar el tráfico entre la red externa y las redes internas. En él se configuran reglas de *firewall* para garantizar la seguridad de la red. Además, se instala Snort como IDS/IPS para detectar intrusiones en la red. Se crean reglas personalizadas para identificar patrones de ataque comunes como escaneos de puertos, intentos de acceso no autorizado y ataques de denegación de servicio (DoS). En base a estas reglas, Snort puede generar alertas que sean utilizadas por el sistema para tomar acciones preventivas.

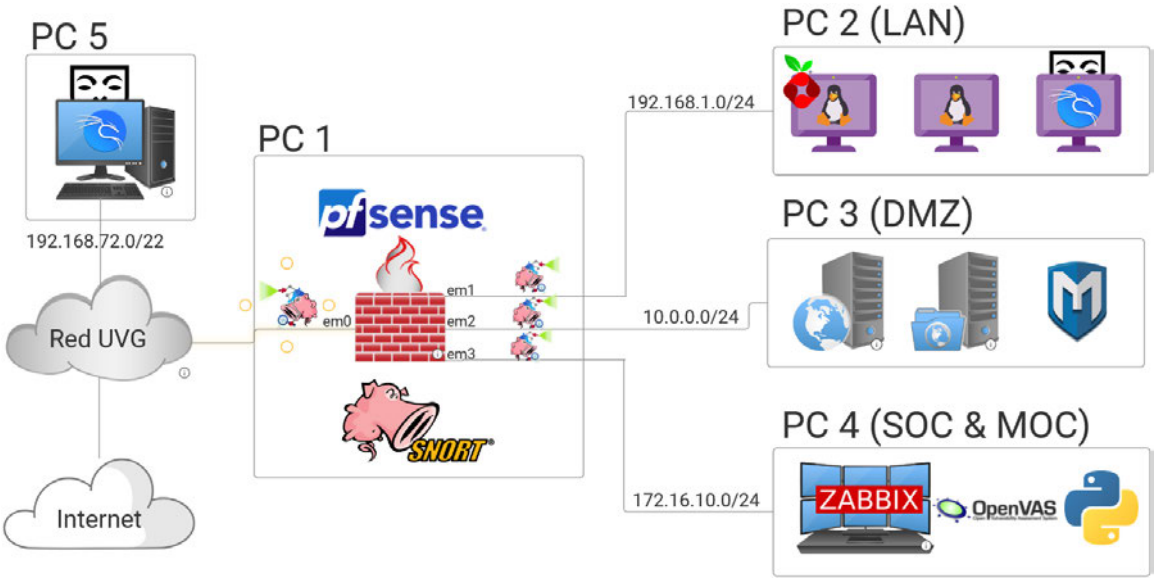
En la PC2 está el segmento LAN, que aloja a usuarios finales y dispositivos de la red interna, que son sujetos de prueba de los ataques y monitoreo. Además, es donde se despliega Pi-hole como servidor DNS para bloquear publicidad y rastreadores, mejorando la seguridad y privacidad de los usuarios. Por último, es donde también se encuentra un usuario con el sistema operativo Kali Linux, que se utiliza para simular ataques y probar la efectividad del IDS/IPS.

En la PC3 está el segmento DMZ, donde se instala la máquina virtual Metasploitable 3, que sirve como objetivo para los ataques de DoS, ya que trae servidores web y FTP configurados de manera vulnerable. Estos son accesibles desde la red externa, configurando reglas de *firewall* en pfSense para permitir el tráfico entrante a estos servicios.

En la PC4 está el segmento SOC & MOC, donde está el servidor de monitoreo Zabbix para supervisar el estado de la red y de todos los dispositivos conectados a ella, esto se realiza mediante agentes o por el protocolo SNMP. El servidor se configura para enviar alertas por correo electrónico en caso de detectar problemas o anomalías en la red. En conjunto se tiene un servidor para OpenVAS que se encarga de programar y realizar escaneos de vulnerabilidades en los *hosts* en la red y generar redes de los hallazgos encontrados. Por último, se tiene una estación Linux donde se está ejecutando un *script* en Python que se encarga de recibir las alertas generadas por Snort para así automatizar respuestas ante incidentes de seguridad como bloquear direcciones IP.

En la PC5 está el atacante externo, que se utiliza para simular ataques DoS desde fuera de la red. Esta PC tiene instalado Kali Linux, que es una distribución de Linux diseñada para pruebas de penetración y auditorías de seguridad. Desde esta PC se realizan ataques a los servicios expuestos en la DMZ.

Figura 2. Topología de la red a implementar.



Nota. La figura muestra la topología de la red segmentada (LAN, DMZ y SOC & MOC) utilizada como base para el despliegue de pfSense, Snort, monitoreo y servicios auxiliares. Elaboración propia.

Se estaba trabajando en un entorno virtualizado, por lo que se utiliza un *hypervisor* para crear las máquinas virtuales que representan los diferentes segmentos de la red. Se utiliza el *hypervisor* VMware Workstation Pro 17, que permite crear máquinas virtuales con diferentes configuraciones de red y recursos.

7.2. Despliegue de pfSense

Antes de comenzar hay que configurar las interfaces de red dentro de VMware Workstation Pro 17 para que se conecten a las redes correspondientes. En la PC1, dentro de VMware Workstation Pro 17, hay que dirigirse a la pestaña *Edit* y seleccionar *virtual network editor*. En esta sección se pueden configurar las interfaces de red virtuales que se utilizan en las máquinas virtuales. Se deben crear cuatro redes virtuales, una para cada interfaz de red que se utiliza en pfSense.

- Red externa (WAN): esta red se conecta a la red local de la Universidad del Valle de Guatemala y da acceso a Internet. Se configura como una red *bridged* para que se conecte a la tarjeta de red física de la computadora.
- Red interna (LAN): esta red se utiliza para los usuarios finales y se configura como una red interna.
- Red DMZ: esta red se utiliza para los servidores que deben ser accesibles desde fuera de la red y se configura como una red interna.
- Red SOC & MOC: esta red se utiliza para monitorear la red, generar alertas y se configura como una red interna.

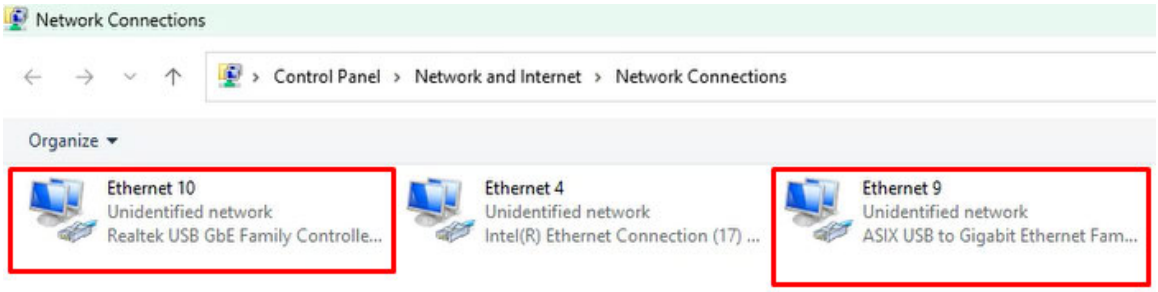
Al ser una computadora común de escritorio, por lo general se tiene únicamente un adaptador de red físico, por lo que los otros tres adaptadores de red son adaptadores de USB-C a Ethernet como se muestra en la figura 3, que se conectan a la computadora física y se configuran como adaptadores de red virtuales en VMware Workstation Pro 17. Estos adaptadores de red se deben configurar como *bridged* para que se conecten a la red interna y tengan acceso a los segmentos de red correspondientes.

Figura 3. Adaptadores de red físicos y USB-C a Ethernet.

Nota. La figura muestra los adaptadores de red físicos y USB-C a Ethernet que se conectan a la computadora física. Elaboración propia.

Para identificar los adaptadores de red en Windows, se puede ir al Panel de Control, luego a *Redes e Internet* y seleccionar *Centro de redes y recursos compartidos*. En esta sección se pueden ver los adaptadores de red disponibles en la computadora, como se muestra en la Figura 4. Aquí se pueden identificar los adaptadores de red físicos y los adaptadores USB-C a Ethernet que se conectan a la computadora física.

Figura 4. Identificador de adaptadores de red en Windows.

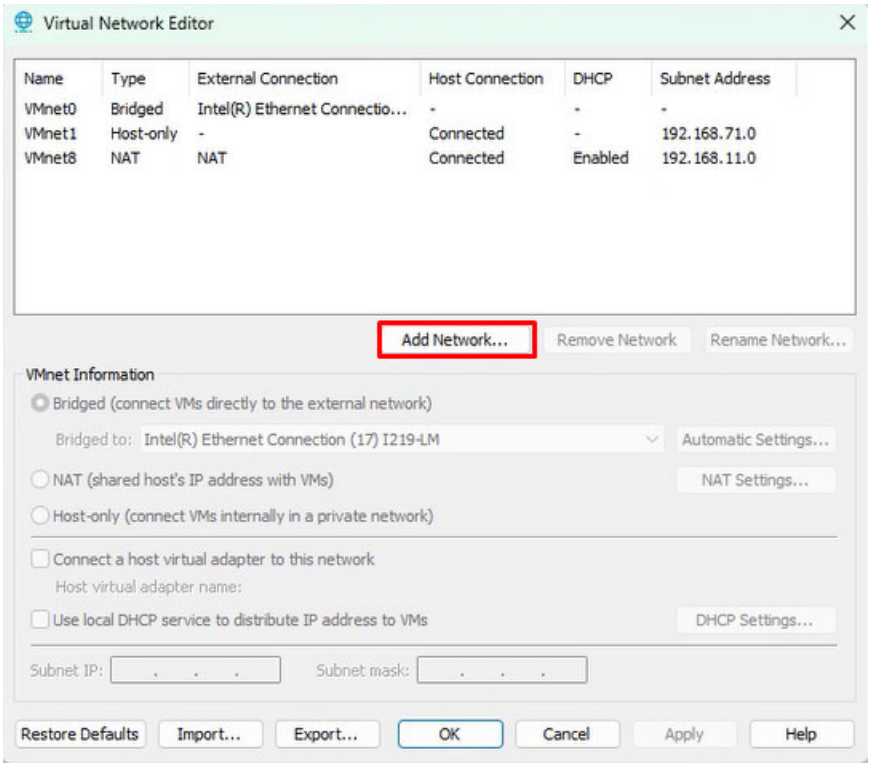


Nota. La figura muestra cómo el sistema operativo Windows enumera los adaptadores físicos y USB-C a Ethernet que luego se asignan a redes virtuales (WAN, LAN, DMZ y SOC) dentro de VMware. Elaboración propia.

7.2.1. Configuración de *virtual network editor* de VMware Workstation Pro 17

Ya que se detectaron los adaptadores de red en Windows, se configura VMware Workstation Pro 17 para que utilice estos adaptadores de red físicos y los asigne a las redes virtuales correspondientes. Para ello, se abre VMware Workstation Pro 17 y se dirige a la pestaña *Edit* y se selecciona *virtual network editor*. En esta sección se pueden ver las redes virtuales disponibles y se pueden crear nuevas redes virtuales, para crear una nueva red virtual se hace clic en el botón *Add Network* como se muestra en la Figura 5.

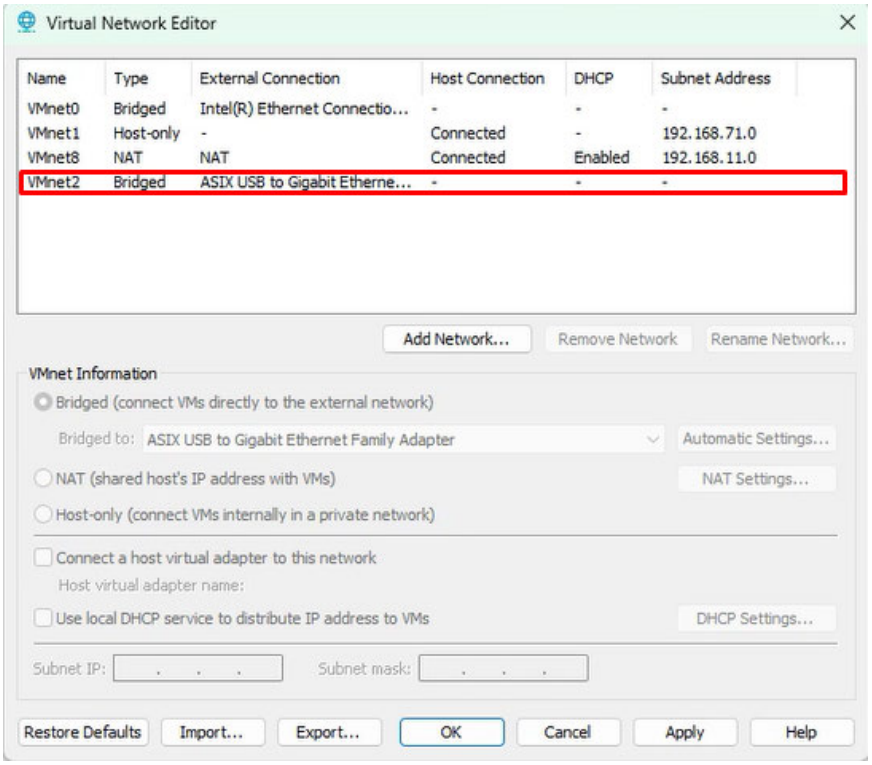
Figura 5. Configuración de *virtual network editor* de VMware Workstation Pro 17.



Nota. Se ilustra el proceso de añadir una nueva red virtual (ej. VMnet2) que es asociada a uno de los segmentos lógicos (LAN, DMZ o SOC) para aislar el tráfico en el laboratorio. Elaboración propia.

Se pregunta por el nombre de la red, se puede dejar el nombre por defecto, que es *VMnet2* que es para la red LAN, con el cuidado de recordar el nombre para la configuración de las máquinas virtuales. Luego se selecciona el tipo de conexión, en este caso, la opción *bridged*. Se selecciona la tarjeta de red física que se utiliza para la conexión, en este caso, uno de los adaptadores USB-C a Ethernet que se conectó a la computadora física.

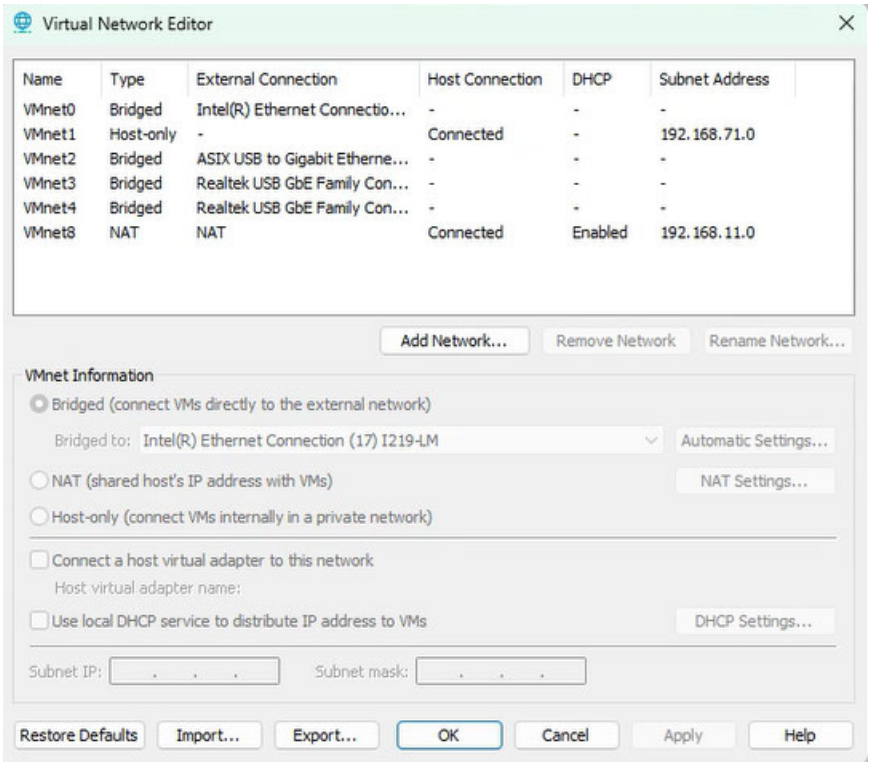
Figura 6. Selección del tipo de conexión y adaptador para red LAN (VMnet2).



Nota. Vista detallada de los parámetros seleccionados (tipo de conexión y adaptador físico puenteado) para una red virtual que facilite el enrutamiento segmentado en pfSense. Elaboración propia.

Se repite el proceso para las otras tres redes virtuales, cambiando el nombre de la red y seleccionando el adaptador USB-C a Ethernet correspondiente. Por ejemplo, para la red DMZ se puede utilizar el nombre *VMnet3* y para la red SOC & MOC se puede utilizar el nombre *VMnet4*.

Figura 7. Vista de redes virtuales configuradas (LAN, DMZ y SOC).

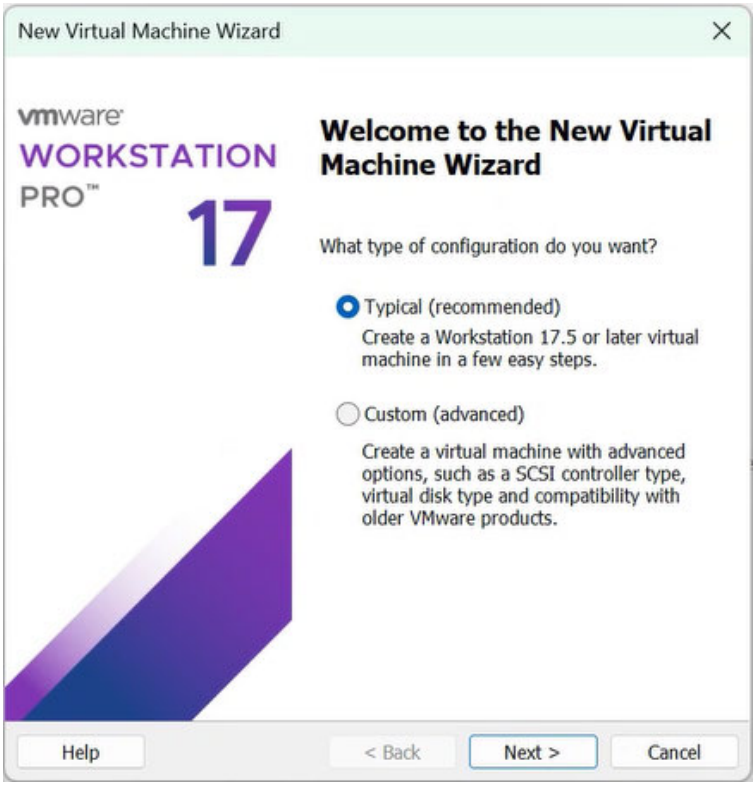


Nota. Se muestran redes virtuales adicionales creadas para mapear los diferentes dominios de broadcast requeridos (LAN, DMZ, SOC) en la arquitectura propuesta. Elaboración propia.

7.2.2. Instalación de pfSense

Para instalar pfSense, se descarga la imagen ISO desde el sitio web oficial de pfSense y se crea una nueva máquina virtual en VMware Workstation Pro 17. Durante la instalación, se siguen los siguientes pasos: se selecciona el tipo de configuración, seleccionando la opción típica.

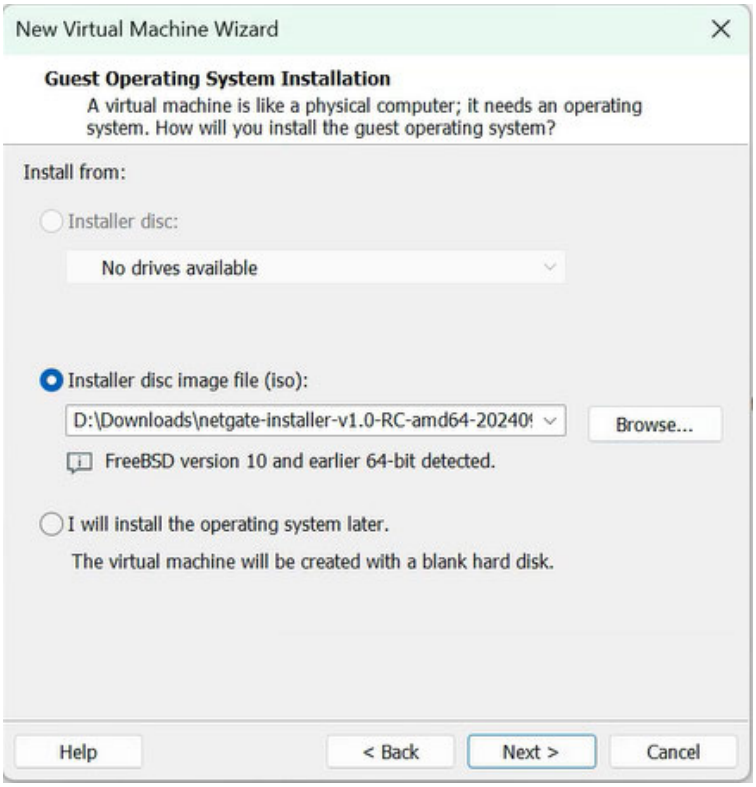
Figura 8. Configuración de máquina virtual para pfSense.



Nota. Pantalla inicial del asistente de creación de máquina virtual, donde se elige la configuración típica para desplegar pfSense de forma rápida en el laboratorio. Elaboración propia.

Se selecciona el tipo de sistema operativo, se selecciona la imagen ISO descargada de pfSense que es una distribución de FreeBSD 64-bit.

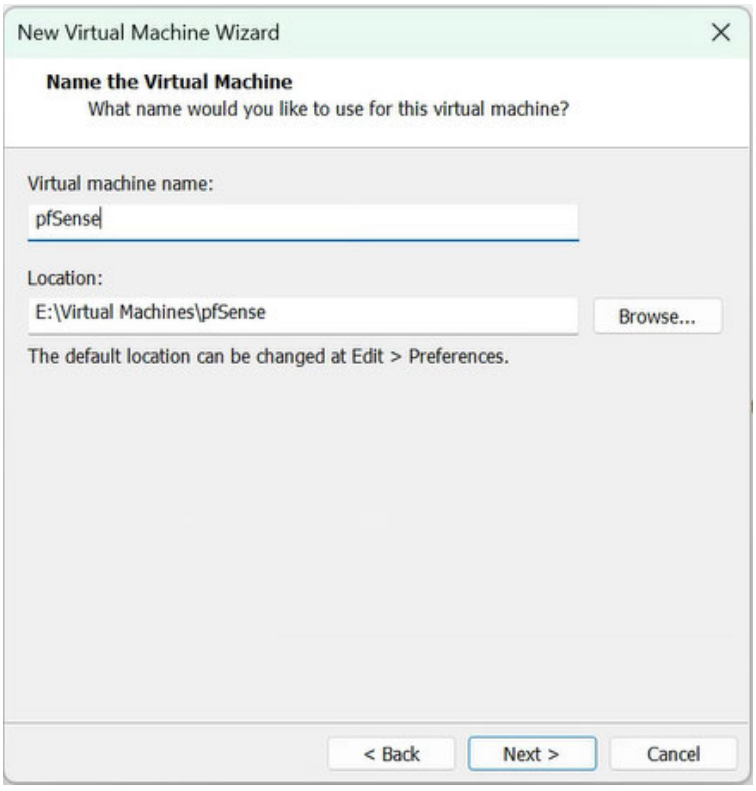
Figura 9. Selección de imagen ISO de pfSense: FreeBSD 64-bit.



Nota. Selección de la ISO de pfSense (FreeBSD 64-bit) que sirve como base del *firewall/router* central en la arquitectura. Elaboración propia.

Se debe asignar un nombre a la máquina virtual y seleccionar la ubicación donde se guardará.

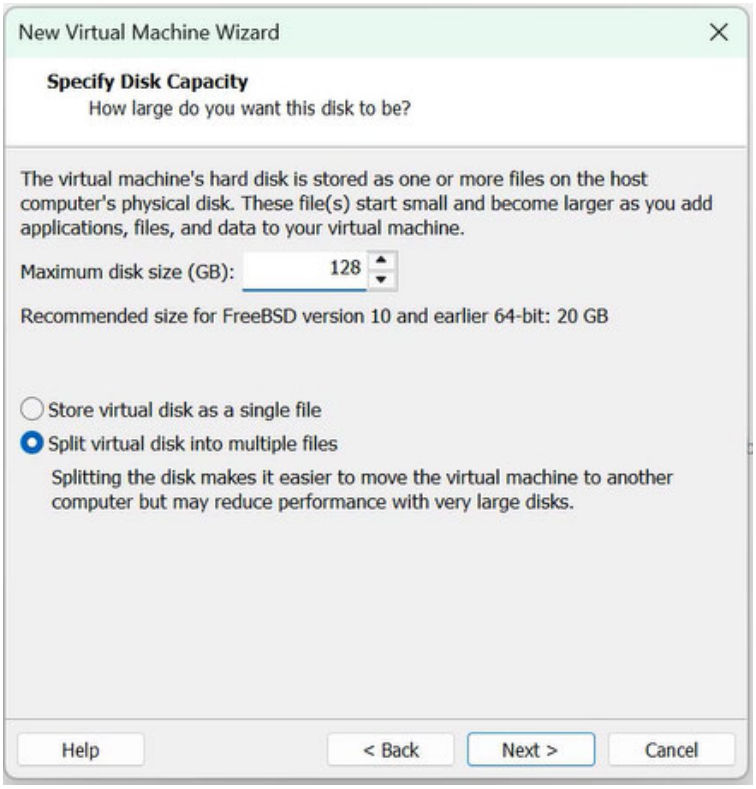
Figura 10. Asignación de nombre y ruta de almacenamiento para VM de pfSense.



Nota. Asignación de nombre y directorio de almacenamiento para la VM de pfSense, facilitando su identificación entre múltiples máquinas. Elaboración propia.

Se debe asignar la capacidad de disco duro, en este caso, un disco duro de 128 GB, que es suficiente para la instalación y configuración de pfSense.

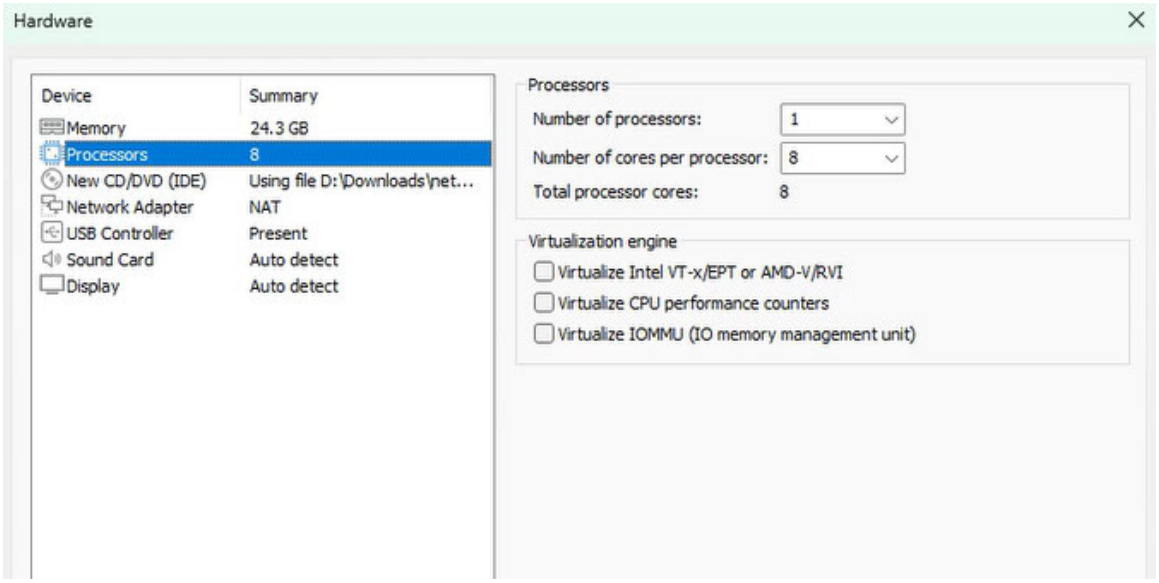
Figura 11. Especificación de capacidad de disco virtual: 128 GB para pfSense.



Nota. Definición del tamaño de disco virtual (128 GB) suficiente para el SO, paquetes como Snort y archivos de registro. Elaboración propia.

Se asigna la cantidad de memoria RAM, en este caso, 24 GB de RAM. Luego se selecciona la cantidad de procesadores, en este caso, un procesador con 8 núcleos.

Figura 12. Asignación de memoria RAM y procesadores (24 GB, 8 núcleos).

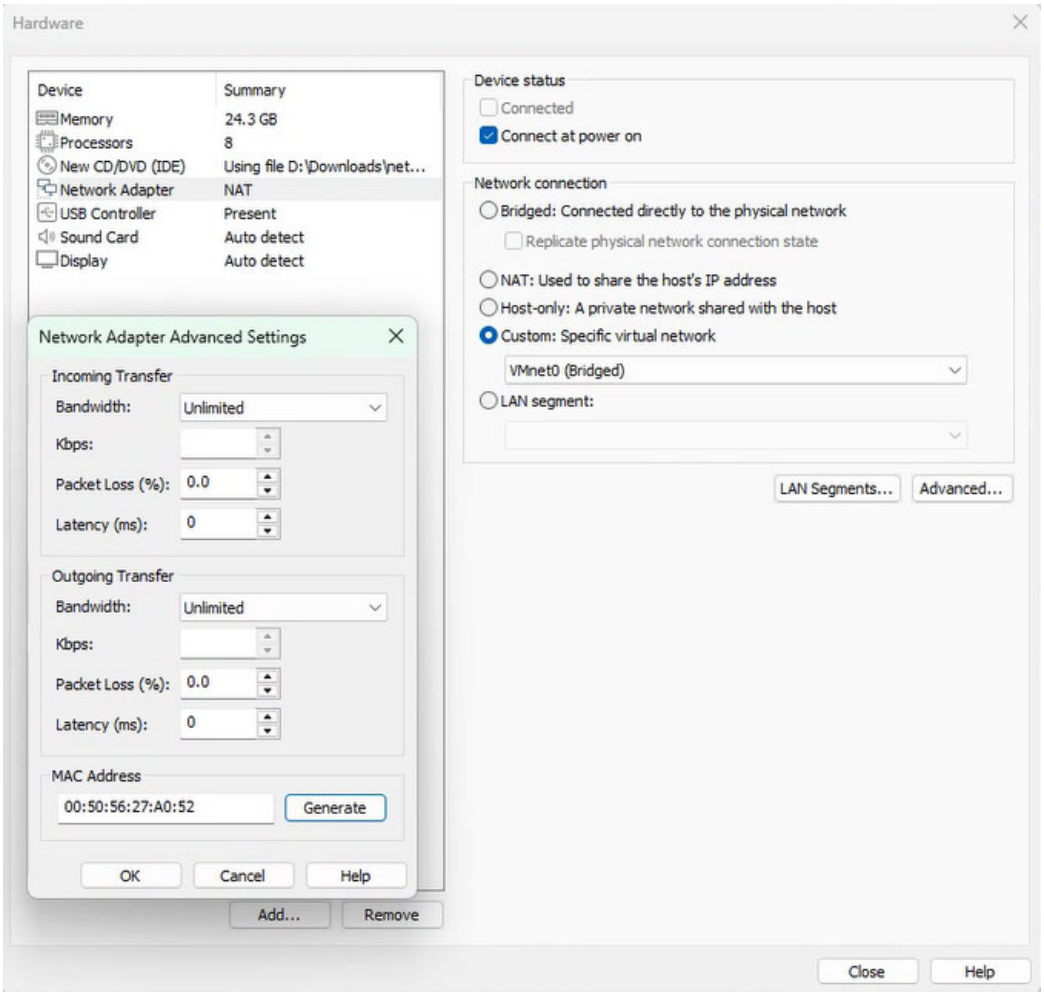


Nota. Asignación de recursos de cómputo (RAM y núcleos). Elaboración propia.

Se debe seleccionar la cantidad de adaptadores de red, en este caso, cuatro adaptadores de red. El primer adaptador de red se conecta a la red externa, el segundo adaptador de red se conecta a la red LAN, el tercer adaptador de red se conecta a la red DMZ y el cuarto adaptador de red se conecta a la red del SOC & MOC. El orden de los primeros dos adaptadores de red es importante, ya que pfSense los identifica como WAN y LAN respectivamente, los demás adaptadores de red se identifican como OPT1, OPT2 y el orden no es relevante.

El primer adaptador (WAN) debe ser el adaptador que se tenga conectado a la red externa, en este caso, *VMnet0*, ya que por defecto es el adaptador que está asociado con la tarjeta de red que tiene la computadora (que tiene acceso a Internet). Se debe seleccionar desde la opción *Custom* para elegir el adaptador específico; y en la sección avanzada se debe generar una dirección MAC aleatoria para evitar conflictos de red.

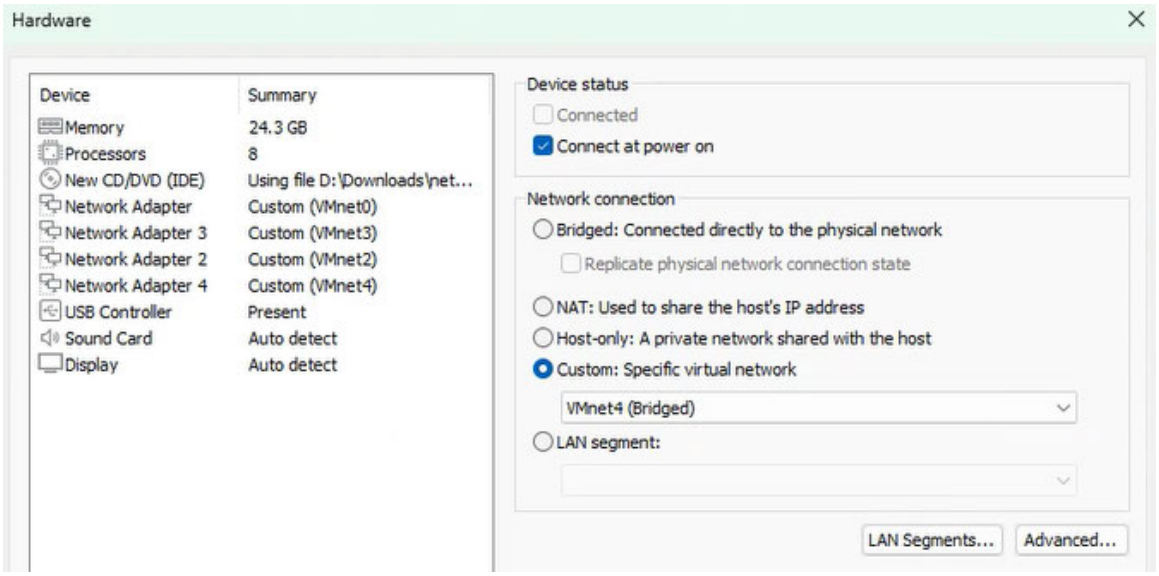
Figura 13. Configuración de primer adaptador de red WAN (VMnet0) con MAC aleatoria.



Nota. Selección del primer adaptador de red (WAN - VMnet0). Elaboración propia.

El segundo adaptador (LAN) se conecta a la red LAN, que es la red interna para usuarios finales, el tercer adaptador (OPT1) se conecta a la red DMZ y el cuarto adaptador (OPT2) se conecta a la red del SOC & MOC.

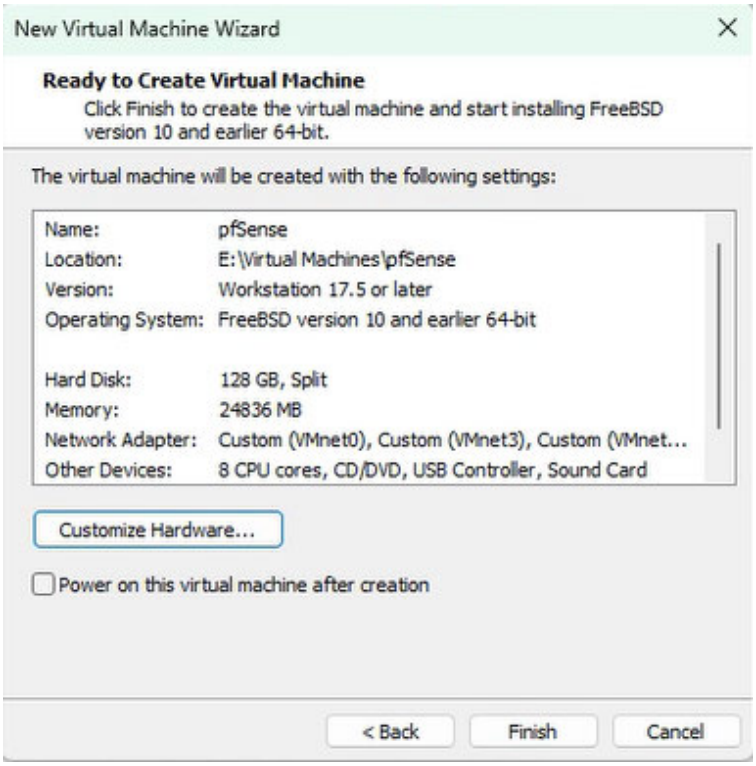
Figura 14. Validación final de cuatro adaptadores de red configurados para pfSense.



Nota. Vista de validación final de interfaces añadidas que soportan segmentación y políticas diferenciadas de seguridad. Elaboración propia.

Una vez configurada la máquina virtual, se debe iniciar la máquina virtual y seguir los pasos de configuración de pfSense.

Figura 15. Pantalla de arranque inicial de pfSense.

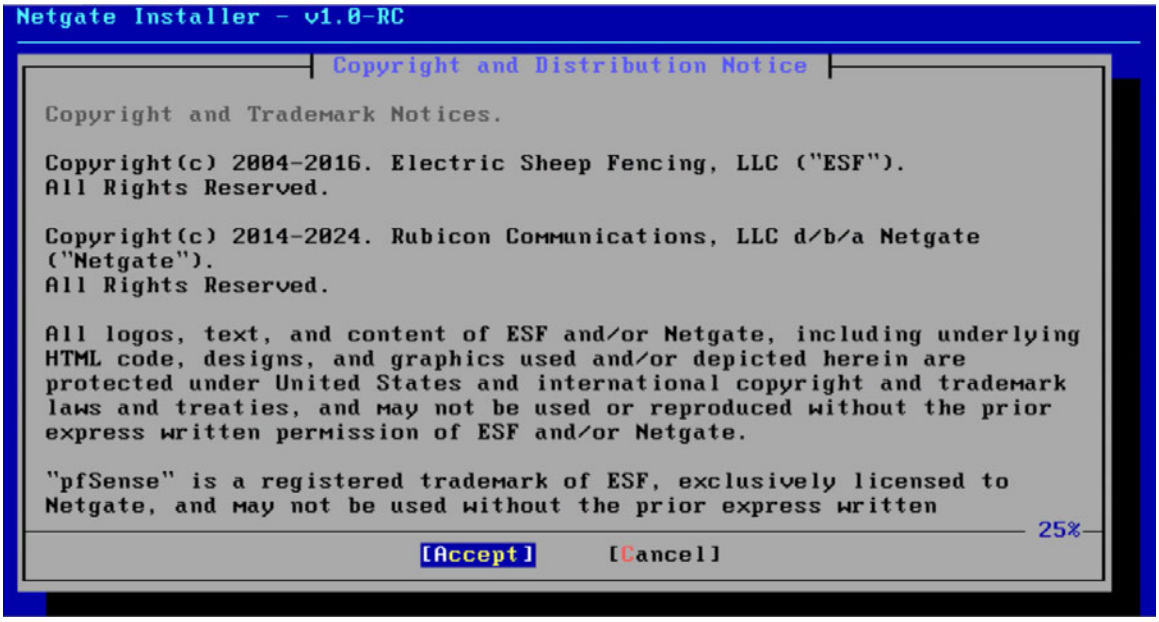


Nota. Inicio del arranque de la VM de pfSense tras completar su aprovisionamiento inicial. Elaboración propia.

7.2.3. Configuración de pfSense

Una vez iniciada la máquina virtual, se debe configurar pfSense siguiendo los siguientes pasos: aceptar los avisos de derechos de autor y marcas registradas.

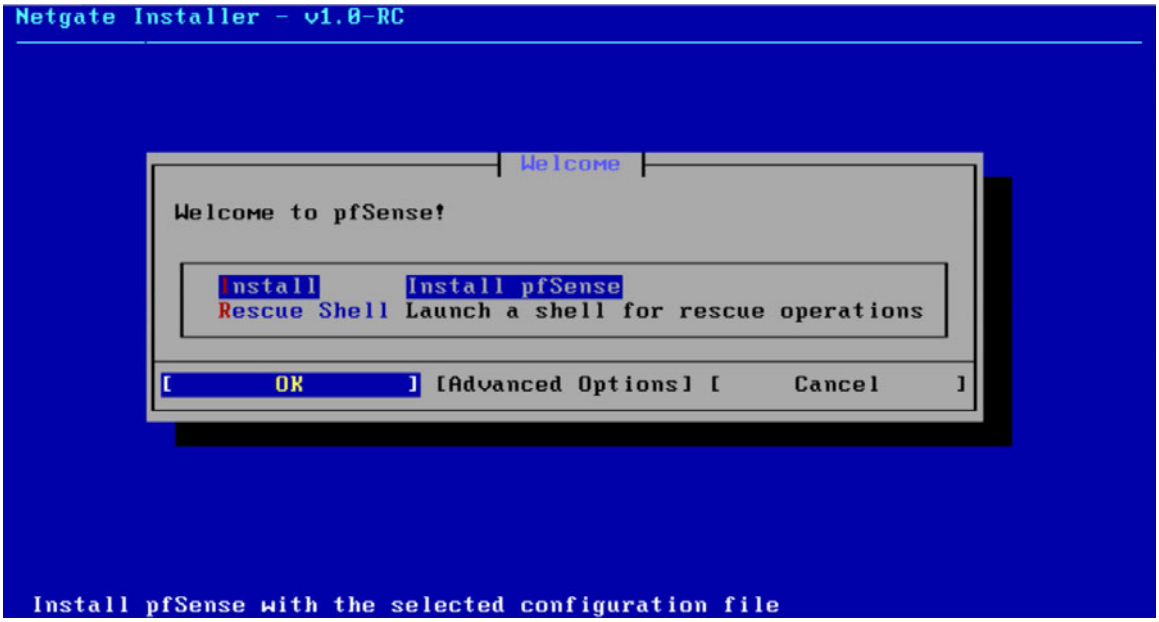
Figura 16. Pantalla de aceptación de licencia y términos de uso.



Nota. Pantalla de aceptación de licencia y términos antes de proceder con la instalación de pfSense. Elaboración propia.

Seleccionar la opción de instalar.

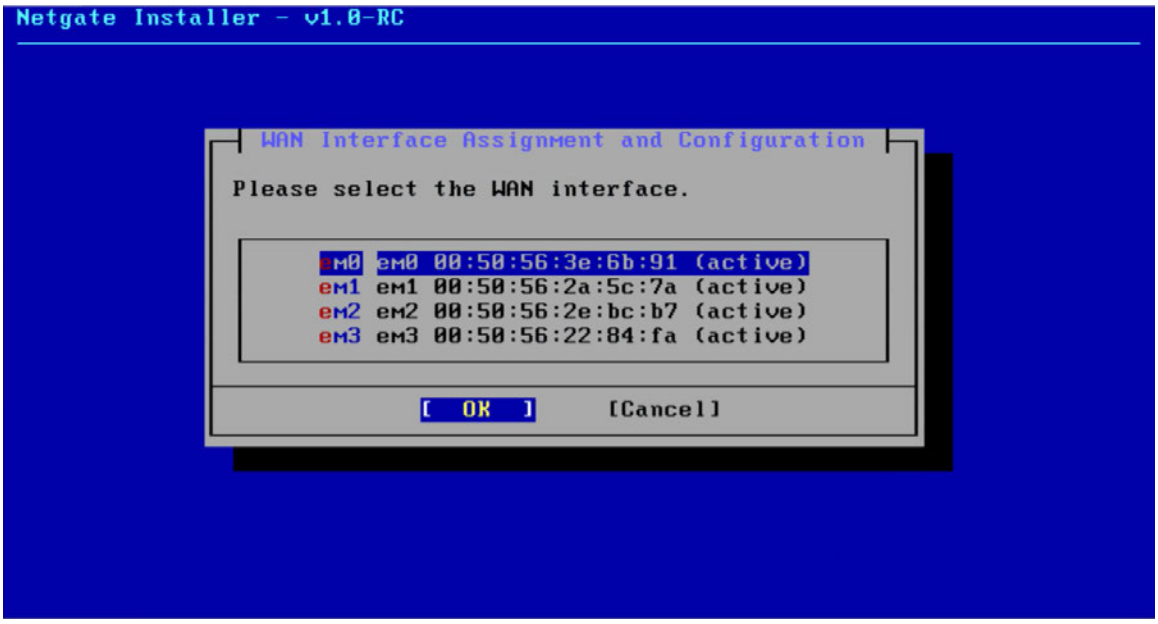
Figura 17. Selección del modo de instalación de pfSense.



Nota. Selección del modo de instalación, paso inicial para preparar el disco y continuar con la configuración asistida. Elaboración propia.

Luego, se debe proceder con la configuración de la red. Primero, se debe seleccionar la interfaz de red WAN, que es la interfaz de red externa, es decir la red de la universidad.

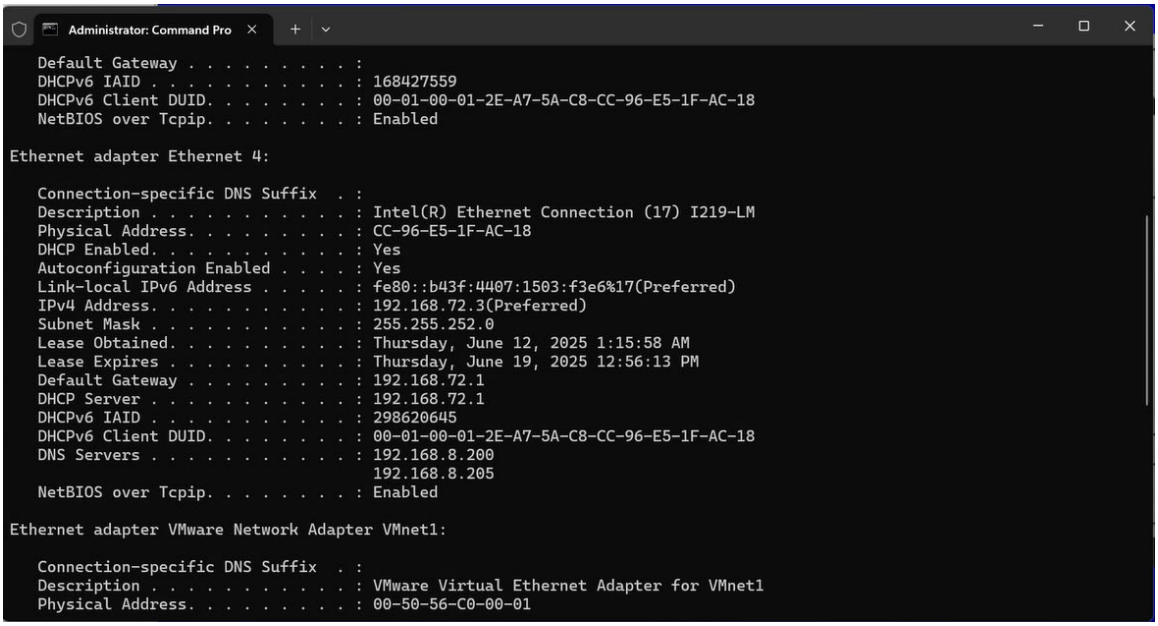
Figura 18. Selección de la interfaz WAN durante la configuración inicial.



Nota. Selección y asociación inicial de la interfaz WAN dentro del asistente de configuración para pfSense, clave para conectividad externa. Elaboración propia.

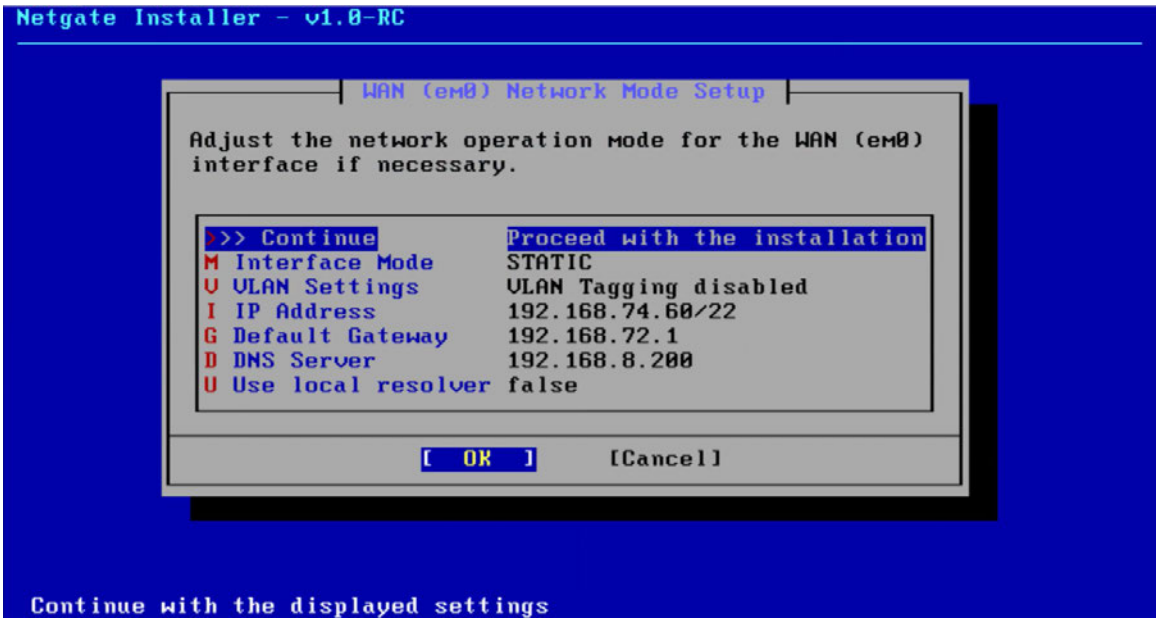
Luego, se debe configurar la interfaz. En el modo se selecciona la opción *Static* y se debe ingresar la dirección IP de la interfaz de red WAN, que es la dirección IP de la red de la universidad. La dirección IP 192.168.74.60/22, ya que la red de la universidad está en el rango de 192.168.72.0/22. El VLAN *Tagging* se deja deshabilitado. En la sección *Default Gateway* se debe ingresar la dirección IP del *gateway* de la red de la universidad, que se puede obtener abriendo una terminal en Windows y ejecutando el comando `ipconfig` como se observa en la Figura 19. La dirección IP del *gateway* es 192.168.72.1. En la sección DNS *Servers* se debe ingresar la dirección IP del servidor DNS de la red de la universidad, que de igual manera se puede obtener del mismo comando. En este caso, la dirección IP del servidor DNS es 192.168.8.200.

Figura 19. Asignación de dirección IP estática para la interfaz WAN.



Nota. Asignación de parámetros IP estáticos para la WAN (dirección, *gateway* y DNS) garantizando resolución y ruteo estable. Elaboración propia.

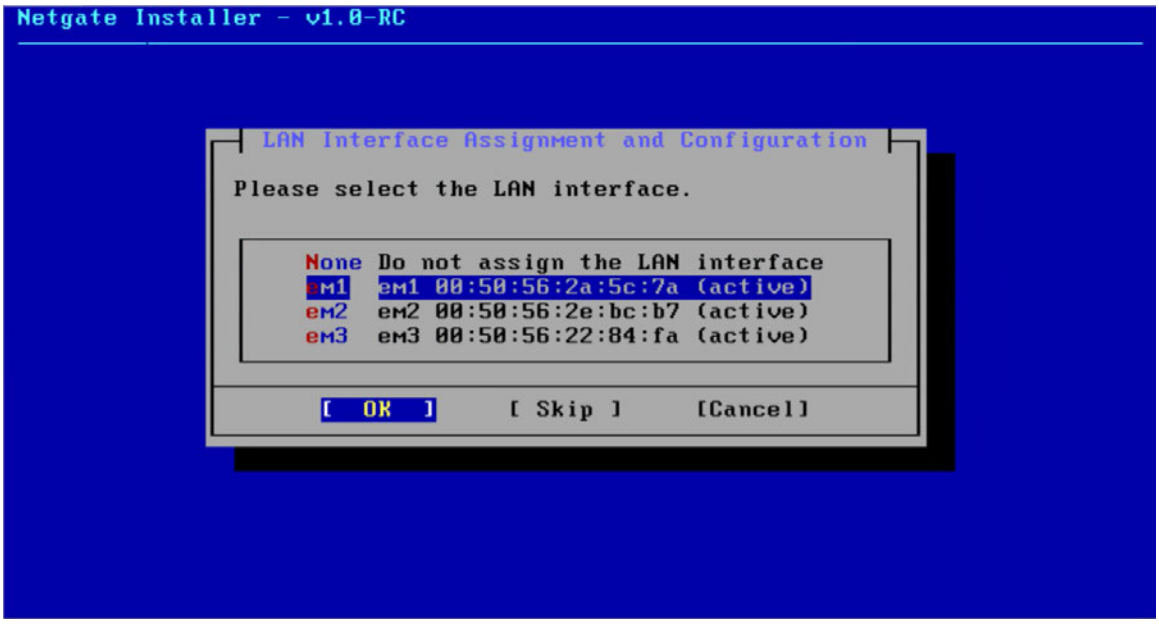
Figura 20. Confirmación de parámetros de red para la interfaz WAN.



Nota. Confirmación de parámetros WAN ingresados antes de aplicar la configuración definitiva en pfSense. Elaboración propia.

Luego, se debe configurar la interfaz de red LAN, que es la interfaz de red interna para usuarios finales. Se selecciona la segunda interfaz de red, que es la interfaz de red LAN.

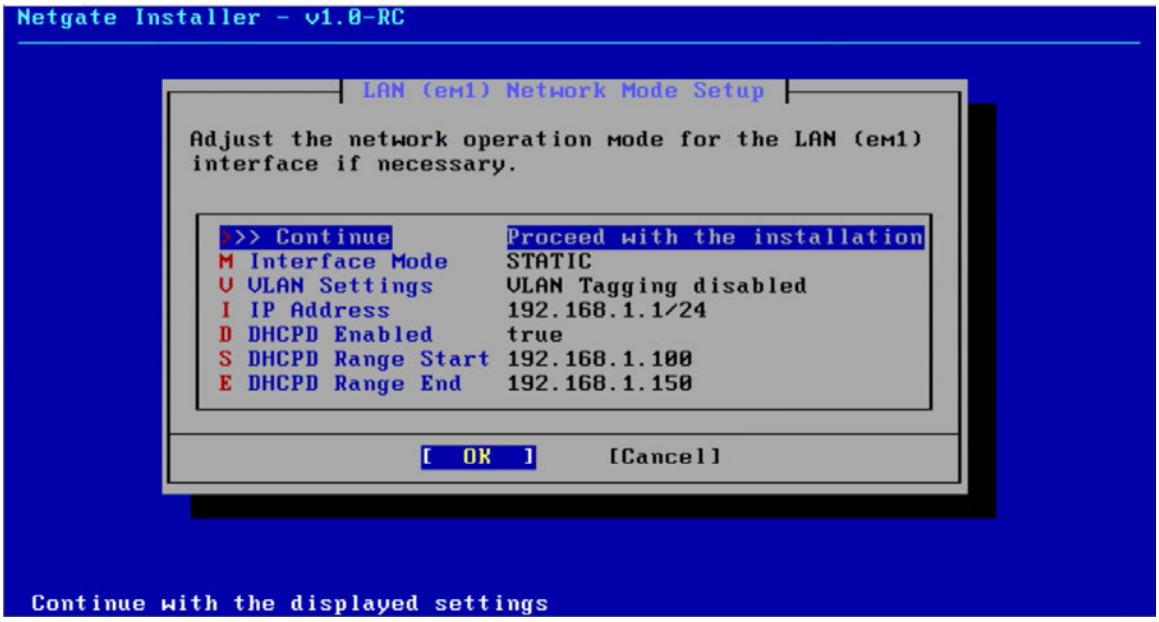
Figura 21. Selección de la interfaz LAN para su configuración.



Nota. Selección de la interfaz LAN interna que provee direccionamiento y servicios (DHCP) a usuarios finales. Elaboración propia.

Luego, se debe configurar la interfaz. En el modo se selecciona la opción *Static* y se debe ingresar la dirección IP de la interfaz de red LAN, que es la dirección IP del *gateway* para los usuarios finales. En este caso, la dirección IP 192.168.1.1/24, que es una dirección IP comúnmente utilizada para redes internas. El VLAN *Tagging* se deja deshabilitado. También es importante habilitar el servidor DHCP para que los usuarios finales puedan obtener una dirección IP automáticamente. Para ello, se debe seleccionar la opción *DHCPD* y se debe ingresar el rango de direcciones IP que se asignan a los usuarios finales. En este caso, el rango de direcciones IP 192.168.1.100 a 192.168.1.150.

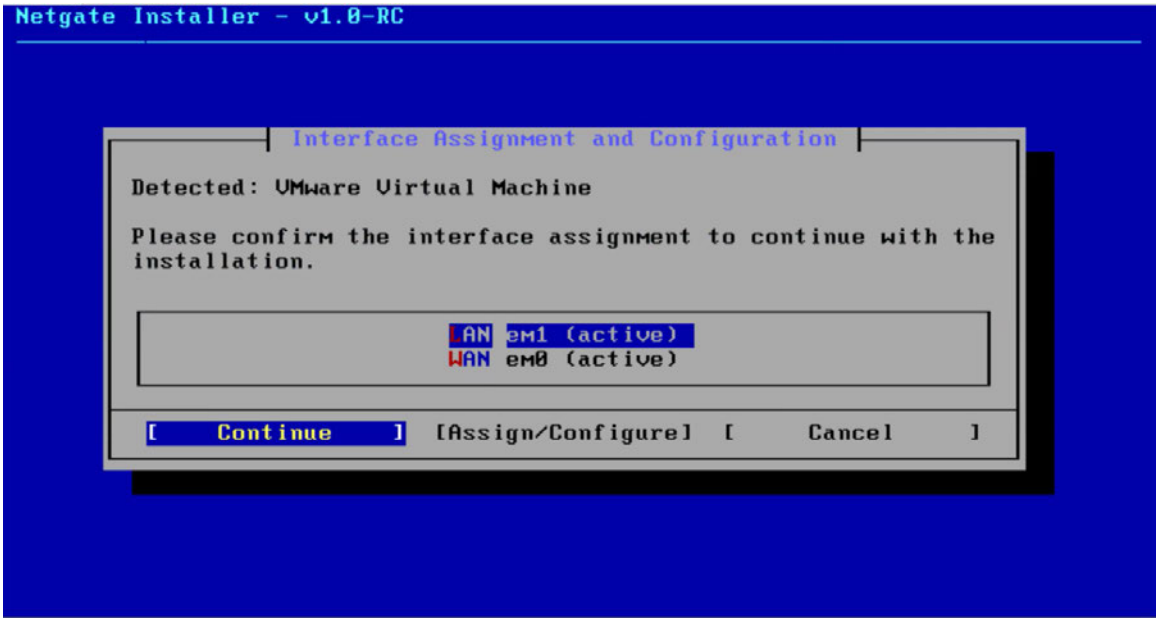
Figura 22. Configuración de la interfaz LAN.



Nota. Configuración de la interfaz LAN (dirección, *gateway* y DNS) garantizando resolución y ruteo estable. Elaboración propia.

Se confirma la configuración de ambas interfaces de red (WAN y LAN) para continuar con la instalación.

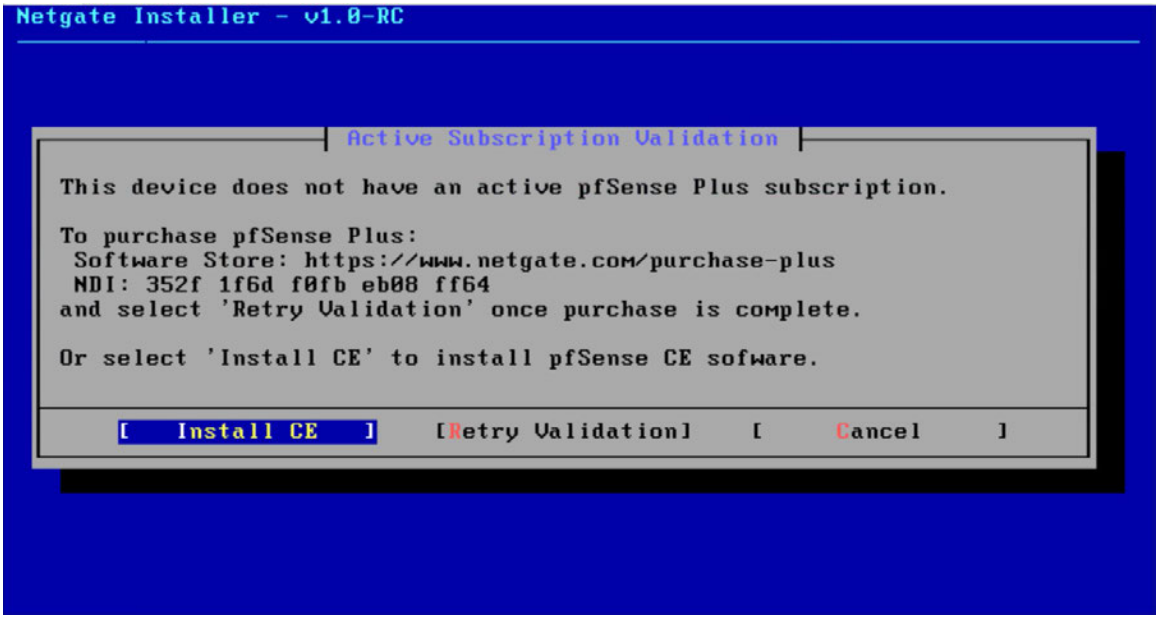
Figura 23. Confirmación de configuración de las interfaces WAN y LAN.



Nota. Confirmación de configuración de las interfaces WAN y LAN antes de continuar con la instalación. Elaboración propia.

Se verifica que la configuración de la red WAN tenga acceso a Internet, intentando acceder a los servidores de pfSense. Si se tiene acceso el mensaje indica que no se tiene una suscripción activa de pfSense Plus, pero esto es normal ya que se instala la versión *Community Edition*.

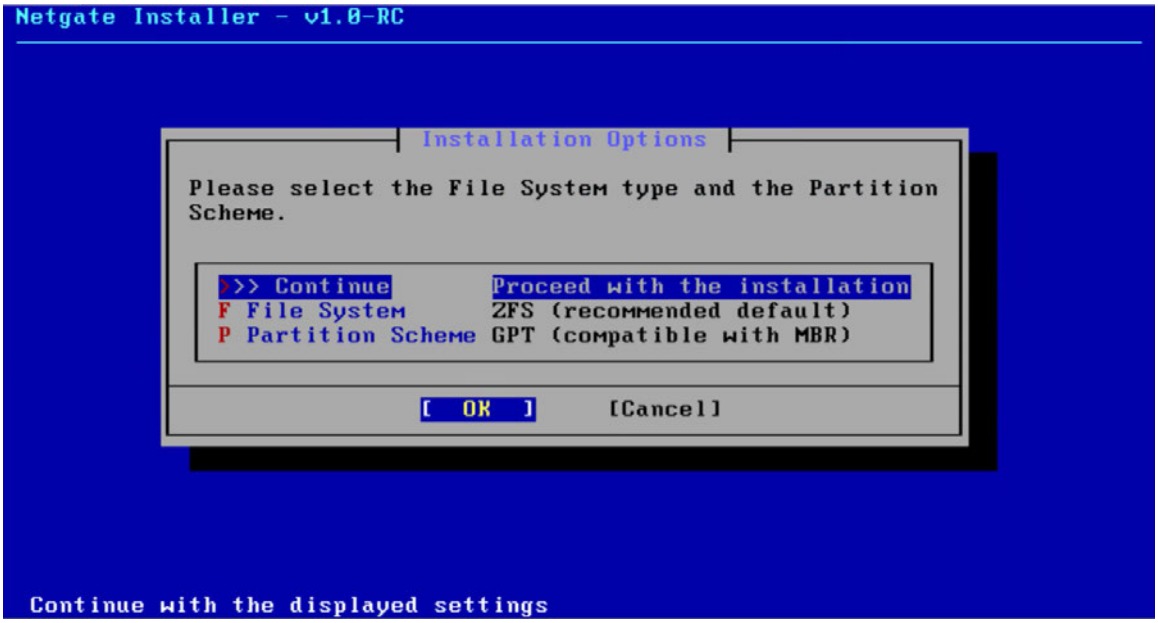
Figura 24. Verificación de conectividad con servidores de pfSense y validación de suscripción.



Nota. Verificación de conectividad externa (acceso a servidores de pfSense) asegurando que la interfaz WAN funciona correctamente y que no se requiere una suscripción activa de pfSense Plus. Elaboración propia.

Luego se debe seleccionar el tipo de sistema de archivos y el esquema de particiones. En este caso, la opción ZFS y el esquema de particiones GPT.

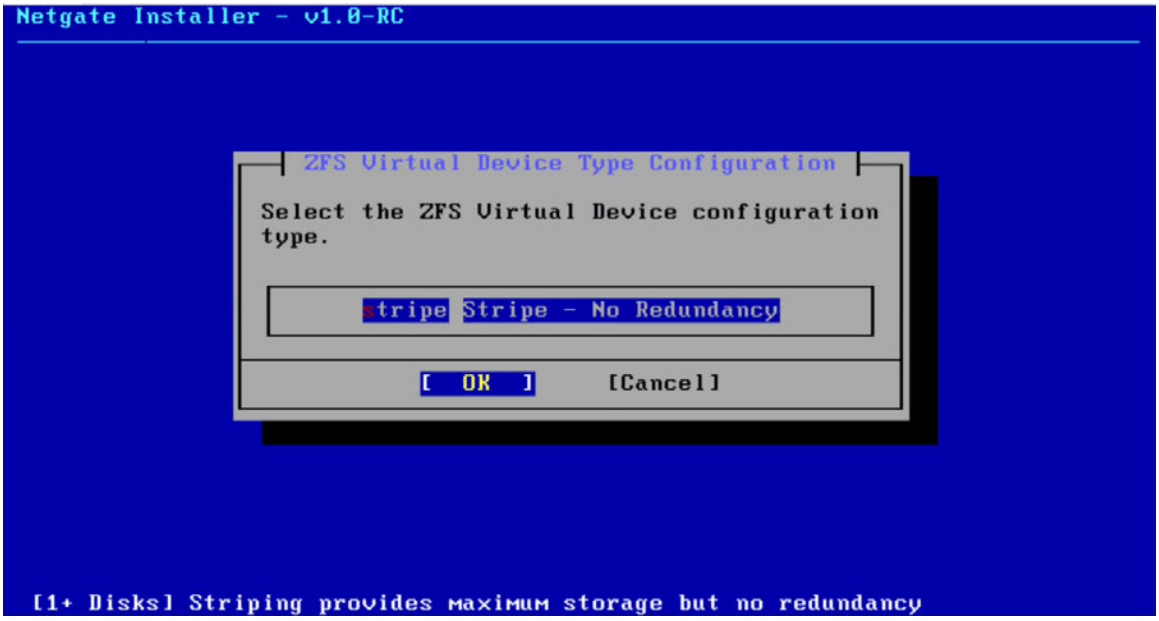
Figura 25. Selección de sistema de archivos ZFS y esquema GPT.



Nota. Selección de esquema de particiones y tipo de sistema de archivos (ZFS/GPT) para asegurar resiliencia y gestión avanzada de almacenamiento. Elaboración propia.

Se selecciona el tipo de configuración de ZFS, en este caso, la opción *Stripe - no redundancy*. Esto se debe a que se está utilizando un solo disco duro virtual para la instalación de pfSense.

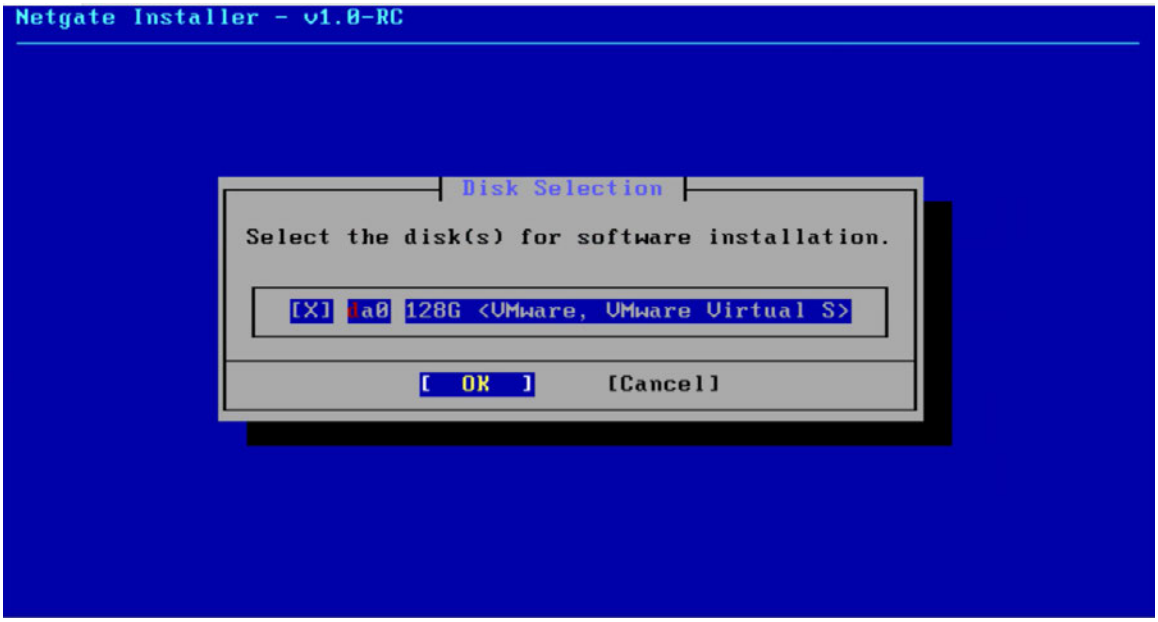
Figura 26. Confirmación de opciones ZFS antes de particionado.



Nota. Confirmación de opciones ZFS previas antes de proceder con el particionado y escritura en disco. Elaboración propia.

Se selecciona el disco duro donde se instala pfSense, en este caso, el disco duro virtual que se creó para la máquina virtual con capacidad de 128 GB.

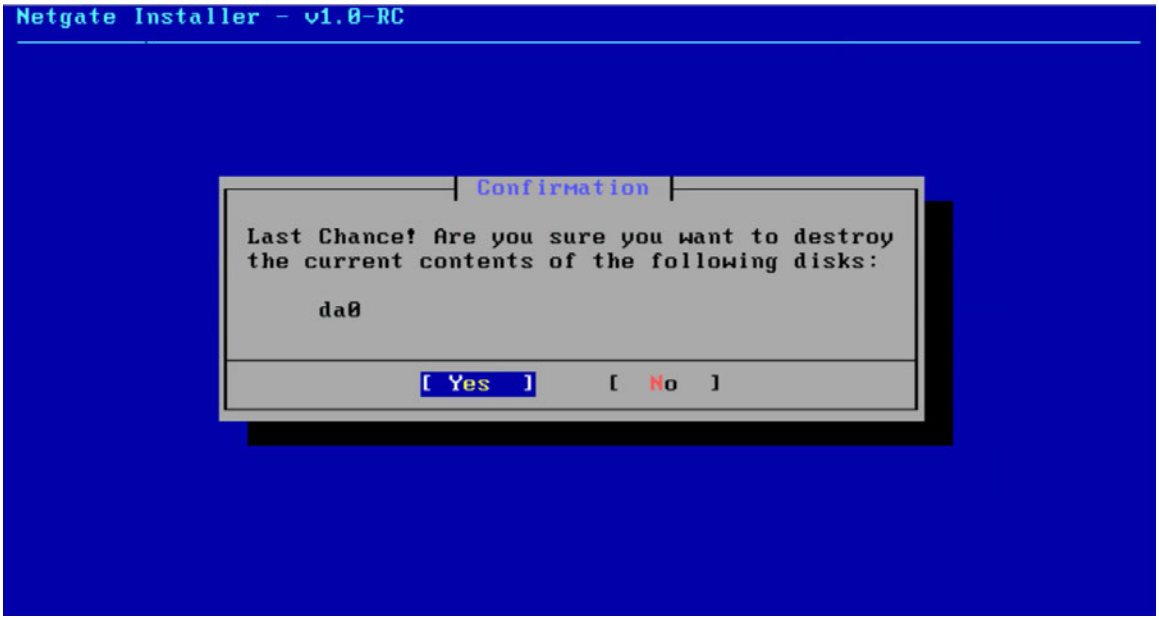
Figura 27. Selección del disco duro virtual para instalación.



Nota. Selección del disco virtual destinado a la instalación de pfSense, asegurando que el sistema se despliegue en el almacenamiento correcto. Elaboración propia.

Se muestra una advertencia indicando que se borran todos los datos del disco duro seleccionado, se debe confirmar que se desea continuar con la instalación.

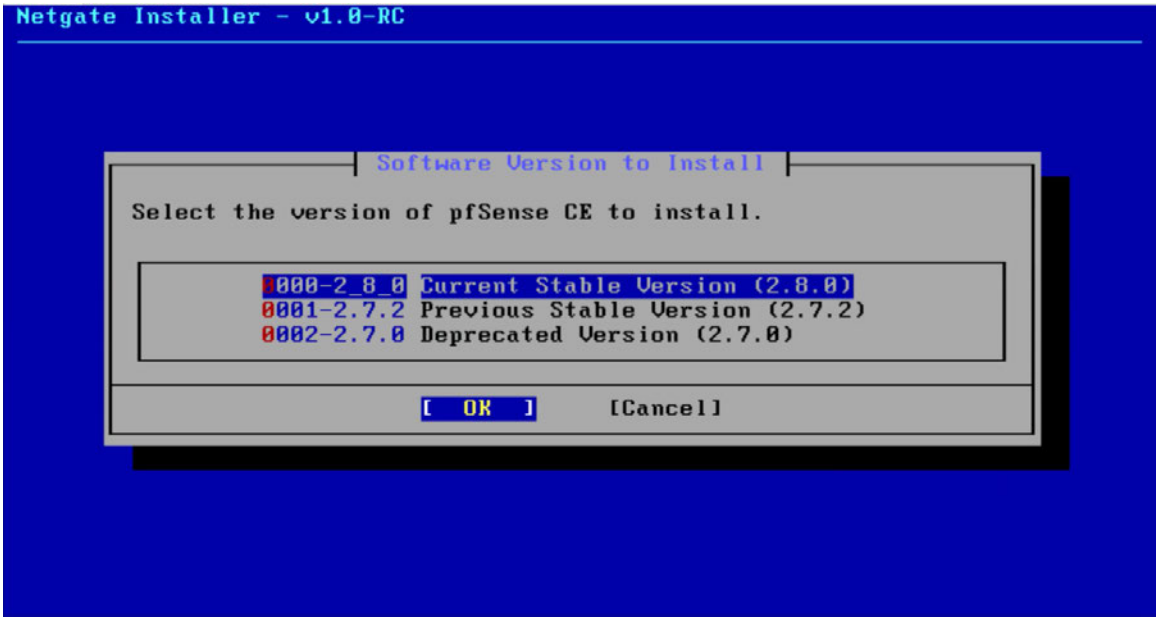
Figura 28. Advertencia de borrado de datos.



Nota. Elaboración propia.

Por último, se debe seleccionar la versión de pfSense que se desea instalar. En este caso, la versión estable más reciente (2.8.0.) Se procede con la instalación y se espera a que se complete.

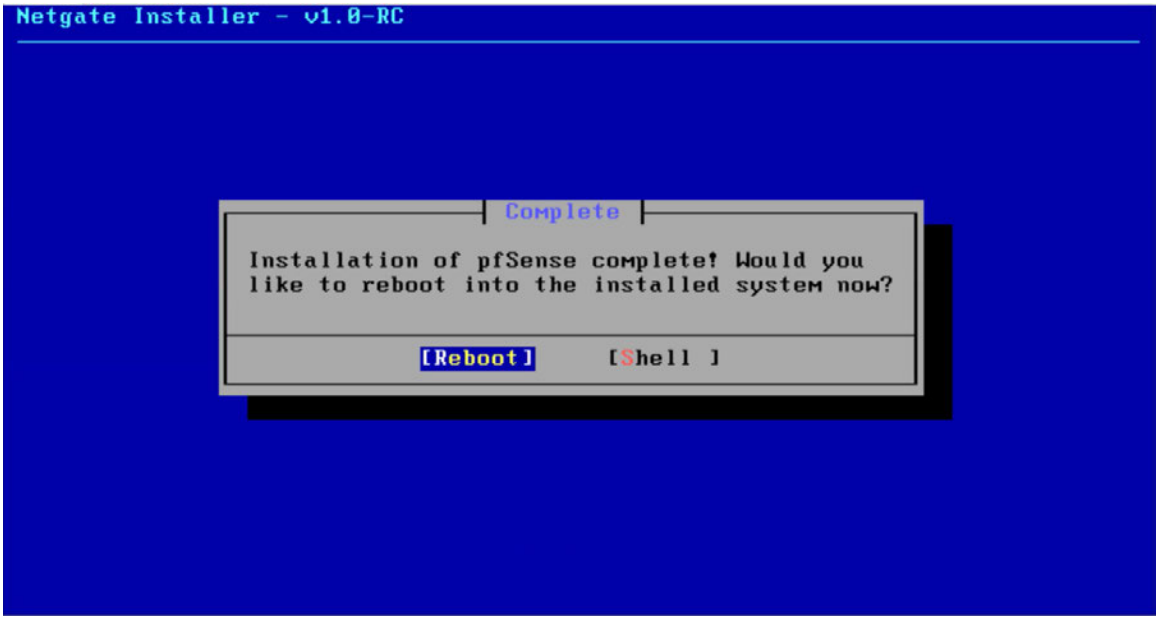
Figura 29. Selección de versión estable de pfSense (2.8.0).



Nota. Pantalla de selección de versión estable (2.8.0) asegurando entorno soportado y actualizado para utilizar Snort. Elaboración propia.

Al terminar la instalación, se muestra una pantalla indicando que la instalación se ha completado correctamente y se debe reiniciar la máquina virtual.

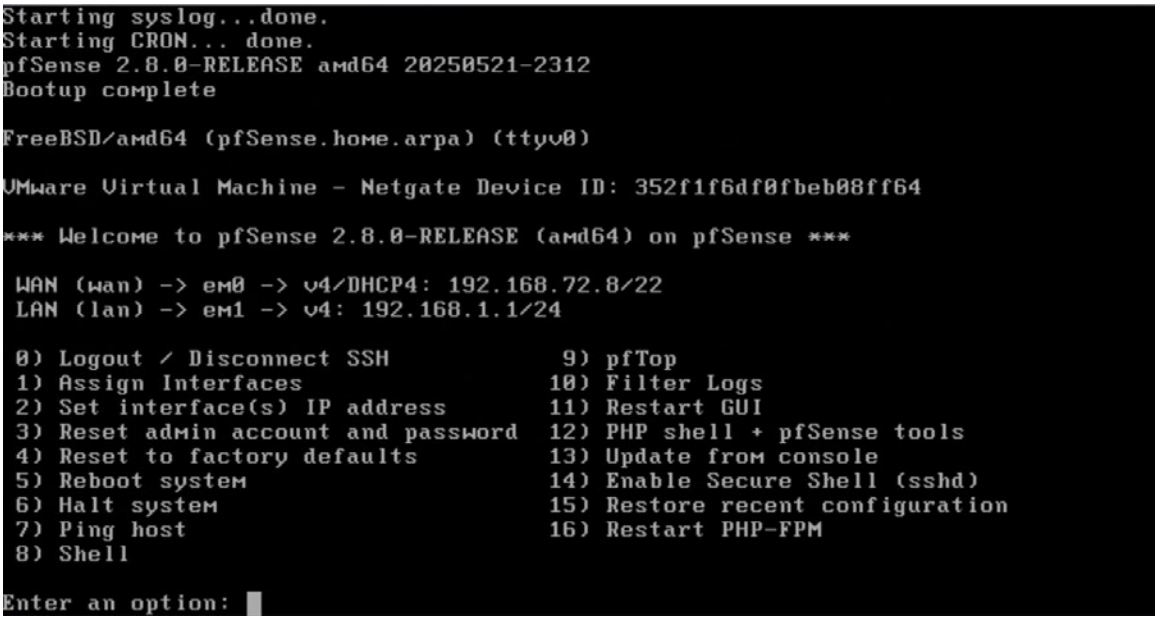
Figura 30. Mensaje de finalización de instalación.



Nota. Pantalla que indica la finalización exitosa de la instalación de pfSense. Elaboración propia.

Una vez reiniciada la máquina virtual, se tiene acceso a la consola de pfSense, donde se pueden ver las diferentes opciones de configuración.

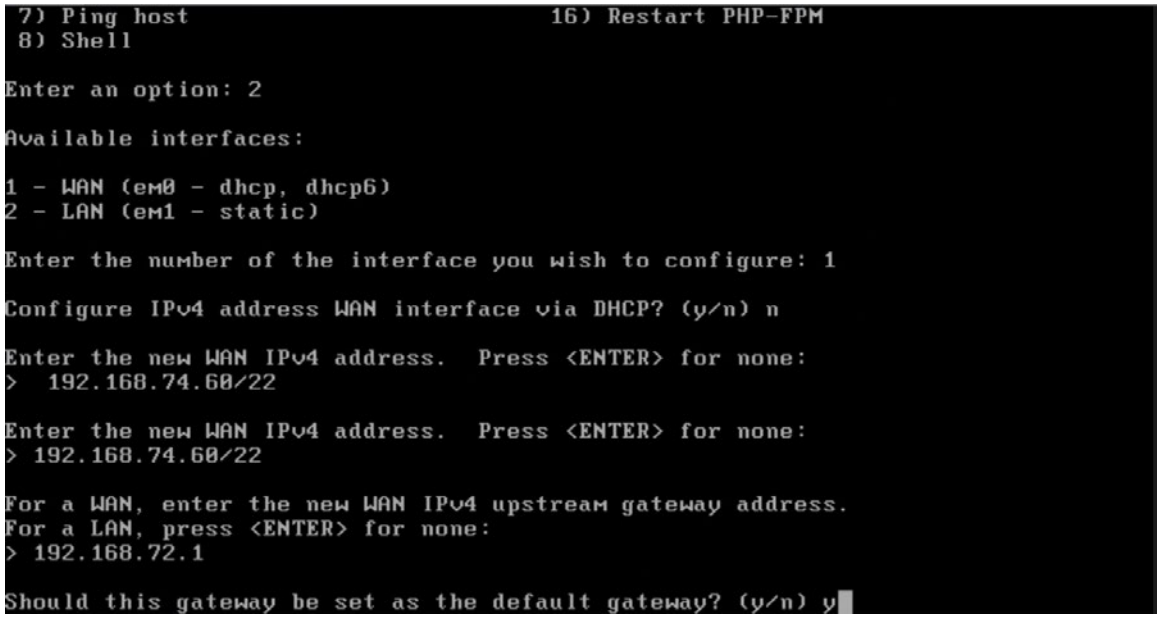
Figura 31. Menú principal de pfSense.



Nota. Menú principal tras instalación exitosa, punto de partida para ajustes avanzados de interfaces y servicios. Elaboración propia.

Se puede observar en la Figura 31 que la interfaz de red WAN, a pesar que en la configuración inicial se seleccionó la opción del modo *Static*, se ha configurado como DHCP y se le ha asignado una dirección IP. Por lo que se debe cambiar a modo *Static* y asignar una dirección IP estática. Se selecciona la opción 2, que es la opción *Set interface(s) IP address*. Se selecciona la interfaz de red WAN, que es la interfaz de red externa, es decir, 1. Cuando se pregunta si se desea configurar por medio de DHCP, se selecciona la opción *n* para no configurar por medio de DHCP. Luego, se debe ingresar la dirección IP con la máscara de red, en este caso, 192.168.74.60/22. Luego, se debe ingresar la dirección IP del *gateway* de la red de la universidad, que es 192.168.72.1. Nos pregunta si se quiere que sea el *gateway* por defecto, se selecciona que sí.

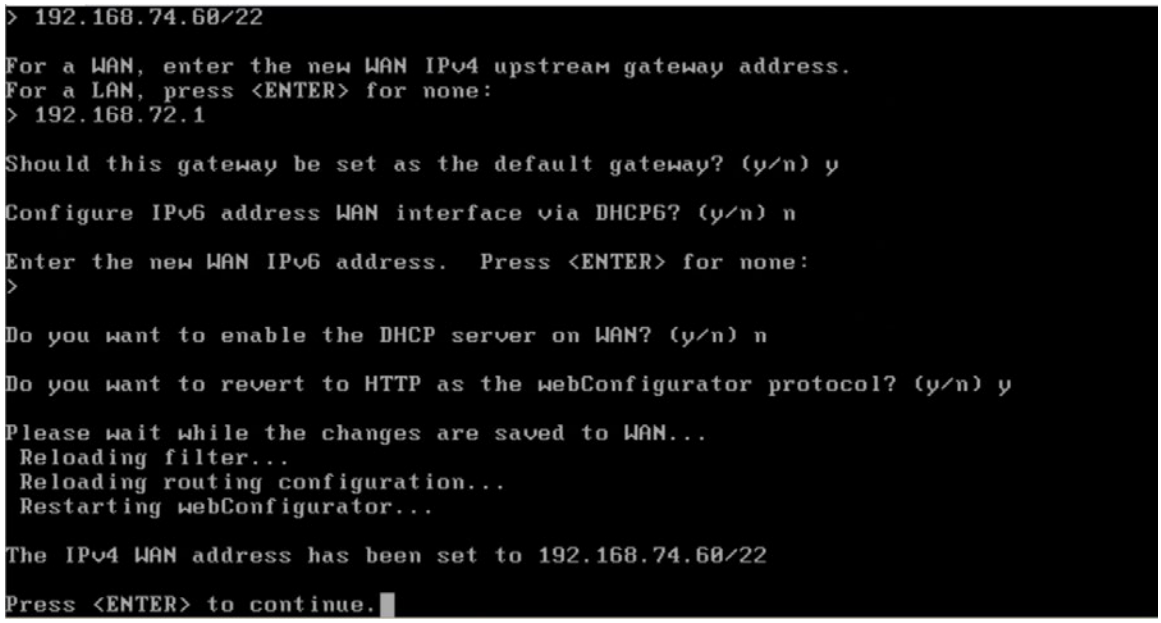
Figura 32. Cambio de interfaz WAN a modo estático vía consola.



Nota. Cambio manual de la interfaz WAN a modo estático para asegurar consistencia de direccionamiento y reglas. Elaboración propia.

En las configuraciones IPv6 se deja en blanco todo ya que en este proyecto no se utiliza el protocolo IPv6. Cuando se pregunta si se desea configurar como servidor DHCP, se selecciona la opción *n* para no configurar como servidor DHCP, ya que configurar un servidor DHCP en la red externa no es necesario y puede causar problemas de conectividad. Por último, se debe colocar HTTP como protocolo de acceso a la interfaz web de pfSense.

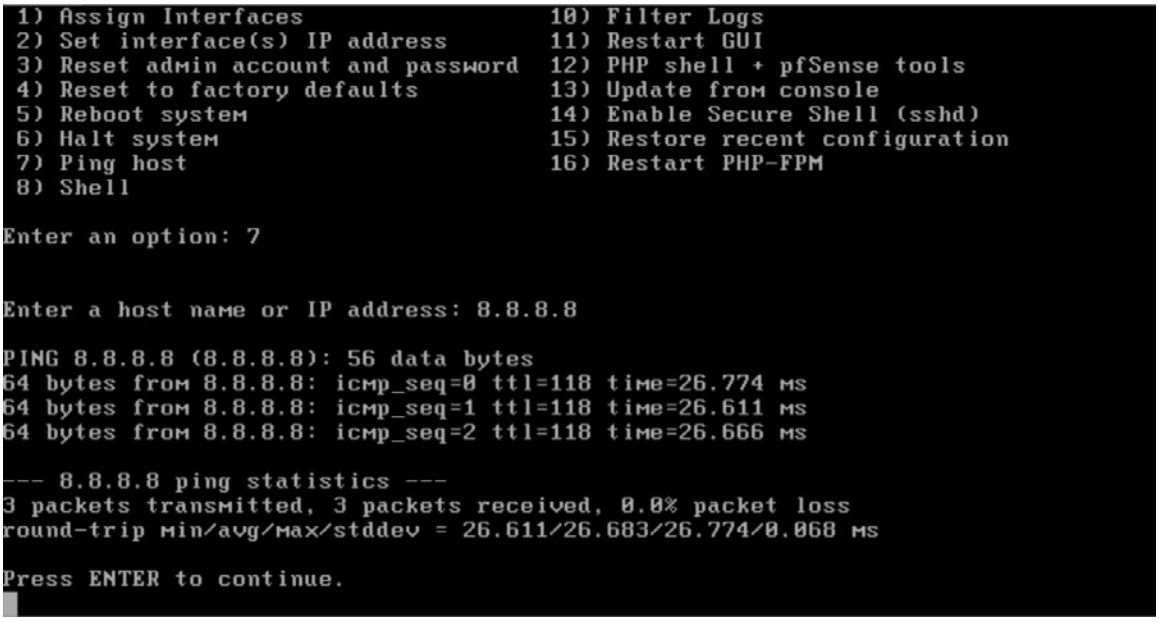
Figura 33. Configuración IPv6, DHCP y protocolo de acceso web.



Nota. Definición de parámetros adicionales para finalizar ajustes WAN.
Elaboración propia.

Para comprobar que la configuración de la red WAN se ha realizado correctamente, se selecciona la opción 7 del menú principal, que es la opción *Ping host*. Se ingresa una IP de un servidor externo, por ejemplo el servidor DNS de Google, que es 8.8.8.8. Si se recibe una respuesta, significa que la configuración de la red WAN se ha realizado correctamente y se tiene acceso a Internet.

Figura 34. Prueba de conectividad a Internet mediante ping.



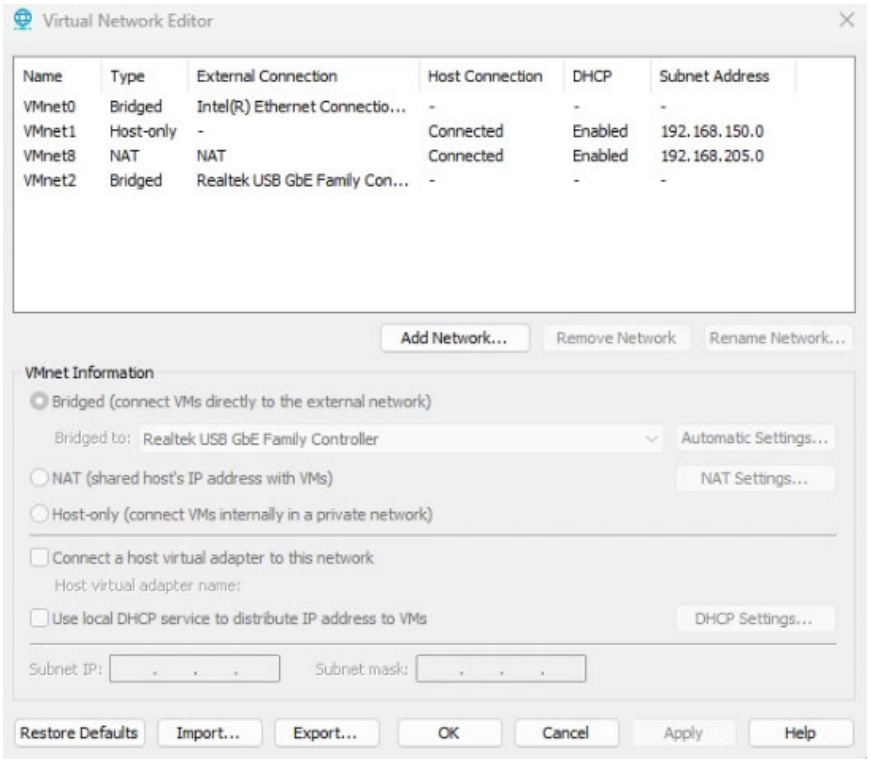
Nota. Prueba de conectividad mediante ping a servidor público (8.8.8.8) validando salida a Internet. Elaboración propia.

7.2.4. Configuración de las interfaces de red vía interfaz web

El siguiente paso sería configurar las otras interfaces de red, ya que actualmente solo se ha configurado la interfaz de red WAN y LAN. Esto se podría realizar desde la consola como se realizó anteriormente, sin embargo al tener configurada la interfaz de red LAN, ahora podemos tener un usuario conectado a esta red y acceder a la interfaz web de pfSense para configurar las otras interfaces de una manera más sencilla.

Para ello se debe acceder a la PC2 donde se encuentra la red LAN como se observa en la Figura 2. Como se mencionó con anterioridad, se necesita tener VMware Workstation Pro 17 y, tener conectado el adaptador USB-C a Ethernet conectado a la otra PC y confirmar que Windows lo reconozca. Luego abrir el *virtual network editor* en la pestaña *Edit*. Presionar el botón de *Add Network* y seleccionar la opción *VMnet2*, que es la red LAN que se configuró en el *virtual network editor* de la PC1.

Figura 35. Preparación de la red LAN (VMnet2) en el *virtual network editor* para conexión de usuarios.



Nota. Vista del *virtual network editor* al preparar la red LAN (VMnet2) donde se conectan los usuarios internos antes de continuar con la configuración vía interfaz web de pfSense. Elaboración propia.

Se necesita crear una nueva máquina virtual con los siguientes requerimientos:

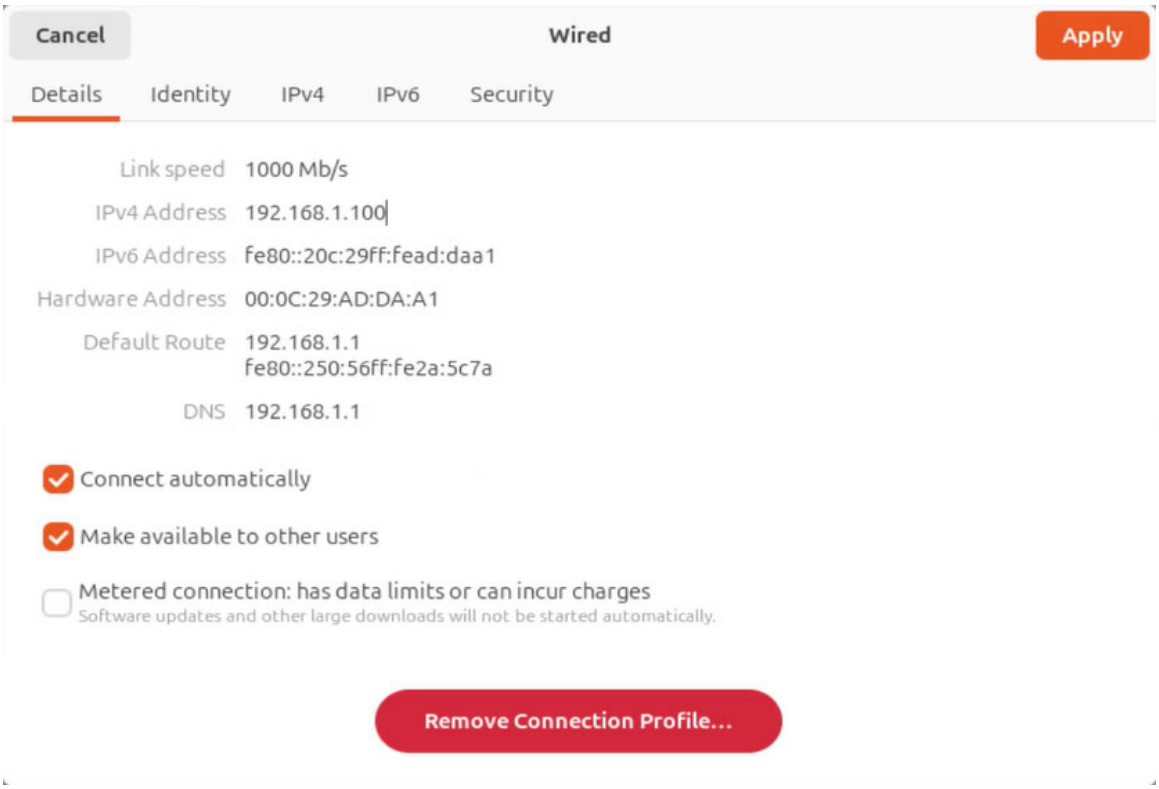
- Sistema operativo: Ubuntu *Desktop* 24.04.2
- Memoria: 4GB
- Espacio de disco duro: 60 GB
- Procesadores/Núcleos por procesador: 1/2
- Adaptador de red: *custom*, VMnet2 (que es la red LAN que se configuró en el *virtual network editor*). Se debe seleccionar la opción*Generate a new MAC address* para evitar conflictos de red.

Una vez creada la máquina virtual, se debe iniciar la máquina virtual y seguir los pasos de instalación de Ubuntu *Desktop* 24.04.2. Lo importante es en el paso de conexión a Internet, seleccionar la opción *Use wired connection.*

Para comprobar que nuestra máquina virtual está conectada a la red LAN de nuestro *firewall* virtual, se puede abrir las configuraciones de red y debería tener una dirección IP

en el rango que configuramos en el servidor DHCP de pfSense. Además, se puede abrir el navegador web y se debería tener acceso a Internet.

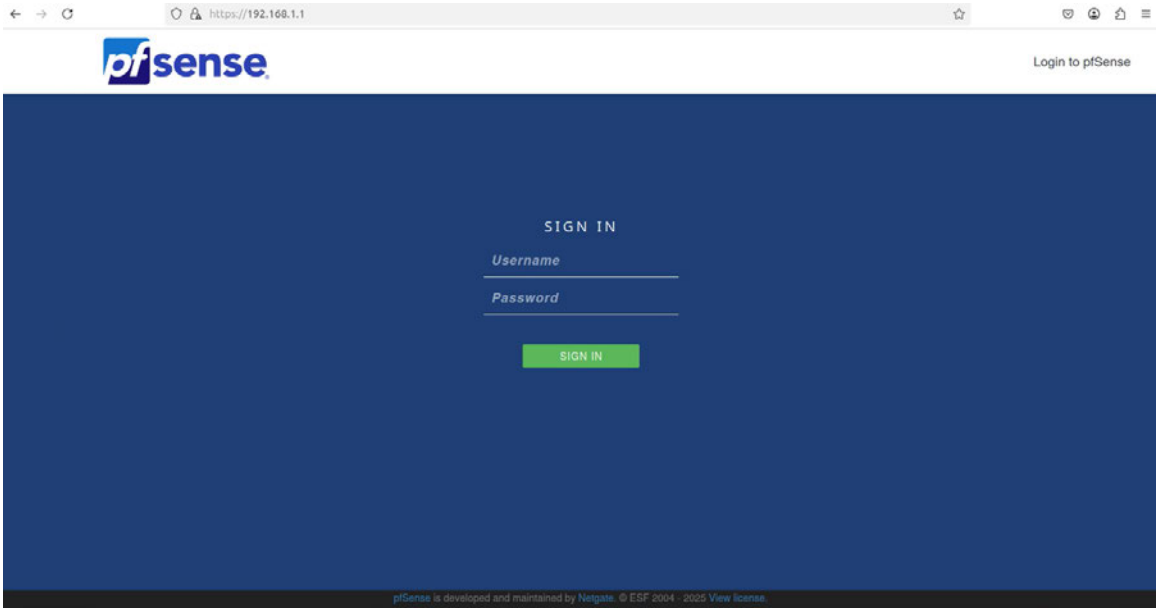
Figura 36. Configuración de red en Ubuntu *Desktop*.



Nota. Confirmación desde el cliente Ubuntu (PC en LAN) de que obtiene parámetros IP vía DHCP del pfSense y conectividad previa a gestión web. Elaboración propia.

Ahora se puede acceder a la interfaz web de pfSense desde el navegador web, ingresando la dirección IP de la interfaz de red LAN, que es el del *gateway* que es 192.168.1.1. Se debe ingresar el usuario y la contraseña, que por defecto son *admin* y *pfSense* respectivamente.

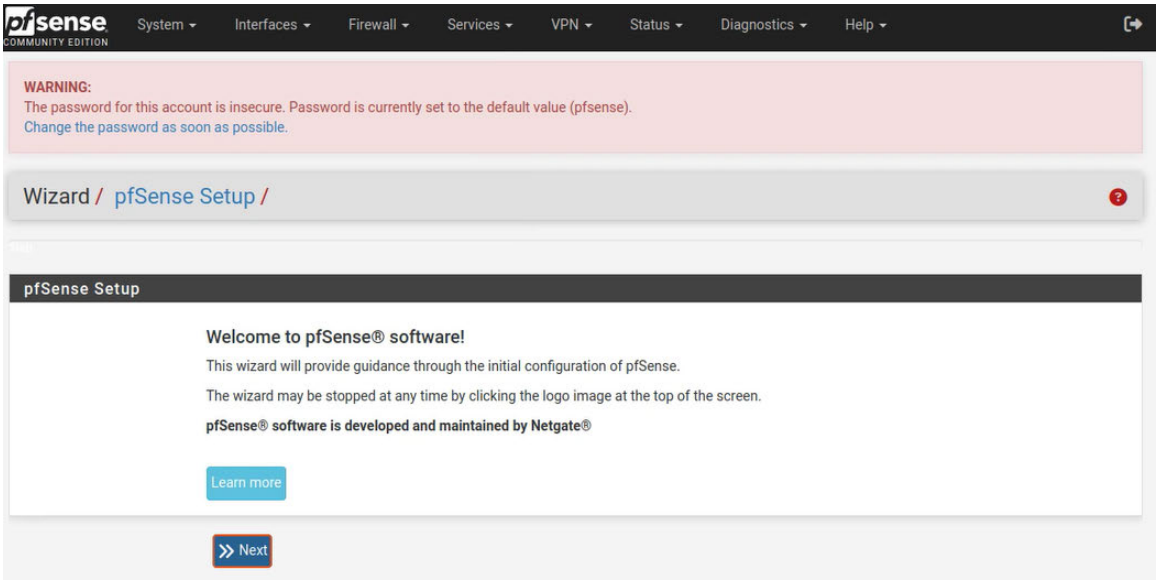
Figura 37. Pantalla de inicio de sesión de la interfaz web de pfSense.



Nota. Primera pantalla de acceso a la interfaz web de pfSense. Elaboración propia.

Al ingresar a la interfaz web de pfSense, se muestra un asistente inicial para configurar algunos parámetros básicos. En la primera pantalla se muestra un mensaje de bienvenida y se debe presionar el botón *Next* para continuar.

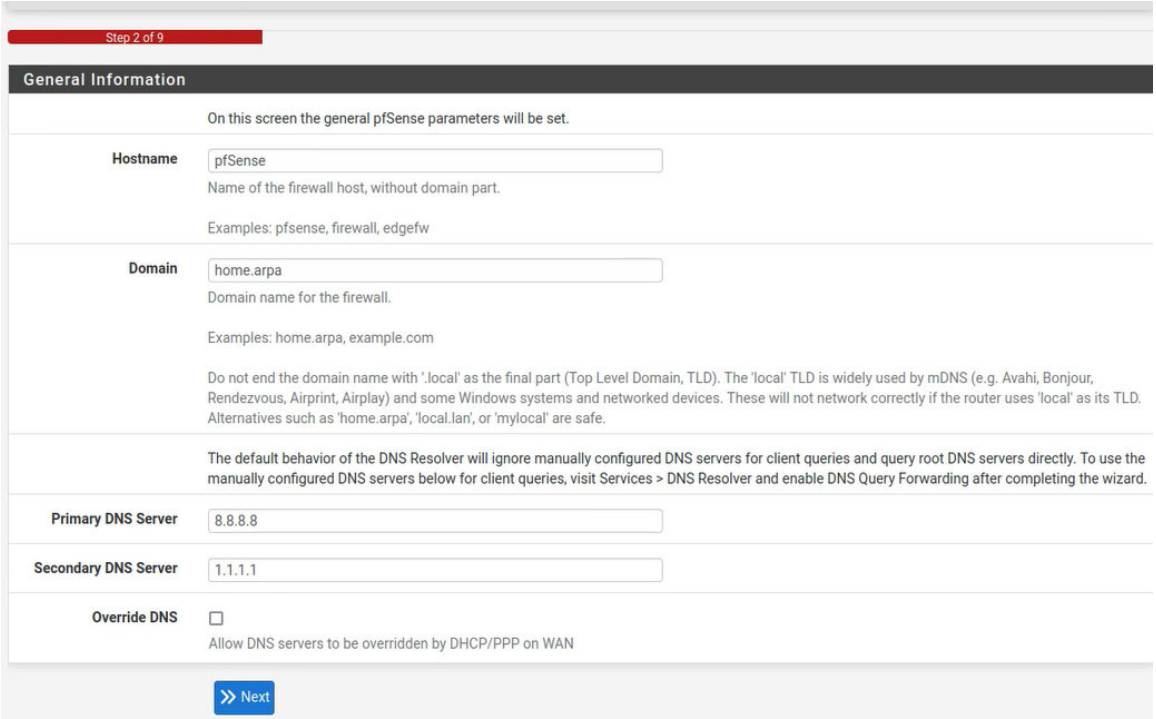
Figura 38. Asistente inicial de configuración de pfSense.



Nota. Pantalla del asistente inicial de pfSense. Elaboración propia.

Una vez dentro de la interfaz web de pfSense, se puede proceder a realizar las configuraciones iniciales. Se pregunta por un *hostname*, que es el nombre del dispositivo en la red, se puede dejar el nombre por defecto, pfSense. Luego se debe ingresar un dominio, que es el dominio al que pertenece el dispositivo, del mismo se puede dejar el por defecto que *home.arpa*. Esto hace que también se pueda acceder al *firewall* por medio del nombre de dominio, en este caso, *pfsense.home.arpa*. En los campos *Primary DNS Server* y *Secondary DNS Server* se puede ingresar la dirección IP del servidor DNS que utiliza la universidad, que es 192.168.8.200 y 192.168.8.205, o utilizar los servidores DNS de Google, que son 8.8.8.8 y 8.8.4.4. Lo importante es deseleccionar la opción *Override DNS* que es la opción que permite que pfSense utilice los servidores DNS de la red externa que recibe por medio de DHCP.

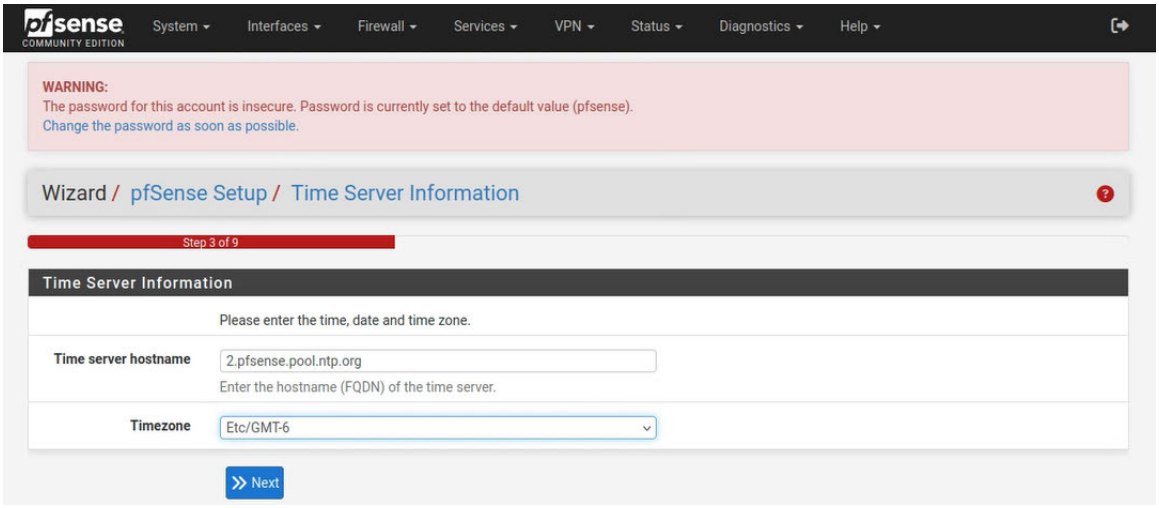
Figura 39. Configuración inicial de la interfaz web de pfSense.



Nota. Formulario del asistente donde se definen *hostname*, dominio y servidores DNS asegurando resolución consistente en la red interna. Elaboración propia.

En la siguiente sección se configura el *time Server*, que es el servidor que se utiliza para sincronizar la hora del sistema. Es importante ya que al momento de revisar los *logs* es necesario tener la hora correcta. Se puede dejar la opción por defecto para el *hostname*, que es *2.pfsense.pool.ntp.org*. Luego se debe ingresar la zona horaria, en este caso, se selecciona *America/Guatemala* o GMT-6, que es la zona horaria de Guatemala.

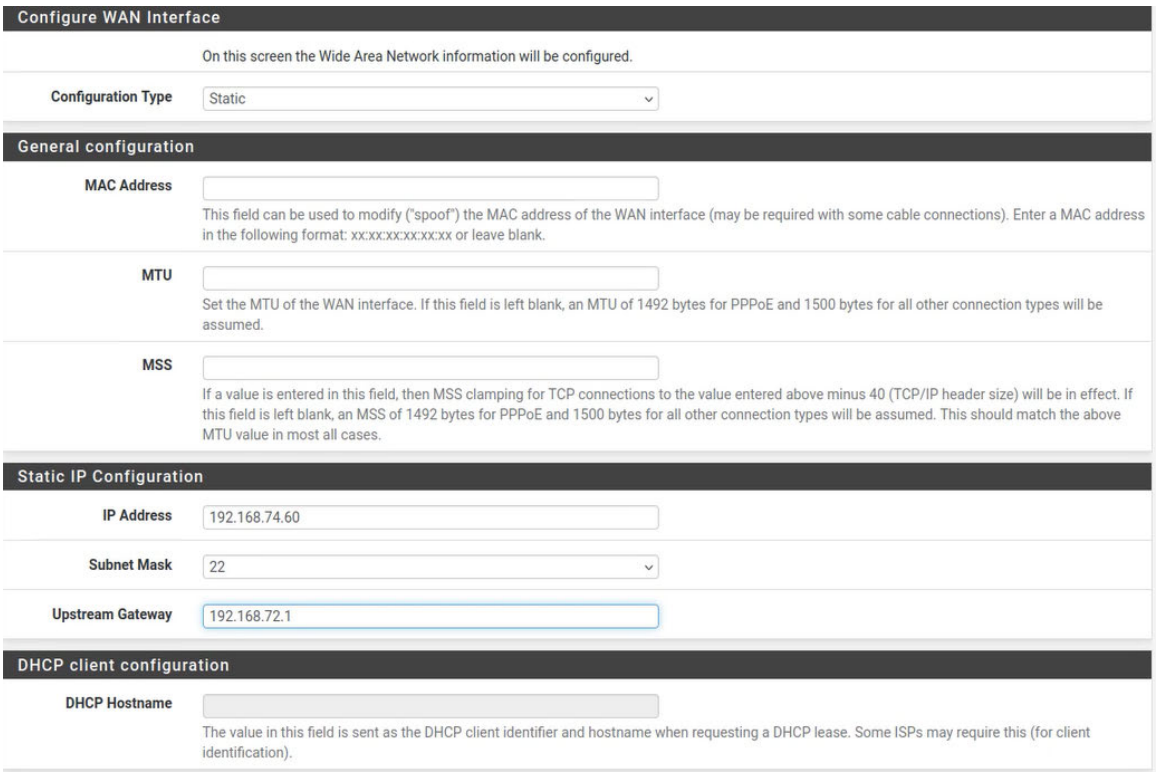
Figura 40. Configuración del *time server* en la interfaz web de pfSense.



Nota. Selección del servidor NTP y zona horaria (America/Guatemala) crítica para correlación correcta de eventos y logs. Elaboración propia.

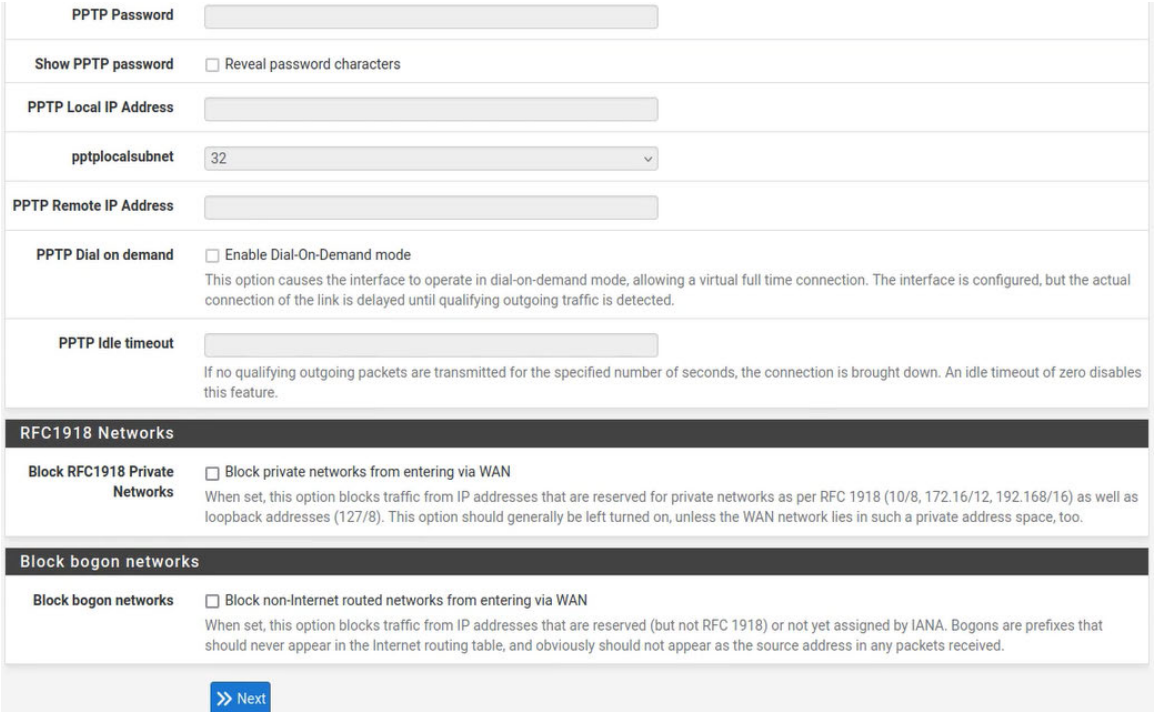
En la siguiente sección se configura la interfaz de red WAN. Anteriormente se configuraron unos parámetros, pero ahora se presentan más opciones. Se dejan las opciones por defecto. Se debe deseleccionar la opción *Block RFC1918 Private Networks*, que es la opción que bloquea las direcciones IP privadas, en este caso, no se quiere ya que la red WAN se encuentra en una red privada y se necesita permitir el acceso a la red interna. También, se debe deseleccionar la opción *Block Bogon Networks*, que es la opción que bloquea las direcciones IP que no están asignadas a ninguna organización.

Figura 41. Configuración de la interfaz de red WAN en la interfaz web de pfSense.



Nota. Configuración de parámetros varios de la interfaz WAN. Elaboración propia.

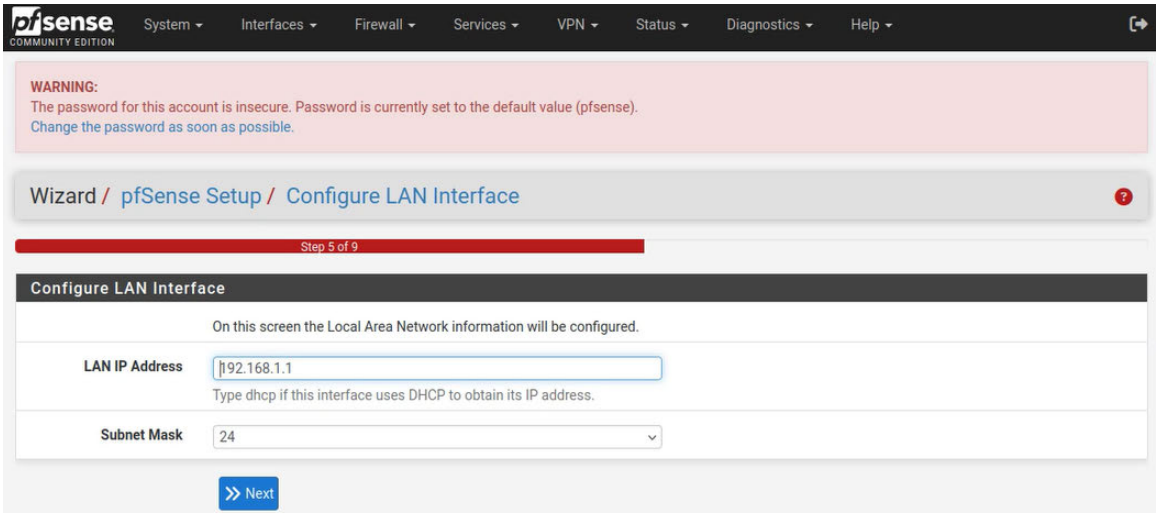
Figura 42. Configuración adicional de la interfaz de red WAN en la interfaz web de pfSense.



Nota. Parámetros avanzados de la interfaz WAN ajustando políticas de bloqueo RFC1918/Bogon para permitir tránsito desde red upstream privada. Elaboración propia.

En la siguiente sección se configura la interfaz de red LAN. Estas opciones son las mismas que se configuraron anteriormente, así que se dejan las mismas.

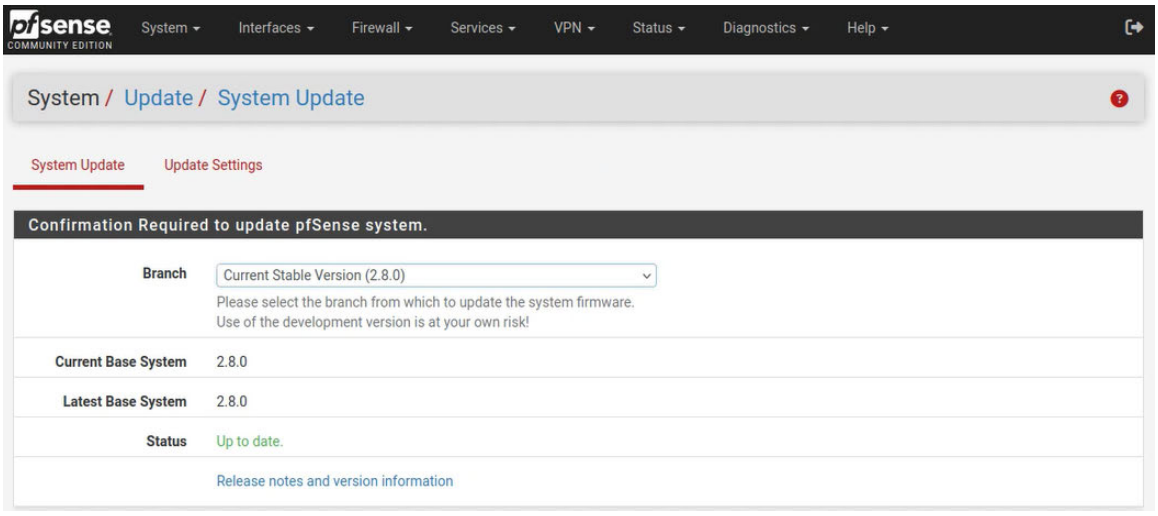
Figura 43. Configuración de la interfaz de red LAN en la interfaz web de pfSense.



Nota. Parámetros de la interfaz LAN como la IP de la interfaz y la máscara de subred. Elaboración propia.

Se debe cambiar la contraseña por defecto así que se ingresa una nueva, y se vuelve a cargar la página para que se apliquen los cambios. Por último, verificar si hay actualizaciones disponibles, se procede a instalarlas. Si no hay actualizaciones disponibles, eso sería por la configuración inicial.

Figura 44. Versión actual en la interfaz web de pfSense.



Nota. Pantalla informativa indicando versión instalada y estado de actualizaciones tras finalizar asistente inicial. Elaboración propia.

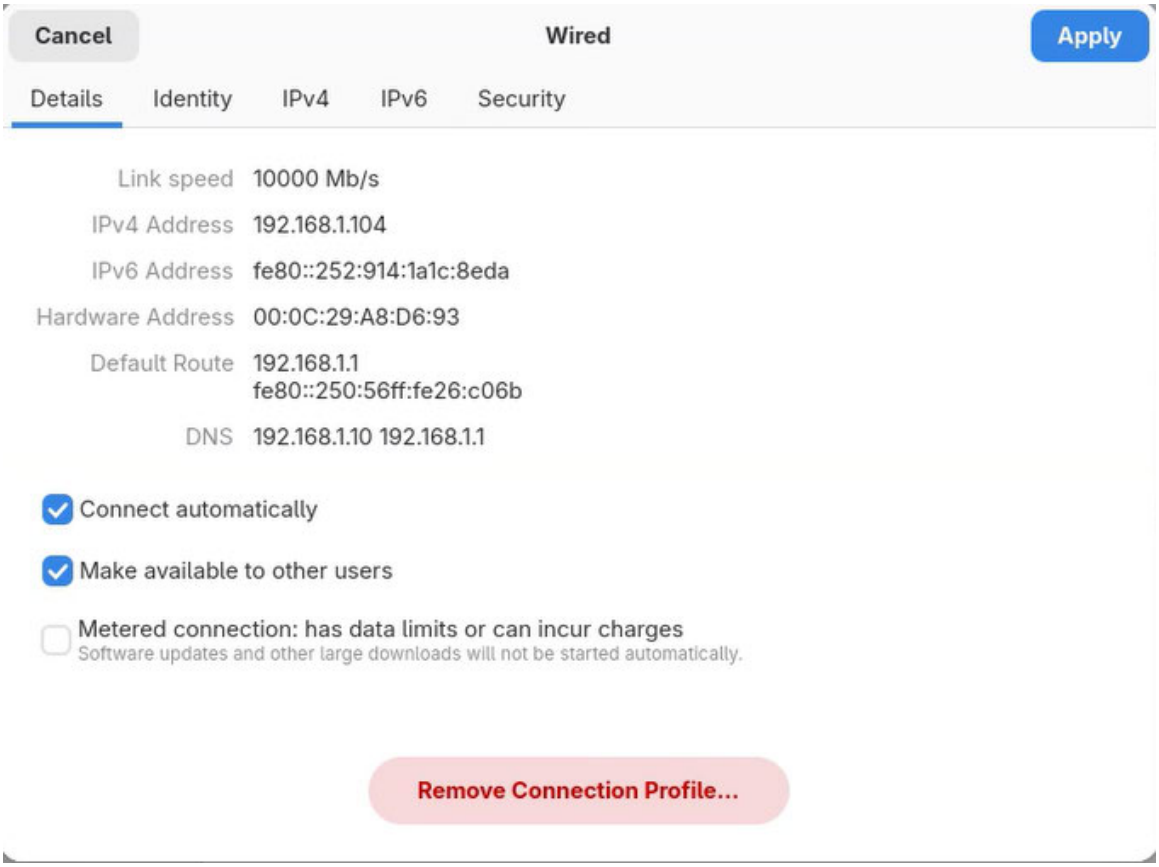
7.2.4.1. Usuario Fedora en la red LAN

Para la creación de la máquina virtual de Fedora en la red LAN, se debe descargar la imagen ISO de Fedora desde el sitio web oficial. Luego, como se realizó anteriormente para el usuario de Ubuntu, se debe crear una nueva máquina virtual en VMware Workstation Pro 17 con los siguientes requerimientos:

- Sistema operativo: Fedora Workstation Live 42.1.1
- Memoria: 4GB
- Espacio de disco duro: 60 GB
- Procesadores/Núcleos por procesador: 1/2
- Adaptador de red: *custom*, VMnet2 (que es la red LAN que se configuró en el *virtual network editor*). Se debe seleccionar la opción *Generate a new MAC address* para evitar conflictos de red.

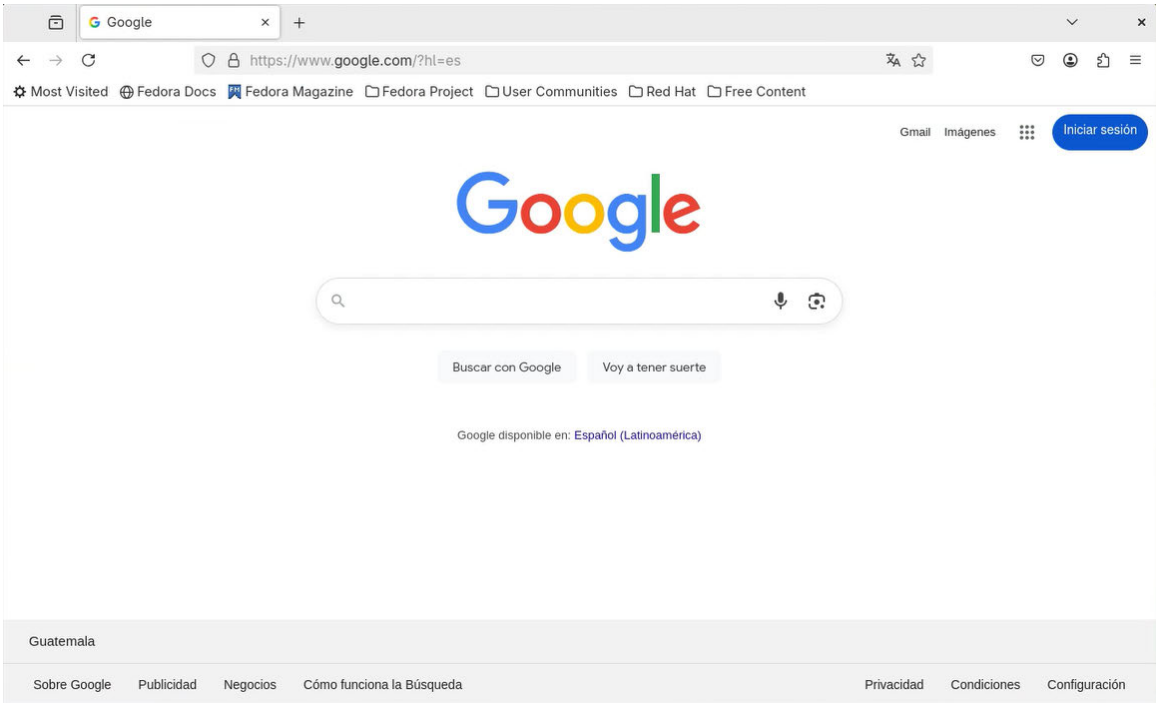
Al instalarse Fedora, para confirmar que se está conectado a la red LAN, se puede abrir los ajustes de red y verificar que se haya obtenido una dirección IP en el rango de la red LAN, y que se tenga acceso a Internet. En este caso, se ve que se ha obtenido la dirección IP 192.168.1.104.

Figura 45. Configuración de red en Fedora.



Nota. Adaptador de Fedora recibiendo IP válida del rango LAN confirmando correcto funcionamiento de DHCP y conectividad. Elaboración propia.

Figura 46. Verificación de conexión a Internet en Fedora.



Nota. Prueba de acceso a Internet desde Fedora corroborando resolución DNS y salida a la WAN a través del *firewall*. Elaboración propia.

7.2.4.2. Usuario Kali Linux en la red LAN

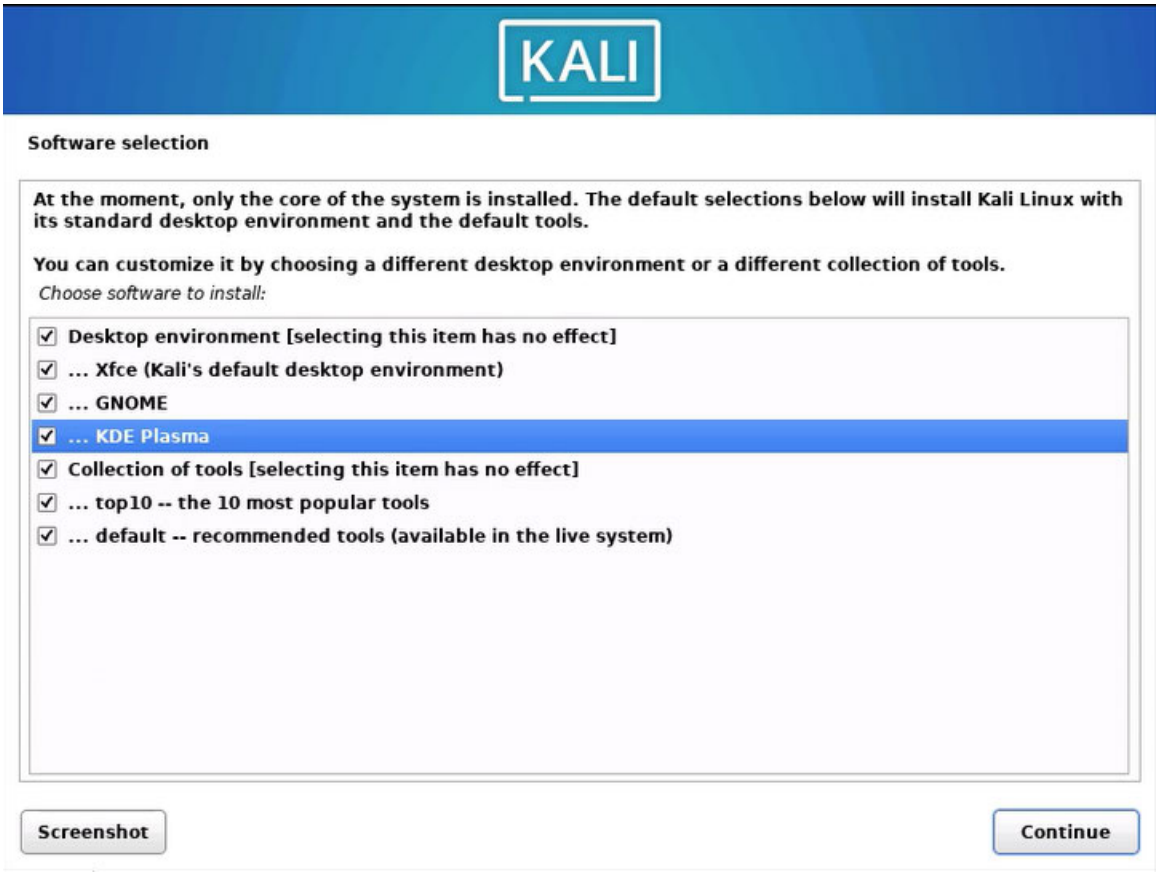
Kali Linux es una distribución de Linux basada en Debian diseñada para pruebas de penetración y auditorías de seguridad. En este caso, se instala Kali Linux en una máquina virtual que está conectada a la red LAN para realizar pruebas de seguridad internas y preparar utilidades de seguridad que se utilizan posteriormente.

Para la creación de la máquina virtual de Kali Linux en la red LAN, se debe descargar la imagen ISO de Kali Linux desde el sitio web oficial. Luego, como se realizó anteriormente se debe crear una nueva máquina virtual en VMware Workstation Pro 17 con los siguientes requerimientos:

- Sistema operativo: Kali Linux 2025.2
- Memoria: 16GB
- Espacio de disco duro: 100 GB
- Procesadores/Núcleos por procesador: 1/8
- Adaptador de red: *custom*, VMnet2 (que es la red LAN que se configuró en el *virtual network editor*). Se debe seleccionar la opción *Generate a new MAC address* para evitar conflictos de red.

Al iniciar la máquina virtual, se debe seleccionar la opción *Graphical Install* para iniciar la instalación de Kali Linux. Luego se procede con la instalación normal, seleccionando las opciones predeterminadas. En la sección de selección de software se eligen todas las casillas.

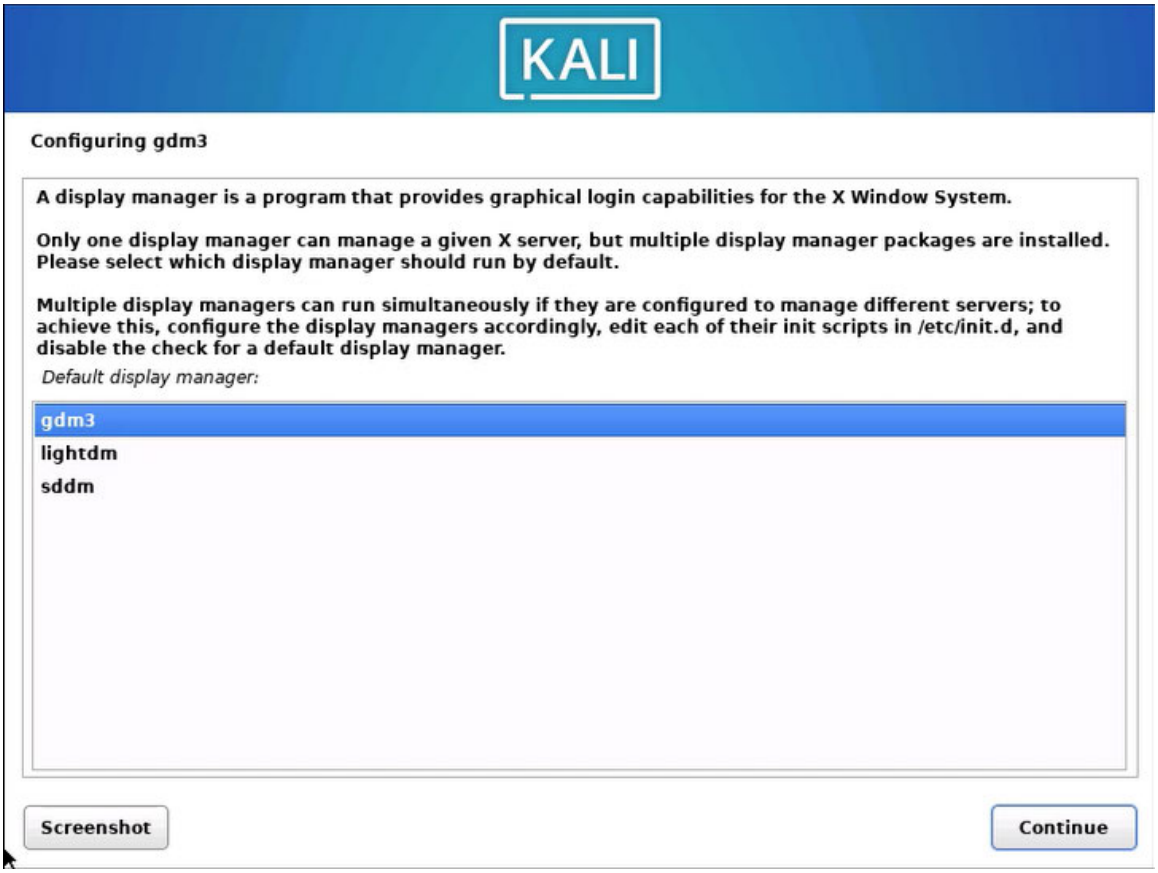
Figura 47. Selección de software en la instalación de Kali Linux.



Nota. Pantalla de selección de metapaquetes para dotar a Kali de herramientas amplias maximizando su utilidad en pruebas posteriores. Elaboración propia.

En la sección *display manager* se selecciona la opción *gdm3*, que es el administrador de ventanas predeterminado de Kali Linux.

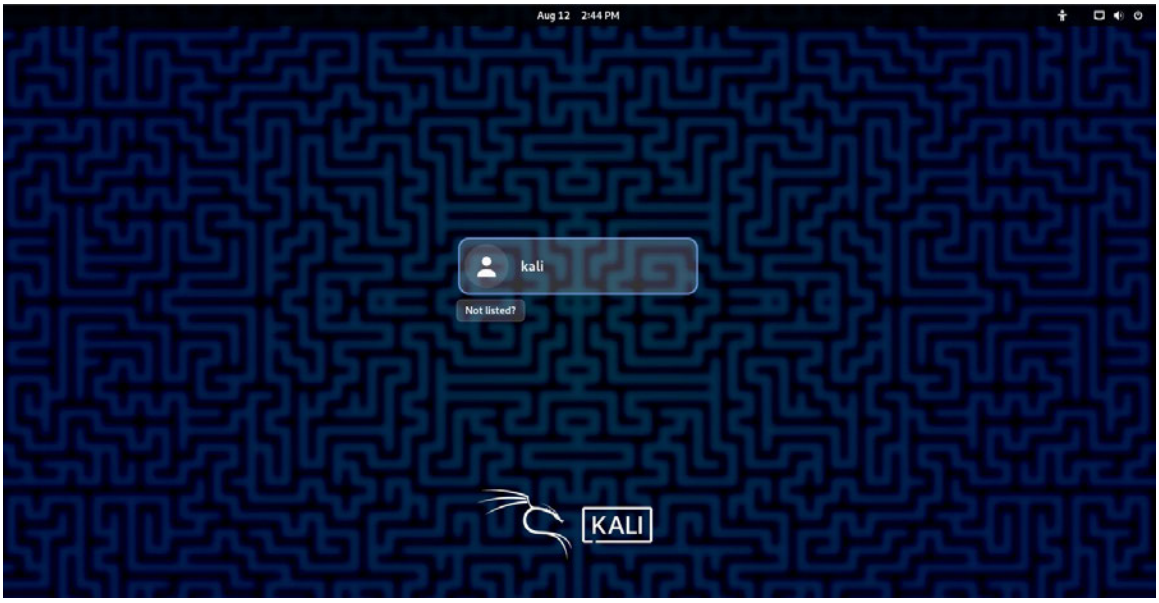
Figura 48. Selección de *display manager* en la instalación de Kali Linux.



Nota. Elección de gdm3 como gestor gráfico estable para facilitar uso de utilidades GUI de seguridad y análisis. Elaboración propia.

Por último se debe instalar el gestor de arranque GRUB, que es el gestor de arranque predeterminado de Kali Linux. Se selecciona la opción instalar GRUB en el disco duro y se selecciona el disco duro donde se ha instalado Kali Linux. En este caso, se selecciona el disco duro /dev/sda. Luego, se procede a reiniciar la máquina virtual.

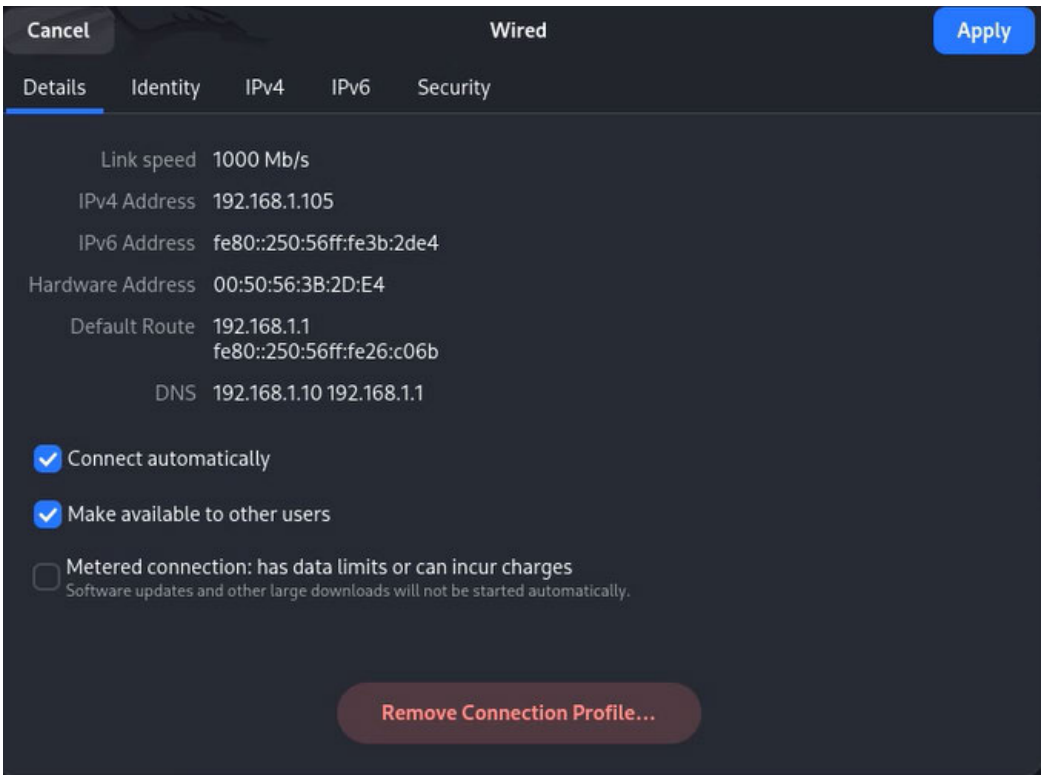
Figura 49. Pantalla de inicio de sesión de Kali Linux.



Nota. Pantalla de inicio de sesión de Kali Linux mostrando el entorno gráfico. Elaboración propia.

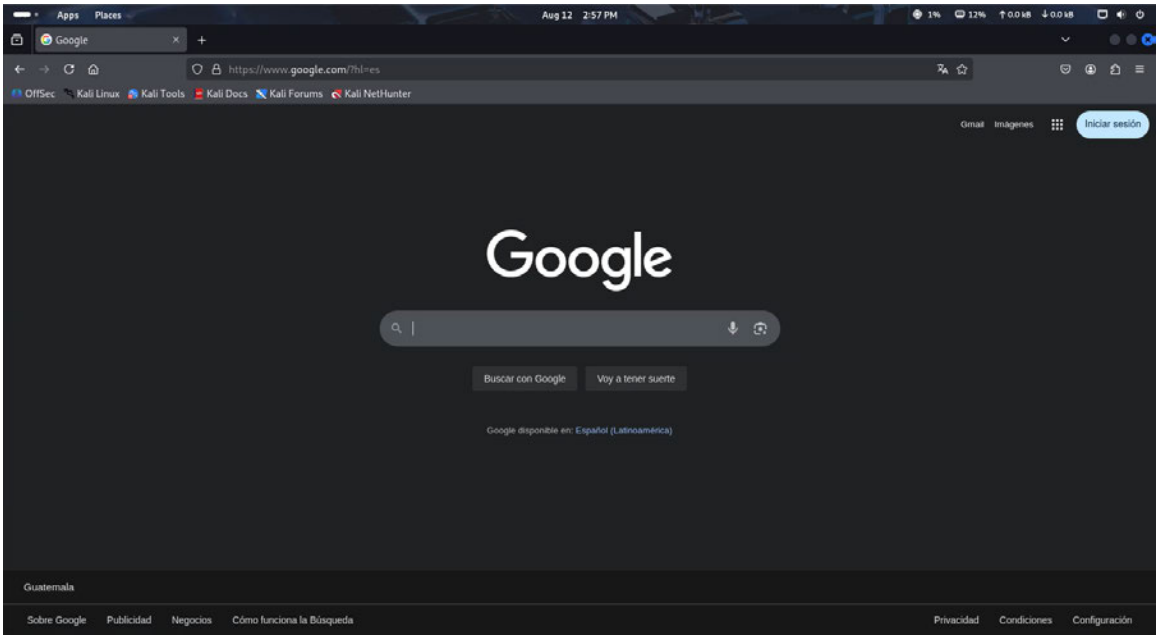
Al instalarse Kali Linux, para confirmar que se está conectado a la red LAN, se puede abrir los ajustes de red y verificar que se haya obtenido una dirección IP en el rango de la red LAN, y que se tenga acceso a Internet. En este caso, se ve que se ha obtenido la dirección IP 192.168.1.105.

Figura 50. Configuración de red en Kali Linux.



Nota. Interfaz de red de Kali obteniendo IP válida del segmento LAN validando conectividad. Elaboración propia.

Figura 51. Verificación de conexión a Internet en Kali Linux.

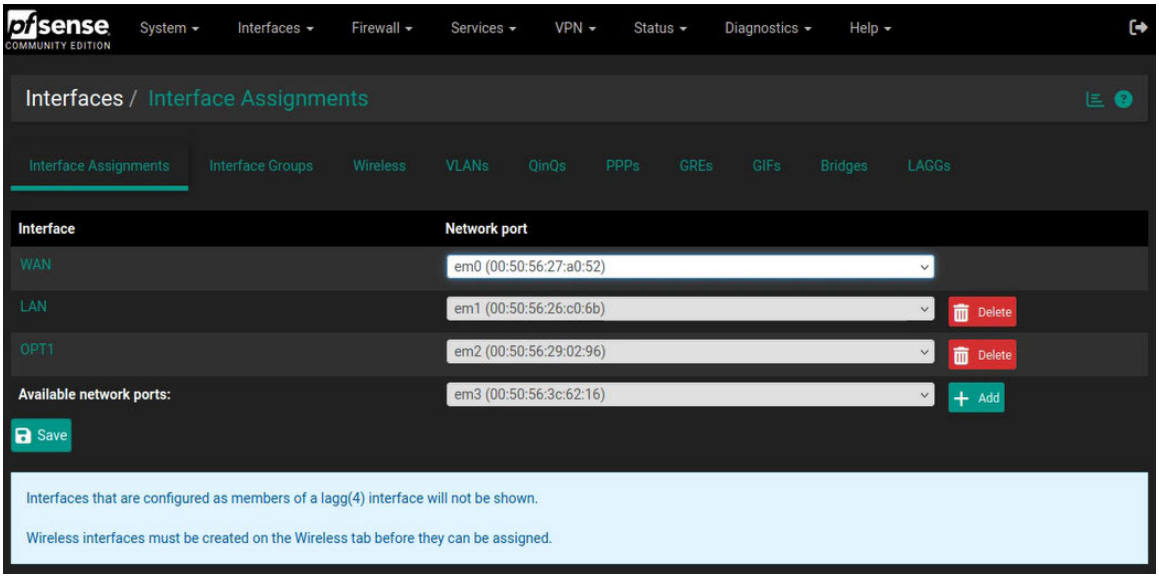


Nota. Prueba de salida a Internet desde Kali accediendo a google.com. Elaboración propia.

7.2.5. Configuración de la red DMZ

Lo siguiente sería configurar la interfaz de red DMZ. Esta zona incluye los servidores que están expuestos a Internet, en este caso es la máquina virtual Metasploitable 3. Para ello, se debe ir a la sección *Interfaces* en el menú principal y seleccionar la opción *Assignments*. En esta sección se pueden ver las interfaces de red que se han configurado hasta el momento, que son la interfaz de red WAN y la interfaz de red LAN. Para agregar una nueva interfaz de red, primero se debe seleccionar el adaptador de red adecuado. En este caso, es para la red DMZ por lo que, se selecciona dicho adaptador, luego se debe seleccionar la opción *Add*.

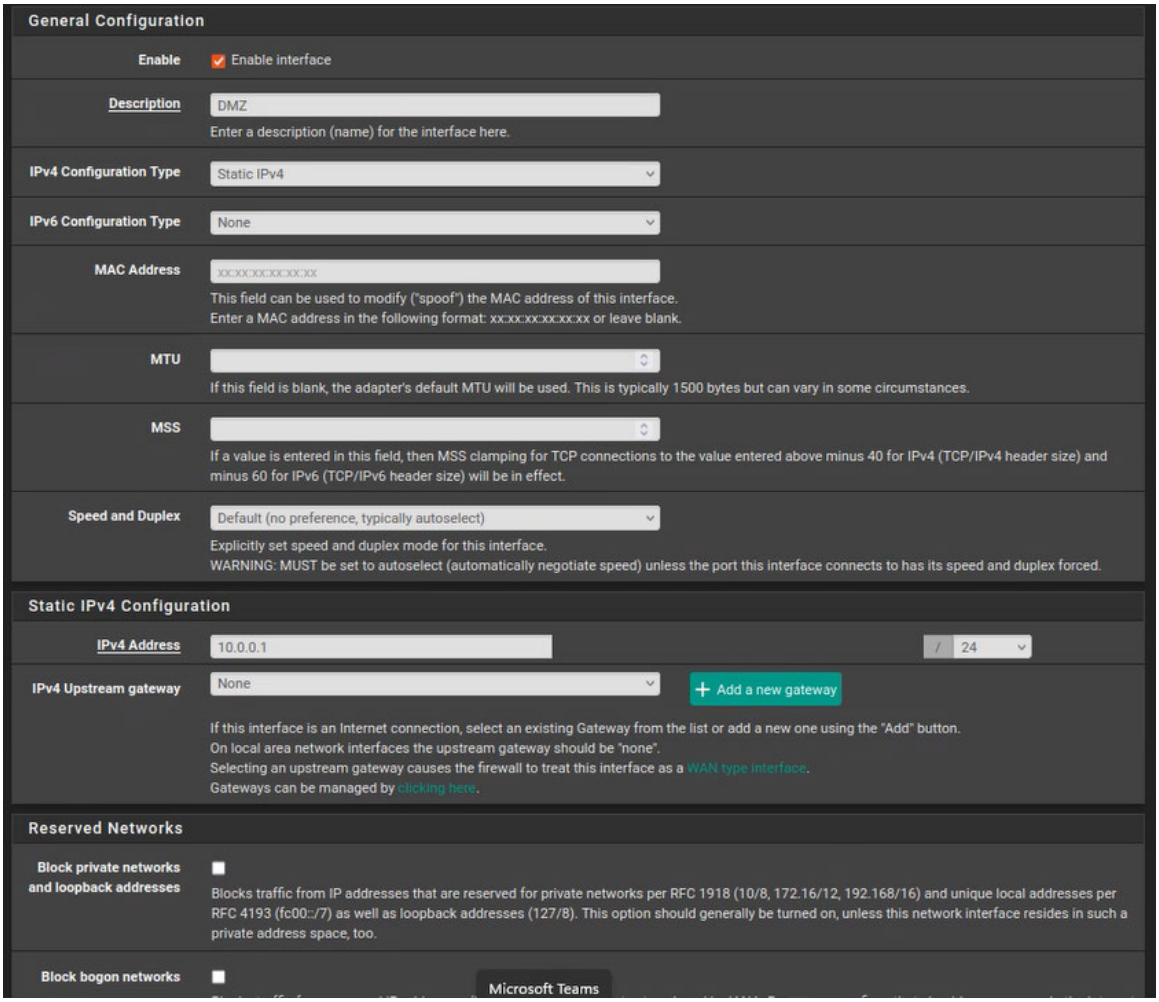
Figura 52. Asignación de nuevo adaptador para la interfaz DMZ en la interfaz web de pfSense.



Nota. Pantalla de asignación donde se añade adaptador para futura DMZ previo a su habilitación y configuración estática. Elaboración propia.

Se debe configurar la interfaz de red DMZ. Para ello, se debe seleccionar la opción OPT1 en el menú desplegable de interfaces. Luego, se debe ingresar un nombre para la interfaz, en este caso, DMZ y se habilita la interfaz. Se define el tipo de configuración IPv4 como estática.

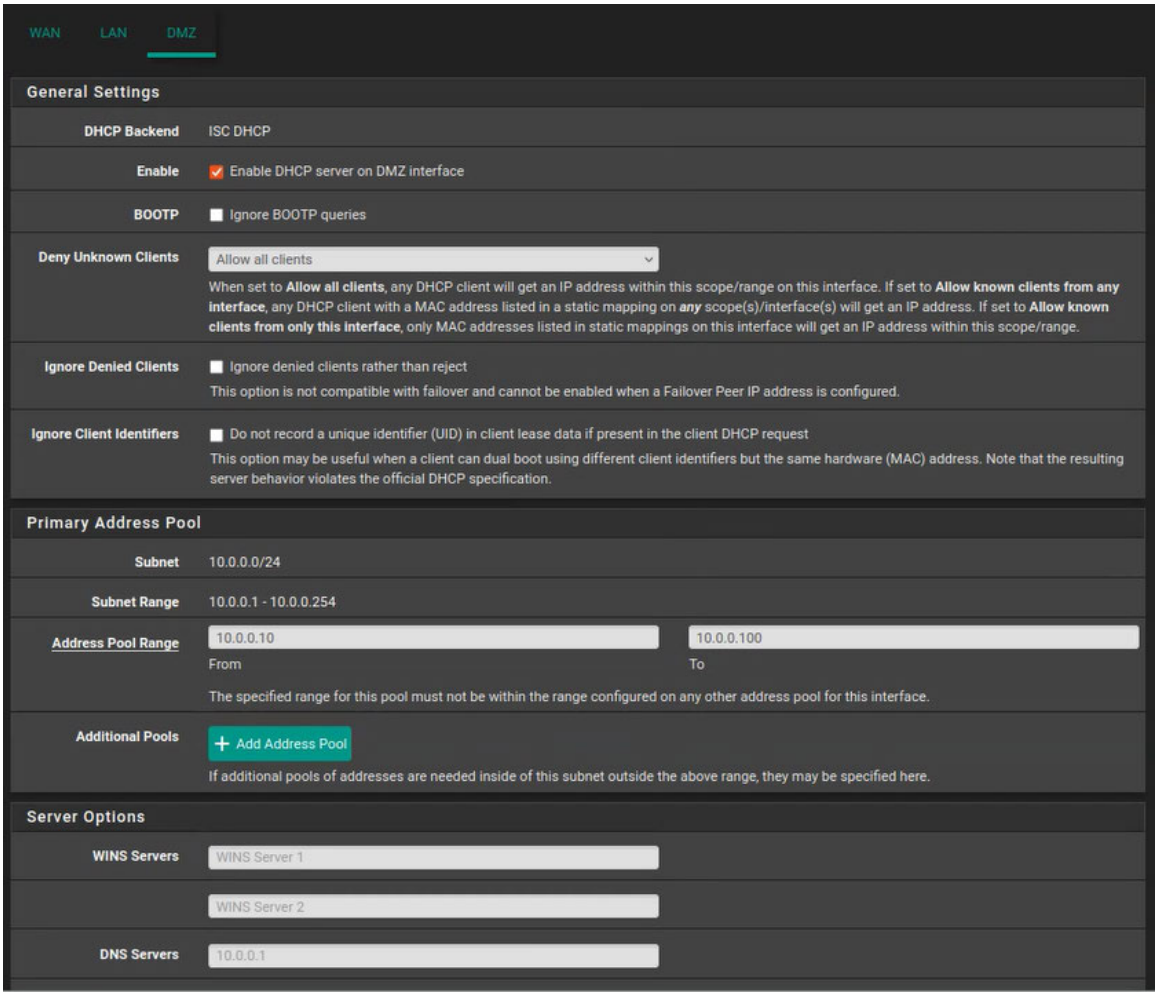
Figura 53. Configuración de la interfaz de red DMZ en la interfaz web de pfSense.



Nota. Habilitación y nombrado de la interfaz OPT1 como DMZ con direccionamiento estático base para servicios públicos controlados. Elaboración propia.

Se debe ingresar la dirección IP de la interfaz de red DMZ, es la dirección IP del *gateway* para los servidores que están en la red DMZ. En este caso, la dirección IP 10.0.0.1/24. Además, se debe habilitar el servidor DHCP para que los servidores que estén en la red DMZ puedan obtener una dirección IP automáticamente y su respectivo servidor DNS. Para ello, se debe seleccionar la opción *Services* en el menú principal y luego seleccionar la opción DHCP *Server*. En esta sección se selecciona la interfaz de red DMZ y se habilita el servidor DHCP. Luego, se debe ingresar el rango de direcciones IP que se asignan a los servidores que estén en la red DMZ. En este caso, el rango de direcciones IP 10.0.0.10 - 10.0.0.100.

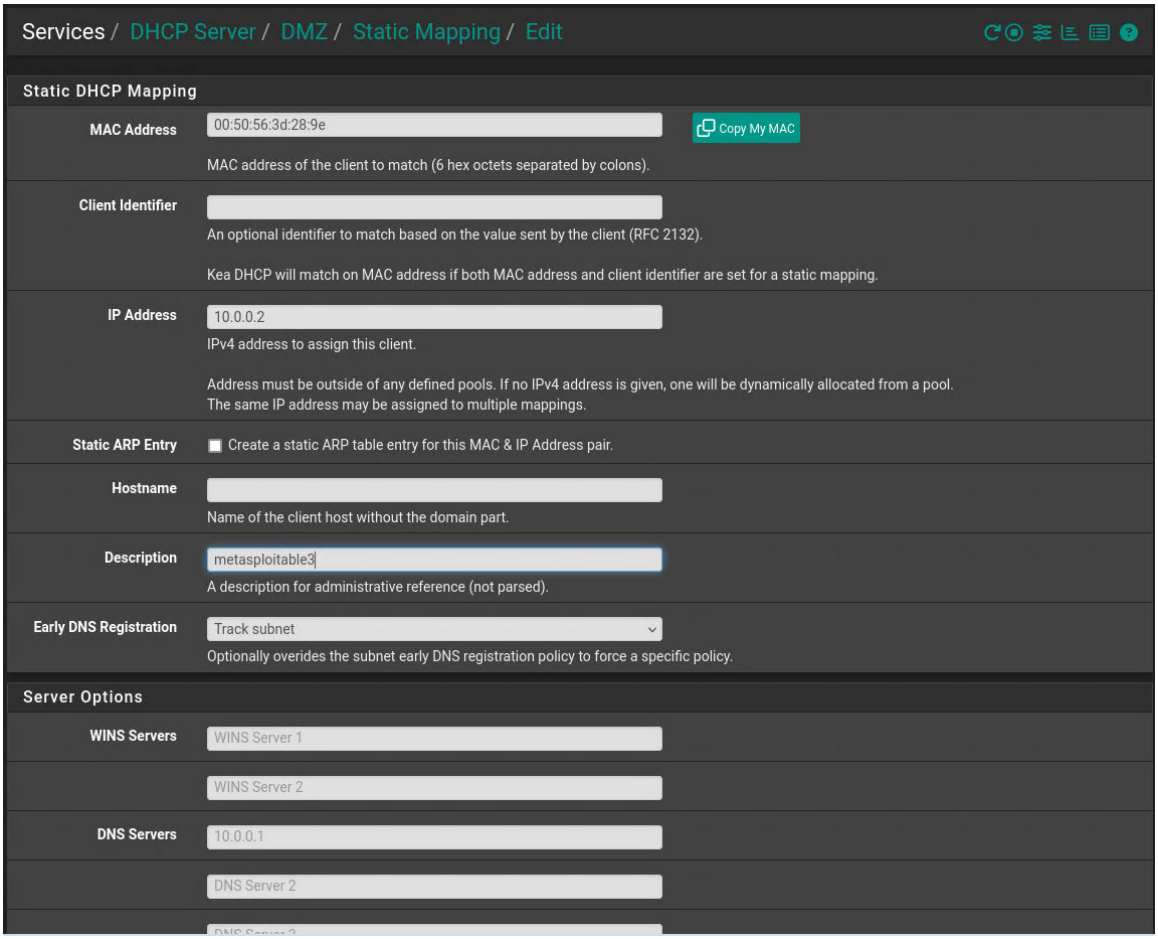
Figura 54. Configuración del servidor DHCP para la red DMZ (rango 10.0.0.10-100).



Nota. Ajuste del servidor DHCP para DMZ definiendo rango controlado (10.0.0.10-10.0.0.100) facilitando pruebas y asignaciones posteriores. Elaboración propia.

En la parte inferior de la página se puede ver la opción DHCP *Static Mappings*, que es la opción que permite asignar direcciones IP estáticas a los servidores que estén en la red DMZ. En este caso, se requiere que Metasploitable 3 tenga una dirección IP estática, ya que no se quiere que cambie la dirección IP cada vez que se reinicie el servidor. Para ello, se debe seleccionar la opción *Add* y luego ingresar la dirección MAC de la máquina virtual, en este caso, es la dirección MAC que se genera en la configuración de la máquina virtual. Luego, se debe ingresar la dirección IP que se asigna al servidor, en este caso, 10.0.0.2. Por último, se debe ingresar una descripción para la asignación de la dirección IP estática, en este caso, Metasploitable 3.

Figura 55. Configuración de asignación estática IP en la interfaz web de pfSense.

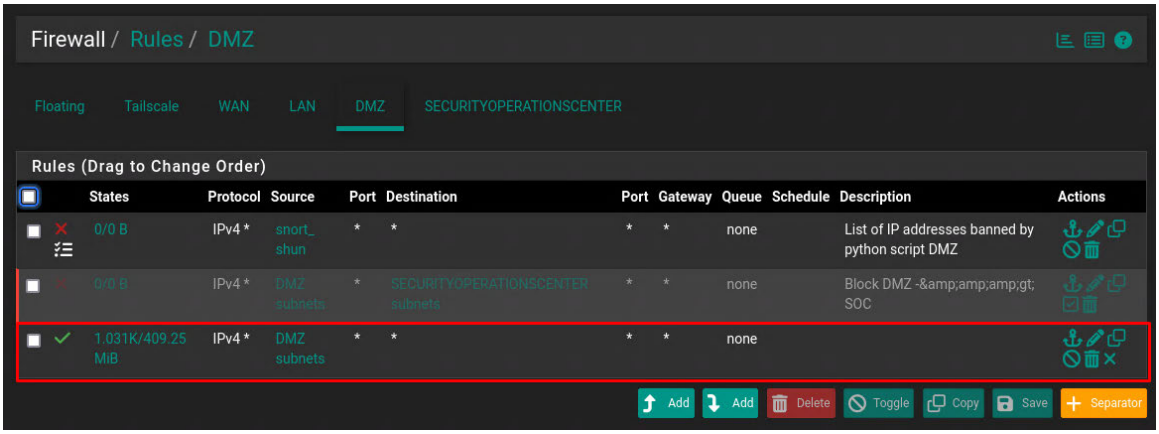


Nota. Creación de mapeo estático DHCP para la máquina virtual Metasploitable 3 en DMZ garantizando consistencia de direccionamiento (10.0.0.2). Elaboración propia.

7.2.5.1. Creación de regla para tener acceso a Internet desde la red DMZ

Para que los servidores que estén en la red DMZ puedan tener acceso a Internet, se debe crear una regla en el *firewall* de pfSense. Para ello, se debe seleccionar la opción *Firewall* en el menú principal y luego seleccionar la opción *Rules*. En esta sección se selecciona la interfaz de red DMZ y se debe crear una nueva regla. Para ello, se debe seleccionar la opción *Add* y luego se debe configurar la regla.

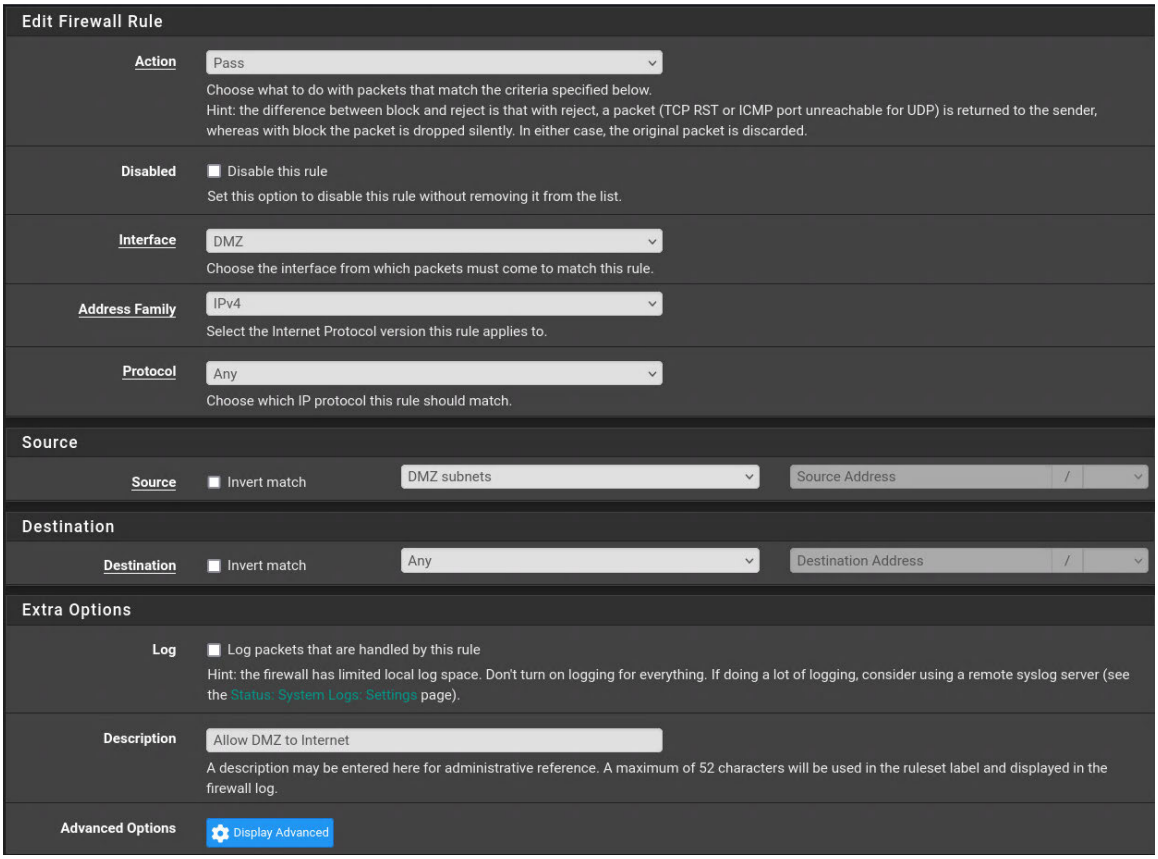
Figura 56. Creación de regla de *firewall* en la interfaz web de pfSense.



Nota. Definición de regla de *firewall* permitiendo tráfico saliente desde DMZ hacia cualquier destino (Internet). Elaboración propia.

Se debe seleccionar la acción de la regla, en este caso, *Pass*, que es la acción que permite el tráfico. Luego, se debe seleccionar el protocolo, en este caso, *Any*, que es el protocolo que permite todo tipo de tráfico. En la sección *Source* se debe seleccionar la opción de la subred de la red DMZ. En la sección *Destination* se debe seleccionar la opción *Any*, que es cualquier dirección IP. Por último, se debe ingresar una descripción para la regla, en este caso, *Allow DMZ to Internet*. Se guarda la regla y se aplican los cambios.

Figura 57. Configuración de regla de *firewall* en la interfaz web de pfSense.



Nota. Resumen de configuración de regla permitiendo acceso a Internet desde DMZ. Elaboración propia.

Esto no se realizó para la red LAN ya que pfSense crea una regla por defecto que permite el acceso a Internet desde la red LAN.

7.2.6. Configuración de la máquina virtual Metasploitable 3 para la red DMZ

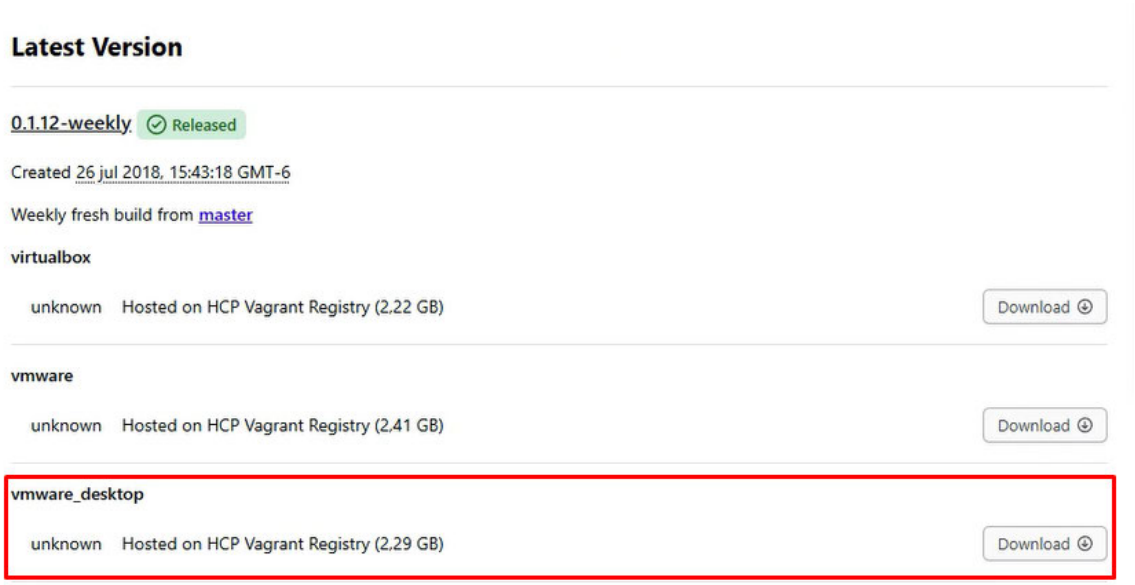
Para la red DMZ se utiliza la máquina virtual Metasploitable 3, que es una máquina virtual vulnerable que se utiliza para realizar pruebas de penetración y aprender sobre seguridad informática. Esta máquina virtual se conecta a la red DMZ que se configuró anteriormente en pfSense. Se utiliza la versión de Ubuntu *Server* 14.04. Para ello se debe descargar la imagen de la máquina virtual desde la página oficial de Metasploitable 3, que se encuentra en el siguiente enlace: https://portal.cloud.hashicorp.com/vagrant/discover/rapid7.

Figura 58. Descarga de la imagen de Metasploitable 3.

Nota. Selección del recurso oficial de Metasploitable 3 necesario para contar con un entorno vulnerable controlado en la DMZ. Elaboración propia.

Luego, se debe descargar la versión llamada *vmware_desktop*, que es la versión que se puede utilizar en VMware Workstation Pro.

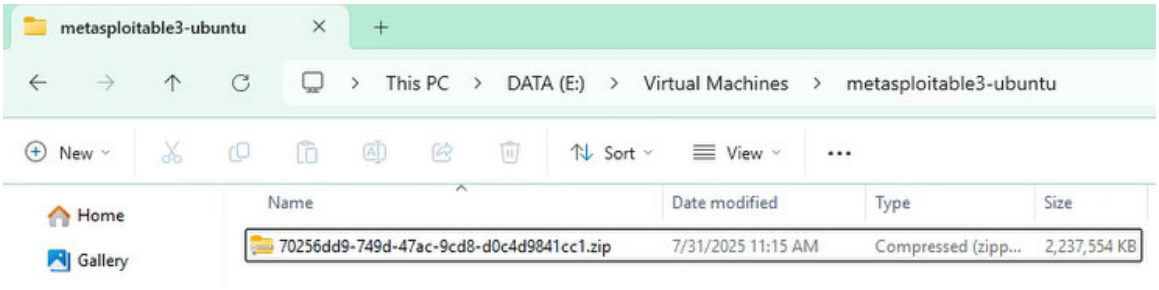
Figura 59. Versión de Metasploitable 3 para VMware.



Nota. Identificación de la variante `vmware_desktop` adecuada para importación directa en VMware Workstation. Elaboración propia.

Una vez descargado el archivo, se puede notar que no tiene ningún tipo de extensión, que pueda usarse en VMware Workstation Pro. Por lo que se debe cambiar la extensión del archivo a *.zip*.

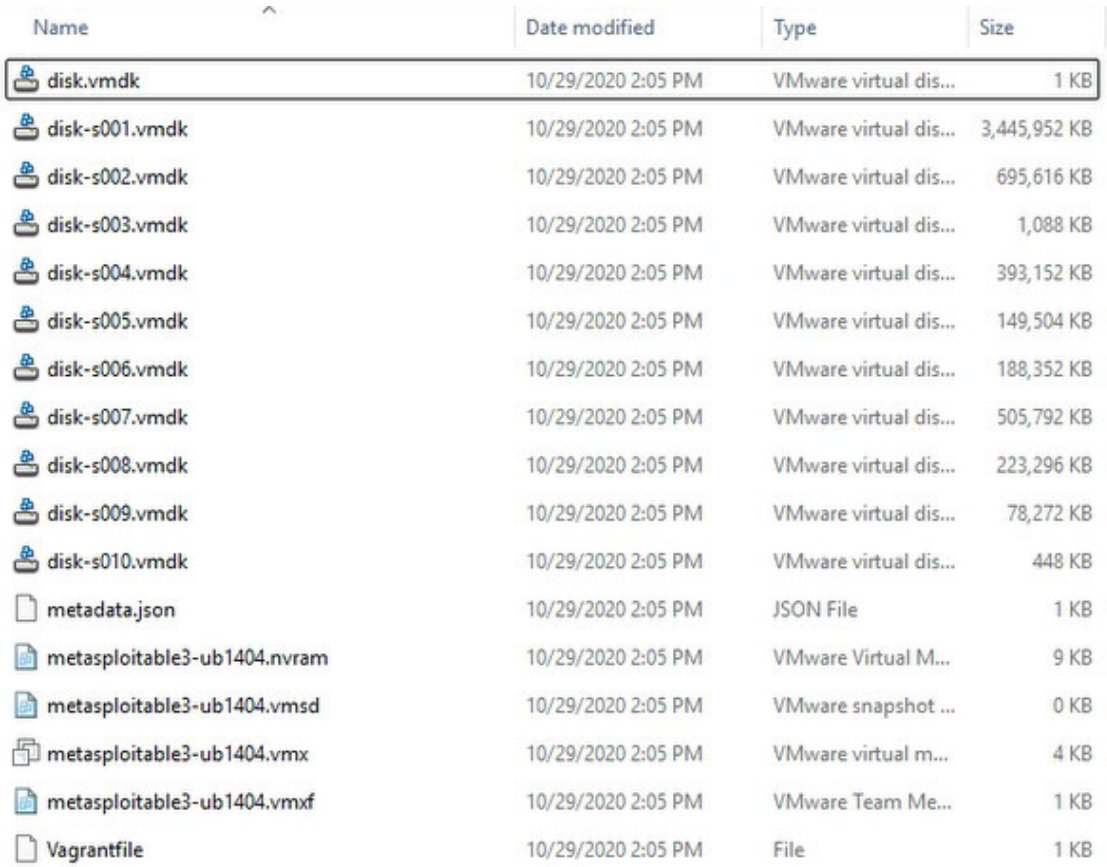
Figura 60. Cambio de extensión del archivo de Metasploitable 3.



Nota. Renombrado del paquete descargado para permitir su descompresión y acceso a archivos de la VM. Elaboración propia.

Luego, se debe descomprimir el archivo y se obtiene un archivo llamado de la misma manera. Se elimina el archivo comprimido anterior y se renombra el archivo de nueva forma con extensión *.zip*. Se debe descomprimir el archivo y se obtienen los archivos necesarios para crear la máquina virtual.

Figura 61. Archivo de Metasploitable 3 descomprimido.

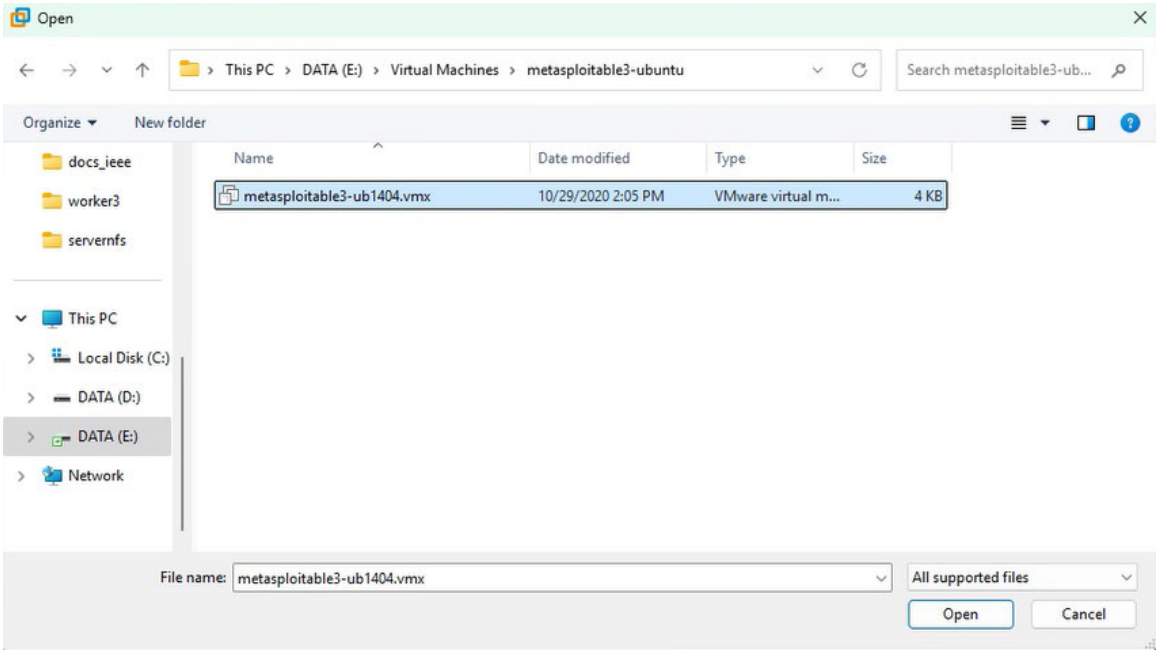


Name	Date modified	Type	Size
disk.vmdk	10/29/2020 2:05 PM	VMware virtual dis...	1 KB
disk-s001.vmdk	10/29/2020 2:05 PM	VMware virtual dis...	3,445,952 KB
disk-s002.vmdk	10/29/2020 2:05 PM	VMware virtual dis...	695,616 KB
disk-s003.vmdk	10/29/2020 2:05 PM	VMware virtual dis...	1,088 KB
disk-s004.vmdk	10/29/2020 2:05 PM	VMware virtual dis...	393,152 KB
disk-s005.vmdk	10/29/2020 2:05 PM	VMware virtual dis...	149,504 KB
disk-s006.vmdk	10/29/2020 2:05 PM	VMware virtual dis...	188,352 KB
disk-s007.vmdk	10/29/2020 2:05 PM	VMware virtual dis...	505,792 KB
disk-s008.vmdk	10/29/2020 2:05 PM	VMware virtual dis...	223,296 KB
disk-s009.vmdk	10/29/2020 2:05 PM	VMware virtual dis...	78,272 KB
disk-s010.vmdk	10/29/2020 2:05 PM	VMware virtual dis...	448 KB
metadata.json	10/29/2020 2:05 PM	JSON File	1 KB
metasploitable3-ub1404.nvram	10/29/2020 2:05 PM	VMware Virtual M...	9 KB
metasploitable3-ub1404.vmsd	10/29/2020 2:05 PM	VMware snapshot ...	0 KB
metasploitable3-ub1404.vmx	10/29/2020 2:05 PM	VMware virtual m...	4 KB
metasploitable3-ub1404.vmxf	10/29/2020 2:05 PM	VMware Team Me...	1 KB
Vagrantfile	10/29/2020 2:05 PM	File	1 KB

Nota. Contenido extraído listo para importación; estructura confirma integridad de la descarga. Elaboración propia.

Luego, se debe abrir VMware Workstation Pro y en la pestaña *File* se selecciona la opción *Open* para abrir el archivo de la máquina virtual. Se debe seleccionar uno de los archivo que se descomprimió anteriormente, en este caso, el archivo llamado *metasploitable3-ub1404.vmx*.

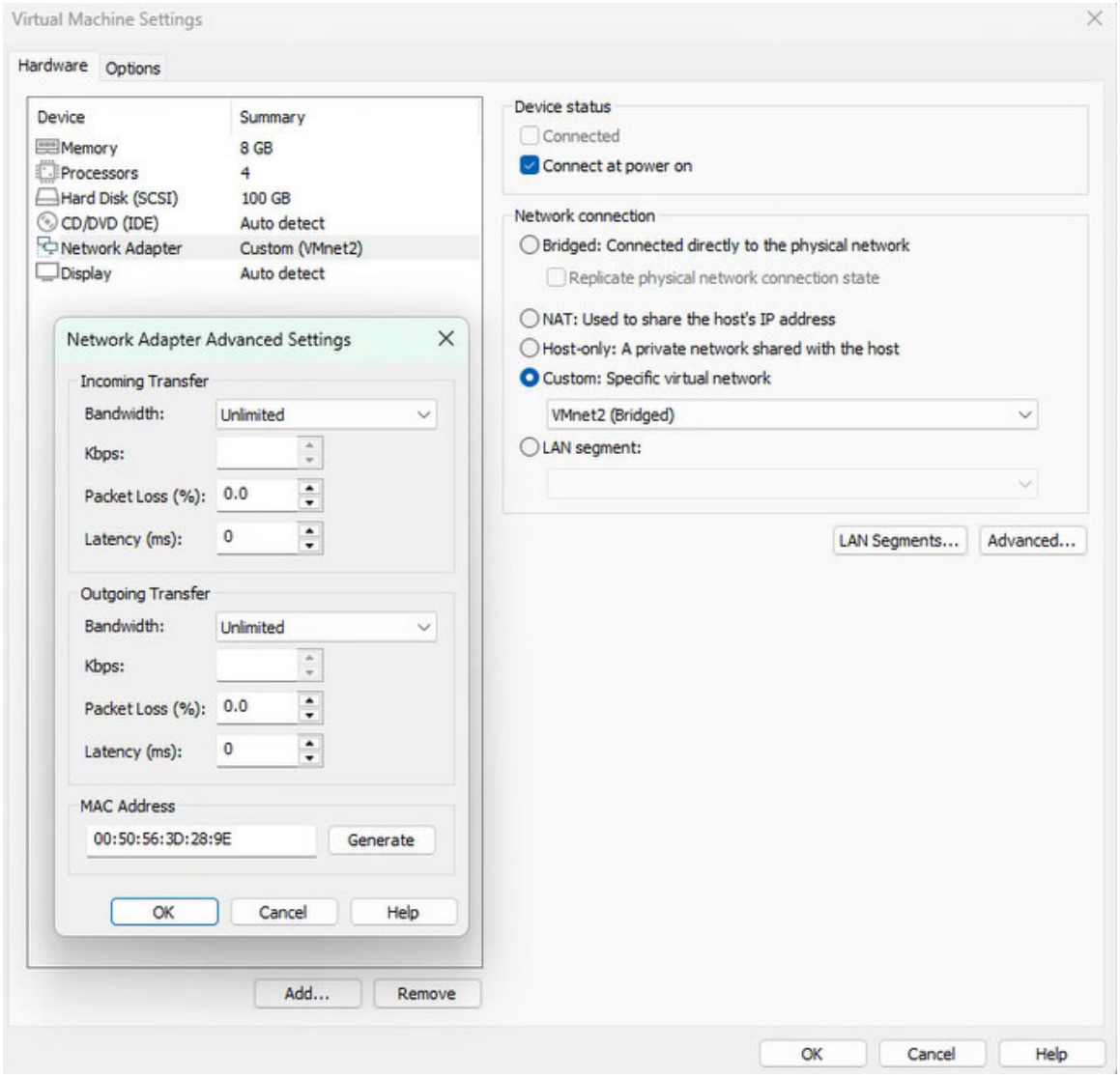
Figura 62. Apertura de la máquina virtual Metasploitable 3 en VMware.



Nota. Importación del archivo .vmx a VMware confirmando detección de hardware virtual. Elaboración propia.

Una vez hecho esto se puede ver que la máquina virtual se ha cargado correctamente y se puede ver la configuración de la máquina virtual. Ahora se procede a cambiar la configuración de la máquina virtual. Lo primero es cambiar la memoria RAM, que se debe cambiar a 8 GB; segundo, cambiar el espacio del disco duro, que se debe cambiar a 100 GB; y por último, cambiar el adaptador de red, que se debe cambiar a la red DMZ que se configuró anteriormente, en este caso, la red *VMnet2*. Se debe seleccionar la opción *Generate a new MAC address* para evitar conflictos de red.

Figura 63. Configuración de la máquina virtual Metasploitable 3 en VMware.



Nota. Ajuste de recursos (RAM, disco, red DMZ) asegurando condiciones realistas para explotación controlada. Elaboración propia.

Una vez configurada la máquina virtual, se procede a encenderla. Al encender la máquina virtual, se puede ver que se está ejecutando el sistema operativo Ubuntu *Server* 14.04. Se debe iniciar sesión con las credenciales por defecto, que son:

- Usuario: *vagrant*
- Contraseña: *vagrant*

Figura 64. Inicio de sesión en Metasploitable 3.

Nota. Acceso inicial con credenciales por defecto. Elaboración propia.

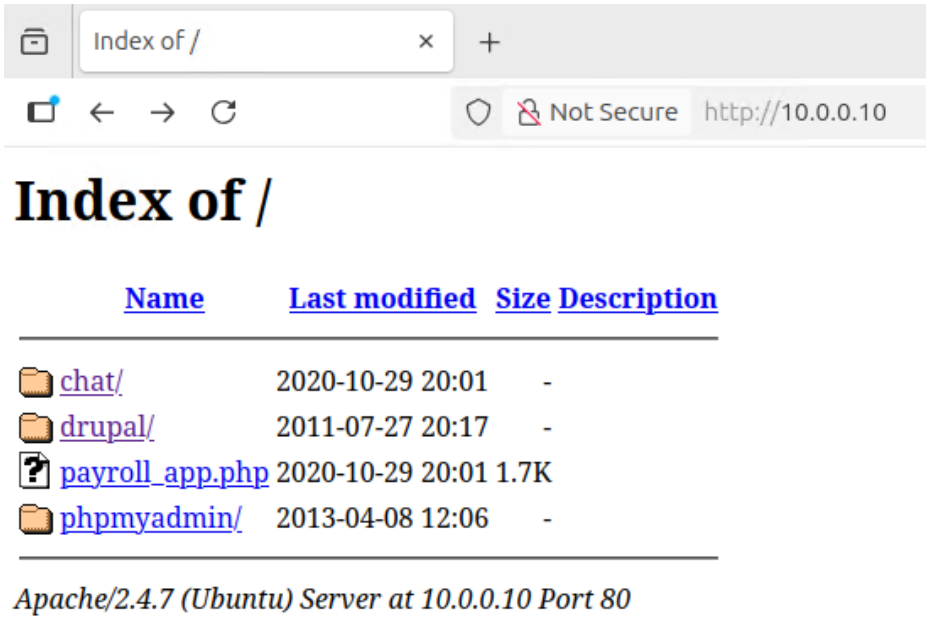
Una vez iniciada la sesión, se puede ver que la máquina virtual está configurada correctamente y se puede acceder a Internet. Para comprobar que la máquina virtual está conectada a la red DMZ, se puede ejecutar el comando *ifconfig* en la terminal. Esto muestra la configuración de red de la máquina virtual, donde se puede ver que tiene una dirección IP en la red DMZ, en este caso, 10.0.0.2, que es correcta ya que es el rango de direcciones IP que se configuró en el servidor DHCP de la red DMZ.

Figura 65. Configuración de red en Metasploitable 3.

Nota. Salida de ifconfig mostrando IP 10.0.0.2 dentro del rango DMZ validando conectividad previa a escaneos. Elaboración propia.

Se puede comprobar que el servidor web está funcionando correctamente accediendo a la dirección IP de la máquina virtual desde un navegador web. Si se puede ver la página de inicio que contiene un directorio con varios archivos y carpetas, significa que el servidor web está funcionando correctamente.

Figura 66. Acceso al sitio web de Metasploitable 3 desde el navegador.



Nota. Validación del servicio web Apache por defecto en Metasploitable 3 confirmando disponibilidad para pruebas. Elaboración propia.

7.2.7. Reenvío de puertos para Metasploitable 3

Para realizar pruebas de penetración en la máquina virtual Metasploitable 3, es necesario configurar el reenvío de puertos en el *firewall*, ya que esto permite que el tráfico de red se redirija a la máquina virtual desde el exterior y se puede acceder.

Antes que nada hay que modificar el archivo de configuración de Apache 2 de Metasploitable 3 para permitir el acceso a los servicios web desde el exterior. Esto se puede hacer editando el archivo `/etc/apache2/sites-available/000-default.conf` y asegurándose de que la siguiente línea esté presente:

Cuadro 5. Configuración de Apache en Metasploitable 3.

```
<VirtualHost *:80>
    ...
    ServerName sini58.example.com
    DocumentRoot /var/www/html
    ...
</VirtualHost>
```

Nota. La tabla muestra la configuración del servidor Apache en Metasploitable 3 con la definición y parámetros de servidor. Elaboración propia.

Este *ServerName* es el nombre de dominio que se utiliza para acceder a los servicios web que se están ejecutando en la máquina virtual Metasploitable 3, este se puede cambiar por cualquier otro nombre de dominio que se desee utilizar.

Luego se debe reiniciar el servicio de Apache 2 para que los cambios surtan efecto, esto se puede hacer ejecutando el siguiente comando:

Cuadro 6. Reinicio del servicio Apache en Metasploitable 3.

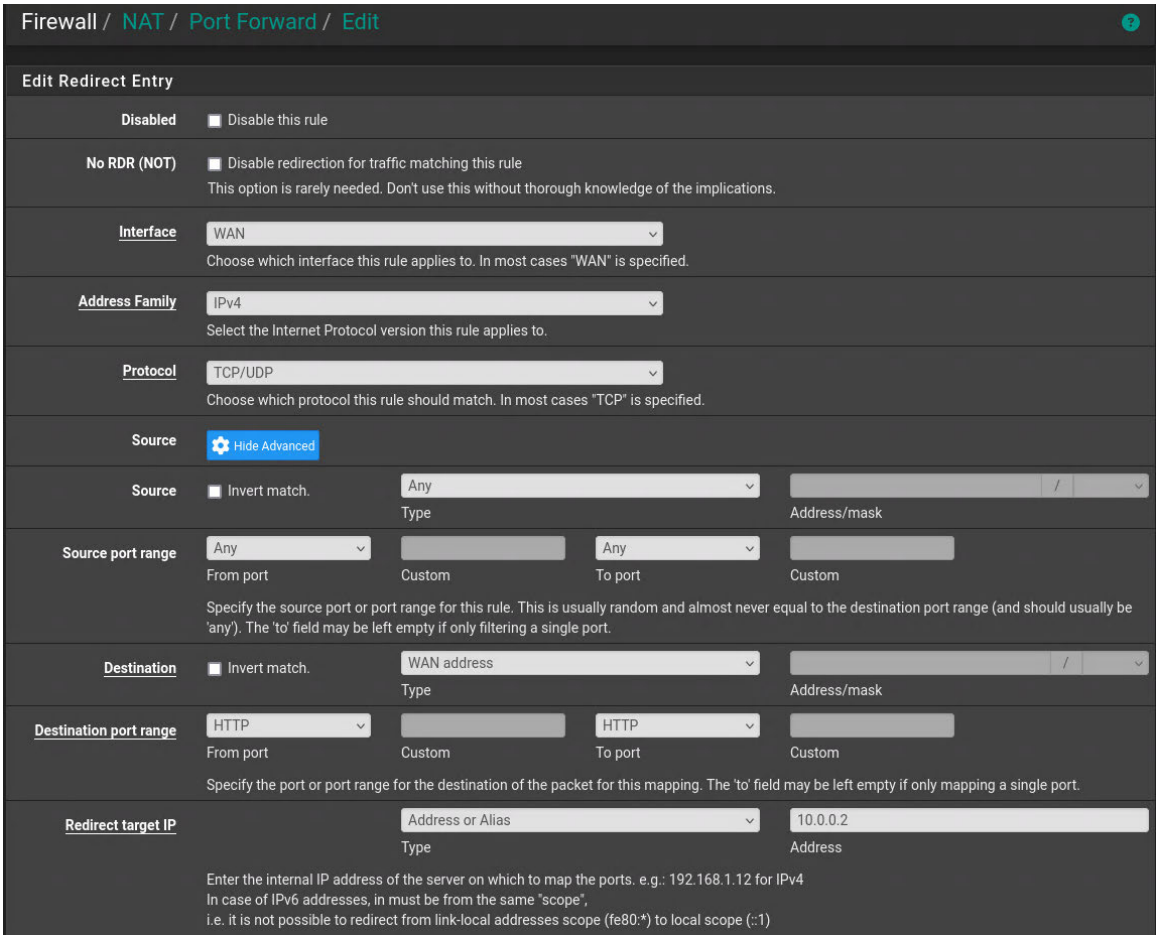
```
1 sudo service apache2 restart
```

Nota. El comando mostrado reinicia el servicio Apache para que los cambios en la configuración surtan efecto. Elaboración propia.

Ahora se puede acceder a la máquina virtual Metasploitable 3 a través de su dirección IP y el nombre de dominio configurado en el archivo de *hosts*, pero únicamente desde la red DMZ. Para acceder a la máquina virtual desde fuera de la red DMZ, es necesario configurar el reenvío de puertos en el *firewall* pfSense.

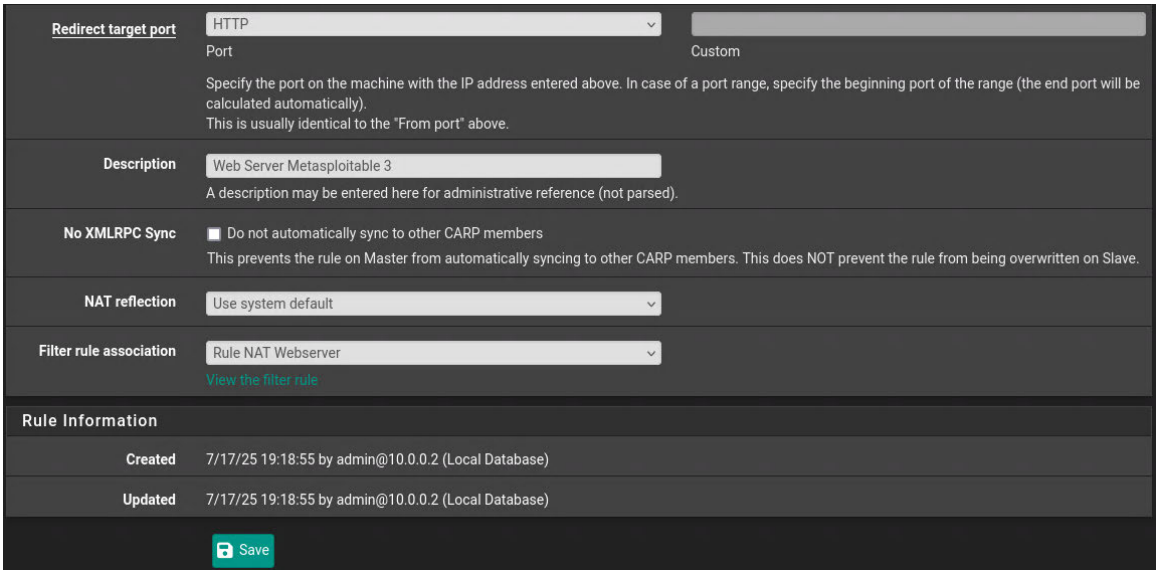
Para ello, se debe ir a la sección *firewall* en el menú principal y seleccionar la opción NAT. En esta sección se pueden ver diferentes pestañas para configurar el NAT. Para este caso específico se debe seleccionar la pestaña *Port Forward* para configurar el reenvío de puertos. Se debe agregar una regla de reenvío de puertos para permitir el acceso a los diferentes servicios que se están ejecutando en la máquina virtual Metasploitable 3. Para agregar una regla de reenvío de puertos, se debe seleccionar la opción *Add*. En la sección de interfaz hay que seleccionar la interfaz WAN, ya que es la interfaz de red externa, de donde se recibe el tráfico. En el protocolo se selecciona la opción TCP/UDP, ya que son los protocolos utilizados por el tráfico web. En la dirección de origen se selecciona deja en *Any*, ya que se quiere permitir el tráfico desde cualquier dirección IP, al igual que el puerto de origen. En la dirección de destino se selecciona la opción WAN *address*, porque es la dirección que está escuchando al exterior de la red. En la sección *Destination port range*, se selecciona la opción puerto HTTP a puerto HTTP. En la sección *Redirect target IP*, se debe ingresar la dirección IP de la máquina virtual Metasploitable 3, en este caso, 10.0.0.2. En la sección *Redirect target port*, se debe ingresar el puerto al que se redirige el tráfico, en este caso, el puerto 80, es decir HTTP. Por último, se debe ingresar una descripción para la regla, en este caso, Web *Server* Metasploitable 3.

Figura 67. Configuración inicial de regla NAT para reenvío HTTP de WAN a DMZ.



Nota. Definición de regla NAT (Port Forward) para mapear tráfico entrante HTTP de WAN al servidor web en DMZ. Elaboración propia.

Figura 68. Configuración avanzada de puertos y redireccionamiento NAT para web Metasploitable 3.



Nota. Pantalla adicional del formulario NAT mostrando parámetros finales antes de aplicar cambios. Elaboración propia.

Una vez configurada la regla de reenvío de puertos, se debe guardar la regla y luego aplicar los cambios. Ahora, se puede acceder al servidor web desde fuera de la red, ingresando la dirección IP pública del *firewall* en el navegador web. Sin embargo, para poder acceder al servidor web desde fuera de la red, se debe configurar el DNS para que resuelva el dominio *sini58.example.com* a la dirección IP del *firewall*. En este caso, no se tiene acceso al servidor DNS de la universidad, por lo que se debe editar el archivo *hosts* en Windows. Para ello, se debe abrir el archivo *hosts* que se encuentra en la ruta *C:\Windows\System32\drivers\etc\hosts*. Se debe abrir el archivo con un editor de texto como Notepad o Notepad++ y agregar la siguiente línea al final del archivo:

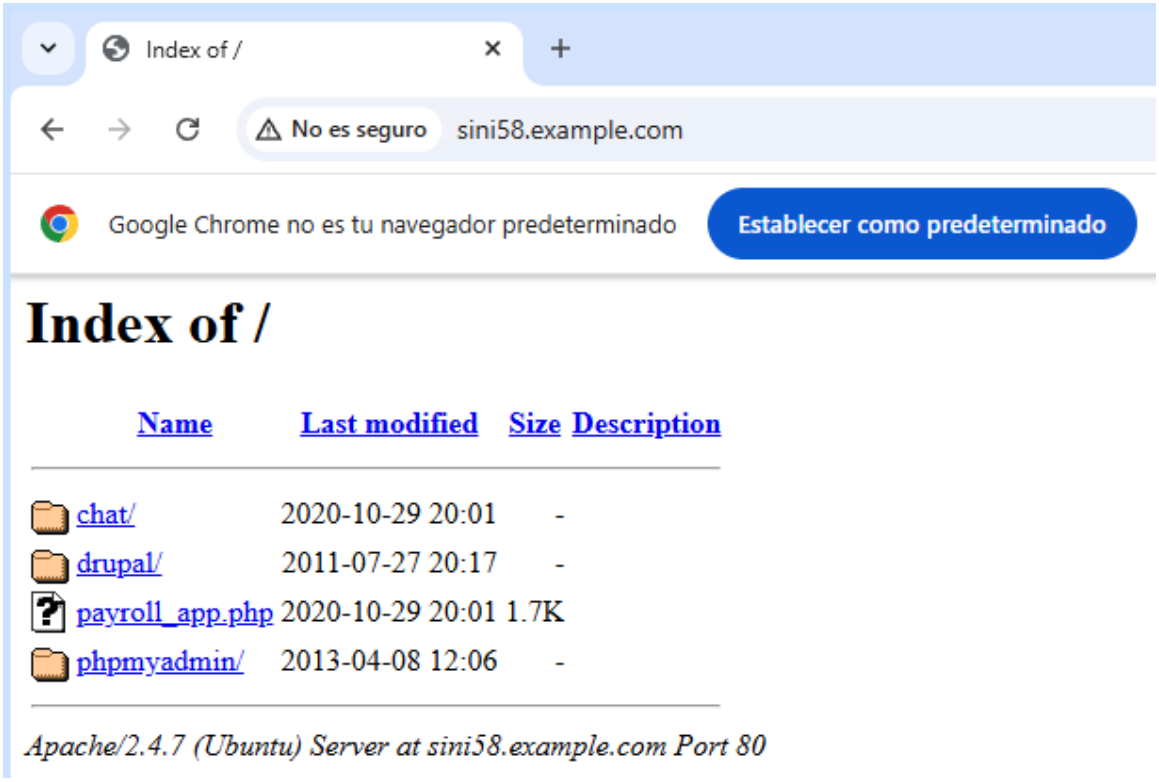
Cuadro 7. Entrada en el archivo *hosts* de Windows.

```
192.168.74.60 sini58.example.com
```

Nota. Asignación estática en archivo *hosts* de Windows vinculando nombre de dominio de Metasploitable 3 a dirección IP pública del *firewall* para acceso desde máquina atacante. Elaboración propia.

Esto le dice a Windows que cuando se intente acceder al dominio *sini58.example.com*, se debe resolver a la dirección IP 192.168.74.60, que es la dirección IP del *firewall*, que está expuesta en la red de la universidad. Luego, se debe guardar el archivo y salir del editor. Ahora, se puede abrir el navegador web y acceder al dominio *sini58.example.com*.

Figura 69. Acceso al sitio web desde el navegador en Windows.

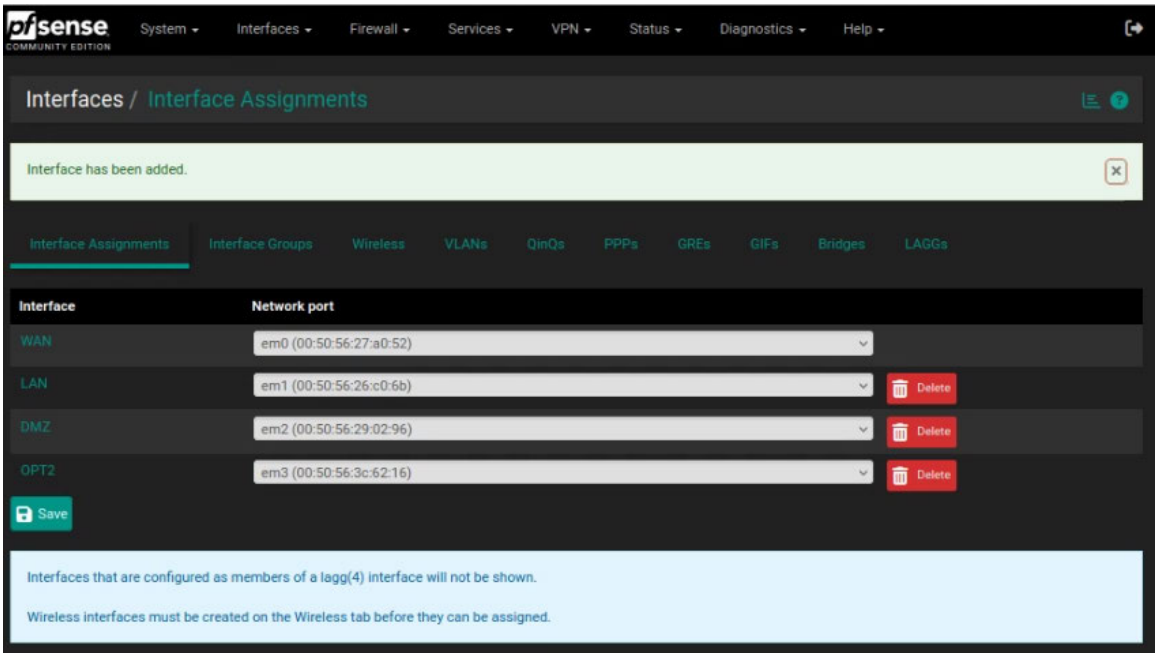


Nota. Validación desde un *host* Windows externo usando entrada en archivo *hosts* para resolver dominio hacia IP pública del *firewall*. Elaboración propia.

7.3. Configuración de la red del centro de operaciones de seguridad y monitoreo (SOC & MOC)

En el siguiente paso, se procede a configurar la interfaz de red del centro de operaciones de seguridad y monitoreo (SOC & MOC). Para ello, se debe ir a la sección *Interfaces* en el menú principal y seleccionar la opción *Assignments*. En esta sección se pueden ver las interfaces de red que se han configurado hasta el momento, que son la interfaz de red WAN, LAN y DMZ. Para agregar una nueva interfaz de red, primero se debe seleccionar el adaptador de red adecuado. En este caso, para la red del SOC & MOC, por lo que, se selecciona dicho adaptador, luego se debe seleccionar la opción *Add*.

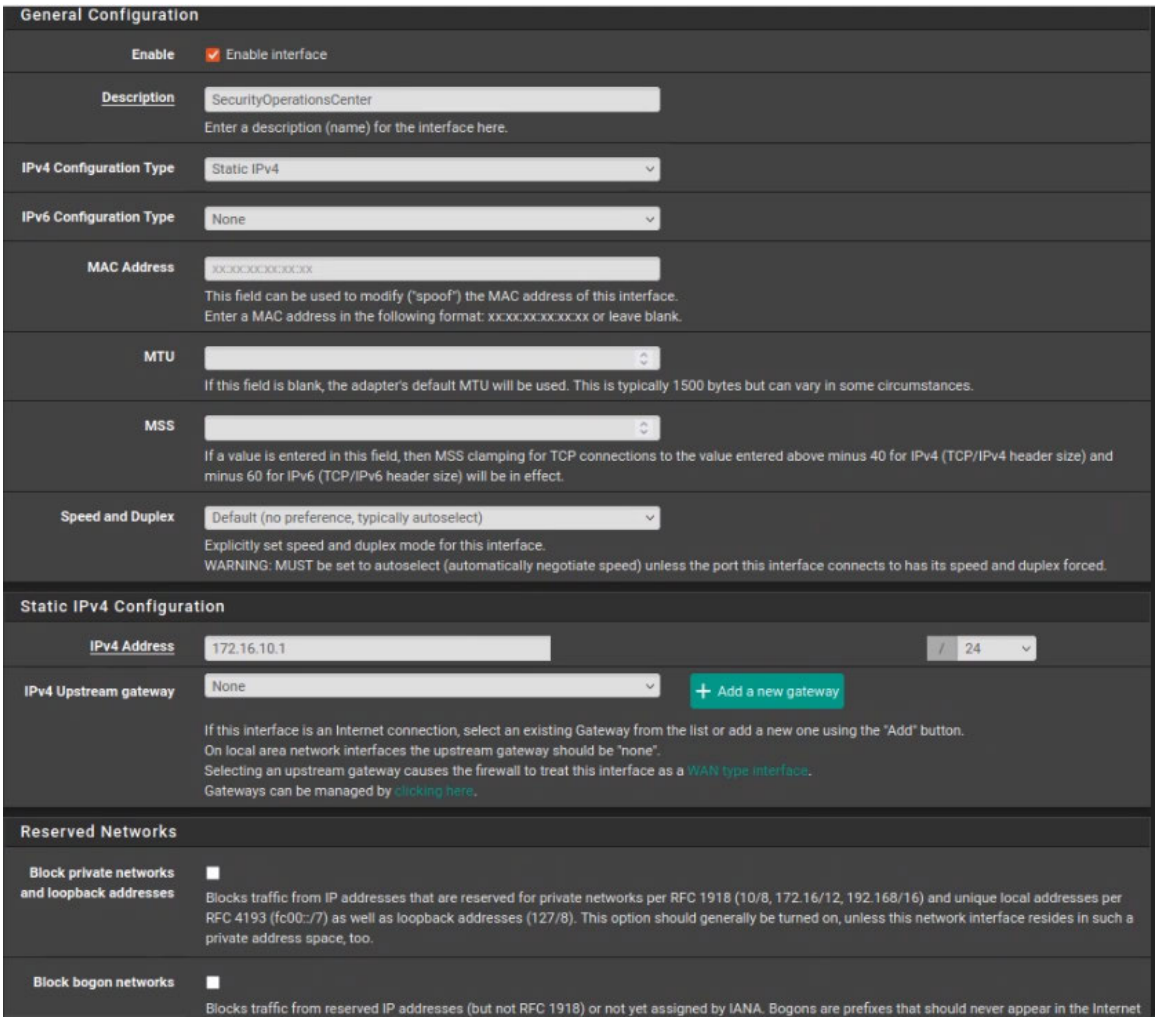
Figura 70. Asignación de nuevo adaptador para la interfaz SOC & MOC en la interfaz web de pfSense.



Nota. Incorporación de nuevo adaptador destinado a la red SOC & MOC preparando segmentación de monitoreo. Elaboración propia.

Se habilita la interfaz y se le asigna un nombre, en este caso, *Security operations center*. Luego se define el tipo de configuración IPv4 como estática. Se debe ingresar la dirección IP de la interfaz de red del SOC & MOC, que es la dirección IP del *gateway* para los dispositivos que están en la red. En este caso, la dirección IP 172.16.10.1/24.

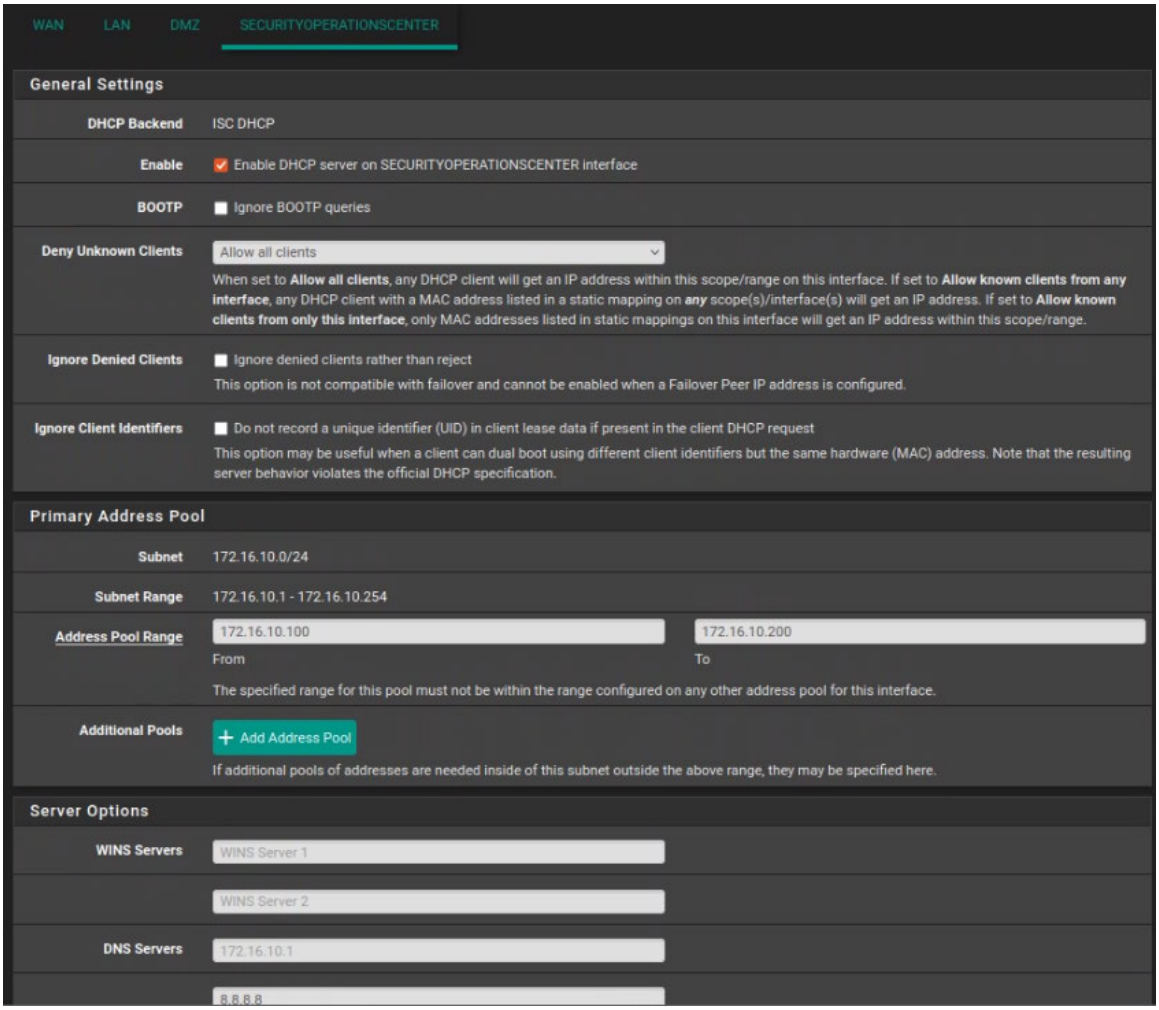
Figura 71. Configuración de la interfaz de red del SOC & MOC en la interfaz web de pfSense.



Nota. Definición de *gateway* interno para red SOC & MOC (172.16.10.1/24) base de servicios de monitoreo. Elaboración propia.

Además, se debe habilitar el servidor DHCP para que los dispositivos que estén en la red puedan obtener una dirección IP automáticamente y su respectivo servidor DNS. Para ello, se debe seleccionar la opción *Services* en el menú principal y luego seleccionar la opción DHCP *Server*. En esta sección se selecciona la interfaz de red del SOC & MOC y se habilita el servidor DHCP. Luego, se debe ingresar el rango de direcciones IP que se asignan a los dispositivos que estén en la red, en este caso, el rango de direcciones IP 172.16.10.100 - 172.16.10.200.

Figura 72. Configuración del servidor DHCP para la red SOC & MOC (rango 172.16.10.100-200).

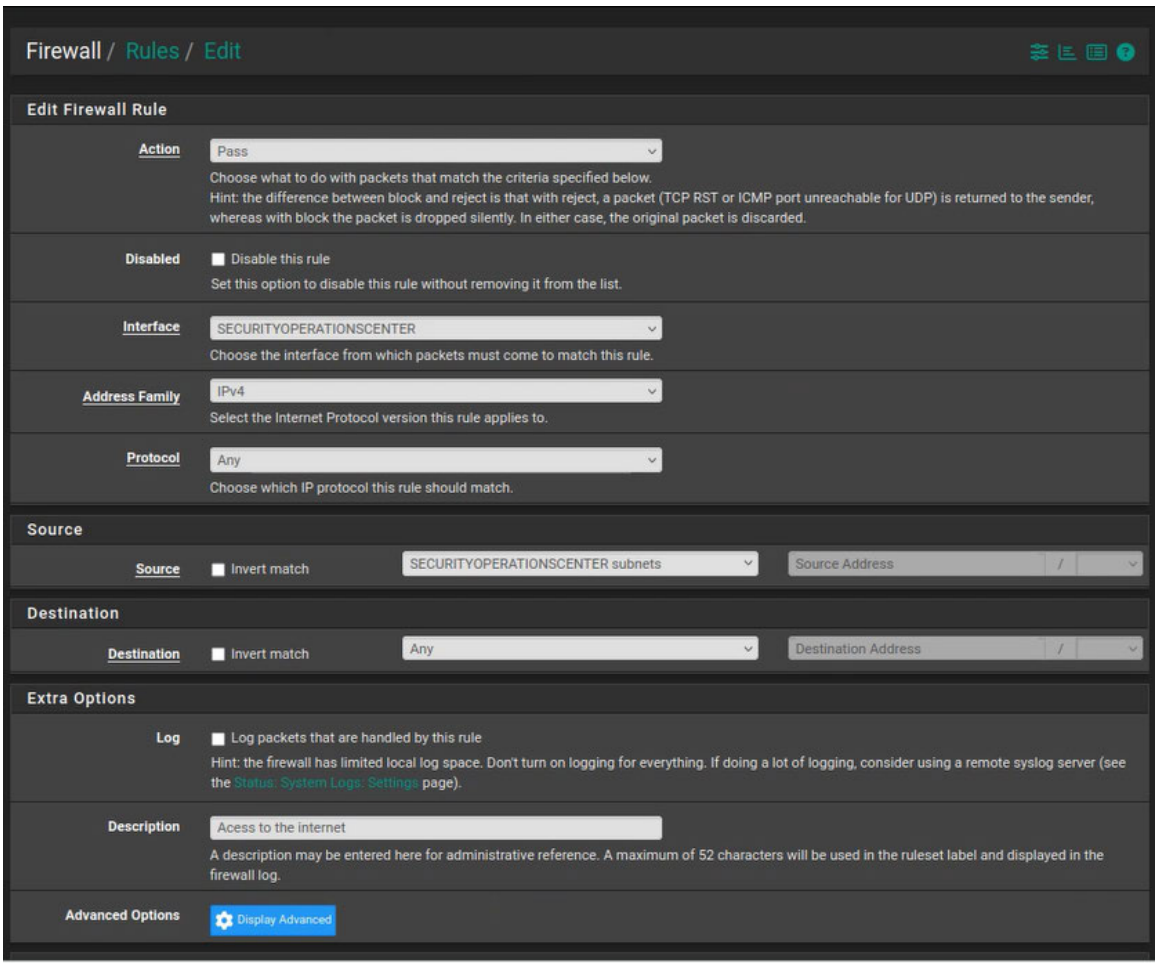


Nota. Ajuste del rango dinámico para dispositivos del SOC & MOC separando espacio para IP estáticas de componentes críticos. Elaboración propia.

Del mismo modo que se hizo con la interfaz DMZ en la sección 7.2.5.1, se deben configurar reglas de *firewall* para permitir el tráfico hacia Internet.

Para ello, se debe ir a la sección *Firewall* en el menú principal y seleccionar la opción *Rules*. En esta sección se selecciona la interfaz del SOC y se pueden ver las reglas de *firewall* que se han configurado hasta el momento, es decir, ninguna. Para agregar una nueva regla, se debe seleccionar la interfaz de red del SOC & MOC en el menú desplegable de interfaces y luego seleccionar la opción *Add*. En la acción se selecciona la opción *Pass*, para así que si se cumple la condición de la regla, se permita el tráfico. En el protocolo se selecciona la opción *Any*, ya que se quiere permitir cualquier tipo de tráfico. En la dirección de origen se selecciona la opción *SOC subnets*, que es para permitir el tráfico desde la red. En la dirección de destino se selecciona la opción *Any*, que es para permitir el tráfico hacia cualquier dirección IP, ya que se quiere que los dispositivos del SOC & MOC puedan acceder a Internet.

Figura 73. Configuración de regla de *firewall* para la interfaz SOC en pfSense.

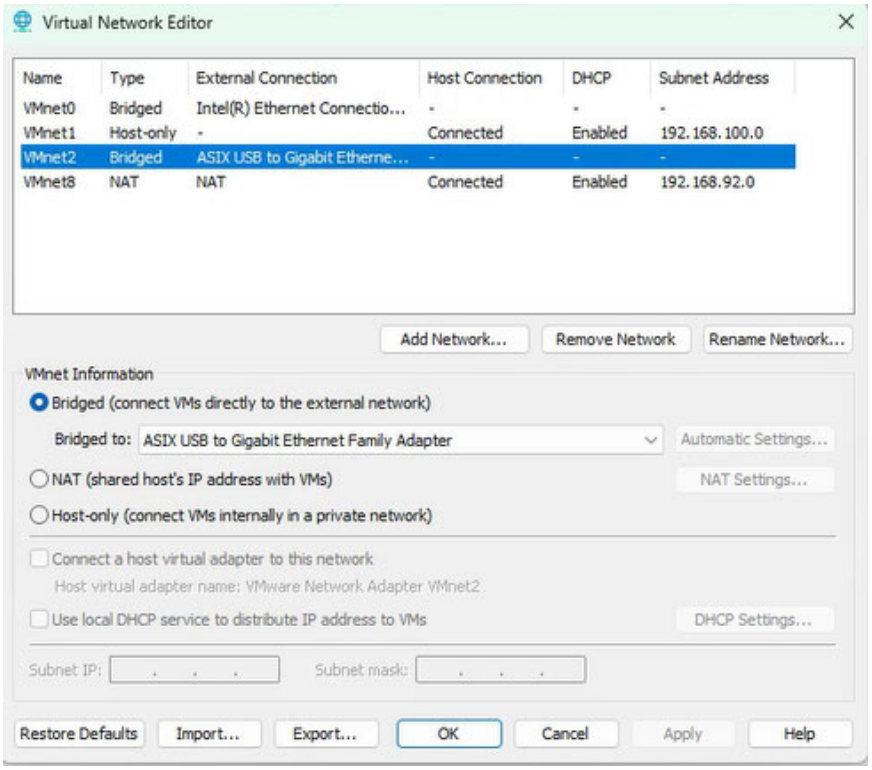


Nota. Regla PASS inicial para otorgar salida controlada desde la red SOC a recursos necesarios. Elaboración propia.

Para confirmar que la configuración de la red del SOC & MOC se ha realizado correctamente, se crea una máquina virtual con el sistema operativo Ubuntu *Desktop* 24.04.2 y conectarla a la red del SOC. En esta máquina virtual se instala Zabbix que sirve para monitorear la red. Más adelante se explica cómo se configura Zabbix para monitorear la red y generar alertas en base al estado de la red. Por ahora solo se crea la máquina virtual y se conecta a la red del SOC para verificar la conectividad.

Para ello, primero se debe ir a la PC4 donde se encuentra la red del SOC & MOC, donde están los dispositivos que se utilizan para monitorear la red. Como se mencionó con anterioridad, se necesita tener el *software VMware Workstation Pro 17*, y tener conectado el adaptador USB-C a Ethernet conectado a la otra PC, y confirmar que Windows lo reconozca. Luego abrir el *virtual network editor* en la pestaña *Edit*. Presionar el botón de *Add Network* y seleccionar la opción *VMnet2*, luego en la opción *bridged* se selecciona el adaptador USB-C a Ethernet que se conecta a la computadora física.

Figura 74. Selección de red virtual para dispositivos de monitoreo del SOC & MOC en *virtual network editor*.



Nota. Creación/selección de red virtual para alojar dispositivos de monitoreo del SOC & MOC aislados de usuarios finales. Elaboración propia.

Zabbix, para un despliegue pequeño como el que se está realizando, requiere de un servidor con los siguientes requerimientos.

Figura 75. Requerimientos de hardware para Zabbix.

Installation size	Monitored metrics[1]	CPU/vCPU cores	Memory (GiB)	Database	Amazon EC2[2]
Small	1 000	2	8	MySQL Server, Percona Server, MariaDB Server, PostgreSQL	m6i.large/m6g.large

Nota. Tabla de referencia de recursos mínimos recomendados para planificar capacidad del servidor de monitoreo Zabbix. Elaboración propia.

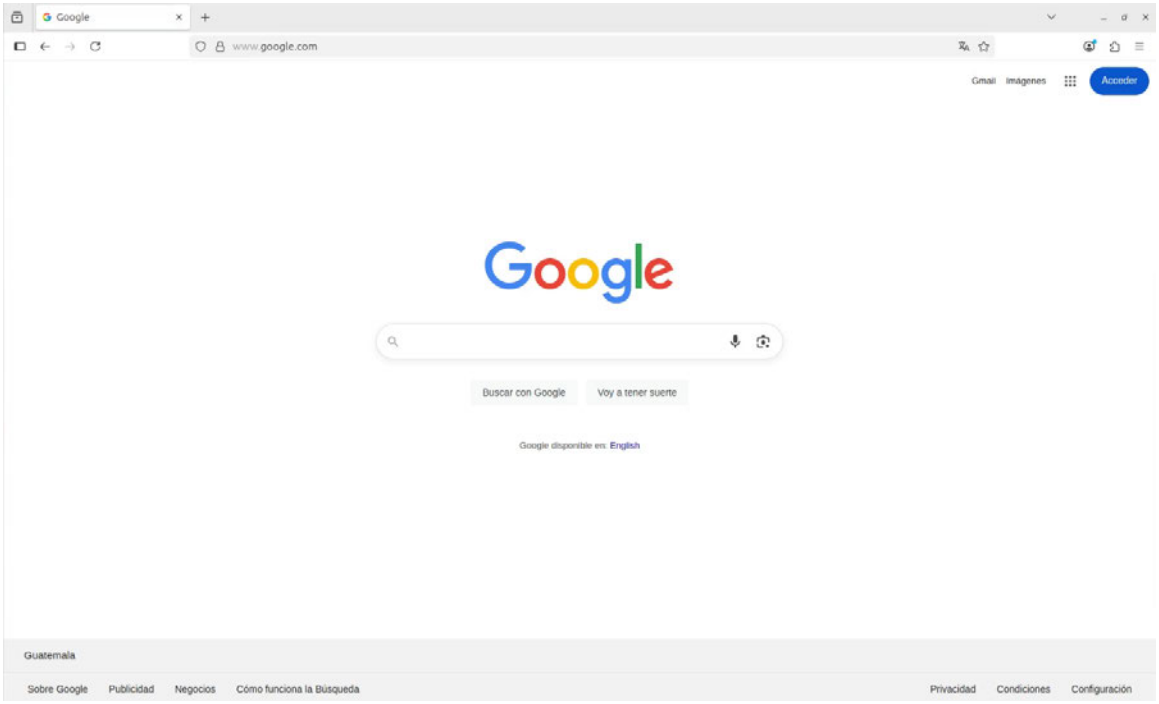
Se necesita crear una nueva máquina virtual con los siguientes requerimientos:

- Sistema operativo: Ubuntu *Desktop* 24.04.2
- Memoria: 8GB
- Espacio de disco duro: 200 GB

- Procesadores/Núcleos por procesador: 1/2
- Adaptador de red: *custom*, VMnet2 (que es la red SOC que se configuró en el *virtual network editor*). Se debe seleccionar la opción *Generate a new MAC address* para evitar conflictos de red.

Para comprobar que la configuración de la red del SOC & MOC se ha realizado correctamente, se puede abrir el navegador web y se debería tener acceso a Internet.

Figura 76. Acceso a Internet desde la máquina virtual en la red del SOC & MOC.



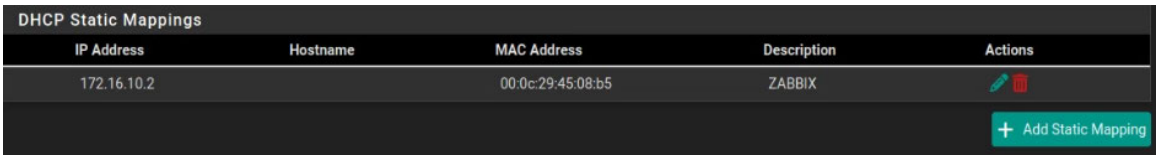
Nota. Cliente SOC accediendo a google.com para verificar conectividad. Elaboración propia.

Ahora mismo, la IP de la máquina virtual es dinámica, ya que se configuró el servidor DHCP para que los dispositivos que estén en la red del SOC puedan obtener una dirección IP automáticamente. Al ser un servidor de monitoreo como Zabbix, lo más sencillo y práctico es que la dirección IP sea estática, por lo que se debe configurar una dirección IP estática en pfSense.

Para ello se debe ir a la sección *Services* en el menú principal y luego seleccionar la opción DHCP *Server*. Luego, en la parte inferior de la página se puede ver la opción *DHCP Static Mappings*, que es la opción que permite asignar direcciones IP estáticas a los dispositivos que estén en la red del SOC. Para ello, se debe seleccionar la opción *Add* y luego ingresar la dirección MAC de la máquina virtual, en este caso, la dirección MAC que se generó en la configuración de la máquina virtual. Luego, se debe ingresar la dirección IP que se asigna

al servidor de monitoreo, en este caso, 172.16.10.2. Notar que esta IP debe estar fuera del rango de direcciones IP que se configuraron en el servidor DHCP.

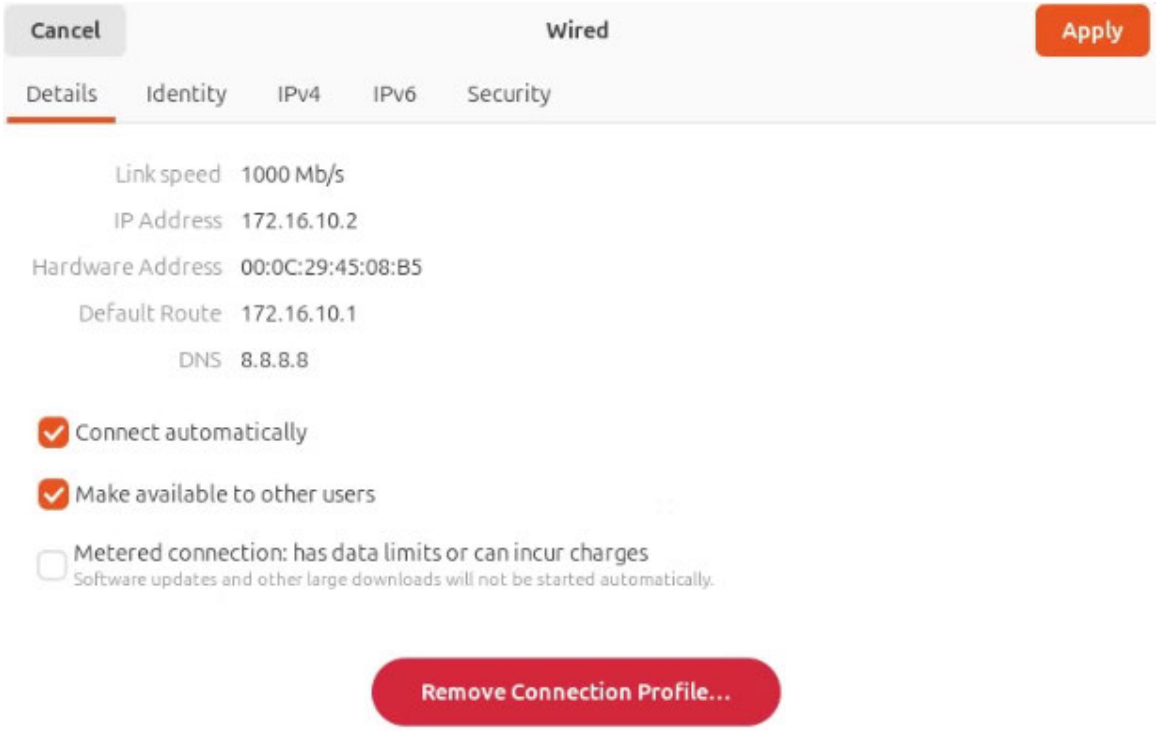
Figura 77. Definición de mapeo estático DHCP para servidor Zabbix (IP 172.16.10.2) en la interfaz web de pfSense.



Nota. Definición de mapeo estático para servidor SOC asegurando IP fija (172.16.10.2) necesaria para agentes y monitoreo. Elaboración propia.

Ahora, se debe reiniciar la máquina virtual para que se aplique la nueva configuración de red. Una vez reiniciada la máquina virtual, se puede comprobar que la dirección IP del servidor de monitoreo es 172.16.10.2.

Figura 78. Configuración de red en Ubuntu *Desktop* en la red del SOC & MOC.



Nota. Confirmación tras reinicio de que el servidor SOC & MOC adoptó la IP estática configurada para operaciones de monitoreo. Elaboración propia.

7.3.1. Instalación y configuración de Kali Linux para OpenVAS

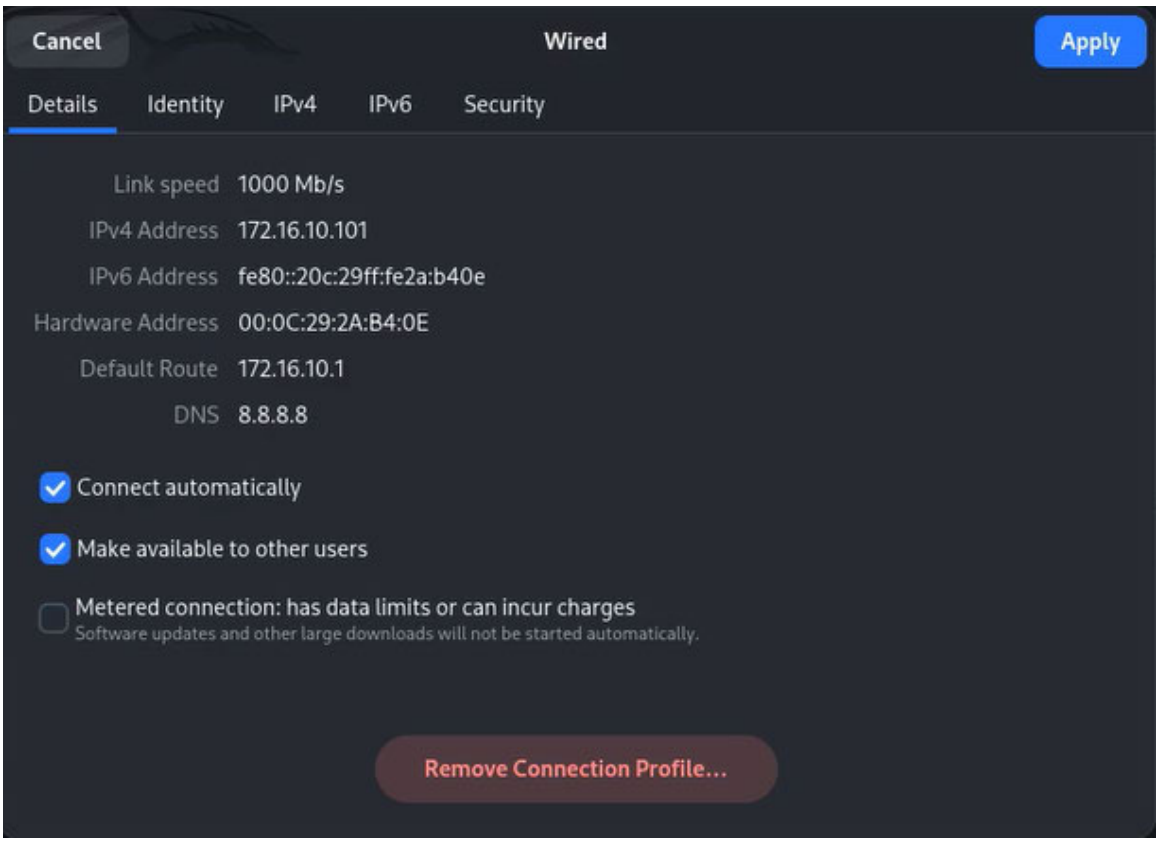
Para la instalación de OpenVAS se utiliza la máquina virtual Kali Linux, ya que este sistema operativo ofrece una instalación simplificada de OpenVAS. Para instalar Kali Linux, se debe crear una nueva máquina virtual que necesita los siguientes requerimientos:

- Sistema operativo: Kali Linux 2025.2
- Memoria: 8GB
- Espacio de disco duro: 100 GB
- Procesadores/Núcleos por procesador: 1/8
- Adaptador de red: *custom*, VMnet2 (que es la red LAN que se configuró en el *virtual network editor*). Se debe seleccionar la opción *Generate a new MAC address* para evitar conflictos de red.

Para la instalación se pueden seguir los pasos de la sección 7.2.4.2.

Para verificar que la instalación de Kali Linux se realizó correctamente, se puede abrir la configuración de red y comprobar que la máquina virtual recibió una dirección IP y tiene conexión a Internet.

Figura 79. Configuración de red en Kali Linux en la red SOC.



Nota. Kali para OpenVAS obteniendo IP en red SOC confirmando disponibilidad de conectividad para escaneos internos. Elaboración propia.

Se puede observar que la máquina virtual tiene una dirección IP en la red SOC y tiene conexión a Internet.

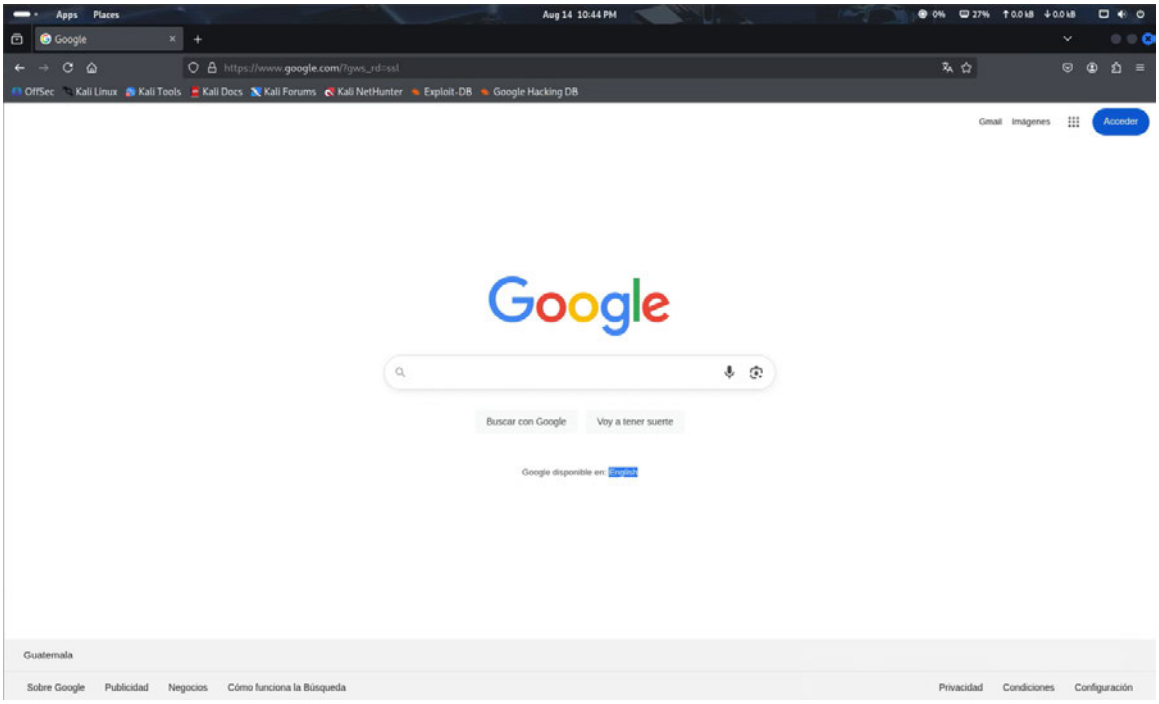


Figura 80. Prueba de conectividad desde Kali Linux OpenVAS en red SOC.

7.4. Instalación y configuración de Kali Linux para pruebas de penetración desde fuera de la red

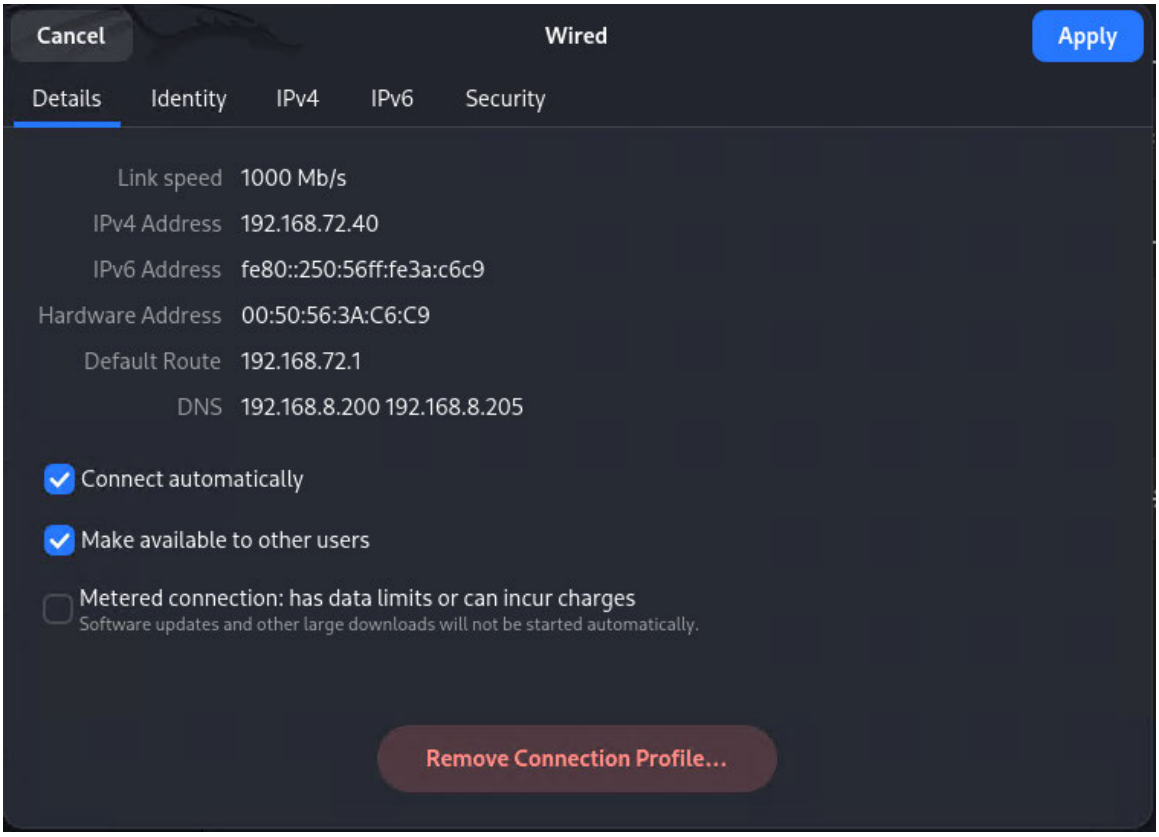
Para instalar Kali Linux, se debe crear una nueva máquina virtual con los siguientes requerimientos:

- Sistema operativo: Kali Linux 2025.2
- Memoria: 24GB
- Espacio de disco duro: 100 GB
- Procesadores/Núcleos por procesador: 1/16
- Adaptador de red: *custom*, VMnet0 (que es la red externa que se configuró en el *virtual network editor*). Se debe seleccionar la opción *Generate a new MAC address* para evitar conflictos de red.

Para la instalación se pueden seguir los pasos de la sección 7.2.4.2.

Para verificar que la instalación de Kali Linux se realizó correctamente, se puede abrir la configuración de red y comprobar que la máquina virtual recibió una dirección IP y tiene conexión a Internet.

Figura 81. Validación de conectividad desde Kali Linux externo (atacante) hacia Internet mediante la red WAN.



Nota. Verificación que la máquina virtual recibió una dirección IP de la red externa, en este caso, la red de la universidad. Elaboración propia.

7.5. Instalación de Pi-hole en la red LAN

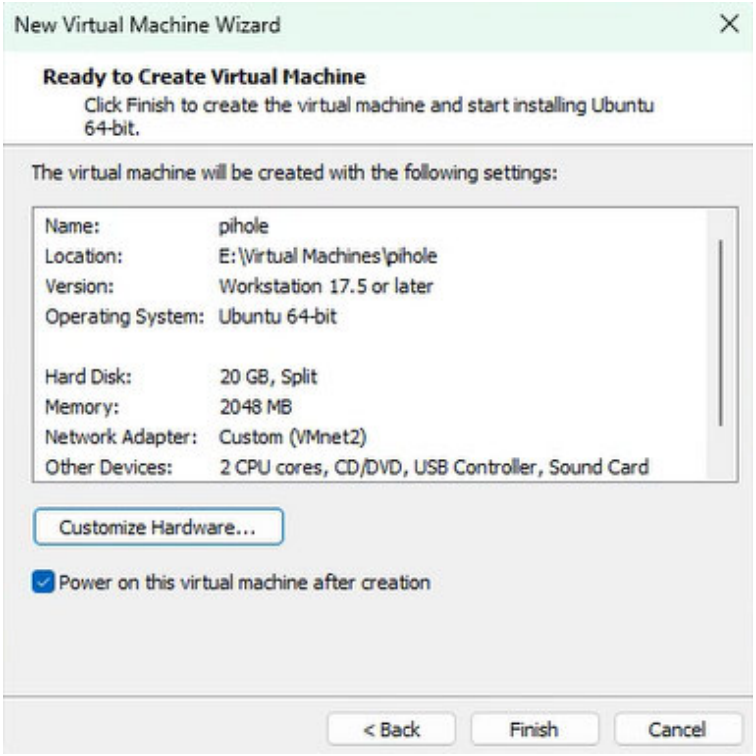
Pi-hole es un *DNS sinkhole* que actúa como un servidor DNS y bloquea anuncios y rastreadores a nivel de red, esto sin necesidad de instalar software adicional en cada dispositivo. En este caso, se instala Pi-hole en una máquina virtual que está conectada a la red LAN, ya que se quiere bloquear anuncios y rastreadores en todos los usuarios conectados a la red LAN.

Para instalar Pi-hole, se debe crear una nueva máquina virtual con los siguientes requerimientos:

- Sistema operativo: Ubuntu *Server* 24.04.2
- Memoria: 2GB
- Espacio de disco duro: 20 GB
- Procesadores/Núcleos por procesador: 1/2

- Adaptador de red: *custom*, VMnet2 (que es la red LAN que se configuró en el *virtual network editor*). Se debe seleccionar la opción *Generate a new MAC address* para evitar conflictos de red.

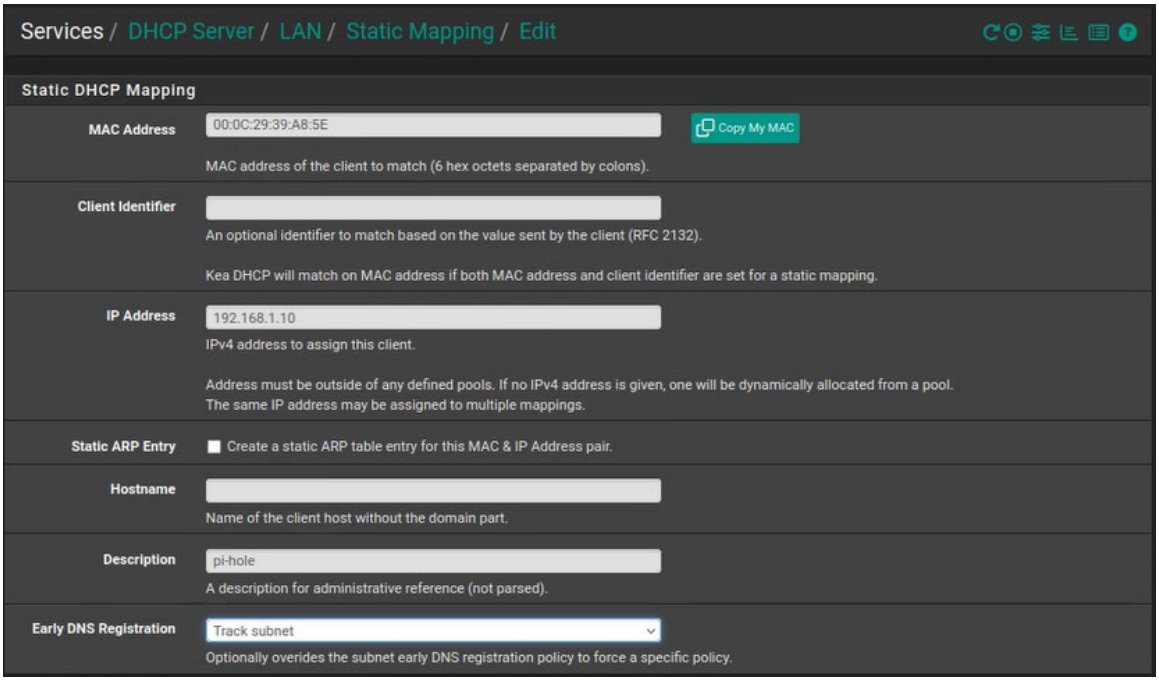
Figura 82. Configuración de la máquina virtual para Pi-hole.



Nota. Parámetros de hardware y red asignados a la VM que actúa como servidor DNS y bloqueador de anuncios (Pi-hole). Elaboración propia.

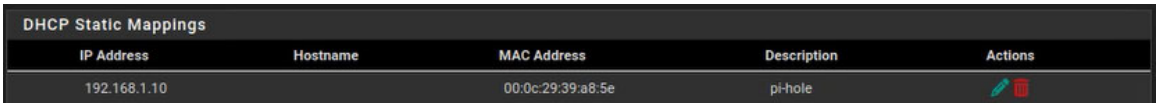
Antes de iniciar la máquina virtual, se debe configurar la IP para que sea estática, ya que se utiliza como servidor DNS para la red LAN y no se quiere que cambie la dirección IP. Para ello, se debe tener la dirección MAC de la máquina virtual; luego se debe ir a la sección *Services* en el menú principal de pfSense y seleccionar DHCP *Server*. Esto se debe realizar en la interfaz de red LAN, por lo que en las pestañas de la parte superior se selecciona la pestaña *LAN*. En la sección *DHCP Static Mappings*, se debe seleccionar la opción *Add* para agregar una nueva asignación estática. En el campo *MAC Address*, se debe ingresar la dirección MAC de la máquina virtual que se creó para Pi-hole. En el campo *IP Address*, se ingresa la dirección IP que se asigna a la máquina virtual, en este caso, 192.168.1.10. La IP debe estar fuera del rango de direcciones IP que se configuraron en el servidor DHCP.

Figura 83. Creación de mapeo DHCP estático para Pi-hole (IP 192.168.1.10) en la interfaz web de pfSense.



Nota. Creación de mapeo DHCP estático para asegurar que Pi-hole utilice siempre la IP 192.168.1.10 en la LAN. Elaboración propia.

Figura 84. Verificación del registro de mapeo estático en DHCP para Pi-hole en la interfaz web de pfSense.

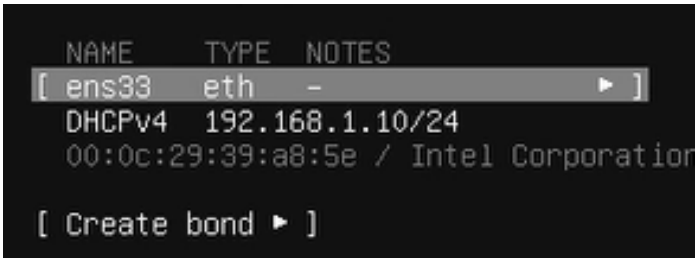


Nota. Confirmación visual del registro de mapeo estático que reserva la IP 192.168.1.10 para Pi-hole. Elaboración propia.

Una vez configurada la asignación estática, se inicia la máquina virtual de Pi-hole.

Al iniciar la máquina virtual, se debe seleccionar la opción *Try or Install Ubuntu Server* y seguir las instrucciones en pantalla para completar la instalación. Durante la instalación, se debe seleccionar el idioma, la distribución del teclado y la zona horaria. Luego, se debe seleccionar la opción de instalar Ubuntu *Server* para la instalación base. En la configuración de red, se debe seleccionar la opción de DHCP para que la máquina virtual obtenga una dirección IP automáticamente del servidor DHCP de pfSense. Sin embargo, como se configuró una asignación estática en el servidor DHCP, la máquina virtual obtiene la dirección IP 192.168.1.10.

Figura 85. Instalación de Ubuntu *Server*.



Nota. Pantalla inicial del instalador de Ubuntu *Server* que sirve de base para desplegar Pi-hole. Elaboración propia.

Se puede observar que la máquina virtual obtiene la dirección IP asignada. En la configuración de *Proxy* se deja en blanco, y para la configuración de almacenamiento se deja por defecto, es decir, se utiliza todo el disco duro asignado a la máquina virtual. Se añade un usuario y se establece una contraseña para el usuario. En la opción de Ubuntu Pro, se selecciona la opción *Skip for Now*. Por último, se debe seleccionar la opción *Install OpenSSH server* para poder acceder a la máquina virtual de forma remota, lo que facilita la administración del servidor Pi-hole.

Una vez completada la instalación, se debe reiniciar la máquina virtual y se inicia sesión con el usuario y la contraseña que se establecieron durante la instalación. Ahora, se debe instalar Pi-hole en la máquina virtual. Para ello, se debe abrir una terminal y ejecutar el siguiente comando:

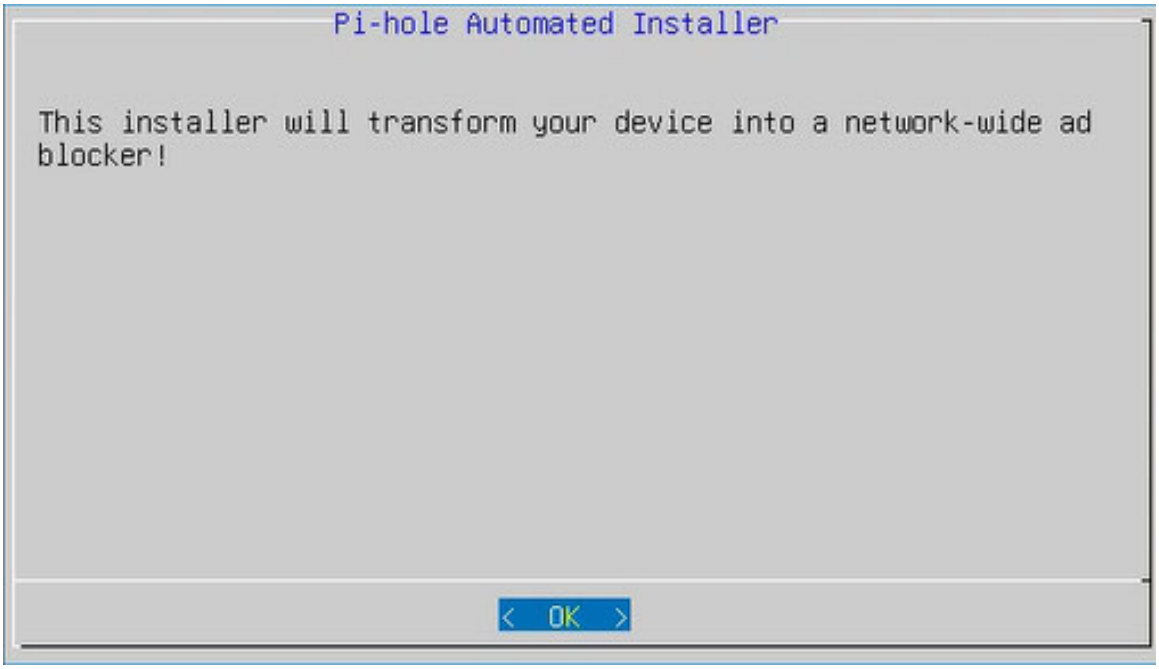
Cuadro 8. Instalación de Pi-hole.

```
1 sudo curl -sSL https://install.pi-hole.net | bash
```

Nota. *Script* oficial de instalación de Pi-hole descargado e inyectado directamente para configurar automáticamente DNS *sinkhole* en servidor LAN. Elaboración propia.

Durante la instalación se muestra un anuncio en la pantalla con el siguiente mensaje: *this installer will transform your device into a network-wide ad blocker!*. Se debe presionar la tecla *Enter* para continuar con la instalación.

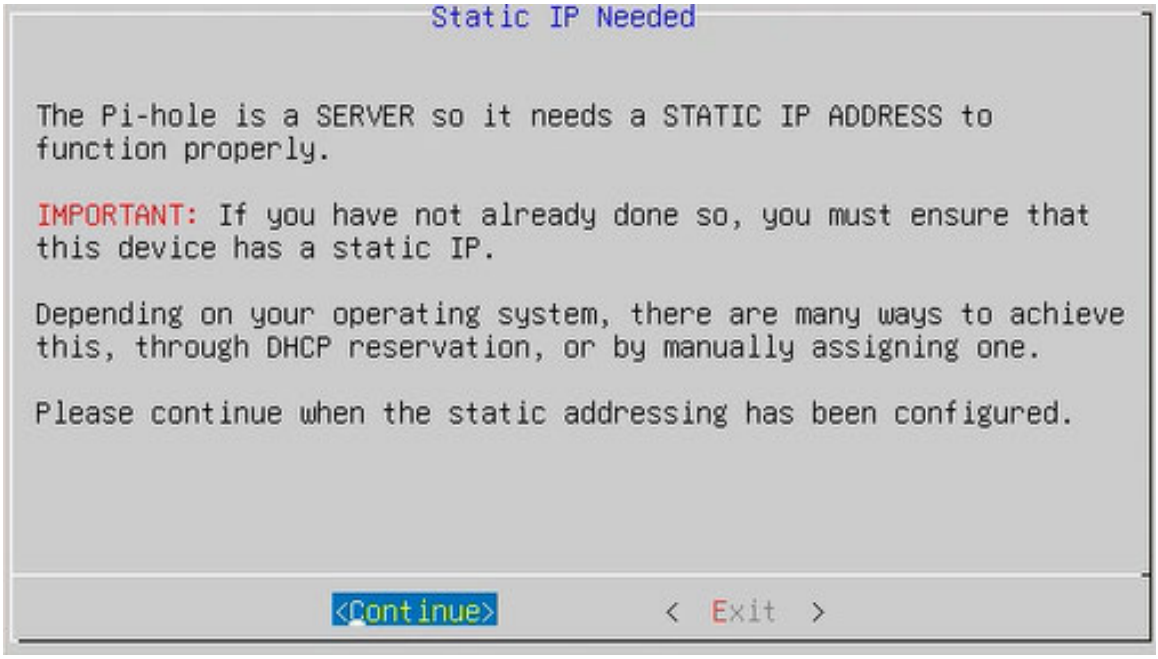
Figura 86. Mensaje inicial del instalador de Pi-hole indicando función de bloqueador de anuncios.



Nota. Mensaje introductorio del instalador indicando conversión del *host* en bloqueador de anuncios a nivel de red. Elaboración propia.

Luego, se muestra un mensaje de advertencia *Pi-hole requires a static IP address to function properly.*. Se debe seleccionar la opción *Continue* para continuar con la instalación, ya que se configuró una asignación estática en el servidor DHCP de pfSense.

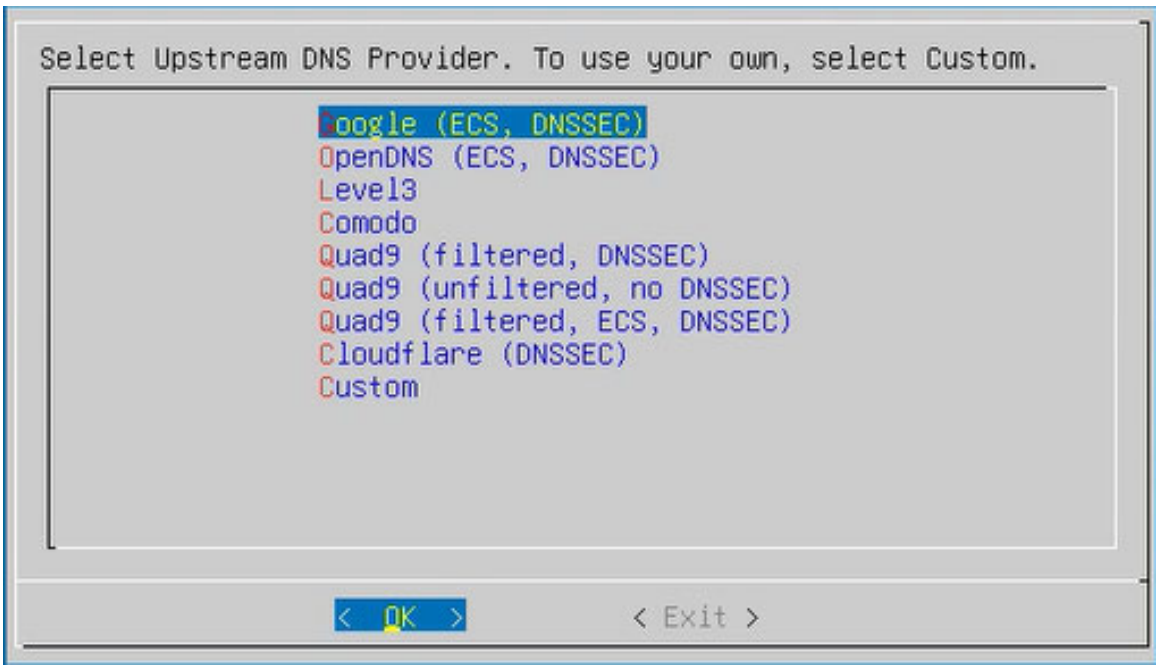
Figura 87. Advertencia IP estática durante la instalación de Pi-hole.



Nota. Aviso que exige dirección IP fija previa; requisito cumplido mediante mapeo estático en pfSense. Elaboración propia.

Luego, se debe seleccionar un proveedor de DNS para que Pi-hole pueda resolver las consultas DNS. En este caso, se puede seleccionar la opción *Google (ECS, DNSSEC)* o cualquier otro proveedor de DNS que se desee utilizar.

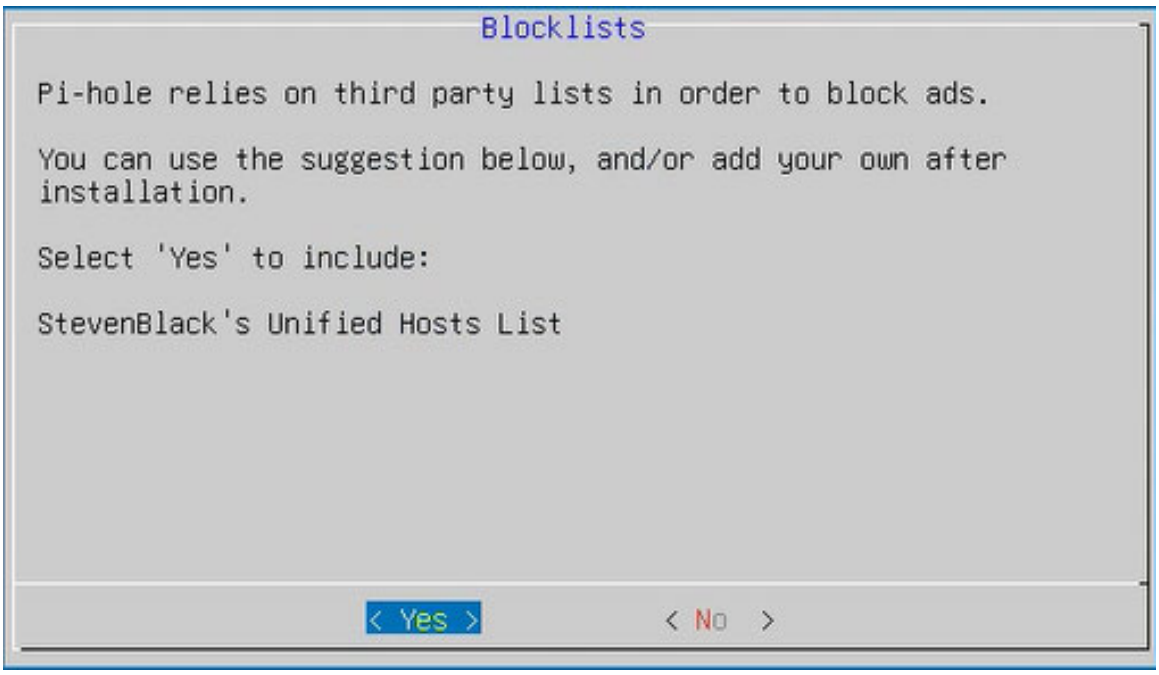
Figura 88. Selección del proveedor de DNS durante la instalación de Pi-hole.



Nota. Elección del servidor DNS upstream (Google ECS DNSSEC en este ejemplo) que resuelve consultas no bloqueadas. Elaboración propia.

Después, Pi-hole informa que para que se bloqueen los anuncios, este depende de *blocklists* de terceros, que son listas de dominios conocidos por servir anuncios. Por defecto, Pi-hole recomienda utilizar la lista *StevenBlack's Unified Hosts List*, sin embargo, después de la instalación se pueden agregar otras listas según las preferencias del usuario.

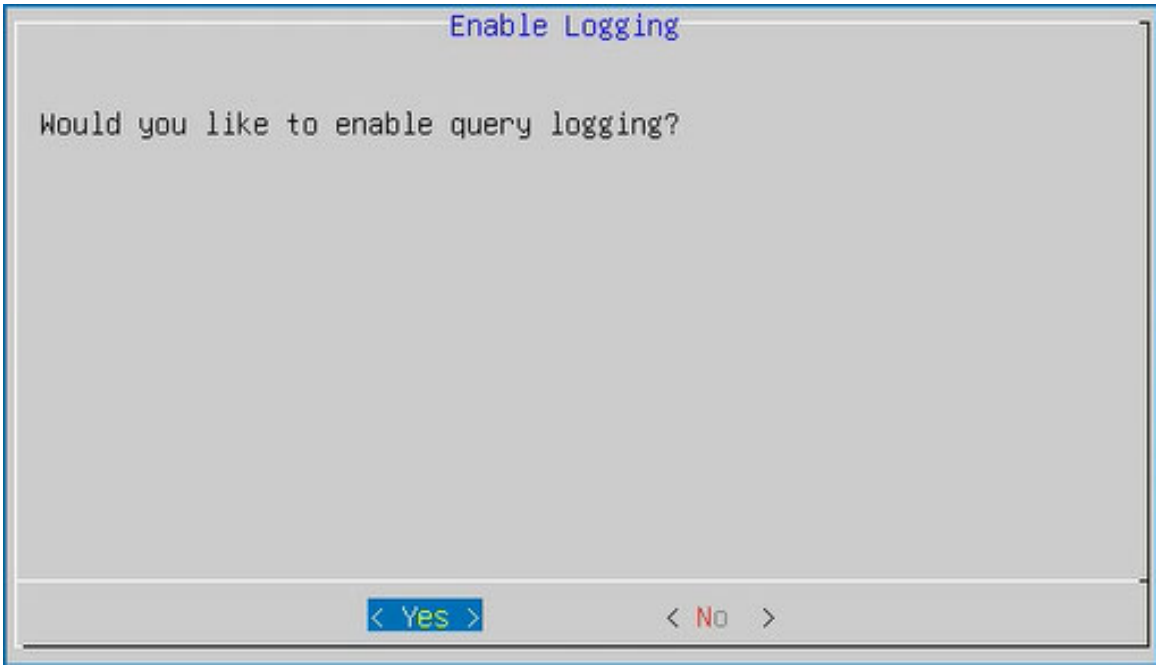
Figura 89. Selección de listas de bloqueo durante la instalación de Pi-hole.



Nota. Selección de la lista base (StevenBlack) que suministra dominios publicitarios iniciales para filtrado. Elaboración propia.

Luego, se pregunta si se quiere habilitar el *query logging*, que es una función que permite registrar todas las consultas DNS que realiza Pi-hole. Se recomienda habilitar esta opción para poder ver qué dominios están siendo bloqueados y cuáles no.

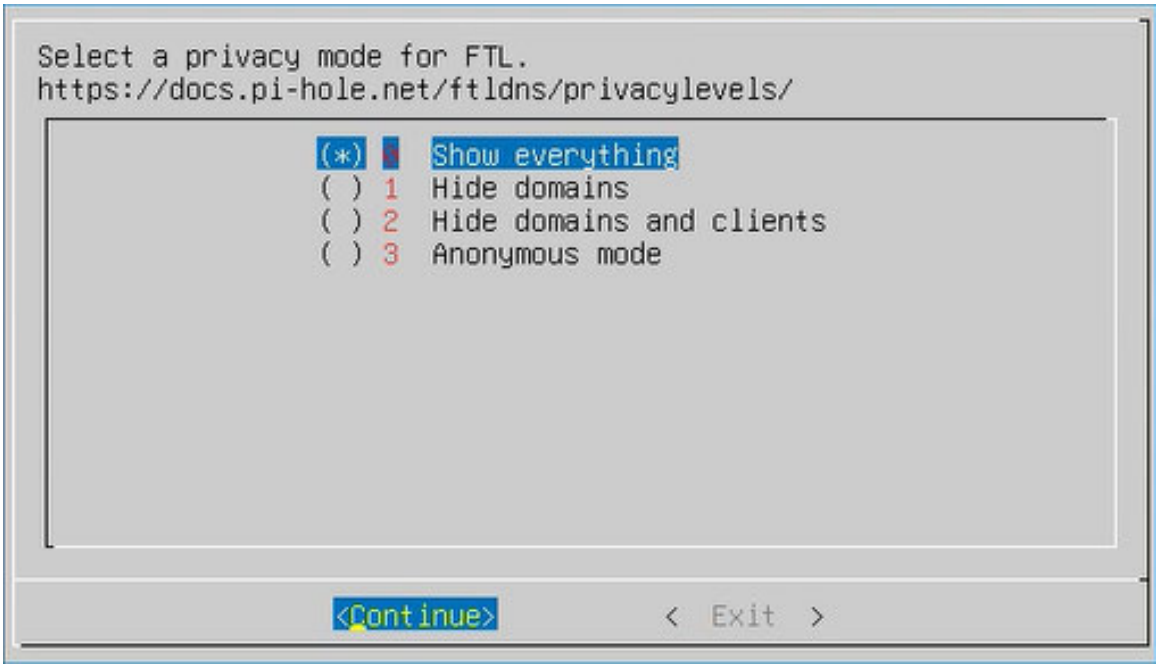
Figura 90. Selección de opciones de registro de consultas durante la instalación de Pi-hole.



Nota. Activación del registro de consultas DNS para análisis y ajuste posterior de políticas de bloqueo. Elaboración propia.

Después, se debe seleccionar el modo de privacidad para FTL (*Fast telemetry logging*), que es el motor de registro de consultas de Pi-hole. Se recomienda seleccionar la opción *Show everything* para que se registren todas las consultas DNS y cualquier información relacionada con las consultas.

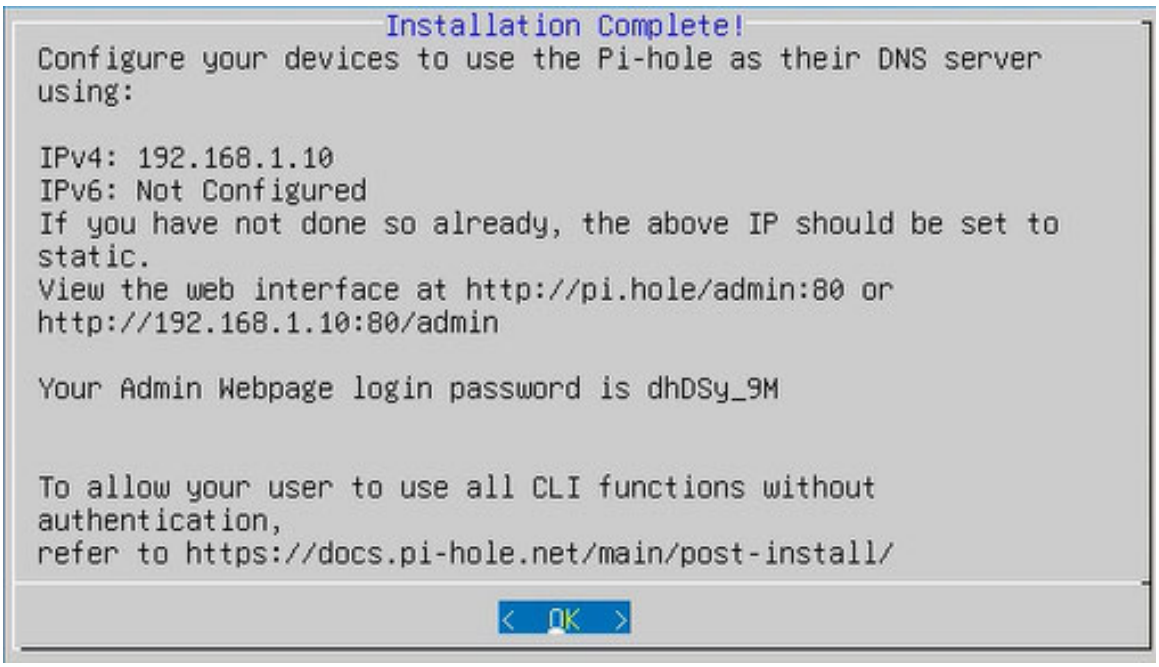
Figura 91. Selección del modo de privacidad para FTL durante la instalación de Pi-hole.



Nota. Modo de privacidad configurado en *Show everything* para máxima visibilidad de eventos de resolución. Elaboración propia.

Una vez completada la instalación, aparece un resumen de la configuración de Pi-hole, incluyendo la dirección IP del servidor, las URLs para acceder a la interfaz web y las credenciales de la misma. Se recomienda anotar esta información para poder acceder al panel de administración más adelante.

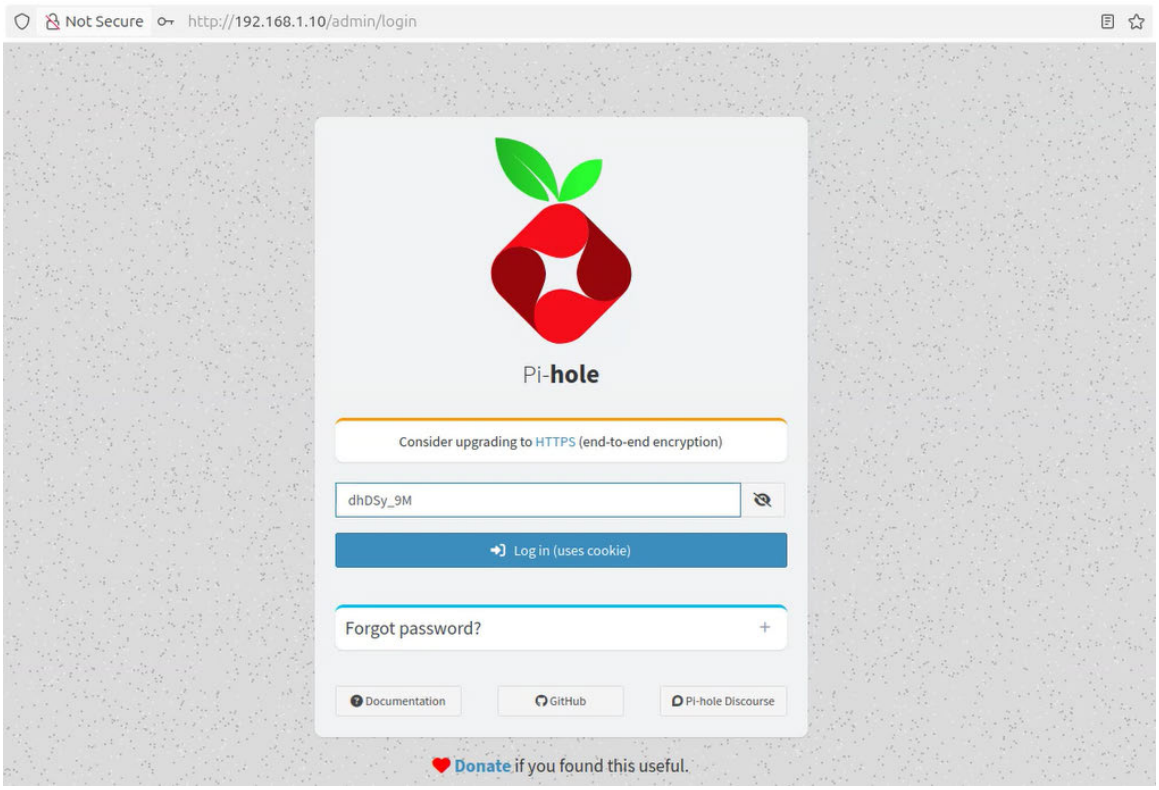
Figura 92. Resumen de la configuración de Pi-hole al finalizar la instalación.



Nota. Pantalla final con credenciales, direcciones de acceso al panel y parámetros clave resultantes de la instalación. Elaboración propia.

Una vez finalizada la instalación, se puede acceder al panel de administración de Pi-hole abriendo un navegador web y accediendo a la URL http://pi.hole/admin:80 o http://192.168.1.10:80/admin. Se ingresa la contraseña que se mencionó en el resumen de la instalación, y se accede al panel de administración de Pi-hole.

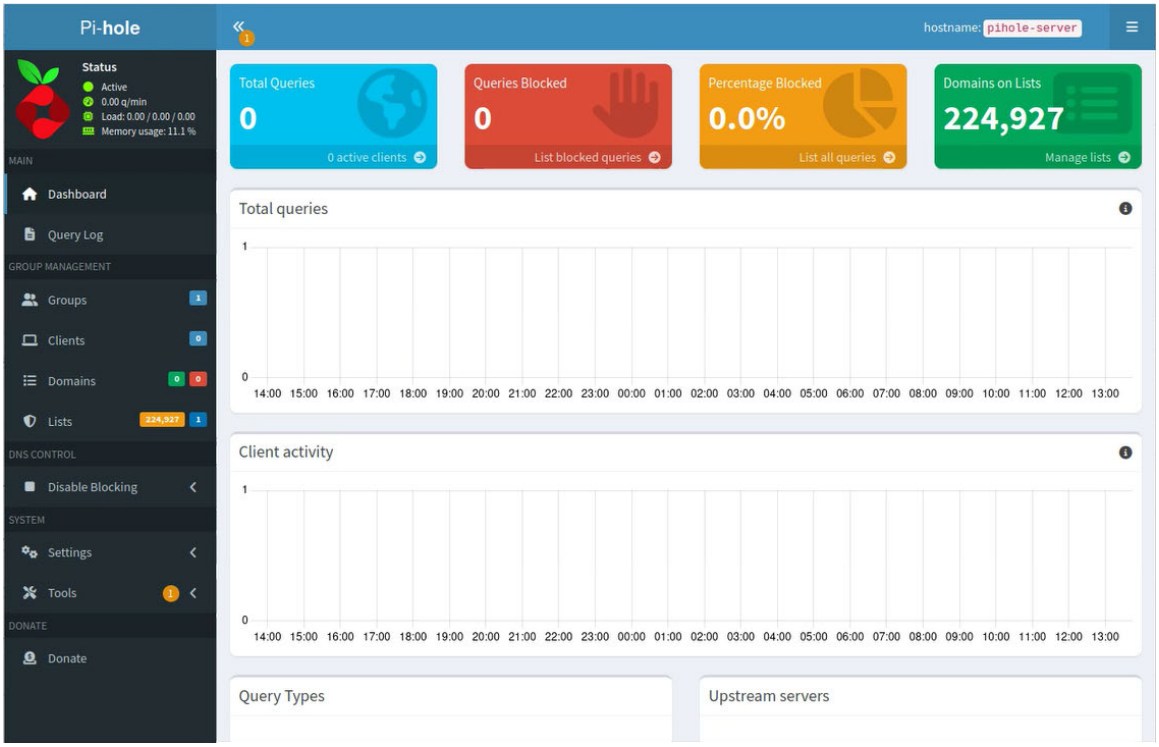
Figura 93. Panel de administración de Pi-hole.



Nota. Vista inicial del *dashboard* de Pi-hole tras autenticación mostrando métricas globales. Elaboración propia.

Así se tiene acceso al panel de administración de Pi-hole, donde se pueden ver las estadísticas de las consultas DNS, los dominios bloqueados, las listas de bloqueo y demás estadísticas.

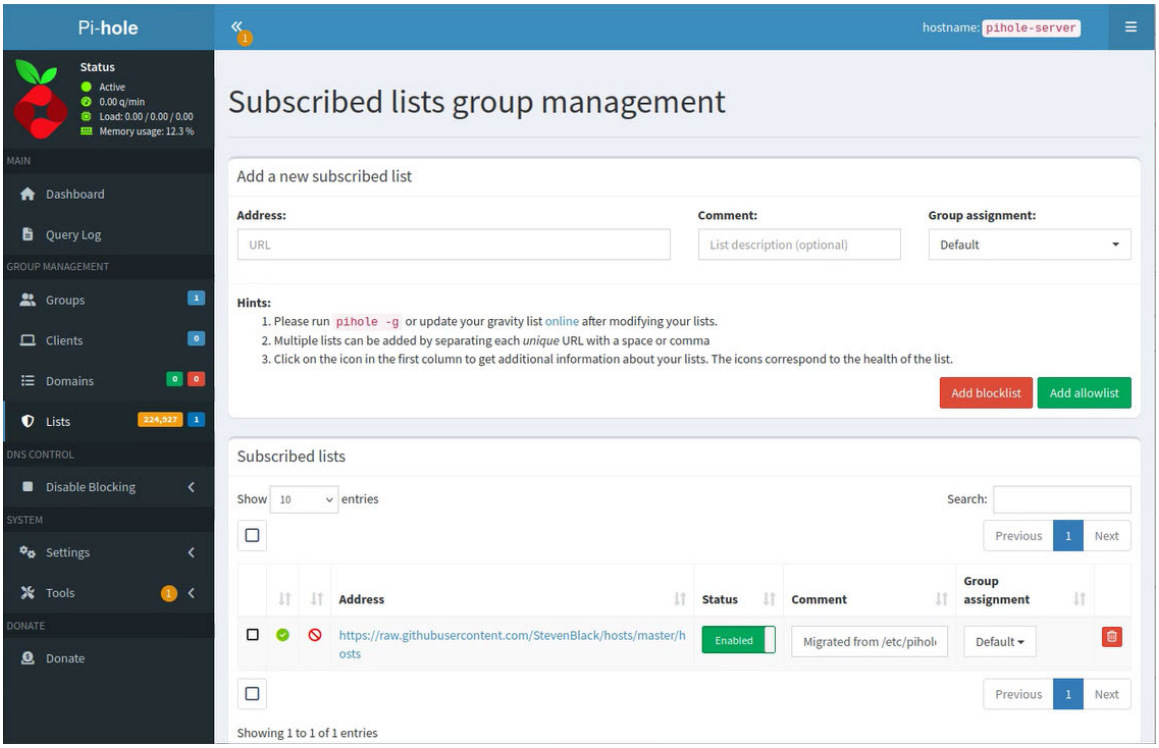
Figura 94. Estadísticas detalladas en el panel de administración de Pi-hole.



Nota. *Dashboard* ampliando estadísticas de consultas totales y bloqueadas para establecer línea base. Elaboración propia.

En el lado izquierdo se tienen las diferentes secciones del panel de administración. Para ver las listas de bloqueo, se debe ir a la sección *Lists*, donde se puede notar que por defecto se tiene la lista de *StevenBlack's Unified Hosts List*.

Figura 95. Listas de bloqueo en el panel de administración de Pi-hole.



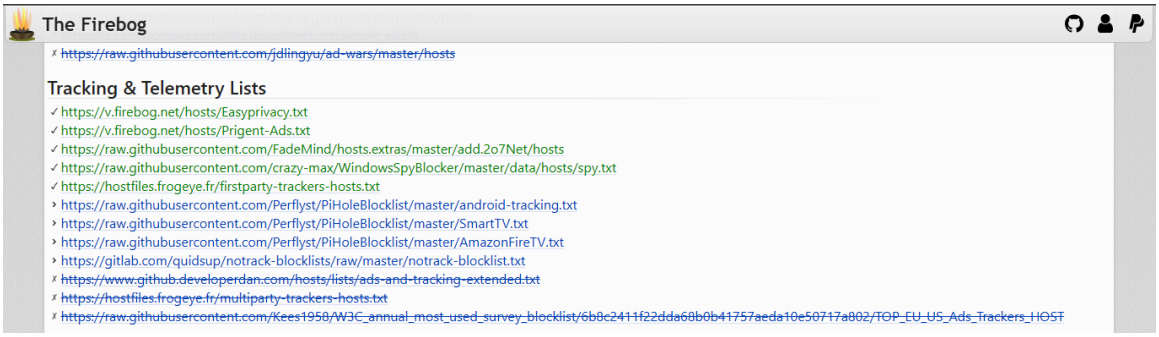
Nota. Sección Lists mostrando la lista StevenBlack activa por defecto para filtrado inicial. Elaboración propia.

7.5.1. Agregar listas de bloqueo adicionales

Las listas de bloqueo son fundamentales para que Pi-hole pueda bloquear anuncios y rastreadores, ya que contienen dominios conocidos por servir anuncios. Existen diferentes tipos de listas de bloqueo, algunas enfocadas en anuncios, otras en rastreadores y otras en malware, otras en contenido para adultos, etc. A continuación, se provee un enlace que contiene un conjunto de listas de bloqueo que se pueden agregar a Pi-hole para mejorar su efectividad: https://firebog.net/.

Por ejemplo, se tiene una sección que contiene listas de bloqueo enfocadas a *Tracking & Telemetry*, es decir, listas que bloquean rastreadores y telemetría.

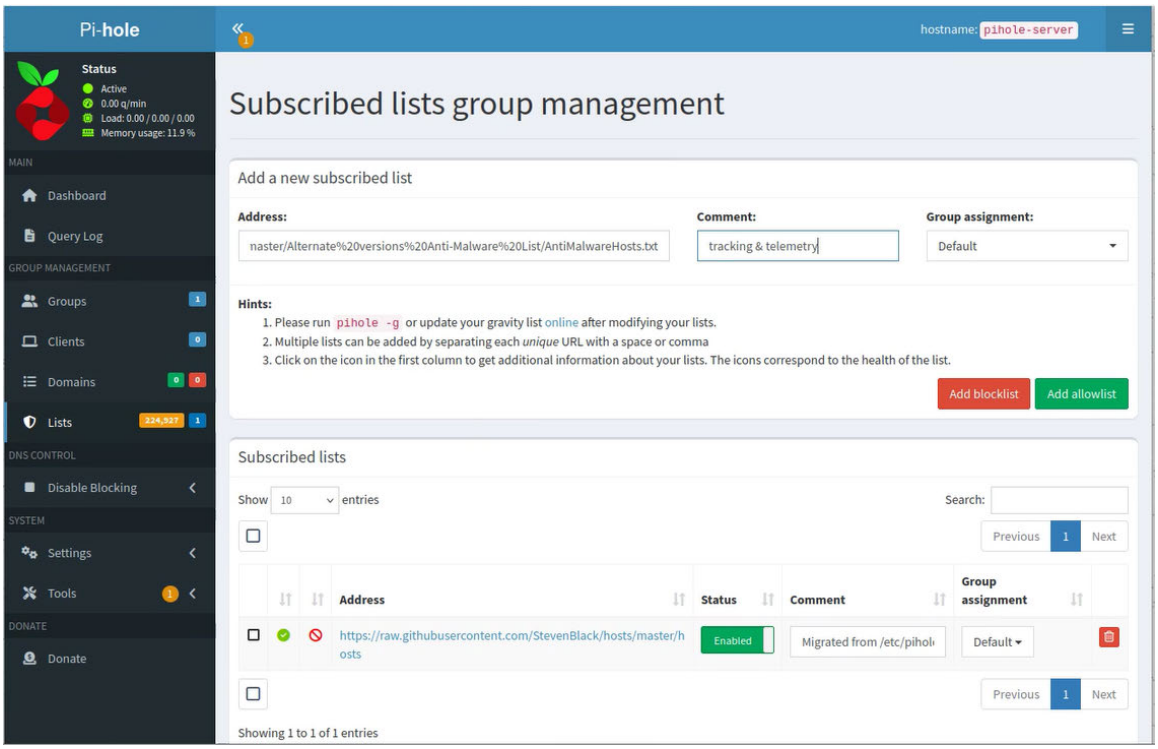
Figura 96. Listas de bloqueo enfocadas a *tracking & telemetry* en el panel de administración de Pi-hole.



Nota. Listado de blocklists orientadas a rastreo y telemetría disponibles para ampliar cobertura de bloqueo. Elaboración propia.

Se puede observar que las listas están en color verde o azul; las listas de color verde están actualizadas y las de color azul están desactualizadas. Para agregar una lista de bloqueo, se debe copiar la URL de la lista, pegarla en la sección *Address*; se puede añadir una descripción en el campo *Comment*, y se debe seleccionar a qué grupo de listas de bloqueo se quiere agregar; en este caso, la opción *Default*. Los grupos son una forma de organizar a los clientes de Pi-hole para que puedan tener diferentes configuraciones de bloqueo. Por ejemplo, se puede crear un grupo para dispositivos móviles, otro para computadoras de escritorio, etc.; o si se quiere tener un grupo con una configuración de bloqueo más agresiva, u otro con una configuración más relajada.

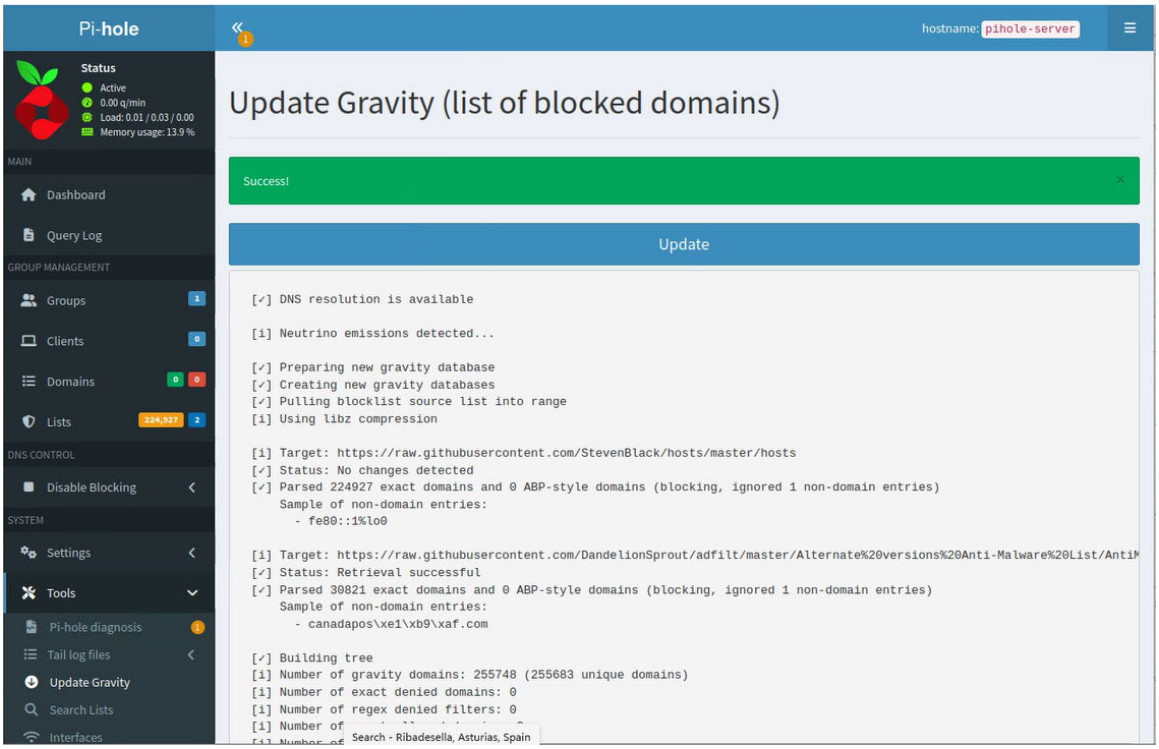
Figura 97. Agregar una lista de bloqueo en el panel de administración de Pi-hole.



Nota. Formulario para incorporar nueva blocklist especificando URL, comentario y grupo destino. Elaboración propia.

Una vez agregada la lista de bloqueo, se debe hacer clic en el botón de *Add blocklist*. Sin embargo, aunque esta se encuentre habilitada, no se aplica hasta que se actualicen las listas de bloqueo. Para ello, se debe ir a la sección *Tools* en el panel de la izquierda, y luego seleccionar la opción *Update Gravity*. Esto actualiza las listas de bloqueo y aplica los cambios realizados.

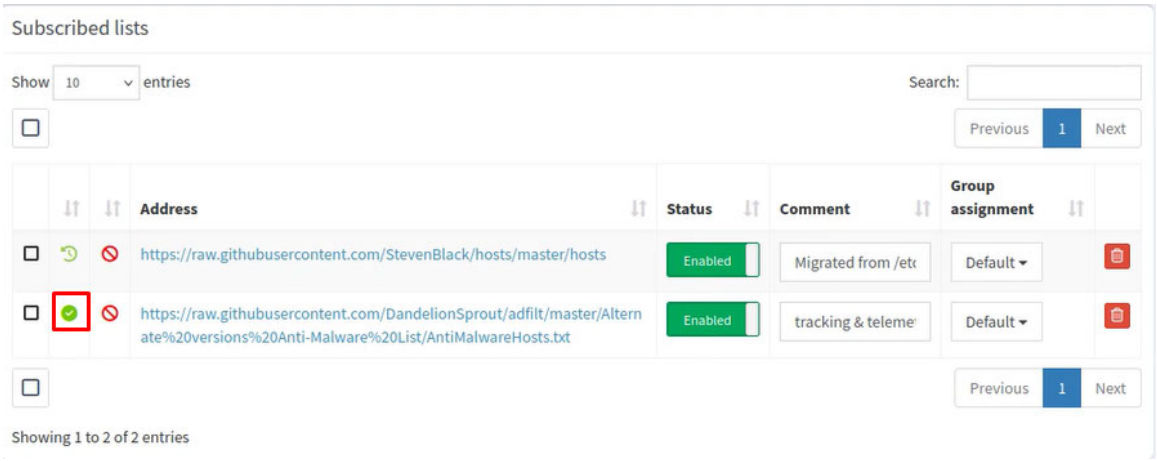
Figura 98. Actualizar las listas de bloqueo en el panel de administración de Pi-hole.



Nota. Ejecución de Update Gravity para descargar y consolidar entradas de nuevas blocklists añadidas. Elaboración propia.

Una vez que se ha actualizado la lista de bloqueo, se puede ver que la lista de bloqueo se ha agregado correctamente y está habilitada.

Figura 99. Lista de bloqueo agregada en el panel de administración de Pi-hole.



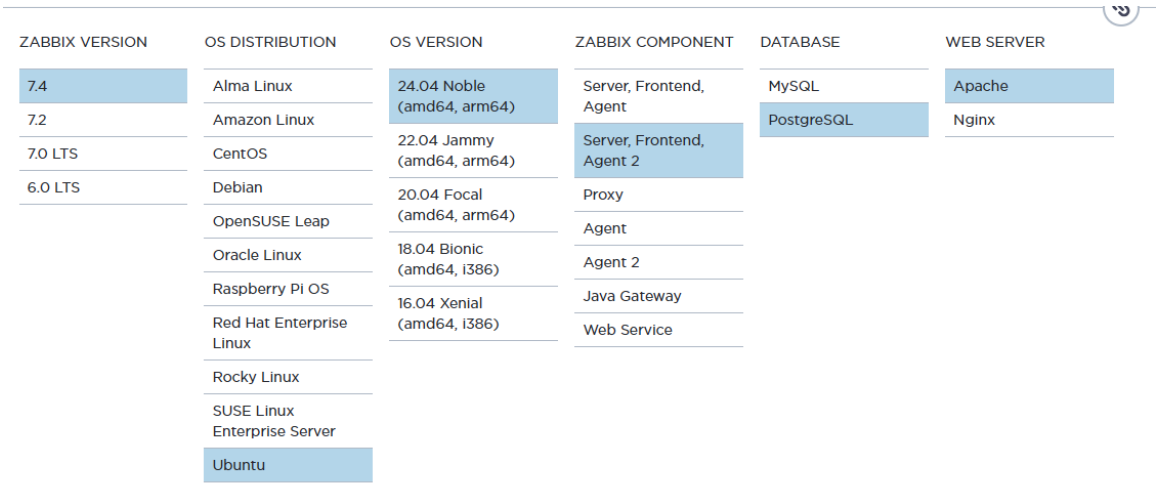
Nota. Verificación de que la nueva blocklist aparece habilitada tras la actualización de Gravity. Elaboración propia.

7.6. Instalación y configuración de Zabbix para el monitoreo de la red

Zabbix es una herramienta de monitoreo de código abierto que permite supervisar el estado de los sistemas, redes y aplicaciones. En este caso, se instala Zabbix en la máquina virtual que está conectada a la red del SOC & MOC para monitorear estado de la red y generar alertas en caso de detectar anomalías. Esta se creó previamente, con los requerimientos necesarios para un despliegue "pequeño"que se puede observar en la Figura 75.

En este caso, se está utilizando el sistema operativo Ubuntu *Desktop* 24.04.2, que es el lanzamiento llamado *Noble*. La instalación de Zabbix se realiza en su versión más reciente, es decir, Zabbix 7.4 utilizando la documentación oficial de Zabbix para esta versión de Ubuntu, que se puede encontrar en el siguiente enlace: https://www.zabbix.com/download?

Figura 100. Tipo de instalación de Zabbix.



Nota. Versión, sistema operativo y arquitectura a utilizar para Zabbix.
Elaboración propia.

Los componentes de Zabbix que se instalan son los siguientes:

- Zabbix *Server*: el servidor principal que almacena los datos de monitoreo y proporciona la interfaz web.
- Zabbix *Frontend*: la interfaz web de Zabbix que permite visualizar los datos de monitoreo y configurar el sistema.
- Zabbix *Agent* 2: una versión más reciente del agente de Zabbix que proporciona una mejor compatibilidad con las últimas versiones de Zabbix. Este agente es para el servidor mismo, y luego se instalan agentes en los dispositivos que se quieren monitorear.

7.6.1. Instalación de Zabbix

Lo primero a realizar es instalar el repositorio de Zabbix. Para ello, se debe abrir una terminal y ejecutar los siguientes comandos:

Cuadro 9. Adición del repositorio de Zabbix en Ubuntu.

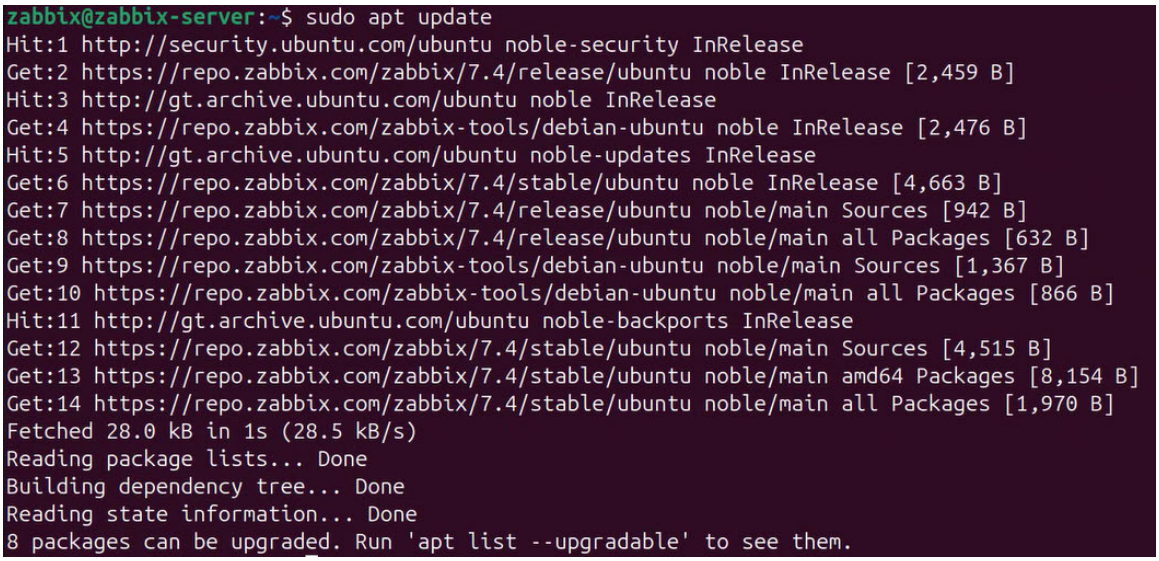
```
1 sudo wget https://repo.zabbix.com/zabbix/7.4/release/ubuntu/pool/main/z/
      zabbix-release/zabbix-release_latest_7.4+ubuntu24.04_all.deb
2 sudo dpkg -i zabbix-release_latest_7.4+ubuntu24.04_all.deb
3 sudo apt update
```

Nota. Descarga e instalación del paquete repositorio oficial de Zabbix 7.4 para Ubuntu 24.04 habilitando acceso a componentes *Server* y *Frontend*.
Elaboración propia.

Figura 101. Descarga del repositorio de Zabbix.

Nota. Descarga del paquete release para añadir el repositorio oficial de Zabbix 7.4 en Ubuntu 24.04. Elaboración propia.

Figura 102. Actualización de índices APT.



Nota. Actualización de índices APT confirmando inclusión del nuevo repositorio antes de instalar componentes. Elaboración propia.

El siguiente es instalar los componentes de Zabbix que se mencionaron anteriormente. Para ello, se deben ejecutar el siguiente comando:

Cuadro 10. Instalación de componentes de Zabbix en Ubuntu.

```
sudo apt install zabbix-server-pgsql zabbix-frontend-php php8.3-pgsql zabbix
    -apache-conf zabbix-sql-scripts zabbix-agent2
```

Nota. El comando instala el servidor Zabbix, *Frontend*, módulos de PostgreSQL, configuración de Apache y el agente Zabbix. Elaboración propia.

Al preguntar si se desea continuar con la instalación, se responde *Y* para confirmar la instalación de los paquetes necesarios.

Figura 103. Instalación de los componentes de Zabbix.

```
zabbix@zabbix-server:~$ sudo apt install zabbix-server-pgsql zabbix-frontend-php php8.3-pgsql zabbix-apache-conf zabb
ix-sql-scripts zabbix-agent2
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  apache2 apache2-bin apache2-data apache2-utils fonts-dejavu fonts-dejavu-extra fping libapache2-mod-php
  libapache2-mod-php8.3 libapr1t64 libaprutil1-dbd-sqlite3 libaprutil1-ldap libaprutil1t64 libevent-core-2.1-7t64
  libevent-extra-2.1-7t64 libevent-pthreads-2.1-7t64 libodbc2 libopenipmi0t64 libpq5 libsodium23 php-bcmath
  php-common php-curl php-gd php-ldap php-mbstring php-xml php8.3-bcmath php8.3-cli php8.3-common php8.3-curl
  php8.3-gd php8.3-ldap php8.3-mbstring php8.3-opcache php8.3-readline php8.3-xml snmpd
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom php-pear odbc-postgresql tdsodbc snmptrapd
  zabbix-nginx-conf postgresql-client postgresql
The following NEW packages will be installed:
  apache2 apache2-bin apache2-data apache2-utils fonts-dejavu fonts-dejavu-extra fping libapache2-mod-php
  libapache2-mod-php8.3 libapr1t64 libaprutil1-dbd-sqlite3 libaprutil1-ldap libaprutil1t64 libevent-core-2.1-7t64
  libevent-extra-2.1-7t64 libevent-pthreads-2.1-7t64 libodbc2 libopenipmi0t64 libpq5 libsodium23 php-bcmath
  php-common php-curl php-gd php-ldap php-mbstring php-xml php8.3-bcmath php8.3-cli php8.3-common php8.3-curl
  php8.3-gd php8.3-ldap php8.3-mbstring php8.3-opcache php8.3-pgsql php8.3-readline php8.3-xml snmpd zabbix-agent2
  zabbix-apache-conf zabbix-frontend-php zabbix-server-pgsql zabbix-sql-scripts
0 upgraded, 44 newly installed, 0 to remove and 8 not upgraded.
Need to get 35.2 MB of archives.
After this operation, 164 MB of additional disk space will be used.
Do you want to continue? [Y/n] 
```

Nota. Instalación simultánea de servidor, *Frontend* PHP, *scripts* SQL y agente 2 sobre backend PostgreSQL. Elaboración propia.

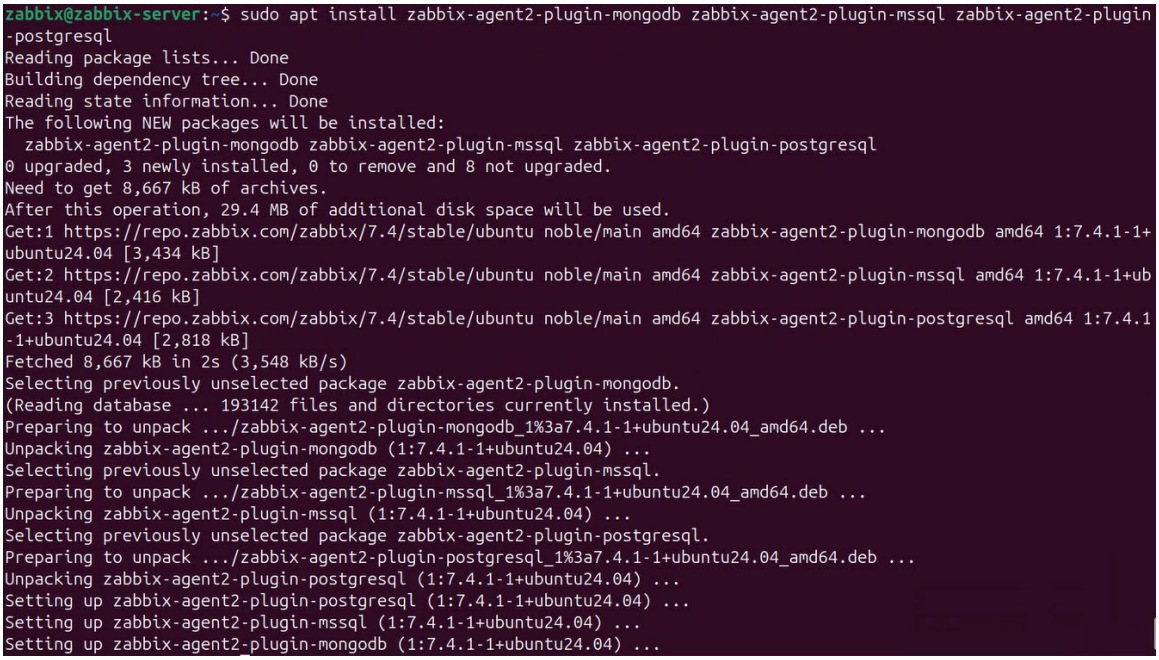
Luego se instalan los *plugins* para *Agent 2* de Zabbix. Para ello, se ejecuta el siguiente comando:

Cuadro 11. Comando para instalar *plugins* de Zabbix *Agent* 2.

```
sudo apt install zabbix-agent2-plugin-mongodb zabbix-agent2-plugin-mssql
    zabbix-agent2-plugin-postgresql
```

Nota. Instalación de extensiones de Zabbix *Agent* 2 para monitoreo nativo de motores de bases de datos populares. Elaboración propia.

Figura 104. *Plugins* instalados para Zabbix *Agent* 2.



Nota. Adición de *plugins* (MongoDB, MSSQL, PostgreSQL) ampliando el conjunto de ítems disponibles. Elaboración propia.

Una vez que se han instalado los componentes de Zabbix, se configura la base de datos para Zabbix *Server*. En este caso, se utiliza PostgreSQL como sistema de gestión de bases de datos.

7.6.2. Instalación de PostgreSQL

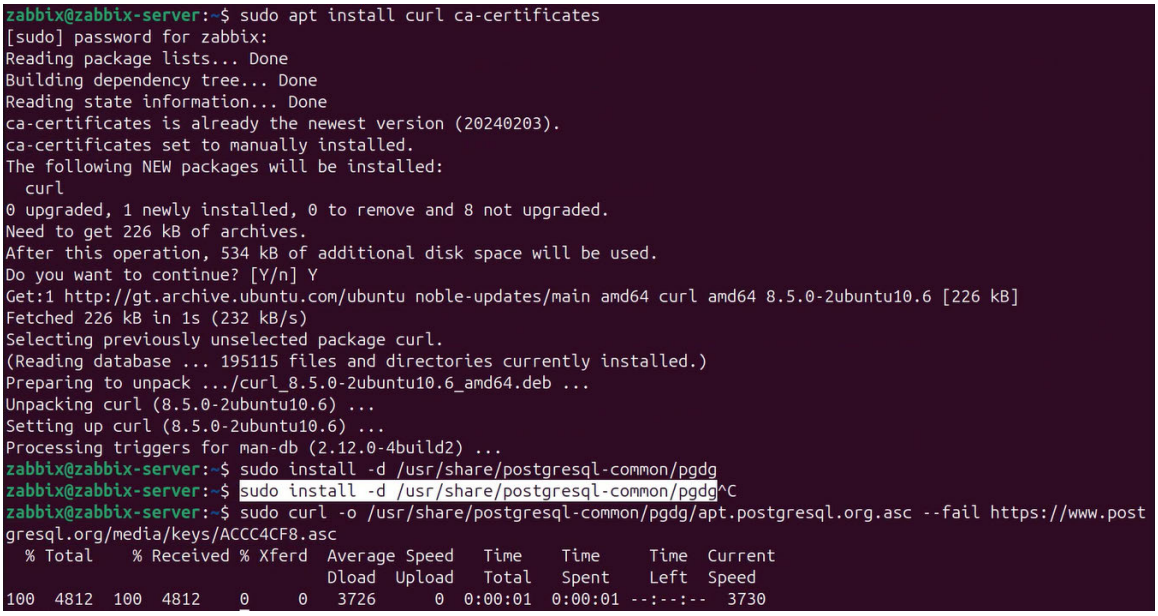
Para instalar PostgreSQL, se deben ejecutar los siguientes comandos:

Cuadro 12. Preparación del repositorio PGDG de PostgreSQL.

```
sudo apt install curl ca-certificates
sudo install -d /usr/share/postgresql-common/pgdg
sudo curl -o /usr/share/postgresql-common/pgdg/apt.postgresql.org.asc --fail
    https://www.postgresql.org/media/keys/ACCC4CF8.asc
```

Nota. Descarga de llave GPG oficial para verificar integridad de paquetes PostgreSQL descargados desde repositorio PGDG autenticado. Elaboración propia.

Figura 105. Preparación inicial de PostgreSQL con llave GPG.



Nota. Preparación inicial obteniendo llave GPG y directorios para habilitar repositorio PGDG. Elaboración propia.

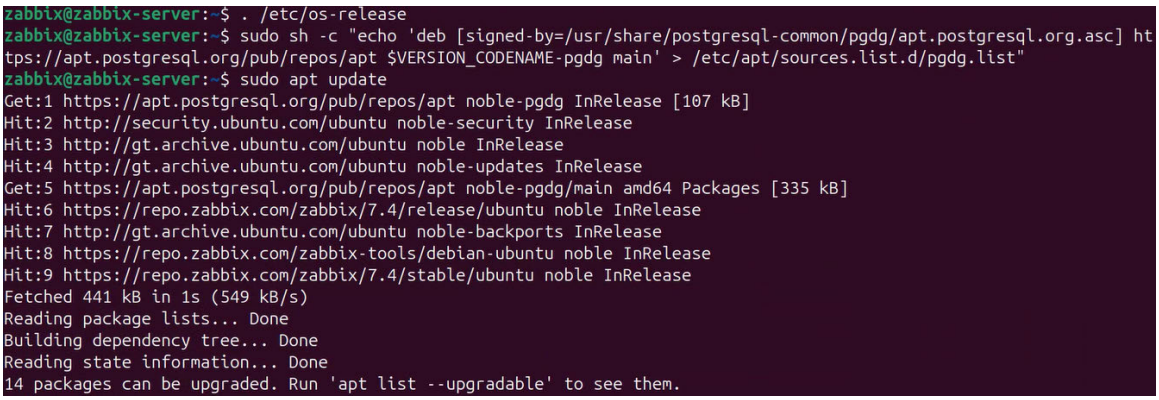
Luego, se debe agregar el repositorio de PostgreSQL a la lista de fuentes del sistema. Para ello, se deben ejecutar los siguientes comandos:

Cuadro 13. Definición e instalación del repositorio de PostgreSQL.

```
. /etc/os-release
sudo sh -c "echo 'deb [signed-by=/usr/share/postgresql-common/pgdg/apt.
    postgresql.org.asc] https://apt.postgresql.org/pub/repos/apt
    $VERSION_CODENAME-pgdg main' > /etc/apt/sources.list.d/pgdg.list"
sudo apt update
sudo apt install postgresql
```

Nota. Adición del repositorio oficial PGDG con llave criptográfica seguida de instalación de motor PostgreSQL requerido por Zabbix. Elaboración propia.

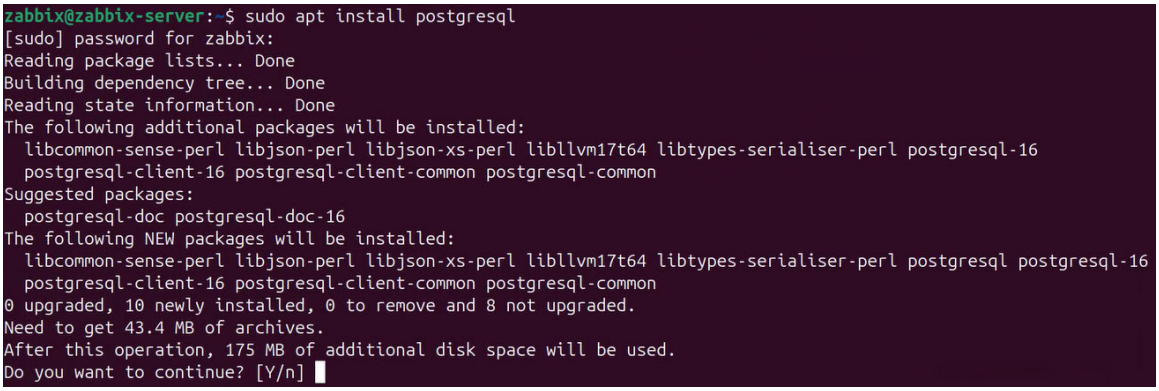
Figura 106. Confirmación de adición del repositorio e instalación de PostgreSQL.



Nota. Añadiendo entrada APT y ejecutando instalación del motor PostgreSQL requerido por Zabbix. Elaboración propia.

Se pregunta si se desea continuar con la instalación, se debe responder *Y* para confirmar la instalación de PostgreSQL.

Figura 107. Confirmación de instalación de paquetes de PostgreSQL.



Nota. Confirmación de paquetes instalados tras aceptar la operación en el nuevo repositorio. Elaboración propia.

Luego de instalar PostgreSQL, se inicializa el clúster de la base de datos.

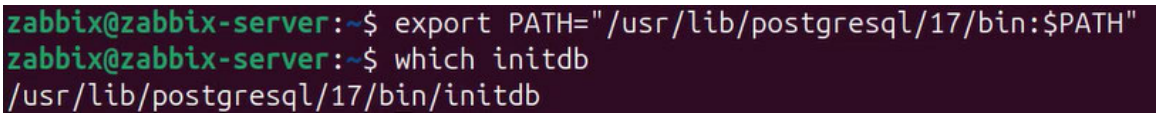
Para poder ejecutar los comandos de PostgreSQL, se agrega la carpeta `/usr/local/postgresql/17/bin` al `PATH`. Para verificar, se puede correr un comando de `PostgreSQL` y, si no se encuentra el comando, significa que no se ha agregado correctamente al `PATH`. Para ello, se ejecutan los siguientes comandos:

Cuadro 14. Exportación del PATH de PostgreSQL.

```
export PATH="/usr/lib/postgresql/17/bin:$PATH"
which initdb
```

Nota. Adición de ruta binarios PostgreSQL 17 al `PATH` de sesión seguida de verificación que está accesible para inicializar clúster. Elaboración propia.

Figura 108. Verificación de exportación del PATH de PostgreSQL.



Nota. Verificación de initdb tras ajustar PATH para utilizar binarios de la versión instalada. Elaboración propia.

Luego se crea el directorio donde está el clúster de la base de datos. Para ello, se ejecutan los siguientes comandos:

Cuadro 15. Creación del directorio del clúster de PostgreSQL.

```
cd /home/zabbix
mkdir pgsql
cd pgsql
mkdir data
cd ~
```

Nota. Creación de estructura de directorios para alojar datos de clúster PostgreSQL separado del sistema. Elaboración propia.

Figura 109. Creación de directorio del clúster de PostgreSQL.

Nota. Estructura de carpetas definida para alojar datos del clúster exclusivo de Zabbix. Elaboración propia.

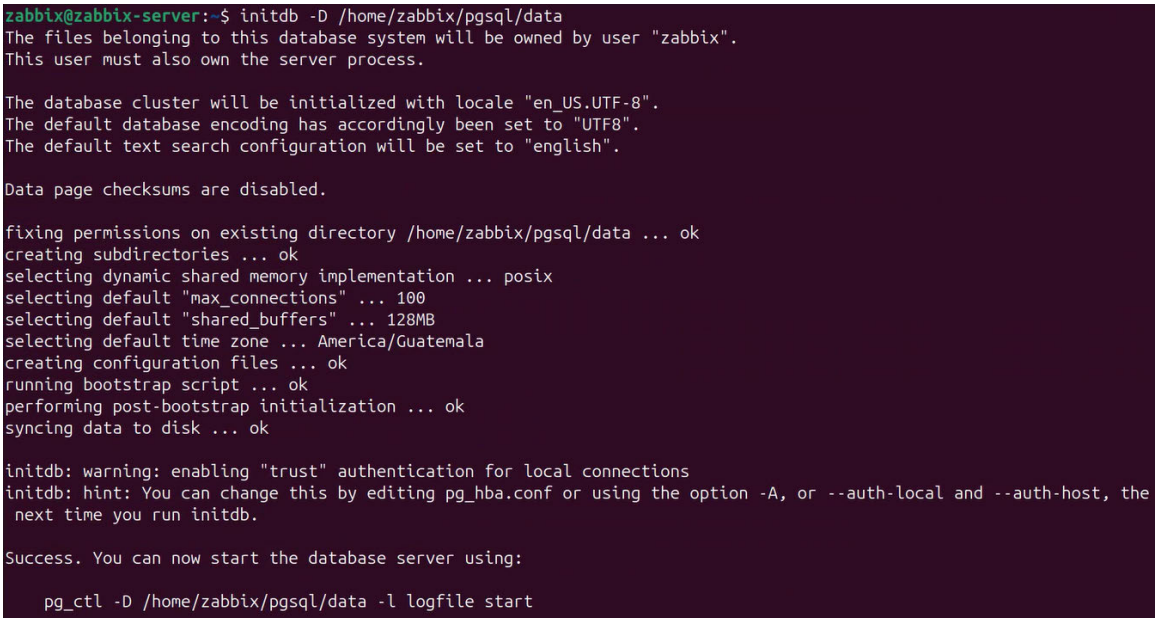
Con el directorio creado, se inicializa el clúster de PostgreSQL. Para ello, se ejecuta el siguiente comando:

Cuadro 16. Comando para inicializar el clúster de PostgreSQL.

```
initdb -D /home/zabbix/pgsql/data
```

Nota. Inicialización del nuevo clúster PostgreSQL en directorio destinado creando estructura interna. Elaboración propia.

Figura 110. Inicialización del clúster de PostgreSQL.



Nota. Ejecución de initdb generando estructura interna (catálogos y directorios de WAL) para el clúster. Elaboración propia.

Ahora se edita el archivo de permisos de acceso a la base de datos. Para ello, se edita el archivo `pg_hba.conf` que se encuentra en el directorio del clúster de la base de datos. Para ello, se ejecuta el siguiente comando:

Cuadro 17. Edición del archivo `pg_hba.conf`.

```
nano /home/zabbix/pgsql/data/pg_hba.conf
```

Nota. Comando para abrir el archivo de configuración de acceso a bases de datos de PostgreSQL en editor de texto. Elaboración propia.

En este archivo se agregan las siguientes líneas al final del archivo:

Cuadro 18. Configuración de `pg_hba.conf`.

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   zabbix          zabbix                                  scram-sha
    -256
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
# IPv6 local connections:
host    all             all             ::1/128                 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     trust
host    replication     all             127.0.0.1/32            trust
host    replication     all             ::1/128                 trust
```

Nota. Configuración de autenticación pg_hba definiendo métodos de acceso: scram-sha-256 para usuario Zabbix, *trust* para conexiones locales y replicación en esta máquina virtual aislada. Elaboración propia.

Esto se realiza de esta manera ya que el Zabbix *Server*, Zabbix *Frontend* y la base de datos de Zabbix se encuentran en la misma máquina virtual. Si no, la configuración sería diferente, teniendo que configurar conexiones TCP y especificando las direcciones IP de los servidores que se conectan a la base de datos.

Una vez editado el archivo, se debe guardar y salir del editor. Para ello, se debe presionar `Ctrl + O` para guardar y luego `Ctrl + X` para salir del editor.

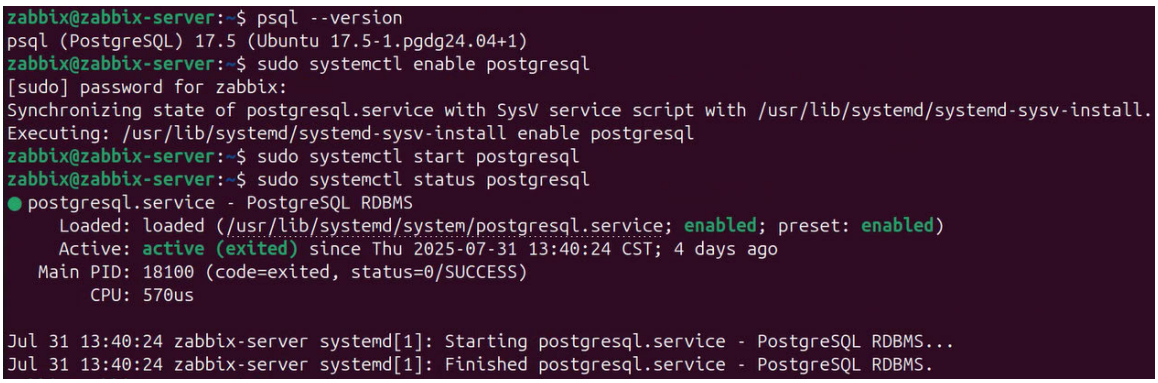
Luego, se debe iniciar el servicio de PostgreSQL. Para ello, se deben ejecutar los siguientes comandos:

Cuadro 19. Inicialización del servicio de PostgreSQL.

```
sudo systemctl enable postgresql
sudo systemctl start postgresql
sudo systemctl status postgresql
```

Nota. Habilitación del servicio PostgreSQL en arranque del sistema, inicio del servicio y verificación de estado operativo. Elaboración propia.

Figura 111. Iniciar servicio de PostgreSQL.



Nota. Estado activo del servicio PostgreSQL tras enable y start verificando readiness del backend. Elaboración propia.

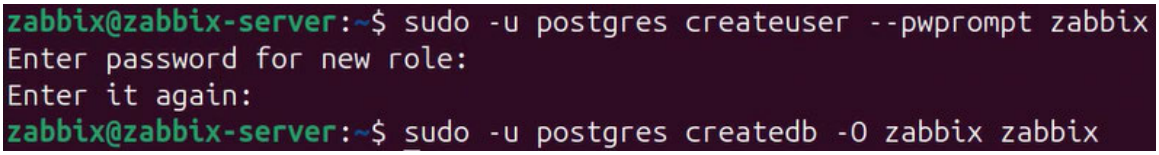
Ahora se debe crear la base de datos y el usuario para Zabbix. Para ello, se deben ejecutar los siguientes comandos:

Cuadro 20. Comando para crear la base de datos de Zabbix.

```
sudo -u postgres createuser --pwprompt zabbix
sudo -u postgres createdb -O zabbix zabbix
```

Nota. Creación de usuario PostgreSQL con prompta de contraseña seguida de creación de base de datos Zabbix asignada al usuario del mismo nombre. Elaboración propia.

Figura 112. Creación de la base de datos de Zabbix.



Nota. Generación de rol y base de datos dedicados para aislar información de monitoreo. Elaboración propia.

Ahora se debe importar el esquema de la base de datos de Zabbix. Para ello, se deben ejecutar los siguientes comandos:

Cuadro 21. Comando para importar el esquema de la base de datos de Zabbix.

```
zcat /usr/share/zabbix/sql-scripts/postgresql/server.sql.gz | sudo -u zabbix
    psql zabbix
```

Nota. Descompresión e inyección del esquema SQL inicial de Zabbix en base de datos PostgreSQL importando todas las tablas, índices y estructura de datos. Elaboración propia.

Luego se debe configurar el archivo de configuración de Zabbix *Server*. Para ello, se debe editar el archivo `zabbix_server.conf` que se encuentra en el directorio `/etc/zabbix/`. Para ello, se debe ejecutar el siguiente comando:

Cuadro 22. Edición del archivo `zabbix_server.conf`.

```
sudo nano /etc/zabbix/zabbix_server.conf
```

Nota. Comando para abrir el archivo principal de configuración de Zabbix *Server* donde se definen parámetros de base de datos y servicios. Elaboración propia.

En este archivo se debe agregar la contraseña del usuario de la base de datos que se creó anteriormente. Para ello, se debe buscar la línea `DBPassword=` y agregar la contraseña del usuario Zabbix que se creó anteriormente.

Cuadro 23. Configuración de `zabbix_server.conf`.

```
DBName=zabbix

### Option: DBSchema
#       Schema name. Used for PostgreSQL.
#
# Mandatory: no
# Default:
# DBSchema=

### Option: DBUser
#       Database user.
#
# Mandatory: no
# Default:
# DBUser=

DBUser=zabbix

### Option: DBPassword
#       Database password.
#       Comment this line if no password is used.
#
# Mandatory: no
# Default:
DBPassword=1101
```

Nota. Parámetros de conexión a PostgreSQL requeridos para que servidor Zabbix autentique y acceda a datos de monitoreo. Elaboración propia.

Una vez editado el archivo, se guarda y se sale del editor. Con ello ya se inician los servicios de Zabbix *Server*, Zabbix *Agent* 2 y Zabbix *Frontend*, en este caso, Apache. Para ello, se ejecutan los siguientes comandos:

Cuadro 24. Inicialización de los servicios de Zabbix.

```
sudo systemctl restart zabbix-server zabbix-agent2 apache2
sudo systemctl enable zabbix-server zabbix-agent2 apache2
```

Nota. Reinicio inmediato de componentes Zabbix y Apache cargando configuración actualizada, seguida de habilitación en arranque automático. Elaboración propia.

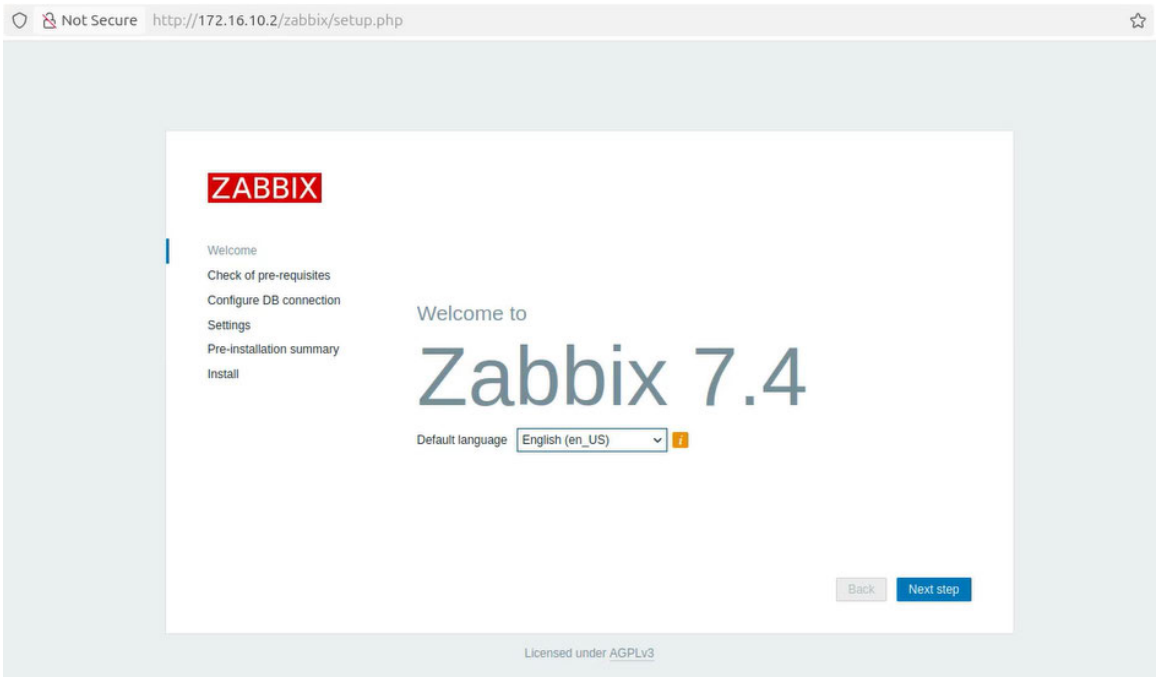
Figura 113. Iniciar servicios de Zabbix.

```
zabbix@zabbix-server:~$ sudo systemctl restart zabbix-server zabbix-agent2 apache2
[sudo] password for zabbix:
zabbix@zabbix-server:~$ sudo systemctl enable zabbix-server zabbix-agent2 apache2
Synchronizing state of zabbix-server.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable zabbix-server
Synchronizing state of zabbix-agent2.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable zabbix-agent2
Synchronizing state of apache2.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable apache2
Created symlink /etc/systemd/system/multi-user.target.wants/zabbix-server.service → /usr/lib/systemd/system/zabbix-se
rver.service.
```

Nota. Reinicio simultáneo de Zabbix *Server*, Zabbix *Agent* 2 y Apache asegurando carga de nueva configuración. Elaboración propia.

Una vez que se han iniciado los servicios, se accede a la interfaz web de Zabbix desde un navegador web. Para ello, se ingresa la dirección IP de la máquina virtual donde se instaló Zabbix en el navegador web. En este caso, la dirección IP 172.16.10.2. La URL es la siguiente: `http://172.16.10.2/zabbix`.

Figura 114. Interfaz web de Zabbix.



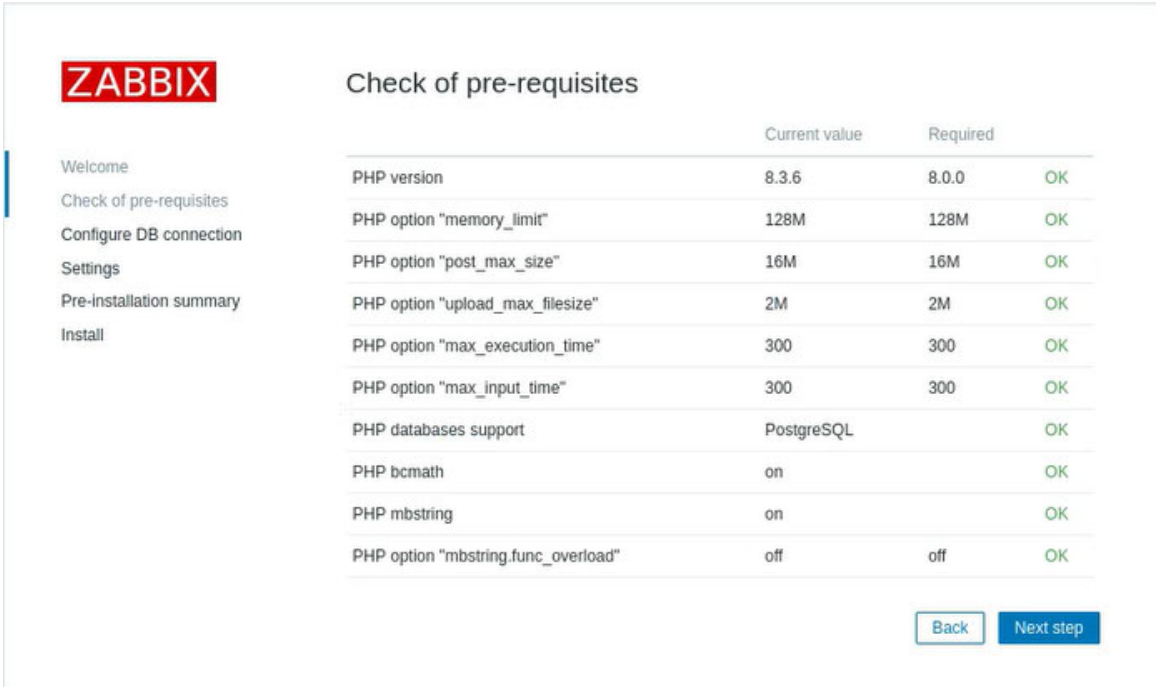
Nota. Pantalla inicial del *Frontend* confirmando disponibilidad HTTP del servicio. Elaboración propia.

7.6.3. Configuración de Zabbix

Una vez que se ha accedido a la interfaz web de Zabbix, se procede a la configuración inicial. En la primera pantalla, se selecciona el idioma y se hace clic en el botón de *Next*

step. En la siguiente pantalla, se verifica que los requisitos del sistema se cumplan. Si todo está correcto, debería aparecer un mensaje de *OK* en todos los requisitos. Se hace clic en el botón de *Next step* para continuar.

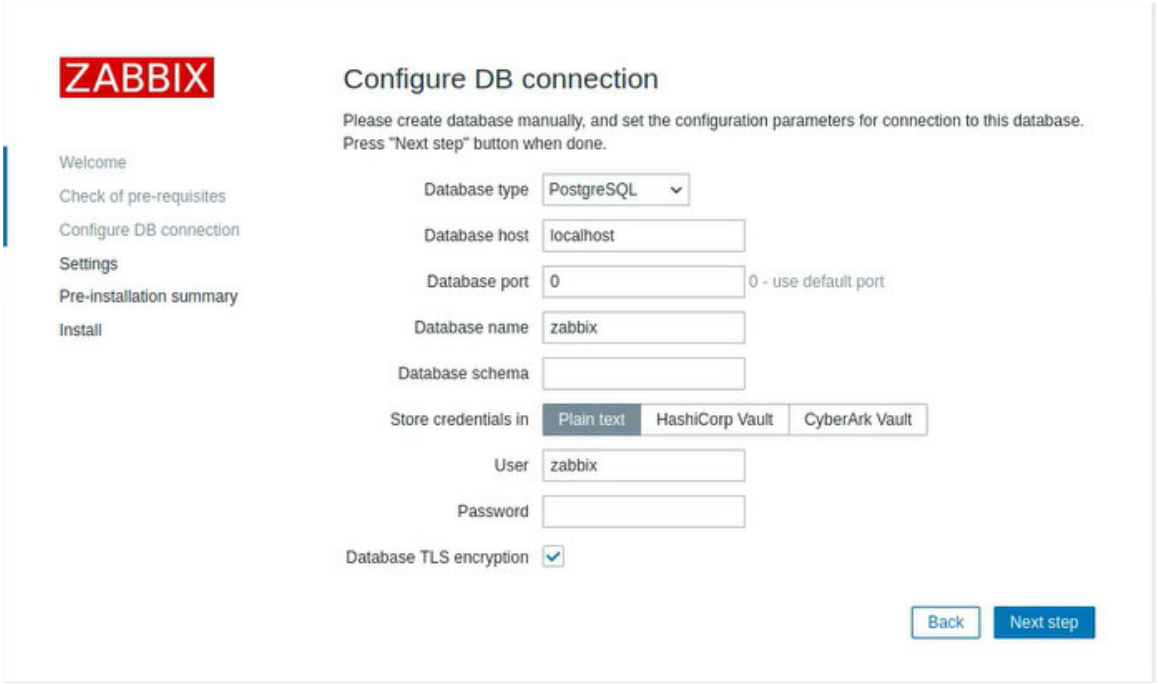
Figura 115. Requisitos del sistema de Zabbix.



Nota. Chequeo de prerequisitos donde cada dependencia aparece en estado OK. Elaboración propia.

En la siguiente pantalla, se configura la conexión a la base de datos. Se selecciona el tipo de base de datos que se está utilizando, en este caso, PostgreSQL. Luego, se ingresa la información de la base de datos que se creó anteriormente, es decir, el nombre de la base de datos (`zabbix`), el usuario (`zabbix`) y la contraseña que se configuró en el archivo `zabbix_server.conf`. Lo demás se deja por defecto. Luego, se hace clic en el botón de *Next step* para continuar.

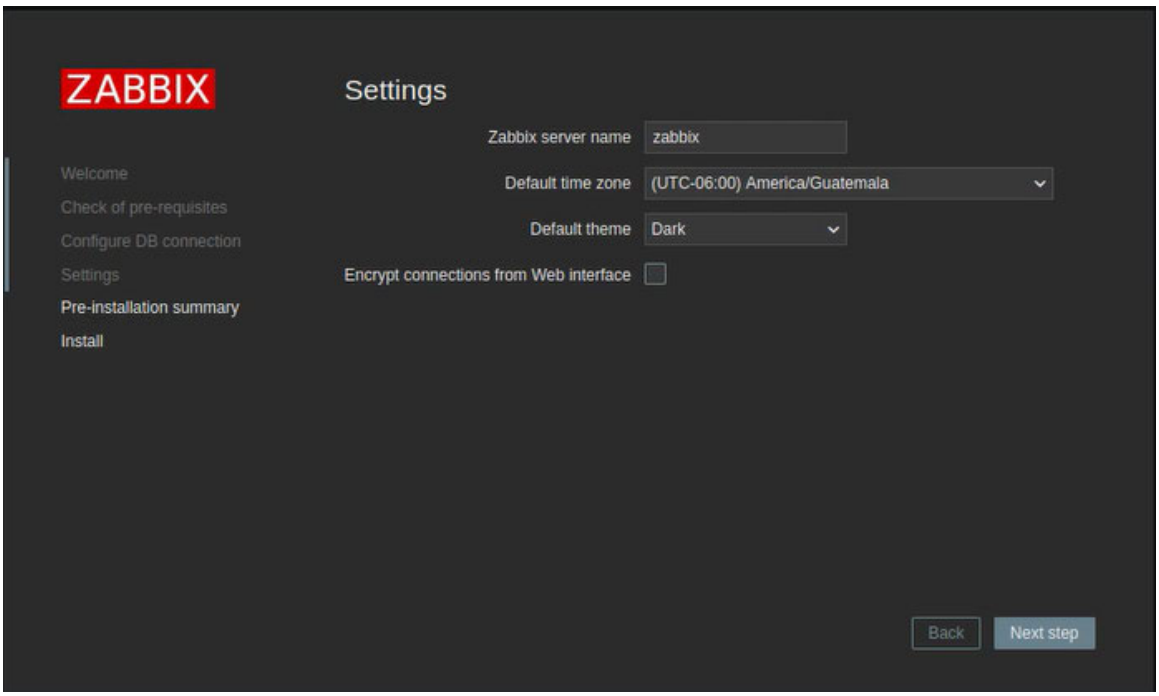
Figura 116. Configuración de la conexión a la base de datos de Zabbix.



Nota. Formulario para definir parámetros de acceso a PostgreSQL (DB, usuario, contraseña). Elaboración propia.

En la siguiente pantalla, se configura el nombre del servidor de Zabbix, la zona horaria y el tema de la interfaz web. Luego, se hace clic en el botón de *Next step* para continuar.

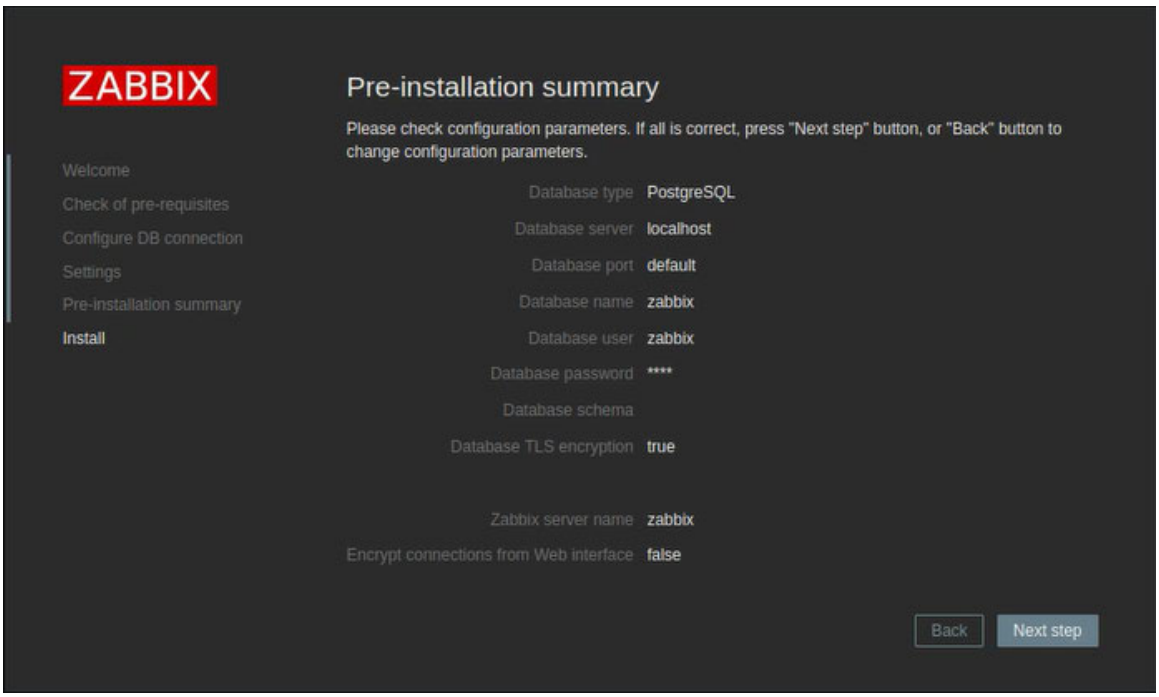
Figura 117. Configuración del nombre del servidor de Zabbix.



Nota. Ajustes de identidad (*Server name*), zona horaria y tema aplicados al entorno. Elaboración propia.

En la siguiente pantalla, se muestra un resumen de la configuración que se ha realizado. Si todo está correcto, se hace clic en el botón de *Next step* para continuar.

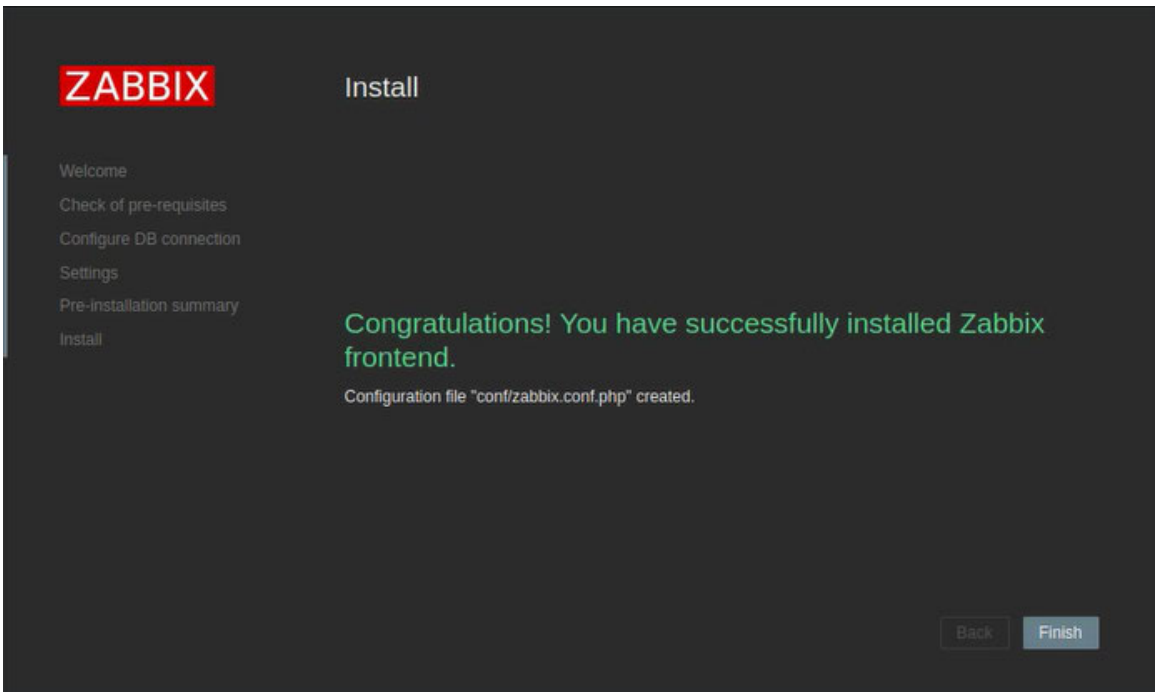
Figura 118. Resumen de la configuración de Zabbix.



Nota. Pantalla de verificación previa a la finalización mostrando parámetros introducidos. Elaboración propia.

Finalmente, se muestra un mensaje de que la instalación y configuración de Zabbix *Frontend* se ha realizado correctamente. Se hace clic en el botón de *Finish* para finalizar el proceso.

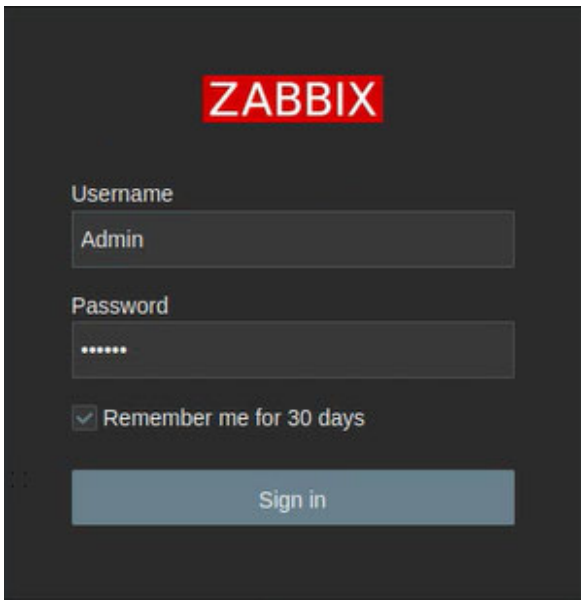
Figura 119. Finalización de la instalación y configuración de Zabbix.



Nota. Mensaje de éxito indicando que el *frontend* quedó listo para autenticación inicial. Elaboración propia.

Una vez que se ha finalizado el proceso, se puede iniciar sesión en la interfaz web de Zabbix utilizando el usuario y la contraseña predeterminados. El usuario predeterminado es `Admin` y la contraseña predeterminada es `zabbix`. Se recomienda cambiar la contraseña después del primer inicio de sesión por razones de seguridad.

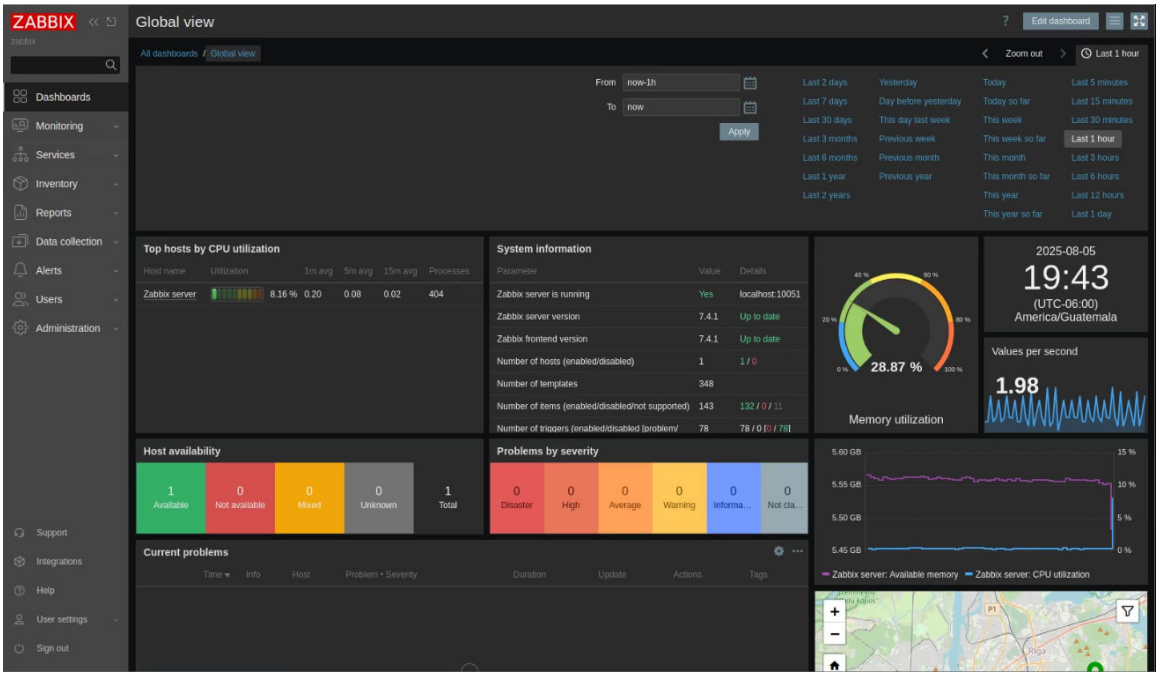
Figura 120. Inicio de sesión en la interfaz web de Zabbix.



Nota. Formulario de autenticación con credenciales por defecto antes de endurecer seguridad. Elaboración propia.

Se puede ver el panel de control de Zabbix. Desde aquí, se pueden agregar *hosts*, configurar plantillas, crear gráficos y monitorear el estado de los sistemas y redes.

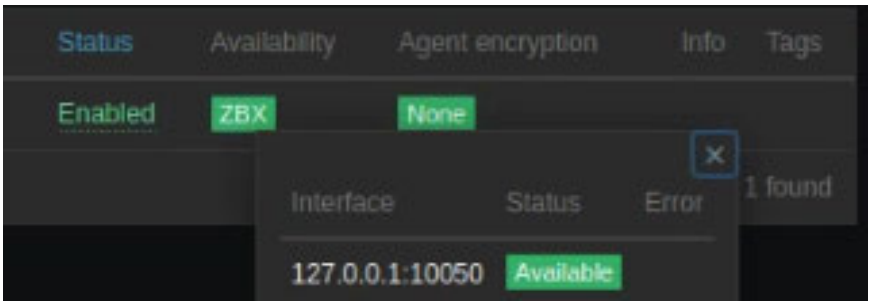
Figura 121. Panel de control de Zabbix.



Nota. *Dashboard* principal mostrando vistas iniciales y navegación para gestión de monitoreo. Elaboración propia.

Para confirmar que el Zabbix *Agent* 2 está funcionando correctamente, se puede ir a la sección *Data Collection* y luego a la sección *Hosts*. Aquí se puede ver el estado de los *hosts* que se están monitoreando. Si el Zabbix *Agent* 2 está funcionando correctamente, debería aparecer un *host* con el nombre de Zabbix *Server* en la lista de *hosts*, y en el status debería aparecer como *Enabled*, y en la columna de *Availability* debería aparecer *ZBX* en color verde, y al poner el cursor sobre el icono de *ZBX* debería aparecer el status diciendo *Available*.

Figura 122. Estado del Zabbix *Agent* 2 en la interfaz web de Zabbix.

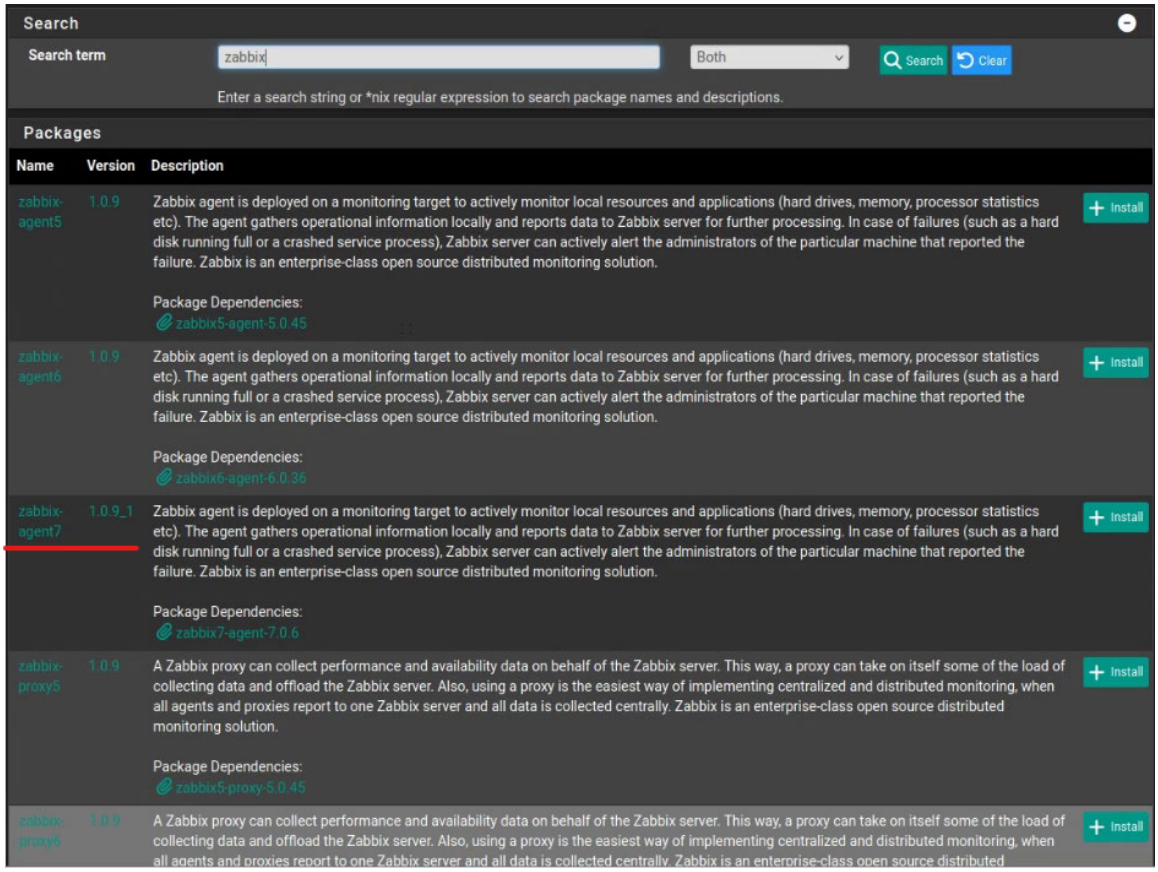


Nota. Lista de *hosts* confirmando disponibilidad del propio servidor (ZBX verde) tras despliegue. Elaboración propia.

7.6.4. Monitoreo de pfSense con Zabbix

Para monitorear pfSense con Zabbix, se instala el Zabbix *Agent* en pfSense. Para ello, se va a la sección *System* en el menú principal y se selecciona la opción *Package Manager*. En esta sección se pueden ver los diferentes paquetes que se pueden instalar en pfSense. Se busca el paquete de Zabbix *Agent* y se instala la versión correspondiente a la versión de Zabbix que se está utilizando, en este caso, la versión 7.

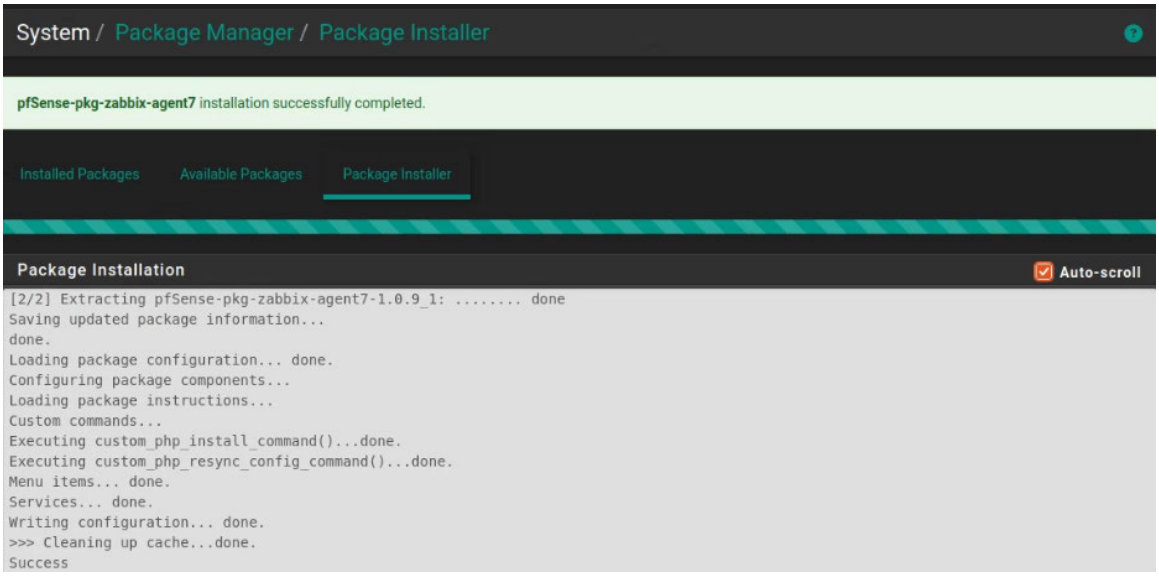
Figura 123. Búsqueda e instalación del paquete Zabbix *Agent* en el Package Manager de pfSense.



Nota. Selección del paquete de agente versión 7 directamente desde el gestor de paquetes de pfSense. Elaboración propia.

Se confirma la instalación del paquete y se espera a que se complete la instalación.

Figura 124. Descarga e instalación en progreso del Zabbix *Agent* en pfSense.

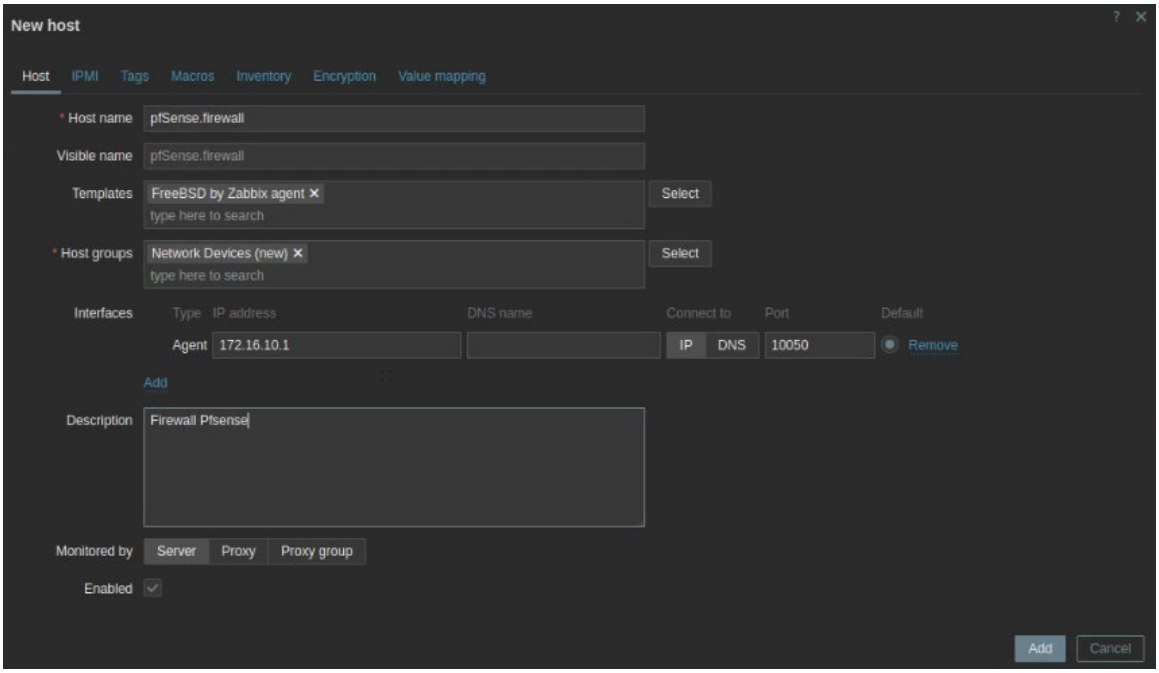


Nota. Proceso de instalación mostrando progreso de descarga e instalación del agente. Elaboración propia.

Antes de configurar el Zabbix *Agent*, se crea un nuevo *host* en la interfaz web de Zabbix para monitorear pfSense. Para ello, se va a la sección *Monitoring* en el menú principal y se selecciona la opción *Hosts*. Se hace clic en el botón de *Create host* para crear un nuevo *host*.

En la sección *Hostname* se ingresa el nombre del *host*, en este caso, *pfSense.firewall*. En la sección *Visible name* se puede ingresar un nombre descriptivo para el *host*, en este caso, pfSense *Firewall*. En la sección *Templates* se agrega la plantilla de *FreeBSD by Zabbix Agent* para que se puedan monitorear los diferentes parámetros del sistema operativo de pfSense. En la sección *Groups* se selecciona el grupo al que pertenece el *host*, en este caso, el grupo de *Network devices*. En la sección *Interfaces* se agrega una nueva interfaz de tipo *Agent* y se ingresa la dirección IP de la interfaz que del SOC & MOC que está conectada a pfSense, en este caso, 172.16.10.1. En la sección *Monitored by* se selecciona *Server* para que el *host* sea monitoreado por el servidor de Zabbix, por último se habilita el *host* y se guarda la configuración.

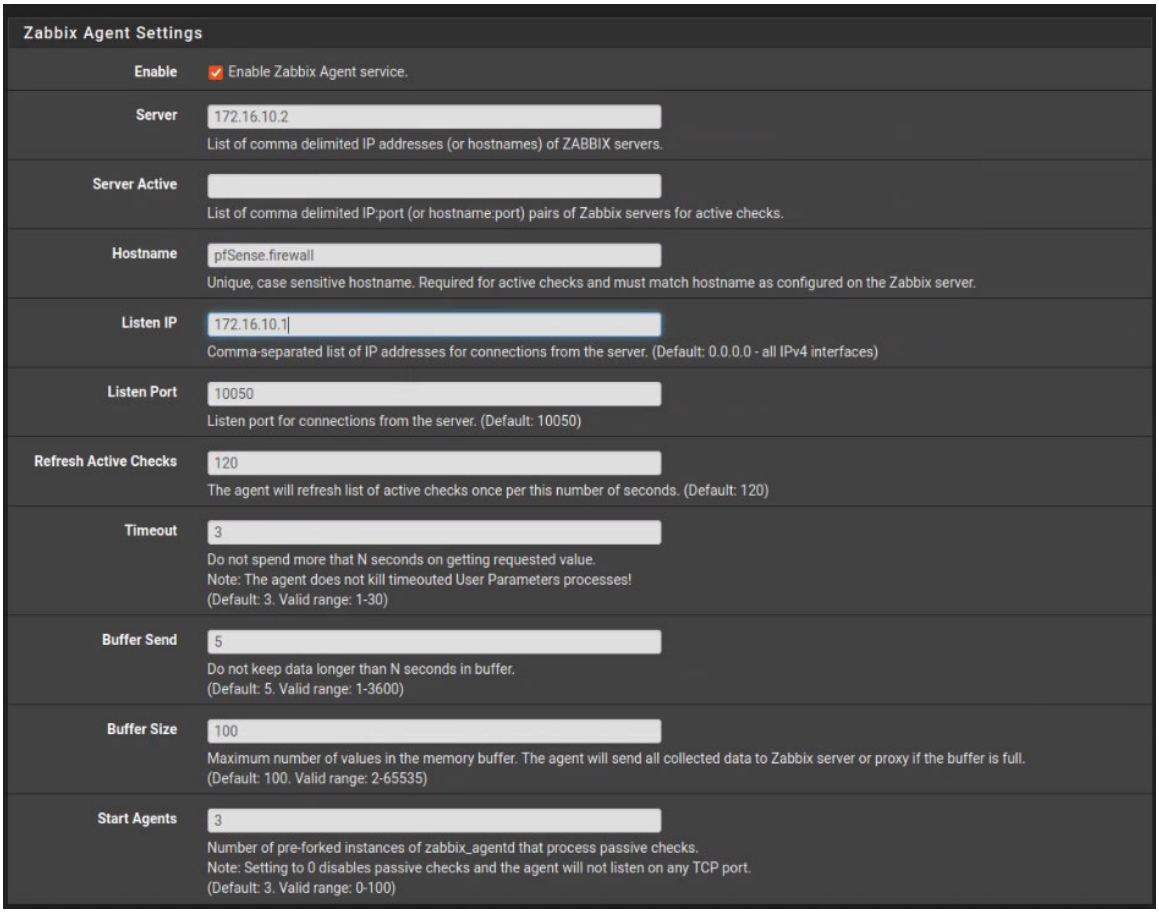
Figura 125. Configuración del nuevo *host* en la interfaz web de Zabbix.



Nota. Parámetros finales del *host* pfSense (interfaces, grupos y plantillas) antes de guardar. Elaboración propia.

Una vez que se ha creado el *host*, se configura el Zabbix *Agent* en pfSense. Para ello, se va a la sección *Services* en el menú principal y se selecciona la opción Zabbix *Agent*. En esta sección se pueden ver las diferentes opciones de configuración del agente. Se habilita el agente y se configura la dirección IP del servidor de Zabbix, en este caso, 172.16.10.2. En la sección *Hostname* se ingresa el nombre del *host* que se creó anteriormente, en este caso, *pfSense.firewall*. En la sección *Listen IP* se ingresa la dirección IP de la interfaz de pfSense que está conectada al SOC & MOC, en este caso, 172.16.10.1; lo demás se deja por defecto.

Figura 126. Configuración del Zabbix *Agent* en la interfaz web de pfSense.



Nota. Pantalla de configuración del agente definiendo servidor (172.16.10.2) y Listen IP de pfSense. Elaboración propia.

Una vez que se ha configurado el Zabbix *Agent*, en la interfaz web de Zabbix

7.6.5. Monitoreo de la red con Zabbix

Para monitorear la red con Zabbix, primero se debe instalar el Zabbix *Agent* en los dispositivos que se desean monitorear. En este caso, se instala el Zabbix *Agent* en todas las máquinas virtuales que se encuentran en todas las redes; LAN, DMZ y SOC & MOC. El modo en que trabajan los agentes es en modo pasivo, ya que los agentes están esperando las solicitudes del servidor de Zabbix para enviar la información de monitoreo. En cambio, en el caso que los agentes se configuren en modo activo, estos iniciarían la comunicación con el servidor de Zabbix, lo cual no es recomendable en este caso, ya que los agentes se encuentran detrás de un *firewall* y no se desea abrir puertos adicionales en el *firewall* para permitir la comunicación del agente con el servidor de Zabbix.

7.6.5.1. Instalación del Zabbix *Agent* en Ubuntu *Desktop* y *Server* 24.04.2

Se cuenta con un usuario utilizando Ubuntu *Desktop* 24.04.2 en la red LAN, el servidor de Pi-hole utiliza Ubuntu Server 24.04.2 y Metasploitable 3 en la red DMZ utiliza Ubuntu *Server* 14.04.

Para instalar el Zabbix *Agent* 2 tanto en Ubuntu *Desktop* 24.04.2 como en Ubuntu *Server* 24.04.2, primero se instala el repositorio de Zabbix. Para ello se ejecutan los siguientes comandos en la terminal:

Cuadro 25. Preparación del repositorio de Zabbix *Agent* en Ubuntu.

```
sudo -s
wget https://repo.zabbix.com/zabbix/7.4/release/ubuntu/pool/main/z/zabbix-release/zabbix-release_latest_7.4+ubuntu24.04_all.deb
dpkg -i zabbix-release_latest_7.4+ubuntu24.04_all.deb
apt update
```

Nota. Descargar e instalar el paquete de repositorio Zabbix 7.4 en cliente Ubuntu 24.04 antes de instalar agente. Elaboración propia.

Figura 127. Descarga e instalación del paquete de Zabbix *Agent* en Ubuntu.

Nota. Preparación de repositorio Zabbix en *host* Ubuntu para instalación del Zabbix *Agent* 2. Elaboración propia.

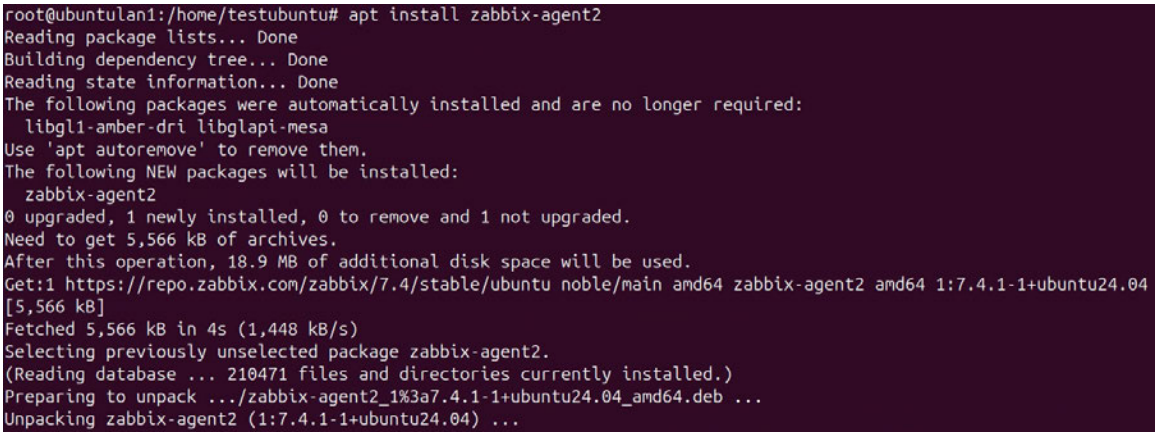
Lo segundo es instalar el Zabbix *Agent* 2 y sus respectivos *plugins*. Para ello, se ejecutan los siguientes comandos en la terminal:

Cuadro 26. Instalación de paquetes de Zabbix *Agent* 2 con *plugins* en Ubuntu.

```
apt install zabbix-agent2
apt install zabbix-agent2-plugin-mongodb zabbix-agent2-plugin-mssql zabbix-agent2-plugin-postgresql
```

Nota. Instalación del agente Zabbix 2 y extensiones para monitoreo de bases de datos en clientes Ubuntu. Elaboración propia.

Figura 128. Confirmación de instalación de Zabbix *Agent* 2 con *plugins* en Ubuntu.



Nota. Instalación de binarios del agente y *plugins* que amplían monitoreo de bases de datos. Elaboración propia.

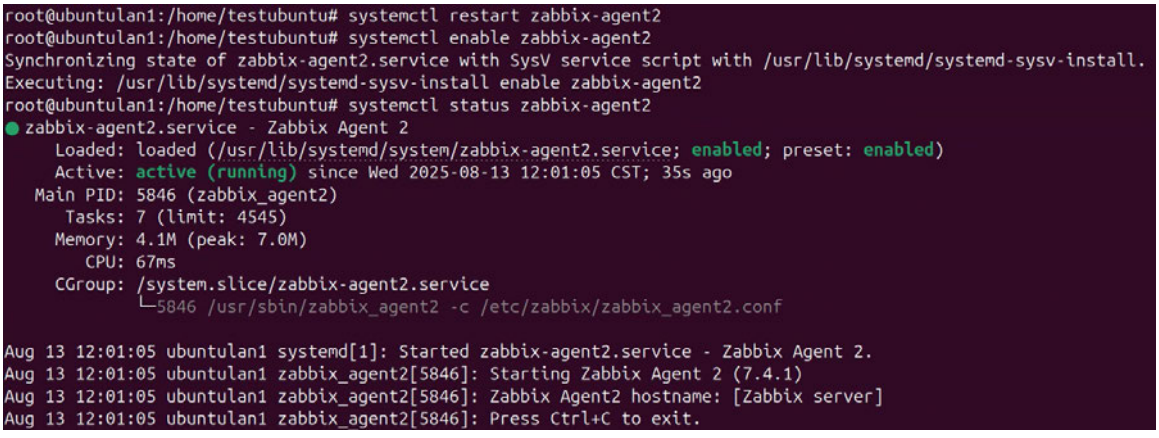
Por último se inicia el servicio del Zabbix *Agent* 2 para que comience a monitorear el sistema. Esto se puede hacer ejecutando el siguiente comando en la terminal:

Cuadro 27. Comando para iniciar el servicio Zabbix *Agent* 2 en Ubuntu.

```
systemctl restart zabbix-agent2
systemctl enable zabbix-agent2
```

Nota. Reinicio del servicio e inclusión en arranque automático para asegurar monitoreo continuo del cliente. Elaboración propia.

Figura 129. Confirmación de arranque del servicio Zabbix *Agent* 2 en Ubuntu.



Nota. Arranque y habilitación del servicio asegurando persistencia tras reinicios. Elaboración propia.

Ahora se configura el agente para que se conecte al servidor de Zabbix. Para ello, se edita el archivo de configuración del agente que se encuentra en `/etc/zabbix/zabbix_agent2.conf`. Para ello, se ejecuta el siguiente comando en la terminal:

Cuadro 28. Comando para editar el archivo de configuración del Zabbix *Agent* 2 en Ubuntu.

```
nano /etc/zabbix/zabbix_agent2.conf
```

Nota. Apertura del archivo de configuración de agente para definir servidor Zabbix y nombre de *host* del cliente. Elaboración propia.

En este archivo se busca la línea `Server=` y se agrega la dirección IP del servidor de Zabbix, en este caso, 172.16.10.1. También, se busca la línea `Hostname=` y se agrega el nombre del *host* que se está monitoreando, en este caso, Ubuntu *Desktop* o Ubuntu *Server* según corresponda. Se puede obtener el *hostname* ejecutando el comando `hostname` en la terminal.

Cuadro 29. Configuración del archivo `zabbix_agent2.conf` en Ubuntu.

```
### Option: Server
#       List of comma delimited IP addresses, optionally in CIDR notation, or DNS names of Zabbix servers and Zab>
#       Incoming connections will be accepted only from the hosts listed here.
#       If IPv6 support is enabled then '127.0.0.1', '::127.0.0.1', '::ffff:127.0.0.1' are treated equally
#       and '::/0' will allow any IPv4 or IPv6 address.
#       '0.0.0.0/0' can be used to allow any IPv4 address.
#       Example: Server=127.0.0.1,192.168.1.0/24,::1,2001:db8::/32,zabbix.example.com
#
#    If left empty or not set will disable passive checks, and Zabbix agent 2 will not listen on the ListenPort.
#
# Mandatory: no
# Default:
# Server=

Server=172.16.10.2


### Option: Hostname
#       List of comma delimited unique, case sensitive hostnames.
#       Required for active checks and must match hostnames as configured on the server.
#       Value is acquired from HostnameItem if undefined.
#
# Mandatory: no
# Default:
# Hostname=

Hostname=ubuntulan1
```

Nota. Parámetros principales del agente especificando servidor central e identificador único del *host* cliente. Elaboración propia.

Para que los cambios tomen efecto, se debe reiniciar el servicio del agente de Zabbix. Esto se puede hacer ejecutando el siguiente comando en la terminal:

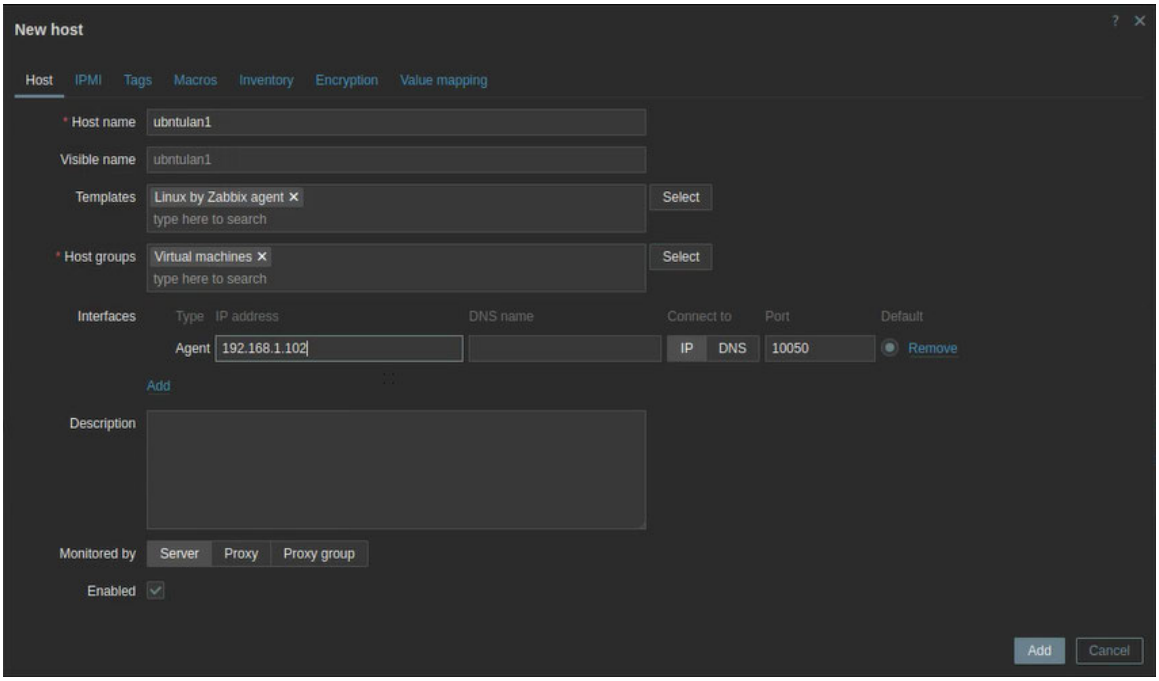
Cuadro 30. Reinicio del Zabbix *Agent* 2 para aplicar cambios en Ubuntu *Desktop*.

```
systemctl restart zabbix-agent2
exit
```

Nota. Reinicio del servicio aplicando nuevos parámetros de configuración iniciando comunicación pasiva con servidor. Elaboración propia.

Ya con el agente instalado y configurado, se debe agregar el *host* en la interfaz web de Zabbix. Para ello, se debe ir a la sección *Monitoring* en el menú principal y seleccionar la opción *Hosts*. Se debe hacer clic en el botón de *Create host* para crear un nuevo *host*. En la sección *Hostname* se deben ingresar los datos requeridos. En la sección *Template* se debe seleccionar el *template* correspondiente al sistema operativo del *host*, en este caso, *Linux by Zabbix Agent*; en el apartado *Host Groups* se selecciona *Virtual Machines*; por último, en la sección *Interfaces* se debe agregar la dirección IP del *host*, en este caso, 192.168.1.102.

Figura 130. Creación del *host* Ubuntu *Desktop* en la interfaz web de Zabbix.



Nota. Registro del *host* Ubuntu *Desktop* asignando plantilla Linux para iniciar recolección de métricas. Elaboración propia.

Se esperan unos segundos para que los cambios tomen efecto, y luego se verifica que el nuevo *host* aparezca en la lista de *hosts* en la interfaz web de Zabbix. Si el *host* aparece con el estado *Enabled* y la disponibilidad es *ZBX* en color verde, significa que el agente está funcionando correctamente y está enviando datos al servidor de Zabbix.

Figura 131. Verificación de disponibilidad del *host* Ubuntu *Desktop* en Zabbix.

Nota. *Host* reportando disponibilidad ZBX verde indicando comunicación correcta con el servidor. Elaboración propia.

Estos mismos pasos se deben realizar para el Zabbix *Agent* en el servidor de Pi-hole que utiliza Ubuntu *Server* 24.04.2. Únicamente se debe cambiar el *hostname* en la configuración en el archivo de configuración del agente.

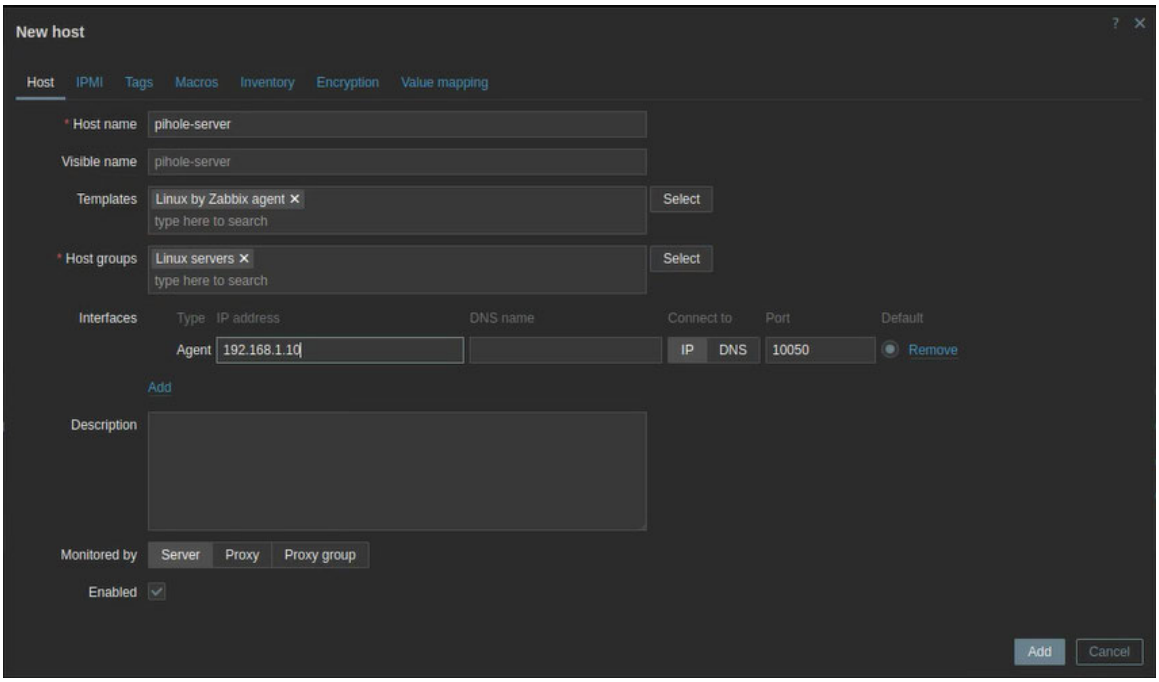
Cuadro 31. Configuración del Zabbix *Agent* 2 para servidor Pi-hole en Ubuntu.

```
### Option: Hostname
#       List of comma delimited unique, case sensitive hostnames.
#       Required for active checks and must match hostnames as configured on the server.
#       Value is acquired from HostnameItem if undefined.
#
# Mandatory: no
# Default:
# Hostname=

Hostname=pihole-server
```

Nota. Configuración del identificador de *host* para el servidor Pi-hole permitiendo su distinción en el servidor central. Elaboración propia.

En la interfaz web de Zabbix, del mismo modo se crea un nuevo *host* para el servidor de Pi-hole.

Figura 132. Creación del *host* Pi-hole en la interfaz web de Zabbix.



Nota. Registro exitoso del servidor Pi-hole dentro del inventario de monitoreo. Elaboración propia.

Esperando unos segundos, se verifica que el nuevo *host* aparezca en la lista de *hosts* en la interfaz web de Zabbix.

Figura 133. Verificación de disponibilidad del segundo *host* Pi-hole en la interfaz web de Zabbix.

Nota. Disponibilidad operativa del segundo *host* añadido confirmando conectividad estable. Elaboración propia.

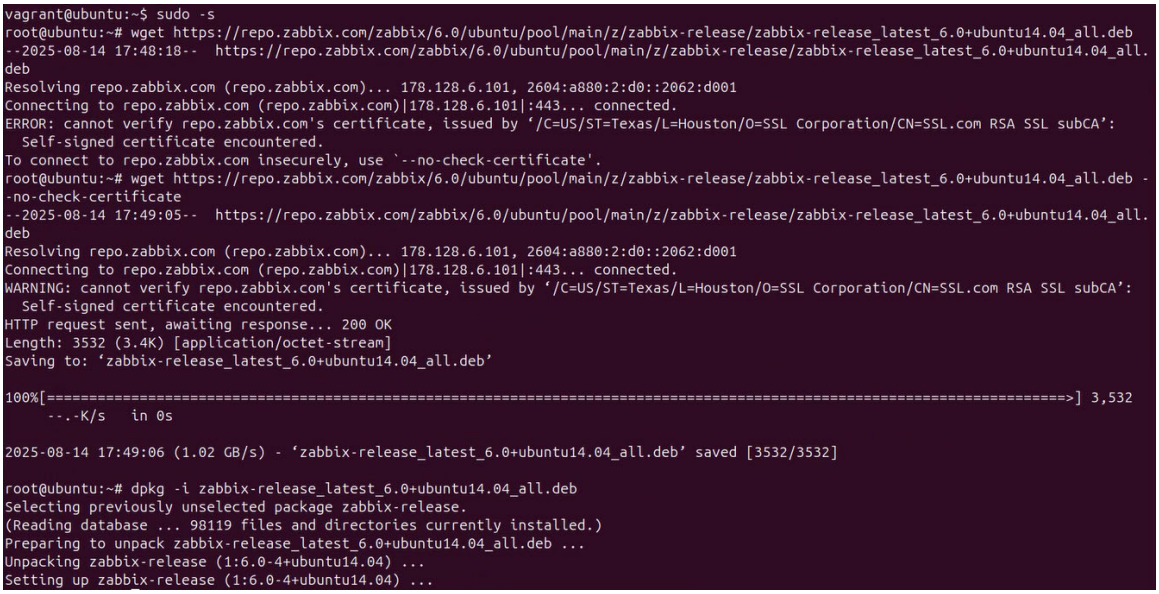
Para el caso de Metasploitable 3, se debe tener en cuenta que esta distribución está basada en Ubuntu *Server* 14.04, y Zabbix *Agent* 2 no es compatible. La última versión compatible de Zabbix *Agent* para Ubuntu *Server* 14.04 es la 6.0, por lo que se debe instalar esta versión.

Cuadro 32. Instalación del repositorio de Zabbix en Ubuntu 14.04.

```
sudo -s
wget https://repo.zabbix.com/zabbix/6.0/ubuntu/pool/main/z/zabbix-release/
    zabbix-release_latest_6.0+ubuntu14.04_all.deb --no-check-certificate
dpkg -i zabbix-release_latest_6.0+ubuntu14.04_all.deb
apt update
```

Nota. Instalación del repositorio Zabbix 6.0 compatible con Ubuntu 14.04 base de Metasploitable 3 seguida de actualización de índices APT. Elaboración propia.

Figura 134. Instalación del repositorio de Zabbix en Metasploitable 3 (Ubuntu 14.04).



Nota. Uso de repositorio 6.0 para compatibilidad con base Ubuntu 14.04 de Metasploitable 3. Elaboración propia.

Lo segundo es instalar el Zabbix *Agent*. Para ello, se ejecuta el siguiente comando en la terminal:

Cuadro 33. Descarga e instalación del Zabbix *Agent* en Metasploitable 3 (Ubuntu 14.04).

```
1 apt install zabbix-agent
```

Nota. Instalación del agente clásico de Zabbix 6.0 compatible con arquitectura heredada de Ubuntu 14.04 en Metasploitable 3. Elaboración propia.

Figura 135. Confirmación de instalación del Zabbix *Agent* en Metasploitable 3.

```
root@ubuntu:~# apt install zabbix-agent
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  zabbix-agent
0 upgraded, 1 newly installed, 0 to remove and 88 not upgraded.
Need to get 160 kB of archives.
After this operation, 770 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu/ trusty/universe zabbix-agent amd64 1:2.2.2+dfsg-1ubuntu1 [160 kB]
Fetched 160 kB in 0s (303 kB/s)
Selecting previously unselected package zabbix-agent.
(Reading database ... 98125 files and directories currently installed.)
Preparing to unpack .../zabbix-agent_1%3a2.2.2+dfsg-1ubuntu1_amd64.deb ...
Unpacking zabbix-agent (1:2.2.2+dfsg-1ubuntu1) ...
Processing triggers for man-db (2.6.7.1-1) ...
Processing triggers for ureadahead (0.100.0-16) ...
ureadahead will be reprofiled on next reboot
Setting up zabbix-agent (1:2.2.2+dfsg-1ubuntu1) ...

Creating config file /etc/zabbix/zabbix_agentd.conf with new version
zabbix-agent start/running, process 2717
Processing triggers for ureadahead (0.100.0-16) ...
```

Nota. Instalación del agente clásico adaptado a limitaciones del sistema heredado. Elaboración propia.

Por último se inicia el servicio del Zabbix *Agent* para que comience a monitorear el sistema. Esto se puede hacer ejecutando el siguiente comando en la terminal:

Cuadro 34. Comando para iniciar el servicio Zabbix *Agent* en Metasploitable 3.

```
service zabbix-agent start
update-rc.d zabbix-agent enable
```

Nota. Inicio del servicio del agente e incorporación al sistema de arranque automático en Metasploitable 3. Elaboración propia.

Figura 136. Confirmación de iniciación del Zabbix *Agent* en Metasploitable 3.

```
root@ubuntu:~# service zabbix-agent start
start: Job is already running: zabbix-agent
root@ubuntu:~# update-rc.d zabbix-agent enable
update-rc.d: warning:  start runlevel arguments (none) do not match zabbix-agent Default-Start values (2 3 4 5)
update-rc.d: warning:  stop runlevel arguments (none) do not match zabbix-agent Default-Stop values (0 1 6)
 System start/stop links for /etc/init.d/zabbix-agent do not exist.
root@ubuntu:~# service zabbix-agent status
zabbix-agent start/running, process 2717
```

Nota. Servicio del agente iniciado y añadido al arranque en distribución antigua. Elaboración propia.

Ahora se configura el agente para que se conecte al servidor de Zabbix. Para ello, se edita el archivo de configuración del agente que se encuentra en `/etc/zabbix/zabbix_agentd.conf`. Para ello, se ejecuta el siguiente comando en la terminal:

Cuadro 35. Comando para editar el archivo de configuración del Zabbix *Agent* en Metasploitable 3.

```
nano /etc/zabbix/zabbix_agentd.conf
```

Nota. Apertura del archivo de configuración del agente para establecer dirección del servidor y *hostname* del cliente. Elaboración propia.

En este archivo se busca la línea `Server=` y se agrega la dirección IP del servidor de Zabbix, en este caso, 172.16.10.1. También, se busca la línea `Hostname=` y se agrega el nombre del *host* que se está monitoreando, en este caso, Ubuntu *Desktop* o Ubuntu *Server* según corresponda. Se puede obtener el *hostname* ejecutando el comando `hostname` en la terminal.

Cuadro 36. Configuración del Zabbix *Agent* 6.0 en Metasploitable 3 (Ubuntu 14.04).

```
### Option: Server
#       List of comma delimited IP addresses (or hostnames) of Zabbix servers.
#       Incoming connections will be accepted only from the hosts listed here.
#       If IPv6 support is enabled then '127.0.0.1', '::127.0.0.1', '::ffff:127.0.0.1' are treated equally.
#
# Mandatory: no
# Default:
# Server=

Server=172.16.10.2



### Option: Hostname
#       Unique, case sensitive hostname.
#       Required for active checks and must match hostname as configured on the server.
#       Value is acquired from HostnameItem if undefined.
#
# Mandatory: no
# Default:
# Hostname=

Hostname=ubuntu
```

Nota. Parámetros esenciales del agente especificando dirección del servidor y nombre único del *host* cliente. Elaboración propia.

Para que los cambios tomen efecto, se debe reiniciar el servicio del agente de Zabbix. Esto se puede hacer ejecutando el siguiente comando en la terminal:

Cuadro 37. Reinicio del Zabbix *Agent* para aplicar cambios en Metasploitable 3.

```
service zabbix-agent restart
exit
```

Nota. Reinicio del servicio para aplicar cambios de configuración en Metasploitable 3. Elaboración propia.

Antes de crear el *host* en la interfaz web de Zabbix, se hace un cambio más. En esta máquina virtual de Metasploitable 3, se tienen creadas ciertas reglas de *firewall* con *iptables* que bloquean todo el tráfico si no cumplen con ciertos criterios. Para ver las reglas se ejecuta el siguiente comando:

Cuadro 38. Comando para la visualización de las reglas de *iptables* en Metasploitable 3.

```
sudo iptables -L -n -v --line-numbers
```

Nota. Listado de reglas de *firewall* de Metasploitable 3 con números de línea para identificar e insertar nuevas reglas permitiendo puerto 10050 del agente. Elaboración propia.

Al ejecutarlo en la terminal de Metasploitable 3, se pueden ver las reglas de *iptables* configuradas.

Cuadro 39. Visualización de las reglas de *iptables* en Metasploitable 3.

```
vagrant@ubuntu:~$ sudo iptables -L -n -v --line-numbers
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1    19772 3895K ACCEPT     all  --  lo     *       0.0.0.0/0            0.0.0.0/0
2     2583 1540K ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           ctstate RELATED,ESTABLISHED
3        0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:8080
4        2   168 ACCEPT     icmp --  *      *       0.0.0.0/0            0.0.0.0/0
5        2   120 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:22
6        0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:80
7        0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:3000
8        0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:80
9        0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:631
10       0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:3306
11       0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:21
12       0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:3500
13       0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:445
14       0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:8181
15       0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:6697
16    1140  210K DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:8989

Chain OUTPUT (policy ACCEPT 2285 packets, 308K bytes)
num   pkts bytes target     prot opt in     out     source               destination
1    19772 3895K ACCEPT     all  --  *      lo      0.0.0.0/0            0.0.0.0/0
```

Nota. Listado de reglas de *firewall* en Metasploitable 3 mostrando puertos aceptados y línea *DROP* final que bloquea tráfico no autorizado. Elaboración propia.

En la línea 16 de *INPUT*, se observa que hay una regla *DROP all*, lo que significa que se bloquean todos los paquetes que no coinciden con las reglas anteriores. Por lo tanto, para que el Zabbix *Agent* pueda comunicarse con el servidor de Zabbix, se agrega una regla que permite el tráfico del puerto 10050, que es el puerto por defecto que utiliza el Zabbix *Agent* para comunicarse con el servidor. Para ello, se ejecutan los siguientes comandos en la terminal:

Cuadro 40. Reglas de *iptables* para permitir el tráfico del Zabbix *Agent*.

```
1 sudo iptables -I INPUT -p tcp --dport 10050 -s 172.16.10.2 -j ACCEPT
2 sudo apt-get install -y iptables-persistent
3 sudo sh -c 'iptables-save > /etc/iptables/rules.v4'
```

Nota. Adición de regla *firewall* para permitir puerto 10050 del agente y persistencia de la configuración entre reinicios. Elaboración propia.

Al volver a ejecutar el comando para ver las reglas de *iptables*, se puede observar que se ha agregado una nueva regla en la parte superior, que permite el tráfico del puerto 10050 con IP de origen 172.16.10.2.

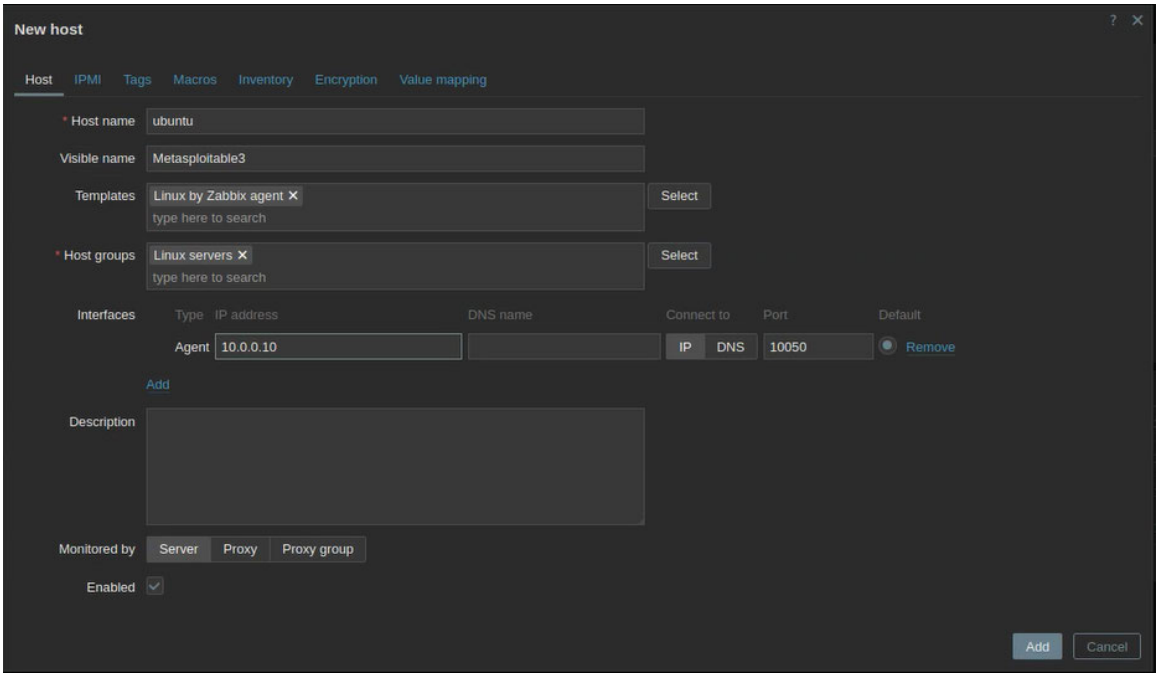
Cuadro 41. Visualización de las reglas de *iptables* en Metasploitable 3 actualizadas.

```
1 Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
2 num   pkts bytes target     prot opt in     out     source
      destination
3 1      301 18145 ACCEPT     tcp  --  *      *       172.16.10.2
      0.0.0.0/0            tcp dpt:10050
```

Nota. Regla de *firewall* nuevamente insertada en primera posición permitiendo comunicación del servidor Zabbix con el agente. Elaboración propia.

De nueva forma se crea un nuevo *host* en la interfaz web de Zabbix para monitorear Metasploitable 3. En este caso, como el *hostname* era ubuntu, en la sección *Visible name* se ingresa Metasploitable 3.

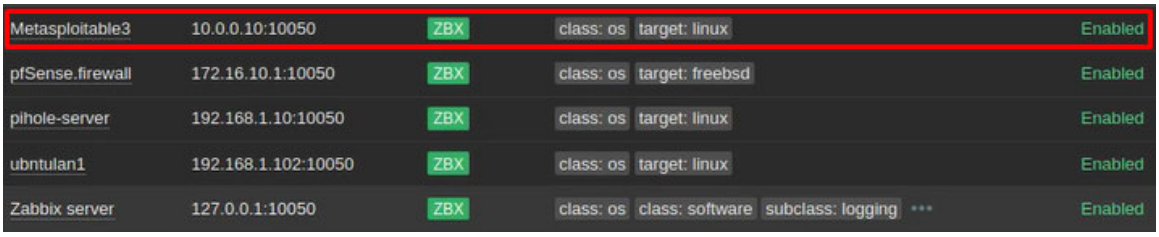
Figura 137. Creación de un nuevo *host* en la interfaz web de Zabbix para Metasploitable 3.



Nota. Alta del nuevo *host* Metasploitable 3 diferenciando nombre visible para claridad en reportes. Elaboración propia.

Se deben esperar unos segundos para que los cambios tomen efecto, y luego se debe verificar que el nuevo *host* aparezca en la lista de *hosts* en la interfaz web de Zabbix.

Figura 138. Verificación de disponibilidad del *host* Metasploitable 3 en Zabbix.



Nota. *Host* Metasploitable 3 disponible evidenciando conectividad pese a versión antigua del agente. Elaboración propia.

7.6.5.2. Instalación del Zabbix *Agent* en Fedora

Para instalar el Zabbix *Agent* en Fedora, primero se deshabilitan los paquetes que Fedora proporciona, ya que estos paquetes pueden entrar en conflicto con los paquetes que Zabbix proporciona. Para ello, se ejecutan los siguientes comandos en la terminal:

Cuadro 42. Comando para editar el archivo de configuración EPEL en Fedora.

```
sudo -s
nano /etc/yum.repos.d/epel.repo
```

Nota. Apertura del archivo de configuración del repositorio EPEL para mantenerlo segregado del repositorio de Zabbix. Elaboración propia.

En el archivo se agrega el siguiente contenido:

Cuadro 43. Configuración para excluir paquetes de Zabbix en Fedora.

```
[fedora]
...
excludepkgs=zabbix*
```

Nota. Directiva para excluir paquetes de Zabbix del repositorio EPEL evitando conflictos con el repositorio oficial. Elaboración propia.

Ahora se procede con la instalación del repositorio de Zabbix. Para ello, se ejecutan los siguientes comandos en la terminal:

Cuadro 44. Instalación del Zabbix *Agent* en Fedora.

```
rpm -Uvh https://repo.zabbix.com/zabbix/7.4/release/rhel/10/noarch/zabbix-
    release-latest-7.4.el10.noarch.rpm
sudo dnf clean all
```

Nota. Adición del repositorio oficial de Zabbix 7.4 compatible con Fedora para acceso a paquetes actualizados. Elaboración propia.

Figura 139. Instalación del repositorio de Zabbix en Fedora.

Nota. Incorporación del repositorio oficial 7.4 para obtener paquetes actualizados del agente en Fedora. Elaboración propia.

Luego se instala el Zabbix *Agent* 2 y sus respectivos *plugins*. Para ello, se ejecutan los siguientes comandos en la terminal:

Cuadro 45. Instalación de paquetes Zabbix *Agent* 2 y *plugins* en Fedora.

```
dnf install zabbix-agent2 --exclude=zabbix-agent
dnf install zabbix-agent2-plugin-mongodb zabbix-agent2-plugin-mssql zabbix-
    agent2-plugin-postgresql
```

Nota. Instalación del agente 2 en Fedora con exclusión del agente antiguo e inclusión de *plugins* para bases de datos. Elaboración propia.

Figura 140. Proceso de instalación del Zabbix *Agent* 2 y *plugins* en Fedora.

Nota. Instalación del agente 2 y módulos complementarios sobre entorno Fedora segregado. Elaboración propia.

Por último, se inicia el servicio del Zabbix *Agent* 2 para que comience a monitorear el sistema. Esto se puede hacer ejecutando el siguiente comando en la terminal:

Cuadro 46. Comando para iniciar el servicio Zabbix *Agent* 2 en Fedora.

```
systemctl restart zabbix-agent2
systemctl enable zabbix-agent2
```

Nota. Reinicio del servicio para aplicar configuración y habilitación para arranque automático en Fedora. Elaboración propia.

Figura 141. Confirmación de arranque del servicio Zabbix *Agent* 2 en Fedora.

Nota. Servicio levantado y habilitado para arranque automático asegurando monitoreo continuo. Elaboración propia.

Ahora se edita el archivo de configuración del agente que se encuentra en `/etc/zabbix/zabbix_agent2.conf`. Para ello, se ejecuta el siguiente comando en la terminal:

Cuadro 47. Edición del archivo de configuración del Zabbix *Agent* 2 en Fedora.

```
nano /etc/zabbix/zabbix_agent2.conf
```

Nota. Apertura del archivo de configuración del agente para establecer parámetros de conexión con el servidor. Elaboración propia.

Se agregan las siguientes líneas al archivo de configuración:

Cuadro 48. Configuración del Zabbix *Agent* 2 en Fedora.

```
Server=172.16.10.2
Hostname=fedora
```

Nota. Parámetros esenciales del agente especificando dirección del servidor y nombre único del *host* cliente para Fedora. Elaboración propia.

Se debe reiniciar el servicio del agente de Zabbix para que los cambios tomen efecto. Esto se puede hacer ejecutando el siguiente comando en la terminal:

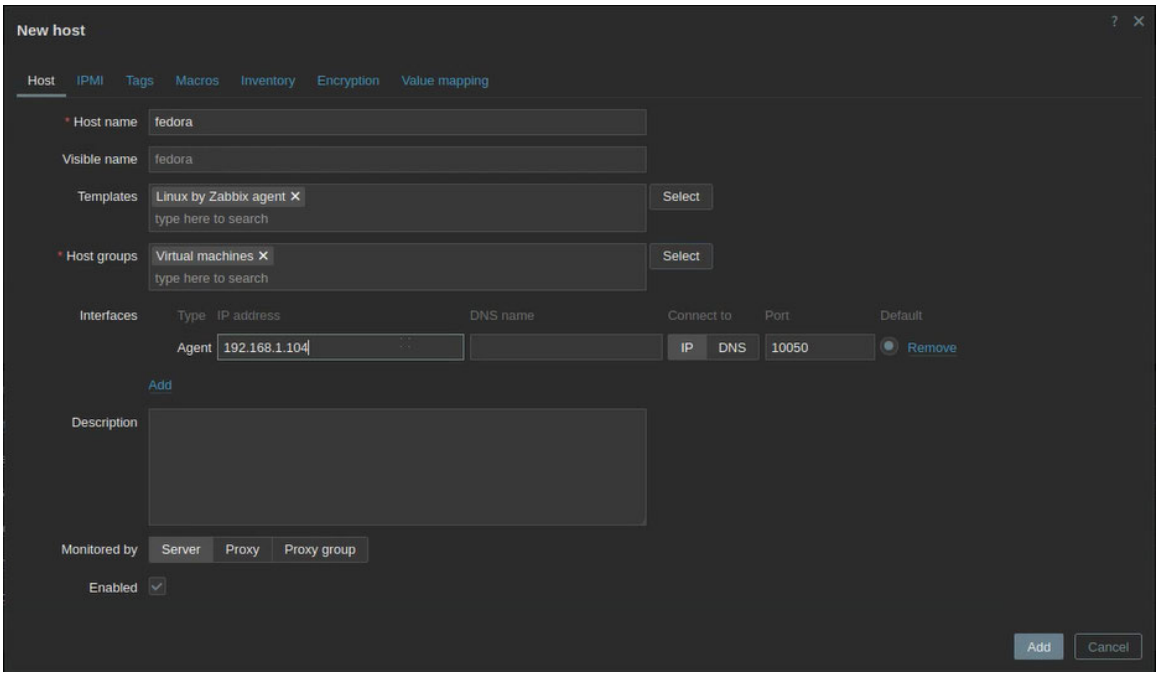
Cuadro 49. Reinicio del servicio Zabbix *Agent* 2 en Fedora.

```
systemctl restart zabbix-agent2
exit
```

Nota. Reinicio del servicio para aplicar nuevos parámetros de configuración en Fedora. Elaboración propia.

Luego, se crea un nuevo *host* en la interfaz web de Zabbix para monitorear Fedora.

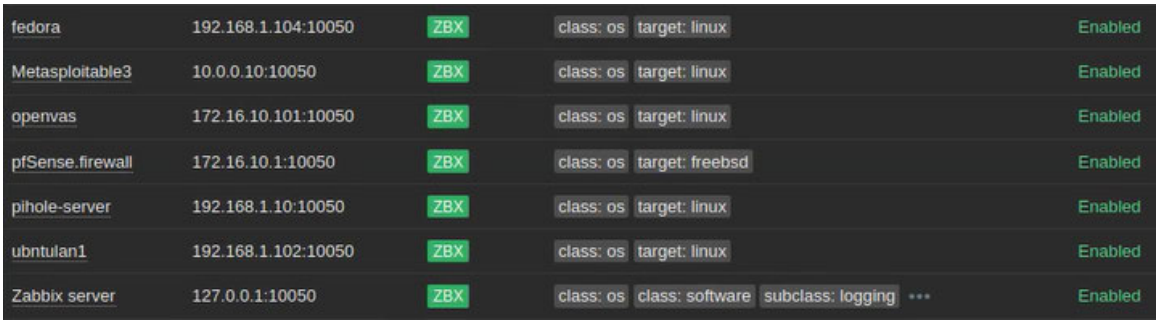
Figura 142. Creación de un nuevo *host* en Zabbix para Fedora.



Nota. Registro del *host* Fedora con plantilla Linux para ampliar cobertura de monitoreo heterogéneo. Elaboración propia.

Se deben esperar unos segundos para que los cambios tomen efecto, y luego se debe verificar que el nuevo *host* aparezca en la lista de *hosts* en la interfaz web de Zabbix.

Figura 143. Verificación del nuevo *host* en la interfaz web de Zabbix para Fedora.



Nota. Estado operativo del *host* Fedora mostrando disponibilidad y recepción de métricas. Elaboración propia.

7.6.5.3. Instalación del Zabbix *Agent* en Kali Linux

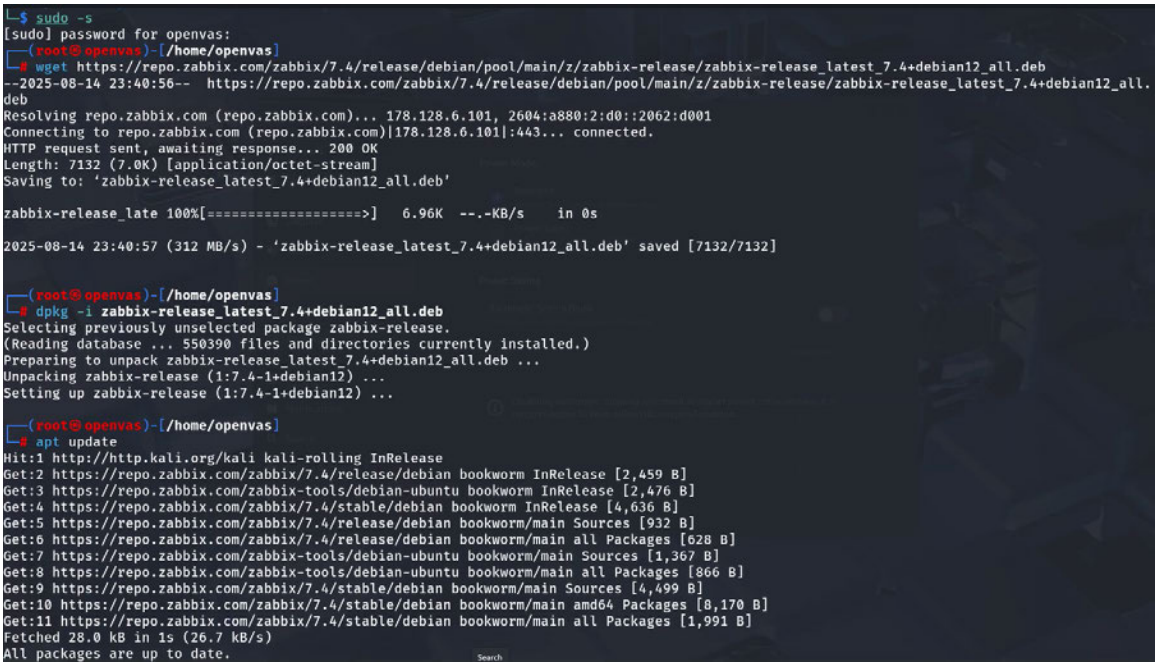
Para instalar el Zabbix *Agent* en Kali Linux, primero se instala el repositorio de Zabbix. Para ello, se ejecutan los siguientes comandos en la terminal:

Cuadro 50. Instalación del Zabbix *Agent* en Kali Linux.

```
sudo -s
wget https://repo.zabbix.com/zabbix/7.4/release/debian/pool/main/z/zabbix-release/zabbix-release_latest_7.4+debian12_all.deb
dpkg -i zabbix-release_latest_7.4+debian12_all.deb
apt update
```

Nota. Adición del repositorio Zabbix 7.4 compatible con Kali para acceso a paquetes del agente. Elaboración propia.

Figura 144. Instalación del repositorio de Zabbix en Kali Linux.



Nota. Adición del repositorio 7.4 en Kali para desplegar agente y *plugins* compatibles. Elaboración propia.

Segundo se instala el Zabbix *Agent* 2 y sus respectivos *plugins*. Para ello, se ejecutan los siguientes comandos en la terminal:

Cuadro 51. Instalación del Zabbix *Agent* 2 en Kali Linux.

```
apt install zabbix-agent2
apt install zabbix-agent2-plugin-mongodb zabbix-agent2-plugin-mssql zabbix-agent2-plugin-postgresql
```

Nota. Instalación del agente 2 en Kali Linux con módulos de bases de datos para escaneo integrado OpenVAS. Elaboración propia.

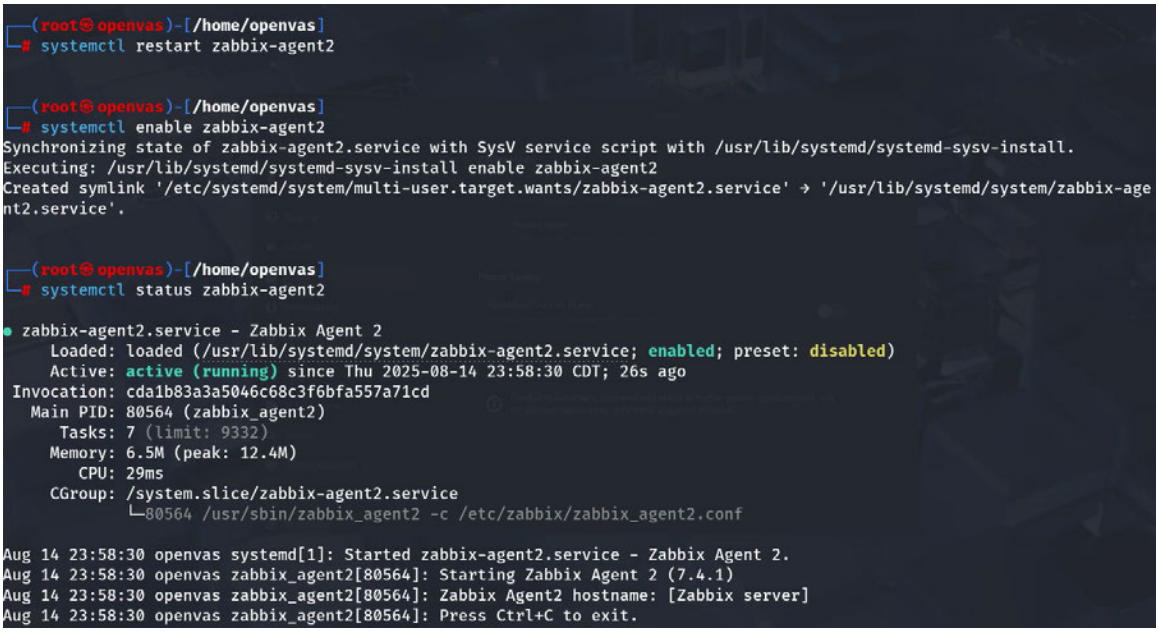
Se inicia el servicio del Zabbix *Agent* 2 para que comience a monitorear el sistema. Esto se puede hacer ejecutando el siguiente comando en la terminal:

Cuadro 52. Comando para iniciar el servicio Zabbix *Agent* 2 en Kali Linux.

```
systemctl restart zabbix-agent2
systemctl enable zabbix-agent2
```

Nota. Inicio del servicio del agente y habilitación en arranque automático de Kali Linux. Elaboración propia.

Figura 145. Confirmación de arranque del servicio Zabbix *Agent* 2 en Kali Linux.



Nota. Servicio del agente habilitado en Kali para integrar nodo de evaluación de vulnerabilidades al monitoreo. Elaboración propia.

Se modifica el archivo de configuración del agente que se encuentra en `/etc/zabbix/zabbix_agent2.conf`. Para ello, se ejecuta el siguiente comando en la terminal:

Cuadro 53. Modificación del archivo de configuración del Zabbix *Agent* 2 en Kali Linux.

```
nano /etc/zabbix/zabbix_agent2.conf
```

Nota. Apertura del archivo de configuración del agente en Kali para establecer parámetros de conexión. Elaboración propia.

Se modifica la línea `Server=` y se agrega la dirección IP del servidor de Zabbix, y la línea `Hostname=` y se agrega el nombre del *host* que se está monitoreando.

Cuadro 54. Configuración del Zabbix *Agent* 2 en Kali Linux.

```
Server=172.16.10.2
Hostname=openvas
```

Nota. Parámetros de configuración especificando servidor central y nombre de *host* OpenVAS en Kali. Elaboración propia.

Se debe reiniciar el servicio del agente de Zabbix para que los cambios tomen efecto. Esto se puede hacer ejecutando el siguiente comando en la terminal:

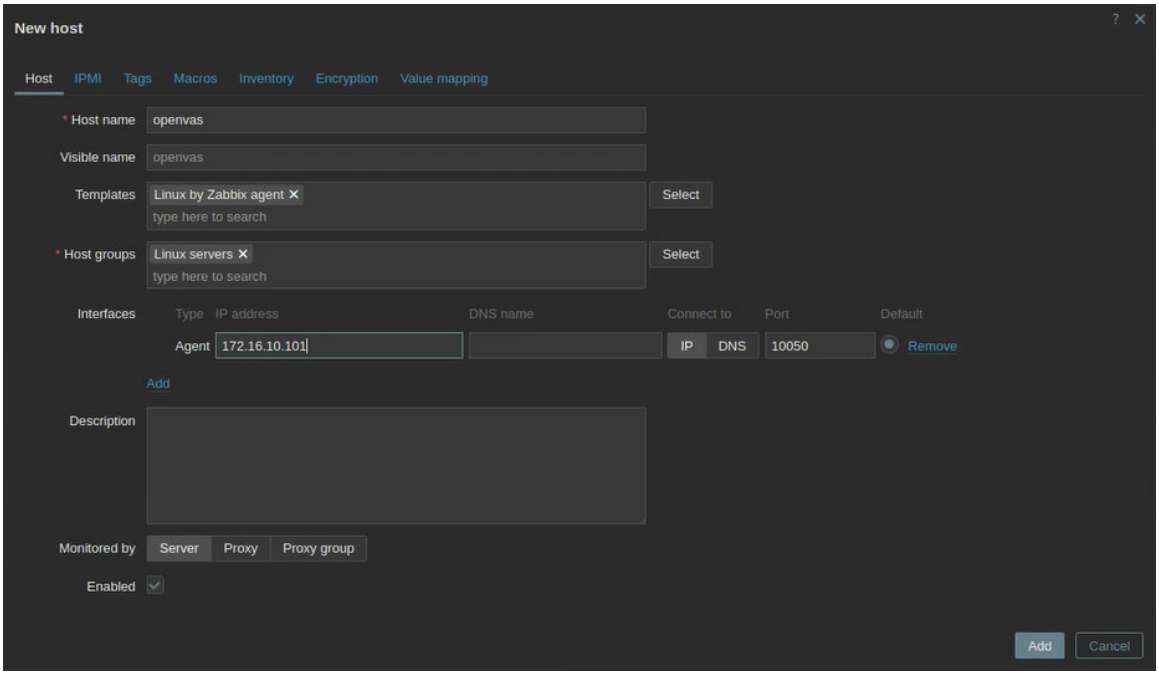
Cuadro 55. Reinicio del servicio Zabbix *Agent* 2 en Kali Linux.

```
systemctl restart zabbix-agent2
exit
```

Nota. Reinicio del servicio para aplicar cambios de configuración en Kali Linux. Elaboración propia.

Ahora se crea el *host* en la interfaz web de Zabbix.

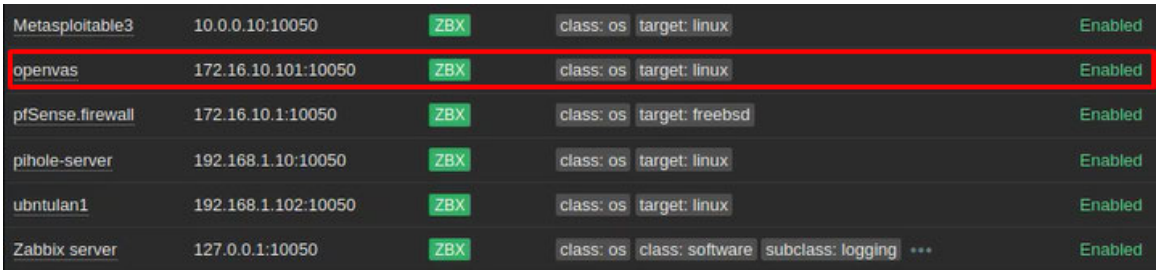
Figura 146. Creación de un nuevo *host* en la interfaz web de Zabbix para Kali Linux.



Nota. Registro del *host* Kali orientado a labores de escaneo integrado al inventario monitoreado. Elaboración propia.

Si todo se configuró correctamente, el nuevo *host* debería aparecer en la interfaz web de Zabbix y estar listo para ser monitoreado.

Figura 147. Verificación del nuevo *host* en la interfaz web de Zabbix para Kali Linux.



Nota. *Host* Kali reportando disponibilidad confirmando recepción de chequeos pasivos. Elaboración propia.

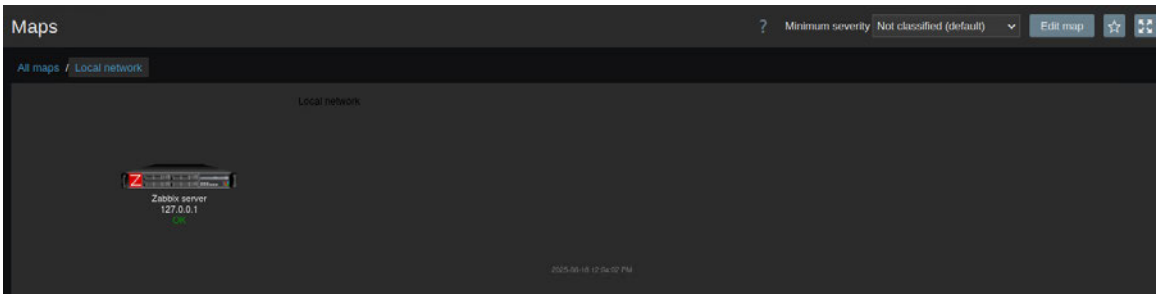
7.6.5.4. Personalización de Zabbix para una mayor facilidad de monitoreo de la red

Para personalizar Zabbix y facilitar el monitoreo de la red, se pueden crear plantillas que agrupen los elementos de monitoreo comunes a varios *hosts*. Esto permite aplicar configuraciones de manera más eficiente y mantener una estructura organizada.

7.6.5.5. Creación de mapa de la red en Zabbix

Para crear un mapa de la red en Zabbix, se va a la sección *Monitoring* y se selecciona la opción *Maps*. Desde allí, se selecciona el que tiene nombre de *local network*, ya que es el que se crea por defecto y se puede editar.

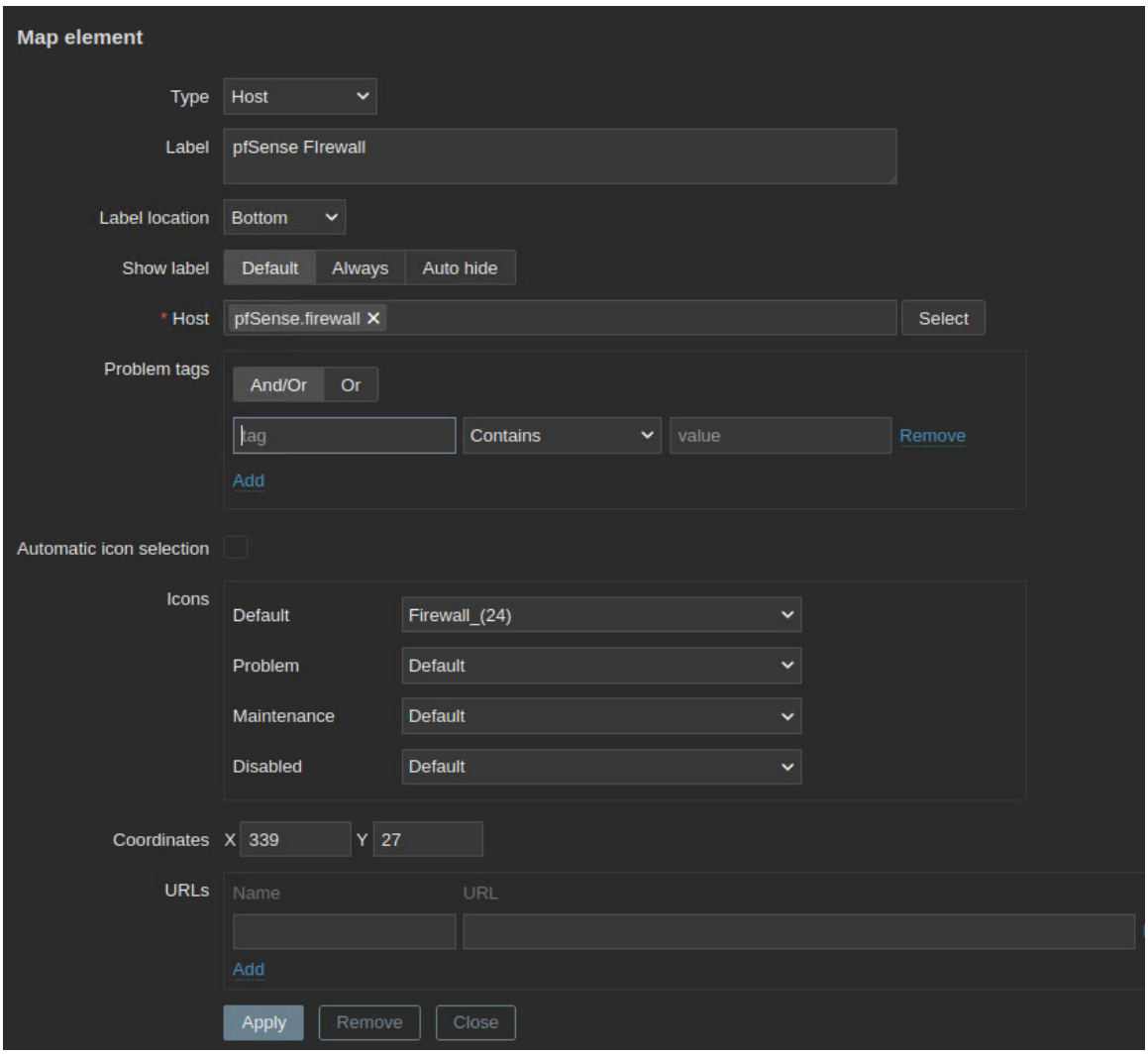
Figura 148. Mapa de la red en la interfaz web de Zabbix.



Nota. Visualización inicial del mapa lógico para relacionar dispositivos monitoreados. Elaboración propia.

Para poder editarlo, se hace clic en el botón de *Edit* en la esquina superior derecha del mapa. Lo primero, es agregar todos los *hosts* que se deseen incluir en el mapa. Esto se puede hacer haciendo clic en el botón de *Add* en la sección *Map Element*. En este caso, se añade el *host* de pfSense.

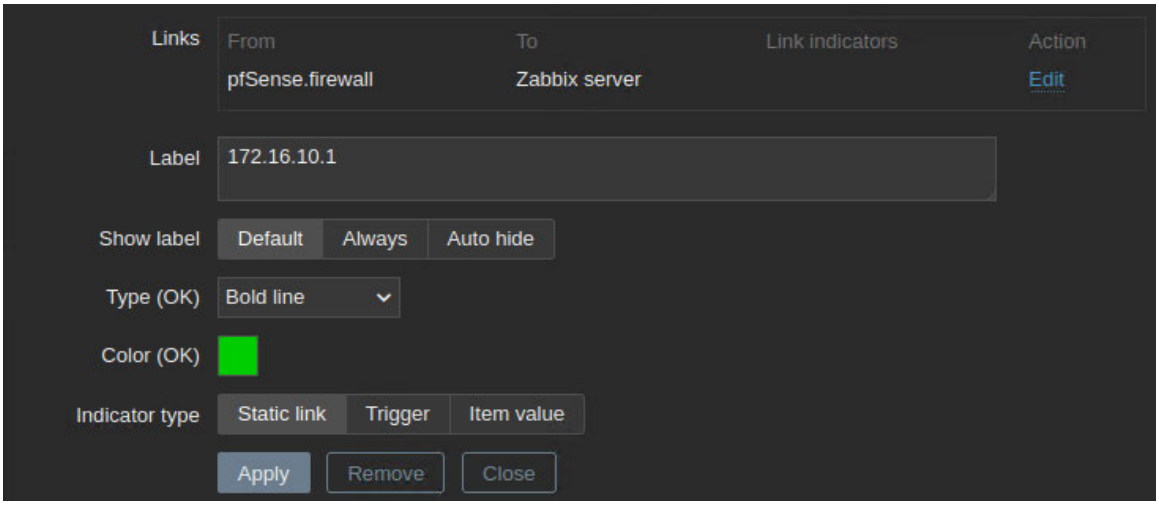
Figura 149. Añadiendo el *host* pfSense al mapa en la interfaz web de Zabbix.



Nota. Inserción del nodo pfSense para reflejar rol de *firewall* perimetral en el mapa. Elaboración propia.

Luego se añade un *link* entre el *host* de pfSense y el de Zabbix *Server*. Esto se puede hacer haciendo seleccionando ambos dispositivos y haciendo clic en el botón de *Add* en la sección *Link*.

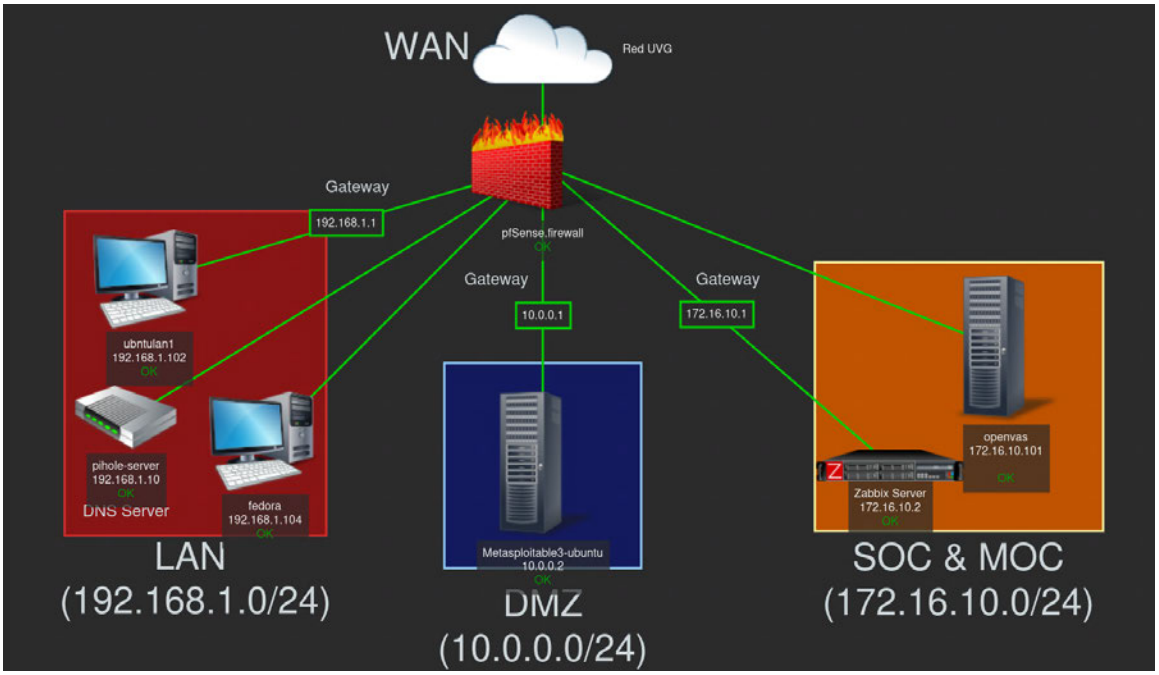
Figura 150. Añadiendo un link entre pfSense y Zabbix *Server* en la interfaz web de Zabbix.



Nota. Creación del enlace lógico representando canal de monitoreo entre *firewall* y servidor. Elaboración propia.

Con el enlace creado, se puede visualizar la relación entre los dispositivos en el mapa de la red. Con estos elementos se puede formar un mapa básico de la red que muestra los dispositivos y sus conexiones. Este mapa se vería así:

Figura 151. Mapa consolidado de la red en la interfaz web de Zabbix.



Nota. Mapa consolidado mostrando enlaces y disposición básica de nodos críticos. Elaboración propia.

Con el mapa creado, este se puede añadir al *dashboard* de Zabbix para una visualización más fácil y accesible.

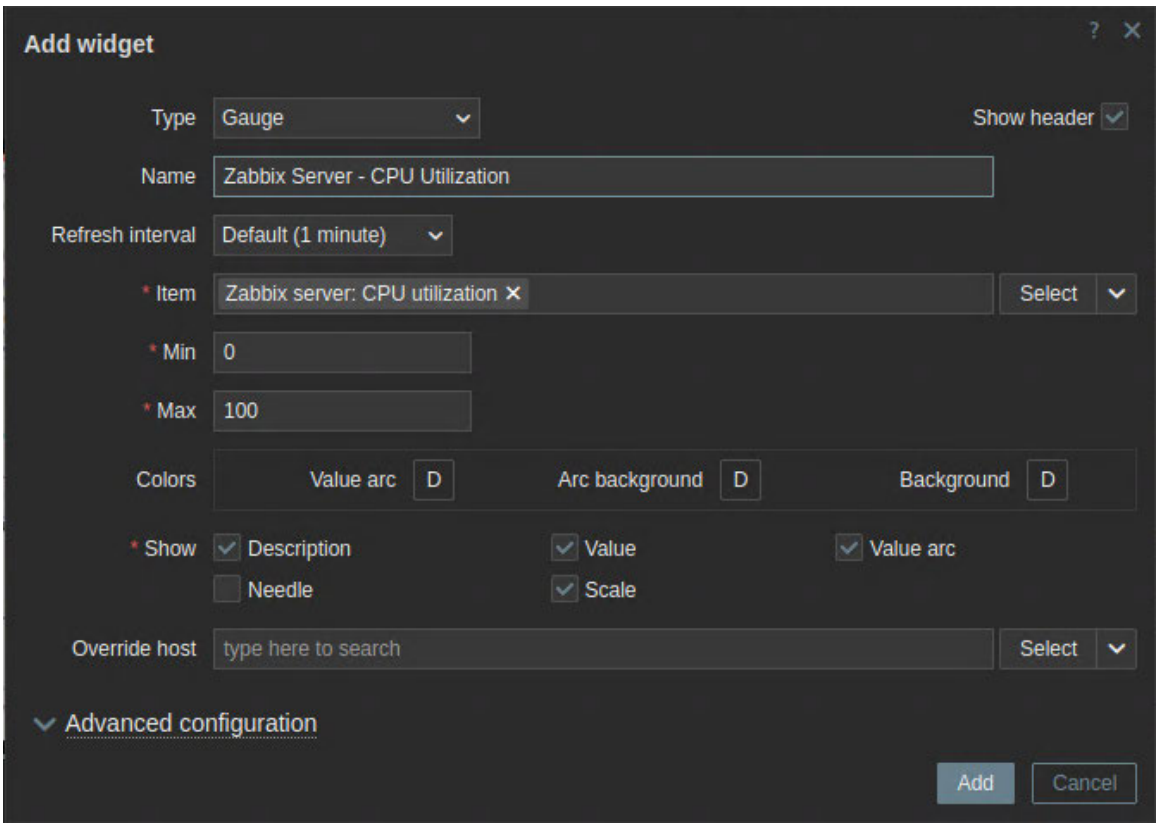
7.6.5.6. Personalización del *dashboard* en Zabbix

Para personalizar el *dashboard* en Zabbix, se va a la sección *Dashboard* en el menú principal. Desde allí, se pueden agregar diferentes *widgets* para mostrar información relevante sobre el estado de la red y los dispositivos monitoreados. Lo primero a realizar es añadir más páginas al *dashboard*. En este caso, se crean las siguientes páginas:

- *Overview*: para mostrar un resumen del estado de la red y de los *hosts* de la misma.
- pfSense: para mostrar información específica sobre el estado del *firewall* y de la interfaz *WAN*.
- Pi-hole: para mostrar información sobre el estado de Pi-hole.
- *LAN*: para mostrar información sobre el estado de la red LAN y de los *hosts* conectados a ella.
- *DMZ*: para mostrar información sobre el estado de la zona *DMZ* y de el servidor Metasploitable 3.
- *SOC & MOC*: para mostrar información sobre el estado del segmento *SOC & MOC*, así como información del Zabbix *Server*.

Para poder visualizar la información de manera más efectiva, se pueden utilizar diferentes tipos de *widgets*, como gráficos, tablas y mapas. Para agregar un *widget*, se debe hacer clic en el botón de *Edit dashboard* en la esquina superior derecha del *dashboard*. Luego, se debe seleccionar la página donde se desea agregar el *widget* y hacer clic en el botón de *Add widget*. Esto abre una ventana para configurar el tipo de *widget* que se quiera agregar. En la sección *type* se puede seleccionar el tipo de *widget* a insertar, en este se usa como ejemplo el tipo *gauge* y *graph*. En la sección *Name* se escribe el nombre del *widget*. En la sección *Refresh interval* se selecciona cada cuánto tiempo se actualizará el *widget*. En la sección *Item* se selecciona el ítem que se desea mostrar en el *widget*. En este caso, el ítem que muestra la utilización de CPU del *host* Zabbix *Server*.

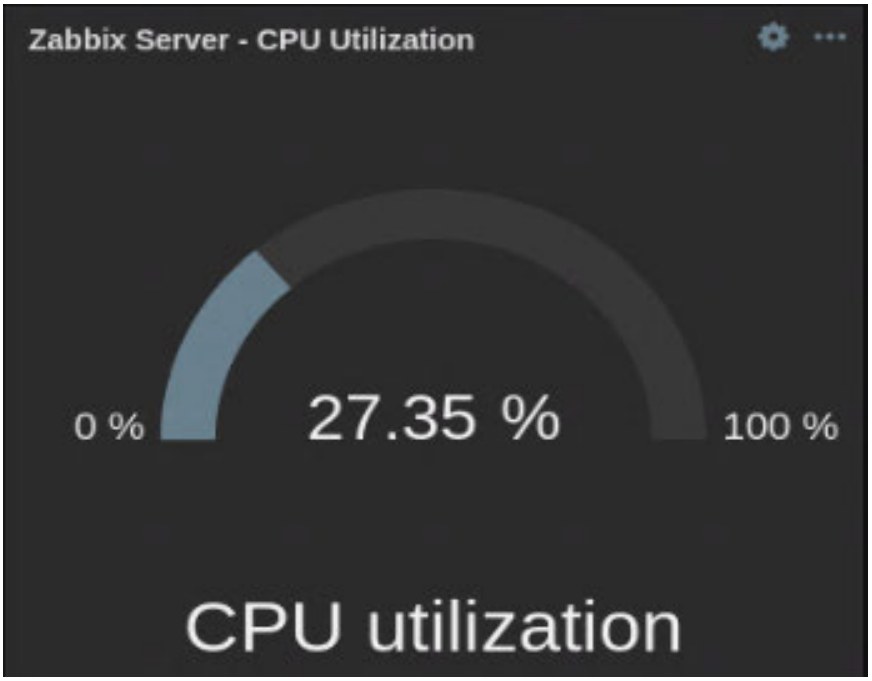
Figura 152. Configuración inicial de un *widget* en la interfaz web de Zabbix.



Nota. Creación de *widget* tipo gauge/graph para observabilidad rápida de recursos críticos. Elaboración propia.

Esto crea un *widget* que muestra la utilización de CPU del *host* Zabbix *Server* en forma de gráfico.

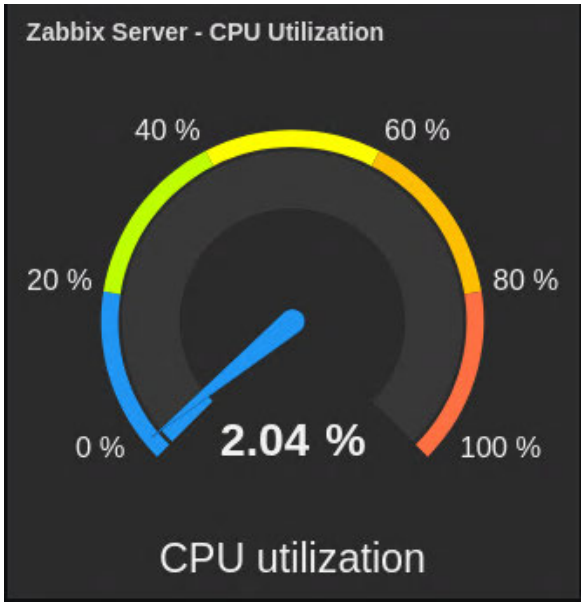
Figura 153. *Widget* de utilización de CPU del *host* Zabbix *Server* en la interfaz web de Zabbix.



Nota. Visualización del *widget* mostrando uso de CPU en tiempo real. Elaboración propia.

Estos mismo *widget* se puede editar para que sea más amigables y fáciles de interpretar. Se pueden cambiar los colores, las etiquetas y la disposición de los elementos para que se adapten mejor a las necesidades del usuario.

Figura 154. Edición y personalización de un *widget* en la interfaz web de Zabbix.



Nota. Creación de *widget* tipo *gauge/graph* para observabilidad rápida de recursos críticos. Elaboración propia.

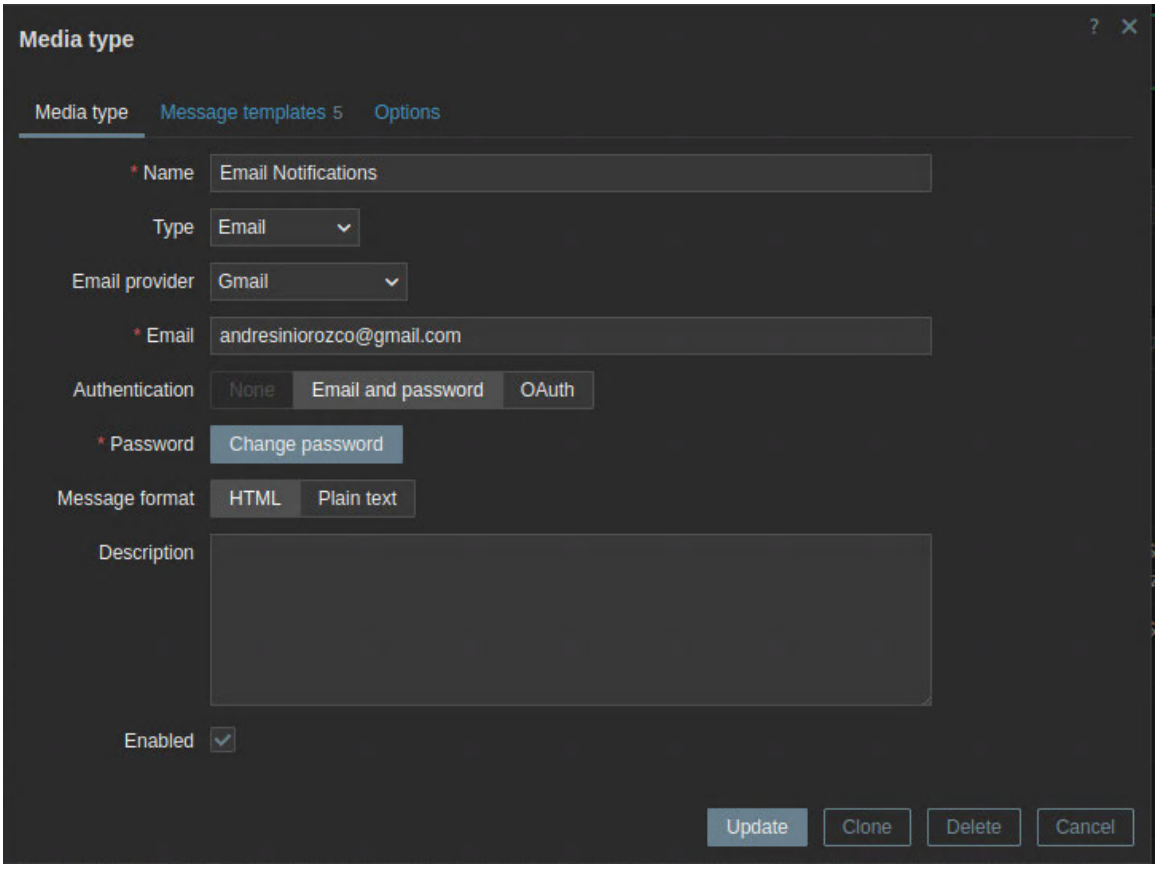
Se pueden utilizar todo tipo de *widgets* para visualizar diferentes métricas y datos relevantes. En el anexo 12.1 se presentan ejemplos de *dashboards* utilizados en este proyecto.

7.6.5.7. Configuración de alertas en Zabbix

Para configurar alertas en Zabbix, se debe ir a la sección *Alerts* en el menú principal y seleccionar la opción *Media Types*. Desde allí, se pueden crear nuevos tipos de medios para definir cómo se deben enviar las alertas generadas por el sistema. Se pueden configurar diferentes métodos de notificación, como correo electrónico, SMS o mensajería instantánea.

En este caso, se configura un tipo de medio para enviar notificaciones por correo electrónico, por lo que se debe seleccionar la opción *Email* y proporcionar la información necesaria. Lo primero es el nombre del medio, que puede ser algo como "Notificaciones por correo electrónico". Luego, se debe seleccionar el tipo que es *Email*. Luego, se debe seleccionar el proveedor de correo electrónico que se utiliza para enviar las notificaciones. En este caso, Gmail. En la sección *Email* se debe proporcionar la dirección de correo electrónico que se utiliza para enviar las notificaciones. En *Authentication* se debe seleccionar la opción *Email and password*, y en los campos *Email* y *Password* se debe proporcionar el correo electrónico y la contraseña que se generó para la aplicación de Zabbix. Para habilitar el uso de contraseñas de aplicaciones referirse al Anexo 12.2.

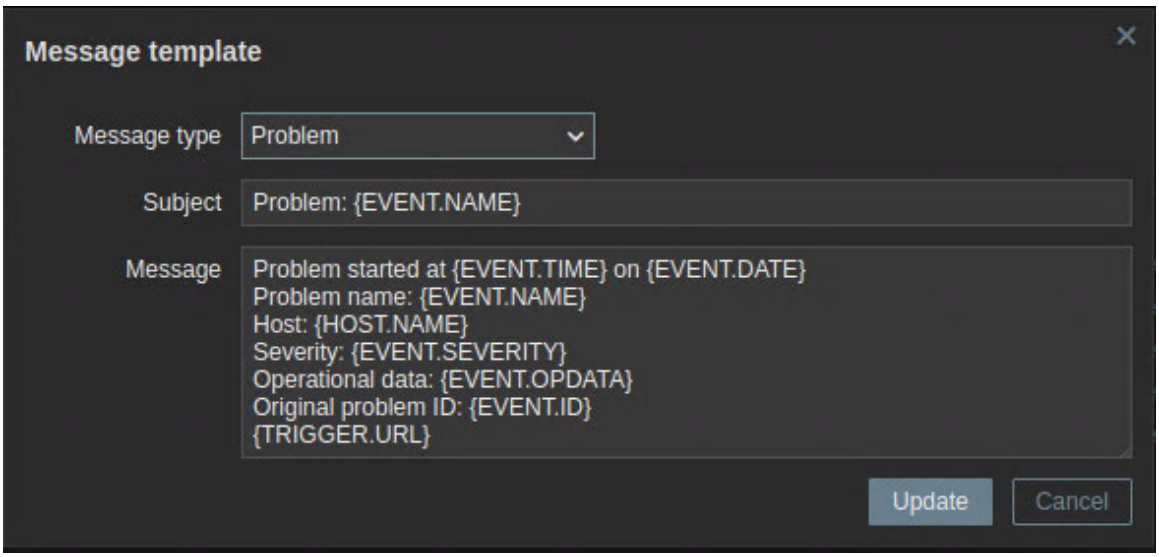
Figura 155. Configuración del medio de notificación por correo electrónico en Zabbix.



Nota. Pantalla de configuración del medio de notificación por correo electrónico. Elaboración propia.

En la pestaña *Message Templates* se definen las plantillas de mensaje que se utilizan para las notificaciones, en este caso, Zabbix ya tiene plantillas predefinidas para enviar los problemas que se hayan detectado, que contiene información como el *host* afectado, la gravedad del problema, la hora en que sucedió, entre otros.

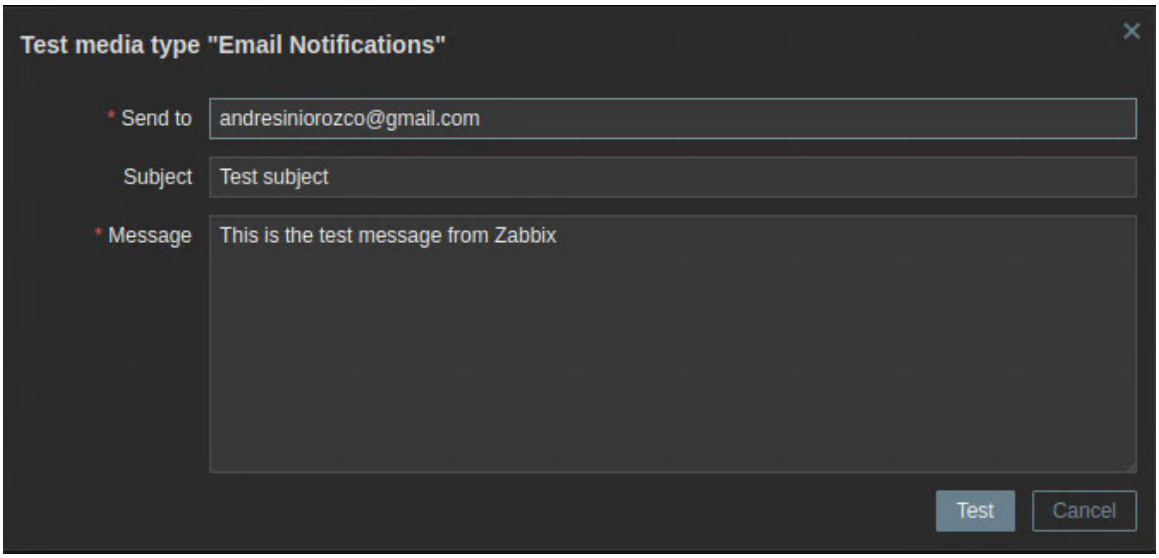
Figura 156. Configuración de las plantillas de mensaje en Zabbix.



Nota. Pantalla de configuración de las plantillas de mensaje en Zabbix. Elaboración propia.

Una vez configurado el medio de notificación por correo electrónico, se puede realizar una prueba para verificar que las notificaciones se envían correctamente. Para ello, se puede utilizar la opción *Test* en la configuración del medio de notificación, donde se debe proporcionar una dirección de correo electrónico de destino para recibir la notificación de prueba.

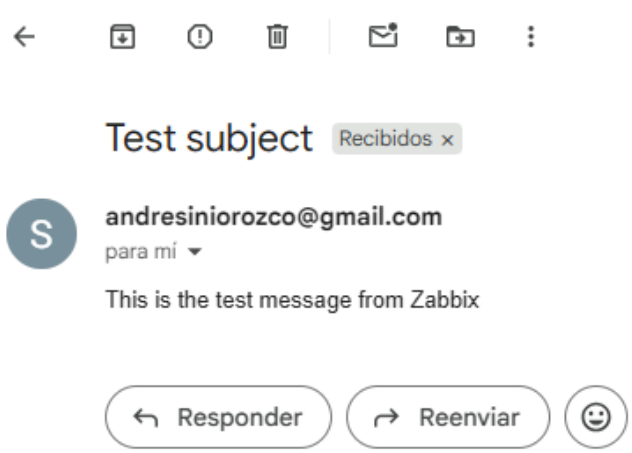
Figura 157. Prueba de envío de notificación en Zabbix.



Nota. Pantalla de prueba de envío de notificación en Zabbix. Elaboración propia.

Si todo está configurado correctamente, se debería recibir un correo electrónico de prueba en la dirección proporcionada.

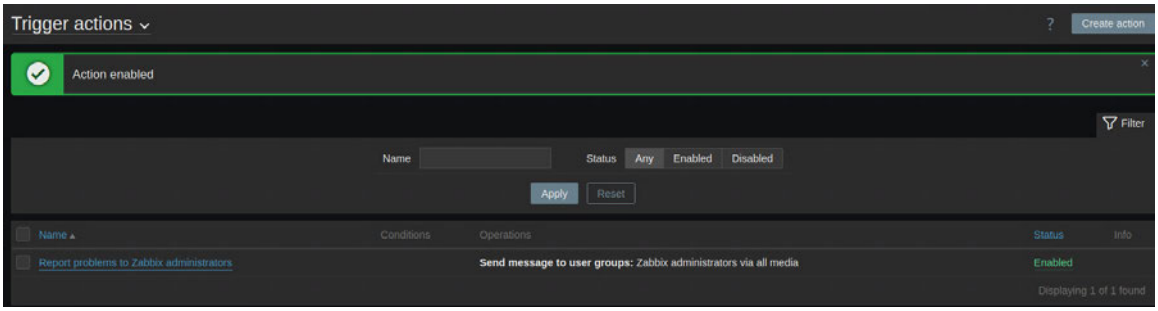
Figura 158. Notificación de prueba recibida en Gmail.



Nota. Ejemplo de notificación de prueba recibida en la bandeja de entrada de Gmail. Elaboración propia.

Luego se tiene que configurar las notificaciones para que se envíen cuando se detecten problemas. Para ello, se debe ir a la sección *Alerts* en el menú principal y seleccionar la opción *Actions*, y en ella *trigger actions*. En esta sección se pueden ver que hay una única opción *Report Problems to Zabbix administrators*, la cual se debe habilitar.

Figura 159. Configuración de las acciones de alerta en Zabbix.

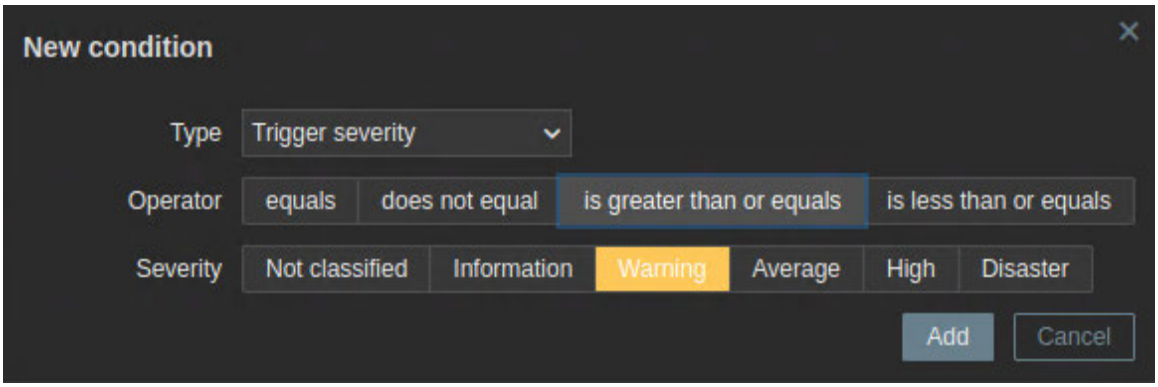


Nota. Pantalla de configuración de las acciones de alerta en Zabbix. Elaboración propia.

Con la opción habilitada, se debe configurar la acción para que envíe notificaciones. Al hacer clic se abre una ventana donde lo primero es seleccionar el nombre de la acción. Lo siguiente son las condiciones que tienen que ocurrir para determinar cuándo se debe enviar la notificación. Se hace clic en *Add* en la sección *Conditions* y se selecciona la opción *Trigger severity* y se selecciona la opción *is greater than or equal to* y se selecciona la opción *Warning*.

Esto significa que se envía una notificación cuando se detecte un problema con una gravedad mayor o igual a *Warning*.

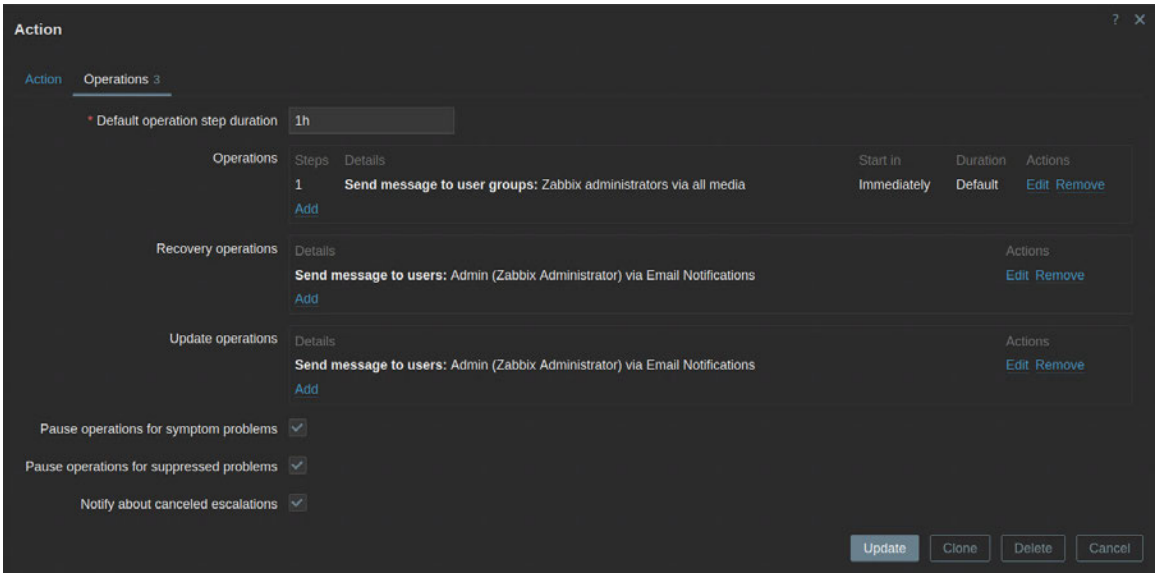
Figura 160. Configuración de las condiciones de alerta en Zabbix.



Nota. Pantalla de configuración de las condiciones de alerta en Zabbix. Elaboración propia.

Luego en la pestaña *Operations* se deja la opción por defecto, que lo que hace es enviar la notificación a todos los administradores de Zabbix, por todos los medios disponibles. En este caso, como solo se tiene un medio de notificación, que es el correo electrónico, se envía la notificación por correo electrónico a todos los administradores de Zabbix. En la sección *Recovery operations* se edita la acción para que envíe un mensaje al administrador por correo electrónico cuando se recupere el problema. Del mismo modo en la sección *Update operations* se edita la acción para que envíe un mensaje al administrador por correo electrónico cuando se actualice el problema.

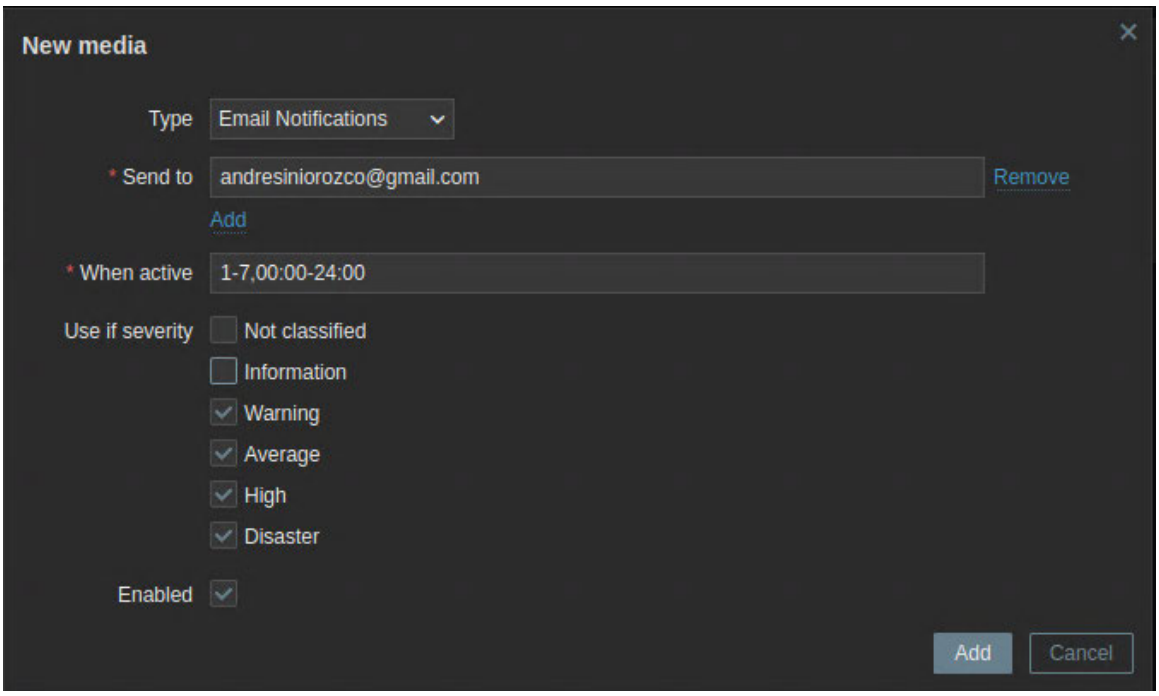
Figura 161. Configuración de las operaciones de alerta en Zabbix.



Nota. Pantalla de configuración de las operaciones de alerta en Zabbix. Elaboración propia.

Por último, se configura los usuarios en Zabbix que reciben las notificaciones. Para ello, se va a la sección *Users* en el menú principal y se selecciona al usuario Admin. En la sección *Media* se agrega el medio de notificación por correo electrónico que se creó anteriormente. Se hace clic en el botón de *Add* y se selecciona el medio de notificación por correo electrónico. Luego, se proporciona la dirección de correo electrónico del usuario que recibe las notificaciones. Además, se puede configurar el horario en el que se desean recibir las notificaciones. En este caso, la opción por defecto que es *24/7*.

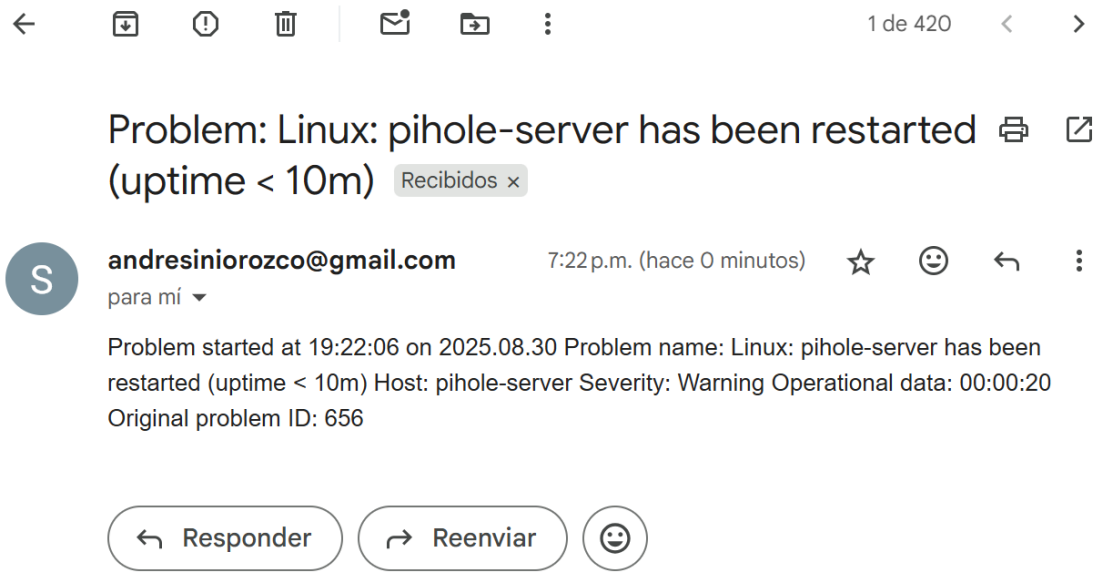
Figura 162. Configuración de los usuarios en Zabbix.



Nota. Pantalla de configuración de los usuarios en Zabbix. Elaboración propia.

Para confirmar la configuración de los usuarios y los medios de notificación, se puede realizar una prueba apagando alguno de los dispositivos monitoreados y verificando si se reciben las notificaciones por correo electrónico. En este caso, se apaga el servidor Pi-hole y se verifica si se recibe la notificación por correo electrónico. Al reiniciar el servidor Pi-hole, se debería recibir una notificación por correo electrónico indicando que el servidor se reinició.

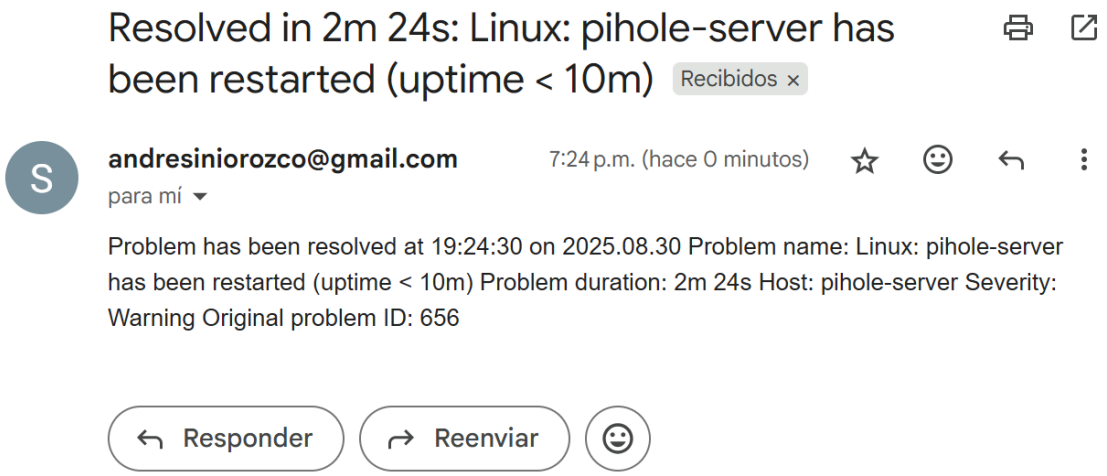
Figura 163. Notificación de problema detectado en Gmail.



Nota. Ejemplo de notificación de problema detectado recibida en la bandeja de entrada de Gmail. Elaboración propia.

Al reconocer el problema en Zabbix, y cerrar el problema también se debe verificar que se haya recibido la notificación correspondiente.

Figura 164. Notificación de problema resuelto en Gmail.



Nota. Ejemplo de notificación de problema resuelto recibida en la bandeja de entrada de Gmail. Elaboración propia.

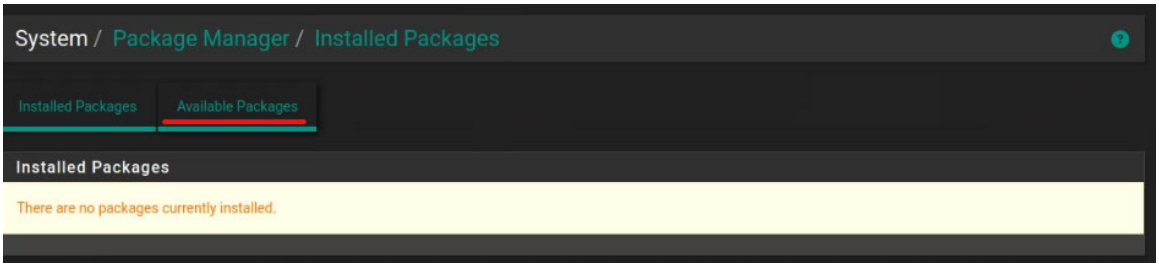
Con esto realizado se tendrían las alertas configuradas en Zabbix para que se envíen notificaciones por correo electrónico cuando se detecten problemas en los dispositivos monitoreados.

7.7. Instalación y configuración de Snort para la detección y prevención de intrusos

Snort es un sistema de detección de intrusos (IDS) y prevención de intrusos (IPS) de código abierto que se utiliza para monitorear el tráfico de red en tiempo real y detectar actividades sospechosas o maliciosas. En este caso, se instala Snort como paquete adicional en pfSense para monitorear el tráfico de red y detectar posibles amenazas.

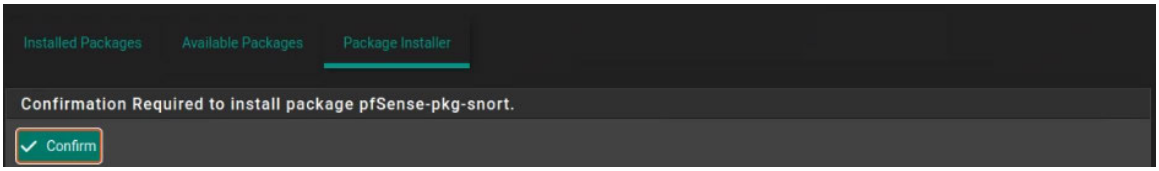
Para instalar Snort en pfSense, se va a la sección *System* en el menú principal y se selecciona la opción *Package Manager*. En esta sección se pueden ver los paquetes instalados y disponibles para instalar. Para instalar Snort, se selecciona la pestaña *Available Packages* y se busca el paquete Snort. Una vez encontrado, se selecciona el paquete y se hace clic en el botón de *Install*. Esto inicia el proceso de instalación del paquete Snort en pfSense.

Figura 165. Instalación del paquete Snort en la interfaz web de pfSense.



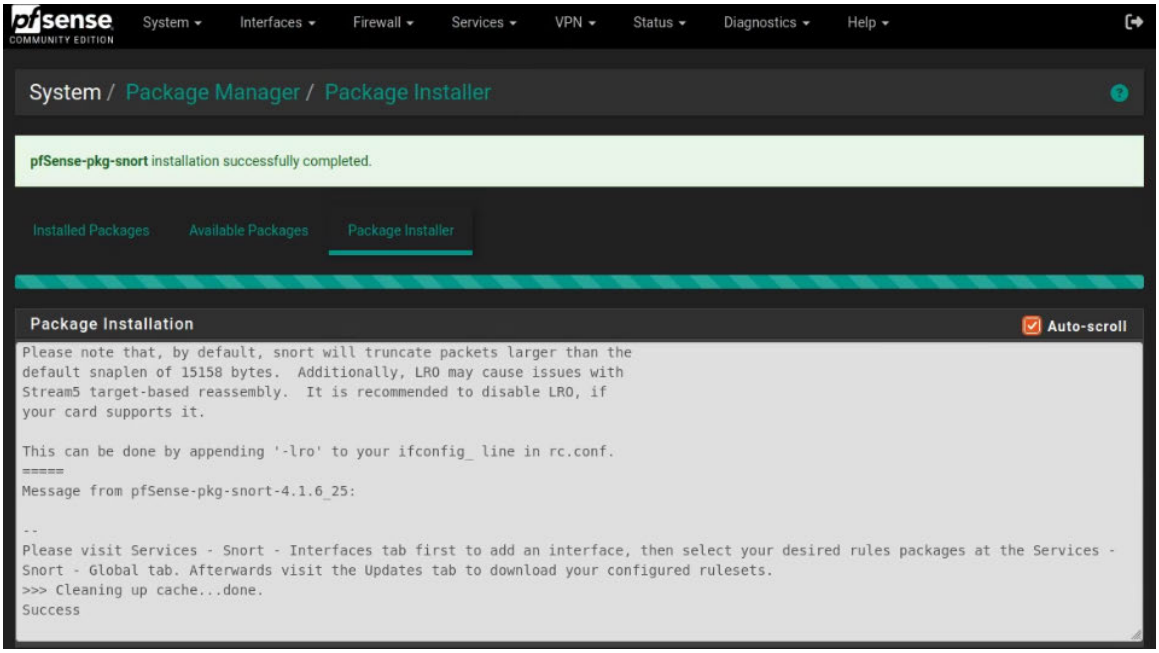
Nota. Selección del paquete Snort desde el gestor de paquetes para habilitar capacidades IDS/IPS. Elaboración propia.

Figura 166. Pantalla inicial de configuración de Snort en la interfaz web de pfSense.



Nota. Pantalla inicial de opciones del paquete previo a habilitar fuentes de reglas y detectores. Elaboración propia.

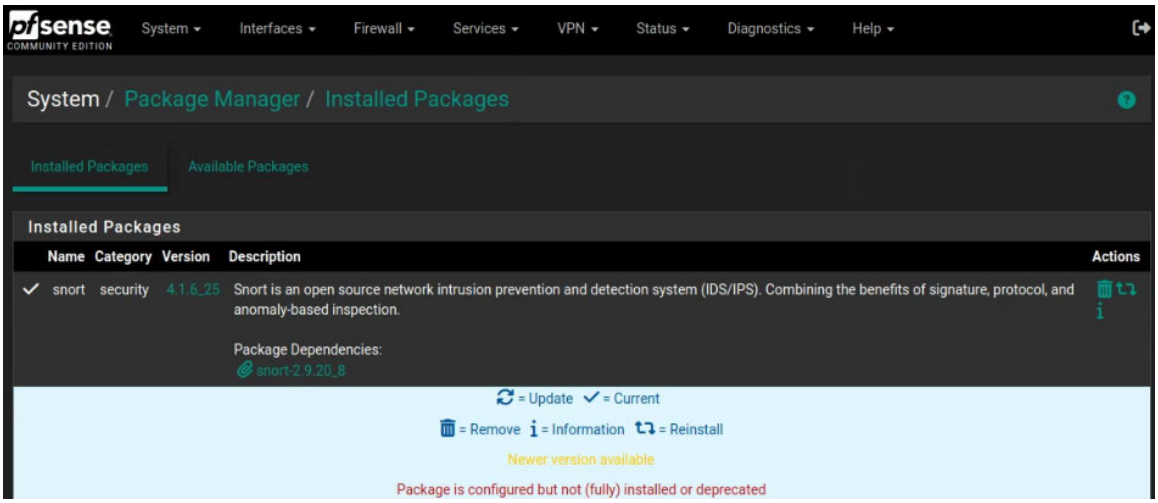
Figura 167. Ajustes adicionales del asistente de configuración de Snort en pfSense.



Nota. Ajustes adicionales del asistente de configuración inicial de Snort. Elaboración propia.

Ahora el paquete de Snort debería aparecer en la lista de paquetes instalados.

Figura 168. Paquete Snort instalado en la interfaz web de pfSense.



Nota. Confirmación de instalación del paquete listo para configuración global. Elaboración propia.

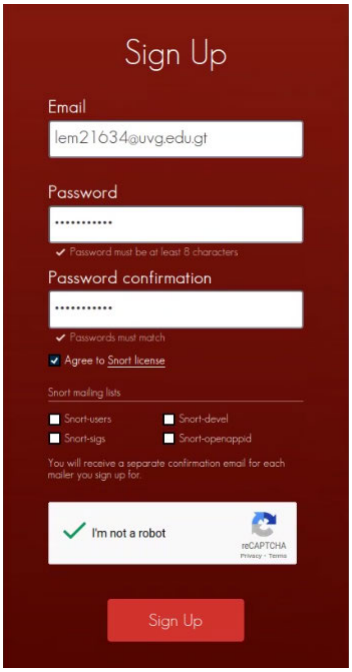
7.7.1. Configuración de Snort para la obtención de reglas oficiales y comunitarias

Una vez instalado el paquete, se puede configurar Snort para obtener varios tipos de reglas que la comunidad de Snort proporciona. Para el alcance de este proyecto se crean reglas personalizadas, pero de igual forma se muestra el proceso de obtención de dichas reglas ya que pueden ser útiles en otros escenarios.

Para ello, se va a la sección *Services* en el menú principal y se selecciona la opción Snort. En esta sección se pueden ver las diferentes pestañas para configurar Snort. En primer lugar se selecciona la pestaña *Global Settings*. Lo primero que se hace es habilitar la opción *Enable Snort VRT* que lo que hace es habilitar las reglas de Snort que se obtienen de tener una cuenta gratiuta en *Snort.org*.

Para ello se crea una cuenta gratuita en el sitio web de *Snort.org* y mediante este link https://www.snort.org/users/sign_up se crea una cuenta y así se obtiene acceso a las reglas de Snort.

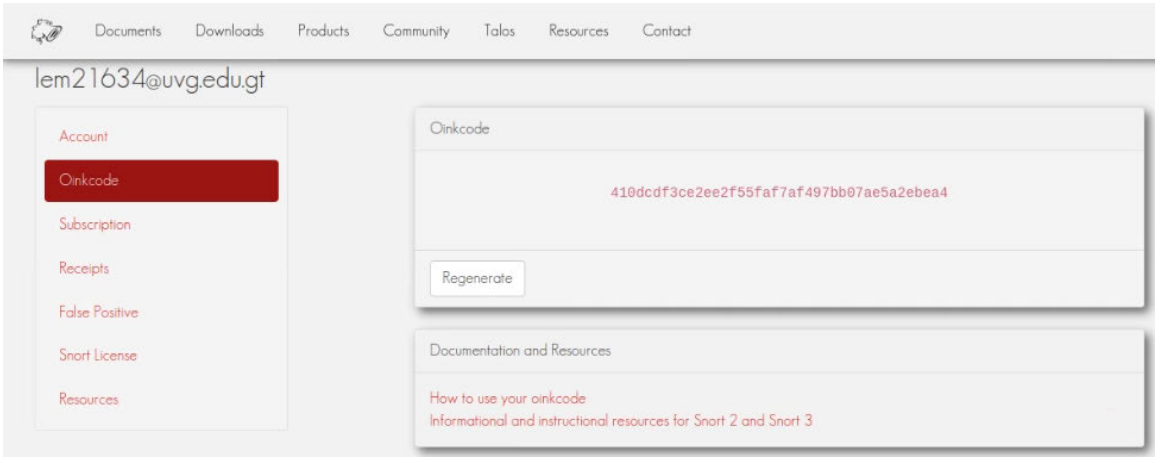
Figura 169. Habilitación de fuentes de reglas VRT en pfSense.



Nota. Activación de fuentes de reglas y parámetros base del motor IDS. Elaboración propia.

Con la cuenta creada, se va a la sección *My Account* y en el panel en la derecha se selecciona la opción *Oinkcode*. El *Oinkcode* es un código que se utiliza para autenticar la cuenta y obtener las reglas de Snort. Se copia el *Oinkcode* y se pega en el campo correspondiente en la sección *Global Settings* en la interfaz web de pfSense.

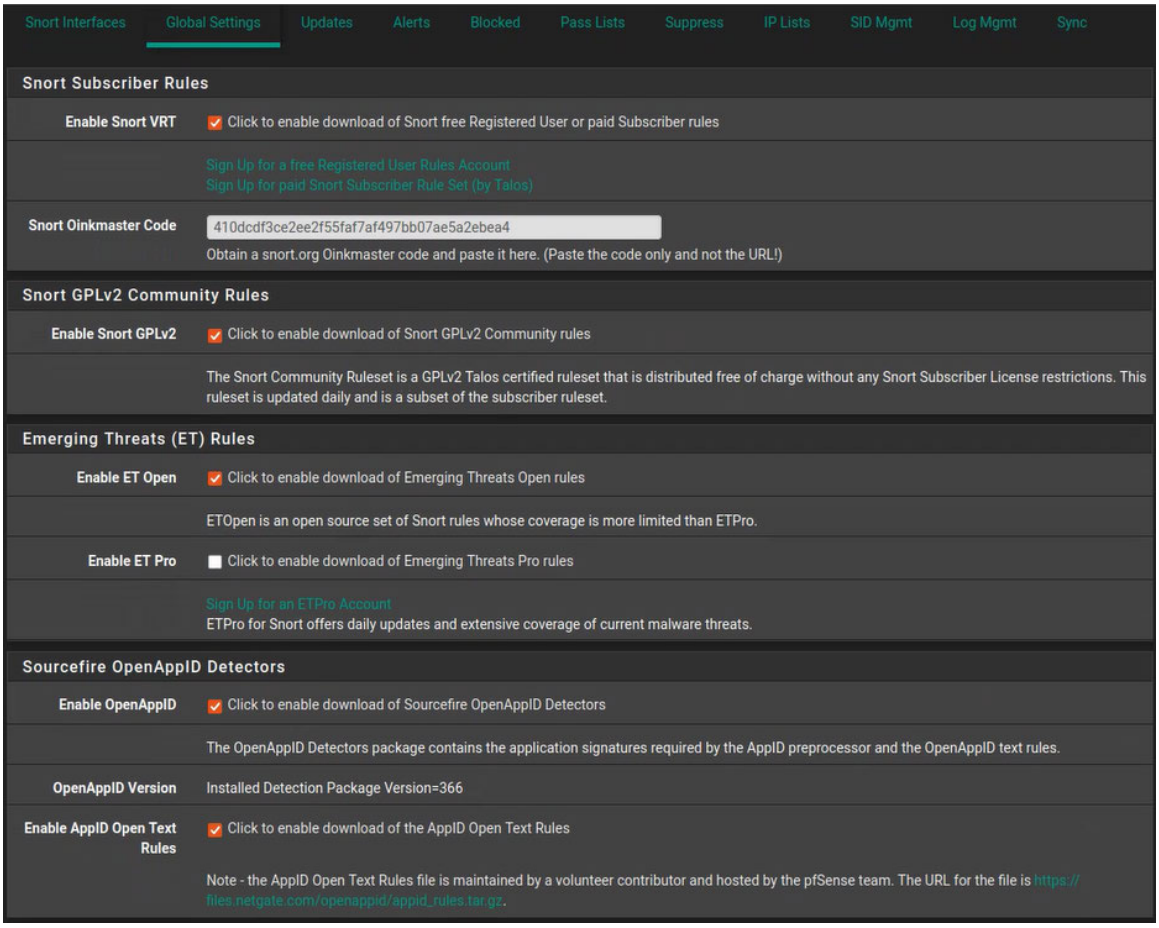
Figura 170. Obtención del Oinkcode en la cuenta de Snort.



Nota. Recuperación del Oinkcode necesario para descargar reglas VRT oficiales. Elaboración propia.

Ya con el *Oinkcode* copiado, se pega en el campo correspondiente en la sección *Global Settings* en la interfaz web de pfSense. En la sección *Snort GPLv2 community rules* se habilita la opción *Enable Snort GPLv2* que son las reglas de la comunidad de Snort. En la sección *Emerging threats rules* se habilita la opción *Enable ET Open* que es una colección gratuita y de libre acceso de reglas que proporciona una amplia gama de reglas que detectan diversas amenazas, incluyendo actividades de escaneo, ataques a diferentes protocolos y listas negras (*blacklists*). Por último, en la sección *Sourcefire OpenAppID Detectors* se habilita la opción *Enable OpenAppID* que es una colección de detectores de aplicaciones que permite identificar aplicaciones específicas en el tráfico de red.

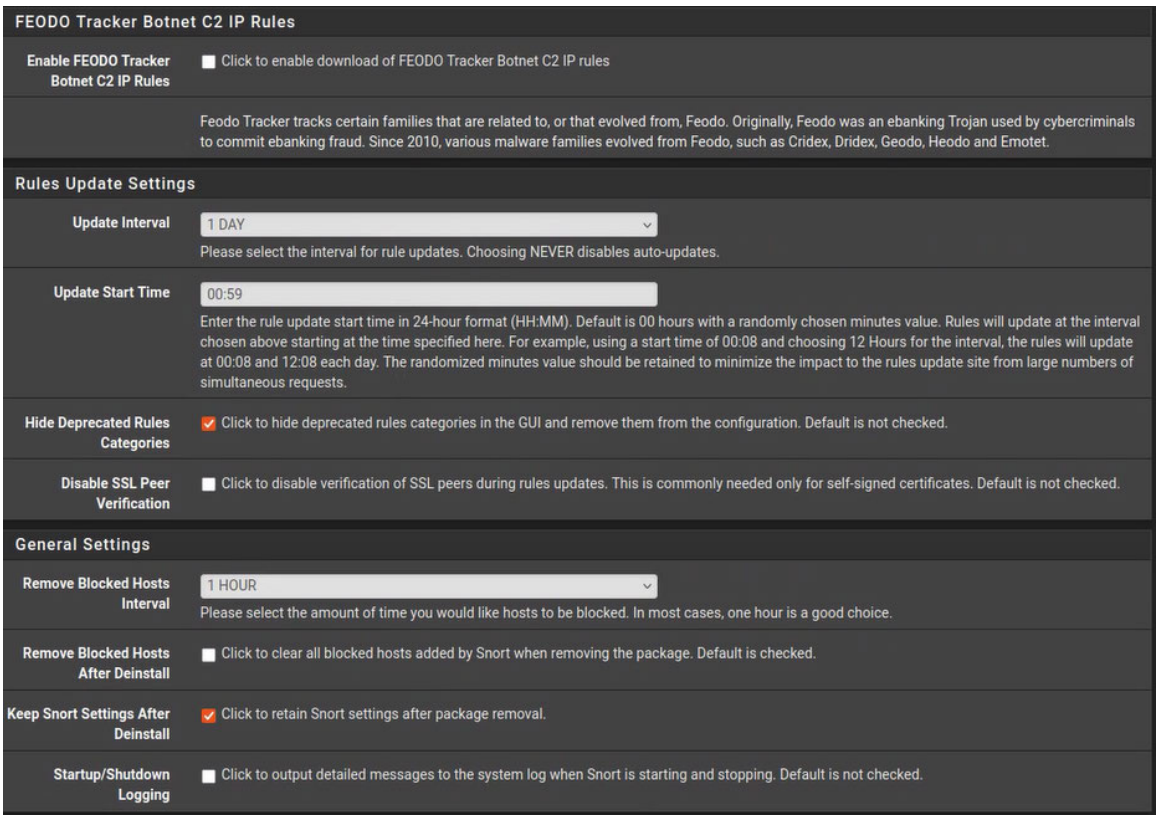
Figura 171. Configuración del Oinkcode y fuentes de reglas additionales en pfSense.



Nota. Entrada del Oinkcode y habilitación de fuentes ET Open y OpenAppID para ampliar cobertura. Elaboración propia.

Luego, en la sección *Rules Update Settings* en *Update Interval* se recomienda cambiar la opción *Never* a algún intervalo de tiempo, para que las reglas se actualicen automáticamente. En este caso, se selecciona la opción *1 Day*. En esa sección también se puede seleccionar la opción *Hide Deprecated Rules Categories* para ocultar las categorías de reglas obsoletas y eliminarlas de la configuración.

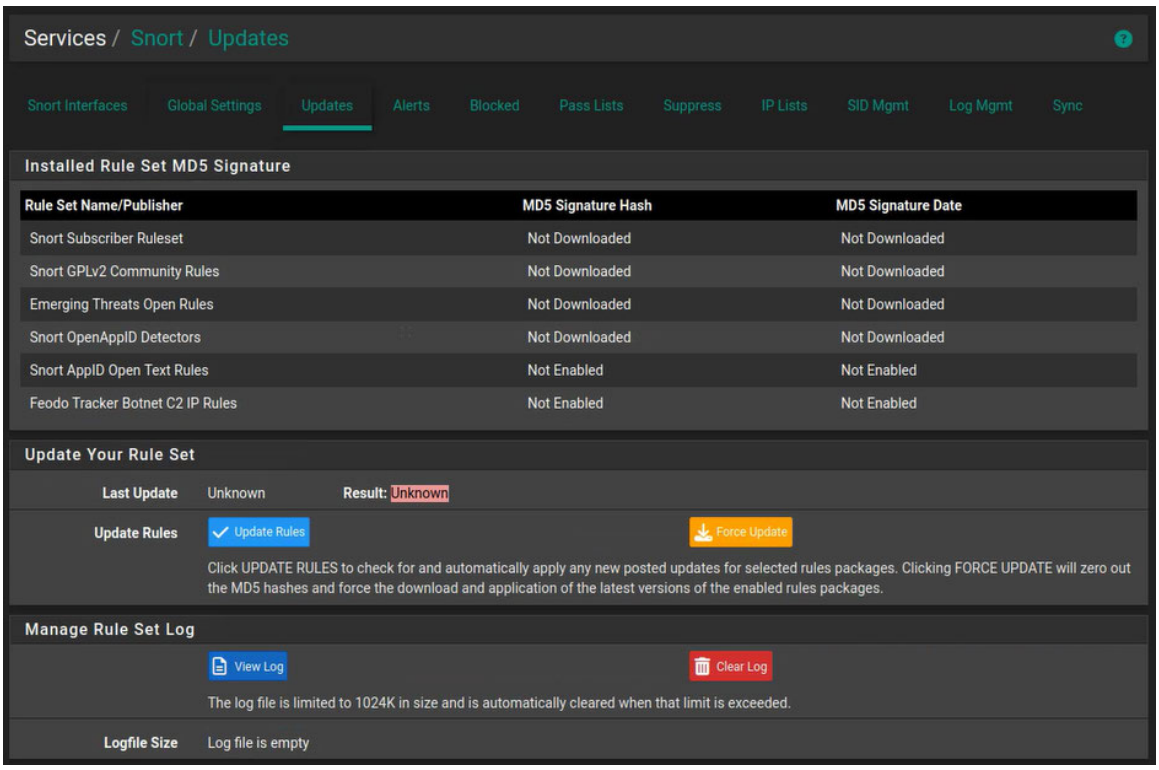
Figura 172. Configuración de intervalos de actualización de reglas en Snort.



Nota. Ajuste de intervalos de actualización y políticas de limpieza de bloqueos. Elaboración propia.

Se guardan los cambios y se accede ahora a la pestaña *Updates* donde se pueden observar los diferentes *Rule Sets* instalados. En la sección *Update your Rule Set* se selecciona la opción *Update Rules* para que se descarguen las reglas de Snort y se actualicen las reglas instaladas. Esto puede tardar unos minutos dependiendo de la velocidad de la conexión a Internet.

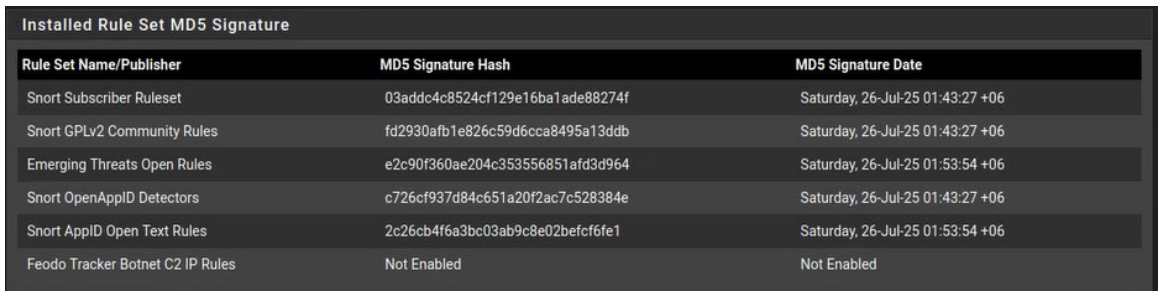
Figura 173. Actualización de las reglas de Snort en la interfaz web de pfSense.



Nota. Ejecución manual de descarga para sincronizar conjuntos de reglas. Elaboración propia.

Una vez que se han actualizado las reglas, se puede observar que se tienen el *MD5 Signature Hash* que es un identificador único de las reglas instaladas y su respectiva fecha de actualización. Esto es útil para verificar que las reglas se han actualizado correctamente y que se están utilizando las últimas versiones de las reglas.

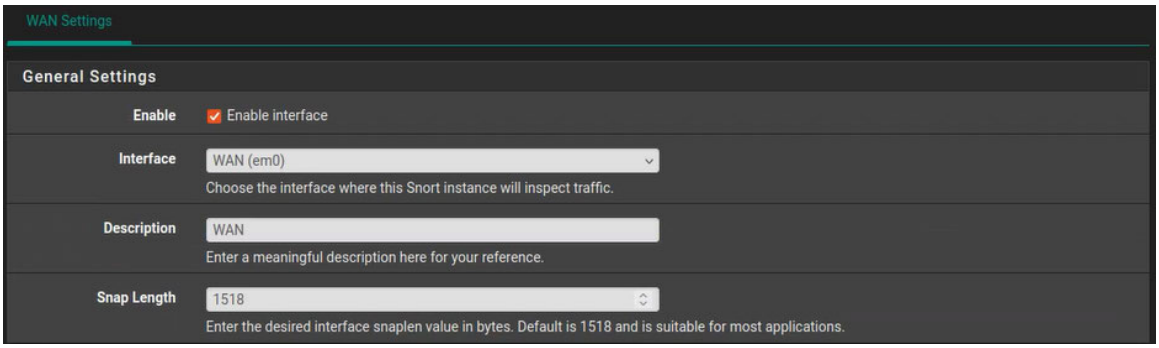
Figura 174. Reglas de Snort actualizadas en la interfaz web de pfSense.



Nota. Confirmación de integridad (hash MD5) tras actualización de firmas IDS. Elaboración propia.

Estas son reglas que la comunidad de Snort ha creado y que se pueden utilizar para detectar diferentes tipos de amenazas. Sin embargo, para entender mejor cómo funcionan estas reglas, se demuestra cómo se pueden crear reglas personalizadas para detectar amenazas específicas. Para ello, se va a la sección *Snort Interfaces* en el menú principal y se selecciona la opción *Add*. En esta sección se pueden ver las diferentes pestañas para configurar Snort en una interfaz específica. En primer lugar, se selecciona la interfaz de red que se desea monitorear, en este caso, la interfaz de red WAN, ya que solo es con el fin de mostrar el funcionamiento básico. Se dejan las opciones por defecto, solo asegurando de habilitar la interfaz y seleccionar la interfaz de red WAN.

Figura 175. Configuración de la interfaz de Snort en la interfaz web de pfSense.



Nota. Asignación de la interfaz WAN y habilitación del sensor para captura de tráfico. Elaboración propia.

Se guarda la configuración y luego que se ha guardado la configuración, nuevas pestañas aparecen en la parte superior de la página. En la pestaña *WAN Rules* se pueden ver las reglas que se han configurado para la interfaz de red WAN. En esta sección se pueden agregar nuevas reglas personalizadas para detectar amenazas específicas. Para ello en la parte *Category Selection* se selecciona la opción *custom.rules* y se pueden agregar las reglas personalizadas en el cuadro de texto correspondiente.

La sintaxis básica de las reglas de Snort es la siguiente:
action protocol source_ip source_port -> destination_ip destination_port.

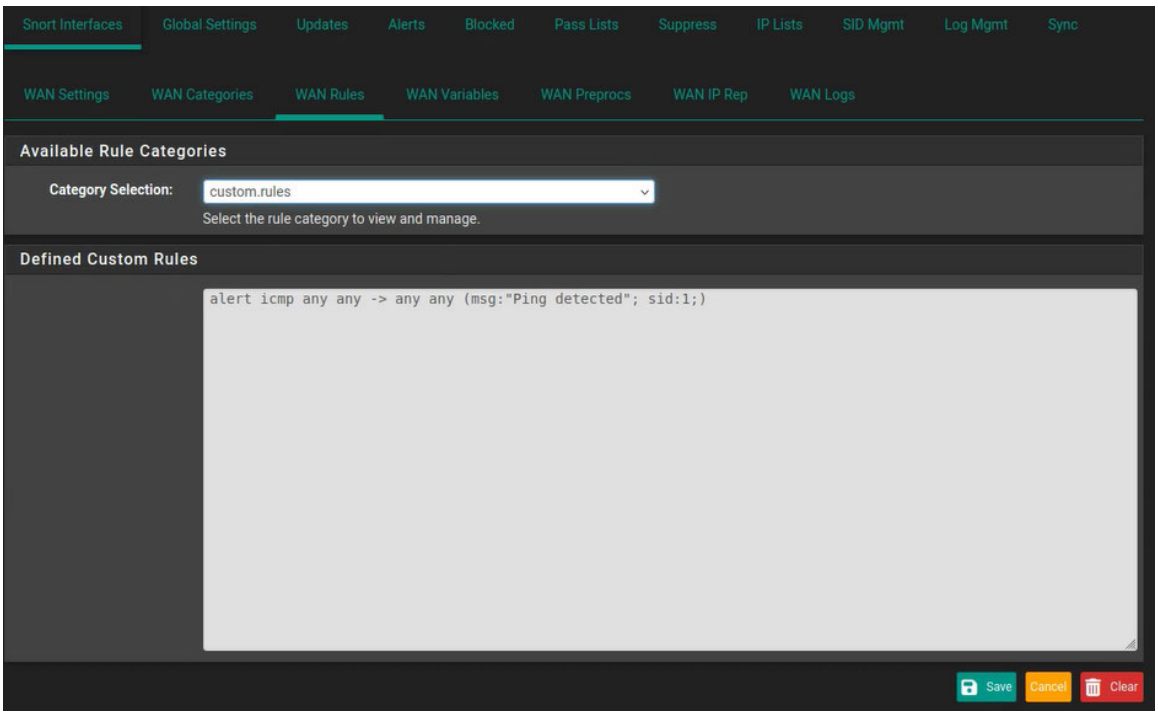
Así que para probar el funcionamiento de las reglas de Snort, se puede crear una regla personalizada que detecte un *ping* a la dirección IP de la red WAN, ya que es la interfaz que se habilitó. Para ello, se puede agregar la siguiente regla personalizada en el cuadro de texto correspondiente:

Cuadro 56. Regla personalizada de Snort de prueba.

```
1 alert icmp any any -> any any (msg:"Ping detected"; sid:1;)
```

Nota. Regla personalizada para detectar todo tráfico ICMP generando alerta indicando *ping* detectado. Elaboración propia.

Figura 176. Regla personalizada de Snort en la interfaz web de pfSense.



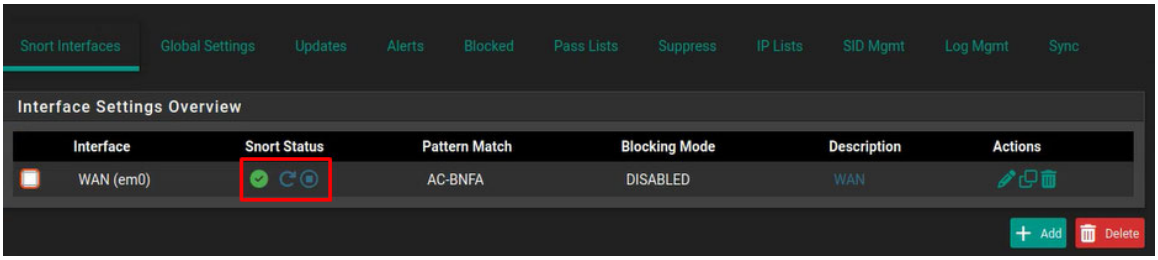
Nota. Edición de *custom.rules* agregando alerta ICMP para validar pipeline de detección. Elaboración propia.

Esta regla lo que hace es detectar cualquier paquete ICMP que se envíe desde cualquier dirección IP y puerto hacia cualquier dirección IP y puerto.

El protocolo ICMP es el protocolo que se utiliza para enviar mensajes de control de red, como los mensajes de *ping*. La regla también incluye un mensaje que aparece cuando se detecta un paquete ICMP, en este caso, *"Ping detected"*. El *sid* es un identificador único de la regla, en este caso, el número 1.

Una vez que se ha agregado la regla personalizada, se debe guardar la configuración y luego activar la interfaz de Snort para que la regla se aplique. Para ello, se debe ir a la pestaña *Snort Interfaces* y en la sección *Snort Status* se debe seleccionar la opción *Start Snort on this interface* para activar la interfaz de Snort.

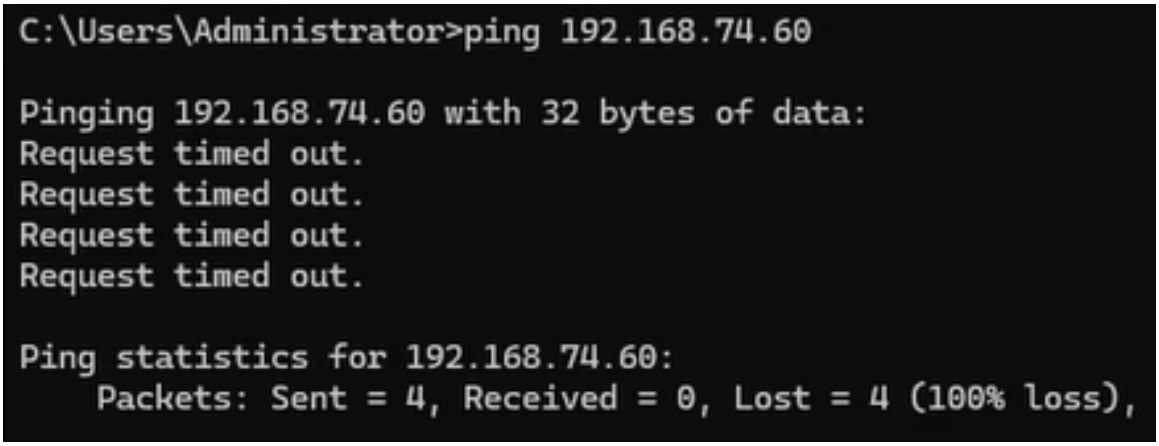
Figura 177. Activación de la interfaz de Snort en la interfaz web de pfSense.



Nota. Inicio del proceso Snort sobre la interfaz seleccionada para aplicar reglas cargadas. Elaboración propia.

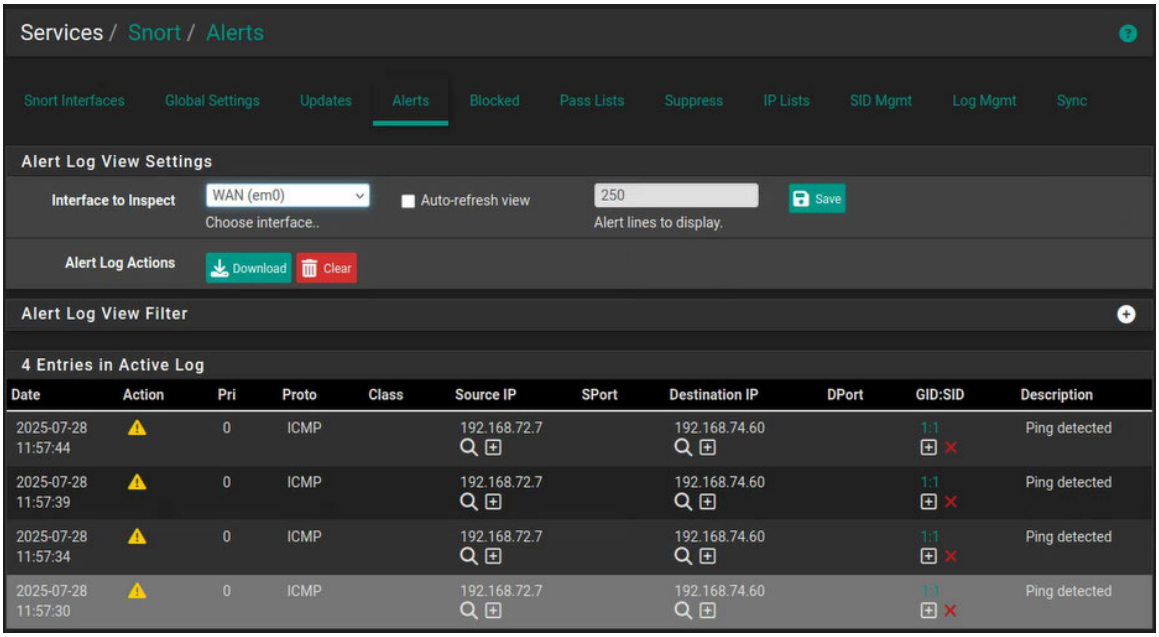
Para comprobar que la regla se ha aplicado correctamente, se puede intentar hacer un *ping* a la dirección IP de la red WAN desde una computadora en la misma red que la interfaz WAN del *firewall*. Si se recibe una alerta en la interfaz de Snort, significa que la regla se ha aplicado correctamente y se ha detectado el *ping*.

Figura 178. Detección de ICMP por Snort (vista principal).



Nota. Evento generado por regla personalizada confirmando detección de tráfico ICMP. Elaboración propia.

Figura 179. Confirmación de detección ICMP en Snort (vista complementaria).



Nota. Visualización complementaria del mismo incidente para verificar consistencia de alertas. Elaboración propia.

En la figura 178 se puede observar el ping que se ha enviado desde la computadora hacia la dirección IP de la red WAN, que aunque no hay respuesta (se debe a que en la interfaz WAN no hay una regla que permita respuesta a los pings), Snort ha detectado el intento de ping y ha generado una alerta.

7.8. Instalación y configuración de OpenVAS para el escaneo de vulnerabilidades

OpenVAS se debe instalar en la máquina virtual de Kali Linux en la red del SOC & MOC que se creó anteriormente. Para ello, se deben seguir los siguientes pasos.

Primero, se actualiza el sistema y los repositorios de Kali Linux ejecutando los siguientes comandos:

Cuadro 57. Actualización de paquetes en Kali Linux.

```
sudo apt update && sudo apt upgrade
```

Nota. Actualización de índices de paquetes y upgrade de paquetes instalados en Kali Linux. Elaboración propia.

Segundo, se instala *Greenbone Community Edition* (GCE), que es la versión gratuita de OpenVAS. Para ello, se ejecuta el siguiente comando:

Cuadro 58. Comando para instalar *Greenbone Community Edition* en Kali Linux.

```
sudo apt install gvm -y
```

Nota. Instalación del paquete GVM que contendrá *Greenbone Community Edition* para escaneo de vulnerabilidades. Elaboración propia.

Después de instalar los paquetes requeridos se corre el *script* para completar la instalación. Para ello, se ejecuta el siguiente comando:

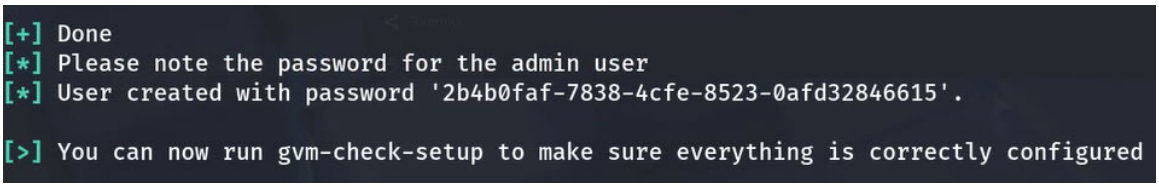
Cuadro 59. Ejecución del *script* de configuración de GVM en Kali Linux.

```
sudo gvm-setup
```

Nota. Ejecución del *script* de configuración que finaliza configuración y genera credenciales administrativas. Elaboración propia.

Al finalizar habría generado una contraseña para el usuario *admin*. Esta contraseña se guarda en un lugar seguro, ya que se necesita para acceder a la interfaz web de OpenVAS.

Figura 180. Resultado de la ejecución del *script* de configuración de GVM en Kali Linux.



Nota. Finalización del setup mostrando credenciales de administrador generadas automáticamente para acceso web. Elaboración propia.

Se verifica que la instalación se haya realizado correctamente ejecutando el siguiente comando:

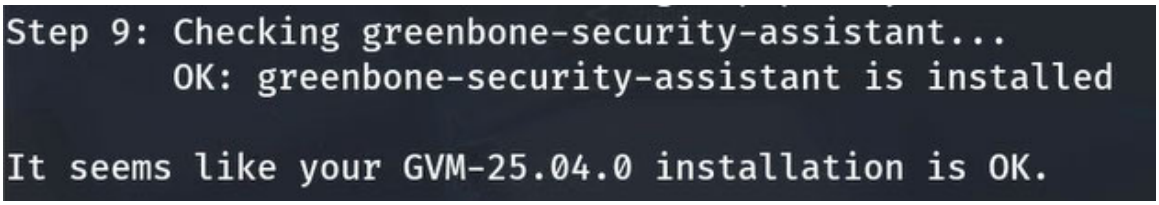
Cuadro 60. Comando para verificar la instalación de GVM en Kali Linux.

```
sudo gvm-check-setup
```

Nota. Verificación de integridad y completitud de la instalación de GVM indicando disponibilidad de servicios. Elaboración propia.

Debería finalizar con un mensaje al final indicando que la instalación se ha realizado correctamente y que no hay problemas de configuración.

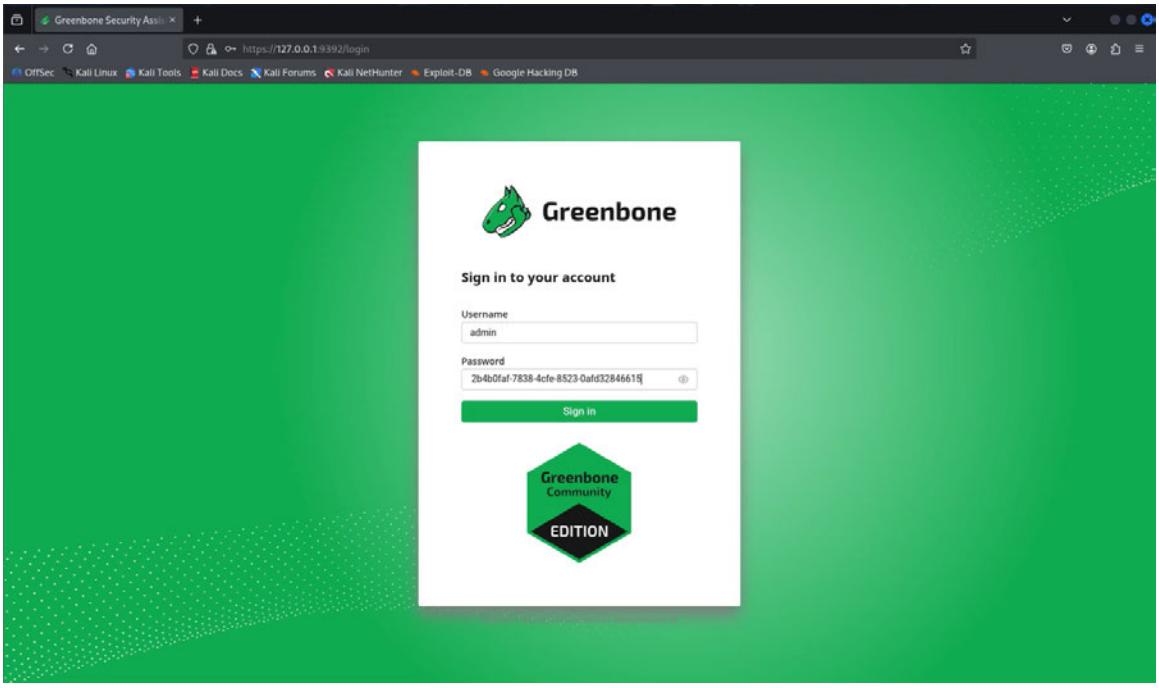
Figura 181. Resultado de la verificación de instalación de GVM en Kali Linux.



Nota. Resultado del check de configuración confirmando que todos los componentes GVM están operativos. Elaboración propia.

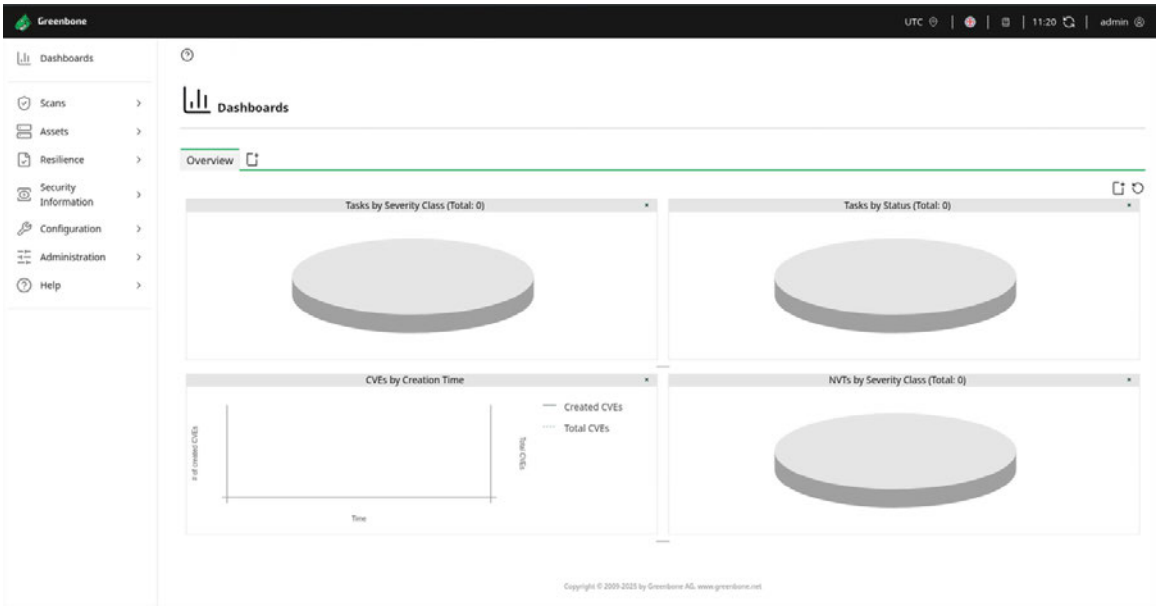
El siguiente paso es ir a la interfaz web de OpenVAS para comenzar a utilizarlo. Para ello, se abre un navegador web y se accede a la dirección `https://127.0.0.1:9392`. Se ingresa el nombre de usuario *admin* y la contraseña generada durante la instalación.

Figura 182. Interfaz web de OpenVAS.



Nota. Pantalla de autenticación de OpenVAS/GVM accesible vía navegador para gestión de escaneos. Elaboración propia.

Figura 183. *Dashboard* principal de OpenVAS.



Nota. Panel de control principal mostrando resumen de actividades y acceso a funciones de escaneo de vulnerabilidades. Elaboración propia.

CAPÍTULO 8

Simulación y mitigación de ataques a la red para evaluar la efectividad de las herramientas implementadas

En este capítulo se describen los diferentes ataques que se simulan en la red para así implementar Snort y observar su efectividad en la detección y prevención de ataques. Primeramente, se describe cómo se simulan los ataques externos e internos a la red, luego se procede a realizar la simulación de dichos ataques antes de configurar Snort para así ver el impacto que estos ataques tienen en la red. Seguido de esto, se procede a configurar Snort para mitigar los ataques y se observa la efectividad de Snort en la detección de ataques. Posteriormente, se implementan dos *scripts* en Python para automatizar el bloqueo de direcciones IP basado en las alertas generadas por Snort con algunos parámetros personalizados, y también para automatizar el desbloqueo de estas mismas. Se describe cómo se utiliza OpenVAS para escanear la red en busca de vulnerabilidades, la generación de reportes y la programación de escaneos automáticos. Finalmente, se muestra cómo Pi-hole hace más segura la navegación por Internet.

8.1. Simulación de ataques en la red

Este tipo de ataques se realizan desde la máquina virtual de Kali Linux que se encuentra fuera de la red del *firewall*, es decir que se encuentra en la misma red que la interfaz WAN, la PC5 en el esquema de la figura 2. A continuación, se describen los diferentes ataques que se simulan desde Kali Linux hacia la red interna.

8.1.1. Lista de ataques a simular

Cuadro 61. Ataques de reconocimiento a simular.

Nombre del ataque	**Descripción**	**Dirección**	**Puerto**
Barrido ICMP (*Ping Sweep*)	Descubrimiento de *hosts* mediante eco ICMP.	Kali Linux (LAN) → Red DMZ o SOC & MOC	N/A
Escaneo TCP SYN	Detección de puertos abiertos enviando paquetes SYN.	Kali Linux (LAN) → Red LAN o DMZ o SOC & MOC	1-65535
Escaneo UDP	Enumeración de servicios sin conexión orientada a UDP.	Kali Linux (LAN) → Red LAN o DMZ o SOC & MOC	1-65535
Detección de versiones/NSE	*Fingerprinting* de servicios y pruebas con *scripts*.	Kali Linux (LAN) → Red DMZ o SOC & MOC	Varios
Banner grabbing (HTTP/FTP)	Obtención de *banners* para identificar software y versión.	Kali Linux (LAN) → Red DMZ o SOC & MOC	80/21

Nota. Reconocimiento activo para perfilar objetivos y servicios expuestos. Elaboración propia.

Cuadro 62. Ataques de fuerza bruta y abuso de autenticación.

Nombre del ataque	**Descripción**	**Dirección**	**Puerto**
SSH *Brute force*	Intentos masivos de contraseña/usuario contra el servicio SSH.	Kali Linux (LAN) → Metasploitable 3 (DMZ)	22 (SSH)
FTP *Brute force*	Ataques de contraseñas al servicio FTP (incluye pruebas de anónimo).	Kali Linux (LAN) → Metasploitable 3 (DMZ)	21 (FTP)
SNMP *community guessing*	Descubrimiento de comunidades SNMP por defecto o débiles.	Kali Linux (LAN) → Metasploitable 3 (DMZ)	161 (SNMP)

Nota. Ataques de credenciales para acceso no autorizado a servicios. Elaboración propia.

Cuadro 63. Envenenamiento y suplantación en red.

Nombre del ataque	Descripción	Dirección	Puerto
ARP *Spoofing*/MITM	Suplantación ARP para interceptar tráfico entre víctima y *gateway*.	Kali Linux (LAN) ↔ *Hosts* en LAN	N/A
Servidor DHCP *rogue*	Respuesta DHCP maliciosa para redirigir tráfico.	Kali Linux (LAN) → *Hosts* en LAN	67/68 (DHCP)

Nota. Ataques internos para desviar tráfico y capturar credenciales. Elaboración propia.

Cuadro 64. Ataques de denegación de servicio (DoS/DDoS).

Nombre del ataque	Descripción	Dirección	Puerto
TCP SYN *flood*	Inundación de solicitudes SYN para agotar la cola de conexiones.	Kali Linux (WAN/LAN) → Metasploitable 3 (DMZ)	80 (HTTP)
HTTP *GET flood*	Alto volumen de peticiones HTTP para saturar el servicio.	Kali Linux (WAN) → Metasploitable 3 (DMZ)	80 (HTTP)
UDP *flood*	Envío masivo de paquetes UDP a un puerto objetivo.	Kali Linux (WAN) → Metasploitable 3 (DMZ)	80 (HTTP)

Nota. Pruebas de disponibilidad para demostrar detección/bloqueo con Snort/IPS. Elaboración propia.

8.1.2. Barrido ICMP (*Ping Sweep*)

Para simular un ataque de barrido ICM, se utiliza la herramienta *fping* que viene preinstalada en Kali Linux. Esta herramienta permite enviar paquetes ICMP de eco a múltiples direcciones IP y es ideal para simular ataques de reconocimiento. El comando para ejecutar el ataque es el siguiente:

Cuadro 65. Comando para simular un ataque de barrido ICMP.

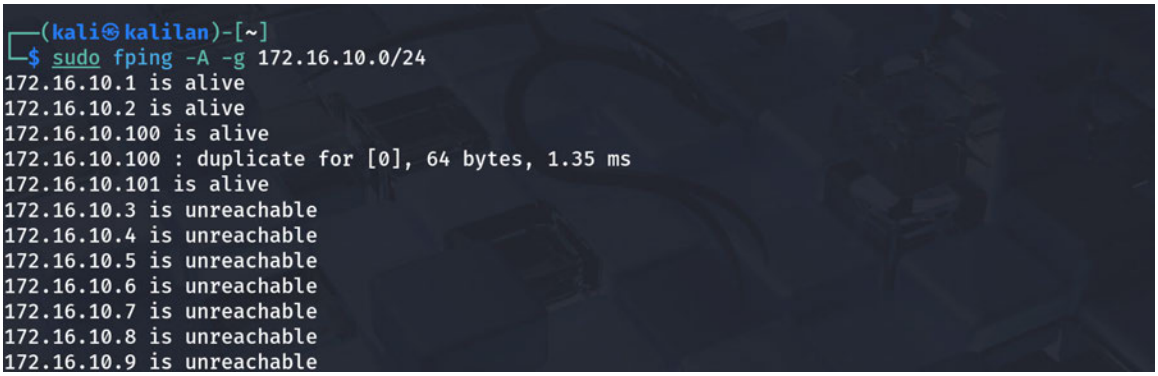
```
sudo fping -a -g <rango_de_ips>
```

Nota. Comando fping para descubrimiento de *hosts* activos mediante ICMP eco requerido en etapa de reconocimiento. Elaboración propia.

En este comando, se utiliza la opción `-a` para mostrar solo las direcciones IP activas y la opción `-g` para generar un rango de direcciones IP. Se reemplaza `<rango_de_ips>` por el rango de direcciones IP que se desea escanear, por ejemplo, `172.16.10.0/24`.

Al ejecutar este comando, se envían paquetes ICMP de eco a todas las direcciones IP en el rango especificado y aparecen las direcciones IP que responden, indicando que están activas en la red. Esto permite identificar los *hosts* en línea y disponibles para posibles ataques posteriores.

Figura 184. Resultado del barrido ICMP en Kali Linux.



Nota. Listado de *hosts* activos detectados mediante fping en el rango especificado. Elaboración propia.

8.1.3. Escaneo TCP SYN

Para simular un ataque de escaneo TCP SYN, se utiliza la herramienta *nmap*. Esta herramienta permite enviar paquetes TCP personalizados y es ideal para simular ataques de reconocimiento. El comando para ejecutar el ataque es el siguiente:

Cuadro 66. Comando para simular un ataque de escaneo TCP SYN.

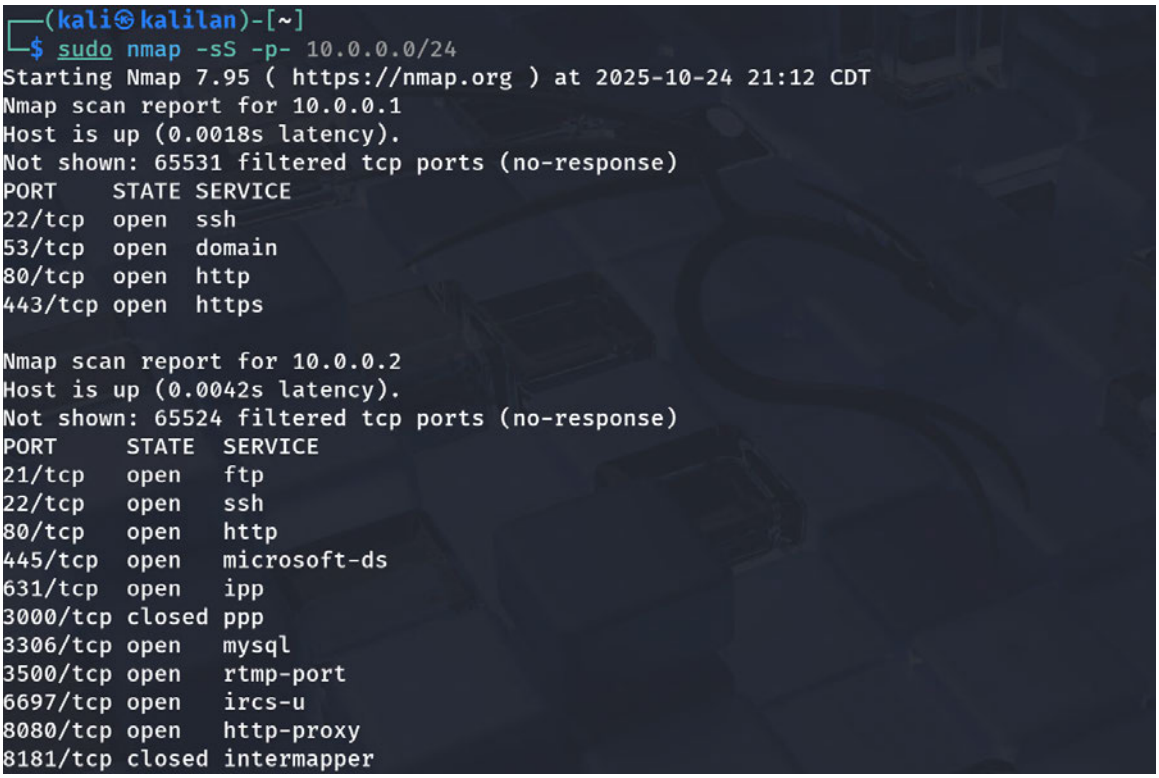
```
sudo nmap -sS -p- <ip_del_servidor o rango_de_ips>
```

Nota. Escaneo SYN con nmap para identificación de puertos abiertos en etapa de reconocimiento. Elaboración propia.

En este comando, se utiliza la opción `-sS` para realizar un escaneo TCP SYN, la opción `-p-` para escanear todos los puertos (del 1 al 65535) y finalmente se especifica la dirección IP o el rango de direcciones IP que se desea escanear.

Al ejecutar este comando, se envían paquetes SYN a todos los puertos del servidor o rango de direcciones IP especificado. Si un puerto está abierto, el servidor responde con un paquete SYN-ACK, lo que indica que el puerto está disponible para conexiones. Si un puerto está cerrado, el servidor responde con un paquete RST, lo que indica que el puerto no está disponible. Esto permite identificar los servicios en ejecución en el servidor y posibles vulnerabilidades asociadas a esos servicios.

Figura 185. Resultado del escaneo TCP SYN en Kali Linux.



Nota. Listado de puertos abiertos y servicios detectados mediante nmap en la red especificada. Elaboración propia.

8.1.4. Escaneo UDP

Para simular un ataque de escaneo UDP, se utiliza la herramienta *nmap*. Esta herramienta permite enviar paquetes UDP personalizados y es ideal para simular ataques de reconocimiento. El comando para ejecutar el ataque es el siguiente:

Cuadro 67. Comando para simular un ataque de escaneo UDP.

```
sudo nmap -sU -p- <ip_del_servidor o rango_de_ips>
```

Nota. Escaneo UDP con nmap para enumeración de servicios sin conexión en etapa de reconocimiento. Elaboración propia.

En este comando, se utiliza la opción `-sU` para realizar un escaneo UDP, la opción `-p-` para escanear todos los puertos (del 1 al 65535) y finalmente se especifica la dirección IP o el rango de direcciones IP que se desea escanear.

Al ejecutar este comando, se envían paquetes UDP a todos los puertos del servidor o rango de direcciones IP especificado.

- *closed*: el *host* devuelve un ICMP *port unreachable* (tipo 3, código 3).
- *open*: el puerto responde con un datagrama válido (p. ej., *banner* o paquete propio del servicio).
- *open/filtered*: no hay respuesta; el puerto puede estar abierto o filtrado por el *firewall*.
- *filtered*: llega un ICMP de filtrado o limitación que impide determinar el estado.

Figura 186. Resultado del escaneo UDP en Kali Linux.

Nota. Listado de puertos UDP abiertos y servicios detectados mediante nmap en la red especificada. Elaboración propia.

En la figura 186 se puede observar un segmento del resultado del escaneo UDP. El resultado indica que el *host* está activo y, aunque se hayan escaneado todos los puertos UDP indica que están en estado *ignored* ya que no se ha recibido respuesta alguna de estos puertos. Sin embargo, sigue siendo útil para identificar posibles servicios en ejecución en el servidor.

8.1.5. Detección de versiones/NSE

Es tipo de escaneo pregunta a los servicios que corren en un equipo qué programa exacto están usando y qué versión y con esa información, un atacante puede buscar vulnerabilidades conocidas de esa versión. El nmap *Scripting engine* (NSE) permite ejecutar *scripts* automáticos que hacen pruebas adicionales, ya sea confirmar versiones, o hasta revisar configuraciones débiles o listar información pública del servicio.

Why importa? Porque conocer qué hay y en qué versión está es el primer paso para elegir un *exploit*. Por eso, aunque es una técnica legítima para auditorías, también es vista como un "ataque de reconocimiento".

Para simular un ataque de detección de versiones/NSE, se utiliza la herramienta *nmap* que viene preinstalada en Kali Linux. Esta herramienta permite enviar paquetes TCP personalizados y es ideal para simular ataques de reconocimiento. El comando para ejecutar el ataque es el siguiente:

Cuadro 68. Comando para simular un ataque de detección de versiones/NSE.

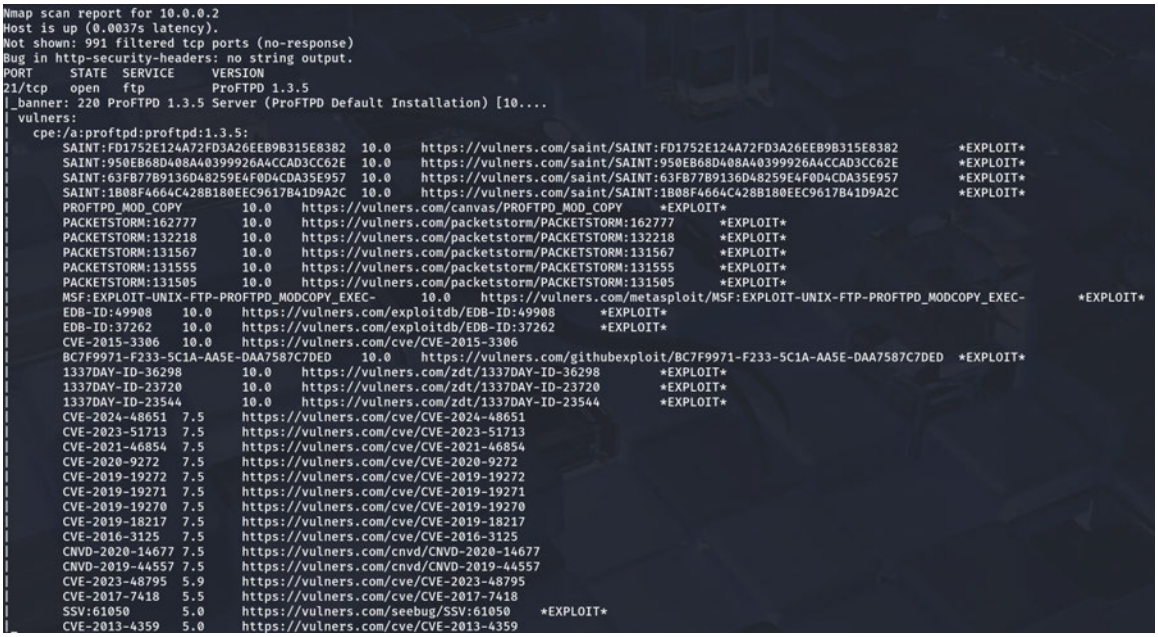
```
1 sudo nmap -sV --script safe,default -T3 <ip_del_servidor o rango_de_ips>
```

Nota. Escaneo nmap con detención de versiones y ejecución de *scripts* NSE para identificación de vulnerabilidades. Elaboración propia.

En este comando, se utiliza la opción `-sV` para detectar versiones de servicios, la opción `-script safe,default` para ejecutar *scripts* NSE seguros y predeterminados, la opción `-T3` para establecer la velocidad del escaneo en un nivel moderado y finalmente se especifica la dirección IP o el rango de direcciones IP que se desea escanear.

Al ejecutar este comando, se envían paquetes TCP a los puertos del servidor o rango de direcciones IP especificado. Si un puerto está abierto, el servidor responde con información sobre el servicio que está en ejecución en ese puerto, incluyendo la versión del software. Los *scripts* NSE también pueden proporcionar información adicional sobre posibles vulnerabilidades asociadas a esos servicios. Esto permite identificar los servicios en ejecución en el servidor y posibles vulnerabilidades asociadas a esos servicios.

Figura 187. Resultado de la detección de versiones/NSE en Kali Linux.



Nota. Listado detallado de servicios, versiones y resultados de *scripts* NSE ejecutados. Elaboración propia.

En la figura 187 se puede observar un segmento del resultado del escaneo de detección de versiones/NSE. En este se puede ver una lista con las vulnerabilidades encontradas en el servidor web Metasploitable 3 relacionadas con FTP.

8.1.6. *Banner grabbing* (HTTP/FTP)

Para simular un ataque de *banner grabbing*, se utiliza la herramienta *netcat* que viene preinstalada en Kali Linux. Esta herramienta permite enviar paquetes TCP personalizados y es ideal para simular ataques de reconocimiento. El comando para ejecutar el ataque es el siguiente:

Cuadro 69. Comando para simular un ataque de *banner grabbing*.

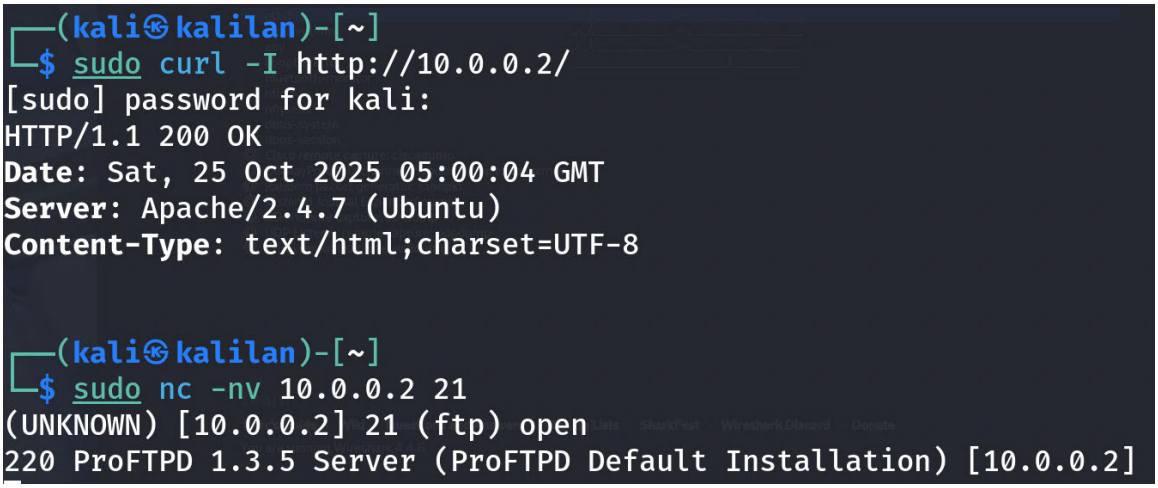
```
sudo curl -I http://<ip_del_servidor>/
sudo nc -nv <ip_del_servidor> 21
```

Nota. Técnicas *curl* y *netcat* para extraer *banners* de servicios HTTP y FTP identificando software y versiones. Elaboración propia.

En este comando, se utiliza la herramienta *curl* para enviar una solicitud HTTP *HEAD* al servidor web Metasploitable 3 y obtener el *banner* del servidor web. Luego, se utiliza la herramienta *netcat* para conectarse al puerto 21 del servidor FTP y obtener el *banner*. Se

debe reemplazar `<ip_del_servidor>` por la dirección IP del servidor web. Al ejecutar estos comandos, se reciben respuestas del servidor web y del servidor FTP que contienen información sobre el software y la versión en ejecución en esos servicios. Esto permite identificar posibles vulnerabilidades asociadas a esos servicios.

Figura 188. Resultado del *banner grabbing* en Kali Linux.



Nota. Banners obtenidos de servicios HTTP y FTP mostrando versiones y detalles del software. Elaboración propia.

8.1.7. SSH *Brute force* hacia el servidor web Metasploitable 3

Para simular un ataque de SSH *Brute force*, se utiliza la herramienta *Hydra* que viene preinstalada en Kali Linux. Esta herramienta permite realizar ataques de fuerza bruta contra servicios de autenticación y es ideal para simular ataques de este tipo. El comando para ejecutar el ataque es el siguiente:

Cuadro 70. Comando para simular un ataque de SSH *Brute force*.

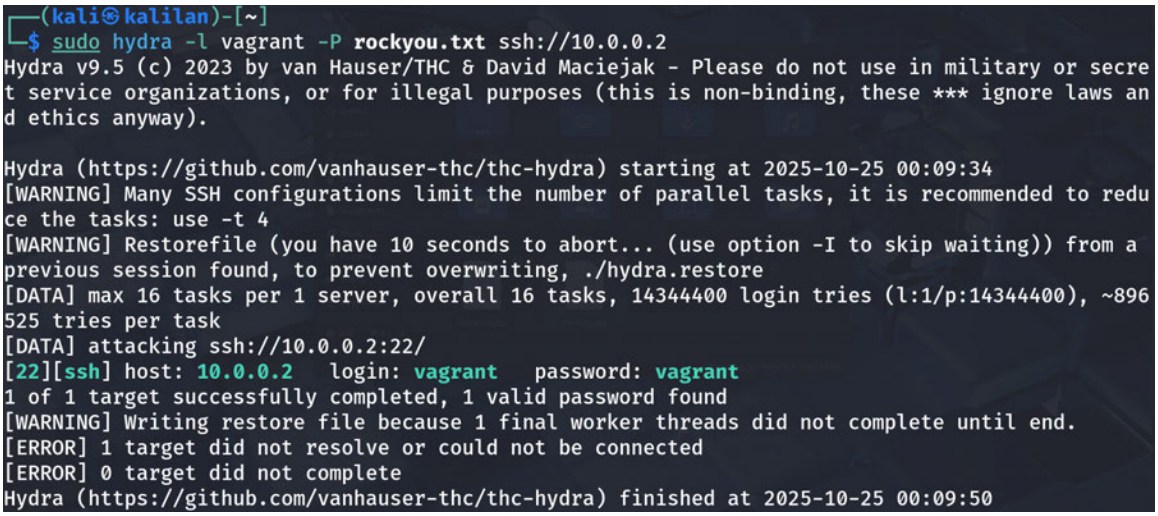
```
1 sudo hydra -l <usuario> -P rockyou.txt ssh://<ip_del_servidor>
```

Nota. Ataque de fuerza bruta con Hydra contra servicio SSH usando diccionario rockyou.txt. Elaboración propia.

En este comando, se utiliza la opción `-l` para especificar el nombre de usuario que se desea atacar, la opción `-P` para especificar la ruta del archivo de contraseñas que se utiliza en el ataque (en este caso, el archivo `rockyou.txt` que viene preinstalado en Kali Linux) y finalmente se especifica la dirección IP del servidor web Metasploitable 3. Se debe reemplazar `<usuario>` por el nombre de usuario que se desea atacar y `<ip_del_servidor>` por la dirección IP del servidor web. Al ejecutar este comando, se intentan múltiples combinaciones de contraseñas para el usuario especificado hasta encontrar la contraseña correcta o agotar la lista de contraseñas. Esto permite identificar cuentas con contraseñas débiles o comunes

que pueden ser vulnerables a ataques de fuerza bruta.

Figura 189. Resultado del ataque de SSH *Brute force* en Kali Linux.



Nota. Detección exitosa de credenciales mediante ataque de fuerza bruta con Hydra. Elaboración propia.

El archivo `rockyou.txt` se encuentra en la ruta `/usr/share/wordlists/rockyou.txt` y contiene una lista de más de 14 millones de contraseñas comunes que se utilizan en ataques de fuerza bruta. El usuario y contraseña de Metasploitable 3 son *vagrant*, que se encuentra en el archivo. Es importante mencionar que este archivo está comprimido en formato `.gz`, por lo que se debe descomprimir antes de utilizarlo. Para descomprimir el archivo, se debe ejecutar el siguiente comando:

Cuadro 71. Comando para descomprimir el archivo rockyou.txt.

```
1 sudo gzip -d /usr/share/wordlists/rockyou.txt.gz
2 sudo mv /usr/share/wordlists/rockyou.txt ./
```

Nota. Descompresión del archivo rockyou.txt.gz y movimiento a directorio actual para uso con Hydra. Elaboración propia.

8.1.8. FTP *Brute force* hacia el servidor web Metasploitable 3

Para simular un ataque de FTP *Brute force*, se utiliza la herramienta *Hydra* que viene preinstalada en Kali Linux. Esta herramienta permite realizar ataques de fuerza bruta contra servicios de autenticación y es ideal para simular ataques de este tipo. El comando para ejecutar el ataque es el siguiente:

Cuadro 72. Comando para simular un ataque de FTP *Brute force*.

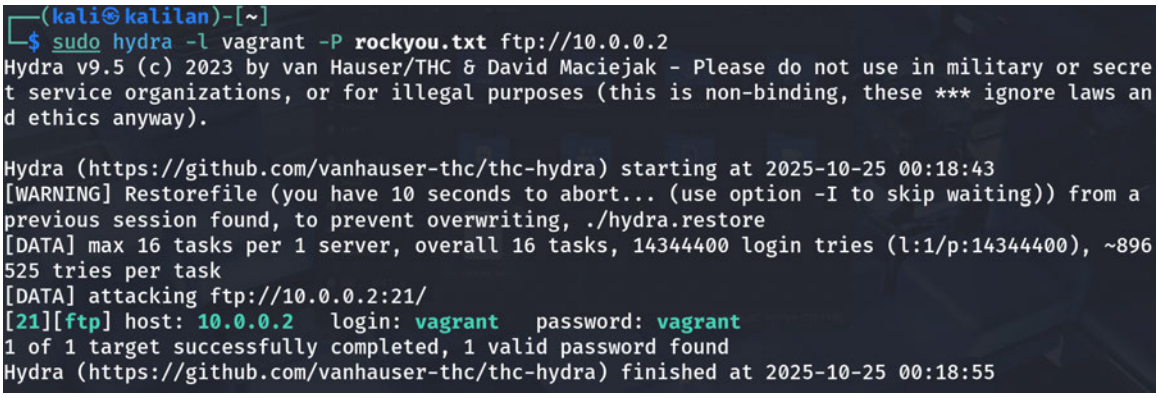
```
sudo hydra -l <usuario> -P rockyou.txt ftp://<ip_del_servidor>
```

Nota. Ataque de fuerza bruta con Hydra contra servicio FTP usando diccionario rockyou.txt. Elaboración propia.

En este comando, se utiliza la opción `-l` para especificar el nombre de usuario que se desea atacar, la opción `-P` para especificar la ruta del archivo de contraseñas que se utiliza en el ataque (en este caso, el archivo `rockyou.txt` que viene preinstalado en Kali Linux) y finalmente se especifica la dirección IP del servidor web Metasploitable 3. Se debe reemplazar `<usuario>` por el nombre de usuario que se desea atacar y `<ip_del_servidor>` por la dirección IP del servidor web.

Al ejecutar este comando, se intentan múltiples combinaciones de contraseñas para el usuario especificado hasta encontrar la contraseña correcta o agotar la lista de contraseñas. Esto permite identificar cuentas con contraseñas débiles o comunes que pueden ser vulnerables a ataques de fuerza bruta.

Figura 190. Resultado del ataque de FTP *Brute force* en Kali Linux.



Nota. Detección exitosa de credenciales mediante ataque de fuerza bruta con Hydra. Elaboración propia.

8.1.9. SNMP *community guessing* hacia el servidor web Metasploitable 3

Para simular un ataque de SNMP *community guessing*, se utiliza la herramienta *onesixtyone* que viene preinstalada en Kali Linux. Esta herramienta permite realizar ataques de fuerza bruta contra comunidades SNMP y es ideal para simular ataques de este tipo. El comando para ejecutar el ataque es el siguiente:

Cuadro 73. Comando para simular un ataque de SNMP *community guessing*.

```
sudo onesixtyone -c /usr/share/doc/onesixtyone/dict.txt <ip_del_servidor>
```

Nota. Ataque de fuerza bruta con *onesixtyone* contra comunidades SNMP usando diccionario dict.txt. Elaboración propia.

En este comando, se utiliza la opción `-c` para especificar la ruta del archivo de comunidades que se utiliza en el ataque (en este caso, el archivo `dict.txt` que viene preinstalado en Kali Linux) y finalmente se especifica la dirección IP del servidor web Metasploitable 3. Se debe reemplazar `<ip_del_servidor>` por la dirección IP del servidor web.

Al ejecutar este comando, se intentan múltiples combinaciones de comunidades SNMP hasta encontrar la comunidad correcta o agotar la lista de comunidades. Esto permite identificar comunidades débiles o comunes que pueden ser vulnerables a ataques de fuerza bruta. Sin embargo, en este caso, no se logró encontrar ninguna comunidad válida en el servidor web Metasploitable 3.

Figura 191. Resultado del ataque de SNMP *community guessing* en Kali Linux.

Nota. Intentos fallidos de detección de comunidades SNMP mediante ataque de fuerza bruta con *onesixtyone*. Elaboración propia.

8.1.10. ARP *Spoofing*

Para simular un ataque de ARP *Spoofing*, se utiliza la herramienta *arpspoof* que viene preinstalada en Kali Linux. Esta herramienta permite enviar paquetes ARP personalizados y es ideal para simular ataques de suplantación en la capa 2. El comando para ejecutar el ataque es el siguiente:

Cuadro 74. Comando para simular un ataque de barrido ARP.

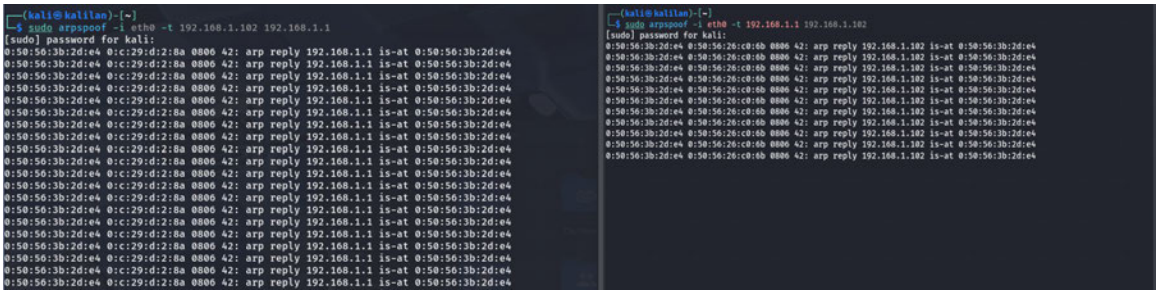
```
sudo arpspoof -i <interfaz_de_red> -t <ip_objetivo> <ip_del_gateway>
sudo arpspoof -i <interfaz_de_red> -t <ip_del_gateway> <ip_objetivo>
```

Nota. Comandos *arpspoof* para suplantación ARP entre víctima y *gateway*, permitiendo interceptar tráfico. Elaboración propia.

Se utilizan dos comandos `arpspoof` para interceptar el tráfico entre la víctima y el *gateway*. En el primer comando, se utiliza la opción `-i` para especificar la interfaz de red que se utiliza (por ejemplo, `eth0`), la opción `-t` para especificar la dirección IP de la víctima y finalmente se especifica la dirección IP del *gateway*. En el segundo comando, se invierten las direcciones IP para interceptar el tráfico en ambas direcciones. Se debe reemplazar `<interfaz_de_red>` por la interfaz de red que se utiliza, `<ip_objetivo>` por la dirección IP de la víctima y `<ip_del_gateway>` por la dirección IP del *gateway*.

Al ejecutar estos comandos, se envían paquetes ARP falsificados a la víctima y al *gateway*, lo que hace que ambos dispositivos actualicen sus tablas ARP con la dirección MAC del atacante. Esto permite al atacante interceptar y redirigir el tráfico entre la víctima y el *gateway*, lo que puede ser utilizado para capturar credenciales o realizar ataques de *Man-in-the-Middle*. Esto también hace que la víctima pierda la conectividad a Internet si no se reenvía el tráfico correctamente.

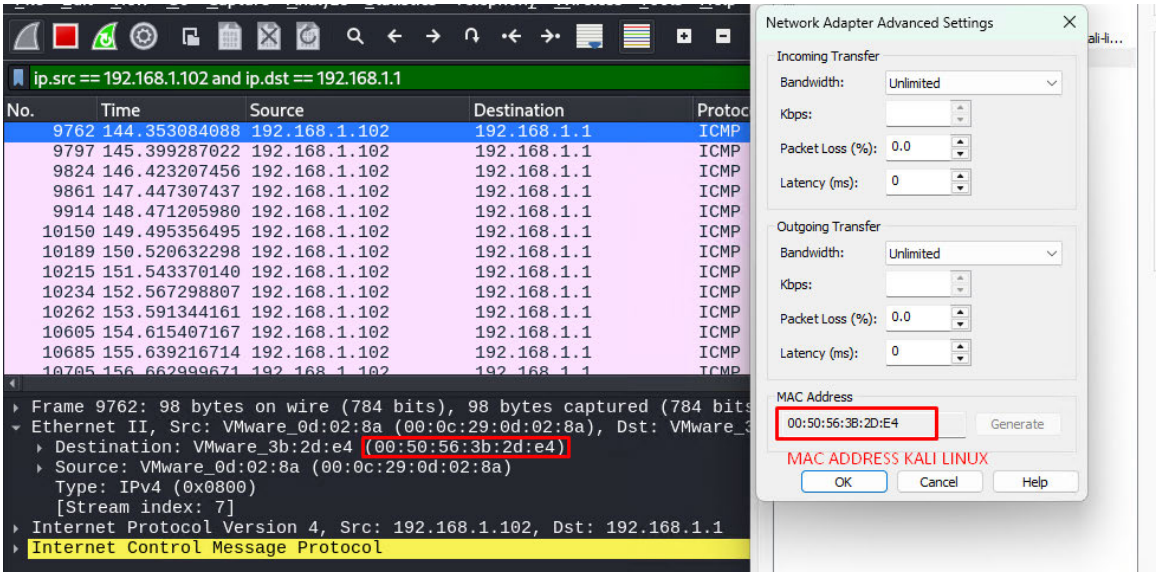
Figura 192. Resultado del ataque de ARP *Spoofing* en Kali Linux.



Nota. Ejecución de *arpspoof* para interceptar tráfico entre víctima y *gateway*. Elaboración propia.

Al intentar navegar por Internet desde la víctima, se observa que la conectividad se pierde y, si además se monitorea el tráfico con *Wireshark* en el atacante, se puede ver que, si en la víctima se intenta hacer un *ping* al *gateway* que normalmente sería 192.168.1.1, el atacante recibe las solicitudes ICMP de eco, al inspeccionar los paquetes, se puede ver que la dirección MAC de destino es la del atacante, confirmando que el ataque de ARP *Spoofing* ha sido exitoso.

Figura 193. Captura de paquetes ICMP durante el ataque de ARP *Spoofing* en Kali Linux.



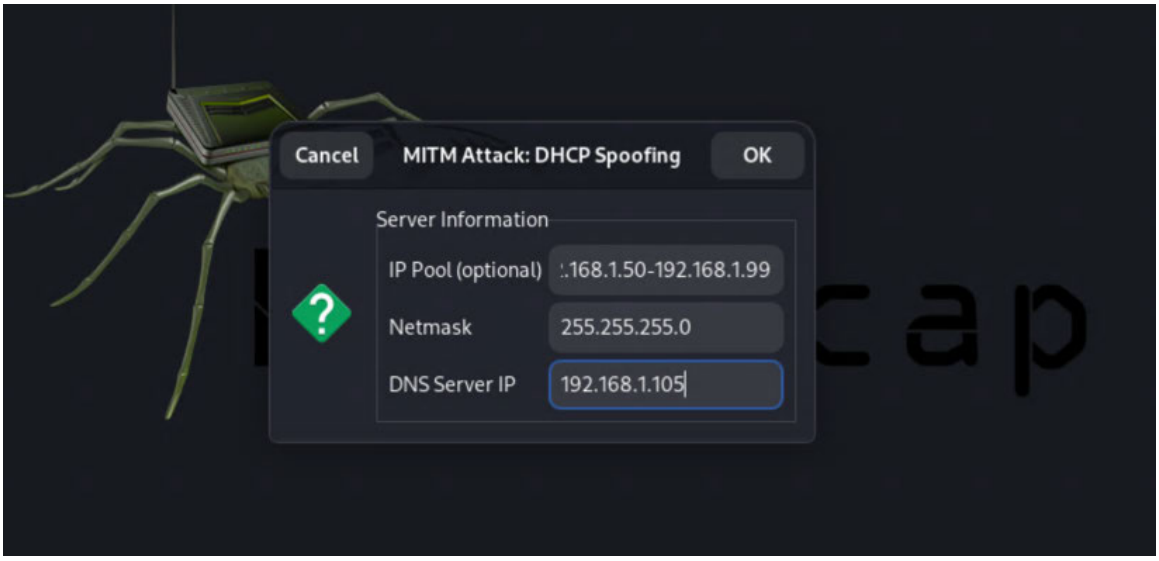
Nota. Captura de paquetes ICMP mostrando que las solicitudes de la víctima son recibidas por el atacante. Elaboración propia.

8.1.11. Servidor DHCP *rogue*

Para simular un ataque de *rogue DHCP Server*, se utiliza la herramienta *ettercap* que viene preinstalada en Kali Linux. Esta herramienta permite crear un servidor DHCP falso y es ideal para simular ataques de este tipo. Esta herramienta tiene una interfaz gráfica que facilita la configuración del servidor DHCP falso. Los pasos para configurar el ataque son los siguientes:

1. Abrir *Ettercap* en modo gráfico ejecutando el comando `sudo ettercap -G` en la terminal de Kali Linux.
2. Seleccionar la interfaz de red que se utiliza para el ataque (por ejemplo, `eth0`).
3. Se hace clic en el cheque en la esquina superior derecha.
4. Seleccionar la opción MITM menu y luego seleccionar la opción DHCP *Spoofing*.
5. Configurar el rango de direcciones IP que se asignan a las víctimas, la máscara de subred y los servidores DNS que se utilizan.
6. Iniciar el ataque haciendo clic en el botón de inicio (play) en la barra de herramientas.

Figura 194. Configuración del ataque de *rogue DHCP server* en Ettercap.

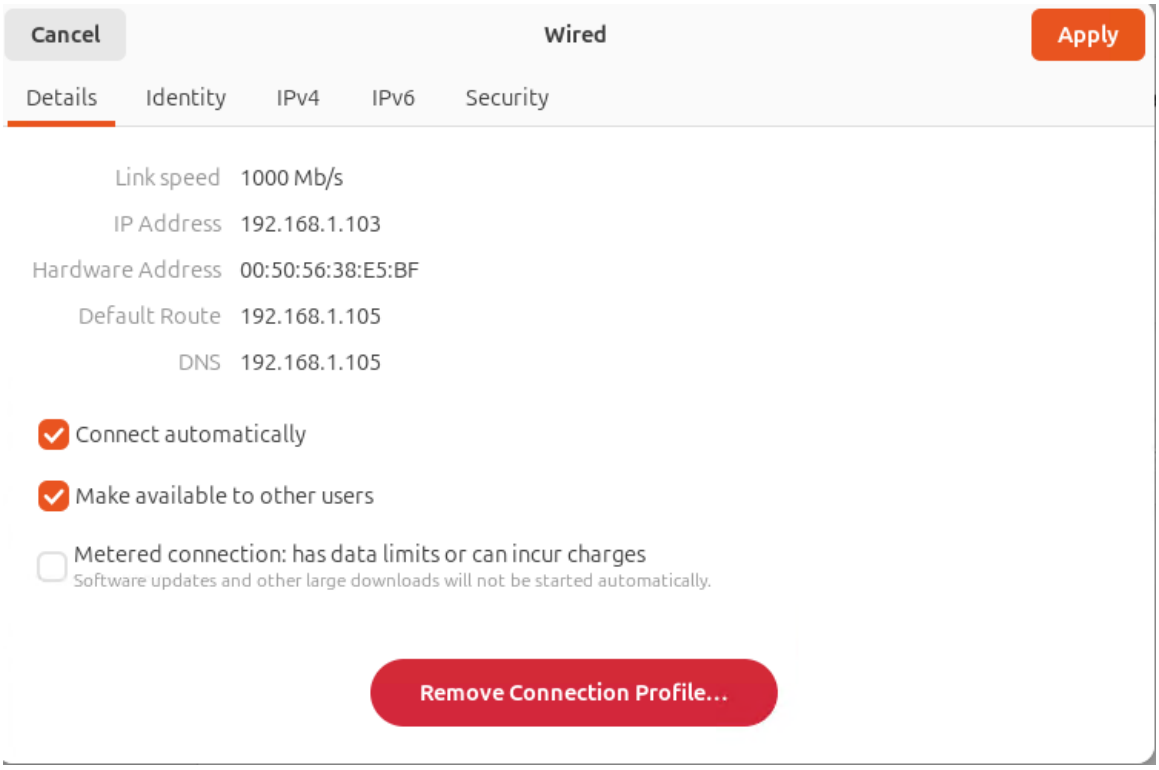


Nota. Interfaz gráfica de Ettercap mostrando la configuración del servidor DHCP falso. Elaboración propia.

En el cliente Ubuntu que actúa como víctima, se va a renovar la concesión DHCP para obtener una nueva dirección IP del servidor DHCP falso. Esto se puede hacer eliminando la configuración de red actual y agregándola nuevamente para forzar la solicitud DHCP.

Al iniciar el ataque, el servidor DHCP falso comienza a responder a las solicitudes DHCP de las víctimas en la red. Cuando una víctima solicite una dirección IP, el servidor DHCP falso le asigna una dirección IP del rango configurado, junto con la máscara de subred y los servidores DNS especificados. Esto permite al atacante redirigir el tráfico de las víctimas a través de su máquina, lo que puede ser utilizado para capturar obtener cualquier credencial o información sensible. Al observar la figura 195, se puede ver que ahora el *gateway* asignado a la víctima es la dirección IP del atacante (Kali Linux).

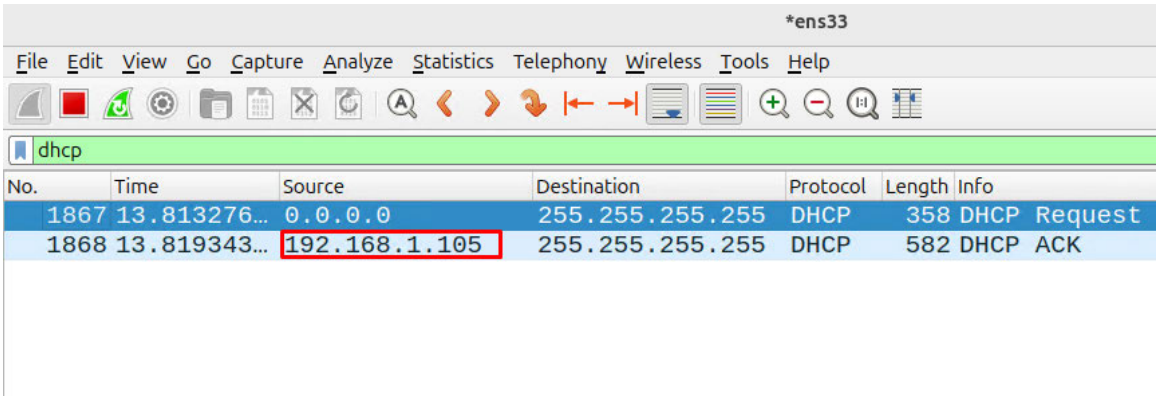
Figura 195. Configuración de red del cliente Ubuntu después del ataque de *rogue DHCP server*.



Nota. Configuración de red del cliente Ubuntu mostrando la dirección IP, máscara de subred y *gateway* asignados por el servidor DHCP falso. Elaboración propia.

Se puede verificar la asignación de la dirección IP por parte del servidor DHCP falso utilizando *Wireshark* en el atacante para capturar los paquetes DHCP en la red. Al inspeccionar los paquetes, se puede ver que el DHCP ACK enviado al cliente contiene la dirección IP, máscara de subred y *gateway* asignados por el servidor DHCP falso.

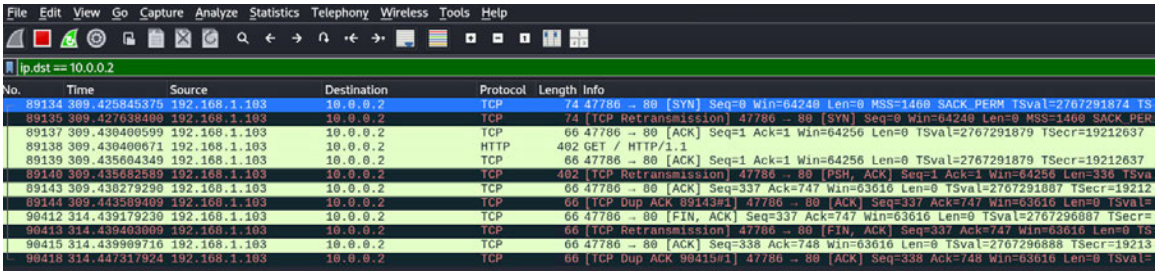
Figura 196. Captura de paquetes DHCP durante el ataque de *rogue DHCP server* en Kali Linux.



Nota. Captura de paquetes DHCP mostrando la asignación de una dirección IP por parte del servidor DHCP falso. Elaboración propia.

Por último, al intentar navegar por Internet desde la víctima, se observa que el tráfico es redirigido a través del atacante, lo que hace vulnerable a la víctima a ataques de *Man-in-the-Middle*.

Figura 197. Captura de paquetes HTTP durante el ataque de *rogue DHCP server* en Kali Linux.



Nota. Captura de paquetes HTTP mostrando que el tráfico de la víctima es redirigido a través del atacante. Elaboración propia.

8.1.12. TCP SYN *flood* hacia el servidor web Metasploitable 3

Para los ataques de denegación de servicio, se utiliza Kali Linux desde la interfaz WAN para atacar al servidor web Metasploitable 3 ubicado en la DMZ, simulando un escenario realista donde un atacante externo intenta saturar los recursos del servidor web.

Para simular un ataque de TCP SYN *flood*, se utiliza la herramienta *hping3* que viene preinstalada en Kali Linux. Esta herramienta permite enviar paquetes TCP personalizados y es ideal para simular ataques de denegación de servicio. El comando para ejecutar el ataque es el siguiente:

Cuadro 75. Comando para simular un ataque de TCP SYN *flood*.

```
sudo hping3 -S --flood -V -p 80 sini58.example.com
```

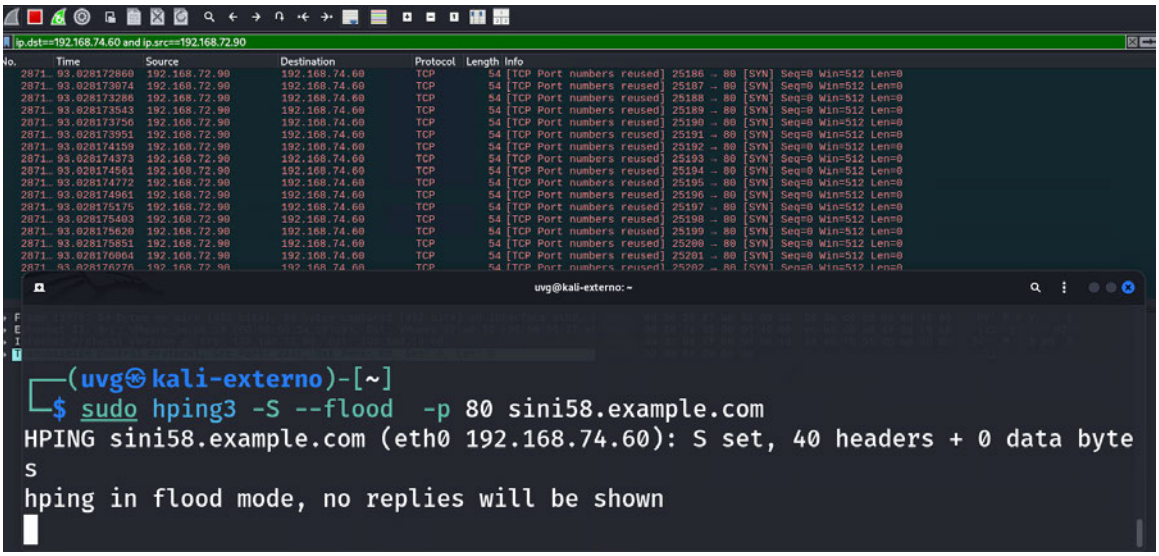
Nota. Ataque de inundación SYN con *hping3* saturando la cola de conexiones del servidor web. Elaboración propia.

En este comando, se utiliza la opción `-S` para enviar paquetes SYN, la opción `-flood` para enviar paquetes lo más rápido posible, la opción `-V` para mostrar información detallada sobre los paquetes enviados y la opción `-p 80` para especificar el puerto de destino (en este caso, el puerto 80 del servidor web Metasploitable 3). Finalmente, se especifica la dirección IP o el nombre de dominio del servidor web Metasploitable 3.

Al ejecutar este comando, se envían múltiples paquetes SYN al servidor web Metasploitable 3, lo que satura la cola de conexiones del servidor y provoca que no pueda atender nuevas solicitudes legítimas. Esto simula un ataque de denegación de servicio, donde el objetivo es agotar los recursos del servidor para interrumpir su funcionamiento normal.

En la figura 198 se puede observar la salida del comando `hping3` en Kali Linux, mostrando el envío masivo de paquetes SYN al servidor web Metasploitable 3, así como la cantidad de paquetes enviados y recibidos de la máquina Metasploitable 3.

Figura 198. Resultado del ataque de TCP SYN *flood* en Kali Linux.

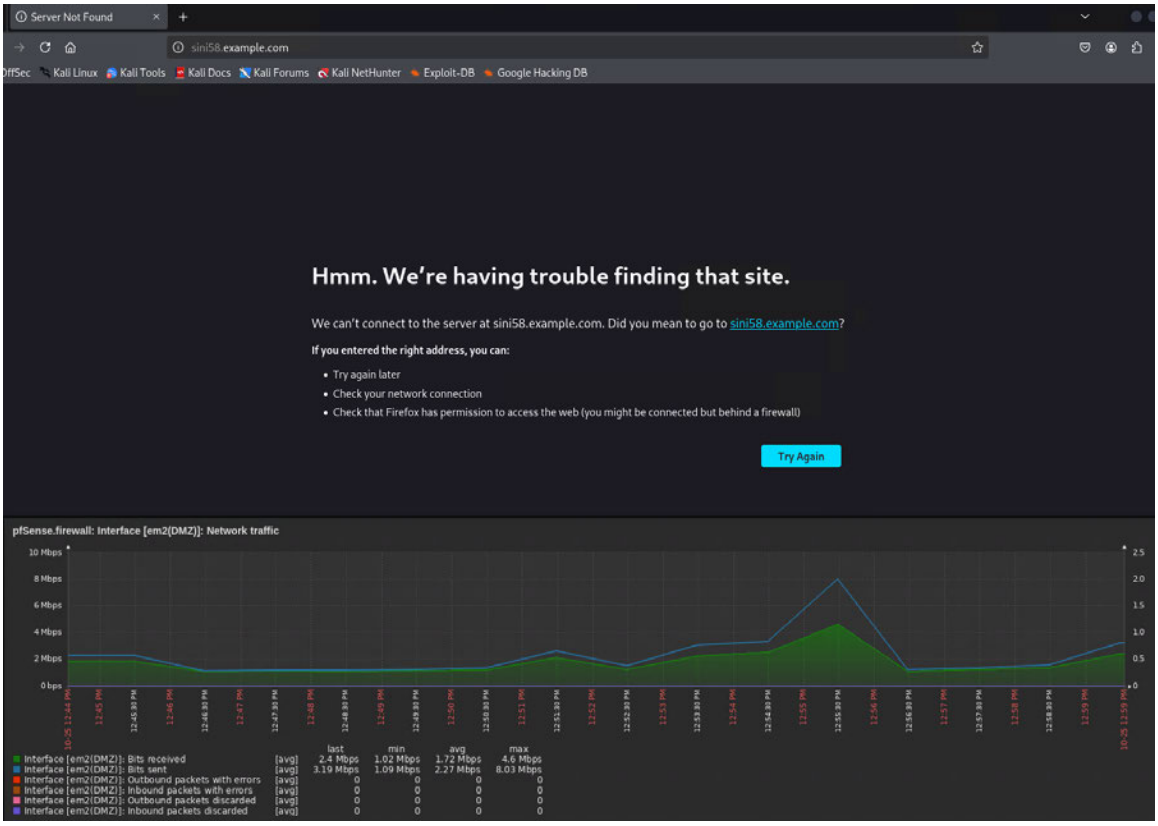


Nota. Ejecución de hping3 mostrando el envío masivo de paquetes SYN al servidor web Metasploitable 3. Elaboración propia.

En la Figura 199 se puede observar el impacto del ataque en el servidor web Metasploitable 3, donde se muestra que el servidor no puede responder a las solicitudes HTTP legítimas debido a la saturación causada por el ataque de TCP SYN *flood*. Hay que destacar que el servidor no tiene capacidad para mitigar este tipo de ataques, y en un entorno real,

para poder llevar a cabo este ataque, se necesitaría una gran cantidad de ancho de banda y recursos desde el lado del atacante.

Figura 199. Impacto del ataque de TCP SYN *flood* en el servidor web Metasploitable 3.



Nota. Intento fallido de acceder al servidor web Metasploitable 3 debido a la saturación causada por el ataque de TCP SYN *flood*. Elaboración propia.

8.1.13. HTTP *GET flood* hacia el servidor web Metasploitable 3

Para simular un ataque de HTTP *GET flood*, se utiliza la herramienta *wrk* que permite realizar pruebas de carga en servidores web y es ideal para simular ataques de denegación de servicio. El comando para ejecutar el ataque es el siguiente:

Cuadro 76. Comando para simular un ataque de HTTP *GET flood*.

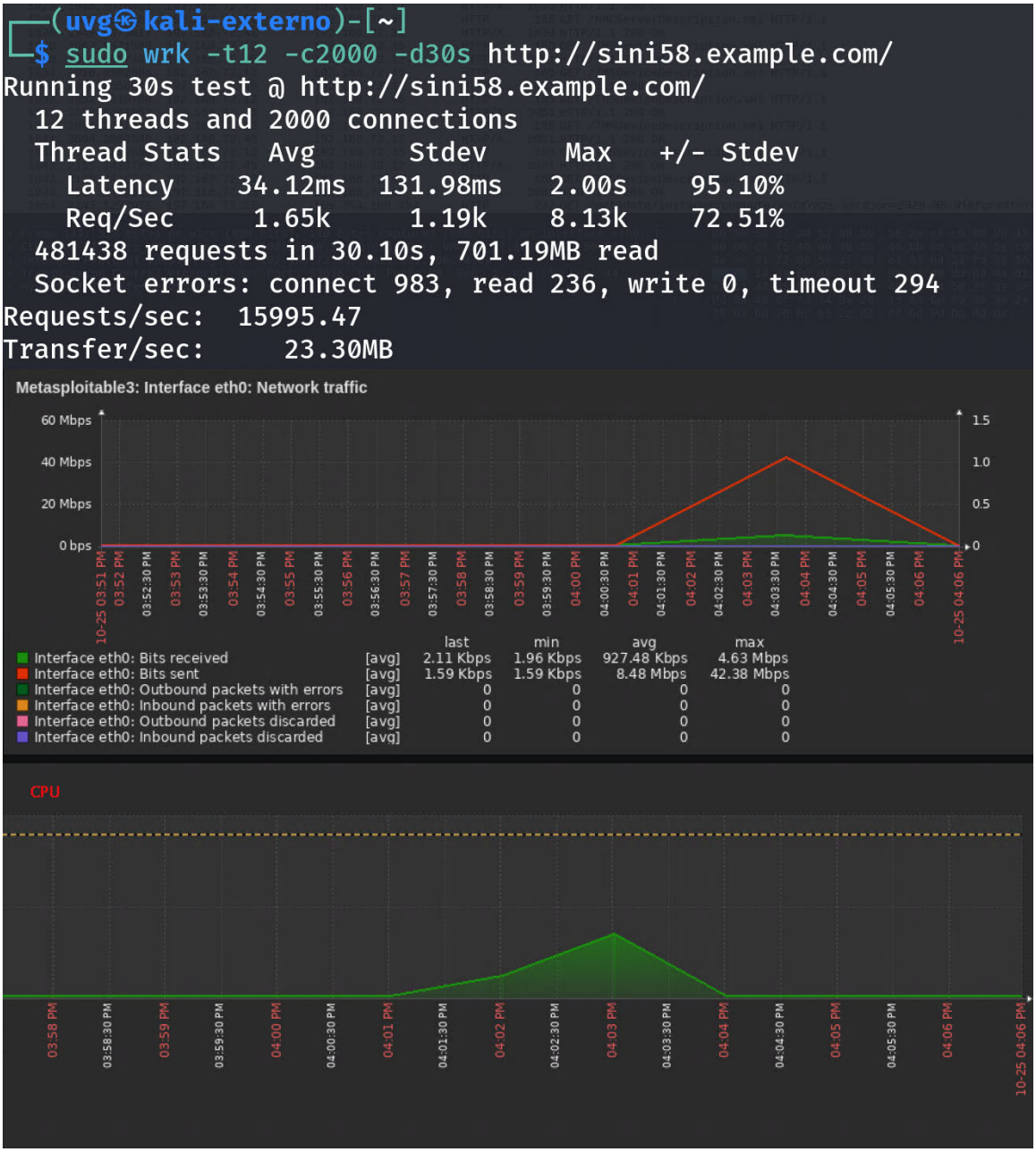
```
sudo wrk -t12 -c2000 -d30s http://sini58.example.com/
```

Nota. Ataque de inundación HTTP *GET* con wrk mediante 2000 conexiones concurrentes durante 30 segundos. Elaboración propia.

En este comando, se utiliza la herramienta *wrk* para enviar múltiples solicitudes HTTP *GET* al servidor web Metasploitable 3. La opción `-t12` especifica el número de hilos a utilizar (en este caso, 12 hilos), la opción `-c2000` especifica el número de conexiones concurrentes (en este caso, 2000 conexiones simultáneas) y la opción `-d30s` especifica la duración de la prueba (en este caso, 30 segundos). Finalmente, se especifica la URL del servidor web Metasploitable 3.

Al ejecutar este comando, se envían múltiples solicitudes HTTP *GET* al servidor web Metasploitable 3, lo que satura los recursos del servidor y provoca que no pueda atender nuevas solicitudes legítimas. Esto simula un ataque de denegación de servicio, donde el objetivo es agotar los recursos del servidor para interrumpir su funcionamiento normal.

Figura 200. Resultado del ataque de HTTP *GET flood* en Kali Linux.



Nota. Ejecución de wrk mostrando el envío masivo de solicitudes HTTP *GET* al servidor web Metasploitable 3 y la saturación de recursos. Elaboración propia.

Se puede observar en la figura 200 que durante el ataque de HTTP *GET flood*, el resultado fue un total de 983 *Socket errors*, lo que indica que el servidor web Metasploitable 3 no pudo manejar la carga de solicitudes y comenzó a rechazar conexiones.

8.2. Desarrollo de reglas personalizadas en Snort y pruebas de detección

Para crear reglas personalizadas en Snort, se debe acceder en pfSense a la sección *Services* en el menú principal y seleccionar la opción Snort. En esta sección se pueden ver las diferentes interfaces de red que tienen Snort habilitado. Se debe habilitar Snort en todas las interfaces de red que se deseen monitorear. En este caso, se habilita Snort en todas las interfaces. Para ello, se debe hacer clic en el icono de lápiz que aparece en la columna de *Edit* para cada interfaz. En la ventana que se abre, se debe marcar la casilla de *Enable* para habilitar Snort en esa interfaz. Se debe repetir este proceso para todas las interfaces. Una vez que se ha habilitado Snort en todas las interfaces, se debe guardar la configuración haciendo clic en el botón de *Save*.

Para agregar reglas personalizadas, se debe ir a la sección Snort *Interfaces* y hacer clic en el icono de lápiz que aparece en la columna de *Edit* para la interfaz en la que se desea agregar las reglas. En la ventana que se abre, se debe ir a la pestaña *Rules* y luego a la subpestaña *Custom rules*. En esta sección se pueden ver las reglas personalizadas que se han creado. Para agregar una nueva regla, se debe escribir la regla en el cuadro de texto que aparece en la parte inferior de la página. Una vez que se ha escrito la regla, se debe hacer clic en el botón de *Save* para guardar la regla.

Lo primero que se debe hacer es crear variables para los segmentos de red que se desean monitorear. En este caso, se crean las variables de la siguiente forma:

Cuadro 77. Variables de red para Snort.

```
ipvar HOME_NET [192.168.1.0/24,10.0.0.0/24,172.16.10.0/24]
ipvar LAN_NET [192.168.1.0/24]
ipvar DMZ_NET [10.0.0.0/24]
ipvar SOCMOC_NET [172.16.10.0/24]
ipvar EXTERNAL_NET !$HOME_NET
```

Nota. Definición de variables para segmentos de red internos y externos utilizadas en reglas personalizadas. Elaboración propia.

8.2.1. Regla personalizada para detección de barrido ICMP

Para detectar un barrido ICMP, se puede crear la siguiente regla personalizada en Snort:

Cuadro 78. Regla personalizada para detección de barrido ICMP.

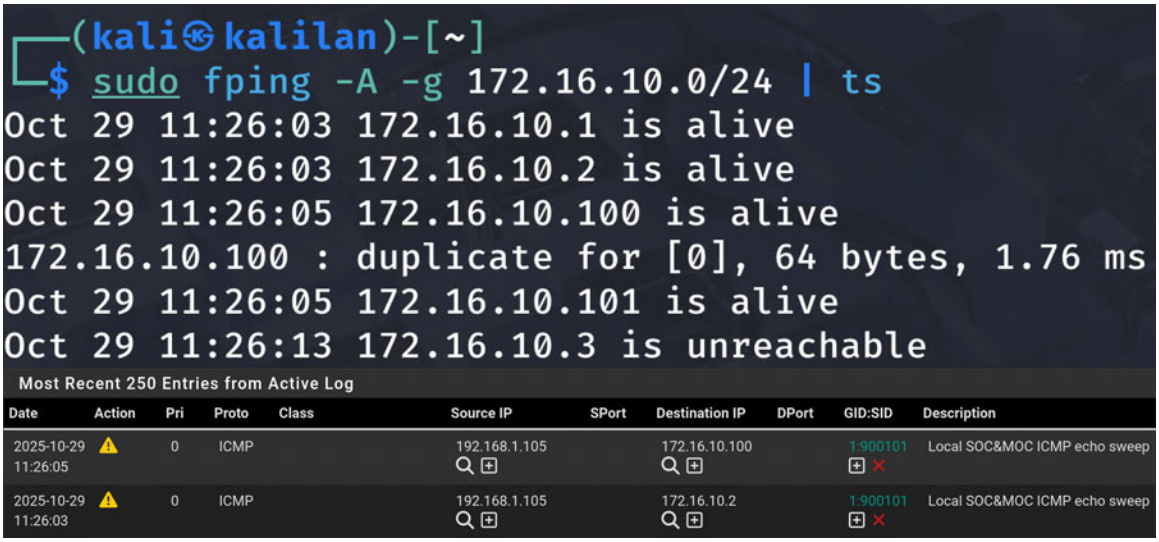
```
alert icmp $HOME_NET any -> $HOME_NET any (msg:"Local ICMP echo sweep";
    itype:8; threshold:type both, track by_src, count 10, seconds 3; sid
    :900100; rev:1;)
```

Nota. Regla Snort para detectar barrido ICMP cuando se alcanzan 10 solicitudes desde un mismo origen en 3 segundos. Elaboración propia.

Esta regla genera una alerta cuando se detectan 10 paquetes ICMP de tipo 8 (ICMP *Echo Request*) provenientes de cualquier dirección IP interna hacia cualquier dirección IP interna en un período de 3 segundos. El parámetro threshold indica que se debe activar la alerta si por lo menos 10 paquetes son detectados en un período de 3 segundos de una misma dirección IP de origen. El identificador único de la regla es 900100 y la revisión es 1.

Para poder probar la regla, se ejecuta el comando de barrido ICMP visto en la sección anterior, y luego se verifica que la alerta se haya generado correctamente en Snort.

Figura 201. Alerta generada por la regla personalizada de barrido ICMP en Snort.



Nota. Alerta en Snort indicando la detección de un barrido ICMP. Elaboración propia.

Al observar la Figura 201, se puede ver que la alerta se ha generado correctamente, indicando que se ha detectado un barrido ICMP en la red interna. Además, por el *timestamp* de ejecución del comando y de la alerta, se nota el tiempo casi inmediato en que se generó la alerta después de ejecutar el comando.

8.2.2. Regla personalizada para detección de escaneo TCP SYN

Para detectar un escaneo TCP SYN, se puede crear la siguiente regla personalizada en Snort:

Cuadro 79. Regla personalizada para detección de escaneo TCP SYN.

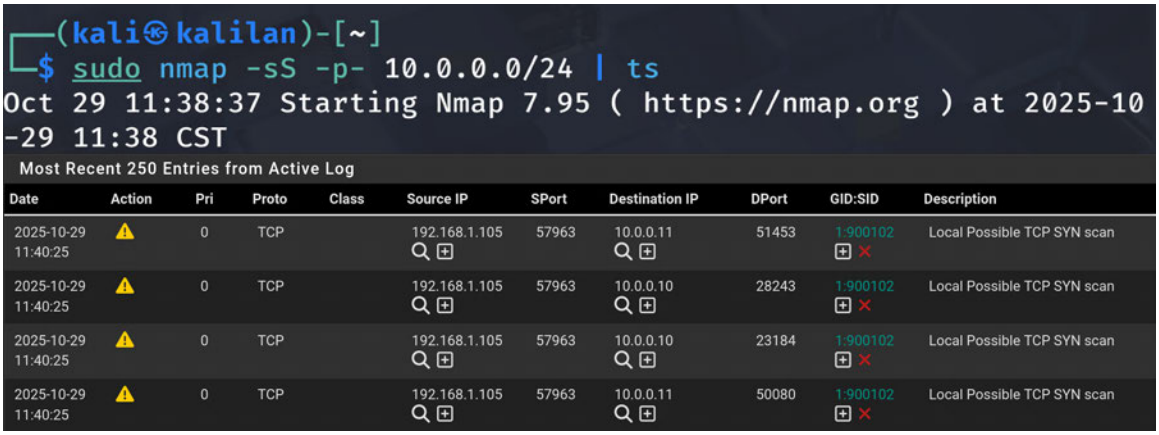
```
alert tcp $HOME_NET any -> $HOME_NET 1:65535 (flags:S; flow:stateless; msg:"
    Local Possible TCP SYN scan"; detection_filter:track by_src, count 80,
    seconds 5; sid:900102; rev:1;)
```

Nota. Regla Snort para detectar escaneo TCP SYN cuando se alcanzan 80 paquetes SYN desde un mismo origen en 5 segundos. Elaboración propia.

Esta regla genera una alerta cuando se detectan 80 paquetes TCP con la bandera SYN activada (indicando el inicio de una conexión) provenientes de cualquier dirección IP interna hacia cualquier dirección IP interna en un rango de puertos del 1 al 65535. Los parámetros de *detection_filter* indican que se debe activar la alerta si por lo menos 80 paquetes son detectados en un período de 5 segundos de una misma dirección IP de origen. El parámetro *flags:S* especifica que solo se deben monitorear los paquetes con la bandera SYN activada. El identificador único de la regla es 900102 y la revisión es 1.

Para poder probar la regla, se ejecuta el comando de escaneo TCP SYN visto en la sección anterior, y luego se verifica que la alerta se haya generado correctamente en Snort.

Figura 202. Alerta generada por la regla personalizada de escaneo TCP SYN en Snort.



Nota. Alerta en Snort indicando la detección de un escaneo TCP SYN. Elaboración propia.

Al observar la Figura 202, se puede ver que la alerta se ha generado correctamente, indicando que se ha detectado un escaneo TCP SYN en la red interna.

8.2.3. Regla personalizada para detección de escaneo UDP

Para detectar un escaneo UDP, se puede crear la siguiente regla personalizada en Snort:

Cuadro 80. Regla personalizada para detección de escaneo UDP.

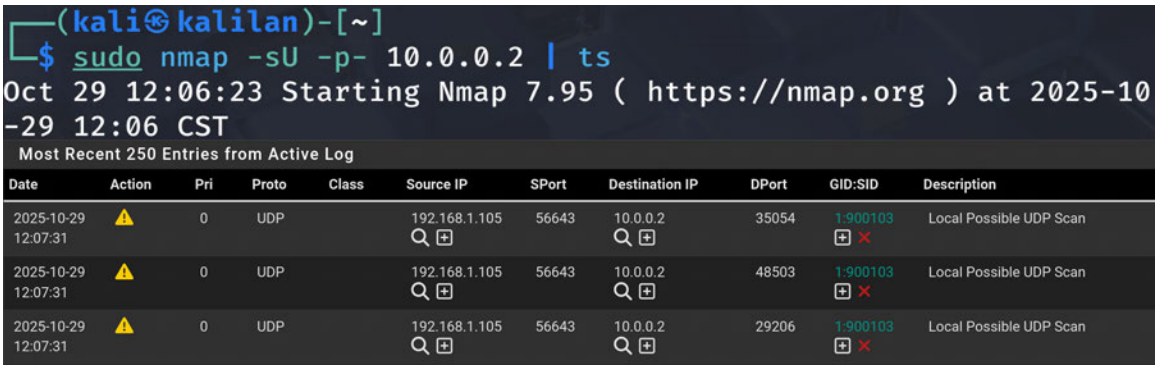
```
alert udp $HOME_NET any -> $HOME_NET 1:65535 (msg:"Local Possible UDP Scan";
    detection_filter:track by_src, count 80, seconds 5; sid:900103; rev:1;)
```

Nota. Regla Snort para detectar escaneo UDP cuando se alcanzan 80 paquetes UDP desde un mismo origen en 5 segundos. Elaboración propia.

Esta regla genera una alerta cuando se detectan 80 paquetes UDP provenientes de cualquier dirección IP interna hacia cualquier dirección IP interna en un rango de puertos del 1 al 65535. Los parámetros de *detection_filter* indican que se debe activar la alerta si por lo menos 80 paquetes son detectados en un período de 5 segundos de una misma dirección IP de origen. El identificador único de la regla es 900103 y la revisión es 1.

Para poder probar la regla, se ejecuta el comando de escaneo UDP visto en la sección anterior, y luego se verifica que la alerta se haya generado correctamente en Snort.

Figura 203. Alerta generada por la regla personalizada de escaneo UDP en Snort.



Nota. Alerta en Snort indicando la detección de un escaneo UDP. Elaboración propia.

Al observar la Figura 203, se puede ver que la alerta se ha generado correctamente, indicando que se ha detectado un escaneo UDP en la red interna.

8.2.4. Regla personalizada para detección de versiones/NSE

Para detectar un escaneo de detección de versiones/NSE, se puede crear la siguiente regla personalizada en Snort:

Cuadro 81. Regla personalizada para detección de versiones/NSE.

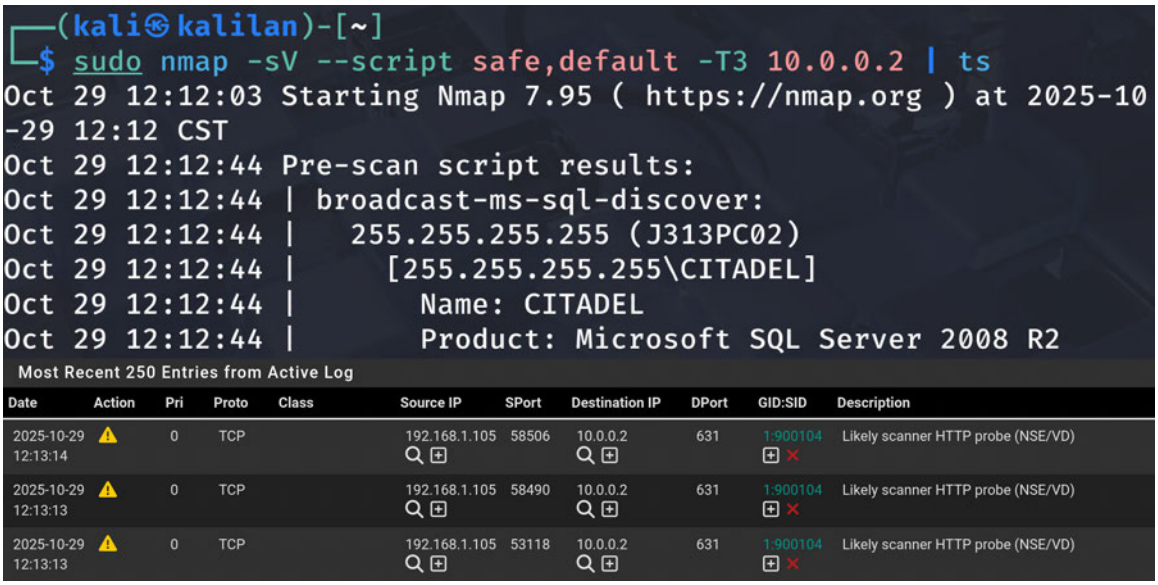
```
alert tcp $HOME_NET any -> $HOME_NET any (msg:"Likely scanner HTTP probe (
    NSE/VD)"; flow:to_server,established; http_header; content:"User-Agent|3
    a| "; content:"Nmap Scripting Engine"; distance:0; within:80;
    detection_filter:track by_src, count 5, seconds 60; sid:900104; rev:1;)
```

Nota. Regla Snort para detectar escaneo de detección de versiones/NSE basado en el encabezado *User-Agent*. Elaboración propia.

Esta regla genera una alerta cuando se detectan 5 paquetes TCP con el encabezado HTTP *User-Agent* que contiene la cadena "Nmap *Scripting Engine*", lo que indica que se está utilizando la herramienta nmap con *scripts* NSE para detectar versiones de servicios. Los parámetros de *detection_filter* indican que se debe activar la alerta si por lo menos 5 paquetes son detectados en un período de 60 segundos de una misma dirección IP de origen. El parámetro *flow:to_server,established* especifica que solo se deben monitorear los paquetes que van hacia el servidor y que pertenecen a una conexión establecida. El identificador único de la regla es 900104 y la revisión es 1.

Para poder probar la regla, se ejecuta el comando de escaneo de detección de versiones/NSE visto en la sección anterior, y luego se verifica que la alerta se haya generado correctamente en Snort.

Figura 204. Alerta generada por la regla personalizada de detección de versiones/NSE en Snort.



Nota. Alerta en Snort indicando la detección de un escaneo de versiones/NSE. Elaboración propia.

Al observar la Figura 204, se puede ver que la alerta se ha generado correctamente, indicando que se ha detectado un escaneo de detección de versiones/NSE en la red interna.

8.2.5. Regla personalizada para *Banner grabbing* HTTP/FTP

Para detectar un intento de *Banner grabbing* en servicios HTTP o FTP, se puede crear la siguiente regla personalizada en Snort:

Cuadro 82. Regla personalizada para detección de *Banner grabbing* HTTP/FTP.

```
1 alert tcp $HOME_NET any -> $HOME_NET 80 (msg:"HTTP banner grab with curl -I
      (HEAD)"; flow:to_server,established; content:"HEAD" ; content:"User-
      Agent|3a| "; content:"curl/"; nocase; distance:0; within:20; sid:900106;
       rev:1;)
2
3 alert tcp $HOME_NET 21 -> $HOME_NET any (msg:"FTP server banner (220)"; flow
      :to_client,established; content:"220 "; sid:900105; rev:1;)
```

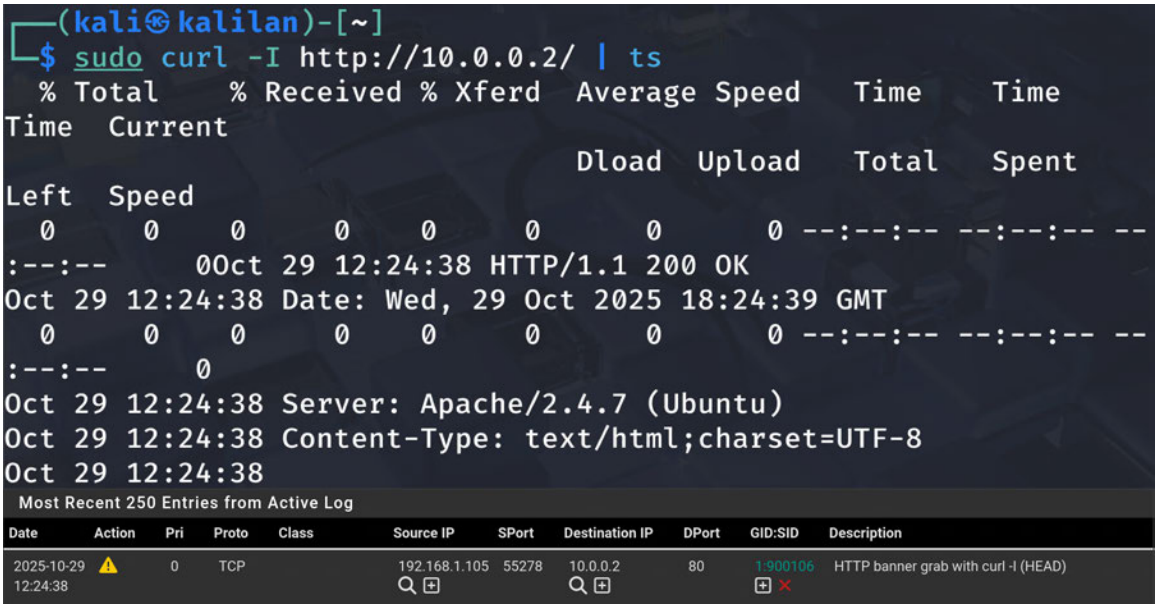
Nota. Reglas Snort para detectar intentos de *banner grabbing* en HTTP con *curl* y en FTP con respuesta 220. Elaboración propia.

La primera regla genera una alerta cuando se detecta un paquete TCP dirigido al puerto 80 (HTTP) que contiene la cadena "*HEAD*.[en] el contenido del paquete, lo que indica un intento de *banner grabbing* utilizando el comando *HEAD* de HTTP. Además, se busca la cadena "*curl*/.[en] el encabezado `User-Agent`, lo que sugiere que se está utilizando la herramienta *curl* para realizar el *banner grabbing*. El identificador único de la regla es 900106 y la revisión es 1.

La segunda regla genera una alerta cuando se detecta un paquete TCP proveniente del puerto 21 (FTP) que contiene la cadena "220 .[en] el contenido del paquete, lo que indica la respuesta del servidor FTP al establecer una conexión. El identificador único de la regla es 900105 y la revisión es 1.

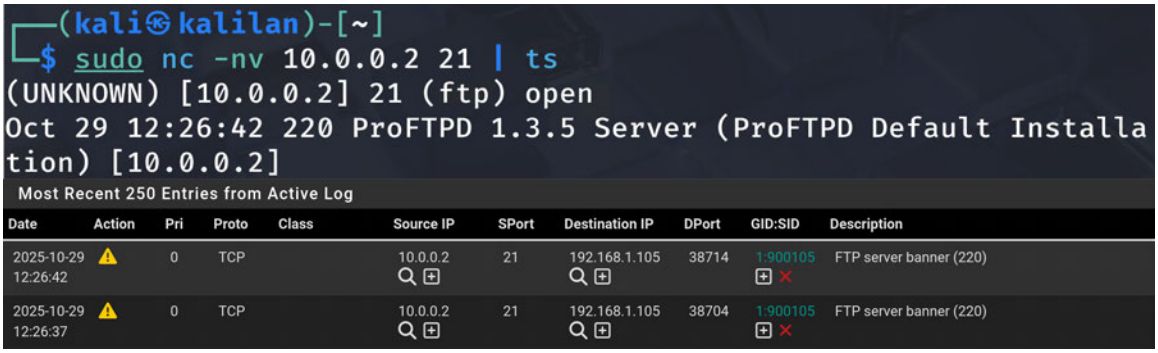
Para poder probar las reglas, se ejecutan los comandos de *banner grabbing* HTTP y FTP vistos en la sección anterior, y luego se verifica que las alertas se hayan generado correctamente en Snort.

Figura 205. Alerta generada por la regla personalizada de *banner grabbing* HTTP en Snort.



Nota. Alerta en Snort indicando la detección de un intento de *banner grabbing* en servicio HTTP. Elaboración propia.

Figura 206. Alerta generada por la regla personalizada de *banner grabbing* FTP en Snort.



Nota. Alerta en Snort indicando la detección de un intento de *banner grabbing* en servicio FTP. Elaboración propia.

Al observar las Figuras 205 y 206, se puede ver que las alertas se han generado correctamente, indicando que se ha detectado un intento de *banner grabbing* en los servicios HTTP y FTP en la red interna.

8.2.6. Regla personalizada para detección de ataque de fuerza bruta SSH

Para detectar un ataque de fuerza bruta SSH, se puede crear la siguiente regla personalizada en Snort:

Cuadro 83. Regla personalizada para detección de ataque de fuerza bruta SSH.

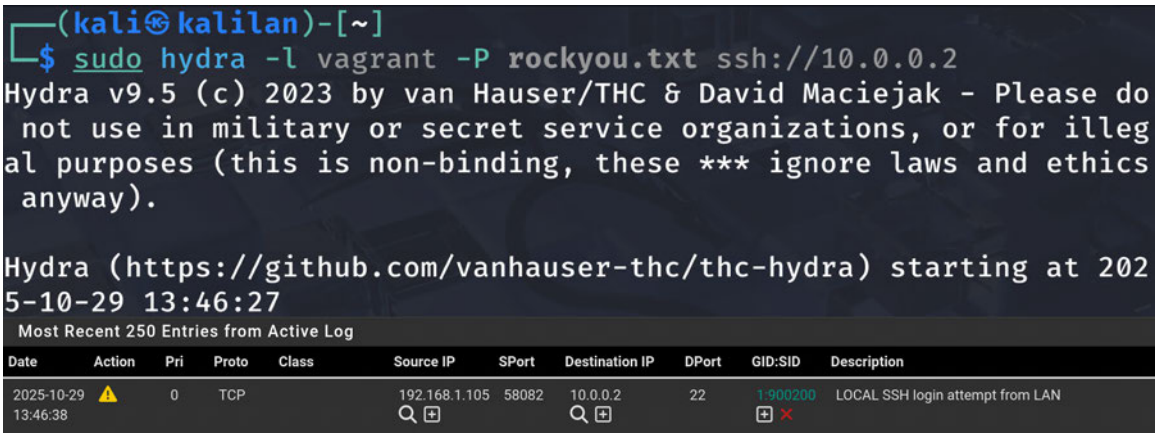
```
alert tcp $LAN_NET any -> 10.0.0.2 22 (flow:to_server,established; msg:"
    LOCAL SSH login attempt from LAN"; content:"SSH-"; threshold:type both,
    track by_src, count 8, seconds 60; sid:900200; rev:1;)
```

Nota. Regla Snort para detectar fuerza bruta SSH cuando se alcanzan 8 intentos de conexión en 60 segundos. Elaboración propia.

Esta regla genera una alerta cuando se detectan 8 intentos de conexión SSH (paquetes TCP con la cadena "SSH-.[en] el contenido) provenientes de cualquier dirección IP en la red LAN hacia la dirección IP 10.0.0.2. El parámetro threshold indica que se debe activar la alerta si por lo menos 8 intentos son detectados en un período de 60 segundos de una misma dirección IP de origen. El identificador único de la regla es 900200 y la revisión es 1.

Para poder probar la regla, se ejecuta el comando de ataque de fuerza bruta SSH visto en la sección anterior, y luego se verifica que la alerta se haya generado correctamente en Snort.

Figura 207. Alerta generada por la regla personalizada de ataque de fuerza bruta SSH en Snort.



Nota. Alerta en Snort indicando la detección de un ataque de fuerza bruta SSH. Elaboración propia.

Al observar la Figura 207, se puede ver que la alerta se ha generado correctamente, indicando que se ha detectado un ataque de fuerza bruta SSH en la red interna.

8.2.7. Regla personalizada para detección de ataque de fuerza bruta FTP

Para detectar un ataque de fuerza bruta FTP, se puede crear la siguiente regla personalizada en Snort:

Cuadro 84. Regla personalizada para detección de ataque de fuerza bruta FTP.

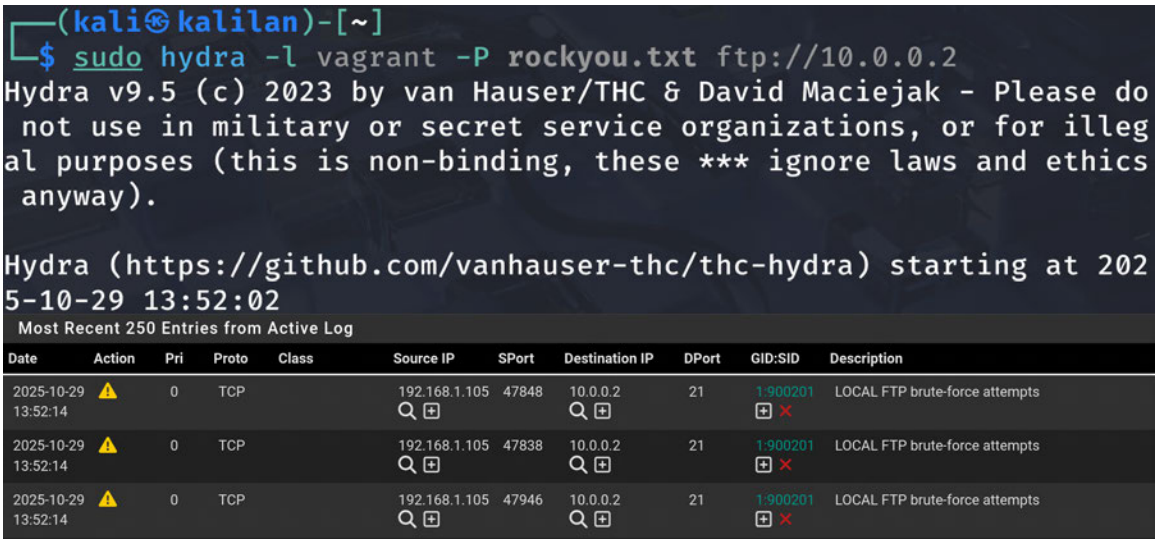
```
alert tcp $LAN_NET any -> 10.0.0.2 21 (flow:to_server,established; msg:"
    LOCAL FTP brute-force attempts"; content:"USER "; nocase;
    detection_filter:track by_src, count 10, seconds 30; sid:900201; rev:1;)
```

Nota. Regla Snort para detectar fuerza bruta FTP cuando se alcanzan 10 intentos de conexión en 30 segundos. Elaboración propia.

Esta regla genera una alerta cuando se detectan 10 intentos de conexión FTP (paquetes TCP con la cadena "*USER*.[en] el contenido) provenientes de cualquier dirección IP en la red LAN hacia la dirección IP 10.0.0.2. El parámetro *detection_filter* indica que se debe activar la alerta si por lo menos 10 intentos son detectados en un período de 30 segundos de una misma dirección IP de origen. El identificador único de la regla es 900201 y la revisión es 1.

Para poder probar la regla, se ejecuta el comando de ataque de fuerza bruta FTP visto en la sección anterior, y luego se verifica que la alerta se haya generado correctamente en Snort.

Figura 208. Alerta generada por la regla personalizada de ataque de fuerza bruta FTP en Snort.



Nota. Alerta en Snort indicando la detección de un ataque de fuerza bruta FTP. Elaboración propia.

Al observar la Figura 208, se puede ver que la alerta se ha generado correctamente, indicando que se ha detectado un ataque de fuerza bruta FTP en la red interna.

8.2.8. Regla personalizada para detección de SNMP *community guessing*

Para detectar un intento de adivinanza de comunidad SNMP, se puede crear la siguiente regla personalizada en Snort:

Cuadro 85. Regla personalizada para detección de SNMP *community guessing*.

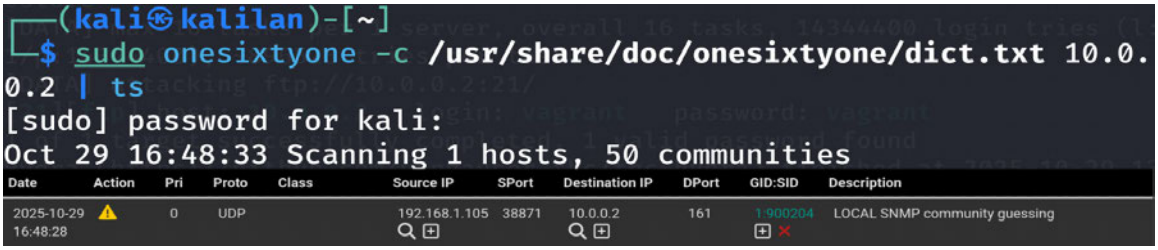
```
alert udp $HOME_NET any -> $HOME_NET 161 (msg:"LOCAL SNMP community guessing
    "; content:"public"; nocase; sid:900204; rev:1;)
```

Nota. Regla Snort para detectar intentos de adivinanza de comunidad SNMP usando cadena predeterminada. Elaboración propia.

Esta regla genera una alerta cuando se detecta un paquete UDP dirigido al puerto 161 (SNMP) que contiene la cadena "*public*.[en] el contenido del paquete, lo que indica un intento de adivinanza de comunidad SNMP utilizando la comunidad predeterminada "*public*". El identificador único de la regla es 900204 y la revisión es 1.

Para poder probar la regla, se ejecuta el comando de adivinanza de comunidad SNMP visto en la sección anterior, y luego se verifica que la alerta se haya generado correctamente en Snort.

Figura 209. Alerta generada por la regla personalizada de SNMP *community guessing* en Snort.



Nota. Alerta en Snort indicando la detección de un intento de adivinanza de comunidad SNMP. Elaboración propia.

Al observar la Figura 209, se puede ver que la alerta se ha generado correctamente, indicando que se ha detectado un intento de adivinanza de comunidad SNMP en la red interna.

8.2.9. Regla personalizada para detección de Arp *Spoofing*

Antes de crear las reglas personalizadas para detectar un ataque de ARP *Spoofing*, es importante añadir el preprocesador de ARP en Snort. Este preprocesador permite a Snort analizar los paquetes ARP y detectar posibles ataques de ARP *Spoofing*. Para habilitar el preprocesador de ARP en pfSense, se debe ir a la sección Snort *Interfaces* y hacer clic en el icono de lápiz que aparece en la columna de *Edit* para la interfaz en la que se desea agregar el preprocesador. En la ventana que se abre, se debe ir a la pestaña *Preprocs* y luego a la subpestaña ARP *spoof setection*. En esta sección se debe marcar la casilla de *Enable ARP spoof detection* para habilitar el preprocesador. En esta sección se puede añadir pares MAC-IP para así hacer que Snort sepa cuáles son las direcciones MAC e IP legítimas en la red.

Figura 210. Habilitación del preprocesador de ARP *spoof detection* en Snort.

Nota. Formulario de configuración del preprocesador de ARP *spoof detection* en Snort. Elaboración propia.

En teoría basta con habilitar el preprocesador de ARP *spoof detection* para que Snort pueda detectar ataques de ARP *Spoofing*. Sin embargo, realizando distintas pruebas se llego a la conclusión que no se añaden automáticamente las reglas necesarias. Entonces, para detectar un ataque de ARP *Spoofing*, se necesitan crear tres reglas personalizadas en Snort, una para cada tipo de ataque:

Cuadro 86. Reglas personalizadas para detección de ARP *Spoofing*.

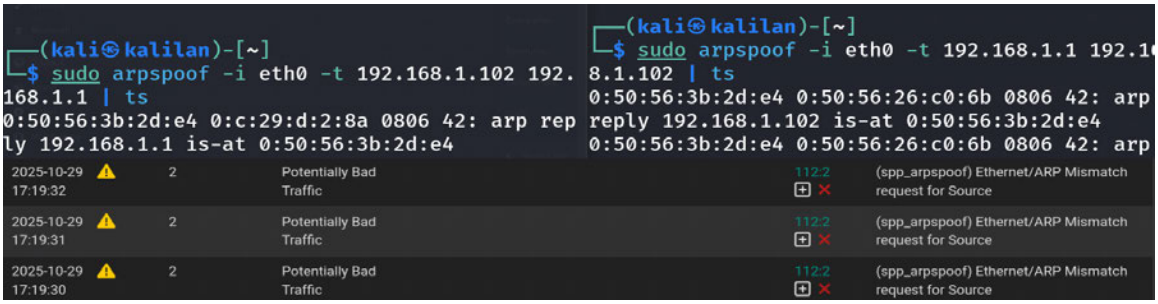
```
1 alert ( msg: "ARPSPOOF_ETHERFRAME_ARP_MISMATCH_SRC"; sid: 2; gid: 112; rev:
      1; metadata: rule-type preproc ; classtype:bad-unknown; )
2
3 alert ( msg: "ARPSPOOF_ETHERFRAME_ARP_MISMATCH_DST"; sid: 3; gid: 112; rev:
      1; metadata: rule-type preproc ; classtype:bad-unknown; )
4
5 alert ( msg: "ARPSPOOF_ARP_CACHE_OVERWRITE_ATTACK"; sid: 4; gid: 112; rev:
      1; metadata: rule-type preproc ; classtype:bad-unknown; )
```

Nota. Tres reglas Snort para detectar discrepancias MAC y sobrescritura de caché ARP indicativas de *spoofing*. Elaboración propia.

Estas reglas generan alertas cuando se detectan inconsistencias en los paquetes ARP que indican un posible ataque de ARP *Spoofing*. La primera regla detecta discrepancias en la dirección MAC de origen, la segunda regla detecta discrepancias en la dirección MAC de destino, y la tercera regla detecta intentos de sobrescribir la caché ARP. Los identificadores

únicos de las reglas son 2, 3 y 4 respectivamente, y todas tienen una revisión de 1. Para poder probar las reglas, se ejecuta el comando de ataque de *ARP Spoofing* visto en la sección anterior, y luego se verifica que las alertas se hayan generado correctamente en Snort.

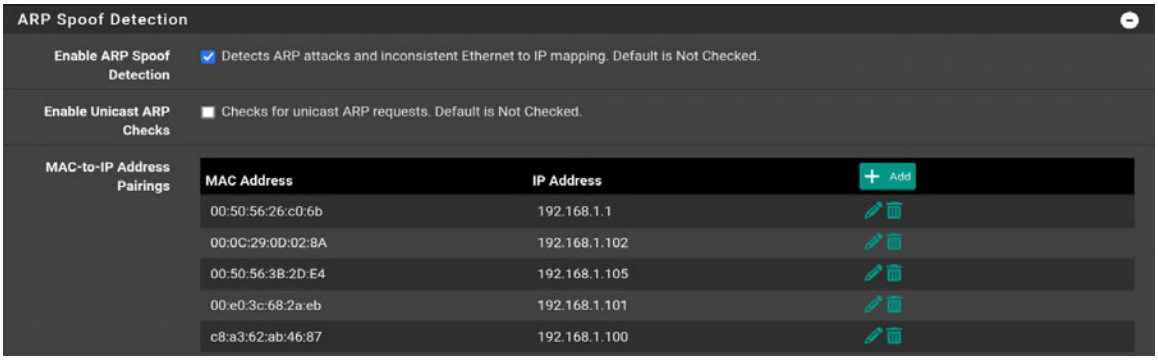
Figura 211. Alertas generadas por las reglas personalizadas de ARP *Spoofing* en Snort.



Nota. Alertas en Snort indicando la detección de un ataque de ARP *Spoofing*. Elaboración propia.

Al observar la Figura 211, se puede ver que las alertas se han generado correctamente, indicando que se ha detectado un ataque de *ARP Spoofing* en la red interna. En este caso, se puede ver que la alerta generado fue la que tiene el mensaje .ARPSPOOF_ETHERFRAME_ARP_MISMATCH_SRC", lo que indica que se detectó una discrepancia en la dirección MAC de origen en los paquetes ARP. Esto ya que anteriormente se había configurado un par MAC-IP legítimo en el preprocesador de ARP *spoof detection*, y el ataque de ARP *Spoofing* intentó utilizar una dirección MAC diferente para la misma dirección IP.

Figura 212. Tabla de pares MAC-IP del segmento LAN.



Nota. Tabla de pares MAC-IP del segmento LAN para detectar inconsistencias. Elaboración propia.

Cabe resaltar que al ser el ataque de un protocolo de capa 2, Snort no puede bloquear las direcciones MAC que realicen el ataque, pero al menos se puede detectar y generar alertas para que el administrador de la red pueda tomar las medidas necesarias.

8.2.10. Regla personalizada para detección de ataque de Servidor DHCP *rogue*

Para detectar un ataque de Servidor DHCP *rogue*, se puede crear la siguiente regla personalizada en Snort:

Cuadro 87. Regla personalizada para detección de ataque de Servidor DHCP *rogue*.

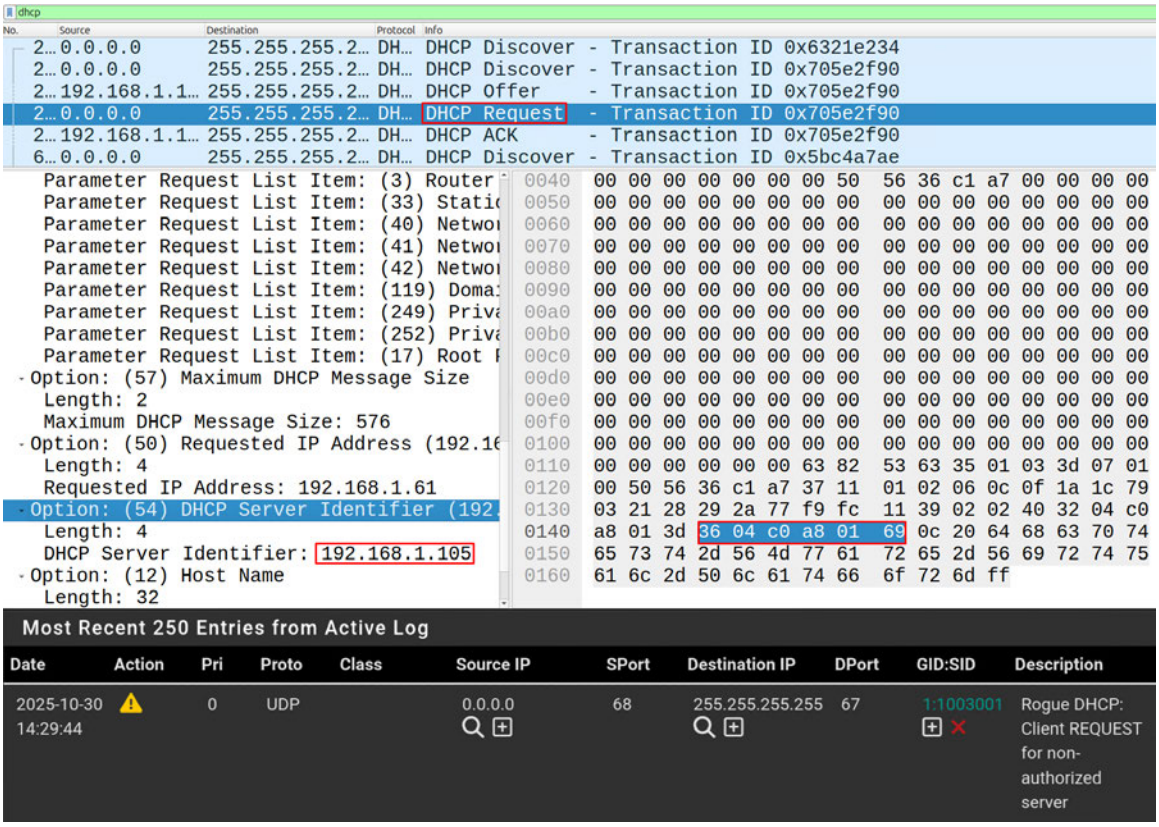
```
alert udp any 68 -> any 67 (msg:"Rogue DHCP: Client REQUEST for non-
    authorized server"; content:"|63 82 53 63|"; content:"|35 01 03|";
    distance:0; within:6; content:"|36 04|";  distance:0; within:40; pcre:"
    /\x36\x04(?!\xC0\xA8\x01\x01)/s"; sid:1003001; rev:1;)
```

Nota. Regla Snort para detectar DHCP *Request* hacia servidor no autorizado verificando *server identifier*. Elaboración propia.

Esta regla genera una alerta cuando se detecta un paquete DHCP *Request* (mensaje con opción 53 igual a 03) enviado por un cliente DHCP que solicita una dirección IP a un servidor DHCP no autorizado. El tráfico analizado corresponde a UDP, desde cualquier dirección IP y puerto 68 (cliente DHCP) hacia cualquier dirección IP y puerto 67 (servidor DHCP). El primer parámetro *content:"|63 82 53 63|"* identifica el cookie mágico de DHCP, que permite confirmar que el paquete pertenece al protocolo DHCP. El segundo parámetro *content:"|35 01 05|"* busca la opción 53 del protocolo DHCP, con el valor 05, que indica que el mensaje es un *Request*. El tercer parámetro *content:"|36 04|"* busca la opción 54, correspondiente al identificador del servidor DHCP (*server identifier*), y el modificador distance y within limitan la búsqueda al rango donde esta opción suele aparecer. El parámetro *pcre:/_x36_x04(?!_xC0_xA8_x01_x01)/s";* utiliza una expresión regular compatible con Perl (PCRE) para verificar que el identificador de servidor DHCP no sea igual a 192.168.1.1 (representado en hexadecimal como C0 A8 01 01). En otras palabras, esta expresión genera una alerta solo si el campo del servidor DHCP es diferente a la dirección IP autorizada 192.168.1.1. El identificador único de la regla es 1003001 y la revisión es 1.

Para poder probar la regla, se ejecuta la serie de pasos para simular un ataque de Servidor DHCP *rogue* visto en la sección anterior, y luego se verifica que la alerta se haya generado correctamente en Snort.

Figura 213. Alerta generada por la regla personalizada de Servidor DHCP *rogue* en Snort.



Nota. Alerta en Snort indicando la detección de un ataque de Servidor DHCP *rogue*. Elaboración propia.

Al observar la Figura 213, se puede ver que la alerta se ha generado correctamente, indicando que se ha detectado un ataque de Servidor DHCP *rogue* en la red interna.

8.2.11. Regla personalizada para detección de TCP SYN *flood*

Para detectar un ataque de TCP SYN *flood*, se puede crear la siguiente regla personalizada en Snort:

Cuadro 88. Regla personalizada para detección de ataque de TCP SYN *flood*.

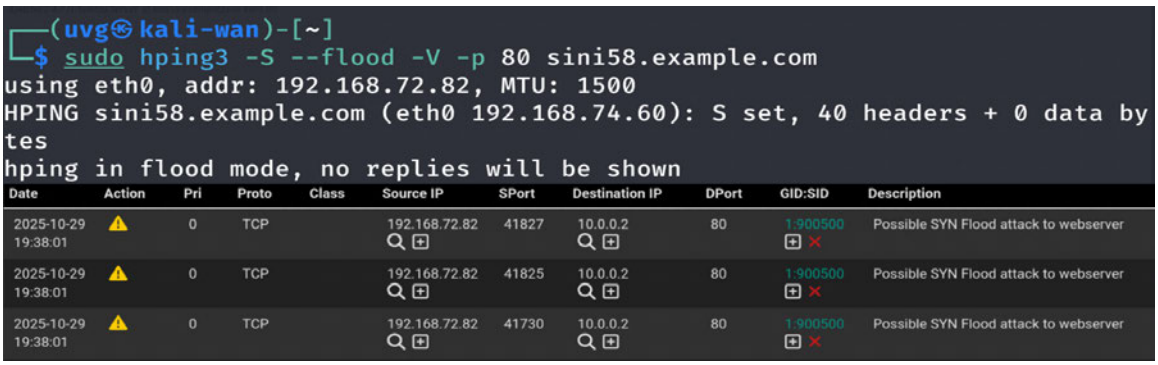
```
alert tcp $EXTERNAL_NET any -> 10.0.0.2 80 (flags:S; msg:"Possible SYN Flood
    attack to webserver"; detection_filter:track by_src, count 200, seconds
    1; sid:900500; rev:1;)
```

Nota. Regla Snort para detectar SYN *flood* cuando se alcanzan 200 paquetes SYN en 1 segundo. Elaboración propia.

Esta regla genera una alerta cuando se detectan 200 paquetes TCP con la bandera SYN activada (indicando el inicio de una conexión) provenientes de cualquier dirección IP externa hacia la dirección IP 10.0.0.2 en el puerto 80. El parámetro *flags:S* indica que se deben monitorear los paquetes con la bandera SYN activada. Los parámetros de *detection_filter* indican que se debe activar la alerta si por lo menos 200 paquetes son detectados en un período de 1 segundo de una misma dirección IP de origen. El identificador único de la regla es 900500 y la revisión es 1.

Para poder probar la regla, se ejecuta el comando de ataque de TCP SYN *flood* visto en la sección anterior, y luego se verifica que la alerta se haya generado correctamente en Snort.

Figura 214. Alerta generada por la regla personalizada de TCP SYN *flood* en Snort.



Nota. Alerta en Snort indicando la detección de un ataque de TCP SYN *flood*. Elaboración propia.

Al observar la Figura 214, se puede ver que la alerta se ha generado correctamente, indicando que se ha detectado un ataque de TCP SYN *flood* en la red interna.

8.2.12. Regla personalizada para detección de HTTP *GET flood*

Para detectar un ataque de HTTP *GET flood*, se puede crear la siguiente regla personalizada en Snort:

Cuadro 89. Regla personalizada para detección de ataque de HTTP *GET flood*.

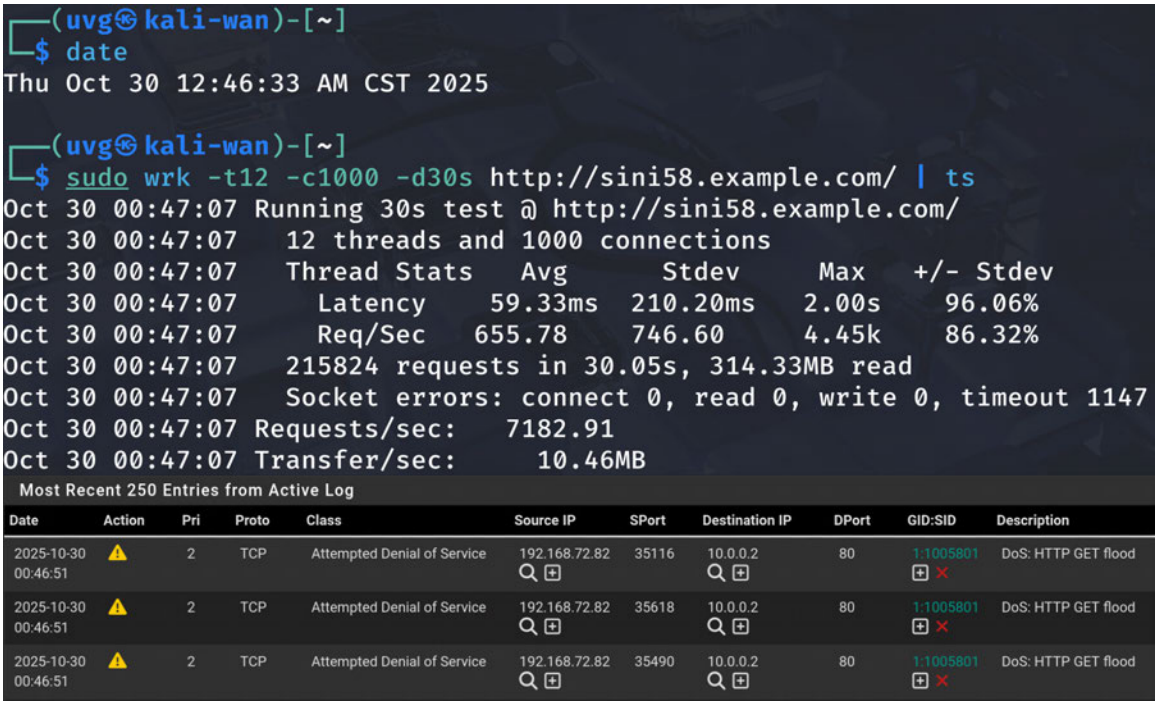
```
alert tcp $EXTERNAL_NET any -> 10.0.0.2 80 (msg:"DoS: HTTP GET flood"; flow:
    to_server,established; content:"GET"; http_method; detection_filter:
    track by_src, count 100, seconds 1; classtype:attempted-dos; sid
    :1005801; rev:1;)
```

Nota. Regla Snort para detectar *GET flood* cuando se alcanzan 100 solicitudes HTTP en 1 segundo. Elaboración propia.

Esta regla genera una alerta cuando se detectan 100 solicitudes HTTP *GET* provenientes de cualquier dirección IP externa hacia la dirección IP 10.0.0.2 en el puerto 80. El parámetro *flow:to_ server,established* especifica que solo se deben monitorear los paquetes que van hacia el servidor y que pertenecen a una conexión establecida. El parámetro *content* especifica que se debe buscar la cadena *GET* en el contenido del paquete. El parámetro *http_ method* indica que se debe analizar el contenido HTTP del paquete. El parámetro *detection_filter* indica que se debe activar la alerta si por lo menos 100 solicitudes son detectadas en un período de 1 segundo de una misma dirección IP de origen. El identificador único de la regla es 1005801 y la revisión es 1.

Para poder probar la regla, se ejecuta el comando de ataque de HTTP *GET flood* visto en la sección anterior, y luego se verifica que la alerta se haya generado correctamente en Snort.

Figura 215. Alerta generada por la regla personalizada de HTTP *GET flood* en Snort.



Nota. Alerta en Snort indicando la detección de un ataque de HTTP *GET flood*. Elaboración propia.

Al observar la Figura 215, se puede ver que la alerta se ha generado correctamente, indicando que se ha detectado un ataque de HTTP *GET flood* en la red interna.

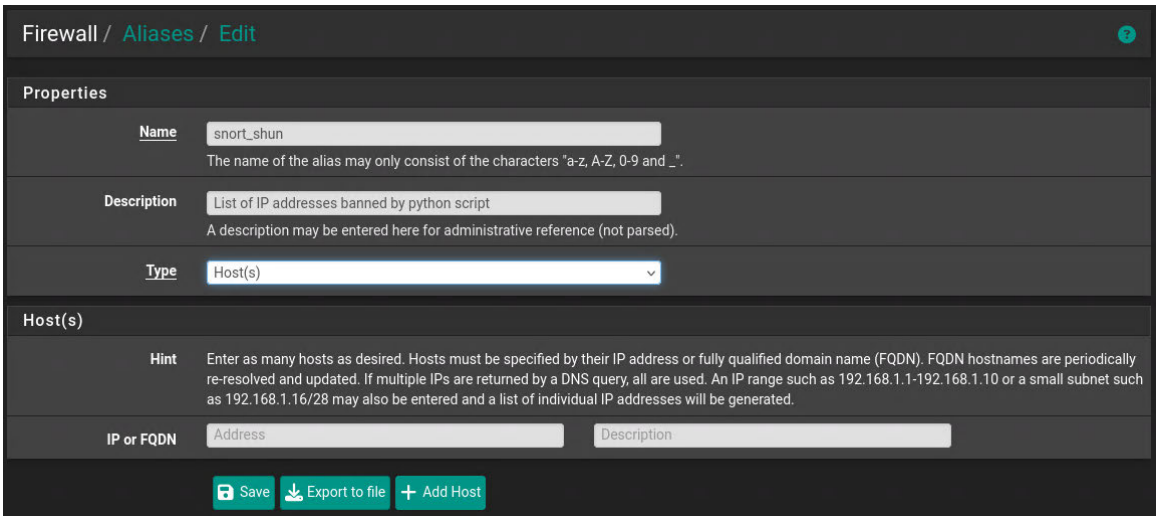
8.3. Bloqueo automático de atacantes basado en alertas de Snort utilizando Python

Python es un lenguaje de programación muy versátil, lo que lo hace ideal para automatizar todo tipo de tareas. En este caso, se utiliza Python para crear un *script* que lea el archivo de alertas de todas las interfaces de Snort y bloquee automáticamente las direcciones IP que generen alertas repetidas en un período de tiempo determinado. Además, de otro *script* que permita desbloquear las direcciones IP bloqueadas si ya no generan alertas en un período de tiempo determinado.

8.3.1. Creación de alias en pfSense para el bloqueo de direcciones IP

Lo primero que se necesita realizar es crear un alias en pfSense que contenga las direcciones IP que se desean bloquear. Para ello, se debe ir a la sección *Firewall* en el menú principal y seleccionar la opción *Aliases*. Para crear un nuevo alias, se debe hacer clic en el botón de *Add*. En la ventana que se abre, se debe proporcionar un nombre para el alias, *snort_ shun* y una descripción opcional. En el campo *Type* se debe seleccionar la opción *Host(s)*. Se guarda la configuración del alias haciendo clic en el botón de *Save*.

Figura 216. Creación de un alias en pfSense para el bloqueo de direcciones IP.



Nota. Formulario de creación de alias para IP bloqueadas por Snort.
Elaboración propia.

8.3.2. Explicación del *script* en Python para bloqueo automático de direcciones IP

El *script* en Python para el bloqueo automático de direcciones IP se encarga de leer el archivo de alertas de Snort y bloquear las direcciones IP que generen alertas repetidas en un período de tiempo determinado. El *script* utiliza la biblioteca *subprocess* para ejecutar

comandos en la terminal de pfSense y la biblioteca *time* para manejar los tiempos de espera. El *script* se ejecuta en un bucle infinito y cada cierto tiempo (por ejemplo, cada 60 segundos) lee el archivo de alertas de Snort y procesa las alertas generadas. Si una dirección IP genera más de un número determinado de alertas (por ejemplo, 5 alertas) en un período de tiempo determinado (por ejemplo, 300 segundos), se bloquea automáticamente esa dirección IP utilizando el alias creado en pfSense. El *script* también se encarga de desbloquear las direcciones IP que ya no generan alertas en un período de tiempo determinado (por ejemplo, 600 segundos).

Cuadro 90. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 1/25.

```
# Autor: Andres Lemus
# Mail: lem21634@uvg.edu.gt
# Fecha de creacion: 15/10/2025
# Version: 1.0
# Descripcion: Script de Auto-Bloqueo para Firewall pfSense basado en alertas de Snort
#!/usr/bin/env python3
# ========================================
# Script de Auto-Bloqueo para Firewall pfSense
# ========================================
# Este script monitorea las alertas generadas por Snort en tiempo real
# y bloquea automaticamente las direcciones IP que generen 3 o mas alertas
# dentro de una ventana de tiempo especifica.
#
# El bloqueo se realiza agregando las IP a un alias en el archivo
# config.xml de pfSense y recargando las reglas del firewall.
# ========================================

# ----------------------------------------
# SECCION DE IMPORTACION DE LIBRERIAS
# ----------------------------------------

# paramiko: Libreria para realizar conexiones SSH y ejecutar comandos remotos
import paramiko

# time: Para manejar pausas y temporizadores en el ciclo de monitoreo
import time

# xml.etree.ElementTree: Para parsear y modificar archivos XML (config.xml)
import xml.etree.ElementTree as ET

# defaultdict: Diccionario que inicializa automaticamente valores por defecto
from collections import defaultdict

# datetime: Para trabajar con fechas y horas (timestamps de alertas)
from datetime import datetime

# ========================================
# SECCION DE CONFIGURACION - PARAMETROS DE CONEXION A PFSENSE
# ========================================
# Estos valores definen como conectarse al servidor pfSense via SSH.
# SSH es un protocolo que permite ejecutar comandos de forma
# segura en un servidor remoto.
# ========================================
```

Nota. Elaboración propia.

Cuadro 91. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 2/25.

```
# Direccion IP del servidor pfSense en la red
PFSENSE_IP = "172.16.10.1"

# Nombre de usuario con permisos administrativos en pfSense
PFSENSE_USER = "admin"

# Contrasena del usuario administrador
PFSENSE_PASSWORD = "1101"

# Puerto SSH (por defecto es 22)
PFSENSE_PORT = 22

# ========================================
# SECCION DE CONFIGURACION - RUTAS DE ARCHIVOS DE ALERTAS DE SNORT
# ========================================
#
# En pfSense, Snort puede monitorear multiples interfaces de red al mismo
# tiempo (LAN, WAN, DMZ, etc.) y guarda las alertas en archivos separados
# para cada interfaz.
#
# Este diccionario mapea el nombre de cada interfaz con la ruta completa
# del archivo donde Snort guarda las alertas de esa interfaz.

ALERT_PATHS = {
    # Interfaz LAN (red local interna): usuarios y dispositivos internos
    "LAN": "/var/log/snort/snort_em112432/alert",

    # Interfaz DMZ (zona desmilitarizada): servidores expuestos parcialmente
    "DMZ": "/var/log/snort/snort_em21556/alert",

    # Interfaz SOC&MOC (centro de operaciones de seguridad y monitoreo)
    "SOC&MOC": "/var/log/snort/snort_em355470/alert",

    # Interfaz WAN (red externa): conexion a Internet
    "WAN": "/var/log/snort/snort_em060741/alert"
}
# ========================================
# SECCION DE CONFIGURACION GENERAL DEL SISTEMA
# ========================================
# Estos parametros controlan el comportamiento del script de auto-bloqueo.
# ========================================
# Ruta completa al archivo de configuracion principal de pfSense
# Este archivo XML contiene toda la configuracion del firewall, incluyendo
# los alias (listas de IP) y las reglas de filtrado
CONFIG_PATH = "/conf/config.xml"
# Nombre del alias en pfSense donde se almacenan las IP bloqueadas
# Un alias es una lista con nombre que agrupa direcciones IP y puede
# ser referenciada en las reglas del firewall para bloquear trafico
ALIAS_NAME = "snort_shun"
# Umbral de alertas: numero minimo de alertas que debe generar una IP
# antes de ser bloqueada automaticamente. Por ejemplo, si es 3,
# una IP sera bloqueada cuando genere su tercera alerta
ALERT_THRESHOLD = 3
```

Nota. Elaboración propia.

Cuadro 92. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 3/25.

```
# Intervalo de revision en segundos: cada cuanto tiempo el script
# revisa los archivos de alertas en busca de nuevas entradas
CHECK_INTERVAL = 10

# Ventana de tiempo en segundos: solo se consideran alertas "recientes"
# que hayan ocurrido dentro de esta ventana. Alertas mas antiguas
# son ignoradas. Esto evita bloquear por alertas muy viejas
ALERT_TIME_WINDOW = 60

# ========================================
# CLASE SNORTALERT - REPRESENTA UNA ALERTA INDIVIDUAL DE SNORT
# ========================================
# Esta clase encapsula la informacion de una alerta de Snort, permitiendo
# parsear (analizar) la linea de texto del archivo de alertas y extraer
# campos importantes como la IP origen, el timestamp, el tipo de ataque, etc.
#
# Cada instancia de SnortAlert representa una sola alerta detectada por Snort.
# ========================================

class SnortAlert:
    """
    Clase que representa una alerta individual de Snort.

    Responsabilidades:
    - Parsear (analizar) una linea del archivo de alertas de Snort
    - Extraer campos clave: timestamp, IP origen, IP destino, protocolo, etc.
    - Validar si la alerta es utilizable (tiene datos completos)
    - Determinar si la alerta es reciente (dentro de la ventana de tiempo)
    """

    def __init__(self, line, interface):
        """
        Constructor de la clase SnortAlert.

        Parametros:
        - line: Linea de texto del archivo de alertas de Snort
        - interface: Nombre de la interfaz de red (LAN, WAN, DMZ, etc.)

        Al crear un objeto SnortAlert, automaticamente se parsea la linea
        para extraer todos los campos relevantes.
        """
        # Guardamos la linea original sin procesar
        self.raw = line

        # Guardamos el nombre de la interfaz de red donde se genero la alerta
        self.interface = interface

        # Parseamos (analizamos) la linea para extraer los campos
        self.parse(line)
```

Nota. Elaboración propia.

Cuadro 93. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 4/25.

```
    def parse(self, line):
        """
        Metodo para parsear (analizar) una linea de alerta de Snort.

        El formato de una linea de alerta de Snort es CSV con la
            siguiente estructura:

        timestamp,campo1,sig_id,campo3,nombre_firma,protocolo,ip_origen,
        puerto_origen,ip_destino,puerto_destino,...

        Ejemplo real:
        10/01/25-11:26:07.205663,1,900103,1,"Local Possible UDP Scan",
            UDP,
        10.0.0.10,9993,172.16.10.101,41811,37488,,0,alert,Allow

        Este metodo extrae y almacena los campos mas importantes en
        atributos del objeto para facilitar su uso posterior.
        """
        try:
            # Dividimos la linea en partes usando la coma como separador
            parts = line.split(',')

            # Verificamos que haya al menos 9 campos (minimo necesario)
            if len(parts) >= 9:
                # Campo 0: Timestamp (fecha y hora de la alerta)
                self.timestamp_str = parts[0].strip()
                # Convertimos el timestamp a un objeto datetime de
                    Python
                self.timestamp = self._parse_timestamp(self.
                    timestamp_str)

                # Campo 2: ID de la firma de Snort (identificador
                    numerico)
                self.signature_id = parts[2]

                # Campo 4: Nombre descriptivo de la firma (tipo de
                    ataque)
                # Removemos las comillas dobles que rodean el nombre
                self.signature_name = parts[4].strip('"')

                # Campo 5: Protocolo de red (TCP, UDP, ICMP, etc.)
                self.protocol = parts[5]

                # Campo 6: Direccion IP de origen (atacante)
                self.source_ip = parts[6]

                # Campo 7: Puerto de origen
                self.source_port = parts[7]

                # Campo 8: Direccion IP de destino (victima)
                self.dest_ip = parts[8]

                # Campo 9: Puerto de destino (si existe)
                self.dest_port = parts[9] if len(parts) > 9 else ""
```

Nota. Elaboración propia.

Cuadro 94. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 5/25.

```
            else:
                # Si no hay suficientes campos, la alerta es invalida
                # Marcamos como None para que se ignore posteriormente
                self.source_ip = None
                self.timestamp = None

        except Exception as e:
            # Si ocurre algun error durante el parseo, mostramos el error
            # y marcamos la alerta como invalida
            print(f"Error al analizar alerta: {e}")
            self.source_ip = None
            self.timestamp = None

    def _parse_timestamp(self, timestamp_str):
        """
        Metodo para convertir el timestamp de Snort a un objeto datetime.

        Snort usa un formato especifico para las fechas y horas:
        MM/DD/YY-HH:MM:SS.microsegundos

        Ejemplo: 10/01/25-11:26:07.205663
        Significa: 1 de octubre de 2025 a las 11:26:07 y 205663
            microsegundos

        Este metodo extrae cada componente (mes, dia, hora, etc.) y
            construye
        un objeto datetime de Python que permite hacer operaciones con
            fechas.

        Parametros:
        - timestamp_str: Cadena con el timestamp en formato Snort

        Retorna:
        - Objeto datetime si el parseo es exitoso
        - None si hay un error (alerta no utilizable)
        """
        try:
            # Separamos la fecha de la hora usando el guion como delimitador
            # Ejemplo: "10/01/25-11:26:07.205663" -> ["10/01/25",
                "11:26:07.205663"]
            date_part, time_part = timestamp_str.split('-')

            # Extraemos mes, dia y anio de la fecha (formato MM/DD/YY)
            # Ejemplo: "10/01/25" -> month=10, day=01, year=25
            month, day, year = date_part.split('/')

            # Extraemos hora, minuto y segundo (ignoramos microsegundos)
            # Ejemplo: "11:26:07.205663" -> "11:26:07" -> hour=11, minute
                =26, second=07
            time_only = time_part.split('.')[0]  # Removemos microsegundos
            hour, minute, second = time_only.split(':')
```

Nota. Elaboración propia.

Cuadro 95. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 6/25.

```
            # Construimos el objeto datetime con los componentes extraidos
            # Nota: asumimos que el anio esta en el rango 2000-2099
            # (25 se convierte en 2025)
            dt = datetime(
                year=2000 + int(year),     # 25 -> 2025
                month=int(month),           # 10 -> octubre
                day=int(day),               # 01
                hour=int(hour),             # 11
                minute=int(minute),         # 26
                second=int(second)          # 07
            )
            return dt

        except Exception:
            # Si ocurre cualquier error durante el parseo (formato
                incorrecto,
            # valores fuera de rango, etc.), retornamos None para marcar
            # que esta alerta no es utilizable
            return None

    def is_valid(self):
        """
        Metodo para verificar si la alerta es valida y utilizable.

        Una alerta es valida si:
        1. Tiene una direccion IP de origen (source_ip no es None)
        2. Tiene un timestamp valido (timestamp no es None)

        Si falta alguno de estos campos, significa que el parseo fallo
        y la alerta debe ser ignorada.

        Retorna:
        - True si la alerta es valida
        - False si la alerta es invalida o incompleta
        """
        return self.source_ip is not None and self.timestamp is not None

    def is_recent(self, max_age_seconds=60):
        """
        Metodo para verificar si la alerta es reciente.

        Una alerta se considera reciente si ocurrio dentro de la ventana
        de tiempo especificada (max_age_seconds).

        Por ejemplo, si max_age_seconds = 60, solo las alertas que
            ocurrieron
        en el ultimo minuto se consideran recientes. Esto evita procesar
        alertas antiguas del archivo de log.

        Parametros:
        - max_age_seconds: Edad maxima en segundos para considerar una
            alerta
                           como reciente (por defecto 60 segundos)
```

Nota. Elaboración propia.

Cuadro 96. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 7/25.

```
Retorna:
        - True si la alerta es reciente (edad <= max_age_seconds)
        - False si la alerta es antigua o no tiene timestamp
        """
        # Si no hay timestamp, la alerta no es utilizable
        if self.timestamp is None:
            return False

        # Obtenemos la fecha y hora actual
        now = datetime.now()

        # Calculamos cuantos segundos han pasado desde que se genero la
            alerta
        age_seconds = (now - self.timestamp).total_seconds()

        # La alerta es reciente si su edad es menor o igual al maximo
            permitido
        return age_seconds <= max_age_seconds

    def __str__(self):
        """
        Metodo especial para obtener una representacion en texto de la
            alerta.

        Este metodo se llama automaticamente cuando usamos print() con un
        objeto SnortAlert, o cuando convertimos el objeto a string.

        Retorna una cadena con el formato:
        [INTERFAZ] timestamp - nombre_firma desde IP_origen

        Ejemplo:
        [LAN] 10/01/25-11:26:07.205663 - Local Possible UDP Scan desde
            10.0.0.10
        """
        return f"[{self.interface}] {self.timestamp_str} - {self.
            signature_name} desde {self.source_ip}"
```

Nota. Elaboración propia.

Cuadro 97. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 8/25.

```
# ========================================
# CLASE PFSENSEAUTOSHUN - CLASE PRINCIPAL DEL SISTEMA DE AUTO-BLOQUEO
# ========================================
# Esta es la clase central que coordina todo el proceso de monitoreo y bloqueo.
#
# Responsabilidades principales:
# 1. Conectarse al servidor pfSense via SSH
# 2. Leer continuamente los archivos de alertas de Snort
# 3. Mantener un contador de alertas por cada IP
# 4. Bloquear automaticamente IP que superen el umbral de alertas
# 5. Sincronizar el estado con el archivo de configuracion de pfSense
# ========================================

class PfSenseAutoShun:
    """
    Clase principal del sistema de auto-bloqueo.

    Esta clase implementa toda la logica de:
    - Conexion SSH a pfSense
    - Lectura incremental de archivos de alertas (solo lineas nuevas)
    - Conteo de alertas por IP origen
    - Bloqueo automatico agregando IP al alias en config.xml
    - Recarga de reglas del firewall para aplicar cambios
    """

    def __init__(self):
        """
        Constructor de la clase PfSenseAutoShun.

        Inicializa las estructuras de datos necesarias para el monitoreo:

        - ssh_client: Cliente SSH para conectarse a pfSense (inicialmente None)
        - alert_counts: Diccionario que mapea cada IP a su lista de alertas
        - shunned_ips: Conjunto (set) de IP que ya han sido bloqueadas
        - last_positions: Diccionario que guarda la ultima linea leida de cada
                          archivo de alertas, para leer solo las lineas nuevas
        """
        # Cliente SSH de paramiko (None hasta que se establezca conexion)
        self.ssh_client = None

        # Diccionario defaultdict: mapea IP -> lista de alertas
        # Ejemplo: {"10.0.0.10": [alerta1, alerta2], "192.168.1.5": [alerta3]}
        # defaultdict(list) crea automaticamente una lista vacia para IP nuevas
        self.alert_counts = defaultdict(list)

        # Conjunto (set) de IP bloqueadas: permite busqueda rapida (O(1))
        # Ejemplo: {"10.0.0.10", "192.168.1.5"}
        self.shunned_ips = set()
```

Nota. Elaboración propia.

Cuadro 98. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 9/25.

```
        # Diccionario que guarda la posicion de lectura en cada archivo
        # Esto permite leer solo las lineas nuevas en cada revision
        # Ejemplo: {"/var/log/snort/snort_em112432/alert": 1523}
        self.last_positions = {}

    def connect(self):
        """
        Metodo para establecer la conexion SSH al servidor pfSense.

        SSH (Secure Shell) es un protocolo de red que permite conectarse
        de forma segura a un servidor remoto y ejecutar comandos en el.

        Este metodo:
        1. Crea un cliente SSH usando la libreria paramiko
        2. Configura la politica de llaves del host (acepta nuevas llaves)
        3. Intenta conectarse a pfSense con las credenciales configuradas
        4. Maneja errores de conexion (timeout, credenciales incorrectas,
            etc.)

        Retorna:
        - True si la conexion fue exitosa
        - False si hubo algun error
        """
        try:
            # Creamos una instancia del cliente SSH de paramiko
            self.ssh_client = paramiko.SSHClient()

            # Configuramos la politica de llaves del host
            # AutoAddPolicy() acepta automaticamente llaves de hosts
                desconocidos
            # (en produccion deberia usarse una politica mas estricta)
            self.ssh_client.set_missing_host_key_policy(paramiko.
                AutoAddPolicy())

            # Mostramos mensaje informativo
            print(f"Conectando a pfSense en {PFSENSE_IP}...")

            # Intentamos establecer la conexion SSH con los parametros
                configurados
            self.ssh_client.connect(
                hostname=PFSENSE_IP,         # Direccion IP del servidor
                port=PFSENSE_PORT,           # Puerto SSH (generalmente 22)
                username=PFSENSE_USER,       # Usuario administrativo
                password=PFSENSE_PASSWORD,   # Contrasena del usuario
                timeout=10                   # Tiempo maximo de espera (10
                    segundos)
            )

            # Si llegamos aqui, la conexion fue exitosa
            print("Conexion exitosa.\n")
            return True
```

Nota. Elaboración propia.

Cuadro 99. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 10/25.

```
        except Exception as e:
            # Si ocurre cualquier error (red, credenciales, timeout, etc.)
            # mostramos el mensaje de error y retornamos False
            print(f"Conexion fallida: {e}")
            return False

    def execute_command(self, command):
        """
        Metodo para ejecutar un comando en el servidor pfSense via SSH.

        Una vez establecida la conexion SSH, podemos ejecutar cualquier comando
        de shell (bash/FreeBSD) en el servidor remoto como si estuvieramos
        conectados directamente a una terminal.

        Este metodo:
        1. Envia el comando al servidor pfSense via SSH
        2. Espera a que el comando termine de ejecutarse
        3. Captura la salida estandar (stdout) y la salida de error (stderr)
        4. Convierte las salidas de bytes a texto UTF-8
        5. Retorna ambas salidas para que el llamador las procese

        Parametros:
        - command: Cadena con el comando a ejecutar (ej: "cat /var/log/alert")

        Retorna:
        - Tupla (output, error) con las salidas del comando
        - (None, mensaje_error) si hay un problema en la ejecucion
        """
        try:
            # Ejecutamos el comando remotamente usando el cliente SSH
            # exec_command retorna 3 objetos: stdin, stdout, stderr
            # - stdin: para enviar entrada al comando (no usado aqui)
            # - stdout: salida estandar del comando
            # - stderr: salida de error del comando
            stdin, stdout, stderr = self.ssh_client.exec_command(command)

            # Leemos la salida estandar y la convertimos de bytes a texto UTF-8
            output = stdout.read().decode('utf-8')

            # Leemos la salida de error y la convertimos de bytes a texto UTF-8
            error = stderr.read().decode('utf-8')

            # Retornamos ambas salidas como una tupla
            return output, error

        except Exception as e:
            # Si ocurre un error durante la ejecucion (conexion perdida, etc.)
            # mostramos el error y retornamos None para output
            print(f"Error ejecutando comando: {e}")
            return None, str(e)
```

Nota. Elaboración propia.

Cuadro 100. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 11/25.

```
    def read_alert_file(self, interface, path):
        """
        Metodo para leer las lineas NUEVAS del archivo de alertas de Snort.

        Este es uno de los metodos mas importantes del sistema. En lugar de
        leer todo el archivo de alertas cada vez (lo cual seria ineficiente)
            ,
        este metodo implementa lectura incremental: solo lee las lineas que
        se han anadido desde la ultima revision.

        Como funciona:
        1. Obtiene el numero total de lineas del archivo actual
        2. Compara con el numero de lineas de la revision anterior
        3. Lee solo las lineas nuevas usando el comando 'tail'
        4. Filtra alertas: solo acepta las que son validas y recientes
        5. Guarda la nueva posicion para la proxima revision

        Parametros:
        - interface: Nombre de la interfaz (LAN, WAN, DMZ, etc.)
        - path: Ruta completa al archivo de alertas en pfSense

        Retorna:
        - Lista de objetos SnortAlert que son validos y recientes
        """
        # Lista que acumulara las alertas nuevas y validas
        alerts = []

        try:
            # Paso 1: Obtener el numero actual de lineas del archivo
            # El comando 'wc -l' cuenta lineas; '2>/dev/null' suprime
                errores
            # '|| echo 0' retorna 0 si el archivo no existe
            size_cmd = f"wc -l < {path} 2>/dev/null || echo 0"
            output, error = self.execute_command(size_cmd)

            if output:
                # Convertimos la salida a numero entero
                current_lines = int(output.strip())

                # Obtenemos la posicion de lectura anterior (0 si es la
                    primera vez)
                last_pos = self.last_positions.get(path, 0)

                # Solo procesamos si hay lineas nuevas
                if current_lines > last_pos:
                    # Calculamos cuantas lineas nuevas hay
                    lines_to_read = current_lines - last_pos

                    # Leemos solo las lineas nuevas con el comando 'tail'
                    # 'tail -n X' muestra las ultimas X lineas del archivo
                    cmd = f"tail -n {lines_to_read} {path} 2>/dev/null"
                    output, error = self.execute_command(cmd)
```

Nota. Elaboración propia.

Cuadro 101. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 12/25.

```
                    if output:
                        # Contadores para estadisticas
                        total_new = 0    # Total de lineas nuevas
                        filtered = 0     # Alertas filtradas (antiguas)

                        # Procesamos cada linea nueva
                        for line in output.strip().split('\n'):
                            # Ignoramos lineas vacias
                            if line.strip():
                                total_new += 1

                                # Creamos un objeto SnortAlert parseando la
                                    linea
                                alert = SnortAlert(line, interface)

                                # Solo agregamos alertas validas Y recientes
                                if alert.is_valid() and alert.is_recent(
                                    ALERT_TIME_WINDOW):
                                    alerts.append(alert)
                                elif alert.is_valid():
                                    # Alerta valida pero fuera de ventana de
                                        tiempo
                                    # (muy antigua, la contamos pero no la
                                        procesamos)
                                    filtered += 1

                        # Si hubo alertas filtradas, mostramos estadistica
                        if filtered > 0:
                            print(f"  [{interface}] {filtered} alerta(s)
                                antigua(s) filtradas (>{ALERT_TIME_WINDOW}s)
                                ")

                    # Actualizamos el puntero de lectura para la proxima
                        revision
                    self.last_positions[path] = current_lines

            except Exception as e:
                # Si ocurre algun error, lo mostramos y continuamos
                print(f"Error leyendo alertas de {interface}: {e}")

        # Retornamos la lista de alertas nuevas y validas
        return alerts
```

Nota. Elaboración propia.

Cuadro 102. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 13/25.

```
    def check_all_alerts(self):
        """
        Metodo para consultar archivos de alertas de TODAS las interfaces.

        Este metodo coordina la lectura de alertas en todas las interfaces
        de red configuradas (LAN, WAN, DMZ, etc.). Para cada interfaz:
        1. Llama a read_alert_file() para obtener alertas nuevas
        2. Acumula todas las alertas en una sola lista

        Esto permite monitorear multiples interfaces simultaneamente y
        detectar ataques provenientes de cualquier segmento de red.

        Retorna:
        - Lista unificada con todas las alertas nuevas de todas las
            interfaces
        """
        # Lista que acumulara alertas de todas las interfaces
        new_alerts = []

        # Iteramos sobre cada par (interfaz, ruta) en el diccionario
            ALERT_PATHS
        for interface, path in ALERT_PATHS.items():
            # Leemos alertas nuevas de esta interfaz
            alerts = self.read_alert_file(interface, path)

            # Agregamos las alertas a la lista unificada
            # extend() agrega todos los elementos de una lista a otra
            new_alerts.extend(alerts)

        # Retornamos todas las alertas de todas las interfaces
        return new_alerts

    def process_alerts(self, alerts):
        """
        Metodo para procesar las alertas nuevas y decidir que IP bloquear.

        Este es el cerebro del sistema de deteccion. Para cada alerta:
        1. Identifica la IP de origen
        2. Acumula la alerta en el historial de esa IP
        3. Verifica si la IP alcanzo el umbral de bloqueo
        4. Muestra estadisticas y resumen

        El sistema mantiene un contador por cada IP y cuando una IP
        alcanza el umbral configurado (por ejemplo, 3 alertas), se marca
        para bloqueo automatico.

        Parametros:
        - alerts: Lista de objetos SnortAlert a procesar

        Retorna:
        - Lista de IP que alcanzaron el umbral y deben ser bloqueadas
        """
```

Nota. Elaboración propia.

Cuadro 103. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 14/25.

```
        # Lista que acumulara las IP que deben bloquearse
        ips_to_shun = []

        # Diccionario para contar alertas NUEVAS por IP en este ciclo
        ip_count_in_batch = defaultdict(int)

        # Procesamos cada alerta individualmente
        for alert in alerts:
            # Extraemos la IP de origen (atacante)
            source_ip = alert.source_ip

            # Incrementamos el contador de alertas nuevas de esta IP
            ip_count_in_batch[source_ip] += 1

            # Si la IP ya esta bloqueada, no necesitamos procesarla mas
            if source_ip in self.shunned_ips:
                print(f"  IP {source_ip} ya bloqueada; se omite.")
                continue

            # Agregamos la alerta al historial de esta IP
            # append() agrega un elemento al final de la lista
            self.alert_counts[source_ip].append(alert)

            # Limitamos el historial a 100 alertas por IP (para no consumir
                mucha memoria)
            # Si hay mas de 100, nos quedamos solo con las 100 mas recientes
            if len(self.alert_counts[source_ip]) > 100:
                self.alert_counts[source_ip] = self.alert_counts[source_ip
                    ][-100:]

            # Obtenemos el contador total de alertas de esta IP
            current_count = len(self.alert_counts[source_ip])

            # Mostramos el progreso hacia el umbral
            print(f"  {source_ip}: {current_count}/{ALERT_THRESHOLD} alertas
                .")

            # Verificamos si la IP alcanzo o supero el umbral de bloqueo
            if current_count >= ALERT_THRESHOLD:
                # Verificamos que no este ya en la lista de bloqueo
                # (doble verificacion por seguridad)
                if source_ip not in self.shunned_ips:
                    # Agregamos la IP a la lista de bloqueo
                    ips_to_shun.append(source_ip)

                    # Mostramos informacion detallada del bloqueo
                    print(f"\nUMBRAL ALCANZADO para {source_ip}")
                    print(f"   Alertas totales: {current_count}")
                    print(f"   Ultimas 3 alertas:")

                    # Mostramos las 3 alertas mas recientes de esta IP
                    # enumerate() da un contador junto con cada elemento
                    for i, a in enumerate(self.alert_counts[source_ip][-3:],
                        1):
                        print(f"     {i}. {a}")
```

Nota. Elaboración propia.

Cuadro 104. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 15/25.

```
        # Mostramos un resumen de todas las IP que generaron alertas en este ciclo
        if ip_count_in_batch:
            print(f"\n  Resumen de alertas en esta revision:")

            # Iteramos sobre cada IP que genero alertas, ordenadas alfabeticamente
            for ip, count in sorted(ip_count_in_batch.items()):
                # Obtenemos el total acumulado de alertas de esta IP
                total = len(self.alert_counts[ip])

                # Determinamos el estado: bloqueada o progreso hacia umbral
                status = "BLOQUEADA" if ip in self.shunned_ips else f"{total}/{ALERT_THRESHOLD}"

                # Mostramos: IP, alertas nuevas en este ciclo, estado total
                print(f"     {ip}: {count} alerta(s) nueva(s), total: {status}")

        # Retornamos la lista de IP que deben bloquearse
        return ips_to_shun

    def get_current_alias_ips(self):
        """
        Metodo para leer el archivo config.xml y obtener las IP bloqueadas.

        pfSense almacena toda su configuracion en un archivo XML llamado
        config.xml. Los alias (listas de IP) se definen en este archivo.

        Este metodo:
        1. Lee el contenido completo de config.xml via SSH
        2. Parsea el XML para navegar por su estructura
        3. Busca el alias con el nombre configurado (ALIAS_NAME)
        4. Extrae y retorna la lista de IP contenidas en ese alias

        Esto es util para sincronizar el estado del script con pfSense,
        especialmente si alguien desbloqueo la IP manualmente desde la
        interfaz web de pfSense.

        Retorna:
        - Lista de direcciones IP actualmente en el alias
        - Lista vacia si no se encuentra el alias o hay un error
        """
        try:
            # Leemos el contenido completo del archivo config.xml
            cmd = f"cat {CONFIG_PATH}"
            output, error = self.execute_command(cmd)

            # Si no hay salida, retornamos lista vacia
            if not output:
                return []

            # Parseamos el contenido XML
            # ET.fromstring() convierte texto XML en un arbol de elementos
            root = ET.fromstring(output)
```

Nota. Elaboración propia.

Cuadro 105. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 16/25.

```
def get_current_alias_ips(self):
        """
        Metodo para leer el archivo config.xml y obtener las IP bloqueadas.

        pfSense almacena toda su configuracion en un archivo XML llamado
        config.xml. Los alias (listas de IP) se definen en este archivo.

        Este metodo:
        1. Lee el contenido completo de config.xml via SSH
        2. Parsea el XML para navegar por su estructura
        3. Busca el alias con el nombre configurado (ALIAS_NAME)
        4. Extrae y retorna la lista de IP contenidas en ese alias

        Esto es util para sincronizar el estado del script con pfSense,
        especialmente si alguien desbloqueo la IP manualmente desde la
        interfaz web de pfSense.

        Retorna:
        - Lista de direcciones IP actualmente en el alias
        - Lista vacia si no se encuentra el alias o hay un error
        """
        try:
            # Leemos el contenido completo del archivo config.xml
            cmd = f"cat {CONFIG_PATH}"
            output, error = self.execute_command(cmd)

            # Si no hay salida, retornamos lista vacia
            if not output:
                return []

            # Parseamos el contenido XML
            # ET.fromstring() convierte texto XML en un arbol de elementos
            root = ET.fromstring(output)

            # Buscamos el alias configurado en la estructura XML
            # findall('.//aliases/alias') busca todos los elementos 'alias'
            # bajo 'aliases' en cualquier nivel del arbol
            for alias in root.findall('.//aliases/alias'):
                # Buscamos el sub-elemento 'name' del alias
                name = alias.find('name')

                # Verificamos si es el alias que buscamos
                if name is not None and name.text == ALIAS_NAME:
                    # Encontramos nuestro alias, buscamos el elemento '
                        address'
                    address = alias.find('address')

                    # Si existe y tiene contenido, extraemos las IP
                    if address is not None and address.text:
                        # pfSense separa multiples IP con espacios
                        # split() divide la cadena en una lista
                        return address.text.split()

            # Si no encontramos el alias, retornamos lista vacia
            return []
```

Nota. Elaboración propia.

Cuadro 106. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 17/25.

```
except Exception as e:
            # Si hay error en el parseo XML o la lectura, mostramos error
            print(f"Error al leer config: {e}")
            return []

    def add_ip_to_alias(self, ip):
        """
        Metodo para agregar una IP al alias en config.xml y aplicar el
            bloqueo.

        Este es el metodo mas critico del sistema. Modifica el archivo de
            configuracion principal de pfSense para agregar una IP al alias
            de bloqueo, luego recarga el firewall para aplicar los cambios.

        Proceso completo:
        1. Lee el archivo config.xml actual
        2. Parsea el XML y busca el alias configurado
        3. Agrega la nueva IP a la lista de direcciones del alias
        4. Guarda una marca temporal en los detalles del alias
        5. Escribe el XML modificado en un archivo temporal
        6. Crea un backup del config.xml original (por seguridad)
        7. Reemplaza config.xml con la version modificada
        8. Recarga las reglas del firewall para aplicar el bloqueo

        Parametros:
        - ip: Direccion IP a bloquear

        Retorna:
        - True si el proceso fue exitoso
        - False si hubo algun error
        """
        try:
            print(f"\nAgregando {ip} al alias {ALIAS_NAME}...")

            # ============================================================
            # PASO 1: Leer el archivo de configuracion actual de pfSense
            # ============================================================
            print(f"   Leyendo config desde {CONFIG_PATH}...")
            cmd = f"cat {CONFIG_PATH}"
            output, error = self.execute_command(cmd)

            # Verificamos que se pudo leer el archivo
            if not output:
                print("   No se pudo leer el archivo de configuracion.")
                return False

            print(f"   Archivo de configuracion leido correctamente ({len(
                output)} bytes).")
```

Nota. Elaboración propia.

Cuadro 107. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 18/25.

```
# ============================================================
            # PASO 2: Parsear (analizar) la estructura XML
            # ============================================================
            print(f"   Analizando XML...")
            # ET.fromstring() convierte el texto XML en un arbol navegable
            root = ET.fromstring(output)
            print(f"   XML analizado con exito.")

            # ============================================================
            # PASO 3: Buscar y modificar el alias objetivo
            # ============================================================
            print(f"   Buscando alias '{ALIAS_NAME}'...")
            alias_found = False

            # Buscamos todos los elementos 'alias' en el XML
            for alias in root.findall('.//aliases/alias'):
                # Buscamos el sub-elemento 'name' de cada alias
                name = alias.find('name')

                # Verificamos si este es el alias que buscamos
                if name is not None and name.text == ALIAS_NAME:
                    alias_found = True
                    print(f"   Alias '{ALIAS_NAME}' encontrado.")

                    # Obtenemos los elementos 'address' y 'detail' del alias
                    address = alias.find('address')
                    detail = alias.find('detail')

                    # Extraemos las IP actuales del alias
                    current_ips = []
                    if address is not None and address.text:
                        # Las IP estan separadas por espacios
                        current_ips = address.text.split()
                    print(f"   IP actuales en el alias: {current_ips}")

                    # Verificamos si la IP ya esta en la lista
                    if ip not in current_ips:
                        print(f"   Agregando {ip} a la lista...")

                        # Agregamos la nueva IP a la lista
                        current_ips.append(ip)

                        # Actualizamos el contenido del elemento 'address'
                        # join() une la lista de IP con espacios
                        address.text = ' '.join(current_ips)

                        # Agregamos una marca temporal en los detalles del
                            alias
                        # Esto ayuda a rastrear cuando se bloqueo cada IP
                        if detail is not None:
                            current_details = []
                            if detail.text:
                                # Los detalles estan separados por ||
                                current_details = detail.text.split('||')
```

Nota. Elaboración propia.

Cuadro 108. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 19/25.

```
                                # Creamos una marca con fecha y hora actual
                                timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                                current_details.append(f"Auto-shun: {timestamp}")

                                # Actualizamos el elemento 'detail'
                                detail.text = '||'.join(current_details)
                        else:
                            print(f"   La IP {ip} ya esta en la lista del alias.")

                        # Salimos del bucle ya que encontramos el alias
                        break

                # Verificamos si encontramos el alias
                if not alias_found:
                    print(f"   No se encontro el alias '{ALIAS_NAME}' en el archivo de configuracion.")
                    return False

                # ============================================================
                # PASO 4: Escribir XML modificado en archivo temporal
                # ============================================================
                print(f"   Convirtiendo XML a cadena...")
                # tostring() convierte el arbol XML de vuelta a texto
                new_config = ET.tostring(root, encoding='unicode')

                # Usamos un archivo temporal para mayor seguridad
                temp_file = "/tmp/config_temp.xml"
                print(f"   Escribiendo archivo temporal {temp_file}...")

                # Escribimos el contenido usando un here-document
                # EOFCONFIG es un delimitador arbitrario
                cmd = f"cat > {temp_file} << 'EOFCONFIG'\n{new_config}\nEOFCONFIG"
                output, error = self.execute_command(cmd)
                if error:
                    print(f"   Error escribiendo archivo temporal: {error}")

                # ============================================================
                # PASO 5: Crear backup del config.xml original
                # ============================================================
                print(f"   Creando respaldo del archivo de configuracion...")
                # $(date +%s) genera un timestamp Unix (segundos desde 1970)
                # Esto crea un backup con nombre unico
                backup_cmd = f"cp {CONFIG_PATH} {CONFIG_PATH}.backup.$(date +%s)"
                output, error = self.execute_command(backup_cmd)
```

Nota. Elaboración propia.

Cuadro 109. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 20/25.

```
            # ============================================================
            # PASO 6: Reemplazar config.xml con la version modificada
            # ============================================================
            print(f"   Reemplazando archivo de configuracion...")
            replace_cmd = f"cp {temp_file} {CONFIG_PATH}"
            output, error = self.execute_command(replace_cmd)
            if error:
                print(f"   Error al reemplazar la configuracion: {error}")

            # ============================================================
            # PASO 7: Recargar reglas del firewall para aplicar cambios
            # ============================================================
            print(f"   Recargando reglas del firewall...")
            # Este script de pfSense recarga las reglas del firewall
            reload_cmd = "/etc/rc.filter_configure"
            output, error = self.execute_command(reload_cmd)

            # Mostramos parte de la salida para verificacion
            if output:
                print(f"   Salida de recarga (primeros 200 caracteres): {
                    output[:200]}")

            # Confirmamos que todo salio bien
            print(f"   IP {ip} agregada correctamente al alias {ALIAS_NAME}.
                ")

            # Agregamos la IP al conjunto local de IP bloqueadas
            self.shunned_ips.add(ip)

            return True

        except Exception as e:
            # Si ocurre cualquier error en el proceso, lo capturamos aqui
            print(f"Error al agregar IP al alias: {e}")
            return False

    def run(self):
        """
        Metodo principal que ejecuta el bucle de monitoreo continuo.

        Este es el corazon del sistema. Una vez iniciado, este metodo:

        1. Muestra informacion inicial del sistema
        2. Sincroniza con el estado actual de pfSense
        3. Entra en un bucle infinito que:
           - Re-sincroniza IP bloqueadas (detecta desbloqueos externos)
           - Lee nuevas alertas de todos los archivos de Snort
           - Procesa alertas y actualiza contadores
           - Bloquea IP que alcancen el umbral
           - Espera un intervalo antes de la siguiente revision
```

Nota. Elaboración propia.

Cuadro 110. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 21/25.

```
        El bucle se ejecuta continuamente hasta que:
        - El usuario presiona Ctrl+C (KeyboardInterrupt)
        - Ocurre un error critico

        Este diseno permite monitoreo en tiempo real 24/7.
        """
        # ================================================================
        # SECCION DE INICIO: Mostrar informacion del sistema
        # ================================================================
        print("=" * 60)
        print("Sistema Auto-Shun iniciado")
        print("=" * 60)

        # Mostramos las interfaces que se estan monitoreando
        print(f"Monitoreando interfaces: {', '.join(ALERT_PATHS.keys())}")

        # Mostramos el umbral de alertas para bloqueo
        print(f"Umbral de alerta: {ALERT_THRESHOLD} alertas")

        # Mostramos cada cuanto se revisan las alertas
        print(f"Intervalo de revision: {CHECK_INTERVAL} segundos")

        # Mostramos la ventana temporal de alertas validas
        print(f"Ventana temporal: solo alertas de los ultimos {
            ALERT_TIME_WINDOW} segundos")

        # Mostramos el nombre del alias configurado
        print(f"Alias configurado: {ALIAS_NAME}")

        # Nota importante sobre la sincronizacion
        print(f"Nota: se sincroniza con la configuracion en cada revision (
            detecta desbloqueos)")
        print("=" * 60)

        # ================================================================
        # SINCRONIZACION INICIAL: Cargar IP ya bloqueadas desde pfSense
        # ================================================================
        # Esto es importante para que el script conozca el estado actual
        # de pfSense, especialmente si se reinicio el script pero pfSense
        # ya tenia IP bloqueadas
        current_ips = self.get_current_alias_ips()

        # Actualizamos nuestro conjunto local de IP bloqueadas
        # update() agrega multiples elementos al conjunto
        self.shunned_ips.update(current_ips)

        # Mostramos el estado inicial
        print(f"\nIP bloqueadas actualmente: {len(self.shunned_ips)}")
        if current_ips:
            for ip in current_ips:
                print(f"  - {ip}")

        print("\nMonitoreo iniciado. Presiona Ctrl+C para detener.\n")
```

Nota. Elaboración propia.

Cuadro 111. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 22/25.

```
        # ================================================================
        # BUCLE PRINCIPAL: Monitoreo continuo
        # ================================================================
        try:
            # Bucle infinito: se ejecuta hasta que se interrumpa
            while True:
                # ------------------------------------------------------
                # FASE 1: Re-sincronizar con pfSense
                # ------------------------------------------------------
                # En cada ciclo, volvemos a leer el alias de pfSense
                # Esto detecta si alguien desbloqueo la IP manualmente
                current_ips = self.get_current_alias_ips()

                # Guardamos una copia del conjunto anterior de IP bloqueadas
                previously_shunned = self.shunned_ips.copy()

                # Actualizamos el conjunto con el estado actual de pfSense
                self.shunned_ips = set(current_ips)

                # Detectamos IP que fueron removidas (desbloqueadas)
                # Operacion de resta de conjuntos: A - B = elementos en A
                    que no estan en B
                unbanned = previously_shunned - self.shunned_ips

                # Si hay IP desbloqueadas, las procesamos
                if unbanned:
                    print(f"\nSe detectaron {len(unbanned)} IP(s) eliminadas
                        del alias (posible desbloqueo):")
                    for ip in unbanned:
                        print(f"   - {ip} eliminada del seguimiento.")

                        # Limpiamos el historial de alertas de IP
                            desbloqueadas
                        # Esto permite que si vuelven a atacar, empiecen
                            desde cero
                        if ip in self.alert_counts:
                            del self.alert_counts[ip]

                # ------------------------------------------------------
                # FASE 2: Leer nuevas alertas de Snort
                # ------------------------------------------------------
                print(f"\n[{datetime.now().strftime('%H:%M:%S')}] Revisando
                    alertas nuevas...")

                # Leemos alertas de todas las interfaces configuradas
                alerts = self.check_all_alerts()
```

Nota. Elaboración propia.

Cuadro 112. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 23/25.

```
                # ------------------------------------------------------
                # FASE 3: Procesar alertas y bloquear si es necesario
                # ------------------------------------------------------
                if alerts:
                    # Informamos cuantas alertas se encontraron
                    print(f"Se encontraron {len(alerts)} alerta(s) recientes
                        (dentro de {ALERT_TIME_WINDOW} s).")

                    # Procesamos las alertas y obtenemos IP que deben
                        bloquearse
                    ips_to_shun = self.process_alerts(alerts)

                    # Intentamos agregar cada IP al alias de bloqueo
                    for ip in ips_to_shun:
                        success = self.add_ip_to_alias(ip)
                        if not success:
                            print(f"   No se pudo agregar {ip} al alias.")
                else:
                    # No hay alertas nuevas en este ciclo
                    print("No hay alertas nuevas en la ventana de tiempo.")

                # ------------------------------------------------------
                # FASE 4: Esperar antes de la siguiente revision
                # ------------------------------------------------------
                print(f"Esperando {CHECK_INTERVAL} segundos para la
                    siguiente revision...")

                # sleep() pausa la ejecucion por los segundos especificados
                time.sleep(CHECK_INTERVAL)

        except KeyboardInterrupt:
            # El usuario presiono Ctrl+C para detener el monitoreo
            print("\nMonitoreo detenido por el usuario.")

        except Exception as e:
            # Capturamos cualquier otro error inesperado
            print(f"\nError en el bucle de monitoreo: {e}")
```

Nota. Elaboración propia.

Cuadro 113. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 24/25.

```
    def close(self):
        """
        Metodo para cerrar la conexion SSH de forma ordenada.

        Es importante cerrar la conexion SSH correctamente cuando el script
        termina. Esto libera recursos del sistema y cierra la sesion de
            forma
        limpia en el servidor pfSense.

        Este metodo se llama automaticamente en el bloque 'finally' de la
        funcion main(), garantizando que la conexion siempre se cierre,
        incluso si ocurre un error.
        """
        # Verificamos si hay una conexion SSH activa
        if self.ssh_client:
            # Cerramos la conexion SSH
            self.ssh_client.close()
            print("Conexion cerrada.")

# ========================================
# FUNCION MAIN - PUNTO DE ENTRADA PRINCIPAL DEL PROGRAMA
# ========================================
# Esta es la funcion principal que se ejecuta cuando se inicia el script.
# Coordina la inicializacion, ejecucion y finalizacion ordenada del sistema.
# ========================================

def main():
    """
    Funcion principal del programa.

    Esta funcion:
    1. Crea una instancia del sistema de auto-bloqueo (PfSenseAutoShun)
    2. Intenta conectarse a pfSense via SSH
    3. Si la conexion es exitosa, inicia el monitoreo continuo
    4. Garantiza que la conexion SSH se cierre correctamente al finalizar

    Retorna:
    - 0 si el programa termino correctamente
    - 1 si no se pudo conectar a pfSense

    El valor de retorno es util para scripts de shell que llamen a este
    programa, permitiendoles saber si hubo exito o fracaso.
    """
    # Creamos una instancia de la clase principal
    shun_system = PfSenseAutoShun()

    # Intentamos conectarnos a pfSense
```

Nota. Elaboración propia.

Cuadro 114. *Script* en Python para bloqueo automático de direcciones IP basado en alertas de Snort parte 25/25.

```
    if shun_system.connect():
        # Si la conexion fue exitosa, iniciamos el monitoreo
        try:
            # Ejecutamos el bucle principal de monitoreo
            shun_system.run()
        finally:
            # El bloque 'finally' se ejecuta SIEMPRE, incluso si hay errores
            # Esto garantiza que la conexion SSH se cierre correctamente
            shun_system.close()
    else:
        # Si no se pudo conectar, mostramos error y retornamos codigo 1
        print("No se pudo conectar a pfSense.")
        return 1

    # Si todo salio bien, retornamos codigo 0 (exito)
    return 0

# ========================================
# BLOQUE DE EJECUCION PRINCIPAL
# ========================================
# Este bloque se ejecuta solo cuando el archivo se ejecuta directamente
# (no cuando se importa como modulo en otro script).
#
# __name__ es una variable especial de Python:
# - Si el archivo se ejecuta directamente, __name__ == "__main__"
# - Si el archivo se importa, __name__ == nombre del modulo
#
# Esto permite que el codigo pueda usarse tanto como programa independiente
# como libreria importable.
# ========================================

if __name__ == "__main__":
    # Ejecutamos la funcion main() y salimos con su codigo de retorno
    # exit() termina el programa con el codigo especificado
    # 0 = exito, cualquier otro numero = error
    exit(main())
```

Nota. Elaboración propia.

8.3.3. Explicación del *script* de desbloqueo automático de direcciones IP

A continuación se presenta el *script* en Python diseñado para desbloquear automáticamente las direcciones IP que han sido bloqueadas por el sistema de auto-bloqueo basado en alertas de Snort. Este *script* monitorea las IP bloqueadas en un alias específico de pfSense y las desbloquea después de que haya transcurrido un período de tiempo definido si no han generado nuevas alertas. En el caso, de que una IP vuelva a generar alertas, el temporizador de bloqueo se reinicia.

Cuadro 115. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 1/19.

```python
# Autor: Andres Lemus
# Mail: lem21634@uvg.edu.gt
# Fecha de creacion: 15/10/2025
# Version: 1.0
# Descripcion: Script de Auto-Desbloqueo para Firewall pfSense basado en tiempo
#!/usr/bin/env python3
# ================================================
# Script de Auto-Desbloqueo para Firewall pfSense
# ================================================
# Este script monitorea las direcciones IP bloqueadas en el alias de pfSense
# y las desbloquea automaticamente despues de que transcurra un periodo de
# tiempo especificado.
#
# Funciona en conjunto con el script auto_shun.py que bloquea IP, mientras
# que este script se encarga de desbloquearlas automaticamente despues de
# un tiempo determinado (por ejemplo, 10 minutos).
#
# El desbloqueo se realiza removiendo las IP del alias en el archivo
# config.xml de pfSense y recargando las reglas del firewall.
# ================================================

# ------------------------------------------------
# SECCION DE IMPORTACION DE LIBRERIAS
# ------------------------------------------------

# paramiko: Libreria para realizar conexiones SSH y ejecutar comandos remotos
# en el servidor pfSense
import paramiko

# time: Para manejar pausas y temporizadores en el ciclo de monitoreo
import time

# xml.etree.ElementTree: Para parsear y modificar archivos XML (config.xml)
# Esta libreria permite leer y modificar la configuracion de pfSense
import xml.etree.ElementTree as ET

# datetime y timedelta: Para trabajar con fechas, horas y calcular
# diferencias de tiempo (duracion del bloqueo)
from datetime import datetime, timedelta

# re: Expresiones regulares para extraer timestamps de los detalles del alias
# Permite buscar patrones especificos en texto (como fechas y horas)
import re
```

Nota. Elaboración propia.

Cuadro 116. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 2/19.

```
# ================================================
# SECCION DE CONFIGURACION - CONEXION A PFSENSE
# ================================================
# Estos valores definen como conectarse al servidor pfSense via SSH.
# Deben coincidir con los valores usados en el script auto_shun.py
# ================================================

# Direccion IP del servidor pfSense en la red
PFSENSE_IP = "172.16.10.1"

# Nombre de usuario con permisos administrativos en pfSense
PFSENSE_USER = "admin"

# Contrasena del usuario administrador (en produccion deberia cifrarse)
PFSENSE_PASSWORD = "1101"

# Puerto SSH (por defecto es 22, puede cambiarse por seguridad)
PFSENSE_PORT = 22

# ================================================
# SECCION DE CONFIGURACION GENERAL DEL SISTEMA
# ================================================
# Estos parametros controlan el comportamiento del script de desbloqueo
# automatico. Permiten ajustar el tiempo de bloqueo y la frecuencia de
# revision del sistema.
# ================================================

# Ruta completa al archivo de configuracion principal de pfSense
# Este archivo XML contiene toda la configuracion del firewall
CONFIG_PATH = "/conf/config.xml"

# Nombre del alias en pfSense donde se almacenan las IP bloqueadas
# Debe ser el mismo alias usado por el script auto_shun.py
ALIAS_NAME = "snort_shun"

# Duracion del bloqueo en horas antes de desbloquear automaticamente
# 0.1667 horas = 10 minutos (calculado como 10/60)
# Puedes ajustar este valor segun tus necesidades:
# - 1.0 = 1 hora
# - 0.5 = 30 minutos
# - 24 = 1 dia
BAN_DURATION_HOURS = 0.1667

# Intervalo de revision en segundos: cada cuanto tiempo el script
# revisa si hay IP que deben desbloquearse
# 300 segundos = 5 minutos
# Es recomendable que sea menor que BAN_DURATION_HOURS para precision
CHECK_INTERVAL = 300
```

Nota. Elaboración propia.

Cuadro 117. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 3/19.

```
# ================================================
# CLASE PFSENSEAUTOUNBAN
# ================================================
# Esta es la clase central que coordina todo el proceso de monitoreo y
# desbloqueo automatico de direcciones IP.
#
# Responsabilidades principales:
# 1. Conectarse al servidor pfSense via SSH
# 2. Leer el alias de IP bloqueadas y sus timestamps de bloqueo
# 3. Calcular cuanto tiempo lleva bloqueada cada IP
# 4. Desbloquear automaticamente las IP que superaron el tiempo limite
# 5. Sincronizar continuamente con el estado actual de pfSense
# ================================================

class PfSenseAutoUnban:
    """
    Clase principal para la funcionalidad de desbloqueo automatico.

    Esta clase implementa toda la logica de:
    - Conexion SSH a pfSense
    - Lectura de IP bloqueadas con sus timestamps
    - Calculo de tiempo de bloqueo transcurrido
    - Desbloqueo automatico de IP expiradas
    - Actualizacion del config.xml de pfSense
    """

    def __init__(self):
        """
        Constructor de la clase PfSenseAutoUnban.

        Inicializa las estructuras de datos necesarias para el monitoreo:

        - ssh_client: Cliente SSH para conectarse a pfSense (inicialmente
          None)
        - ban_timestamps: Diccionario que mapea cada IP bloqueada a su
                          timestamp de bloqueo (cuando fue bloqueada)
                          Este diccionario se carga desde los detalles del
                          alias en config.xml
        """
        # Cliente SSH de paramiko (None hasta que se establezca conexion)
        self.ssh_client = None

        # Diccionario que almacena timestamps de bloqueo por IP
        # Formato: {"192.168.1.100": datetime(2025, 10, 7, 14, 30, 0), ...}
        # Este diccionario se carga desde el campo 'detail' del alias en
          pfSense
        self.ban_timestamps = {}
```

Nota. Elaboración propia.

Cuadro 118. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 4/19.

```
    def connect(self):
        """
        Metodo para establecer la conexion SSH al servidor pfSense.

        SSH (Secure Shell) es un protocolo de red que permite conectarse
        de forma segura a un servidor remoto y ejecutar comandos en el.

        Este metodo:
        1. Crea un cliente SSH usando la libreria paramiko
        2. Configura la politica de llaves del host (acepta nuevas llaves)
        3. Intenta conectarse a pfSense con las credenciales configuradas
        4. Maneja errores de conexion (timeout, credenciales incorrectas,
            etc.)

        Retorna:
        - True si la conexion fue exitosa
        - False si hubo algun error
        """
        try:
            # Creamos una instancia del cliente SSH de paramiko
            self.ssh_client = paramiko.SSHClient()

            # Configuramos la politica de llaves del host
            # AutoAddPolicy() acepta automaticamente llaves de hosts
                desconocidos
            # (en produccion deberia usarse una politica mas estricta)
            self.ssh_client.set_missing_host_key_policy(paramiko.
                AutoAddPolicy())

            # Mostramos mensaje informativo
            print(f"Connecting to pfSense at {PFSENSE_IP}...")

            # Intentamos establecer la conexion SSH con los parametros
                configurados
            self.ssh_client.connect(
                hostname=PFSENSE_IP,         # Direccion IP del servidor
                port=PFSENSE_PORT,           # Puerto SSH (generalmente 22)
                username=PFSENSE_USER,       # Usuario administrativo
                password=PFSENSE_PASSWORD,   # Contrasena del usuario
                timeout=10                   # Tiempo maximo de espera (10
                    segundos)
            )

            # Si llegamos aqui, la conexion fue exitosa
            print("Connected successfully!\n")
            return True

        except Exception as e:
            # Si ocurre cualquier error (red, credenciales, timeout, etc.)
            # mostramos el mensaje de error y retornamos False
            print(f"Connection failed: {e}")
            return False
```

Nota. Elaboración propia.

Cuadro 119. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 5/19.

```
    def execute_command(self, command):
        """
        Metodo para ejecutar un comando en el servidor pfSense via SSH.

        Una vez establecida la conexion SSH, podemos ejecutar cualquier comando
        de shell (bash/FreeBSD) en el servidor remoto como si estuvieramos
        conectados directamente a una terminal.

        Este metodo:
        1. Envia el comando al servidor pfSense via SSH
        2. Espera a que el comando termine de ejecutarse
        3. Captura la salida estandar (stdout) y la salida de error (stderr)
        4. Convierte las salidas de bytes a texto UTF-8
        5. Retorna ambas salidas para que el llamador las procese

        Parametros:
        - command: Cadena con el comando a ejecutar (ej: "cat /conf/config.xml")

        Retorna:
        - Tupla (output, error) con las salidas del comando
        - (None, mensaje_error) si hay un problema en la ejecucion
        """
        try:
            # Ejecutamos el comando remotamente usando el cliente SSH
            # exec_command retorna 3 objetos: stdin, stdout, stderr
            stdin, stdout, stderr = self.ssh_client.exec_command(command)

            # Leemos la salida estandar y la convertimos de bytes a texto UTF-8
            output = stdout.read().decode('utf-8')

            # Leemos la salida de error y la convertimos de bytes a texto UTF-8
            error = stderr.read().decode('utf-8')

            # Retornamos ambas salidas como una tupla
            return output, error

        except Exception as e:
            # Si ocurre un error durante la ejecucion (conexion perdida, etc.)
            # mostramos el error y retornamos None para output
            print(f"Error executing command: {e}")
            return None, str(e)
```

Nota. Elaboración propia.

Cuadro 120. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 6/19.

```
    def get_banned_ips_with_timestamps(self):
        """
        Metodo para obtener las IP bloqueadas y sus timestamps de bloqueo.

        Este metodo es crucial para el sistema de desbloqueo automatico.
        Lee el archivo config.xml de pfSense, extrae el alias de IP
            bloqueadas,
        y obtiene tanto las IP como sus timestamps de bloqueo.

        Los timestamps se almacenan en el campo 'detail' del alias, donde
        el script auto_shun.py guarda la fecha y hora de bloqueo en formato:
        "Auto-shun: YYYY-MM-DD HH:MM:SS"

        Proceso:
        1. Lee el archivo config.xml via SSH
        2. Parsea el XML y busca el alias configurado
        3. Extrae la lista de IP del elemento 'address'
        4. Extrae los detalles del elemento 'detail'
        5. Usa expresiones regulares para extraer timestamps de los detalles
        6. Asocia cada IP con su timestamp correspondiente
        7. Si no hay timestamp, usa la hora actual como fallback

        Retorna:
        - Diccionario {IP: datetime_objeto} con las IP y sus timestamps
        - Diccionario vacio si no hay IP bloqueadas o hay un error
        """
        try:
            # Leemos el contenido completo del archivo config.xml
            cmd = f"cat {CONFIG_PATH}"
            output, error = self.execute_command(cmd)

            # Si no hay salida, retornamos diccionario vacio
            if not output:
                return {}

            # Parseamos el contenido XML
            # ET.fromstring() convierte texto XML en un arbol de elementos
            root = ET.fromstring(output)

            # Buscamos el alias snort_shun en la estructura XML
            # Iteramos sobre todos los alias hasta encontrar el nuestro
            for alias in root.findall('.//aliases/alias'):
                # Buscamos el sub-elemento 'name' del alias
                name = alias.find('name')
```

Nota. Elaboración propia.

Cuadro 121. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 7/19.

```
                # Verificamos si este es el alias que buscamos
                if name is not None and name.text == ALIAS_NAME:
                    # Encontramos el alias correcto, obtenemos sus elementos
                    address = alias.find('address')  # Contiene las IP
                    detail = alias.find('detail')    # Contiene los
                        timestamps

                    # Diccionario para almacenar IP con sus timestamps
                    banned_ips = {}

                    # Verificamos que hay IP en el alias
                    if address is not None and address.text:
                        # Separamos las IP (estan separadas por espacios)
                        ips = address.text.split()

                        # Parseamos los detalles para obtener timestamps
                        details = []
                        if detail is not None and detail.text:
                            # Los detalles estan separados por ||
                            details = detail.text.split('||')

                        # Asociamos cada IP con su detalle correspondiente
                        # enumerate() nos da el indice y el valor
                        for i, ip in enumerate(ips):
                            ban_time = None

                            # Intentamos obtener el timestamp del detalle
                            if i < len(details):
                                detail_text = details[i]

                                # Buscamos el patron "Auto-shun: YYYY-MM-DD
                                    HH:MM:SS"
                                # usando expresiones regulares
                                # \d{4} = 4 digitos (anio)
                                # \d{2} = 2 digitos (mes, dia, hora, minuto,
                                    segundo)
                                # \s+ = uno o mas espacios
                                match = re.search(r'Auto-shun:\s*(\d{4}-\d
                                    {2}-\d{2}\s+\d{2}:\d{2}:\d{2})',
                                    detail_text)
```

Nota. Elaboración propia.

Cuadro 122. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 8/19.

```
                                # Si encontramos el patron, extraemos el timestamp
                                if match:
                                    # group(1) obtiene el primer grupo capturado (la fecha)
                                    timestamp_str = match.group(1)
                                    try:
                                        # Convertimos la cadena a objeto datetime
                                        # strptime() parsea una cadena segun un formato
                                        ban_time = datetime.strptime(timestamp_str, "%Y-%m-%d %H:%M:%S")
                                    except:
                                        # Si falla la conversion, ban_time queda None
                                        pass
                            # Si no encontramos timestamp, usamos hora actual como fallback
                            # Esto puede pasar con IP bloqueadas manualmente o por
                            # versiones antiguas del script
                            if ban_time is None:
                                ban_time = datetime.now()
                                print(f"   No timestamp found for {ip}, using current time")

                            # Agregamos la IP y su timestamp al diccionario
                            banned_ips[ip] = ban_time

                    # Retornamos el diccionario con todas las IP y timestamps
                    return banned_ips

            # Si no encontramos el alias, retornamos diccionario vacio
            return {}

        except Exception as e:
            # Si hay error en el parseo XML o la lectura, mostramos error
            print(f"Error reading config: {e}")
            return {}
```

Nota. Elaboración propia.

Cuadro 123. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 9/19.

```
    def remove_ip_from_alias(self, ip):
        """
        Metodo para remover una IP del alias de bloqueo en config.xml.

        Este metodo es el opuesto al add_ip_to_alias() del script auto_shun.
            py.
        Modifica el archivo de configuracion principal de pfSense para
            remover
        una IP especifica del alias de bloqueo, efectivamente
            desbloqueandola.

        Proceso completo:
        1. Lee el archivo config.xml actual
        2. Parsea el XML y busca el alias configurado
        3. Extrae las IP actuales y sus detalles
        4. Remueve la IP especificada y su detalle correspondiente
        5. Actualiza el XML con las listas modificadas
        6. Escribe el XML modificado en un archivo temporal
        7. Crea un backup del config.xml original (por seguridad)
        8. Reemplaza config.xml con la version modificada
        9. Recarga las reglas del firewall para aplicar el desbloqueo

        Parametros:
        - ip: Direccion IP a desbloquear

        Retorna:
        - True si el proceso fue exitoso
        - False si hubo algun error o la IP no estaba en el alias
        """
        try:
            print(f"\nRemoving {ip} from {ALIAS_NAME} alias...")

            # ================================================
            # PASO 1: Leer el archivo de configuracion
            # ================================================
            print(f"   Reading config from {CONFIG_PATH}...")
            cmd = f"cat {CONFIG_PATH}"
            output, error = self.execute_command(cmd)

            # Verificamos que se pudo leer el archivo
            if not output:
                print("   Failed to read config file")
                return False

            print(f"   Config file read successfully ({len(output)} bytes)")
```

Nota. Elaboración propia.

Cuadro 124. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 10/19.

```
            # ================================================
            # PASO 2: Parsear la estructura XML
            # ================================================
            print(f"   Parsing XML...")
            # ET.fromstring() convierte el texto XML en un arbol navegable
            root = ET.fromstring(output)
            print(f"   XML parsed successfully")

            # ================================================
            # PASO 3: Buscar y modificar el alias
            # ================================================
            print(f"   Looking for alias '{ALIAS_NAME}'...")
            alias_found = False

            # Buscamos todos los elementos 'alias' en el XML
            for alias in root.findall('.//aliases/alias'):
                # Buscamos el sub-elemento 'name' de cada alias
                name = alias.find('name')

                # Verificamos si este es el alias que buscamos
                if name is not None and name.text == ALIAS_NAME:
                    alias_found = True
                    print(f"   Found alias '{ALIAS_NAME}'")

                    # Obtenemos los elementos 'address' y 'detail' del alias
                    address = alias.find('address')
                    detail = alias.find('detail')

                    # Extraemos las IP actuales del alias
                    current_ips = []
                    if address is not None and address.text:
                        # Las IP estan separadas por espacios
                        current_ips = address.text.split()

                    # Extraemos los detalles actuales del alias
                    current_details = []
                    if detail is not None and detail.text:
                        # Los detalles estan separados por ||
                        current_details = detail.text.split('||')

                    print(f"   Current IP in alias: {current_ips}")
```

Nota. Elaboración propia.

Cuadro 125. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 11/19.

```
                    # Verificamos si la IP esta en la lista
                    if ip in current_ips:
                        print(f"   Removing {ip} from the list...")

                        # Obtenemos el indice de la IP en la lista
                        # Esto es importante para remover el detalle correspondiente
                        ip_index = current_ips.index(ip)

                        # Removemos la IP de la lista usando pop()
                        # pop(indice) remueve y retorna el elemento en ese indice
                        current_ips.pop(ip_index)

                        # Tambien removemos la entrada de detalle correspondiente
                        # Esto mantiene la sincronizacion entre IP y detalles
                        if ip_index < len(current_details):
                            current_details.pop(ip_index)

                        # Actualizamos el contenido del elemento 'address'
                        # Si quedan IP, las unimos con espacios; si no, dejamos vacio
                        address.text = ' '.join(current_ips) if current_ips else ''

                        # Actualizamos el contenido del elemento 'detail'
                        if detail is not None:
                            detail.text = '||'.join(current_details) if current_details else ''
                    else:
                        # La IP no esta en la lista (ya fue removida o nunca existio)
                        print(f"   {ip} not found in alias list!")
                        return False

                    # Salimos del bucle ya que encontramos el alias
                    break

            # Verificamos si encontramos el alias
            if not alias_found:
                print(f"   Alias '{ALIAS_NAME}' not found in config")
                return False
```

Nota. Elaboración propia.

Cuadro 126. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 12/19.

```
            # ============================================
            # PASO 4: Escribir XML en archivo temporal
            # ============================================
            print(f"   Converting XML to string...")
            # tostring() convierte el arbol XML de vuelta a texto
            new_config = ET.tostring(root, encoding='unicode')

            # Usamos un archivo temporal para mayor seguridad
            temp_file = "/tmp/config_unban_temp.xml"

            print(f"   Writing to temporary file {temp_file}...")
            # Escribimos el contenido usando un here-document
            # EOFCONFIG es un delimitador arbitrario
            cmd = f"cat > {temp_file} << 'EOFCONFIG'\n{new_config}\
                nEOFCONFIG"
            output, error = self.execute_command(cmd)

            if error:
                print(f"   Error writing temp file: {error}")

            # ============================================
            # PASO 5: Crear backup del config.xml
            # ============================================
            print(f"   Creating backup...")
            # $(date +%s) genera un timestamp Unix (segundos desde 1970)
            # Esto crea un backup con nombre unico
            backup_cmd = f"cp {CONFIG_PATH} {CONFIG_PATH}.backup.$(date +%s)
                "
            output, error = self.execute_command(backup_cmd)

            # ============================================
            # PASO 6: Reemplazar config.xml
            # ============================================
            print(f"   Replacing config file...")
            replace_cmd = f"cp {temp_file} {CONFIG_PATH}"
            output, error = self.execute_command(replace_cmd)

            if error:
                print(f"   Error replacing config: {error}")
```

Nota. Elaboración propia.

Cuadro 127. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 13/19.

```
            # ============================================
            # PASO 7: Recargar reglas del firewall
            # ============================================
            print(f"   Reloading firewall rules...")
            # Este script de pfSense recarga las reglas del firewall
            # Esto hace que el desbloqueo sea efectivo inmediatamente
            reload_cmd = "/etc/rc.filter_configure"
            output, error = self.execute_command(reload_cmd)

            # Mostramos parte de la salida para verificacion
            if output:
                print(f"   Reload output: {output[:200]}")  # Primeros 200
                    caracteres

            # Confirmamos que todo salio bien
            print(f"   Successfully removed {ip} from {ALIAS_NAME}")

            return True

        except Exception as e:
            # Si ocurre cualquier error en el proceso, lo capturamos aqui
            print(f"Error removing IP from alias: {e}")
            return False
```

Nota. Elaboración propia.

Cuadro 128. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 14/19.

```
    def check_expired_bans(self):
        """
        Metodo para revisar bloqueos expirados y removerlos automaticamente.

        Este es el metodo central del sistema de desbloqueo automatico.
        En cada ejecucion:
        1. Obtiene todas las IP bloqueadas con sus timestamps
        2. Calcula cuanto tiempo lleva bloqueada cada IP
        3. Compara con el tiempo maximo de bloqueo configurado
        4. Identifica IP cuyo bloqueo ha expirado
        5. Desbloquea automaticamente las IP expiradas
        6. Muestra informacion detallada del estado de cada IP

        Este metodo se ejecuta periodicamente en el bucle principal,
        permitiendo el desbloqueo automatico sin intervencion manual.
        """
        print(f"\n[{datetime.now().strftime('%H:%M:%S')}] Checking for
            expired bans...")

        # ================================================
        # PASO 1: Obtener IP bloqueadas
        # ================================================
        banned_ips = self.get_banned_ips_with_timestamps()

        # Si no hay IP bloqueadas, no hay nada que hacer
        if not banned_ips:
            print("   No IP currently banned")
            return

        print(f"   Found {len(banned_ips)} banned IP(s)")

        # ================================================
        # PASO 2: Calcular tiempos y determinar IP
        # ================================================

        # Obtenemos la fecha y hora actual
        now = datetime.now()

        # Creamos un timedelta con la duracion maxima de bloqueo
        # timedelta representa una diferencia de tiempo
        expiration_threshold = timedelta(hours=BAN_DURATION_HOURS)

        # Lista que acumulara las IP que deben desbloquearse
        ips_to_unban = []
```

Nota. Elaboración propia.

Cuadro 129. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 15/19.

```
        # Iteramos sobre cada IP bloqueada y su timestamp
        for ip, ban_time in banned_ips.items():
            # Calculamos cuanto tiempo lleva bloqueada la IP
            # Esto es una resta de datetime que resulta en timedelta
            time_banned = now - ban_time

            # Calculamos cuanto tiempo falta para que expire el bloqueo
            time_remaining = expiration_threshold - time_banned

            # Verificamos si el bloqueo ha expirado
            if time_banned >= expiration_threshold:
                # El tiempo bloqueado es mayor o igual al umbral: EXPIRADO
                ips_to_unban.append(ip)
                print(f"   {ip} - Banned for {time_banned}, EXPIRED!")
            else:
                # Todavia esta dentro del periodo de bloqueo
                # Calculamos las horas y minutos restantes para mostrar
                hours_remaining = time_remaining.total_seconds() / 3600
                minutes_remaining = (time_remaining.total_seconds() % 3600)
                    / 60

                # Mostramos el estado con tiempo restante
                print(f"   {ip} - Banned for {time_banned}, expires in {int(
                    hours_remaining)}h {int(minutes_remaining)}m")

        # ================================================
        # PASO 3: Desbloquear IP expiradas
        # ================================================
        if ips_to_unban:
            # Hay IP que deben desbloquearse
            print(f"\n   Unbanning {len(ips_to_unban)} IP(s)...")

            # Procesamos cada IP que debe desbloquearse
            for ip in ips_to_unban:
                # Intentamos remover la IP del alias
                success = self.remove_ip_from_alias(ip)

                # Mostramos el resultado de la operacion
                if success:
                    print(f"   {ip} has been unbanned")
                else:
                    print(f"   Failed to unban {ip}")
        else:
            # No hay IP expiradas en este momento
            print(f"   No expired bans to remove")
```

Nota. Elaboración propia.

Cuadro 130. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 16/19.

```
    def run(self):
        """
        Metodo principal que ejecuta el bucle de monitoreo continuo.

        Este es el corazon del sistema de desbloqueo automatico.
        Una vez iniciado, este metodo:

        1. Muestra informacion inicial del sistema
        2. Obtiene el estado actual de IP bloqueadas
        3. Entra en un bucle infinito que:
           - Revisa bloqueos expirados
           - Desbloquea IP que superaron el tiempo limite
           - Espera un intervalo antes de la siguiente revision

        El bucle se ejecuta continuamente hasta que:
        - El usuario presiona Ctrl+C (KeyboardInterrupt)
        - Ocurre un error critico

        Este diseno permite monitoreo en tiempo real 24/7, desbloqueando
        IP automaticamente sin intervencion manual.
        """
        # ================================================
        # SECCION DE INICIO: Informacion del sistema
        # ================================================
        print("=" * 50)
        print("Auto-Unban System Started")
        print("=" * 50)

        # Mostramos el nombre del alias monitoreado
        print(f"Alias name: {ALIAS_NAME}")

        # Mostramos la duracion del bloqueo antes de desbloquear
        print(f"Ban duration: {BAN_DURATION_HOURS} hour(s)")

        # Mostramos cada cuanto se revisan los bloqueos
        print(f"Check interval: {CHECK_INTERVAL} seconds ({CHECK_INTERVAL //
            60} minutes)")
        print("=" * 50)

        # ================================================
        # REVISION INICIAL: Estado actual de IP
        # ================================================
        # Esto da visibilidad del estado inicial del sistema al iniciarlo
        banned_ips = self.get_banned_ips_with_timestamps()

        print(f"\nCurrently banned IP: {len(banned_ips)}")
        if banned_ips:
            # Iteramos sobre cada IP bloqueada y mostramos cuando fue
                bloqueada
            for ip, ban_time in banned_ips.items():
                # Formateamos el timestamp en formato legible
                print(f"  - {ip} (banned at {ban_time.strftime('%Y-%m-%d %H
                    :%M:%S')})")
```

Nota. Elaboración propia.

Cuadro 131. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 17/19.

```
        print("\nMonitoring started... (Press Ctrl+C to stop)\n")

        # ================================================
        # BUCLE PRINCIPAL: Monitoreo continuo
        # ================================================
        try:
            # Bucle infinito: se ejecuta hasta que se interrumpa
            while True:
                # ----------------------------------------
                # FASE 1: Revisar bloqueos expirados
                # ----------------------------------------
                # Este metodo hace todo el trabajo de:
                # - Obtener IP bloqueadas
                # - Calcular tiempos
                # - Desbloquear IP expiradas
                self.check_expired_bans()

                # ----------------------------------------
                # FASE 2: Esperar siguiente revision
                # ----------------------------------------
                print(f"\n   Waiting {CHECK_INTERVAL} seconds until next
                    check...")

                # sleep() pausa la ejecucion por los segundos especificados
                # Esto evita consumir recursos del sistema revisando
                    constantemente
                time.sleep(CHECK_INTERVAL)

        except KeyboardInterrupt:
            # El usuario presiono Ctrl+C para detener el monitoreo
            print("\n\nMonitoring stopped by user")

        except Exception as e:
            # Capturamos cualquier otro error inesperado
            print(f"\nError in monitoring loop: {e}")

    def close(self):
        """
        Metodo para cerrar la conexion SSH de forma ordenada.

        Es importante cerrar la conexion SSH correctamente cuando el script
        termina. Esto libera recursos del sistema y cierra la sesion de
            forma
        limpia en el servidor pfSense.

        Este metodo se llama automaticamente en el bloque 'finally' de la
        funcion main(), garantizando que la conexion siempre se cierre,
        incluso si ocurre un error.
        """
        # Verificamos si hay una conexion SSH activa
        if self.ssh_client:
            # Cerramos la conexion SSH
            self.ssh_client.close()
            print("Connection closed.")
```

Nota. Elaboración propia.

Cuadro 132. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 18/19.

```python
# ================================================
# FUNCION MAIN - PUNTO DE ENTRADA PRINCIPAL
# ================================================
# Esta es la funcion principal que se ejecuta cuando se inicia el script.
# Coordina la inicializacion, ejecucion y finalizacion ordenada del sistema.
# ================================================

def main():
    """
    Funcion principal del programa.

    Esta funcion:
    1. Crea una instancia del sistema de desbloqueo automatico (PfSenseAutoUnban)
    2. Intenta conectarse a pfSense via SSH
    3. Si la conexion es exitosa, inicia el monitoreo continuo
    4. Garantiza que la conexion SSH se cierre correctamente al finalizar

    Retorna:
    - 0 si el programa termino correctamente
    - 1 si no se pudo conectar a pfSense

    El valor de retorno es util para scripts de shell que llamen a este
    programa, permitiendoles saber si hubo exito o fracaso.
    """
    # Creamos una instancia de la clase principal
    unban_system = PfSenseAutoUnban()

    # Intentamos conectarnos a pfSense
    if unban_system.connect():
        # Si la conexion fue exitosa, iniciamos el monitoreo
        try:
            # Ejecutamos el bucle principal de monitoreo
            unban_system.run()
        finally:
            # El bloque 'finally' se ejecuta SIEMPRE, incluso si hay errores
            # Esto garantiza que la conexion SSH se cierre correctamente
            unban_system.close()
    else:
        # Si no se pudo conectar, mostramos error y retornamos codigo 1
        print("Failed to connect to pfSense")
        return 1

    # Si todo salio bien, retornamos codigo 0 (exito)
    return 0
```

Nota. Elaboración propia.

Cuadro 133. *Script* en Python para desbloqueo automático de direcciones IP basado en tiempo parte 19/19.

```
# ================================================
# BLOQUE DE EJECUCION PRINCIPAL
# ================================================
# Este bloque se ejecuta solo cuando el archivo se ejecuta directamente
# (no cuando se importa como modulo en otro script).
#
# __name__ es una variable especial de Python:
# - Si el archivo se ejecuta directamente, __name__ == "__main__"
# - Si el archivo se importa, __name__ == nombre del modulo
#
# Esto permite que el codigo pueda usarse tanto como programa independiente
# como libreria importable.
# ================================================

if __name__ == "__main__":
    # Ejecutamos la funcion main() y salimos con su codigo de retorno
    # exit() termina el programa con el codigo especificado
    # 0 = exito, cualquier otro numero = error
    exit(main())
```

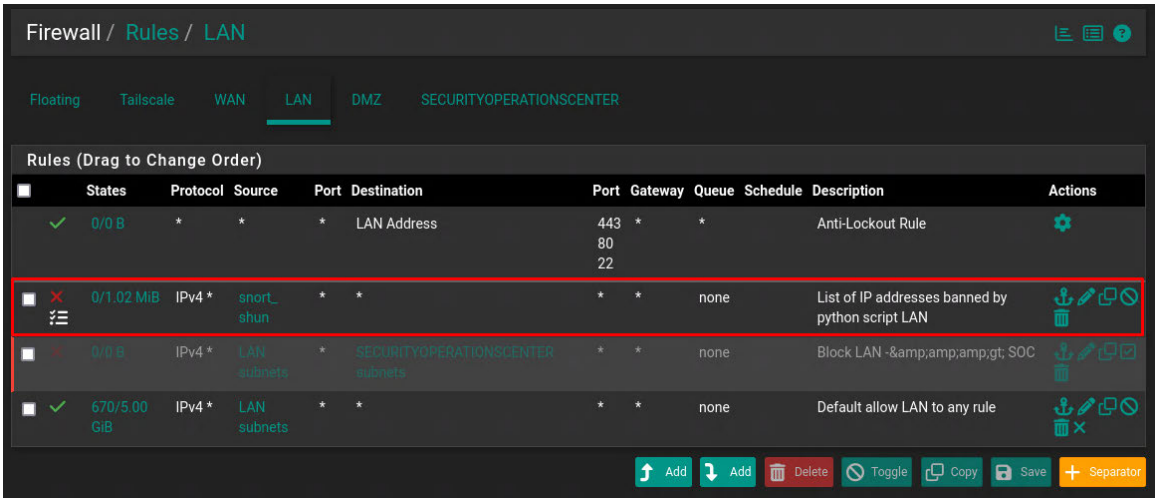
Nota. Elaboración propia.

8.3.4. Creación de las reglas en pfSense para usar los *scripts* de bloqueo y desbloqueo automático de direcciones IP

Para que los *scripts* de bloqueo y desbloqueo automático de direcciones IP funcionen correctamente, es necesario configurar las reglas del *firewall* en pfSense para que utilicen el alias de direcciones IP bloqueadas creado por el *script* de bloqueo automático.

En la interfaz de pfSense, se debe acceder a la sección *Firewall* y luego a *Rules*. Aquí, se deben crear las reglas necesarias para bloquear el tráfico de las IP que están en el alias creado por el *script* de bloqueo automático. Esta regla debe colocarse en la parte superior de las reglas para asegurarse de que se aplique antes que otras reglas.

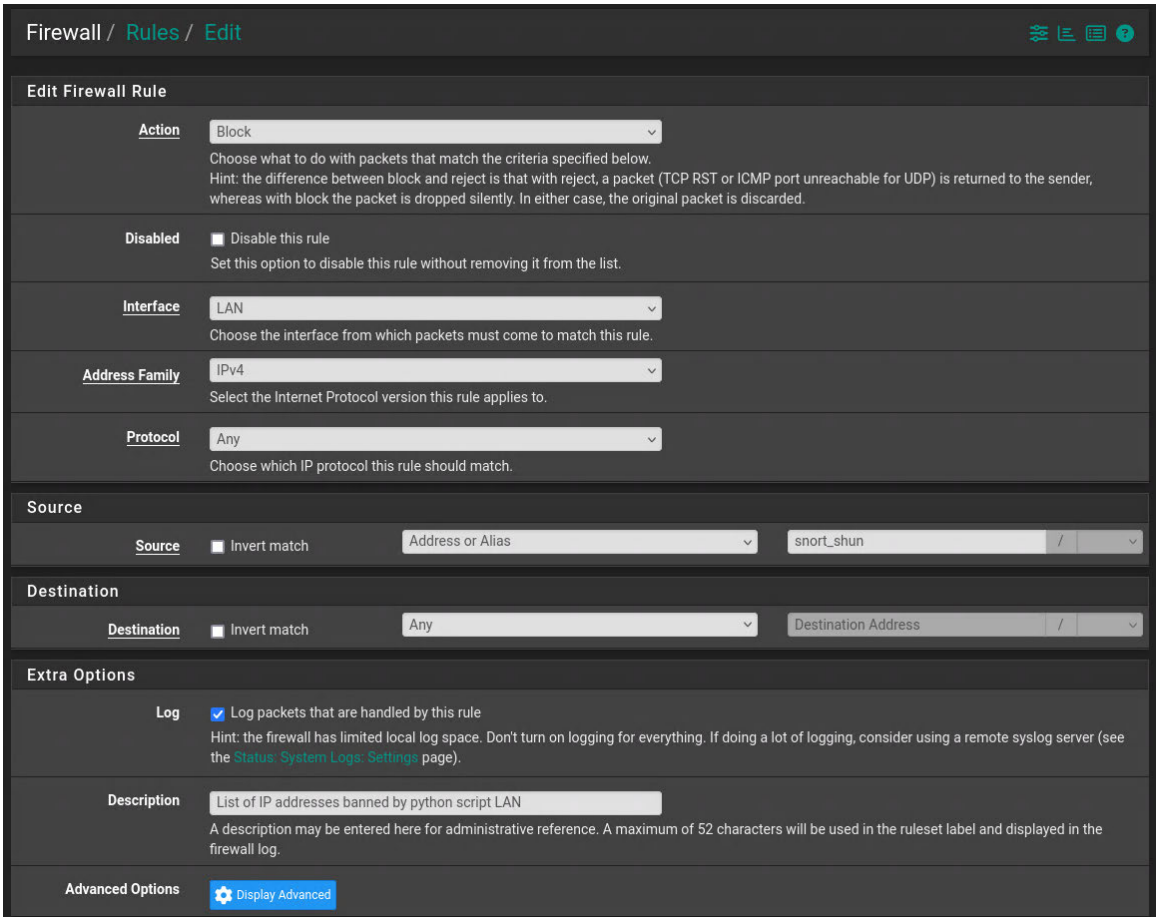
Figura 217. Reglas del *firewall* en pfSense para usar el alias de direcciones IP bloqueadas.



Nota. Reglas del *firewall* en la interfaz web de pfSense. Elaboración propia.

Dentro de la regla en la sección *Action* se debe seleccionar *Block* para bloquear el tráfico, en *Interface* se debe seleccionar la interfaz de red correspondiente, en *Address Family* se debe seleccionar *IPv4*, en *Protocol* se debe seleccionar *Any* para bloquear todo tipo de tráfico. Lo más importante es en la sección *Source* seleccionar *Address or Alias* y en el campo que aparece seleccionar el alias creado por el *script* de bloqueo automático, en este caso, *snort_ shun*.

Figura 218. Configuración de la regla del *firewall* en pfSense para usar el alias de direcciones IP bloqueadas.



Nota. Regla de bloqueo de direcciones IP en la interfaz web de pfSense. Elaboración propia.

Esto se debe realizar en todas las interfaces que se tenga activo Snort y se quiera activar el bloqueo de direcciones IP automáticamente.

8.3.5. Ejecución de los *scripts* de bloqueo y desbloqueo automático de direcciones IP

Para ejecutar los *scripts* de bloqueo y desbloqueo automático de direcciones IP, se deben abrir dos terminales en la máquina donde se encuentran los *scripts*. En cada terminal se debe navegar hasta la carpeta donde se encuentran los *scripts* y ejecutar cada uno de ellos con el siguiente comando:

Cuadro 134. Comando para ejecutar los *scripts* de bloqueo y desbloqueo automático de direcciones IP.

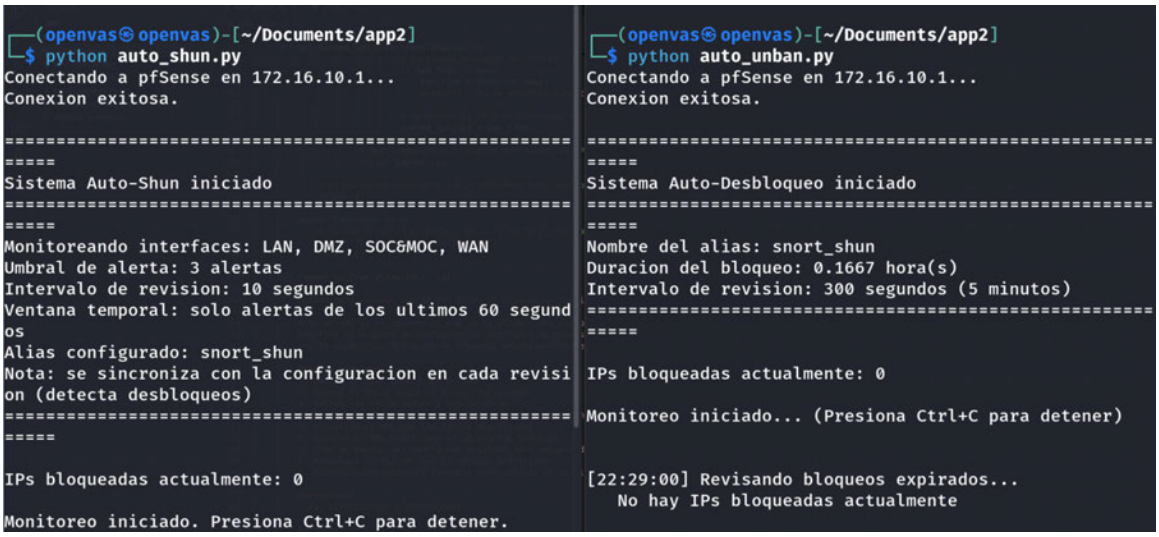
```
# En la primera terminal se ejecuta el script de bloqueo automatico de IP
cd /home/openvas/Documents/app
python auto_shun.py

# En la segunda terminal se ejecuta el script de desbloqueo automatico de IP
cd /home/openvas/Documents/app
python auto_unban.py
```

Nota. Comando para ejecutar los *scripts* de bloqueo y desbloqueo automático de direcciones IP en la terminal. Elaboración propia.

8.3.6. Funcionamiento de los *scripts* para el bloqueo y desbloqueo automático de direcciones IP

Al ejecutar ambos *scripts*, se puede observar en la salida de cada terminal cómo funcionan ambos *scripts*. En la terminal donde se ejecuta el *script* de bloqueo automático de direcciones IP, se puede ver cómo se van bloqueando las IP que generan alertas en Snort y se van agregando al alias de direcciones IP bloqueadas en pfSense. En la terminal donde se ejecuta el *script* de desbloqueo automático de direcciones IP, se puede ver cómo se van revisando las IP bloqueadas y se van desbloqueando aquellas que han superado el tiempo máximo de bloqueo configurado. Solo se muestra la salida del *script* de bloqueo automático de direcciones IP, ya que la salida del *script* de desbloqueo automático de direcciones IP es similar pero con mensajes relacionados al desbloqueo de IP.

Figura 219. Salida del *script* de bloqueo automático de direcciones IP.

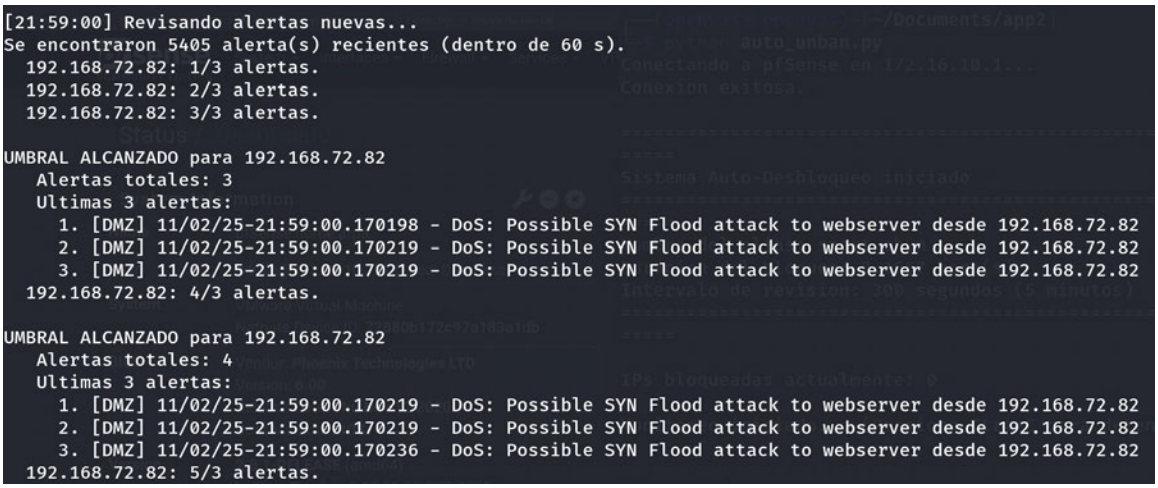


Nota. Salida del *script* de bloqueo automático de direcciones IP en la terminal. Elaboración propia.

En este caso, se configuró que se verificara por alertas nuevas cada 10 segundos, que el tiempo de la ventana de alertas fuera de 1 minuto y que el número de alertas para bloquear una IP fuera de 3. Por lo tanto, cada vez que una IP genera 3 alertas en un minuto, se bloquea automáticamente y se agrega al alias de direcciones IP bloqueadas en pfSense. El tiempo de bloqueo configurado en el *script* de desbloqueo automático de direcciones IP fue de 10 minutos, por lo que cada vez que una IP ha estado bloqueada por más de 10 minutos, se desbloquea automáticamente y se remueve del alias de direcciones IP bloqueadas en pfSense. Esto se verifica cada 5 minutos.

Al momento de realizar la prueba, y generar varias alertas desde una IP atacante, se puede observar en la salida del *script* de bloqueo automático de direcciones IP cómo se avisa que la IP ha sido bloqueada y agregada al alias de direcciones IP bloqueadas en pfSense.

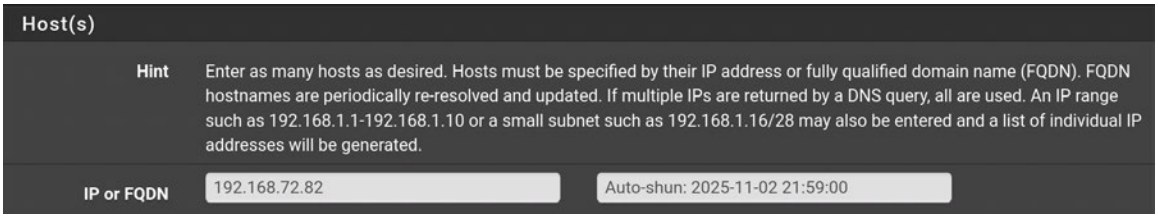
Figura 220. Salida del *script* de desbloqueo automático de direcciones IP.



Nota. Salida del *script* de desbloqueo automático de direcciones IP en la terminal. Elaboración propia.

Al revisar pfSense, se puede ver en el alias de direcciones IP bloqueadas que la IP atacante ha sido agregada correctamente.

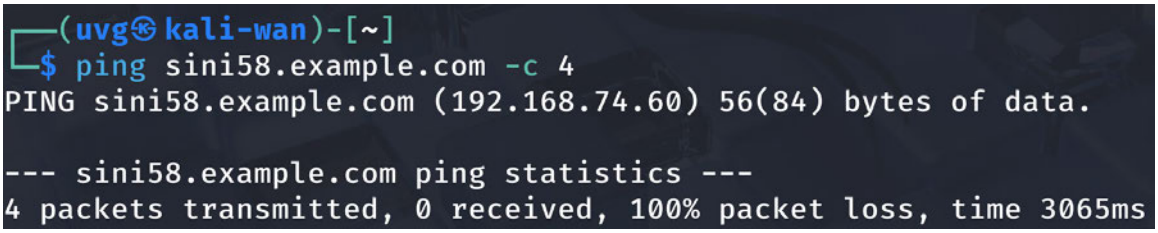
Figura 221. Alias de direcciones IP bloqueadas en pfSense.



Nota. Alias de direcciones IP bloqueadas en la interfaz web de pfSense. Elaboración propia.

Al ya estar la IP en el alias de direcciones IP bloqueadas, al volver a intentar generar algún tipo de tráfico desde esa IP, se puede observar en los registros del *firewall* de pfSense que el tráfico es bloqueado correctamente.

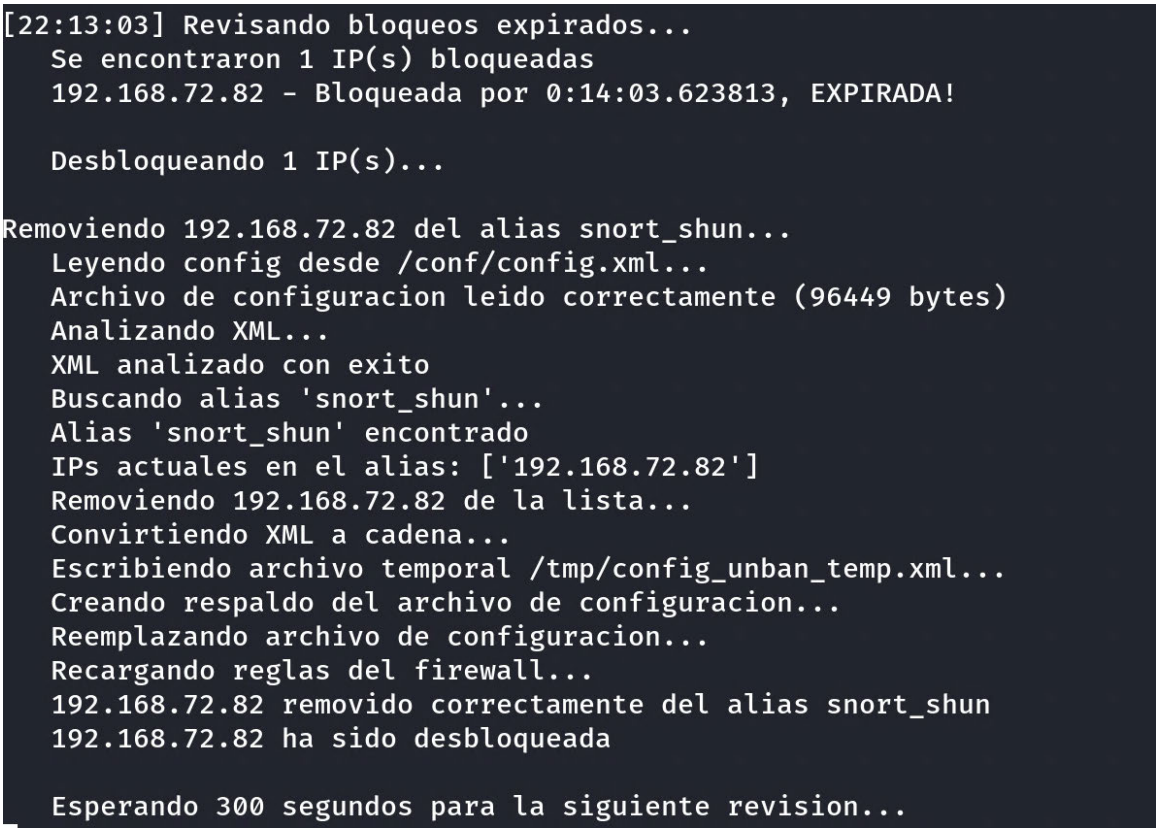
Figura 222. Intento de generación de tráfico desde direcciones IP bloqueada.



Nota. Intento de ping desde la IP bloqueada hacia la interfaz WAN de pfSense.

En la salida del *script* de desbloqueo automático de direcciones IP, se puede observar cómo se revisan las IP bloqueadas y se avisa que la IP ha sido desbloqueada y removida del alias de direcciones IP bloqueadas en pfSense después de haber superado el tiempo máximo de bloqueo configurado.

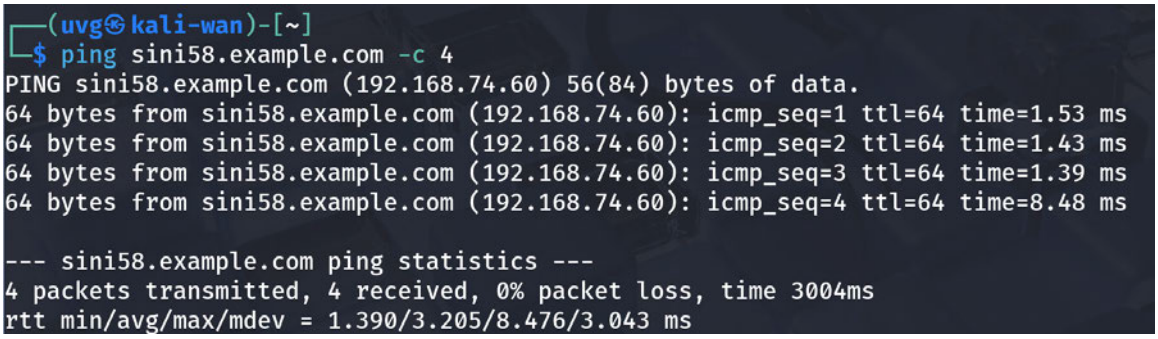
Figura 223. Salida del *script* de desbloqueo automático de direcciones IP mostrando IP desbloqueada.



Nota. Salida del *script* de desbloqueo automático de direcciones IP en la terminal. Elaboración propia.

Al ser removida la IP del alias de direcciones IP bloqueadas en pfSense, al volver a intentar generar tráfico desde esa IP, se puede observar que el tráfico ya no es bloqueado y se permite la conexión correctamente.

Figura 224. Intento de generación de tráfico desde direcciones IP desbloqueada.



Nota. Ping exitoso desde la IP desbloqueada hacia la interfaz WAN de pfSense.

8.3.7. Hacer persistente la ejecución de los *scripts* al iniciar el sistema

Para hacer que ambos *scripts* se ejecuten al iniciar el sistema, se procede a crear un servicio *systemd* para cada uno. Esto permite que los *scripts* se inicien automáticamente en segundo plano cada vez que se reinicie el sistema.

Lo primero es crear un archivo de servicio para el *script* de bloqueo automático de direcciones IP. Se crea un archivo llamado *auto_shun.service* en la misma carpeta que contiene el *script* con el siguiente contenido:

Cuadro 135. Archivo de servicio *systemd* para el *script* de bloqueo automático de direcciones IP.

```
[Unit]
# Autor: Andres Lemus
# Mail: lem21634@uvg.edu.gt
# Fecha de creacion: 15/10/2025
# Version: 1.0
# Descripcion: Servicio systemd para el script de auto-baneo de IP en
    pfSense

Description=Auto-Shun Service for pfSense Firewall
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
User=openvas
WorkingDirectory=/home/openvas/Documents/app
ExecStart=/usr/bin/python3 /home/openvas/Documents/app/auto_shun.py
Restart=always
RestartSec=10
StandardOutput=journal
StandardError=journal

# Security settings
NoNewPrivileges=true
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

Nota. Elaboración propia.

Luego, se crea otro archivo de servicio para el *script* de desbloqueo automático de direcciones IP. Se crea un archivo llamado *auto_ unban.service* en la misma carpeta que contiene el *script* con el siguiente contenido:

Cuadro 136. Archivo de servicio *systemd* para el *script* de desbloqueo automático de direcciones IP.

```
[Unit]
# Autor: Andres Lemus
# Mail: lem21634@uvg.edu.gt
# Fecha de creacion: 15/10/2025
# Version: 1.0
# Descripcion: Servicio systemd para el script de auto-desbaneo de IP en
    pfSense
Description=Auto-Unban Service for pfSense Firewall
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
User=openvas
WorkingDirectory=/home/openvas/Documents/app
ExecStart=/usr/bin/python3 /home/openvas/Documents/app/auto_unban.py
Restart=always
RestartSec=10
StandardOutput=journal
StandardError=journal

# Security settings
NoNewPrivileges=true
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

Nota. Elaboración propia.

El objetivo de estos archivos de servicio es definir cómo y cuándo se deben iniciar los *scripts* de bloqueo y desbloqueo automático de direcciones IP.

Luego se crea un *script* de shell llamado *install_ services.sh* en la misma carpeta que contiene los *scripts* de Python y los archivos de servicio. Este *script* se encarga de copiar los archivos de servicio a la carpeta adecuada, recargar el demonio *systemd*, habilitar los servicios para que se inicien automáticamente al arrancar el sistema y finalmente iniciar ambos servicios. El contenido del *script* es el siguiente:

Cuadro 137. *Script* de shell para instalar y habilitar los servicios *systemd* parte 1/6.

```
# Autor: Andres Lemus
# Mail: lem21634@uvg.edu.gt
# Fecha de creacion: 15/10/2025
# Version: 1.0
# Descripcion: Script de instalacion de servicios systemd para auto-shun y auto-unban
#!/bin/bash
# ============================================================================
# Script de Instalacion de Servicios Auto-Shun/Auto-Unban
# ============================================================================
# Este script automatiza la instalacion y configuracion de los servicios
# systemd para los scripts de auto-bloqueo (auto_shun.py) y auto-desbloqueo
# (auto_unban.py) de IP en pfSense.
#
# Modo de uso: sudo bash install_services.sh
#
# Requisitos:
# - Sistema operativo con systemd (Ubuntu, Debian, CentOS, etc.)
# - Privilegios de root (debe ejecutarse con sudo)
# - Python 3 instalado
# - pip3 disponible para instalar dependencias
# ============================================================================

# ----------------------------------------------------------------------------
# Configuracion de comportamiento del script
# ----------------------------------------------------------------------------
# set -e hace que el script termine inmediatamente si cualquier comando
# falla (retorna un codigo de error diferente de 0).
# Esto previene que errores pasen desapercibidos y causen problemas posteriores.
set -e

# ----------------------------------------------------------------------------
# Mensaje de bienvenida
# ----------------------------------------------------------------------------
# Mostramos un banner inicial para identificar claramente el proceso
echo "========================================="
echo "Auto-Shun/Unban Service Installer"
echo "========================================="
echo ""
```

Nota. Elaboración propia.

Cuadro 138. *Script* de shell para instalar y habilitar los servicios *systemd* parte 2/6.

```
# ------------------------------------------------------
# Verificacion de privilegios de root
# ------------------------------------------------------
# Este script necesita permisos de root porque:
# 1. Copia archivos a /etc/systemd/system/
# 2. Ejecuta comandos systemctl que requieren privilegios
# 3. Instala paquetes de Python con pip3
#
# $EUID es una variable especial que contiene el User ID efectivo
# El root siempre tiene EUID = 0
# -ne significa "not equal" (no igual)
if [ "$EUID" -ne 0 ]; then
    echo "ERROR: Por favor ejecute como root (use sudo)"
    exit 1  # Termina el script con codigo de error 1
fi

# ------------------------------------------------------
# Definicion de rutas y directorios
# ------------------------------------------------------

# Determinamos la ruta absoluta del directorio donde esta este script
# Esto es importante porque el script puede ejecutarse desde cualquier
    ubicacion
# y necesitamos saber donde estan los archivos .service
#
# Explicacion de la construccion:
# - ${BASH_SOURCE[0]} = ruta del script actual
# - dirname = extrae el directorio de una ruta
# - cd + pwd = cambia al directorio y obtiene su ruta absoluta
# - $( ... ) = ejecuta el comando y captura su salida
SCRIPT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

# Directorio donde systemd busca los archivos de servicio
# Este es el directorio estandar en sistemas Linux con systemd
SERVICE_DIR="/etc/systemd/system"

# Mostramos las rutas para que el usuario sepa donde se trabaja
echo "Directorio del script: $SCRIPT_DIR"
echo "Directorio de servicios: $SERVICE_DIR"
echo ""

# ------------------------------------------------------
# Verificacion e instalacion de dependencias de Python
# ------------------------------------------------------
# Los scripts auto_shun.py y auto_unban.py requieren la libreria paramiko
# para realizar conexiones SSH a pfSense. Aqui verificamos si esta instalada
# y la instalamos si es necesario.
```

Nota. Elaboración propia.

Cuadro 139. *Script* de shell para instalar y habilitar los servicios *systemd* parte 3/6.

```
echo "Verificando dependencias de Python..."

# Intentamos importar paramiko en Python
# - python3 -c "..." ejecuta codigo Python directamente desde la linea de comandos
# - 2>/dev/null redirige los errores a /dev/null (los oculta)
# - ! invierte el resultado: si el comando falla, la condicion es verdadera
if ! python3 -c "import paramiko" 2>/dev/null; then
    # Si paramiko NO esta instalado, lo instalamos
    echo "   Instalando paramiko..."
    pip3 install paramiko
else
    # Si paramiko YA esta instalado, informamos
    echo "   OK: paramiko ya esta instalado"
fi
echo ""

# -------------------------------------------------------
# Copia de archivos de servicio
# -------------------------------------------------------
# Los archivos .service definen como systemd debe ejecutar y gestionar
# nuestros scripts Python. Estos archivos deben copiarse al directorio
# de systemd para que el sistema los reconozca.

echo "Instalando archivos de servicio..."

# Copiamos el archivo de servicio auto-shun desde el directorio del script
# al directorio de systemd. Este archivo define el servicio de auto-bloqueo.
cp "$SCRIPT_DIR/auto-shun.service" "$SERVICE_DIR/"

# Copiamos el archivo de servicio auto-unban. Este archivo define el
# servicio de auto-desbloqueo automatico.
cp "$SCRIPT_DIR/auto-unban.service" "$SERVICE_DIR/"

echo "   OK: Archivos de servicio copiados"
echo ""

# -------------------------------------------------------
# Configuracion de permisos de archivos
# -------------------------------------------------------
# Los archivos de servicio deben tener permisos especificos para que
# systemd pueda leerlos pero no puedan ser modificados por usuarios normales.
#
# chmod 644 significa:
# - 6 (propietario/root): lectura (4) + escritura (2) = 6
# - 4 (grupo): solo lectura = 4
# - 4 (otros): solo lectura = 4
#
# Esto es importante por seguridad: solo root puede modificar los servicios.
```

Nota. Elaboración propia.

Cuadro 140. *Script* de shell para instalar y habilitar los servicios *systemd* parte 4/6.

```
echo "Configurando permisos..."

# Establecemos permisos 644 para el servicio auto-shun
chmod 644 "$SERVICE_DIR/auto-shun.service"

# Establecemos permisos 644 para el servicio auto-unban
chmod 644 "$SERVICE_DIR/auto-unban.service"

echo "   OK: Permisos configurados"
echo ""

# ------------------------------------------------------
# Recarga del demonio systemd
# ------------------------------------------------------
# Despues de copiar nuevos archivos de servicio o modificar los existentes,
# es necesario recargar la configuracion de systemd para que reconozca los
# cambios.
#
# daemon-reload no reinicia servicios, solo recarga las definiciones.
# Es equivalente a decirle a systemd: "lee nuevamente los archivos .service"

echo "Recargando demonio systemd..."

# Ejecutamos daemon-reload para que systemd reconozca los nuevos servicios
systemctl daemon-reload

echo "   OK: Demonio recargado"
echo ""

# ------------------------------------------------------
# Habilitacion de servicios para inicio automatico
# ------------------------------------------------------
# "Enable" configura los servicios para que se inicien automaticamente
    cuando
# el sistema operativo arranca. Esto es esencial para que los servicios de
# monitoreo funcionen continuamente, incluso despues de un reinicio del
    servidor.
#
# Cuando un servicio esta "enabled", systemd crea enlaces simbolicos en los
# directorios de arranque apropiados.

echo "Habilitando servicios para inicio automatico..."

# Habilitamos el servicio auto-shun para que inicie al arrancar el sistema
systemctl enable auto-shun.service

# Habilitamos el servicio auto-unban para que inicie al arrancar el sistema
systemctl enable auto-unban.service

echo "   OK: Servicios habilitados para inicio automatico"
echo ""
```

Nota. Elaboración propia.

Cuadro 141. *Script* de shell para instalar y habilitar los servicios *systemd* parte 5/6.

```
# ------------------------------------------------------
# Inicio de servicios
# ------------------------------------------------------
# Una vez que los servicios estan instalados y habilitados, los iniciamos
# inmediatamente. Esto permite que comiencen a monitorear sin necesidad de
# esperar a un reinicio del sistema.
#
# "start" ejecuta los servicios en segundo plano (background) y retorna
# el control al shell inmediatamente.

echo "Iniciando servicios..."

# Iniciamos el servicio auto-shun (monitoreo y bloqueo de IP)
systemctl start auto-shun.service

# Iniciamos el servicio auto-unban (desbloqueo automatico de IP)
systemctl start auto-unban.service

echo "   OK: Servicios iniciados"
echo ""

# ------------------------------------------------------
# Resumen final de instalacion
# ------------------------------------------------------
# Mostramos un mensaje de confirmacion y el estado actual de los servicios
# para verificar que todo esta funcionando correctamente.

echo "========================================="
echo "Instalacion Completada Exitosamente"
echo "========================================="
echo ""

# ------------------------------------------------------
# Verificacion de estado de servicios
# ------------------------------------------------------
# Mostramos el estado de ambos servicios para confirmar que estan activos.
# Opciones de systemctl status:
# --no-pager : muestra la salida directamente sin usar 'less'
# -l : muestra lineas completas sin truncar
# head -n 5 : muestra solo las primeras 5 lineas (informacion esencial)

echo "Estado de Servicios:"

# Mostramos las primeras 5 lineas del estado de auto-shun
systemctl status auto-shun.service --no-pager -l | head -n 5
echo ""

# Mostramos las primeras 5 lineas del estado de auto-unban
systemctl status auto-unban.service --no-pager -l | head -n 5
echo ""
```

Nota. Elaboración propia.

Cuadro 142. *Script* de shell para instalar y habilitar los servicios *systemd* parte 6/6.

```
# ------------------------------------------------------
# Comandos utiles para administracion
# ------------------------------------------------------
# Proporcionamos una guia de referencia rapida con los comandos mas comunes
# para gestionar los servicios. Esto facilita la administracion diaria.

echo "Comandos Utiles:"
echo ""

# Comando para ver logs en tiempo real del servicio auto-shun
# -u especifica la unidad (servicio)
# -f significa 'follow' (seguir, similar a tail -f)
echo "  Ver logs de auto-shun:      sudo journalctl -u auto-shun.service -f"

# Comando para ver logs en tiempo real del servicio auto-unban
echo "  Ver logs de auto-unban:     sudo journalctl -u auto-unban.service -f
    "
echo ""

# Comando para detener ambos servicios inmediatamente
echo "  Detener servicios:          sudo systemctl stop auto-shun auto-unban
    "

# Comando para iniciar ambos servicios
echo "  Iniciar servicios:          sudo systemctl start auto-shun auto-
    unban"

# Comando para reiniciar ambos servicios (detener + iniciar)
echo "  Reiniciar servicios:        sudo systemctl restart auto-shun auto-
    unban"
echo ""

# Comando para verificar el estado actual de ambos servicios
echo "  Verificar estado:           sudo systemctl status auto-shun auto-
    unban"

# Comando para deshabilitar el inicio automatico en el arranque
echo "  Deshabilitar inicio auto:   sudo systemctl disable auto-shun auto-
    unban"
echo ""
```

Nota. Elaboración propia.

De la misma forma se tiene que crear un *script* de shell llamado *uninstall_ services.sh* en la misma carpeta que contiene los *scripts* de Python y los archivos de servicio. Este *script* se encarga de detener ambos servicios, deshabilitarlos para que no se inicien automáticamente al arrancar el sistema, eliminar los archivos de servicio del directorio adecuado y recargar el demonio *systemd*. El contenido del *script* es el siguiente:

Cuadro 143. *Script* de shell para desinstalar y eliminar los servicios *systemd* parte 1/4.

```
# Autor: Andres Lemus
# Mail: lem21634@uvg.edu.gt
# Fecha de creacion: 15/10/2025
# Version: 1.0
# Descripcion: Script de desinstalacion de servicios systemd para auto-shun y auto-unban
#!/bin/bash
# ============================================================
# Script de Desinstalacion de Servicios Auto-Shun/Auto-Unban
# ============================================================
# Este script automatiza la desinstalacion completa de los servicios systemd
# para los scripts de auto-bloqueo (auto_shun.py) y auto-desbloqueo
# (auto_unban.py) de IP en pfSense.
#
# Modo de uso: sudo bash uninstall_services.sh
#
# Lo que hace este script:
# 1. Detiene los servicios en ejecucion
# 2. Deshabilita el inicio automatico de los servicios
# 3. Elimina los archivos .service del sistema
# 4. Recarga la configuracion de systemd
#
# Nota: Los scripts Python (.py) NO se eliminan, solo los servicios systemd.
#
# Requisitos:
# - Sistema operativo con systemd
# - Privilegios de root (debe ejecutarse con sudo)
# ============================================================

# ------------------------------------------------------------
# Configuracion de comportamiento del script
# ------------------------------------------------------------
# set -e hace que el script termine inmediatamente si cualquier comando
# falla (retorna un codigo de error diferente de 0).
# Sin embargo, usamos "|| true" en comandos que pueden fallar legitimamente
# (por ejemplo, si un servicio ya esta detenido).
set -e

# ------------------------------------------------------------
# Mensaje de bienvenida
# ------------------------------------------------------------
# Mostramos un banner inicial para identificar claramente el proceso
echo "============================================"
echo "Auto-Shun/Unban Service Uninstaller"
echo "============================================"
echo ""
```

Nota. Elaboración propia.

Cuadro 144. *Script* de shell para desinstalar y eliminar los servicios *systemd* parte 2/4.

```
# ------------------------------------------------------
# Verificacion de privilegios de root
# ------------------------------------------------------
# Este script necesita permisos de root porque:
# 1. Modifica archivos en /etc/systemd/system/
# 2. Ejecuta comandos systemctl que requieren privilegios
# 3. Detiene servicios del sistema
#
# $EUID es una variable especial que contiene el User ID efectivo
# El root siempre tiene EUID = 0
# -ne significa "not equal" (no igual)
if [ "$EUID" -ne 0 ]; then
    echo "ERROR: Por favor ejecute como root (use sudo)"
    exit 1  # Termina el script con codigo de error 1
fi

# ------------------------------------------------------
# Definicion de rutas
# ------------------------------------------------------
# Directorio donde systemd almacena los archivos de servicio
# Este es el directorio estandar en sistemas Linux con systemd
SERVICE_DIR="/etc/systemd/system"

# ------------------------------------------------------
# Detencion de servicios
# ------------------------------------------------------
# Primero debemos detener los servicios que estan en ejecucion antes de
# poder deshabilitarlos o eliminar sus archivos de configuracion.
#
# Explicacion de las opciones:
# - systemctl stop: detiene el servicio especificado
# - 2>/dev/null: redirige mensajes de error a /dev/null (los oculta)
# - || true: si el comando falla, retorna true (exito) de todas formas
#
# Usamos "|| true" porque el servicio podria no estar corriendo o no existir
    ,
# y no queremos que el script falle en esos casos.

echo "Deteniendo servicios..."

# Detenemos el servicio auto-shun (monitoreo y bloqueo de IP)
# Si el servicio no esta corriendo o no existe, el error se ignora
systemctl stop auto-shun.service 2>/dev/null || true

# Detenemos el servicio auto-unban (desbloqueo automatico de IP)
# Si el servicio no esta corriendo o no existe, el error se ignora
systemctl stop auto-unban.service 2>/dev/null || true

echo "   OK: Servicios detenidos"
echo ""
```

Nota. Elaboración propia.

Cuadro 145. *Script* de shell para desinstalar y eliminar los servicios *systemd* parte 3/4.

```
# ------------------------------------------------------
# Deshabilitacion de servicios
# ------------------------------------------------------
# "Disable" elimina la configuracion de inicio automatico de los servicios.
# Esto significa que los servicios ya no se iniciaran automaticamente cuando
# el sistema arranque.
#
# Cuando un servicio esta "enabled", systemd crea enlaces simbolicos en los
# directorios de arranque. Al hacer "disable", estos enlaces se eliminan.
#
# Explicacion de opciones:
# - systemctl disable: deshabilita el inicio automatico del servicio
# - 2>/dev/null: oculta mensajes de error
# - || true: continua aunque el servicio no este habilitado

echo "Deshabilitando servicios..."

# Deshabilitamos auto-shun para que no inicie automaticamente
# Si el servicio no esta habilitado o no existe, el error se ignora
systemctl disable auto-shun.service 2>/dev/null || true

# Deshabilitamos auto-unban para que no inicie automaticamente
# Si el servicio no esta habilitado o no existe, el error se ignora
systemctl disable auto-unban.service 2>/dev/null || true

echo "   OK: Servicios deshabilitados"
echo ""

# ------------------------------------------------------
# Eliminacion de archivos de servicio
# ------------------------------------------------------
# Una vez que los servicios estan detenidos y deshabilitados, podemos
# eliminar de forma segura sus archivos de definicion (.service) del sistema
#
# Esto completa la desinstalacion del componente systemd, aunque los scripts
# Python originales permanecen intactos en su directorio.
#
# Explicacion del comando rm:
# - rm: comando para eliminar archivos
# - -f: fuerza la eliminacion sin pedir confirmacion, no muestra error
#       si el archivo no existe

echo "Eliminando archivos de servicio..."

# Eliminamos el archivo de definicion del servicio auto-shun
# -f asegura que no haya error si el archivo ya fue eliminado
rm -f "$SERVICE_DIR/auto-shun.service"
```

Nota. Elaboración propia.

Cuadro 146. *Script* de shell para desinstalar y eliminar los servicios *systemd* parte 4/4.

```
# Eliminamos el archivo de definicion del servicio auto-unban
# -f asegura que no haya error si el archivo ya fue eliminado
rm -f "$SERVICE_DIR/auto-unban.service"

echo "   OK: Archivos de servicio eliminados"
echo ""

# ------------------------------------------------------
# Recarga del demonio systemd y limpieza
# ------------------------------------------------------
# Despues de eliminar archivos de servicio, debemos recargar la
    configuracion
# de systemd para que deje de tener referencias a los servicios eliminados.
#
# Tambien limpiamos cualquier estado de "fallo" que pudieran tener los
# servicios para mantener limpio el registro de systemd.

echo "Recargando demonio systemd..."

# Recargamos la configuracion de systemd para que reconozca los cambios
# Esto hace que systemd "olvide" los servicios que acabamos de eliminar
systemctl daemon-reload

# Limpiamos el estado de fallo de todos los servicios
# reset-failed elimina el registro de servicios que fallaron
# 2>/dev/null: oculta errores si no hay servicios fallidos
# || true: continua aunque no haya servicios para resetear
systemctl reset-failed 2>/dev/null || true

echo "   OK: Demonio recargado"
echo ""

# ------------------------------------------------------
# Mensaje final de confirmacion
# ------------------------------------------------------
# Informamos al usuario que la desinstalacion fue exitosa y le recordamos
# que los scripts Python originales no fueron eliminados.

echo "=========================================="
echo "Desinstalacion Completada Exitosamente"
echo "=========================================="
echo ""
echo "Los servicios han sido removidos del sistema."
echo ""
echo "NOTA IMPORTANTE:"
echo "Los scripts Python NO fueron eliminados y permanecen en:"
echo "  /home/openvas/Documents/app2/"
echo ""
echo "Si desea eliminar completamente el sistema, borre manualmente"
echo "ese directorio con: sudo rm -rf /home/openvas/Documents/app2/"
echo ""
```

Nota. Elaboración propia.

Para instalar los servicios, se debe ejecutar el *script* de instalación con privilegios de superusuario utilizando el siguiente comando en la terminal de Kali Linux:

Cuadro 147. Comando para instalar los servicios *systemd*.

```
sudo bash /home/openvas/Documents/app/install_services.sh
```

Nota. Comando para ejecutar el *script* de instalación de servicios *systemd* con privilegios de superusuario. Elaboración propia.

Lo que hace este *script* instalar las dependencias de Python necesarias, configurar ambos servicios para que se inicien automáticamente al arrancar el sistema, iniciar ambos servicios inmediatamente y mostrar un resumen del estado de los servicios.

Figura 225. Salida del *script* de instalación de servicios *systemd*.

```
● auto-shun.service - Auto-Shun Service for pfSense Firewall
     Loaded: loaded (/etc/systemd/system/auto-shun.service; enabled; preset: disabled)
     Active: active (running) since Sun 2025-11-02 22:47:14 CST; 34ms ago
 Invocation: 63a0f8f480d5463bb34e7159300f6057
   Main PID: 1897266 (python3)

● auto-unban.service - Auto-Unban Service for pfSense Firewall
     Loaded: loaded (/etc/systemd/system/auto-unban.service; enabled; preset: disabled)
     Active: active (running) since Sun 2025-11-02 22:47:14 CST; 21ms ago
 Invocation: 0be47cfcfed64507a6554a3698e7f47f
   Main PID: 1897269 (python3)

Comandos Utiles:

  Ver logs de auto-shun:      sudo journalctl -u auto-shun.service -f
  Ver logs de auto-unban:     sudo journalctl -u auto-unban.service -f

  Detener servicios:          sudo systemctl stop auto-shun auto-unban
  Iniciar servicios:          sudo systemctl start auto-shun auto-unban
  Reiniciar servicios:        sudo systemctl restart auto-shun auto-unban

  Verificar estado:           sudo systemctl status auto-shun auto-unban
  Deshabilitar inicio auto:   sudo systemctl disable auto-shun auto-unban
```

Nota. Salida del *script* de instalación de servicios en la terminal. Elaboración propia.

8.3.8. Comprobación de la creación del *systemd* para los *scripts*

Lo primero es verificar que el servicio de bloqueo automático de direcciones IP se esté ejecutando luego de reiniciar el sistema. Por lo que se reinicia Kali Linux y luego de iniciar sesión se ejecuta el siguiente comando en la terminal para verificar el estado del servicio:

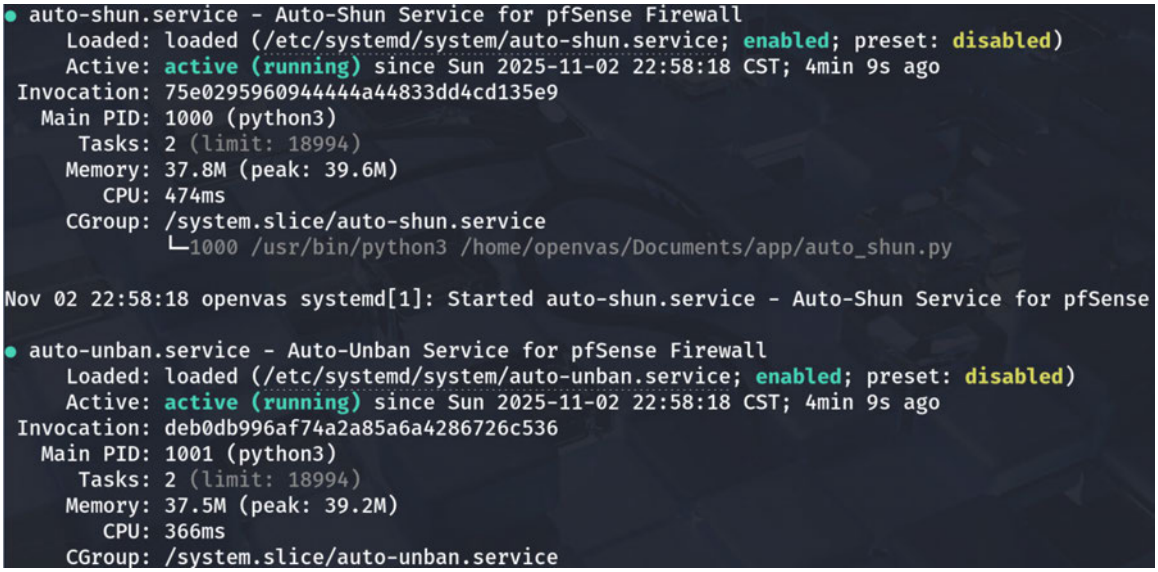
Cuadro 148. Comando para verificar el estado del servicio de bloqueo automático de direcciones IP.

```
sudo systemctl status auto-shun auto-unban
```

Nota. Comando para verificar el estado de ambos servicios *systemd* en la terminal. Elaboración propia.

En la salida del comando se puede observar que ambos servicios están activos y en ejecución, lo que indica que se han configurado correctamente para iniciarse automáticamente al arrancar el sistema.

Figura 226. Estado de los servicios de bloqueo y desbloqueo automático de direcciones IP.

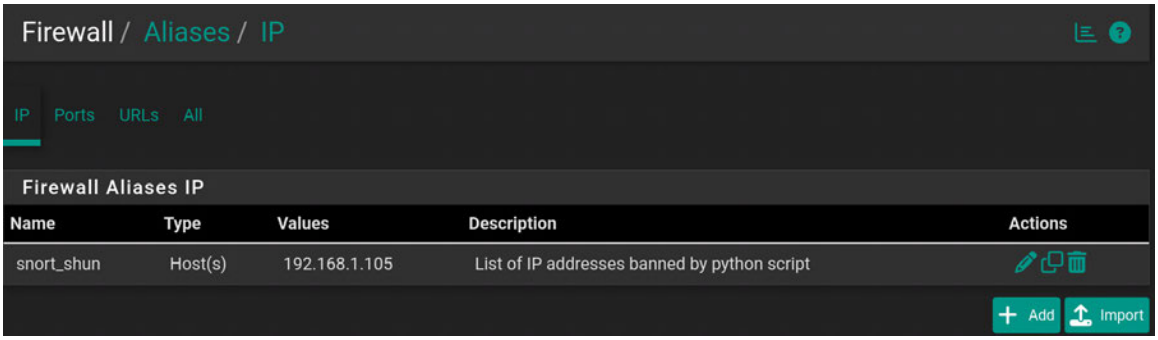


Nota. Salida del comando de verificación del estado de los servicios en la terminal. Elaboración propia.

Al confirmar que ambos servicios están activos, se puede proceder a probar su funcionamiento generando tráfico malicioso desde una IP externa para verificar que el servicio de bloqueo automático de direcciones IP funcione correctamente y bloquee la IP en pfSense. Luego de esperar el tiempo configurado para el desbloqueo automático de direcciones IP, se puede verificar que la IP haya sido removida del alias de direcciones IP bloqueadas en pfSense, confirmando así que el servicio de desbloqueo automático de direcciones IP también funciona correctamente.

Esta vez se realiza un ataque interno desde la red LAN. Al generar tráfico malicioso desde una IP interna, se puede observar la alias de direcciones IP bloqueadas en pfSense y verificar que la IP interna haya sido bloqueada correctamente.

Figura 227. Verificación del bloqueo de una IP interna en pfSense.



Nota. Confirmación visual en pfSense de que la IP interna ha sido bloqueada correctamente.

8.4. Escaneo de vulnerabilidades con OpenVAS

8.4.1. Actualización de bases de datos de vulnerabilidades

Antes de realizar un escaneo de vulnerabilidades, es importante asegurarse de que las bases de datos de vulnerabilidades estén actualizadas. Para ello, se debe ir a la sección *Administration* en el menú principal y seleccionar la opción *Feed Status*. En esta sección se pueden ver las diferentes bases de datos de vulnerabilidades que utiliza OpenVAS, como NVTs, SCAP y CERT. En el apartado de *Status* se puede ver la fecha de la última actualización de cada base de datos. Para actualizar las bases de datos, se deben ejecutar los siguientes comandos en la terminal de Kali Linux:

Cuadro 149. Actualización de bases de datos de vulnerabilidades en OpenVAS.

```
sudo greenbone-nvt-sync
sudo greenbone-scapdata-sync
sudo greenbone-certdata-sync
sudo greenbone-feed-sync
```

Nota. Comandos para actualizar las bases de datos de vulnerabilidades en OpenVAS. Elaboración propia.

Estos comandos se deben ejecutar uno por uno y pueden tardar varios minutos en completarse. Una vez que se han actualizado las bases de datos, se debe verificar que la actualización se haya realizado correctamente. Para ello, se debe volver a la sección *Feed Status* en la interfaz web de OpenVAS y verificar que la fecha de la última actualización de cada base de datos sea reciente.

Figura 228. Estado de las bases de datos de vulnerabilidades en OpenVAS.

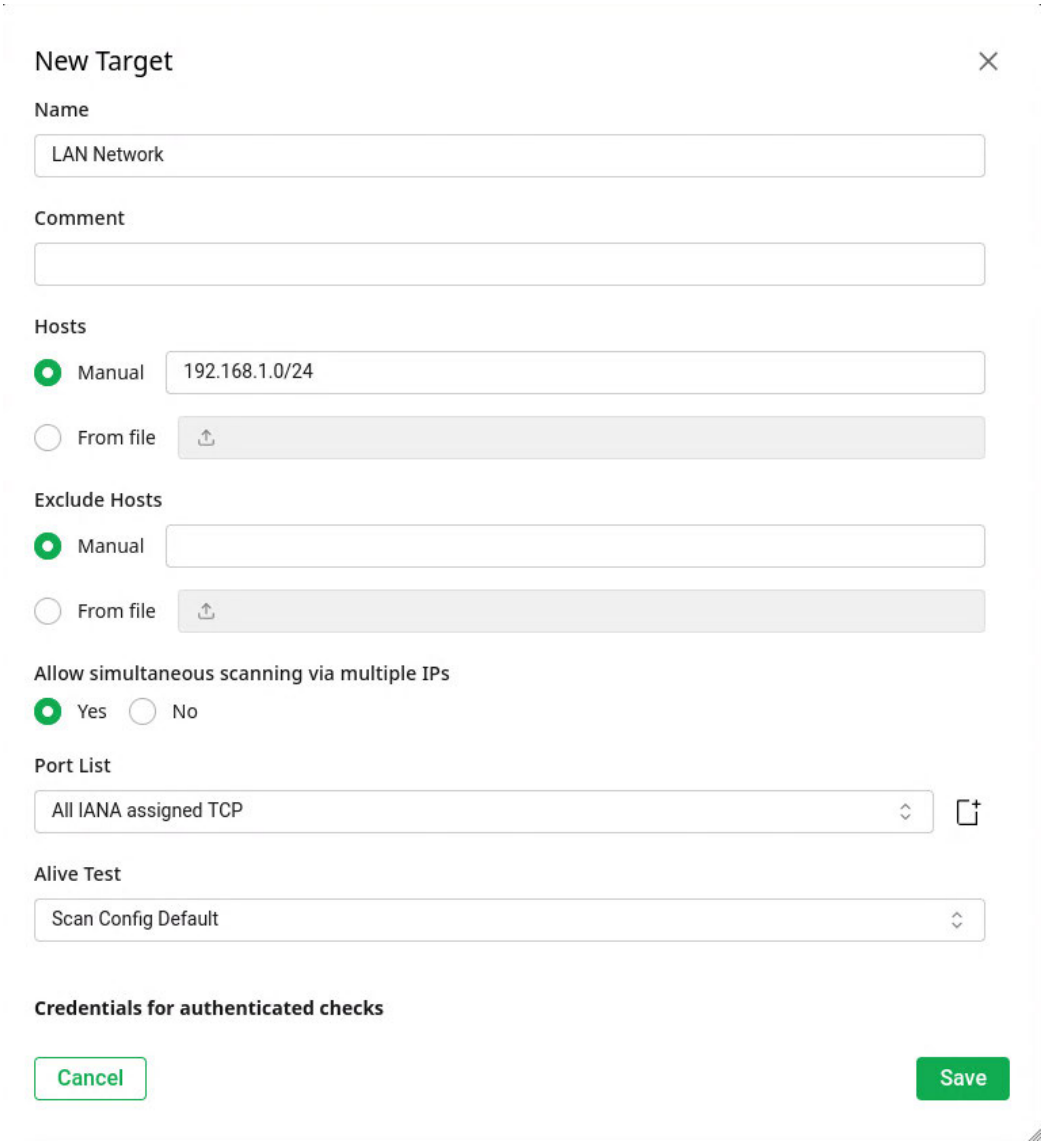
Nota. Confirmación visual en interfaz web de que las firmas NVT, SCAP y CERT están actualizadas. Elaboración propia.

8.4.2. Creación de un escaneo de vulnerabilidades

Para crear un escaneo de vulnerabilidades, se debe ir a la sección *Scans* en el menú principal y seleccionar la opción *Tasks*. En esta sección se pueden ver las diferentes tareas de escaneo que se han creado. Para crear una nueva tarea de escaneo, se debe hacer clic en el botón de *New Task*. En la ventana que se abre, se debe proporcionar un nombre para la tarea de escaneo y seleccionar el objetivo del escaneo. El objetivo puede ser una dirección IP, un rango de direcciones IP o un nombre de dominio. En este caso, se realiza un escaneo de tipo *Discovery* para identificar los dispositivos conectados a la red LAN.

Por ello en el nombre de la tarea se escribe *Discovery LAN Network*. En el campo *Target* se debe seleccionar la opción *New Target* para crear un nuevo objetivo. En la ventana que se abre, se debe proporcionar un nombre para el objetivo y la dirección IP o el rango de direcciones IP que se desea escanear. En este caso, se escanea la red LAN, que tiene el rango de direcciones IP 192.168.1.0/24. Por ello, en el campo *Name* se pone *LAN Network* y en el campo *Hosts* se pone 192.168.1.0/24. Se guarda la configuración del objetivo y se vuelve a la ventana de creación de la tarea de escaneo.

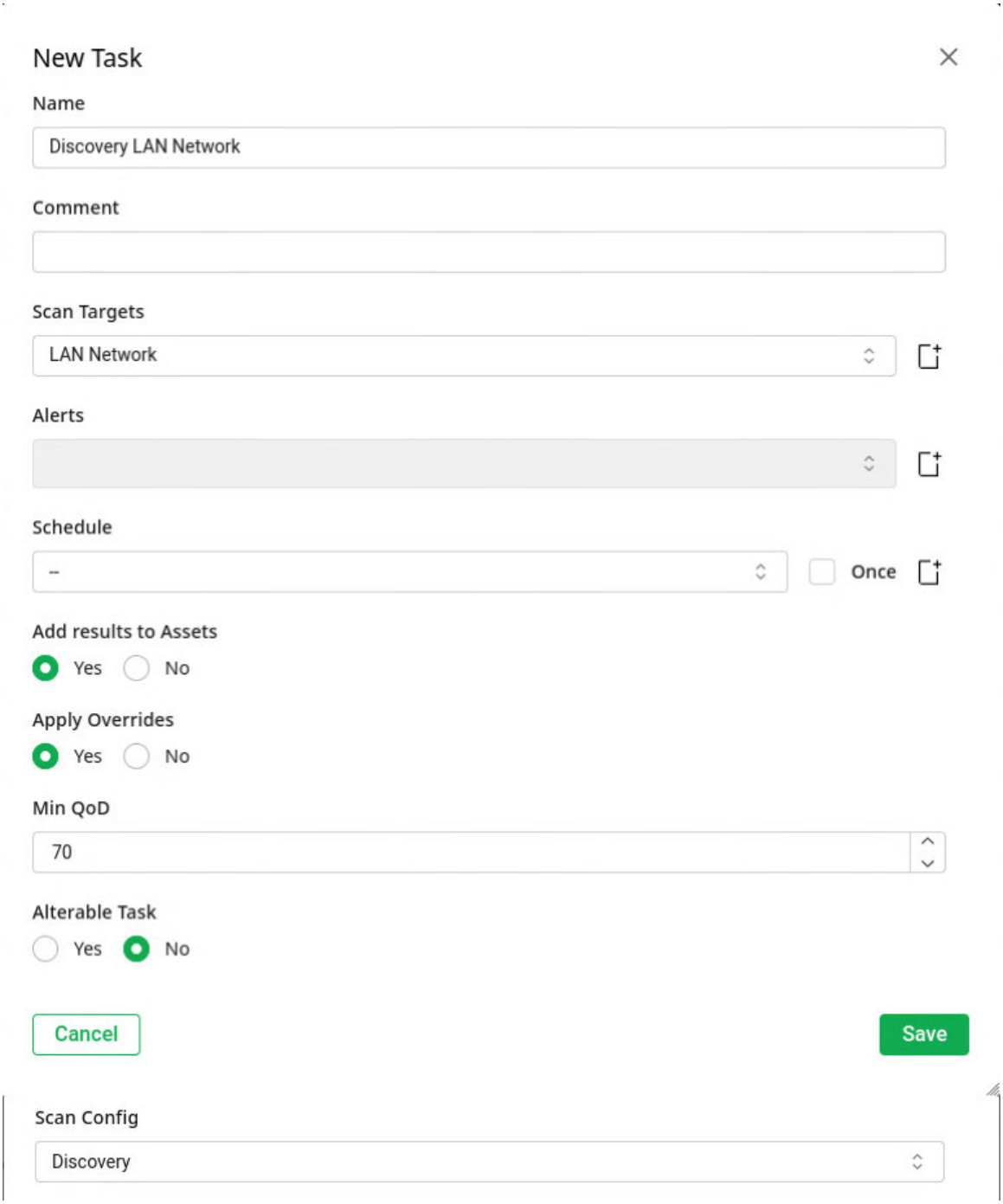
Figura 229. Creación de una nueva tarea de escaneo en OpenVAS.



Nota. Formulario de creación de tarea especificando objetivo y perfil de escaneo. Elaboración propia.

En la sección *Scan config* se debe seleccionar el perfil de escaneo que se desea utilizar. En este caso, se selecciona el perfil de *Discovery* que es un escaneo básico que busca identificar los dispositivos conectados a la red y recopilar información básica sobre ellos. Las demás opciones se pueden dejar por defecto.

Figura 230. Configuración de la tarea de escaneo en OpenVAS.



Nota. Selección del perfil de escaneo Discovery para mapeo inicial de la red. Elaboración propia.

Se guarda la configuración de la tarea de escaneo y se vuelve a la sección *Tasks*. En esta sección se puede ver la nueva tarea de escaneo que se ha creado. Para iniciar el escaneo, se debe hacer clic en el botón de *Start* que se encuentra en la parte derecha de la tarea de

escaneo.

Figura 231. Ejecución del escaneo de la red en OpenVAS.

Nota. Ejecución manual de la tarea de escaneo previamente configurada. Elaboración propia.

El escaneo puede tardar varios minutos en completarse, dependiendo del tamaño de la red y del perfil de escaneo que se haya seleccionado. En la sección *Status* se puede ver el progreso del escaneo y el tiempo estimado para que se complete. Una vez que el escaneo se ha completado, se puede ver el informe de resultados haciendo clic en el nombre de la tarea de escaneo.

8.4.3. Programar escaneos automáticos

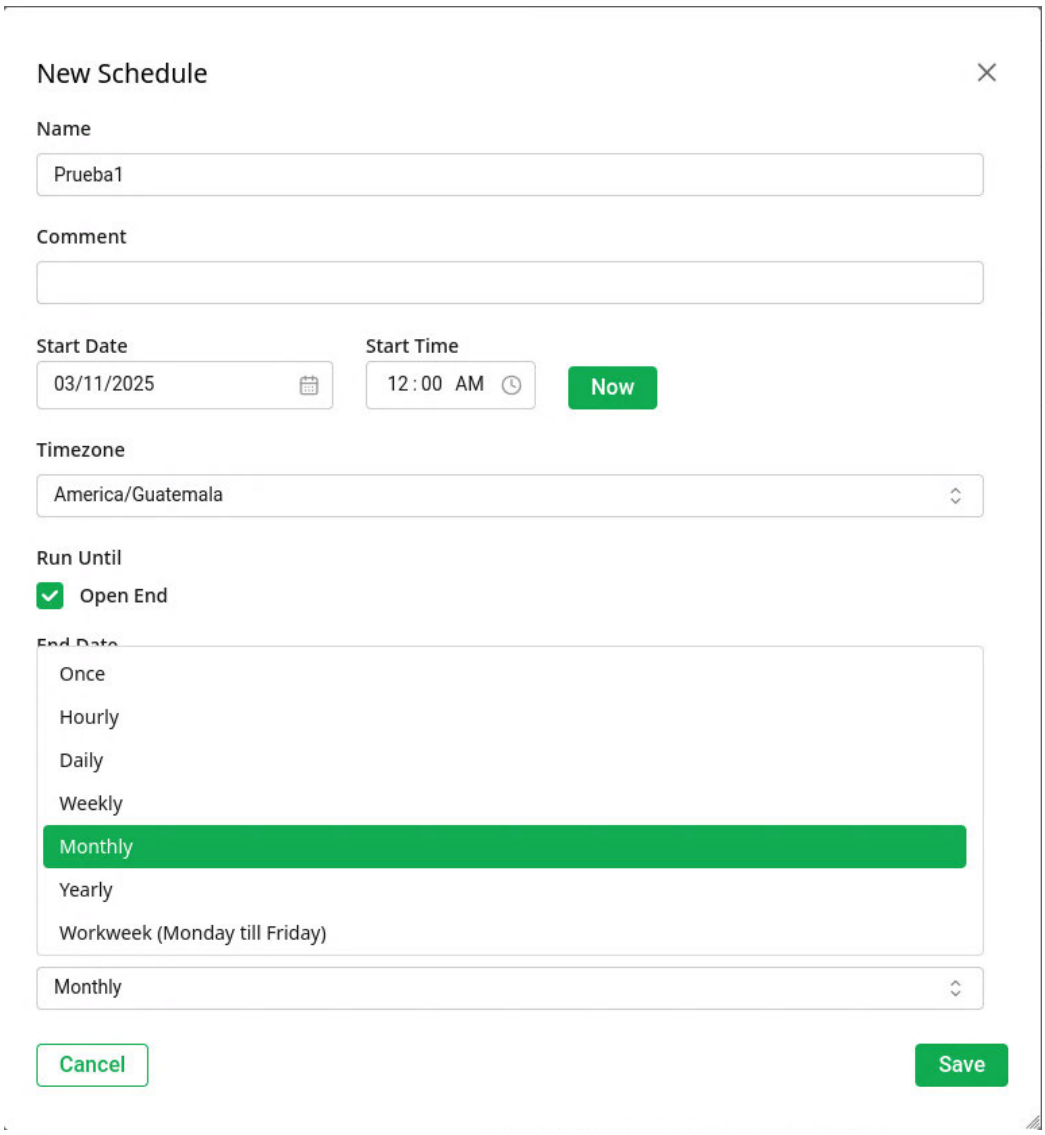
Para programar escaneos automáticos en OpenVAS, se debe acceder a la sección *Configuration* en el menú principal y seleccionar la opción *Schedules*. En esta sección se pueden ver los diferentes horarios de escaneo que se han creado. Para crear un nuevo horario de escaneo, se debe hacer clic en el botón de *New Schedule*.

Figura 232. Creación de un nuevo horario de escaneo en OpenVAS.

Nota. Formulario de creación de horario para escaneos automáticos. Elaboración propia.

En la ventana que se abre, se debe proporcionar un nombre para el horario de escaneo y seleccionar la cuando se desea que empiece el escaneo. Se puede seleccionar una fecha y hora específicas y la zona horaria. Se puede seleccionar la opción *Run Until* para especificar una fecha y hora en la que se desea que el escaneo termine, aunque normalmente no es necesario y se selecciona la opción *Open End* para que el escaneo continúe hasta que se complete. En la sección *Recurrence* se puede seleccionar la frecuencia con la que se desea que se realice el escaneo.

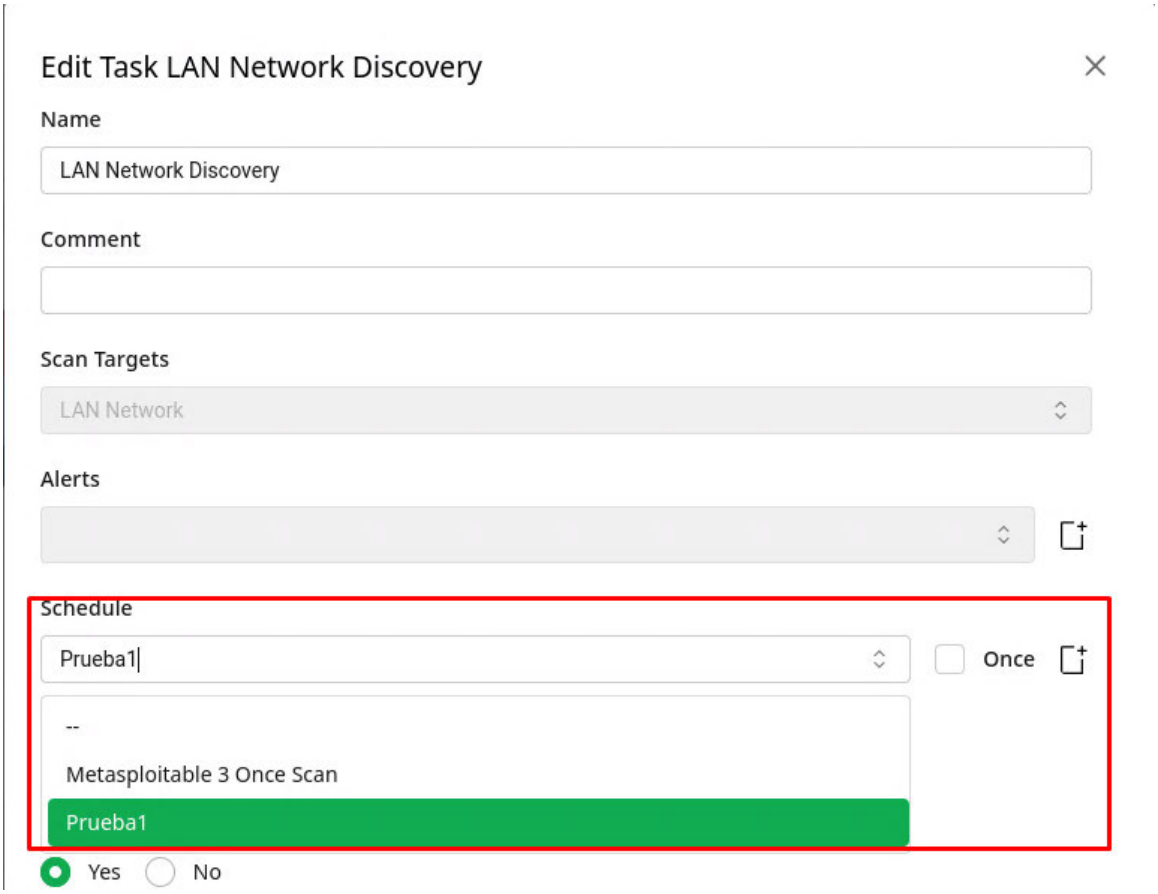
Figura 233. Configuración del horario de escaneo en OpenVAS.



Nota. Definición de recurrencia para escaneos automáticos. Elaboración propia.

Se procede a guardar la configuración del horario de escaneo y se vuelve a la sección *Schedules*. En esta sección se puede ver el nuevo horario de escaneo que se ha creado. Para asignar este horario a una tarea de escaneo, se debe ir a la sección *Scans* en el menú principal y seleccionar la opción *Tasks*. En esta sección se pueden ver las diferentes tareas de escaneo que se han creado. Se selecciona la tarea de escaneo a la que se desea asignar el horario y se hace clic en el botón de *Edit* que se encuentra en la parte derecha de la tarea de escaneo. En la ventana que se abre, se debe seleccionar el horario de escaneo que se ha creado anteriormente en la sección *Schedule*.

Figura 234. Asignación del horario de escaneo a una tarea en OpenVAS.



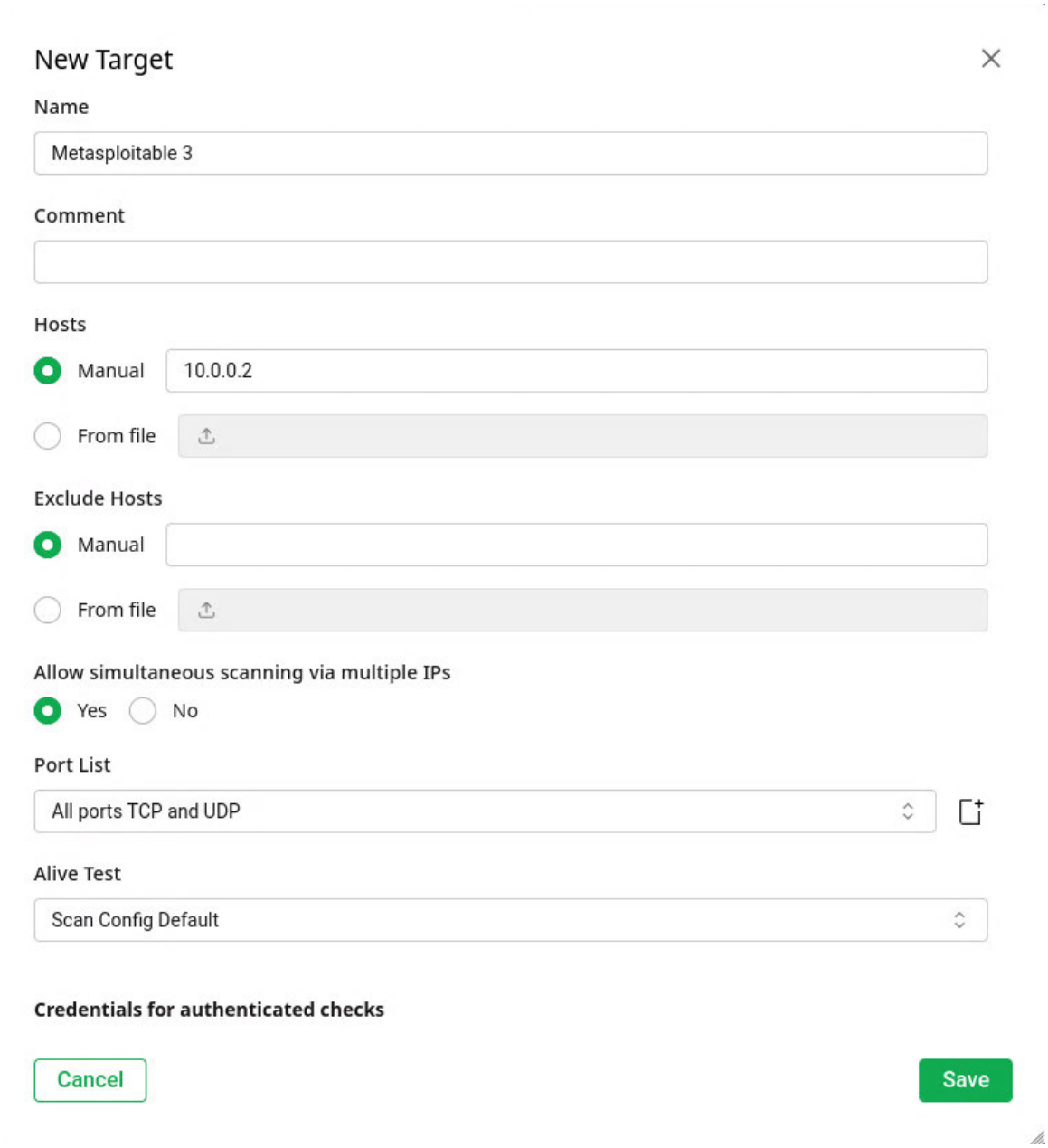
Nota. Selección del horario previamente creado para la tarea de escaneo. Elaboración propia.

Si se desea que el escaneo se realice una sola vez en la fecha y hora especificadas, se debe activar la opción *Once*.

8.4.4. Escaneo de vulnerabilidades del servidor Metasploitable 3

Como se mencionó anteriormente, se tiene un servidor Metasploitable 3 en la red DMZ que se utiliza para probar las capacidades de detección de vulnerabilidades OpenVAS. Para ello, se debe crear una nueva tarea de escaneo en OpenVAS, siguiendo los mismos pasos que se describieron anteriormente. En este caso, se realiza un escaneo de tipo *Full and fast* para identificar todas las vulnerabilidades conocidas en el servidor Metasploitable 3. Por ello en el nombre de la tarea se escribe *Full scan* Metasploitable 3. En el campo *Target* se debe seleccionar la opción *New target* para crear un nuevo objetivo. En la ventana que se abre, se debe proporcionar un nombre para el objetivo y la dirección IP del servidor Metasploitable 3, en este caso, es 10.0.0.2.

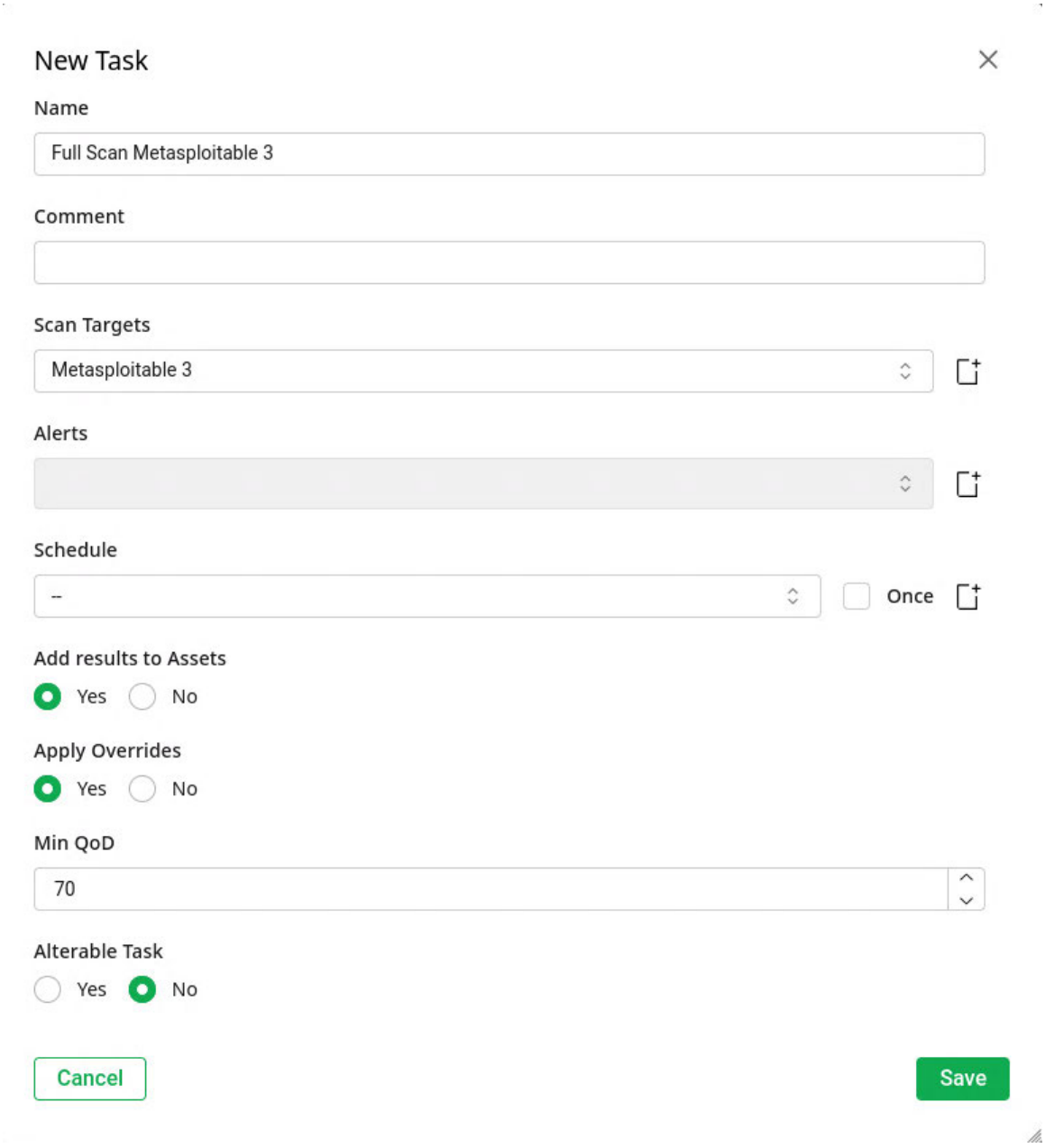
Figura 235. Creación de tarea de escaneo completo en OpenVAS para Metasploitable 3.



Nota. Formulario de creación de tarea especificando objetivo y perfil de escaneo. Elaboración propia.

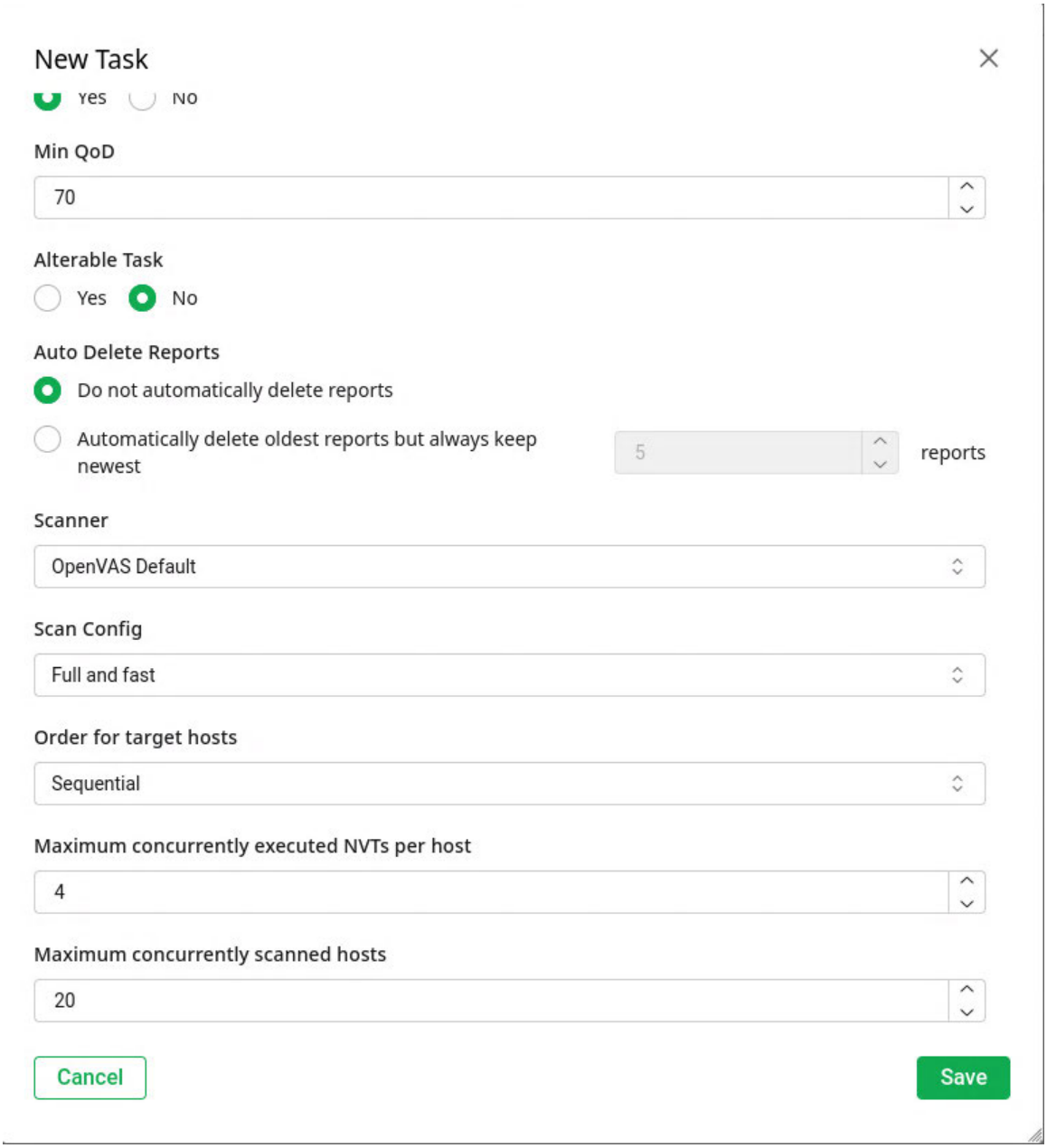
Regresando a la ventana de creación de la tarea de escaneo, en la sección *Scanner* se selecciona la opción OpenVAS *Default*; y en la sección *Scan config* se debe seleccionar el perfil de escaneo que se desea utilizar. En este caso, se selecciona el perfil de *Full and fast* que es un escaneo completo que busca identificar todas las vulnerabilidades conocidas en el objetivo. Las demás opciones se pueden dejar por defecto.

Figura 236. Configuración del perfil de escaneo *Full and fast* en OpenVAS.



Nota. Selección del perfil de escaneo *Full and fast* para análisis exhaustivo del servidor objetivo. Elaboración propia.

Figura 237. Inicio del escaneo completo de Metasploitable 3 en OpenVAS.



Nota. Ejecución manual de la tarea de escaneo previamente configurada. Elaboración propia.

Se guarda la configuración de la tarea de escaneo y se vuelve a la sección *Tasks*. En esta sección se puede ver la nueva tarea de escaneo que se ha creado. Para iniciar el escaneo, se debe hacer clic en el botón de *Start* que se encuentra en la parte derecha de la tarea de escaneo.

Una vez que el escaneo se ha completado, se puede ver el informe de resultados haciendo clic en el nombre de la tarea de escaneo, o se puede ir a la sección *Reports* en el menú

principal y seleccionar el informe correspondiente.

Figura 238. Resumen de resultados del escaneo en OpenVAS.

Nota. Resumen de hallazgos categorizados por severidad tras escaneo exhaustivo del servidor Metasploitable 3. Elaboración propia.

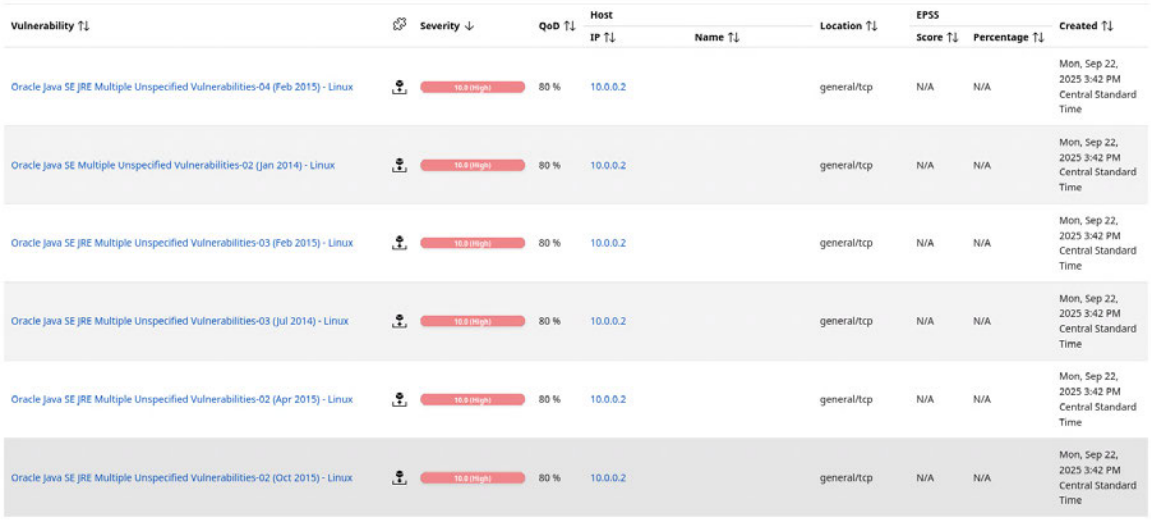
El reporte organiza los resultados en diversas secciones que permiten un análisis exhaustivo. La sección *Results* muestra las vulnerabilidades identificadas, los *hosts* escaneados aparecen en *Hosts*, mientras que los puertos expuestos se detallan en *Ports*. Las aplicaciones detectadas figuran en *Applications* y *Operating Systems* especifica los sistemas operativos encontrados. En CVE se presentan las vulnerabilidades con identificador *Common Vulnerabilities and Expositions*, mientras que *Closed* CVE recoge aquellas ya corregidas. Adicionalmente, *TLS Certificates* contiene los certificados detectados y *Error Messages* documenta los errores generados durante el proceso de escaneo.

Figura 239. Detalle de vulnerabilidades identificadas en el escaneo de OpenVAS.

Nota. Detalle de vulnerabilidades específicas identificadas en el servidor objetivo, incluyendo descripciones y referencias. Elaboración propia.

La sección *Results* es la que más interesa, ya que es donde se pueden ver las vulnerabilidades. En esta sección, también aparece la severidad de cada una, el QoD (*quality of detection*), el *host* donde se han encontrado y los *ports* donde están expuestas.

Figura 240. Listado completo de vulnerabilidades detectadas en OpenVAS.



Vulnerability	Severity	QoD	Host		Location	EPSS		Created
			IP	Name		Score	Percentage	
Oracle Java SE JRE Multiple Unspecified Vulnerabilities-04 (Feb 2015) - Linux	10.0 (High)	80 %	10.0.0.2		general/tcp	N/A	N/A	Mon, Sep 22, 2025 3:42 PM Central Standard Time
Oracle Java SE Multiple Unspecified Vulnerabilities-02 (Jan 2014) - Linux	10.0 (High)	80 %	10.0.0.2		general/tcp	N/A	N/A	Mon, Sep 22, 2025 3:42 PM Central Standard Time
Oracle Java SE JRE Multiple Unspecified Vulnerabilities-03 (Feb 2015) - Linux	10.0 (High)	80 %	10.0.0.2		general/tcp	N/A	N/A	Mon, Sep 22, 2025 3:42 PM Central Standard Time
Oracle Java SE JRE Multiple Unspecified Vulnerabilities-03 (Jul 2014) - Linux	10.0 (High)	80 %	10.0.0.2		general/tcp	N/A	N/A	Mon, Sep 22, 2025 3:42 PM Central Standard Time
Oracle Java SE JRE Multiple Unspecified Vulnerabilities-02 (Apr 2015) - Linux	10.0 (High)	80 %	10.0.0.2		general/tcp	N/A	N/A	Mon, Sep 22, 2025 3:42 PM Central Standard Time
Oracle Java SE JRE Multiple Unspecified Vulnerabilities-02 (Oct 2015) - Linux	10.0 (High)	80 %	10.0.0.2		general/tcp	N/A	N/A	Mon, Sep 22, 2025 3:42 PM Central Standard Time

Nota. Listado completo de vulnerabilidades detectadas con opciones para exportar y analizar resultados. Elaboración propia.

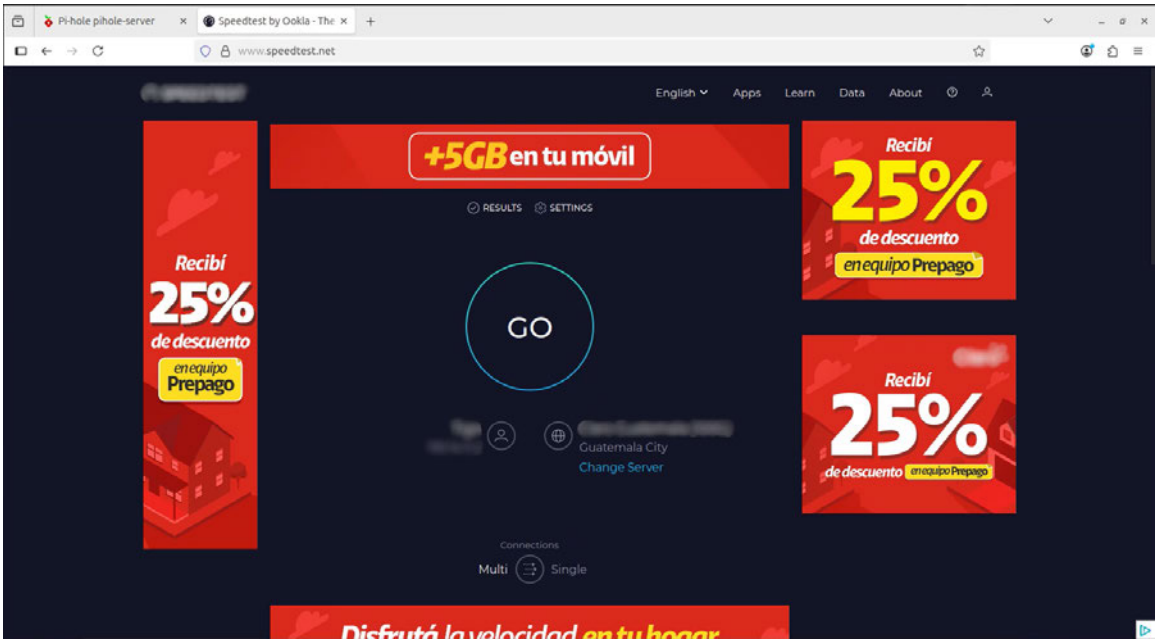
Si se hace clic en una vulnerabilidad específica, se puede ver más información sobre la vulnerabilidad, como su descripción, el resultado de la detección, el método de detección, el impacto, la solución recomendada, y los CVE relacionados.

Por ejemplo, si se hace clic en la vulnerabilidad SSH *Brute force logins with default credentials reporting* se puede ver que es una vulnerabilidad de severidad alta, que se ha encontrado.

8.5. Funcionamiento de Pi-hole

Para probar el funcionamiento de Pi-hole, se puede abrir un navegador web en cualquier dispositivo conectado a la red LAN y acceder a un sitio web que contenga anuncios. En este caso, https://www.speedtest.net/ es un sitio web que contiene anuncios.

Figura 241. Sitio web de Speedtest.net antes de aplicar Pi-hole.

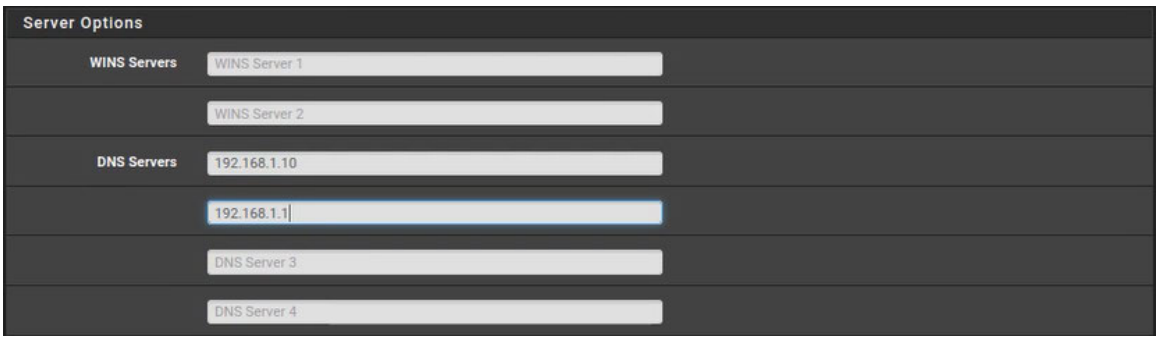


Nota. Carga original del sitio con anuncios visibles previo a redirigir la resolución DNS hacia Pi-hole. Elaboración propia.

En la imagen anterior se puede ver que el sitio web de *Speedtest.net* contiene anuncios. Ahora, se debe cambiar la configuración de DNS del dispositivo para que apunte a la dirección IP del servidor Pi-hole. Esto debería de hacerse en cada dispositivo que se conecte a la red LAN, sin embargo, al estar trabajando con pfSense, se puede configurar el servidor DNS en el servidor DHCP de pfSense para que todos los dispositivos que obtengan una dirección IP del servidor DHCP de pfSense apunten al servidor Pi-hole como servidor DNS.

Para ello, se debe ir a la sección *Services* en el menú principal y luego seleccionar la opción DHCP *Server*. En la pestaña *LAN*, se debe buscar la opción DNS *servers* y se debe ingresar la dirección IP del servidor Pi-hole, en este caso, es 192.168.1.10. Como servidor DNS secundario se agrega la dirección IP del *gateway* de la red LAN (anteriormente el servidor primario) que es 192.168.1.1. Este era antes nuestro servidor primario ya que obtenía las direcciones IP de los servidores DNS que especificamos en la configuración inicial de pfSense, pero ahora se ha cambiado para que apunte al servidor Pi-hole.

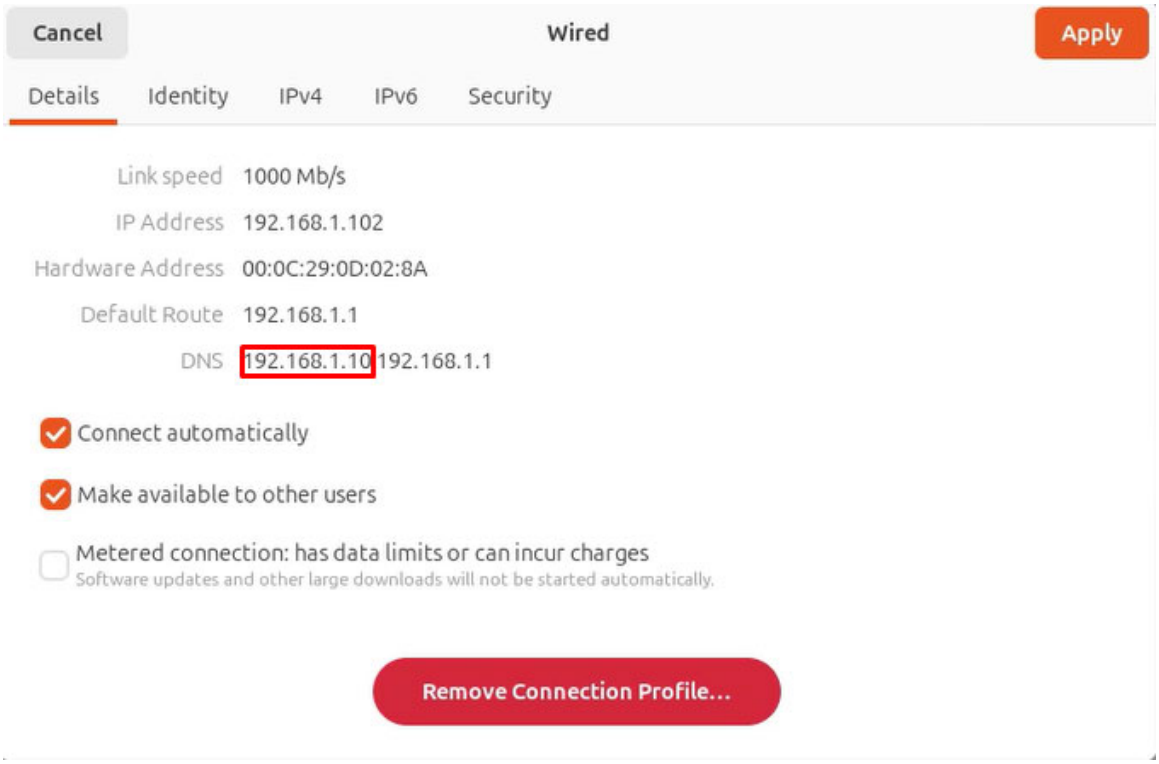
Figura 242. Configuración del servidor DNS en el servidor DHCP de pfSense.



Nota. Ajuste de opciones DHCP para forzar a los clientes a usar Pi-hole (192.168.1.10) como DNS primario. Elaboración propia.

Una vez configurado el servidor DNS en el servidor DHCP de pfSense, se debe guardar los cambios y aplicar la configuración. Una manera de verificar que la configuración se ha aplicado correctamente es reiniciar el dispositivo o renovar la dirección IP del dispositivo para que obtenga la nueva configuración del servidor DHCP.

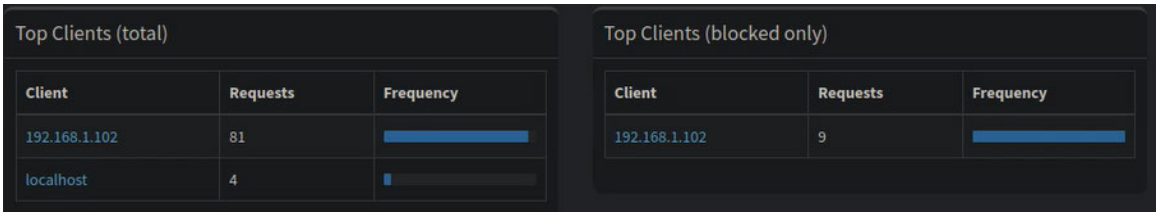
Figura 243. Renovación de la dirección IP en el dispositivo.



Nota. Renovación del lease DHCP para aplicar la nueva configuración DNS apuntando al servidor Pi-hole. Elaboración propia.

Si se ingresa a la interfaz web de Pi-hole ahora se puede ver que ya hay un cliente conectado.

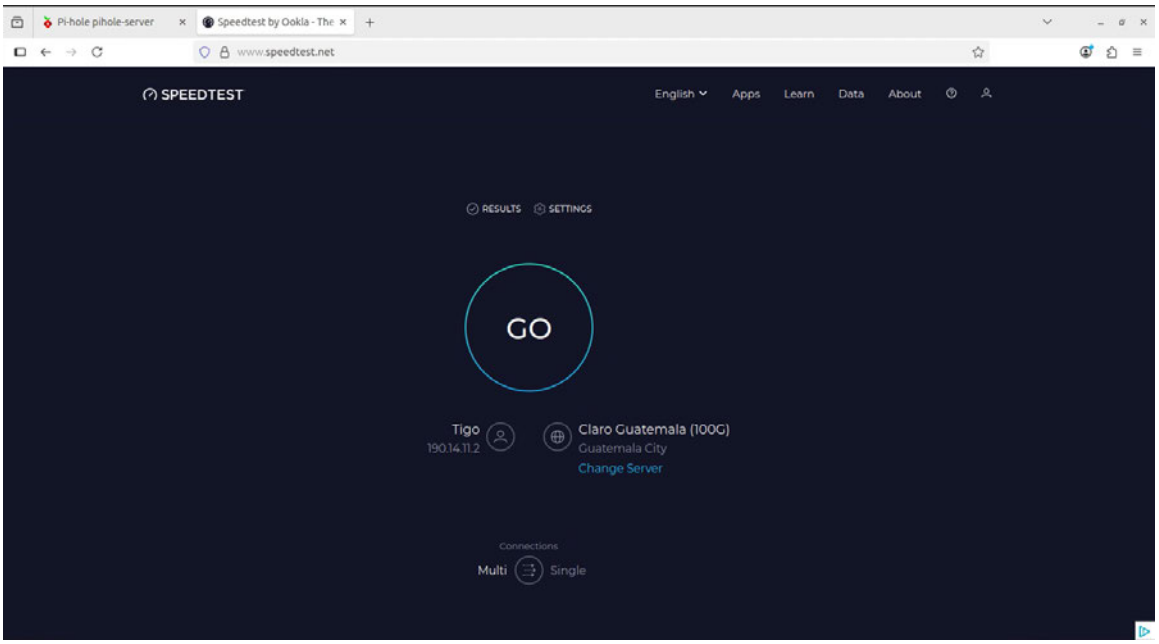
Figura 244. Cliente conectado a Pi-hole.



Nota. *Dashboard* evidenciando primer cliente activo enviando consultas filtradas a través de Pi-hole. Elaboración propia.

Ahora, al volver a acceder al sitio web de *Speedtest.net*, se puede ver que los anuncios han sido bloqueados por Pi-hole.

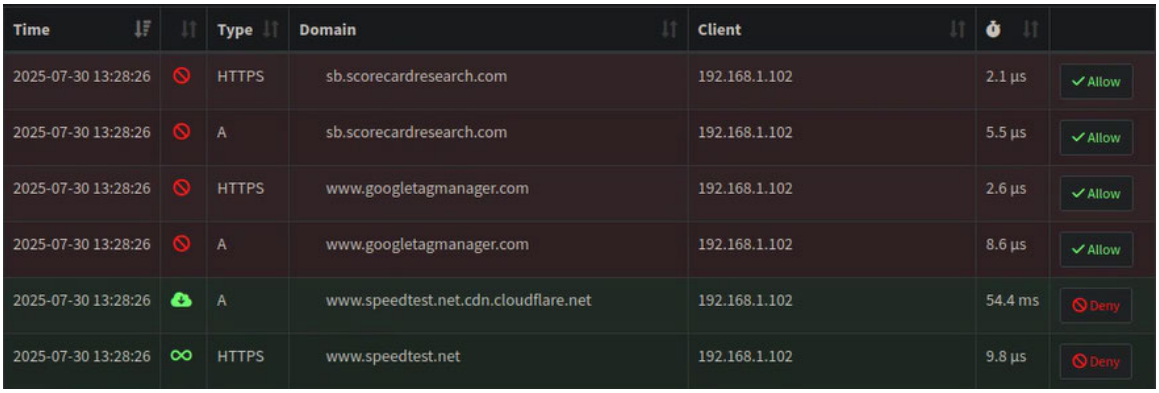
Figura 245. Sitio web de Speedtest.net después de aplicar Pi-hole.



Nota. Página tras adopción de Pi-hole mostrando ausencia de elementos publicitarios previamente cargados. Elaboración propia.

Se puede observar el momento en el que se realiza la consulta a www.speedtest.net y cómo Pi-hole bloquea muchas otras consultas que no son necesarias para el funcionamiento del sitio web, pero que también realizan al momento de cargar la página.

Figura 246. Registro de consultas en Pi-hole al acceder a Speedtest.net.



Time	Type	Domain	Client		
2025-07-30 13:28:26	HTTPS	sb.scorecardresearch.com	192.168.1.102	2.1 µs	Allow
2025-07-30 13:28:26	A	sb.scorecardresearch.com	192.168.1.102	5.5 µs	Allow
2025-07-30 13:28:26	HTTPS	www.googletagmanager.com	192.168.1.102	2.6 µs	Allow
2025-07-30 13:28:26	A	www.googletagmanager.com	192.168.1.102	8.6 µs	Allow
2025-07-30 13:28:26	A	www.speedtest.net.cdn.cloudflare.net	192.168.1.102	54.4 ms	Deny
2025-07-30 13:28:26	HTTPS	www.speedtest.net	192.168.1.102	9.8 µs	Deny

Nota. Registro de consultas inicial evidenciando dominios permitidos y bloqueados durante la carga del sitio. Elaboración propia.

En esta figura claramente se puede ver que Pi-hole ha bloqueado varias consultas a dominios conocidos por servir anuncios y rastreadores. En este específicamente se bloquea el sistema de anuncios de Amazon.

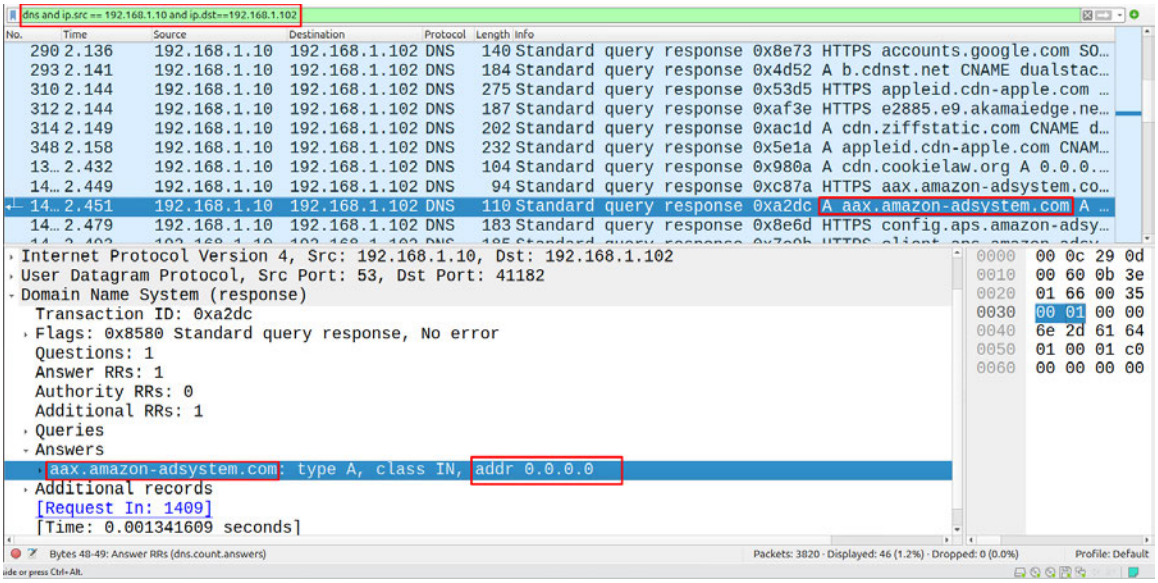
Figura 247. Continuación del registro de consultas en Pi-hole al acceder a Speedtest.net.

2025-07-30 13:28:27	A	aax.amazon-adsystem.com	192.168.1.102	2.4 µs	Allow
2025-07-30 13:28:27	HTTPS	aax.amazon-adsystem.com	192.168.1.102	9.8 µs	Allow
2025-07-30 13:28:27	HTTPS	cdn.cookielaw.org	192.168.1.102	2.6 µs	Allow
2025-07-30 13:28:27	A	cdn.cookielaw.org	192.168.1.102	9.1 µs	Allow

Nota. Continuación del log mostrando bloqueo de múltiples dominios de publicidad y rastreo (ej. Amazon Ads). Elaboración propia.

Para ver de una forma más clara lo que está realizando Pi-hole, se puede ir a la sección *Query Log* en el menú principal de la interfaz web de Pi-hole. En esta sección se pueden ver todas las consultas DNS que se han realizado desde los dispositivos conectados a la red LAN. Se puede ver el nombre de algún dominio que se ha consultado y bloqueado, para después ir a Wireshark y verificar el paquete DNS correspondiente. Por ejemplo, se puede ver que se ha bloqueado la consulta al dominio aax.amazon-adsystem.com al momento de cargar la página de Speedtest.net. Si se observa el paquete DNS correspondiente en Wireshark donde se entrega la respuesta a la consulta tipo A de ese dominio, que debería de ser la dirección IP del servidor de anuncios, se puede ver que la respuesta es la dirección IP 0.0.0.0; lo que indica que Pi-hole ha bloqueado la consulta y no ha entregado una dirección IP válida para ese dominio. Se utiliza la IP 0.0.0.0 para indicar que el dominio no existe o no está disponible.

Figura 248. Paquete DNS bloqueado por Pi-hole en Wireshark.



Nota. Captura de Wireshark mostrando respuesta DNS de Pi-hole con IP 0.0.0.0.

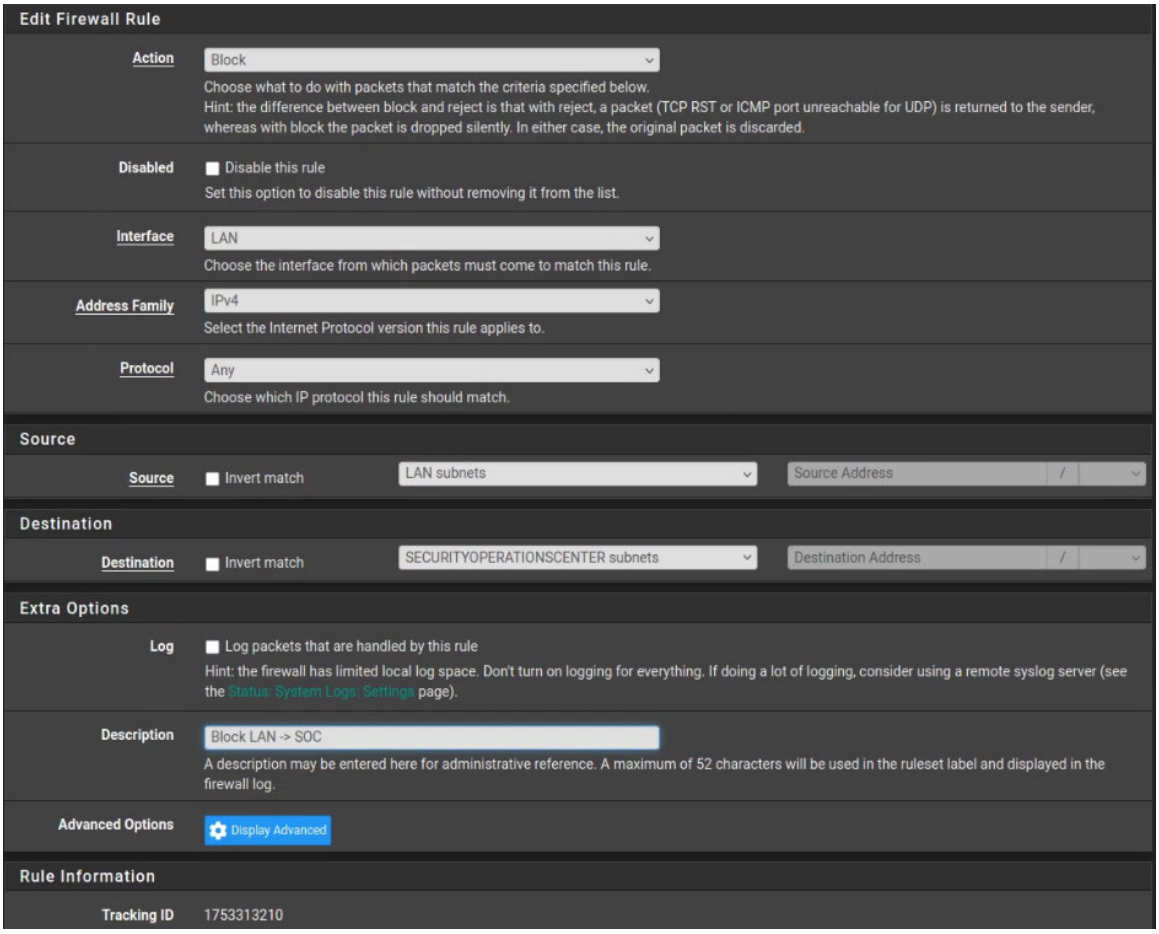
8.6. Creación de reglas de *firewall* como medida de mitigación

El alcance de este proyecto se centraba en la implementación de Snort como sistema de detección en la red, y poner a prueba su funcionamiento mediante la generación de tráfico malicioso. Sin embargo, en un entorno real, es importante tener distintas capas de seguridad para proteger la red y los dispositivos conectados a ella. Una de estas capas es la implementación de reglas de *firewall* para limitar el tráfico entre las diferentes redes y segmentos de la red. Como se vio anteriormente, varios de los ataques fueron entre segmentos de la red, por lo que una medida de mitigación adicional sería implementar reglas de *firewall* para limitar el tráfico entre las diferentes redes y segmentos de la red. Por ejemplo, un segmento importante de la red es el SOC & MOC, que es donde se encuentran los dispositivos de monitoreo y gestión de la red. Por lo tanto, se deben implementar reglas de *firewall* para limitar el tráfico entre la red del SOC & MOC y las demás redes de la infraestructura, ya que no se quiere que los dispositivos de las redes LAN, donde se encuentran los usuarios, y DMZ puedan acceder a estos dispositivos. Sin embargo, se quiere que los dispositivos del SOC & MOC puedan acceder a Internet y estas otras redes para poder realizar las tareas de monitoreo.

Para crear estas reglas en pfSense se debe ir a la sección *Firewall* en el menú principal y seleccionar la opción *Rules*. En esta sección se pueden ver las pestañas con las diferentes interfaces, se selecciona la interfaz del LAN. Se selecciona la opción *Add* con el símbolo de la flecha para arriba para hacer que la regla se aplique antes de las reglas que ya están configuradas. En la acción se selecciona la opción *Block*, para así que si se cumple la condición de la regla, se bloquee el tráfico. En el protocolo se selecciona la opción *any*, que es para

bloquear todo tipo de tráfico. En la dirección de origen se selecciona la opción *LAN subnets*, que es para bloquear el tráfico desde la red LAN. En la dirección de destino se selecciona la opción *SOC subnets*, que es para bloquear el tráfico hacia la red del SOC. Se guarda y se aplica la regla.

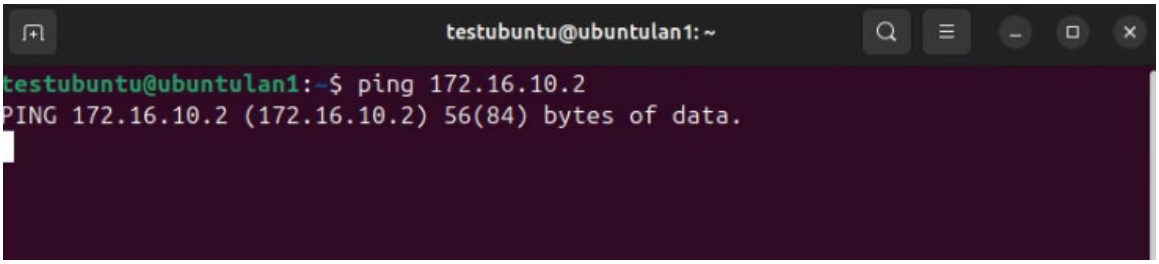
Figura 249. Regla de bloqueo de tráfico desde LAN hacia SOC en pfSense.



Nota. Regla BLOCK entre LAN y SOC para limitar superficie de ataque hacia activos de monitoreo. Elaboración propia.

Para comprobar que la regla se ha aplicado correctamente, se puede intentar hacer un *ping* desde la máquina virtual que está conectada a la red LAN hacia la dirección IP del servidor de monitoreo, en este caso, es 172.16.10.2. Si se recibe una respuesta, significa que la regla no se ha aplicado correctamente. Si no se recibe una respuesta, significa que la regla se ha aplicado correctamente y el tráfico desde la red LAN hacia la red del SOC está bloqueado.

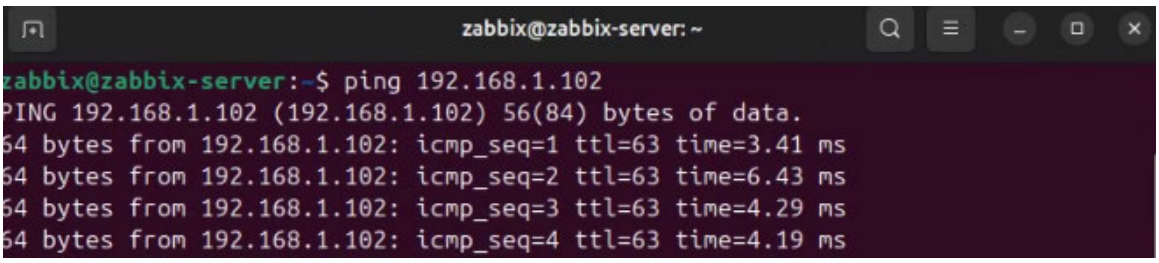
Figura 250. Resultado del ping desde la red LAN hacia el SOC.



Nota. Prueba fallida de ping mostrando bloqueo esperado reforzando política de aislamiento. Elaboración propia.

En cambio, si se intenta hacer un *ping* desde la máquina virtual que está conectada a la red del SOC hacia la dirección IP de la máquina virtual que está conectada a la red LAN, en este caso, 192.168.1.102. Si se recibe una respuesta, significa que la regla se ha aplicado correctamente y el tráfico desde la red del SOC hacia la red LAN está permitido. Si no se recibe una respuesta, significa que la regla no se ha aplicado correctamente y el tráfico desde la red del SOC hacia la red LAN está bloqueado.

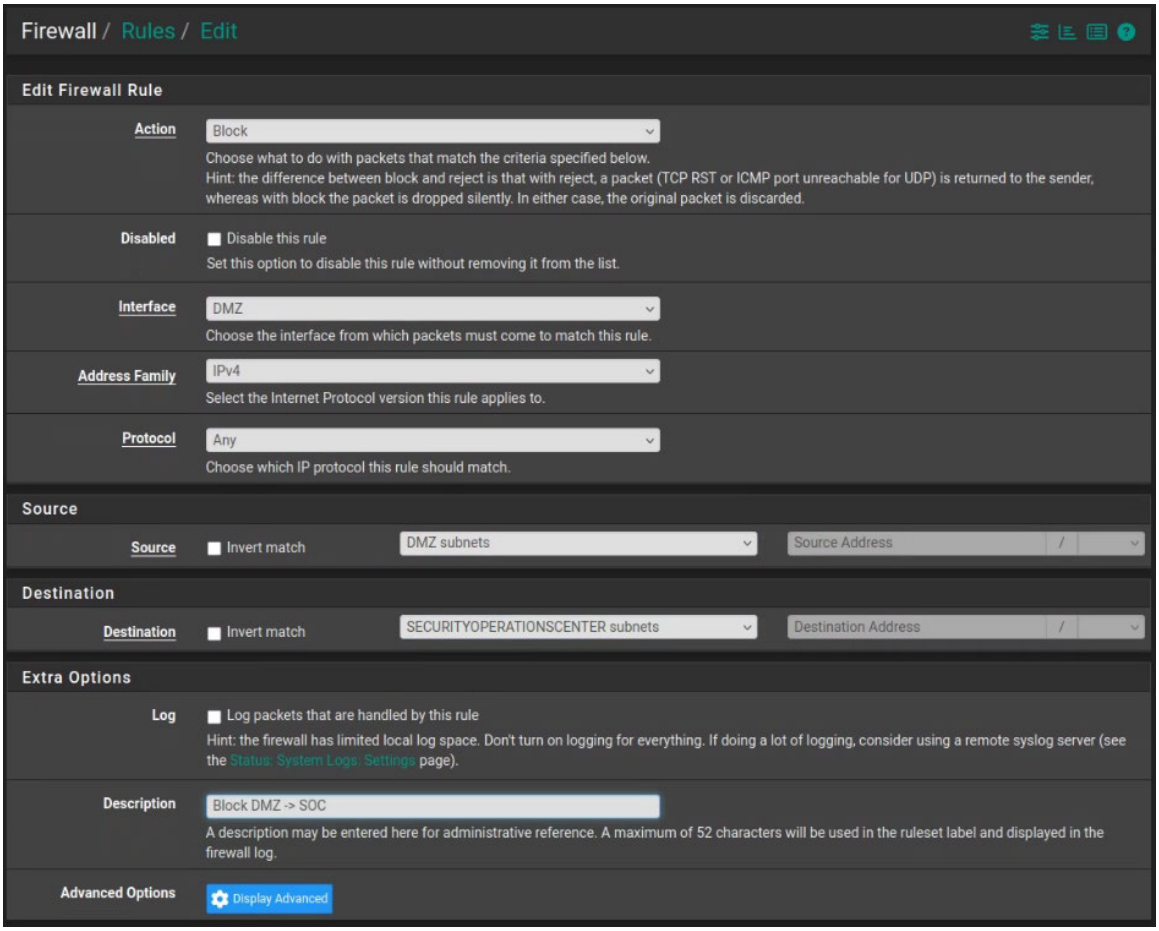
Figura 251. Resultado del ping desde el SOC hacia la red LAN.



Nota. Respuesta positiva de ping evidenciando política asimétrica: SOC puede monitorear recursos en LAN. Elaboración propia.

Del mismo modo se hace con la interfaz de red DMZ, se selecciona la interfaz de red DMZ y se configura una regla de *firewall* para bloquear el tráfico desde la red DMZ hacia la red del SOC. En este caso, se selecciona la opción *Block* en la acción, se selecciona la opción *any* en el protocolo, se selecciona la opción *DMZ subnets* en la dirección de origen y se selecciona la opción *SOC subnets* en la dirección de destino. Se guarda y se aplica la regla.

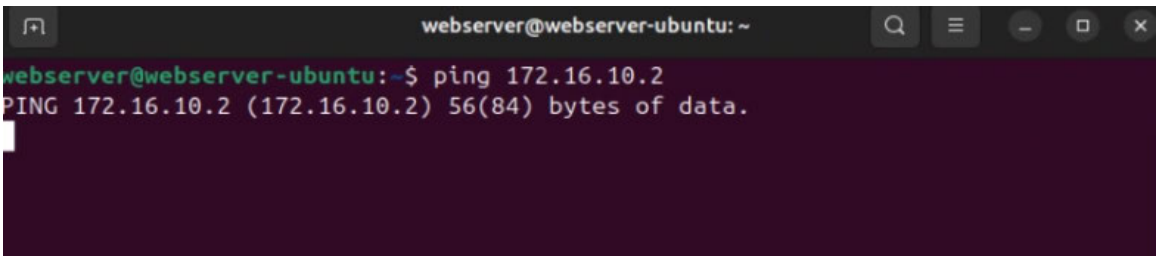
Figura 252. Regla de bloqueo de tráfico desde DMZ hacia SOC en pfSense.



Nota. Regla BLOCK entre DMZ y SOC evitando que servicios expuestos interfieran con infraestructura de monitoreo. Elaboración propia.

Para comprobar que la regla se ha aplicado correctamente, se puede intentar hacer un *ping* desde la máquina virtual que está conectada a la red DMZ hacia la dirección IP del servidor de monitoreo, en este caso, 172.16.10.2. Si se recibe una respuesta, significa que la regla no se ha aplicado correctamente. Si no se recibe una respuesta, significa que la regla se ha aplicado correctamente y el tráfico desde la red DMZ hacia la red del SOC está bloqueado.

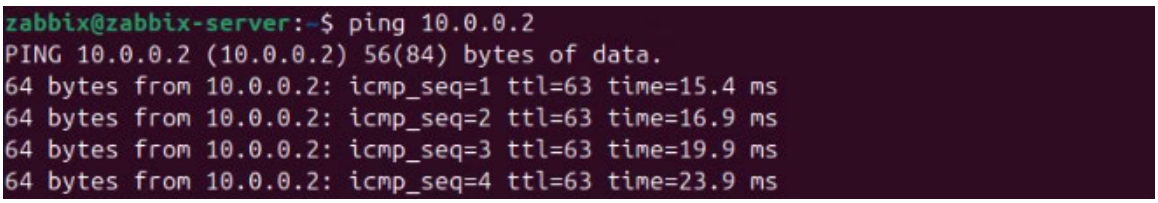
Figura 253. Resultado del ping desde la red DMZ hacia el SOC.



Nota. Intento de ping bloqueado confirmando aislamiento entre DMZ y SOC según política de seguridad. Elaboración propia.

En cambio, si se intenta hacer un *ping* desde la máquina virtual que está conectada a la red del SOC hacia la dirección IP del *webserver* que está conectada a la red DMZ, en este caso, 10.0.0.2 . Si se recibe una respuesta, significa que la regla se ha aplicado correctamente y el tráfico desde la red del SOC hacia la red DMZ está permitido. Si no se recibe una respuesta, significa que la regla no se ha aplicado correctamente y el tráfico desde la red del SOC hacia la red DMZ está bloqueado.

Figura 254. Resultado del ping desde el SOC hacia la red DMZ.



Nota. Ping exitoso desde SOC hacia DMZ mostrando visibilidad necesaria para monitoreo y pruebas. Elaboración propia.

Con esto se ha completado la configuración de las redes LAN, DMZ y SOC en pfSense, y se ha verificado que las reglas de *firewall* se han aplicado correctamente.

CAPÍTULO 9

Conclusiones

1. El despliegue de pfSense como *firewall* y *router* ha demostrado ser una solución efectiva ya que proporciona una amplia gama de funcionalidades como la posibilidad de crear reglas de filtrado avanzadas, gestionar el tráfico de red e integrar servidores DHCP y un conjunto de paquetes adicionales que mejoran la funcionalidad nativa del sistema. Se logró una implementación exitosa asegurando la conectividad entre segmentos e Internet, garantizando la seguridad de la red mediante políticas de acceso adecuadas.

2. La adición de Snort como sistema de detección y prevención de intrusiones ha fortalecido significativamente la seguridad de la red. Snort ha demostrado ser capaz de identificar y mitigar diversas amenazas, incluyendo ataques de denegación de servicio (DoS), escaneos de puertos y actividades maliciosas. La configuración adecuada de Snort permitió una detección precisa y una respuesta rápida a incidentes de seguridad, contribuyendo a mantener la integridad y disponibilidad de los recursos de la red.

3. La implementación del conjunto de reglas personalizadas para Snort ha mejorado la capacidad de detección de amenazas específicas a la red. Estas reglas han sido diseñadas para abordar vulnerabilidades y patrones de ataque relevantes, lo que ha resultado en la facilidad de detección de actividades maliciosas. Sin embargo, es importante destacar que el listado de reglas creado no es lo suficientemente exhaustivo para cubrir todas las posibles amenazas. Por lo tanto, se recomienda una revisión y actualización continua de las reglas para adaptarse a las nuevas amenazas y mejorar la eficacia del sistema de detección de intrusiones.

4. Los *scripts* desarrollados para la automatización de bloqueo y desbloqueo de direcciones IP han simplificado la gestión de la seguridad en la red, permitiendo una respuesta más rápida ante incidentes. Además, añaden una capa de personalización al incluir criterios específicos para el bloqueo de direcciones IP, lo cual evita bloqueos innecesarios.

5. Zabbix es una herramienta de monitoreo y gestión de infraestructura que permite a los administradores de sistemas supervisar el estado de los dispositivos y servicios en tiempo real. Su implementación en la red ha permitido una visibilidad completa

de la infraestructura, facilitando la detección y resolución de problemas de manera proactiva.

6. OpenVAS demostró ser una solución completa para la identificación de vulnerabilidades en los sistemas y servicios desplegados en la red. Su capacidad para realizar evaluaciones periódicas y generar reportes detallados mostró ser una herramienta valiosa para mantener la seguridad de la red y mitigar riesgos potenciales.

7. El bloqueo de anuncios mediante Pi-hole es efectivo para mejorar la experiencia de navegación al reducir la cantidad de anuncios y rastreadores que afectan el rendimiento y la privacidad del usuario. Es importante mencionar que, aunque Pi-hole no bloquea todos los tipos de anuncios, su implementación ha demostrado ser una herramienta valiosa para mejorar la experiencia de navegación en general. En comparación con otros métodos de bloqueo de anuncios, Pi-hole ofrece una solución más cómoda y efectiva, ya que se integra en la red local y no requiere la instalación de extensiones en cada dispositivo. En el caso del bloqueo de anuncios mediante un servidor DNS, se debe tener en cuenta que la lista de bloqueo puede afectar la experiencia del usuario al bloquear sitios web legítimos. Por lo tanto, es recomendable mantener una lista de bloqueo actualizada y revisarla periódicamente para evitar problemas.

CAPÍTULO 10

Recomendaciones

- Realizar el sistema en un entorno no virtualizado para evaluar el desempeño y la estabilidad en condiciones más cercanas a un entorno de producción. Esto permitirá identificar posibles limitaciones o problemas que puedan surgir en un entorno real y ajustar las configuraciones y optimizaciones necesarias para garantizar un rendimiento óptimo. Sin embargo, es importante considerar que la implementación en un entorno físico puede implicar costos adicionales y requerir una infraestructura adecuada para soportar el sistema.
- Las pruebas de ataque realizadas podrían ampliarse para incluir escenarios con mayor carga y variedad de técnicas. Esto ayudaría a evaluar la robustez del sistema bajo condiciones más extremas y a identificar posibles puntos débiles que podrían ser explotados por atacantes sofisticados.
- Implementar un SIEM (*security information and event management*) al sistema en complemento con Snort para tener una visión más integral de la seguridad de la red. Un SIEM puede correlacionar eventos de múltiples fuentes, proporcionando alertas más precisas y facilitando la investigación de incidentes de seguridad. Wazuh y Splunk son ejemplos populares de SIEM que podrían integrarse con Snort para mejorar la capacidad de monitoreo y respuesta ante incidentes.
- Emplear el sistema propuesto utilizando otras tecnologías *open source* para la detección y prevención de intrusiones, como Suricata o Bro/Zeek; para el *firewall*, se podría considerar OPNsense o IPFire. Esto permitiría comparar el desempeño, facilidad de configuración y efectividad en la detección de amenazas entre diferentes soluciones, ayudando a identificar la mejor opción para distintos escenarios y necesidades. Incluso la combinación de múltiples sistemas IDS/IPS podría ofrecer una defensa en profundidad más robusta. Además, se podrían explorar técnicas avanzadas de detección basadas en inteligencia artificial y aprendizaje automático para mejorar la capacidad de identificación de amenazas emergentes.
- Profundizar en la creación y la optimización de reglas personalizadas para Snort, adaptándolas a las amenazas específicas que enfrenta la red en cuestión, incluso utilizar

todas las reglas disponibles en las fuentes comunitarias y comerciales que ofrece Snort para así realizar una comparación de efectividad entre las reglas personalizadas y las reglas estándar. También se puede explorar la opción de utilizar estas reglas para enfocarse en otro aspecto que no sea la creación de reglas personalizadas, como la optimización del rendimiento del sistema, la integración con otras herramientas de seguridad o la integración del sistema utilizando contenedores Docker para facilitar su despliegue y gestión.

CAPÍTULO 11

Referencias

[1] S. Dwiyatno, W. A. Andriani, A. P. Sari y Sulistiyono, «Implemetation of Snort IPS using PfSense as network forensic in Smk XYZ,» en *Proceedings of the 1st International Multidisciplinary Conference on Education, Technology, and Engineering (IMCETE 2019)*, Atlantis Press, 2020, págs. 186-192, ISBN: 978-94-6252-918-2. DOI: 10.2991/assehr.k.200303.044. https://doi.org/10.2991/assehr.k.200303.044.

[2] W. Buqing, «Analysis of a new firewall constructed on Pfsense with Snort to defend against common internet intrusions,» *Applied and Computational Engineering*, vol. 43, págs. 244-250, feb. de 2024. DOI: 10.54254/2755-2721/43/20230841.

[3] N. Gavrilovic, V. Ciric y N. Lozo, «Snort ids system visualization interface for alert analysis,» *Serbian Journal of Electrical Engineering*, vol. 19, págs. 67-78, ene. de 2022. DOI: 10.2298/SJEE2201067G.

[4] D. Mena Caballinas, «Seguridad en internet de las cosas,» Tesis de mtría., Escuela Técnica Superior de Ingeniería de Sistemas Informáticos, 2019.

[5] A. Trisolino, «Analysis of security configuration for IDS/IPS,» Tesis de mtría., Politecnico di Torino, 2023.

[6] J. M. Kizza, *Guide to Computer Network Security*, 6.ª ed. Springer Nature, 2024, ISBN: 9783031475498.

[7] Fortinet. «What is network security?» https://www.fortinet.com/resources/cyberglossary/what-is-network-security.

[8] M. Ciampa, *CompTIA Security+ Guide to Network Security Fundamentals*, 8.ª ed. Cengage Learning US, 2024, ISBN: 979-82-14-00074-9.

[9] A. Thomas, *IDS and IPS with Snort 3. Get up and running with Snort 3 and discover effective solutions to your security issues.* Packt Publishing, Limited, 2024, ISBN: 9781800566163.

[10] IBM. «What is network security?» https://www.ibm.com/think/topics/network-security.

[11] Imperva. «Cybersecurity threats.» https://www.imperva.com/learn/application-security/cyber-security-threats/.

[12] Cloudflare. «What is a denial-of-service (DoS) attack?» https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/.

[13] Fortinet. «DoS attack vs DDoS attack.» https://www.fortinet.com/resources/cyberglossary/dos-vs-ddos.

[14] Fortinet. «What is network segmentation?» https://www.fortinet.com/resources/cyberglossary/network-segmentation.

[15] Fortinet. «What is a DMZ network?» https://www.fortinet.com/resources/cyberglossary/what-is-dmz.

[16] Netgate. «pfSense software documentation: introduction.» https://docs.netgate.com/pfsense/en/latest/general/index.html.

[17] Snort. «Snort FAQ.» https://www.snort.org/faq/what-is-snort.

[18] B. Lenaerts-Bergmans. «Snort and Snort rules explained.» https://www.crowdstrike.com/en-us/cybersecurity-101/threat-intelligence/snort-rules/.

[19] T. S. Project. «Snort users manual 2.9.16.» http://manual-snort-org.s3-website-us-east-1.amazonaws.com/snort_manual.html.

[20] Fortinet. «What is SNMP? How does simple network management protocol work?» https://www.fortinet.com/resources/cyberglossary/simple-network-management-protocol.

[21] V. Cooper. «What is SNMP? How does simple network management protocol work?» https://www.splashtop.com/blog/snmp.

[22] Z. SIA. «What is Zabbix.» https://www.zabbix.com/documentation/current/en/manual/introduction/about.

[23] G. AG. «Greenbone OpenVAS.» https://www.openvas.org/.

[24] G. AG. «Architecture.» https://docs.greenbone.net/GSM-Manual/gos-24.10/en/architecture.html.

[25] G. AG. «Glossary.» https://docs.greenbone.net/GSM-Manual/gos-24.10/en/glossary.html#cpe.

[26] Rapid7. «Metasploitable 3.» https://github.com/rapid7/metasploitable3/wiki.

[27] Pi-hole. «Pi-hole documentation: About Pi-hole.» https://docs.pi-hole.net/.

[28] J. Networks. «DNS sinkhole overview.» https://www.juniper.net/documentation/us/en/software/atp-cloud/atp-cloud-admin-guide/topics/concept/atp-cloud-dns-sinkhole-overview.html.

[29] G. Lyon. «The official Nmap project guide to network discovery and security scanning.» https://nmap.org/book/toc.html.

[30] Cisco. «Configuring DMZ.» https://www.cisco.com/c/dam/assets/sol/sb/isa500_emulator/help/guide/ad1681599.html.

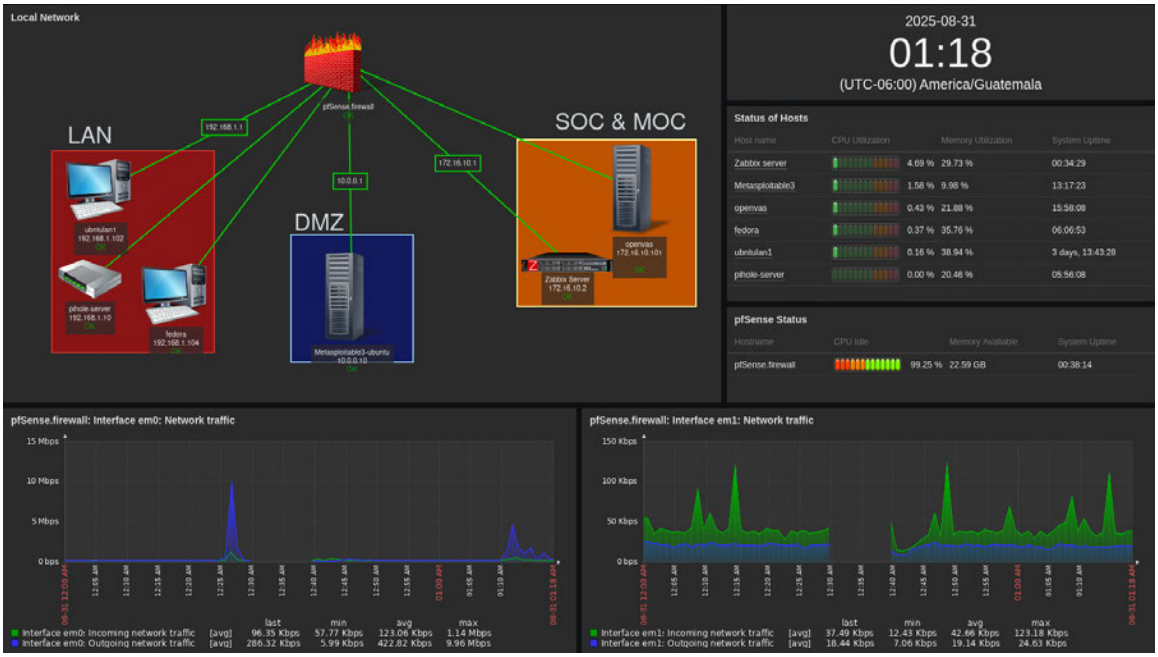
CAPÍTULO 12

Anexos

12.1. *Dashboards* utilizados en Zabbix

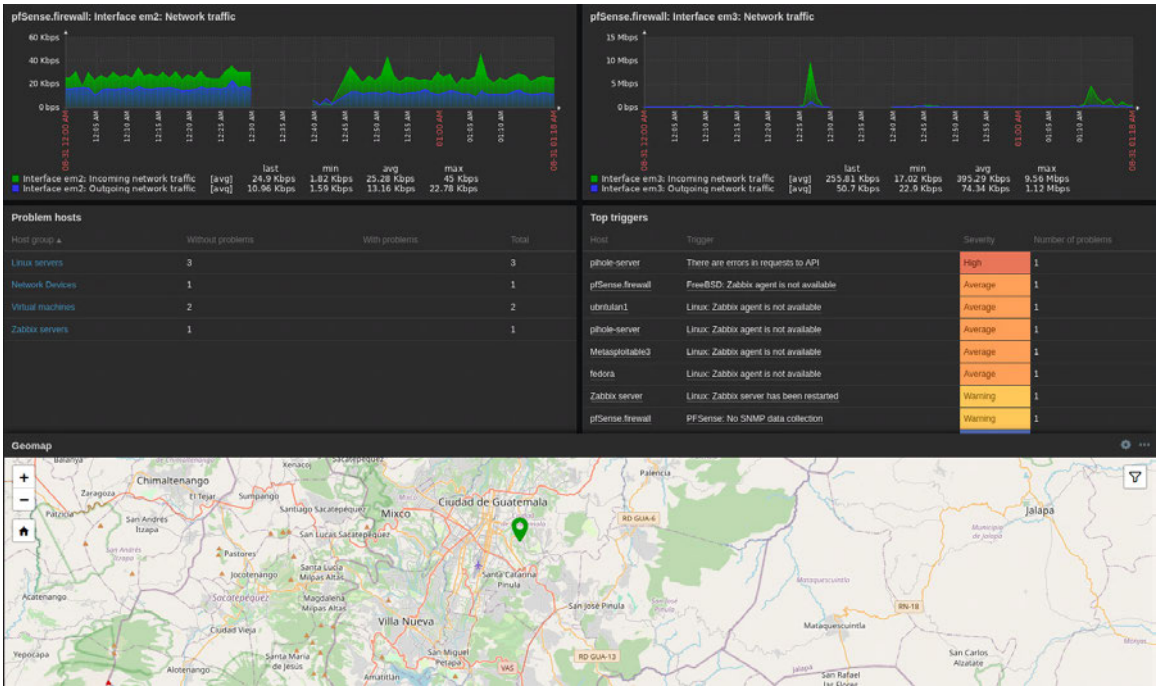
12.1.1. *Overview*

Figura 255. *Dashboard Overview* en Zabbix.



Nota. Vista general del estado de la red, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

Figura 256. *Dashboard* de pfSense en Zabbix.



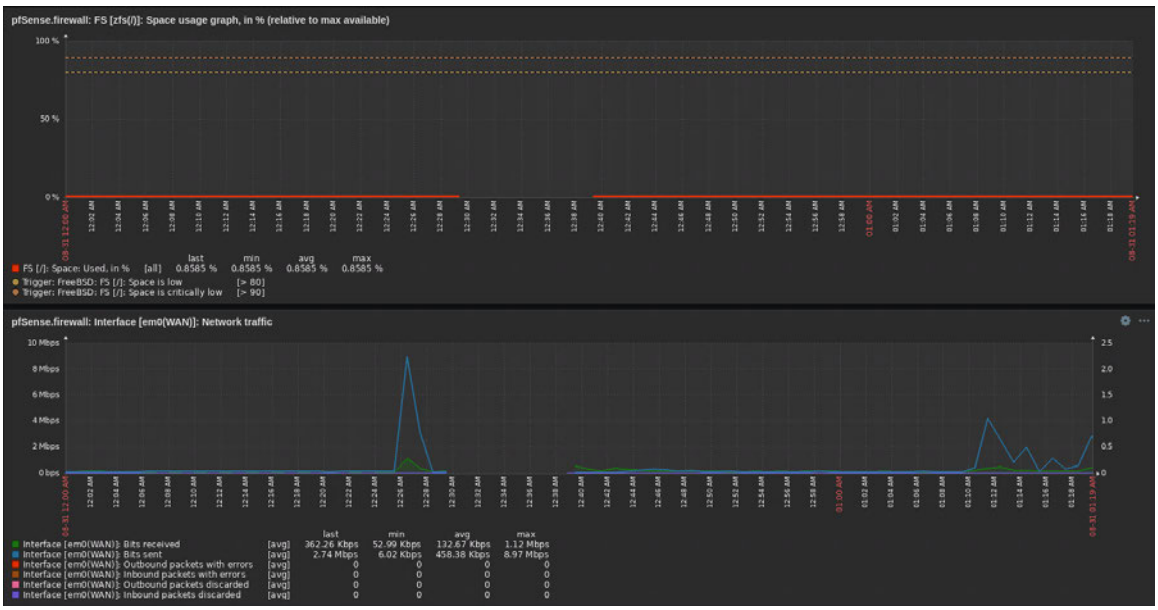
Nota. Vista del estado de la red a través del *firewall* pfSense, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

12.1.2. pfSense

Figura 257. *Dashboard* de pfSense: métricas de tráfico de red.

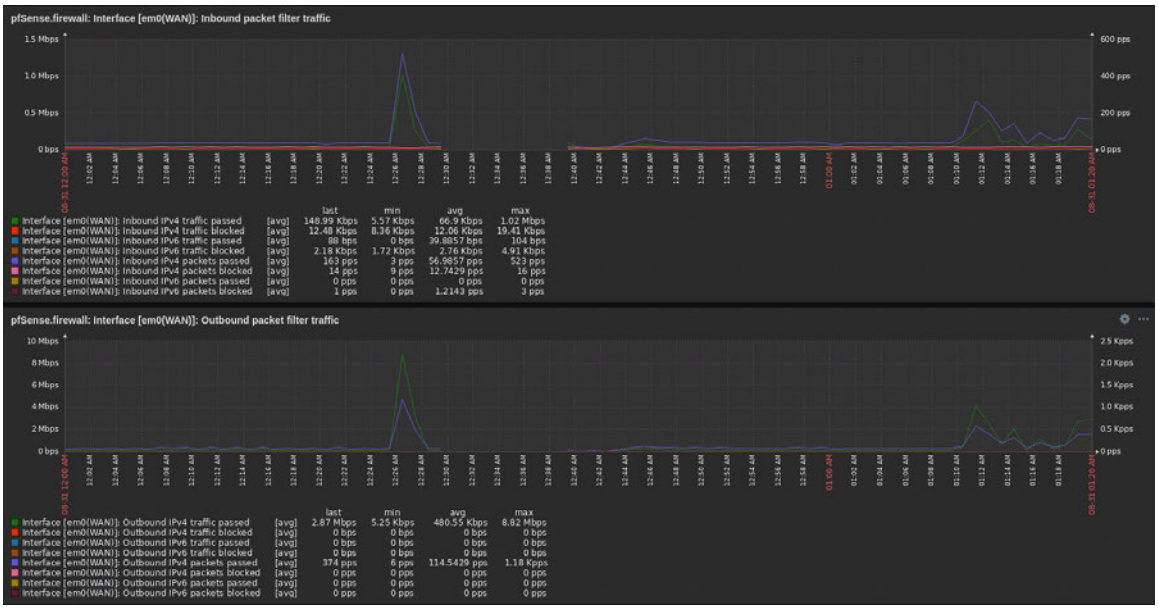
Nota. Vista del estado de la red a través del *firewall* pfSense, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

Figura 258. *Dashboard* de pfSense: monitoreo de interfaces y estados.



Nota. Vista del estado de la red a través del *firewall* pfSense, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

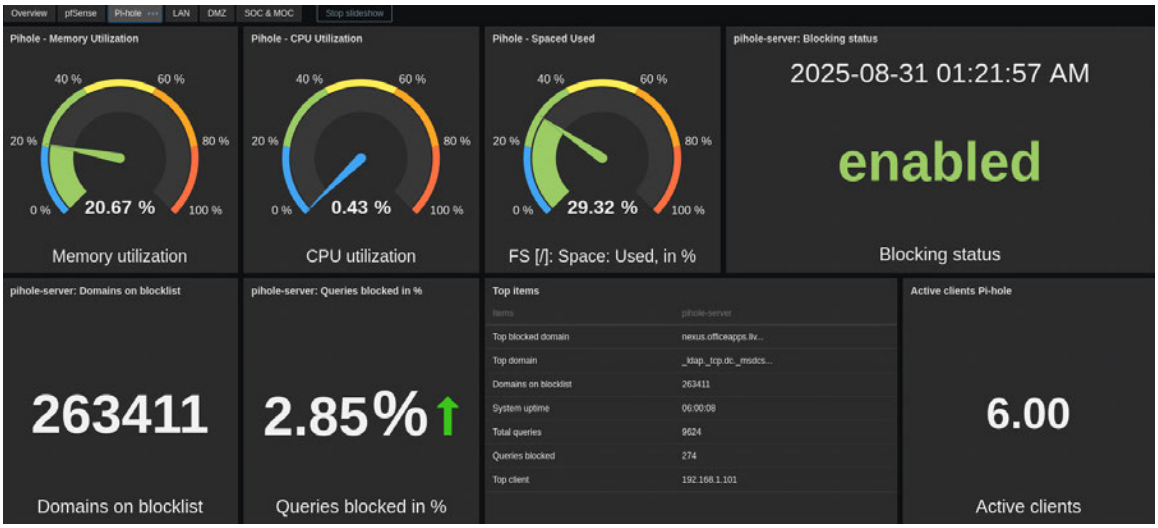
Figura 259. *Dashboard* de pfSense: uso de CPU y memoria del *firewall*.



Nota. Vista del estado de la red a través del *firewall* pfSense, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

12.1.3. Pi-hole

Figura 260. *Dashboard* de Pi-hole en Zabbix.



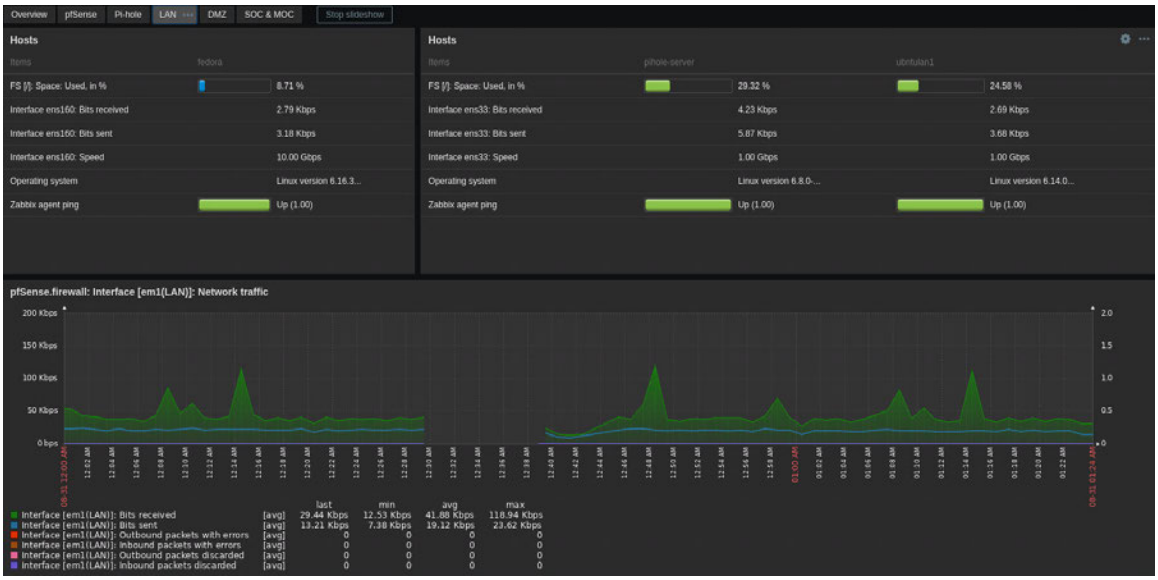
Nota. Vista del estado de la red a través del bloqueador de anuncios Pi-hole, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

Figura 261. *Dashboard* de Pi-hole: estadísticas de bloqueo y consultas DNS.

Nota. Vista del estado de la red a través del bloqueador de anuncios Pi-hole, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

12.1.4. LAN

Figura 262. *Dashboard* de LAN en Zabbix.



Nota. Vista del estado de la red LAN, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

Figura 263. *Dashboard* LAN: gráficas de rendimiento de dispositivos internos.

Nota. Vista del estado de la red LAN, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

12.1.5. DMZ

Figura 264. *Dashboard* DMZ en Zabbix.

Nota. Vista del estado de la red en la DMZ, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

Figura 265. *Dashboard* DMZ: métricas de recursos y disponibilidad de servidores.

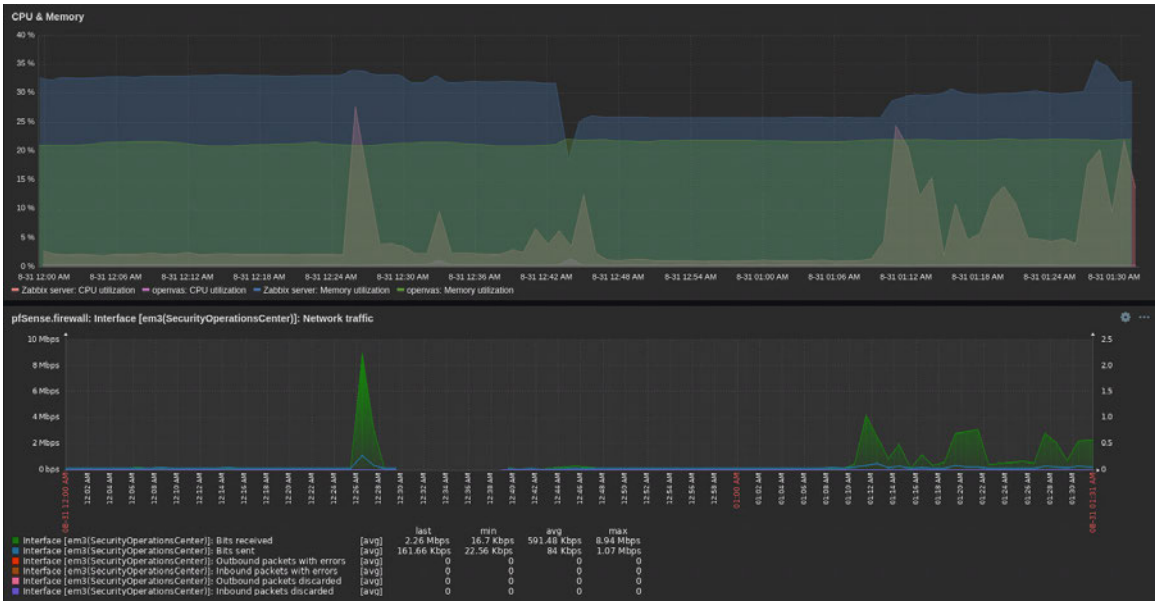
Nota. Vista del estado de la red DMZ, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

12.1.6. SOC & MOC

Figura 266. *Dashboard* SOC en Zabbix.

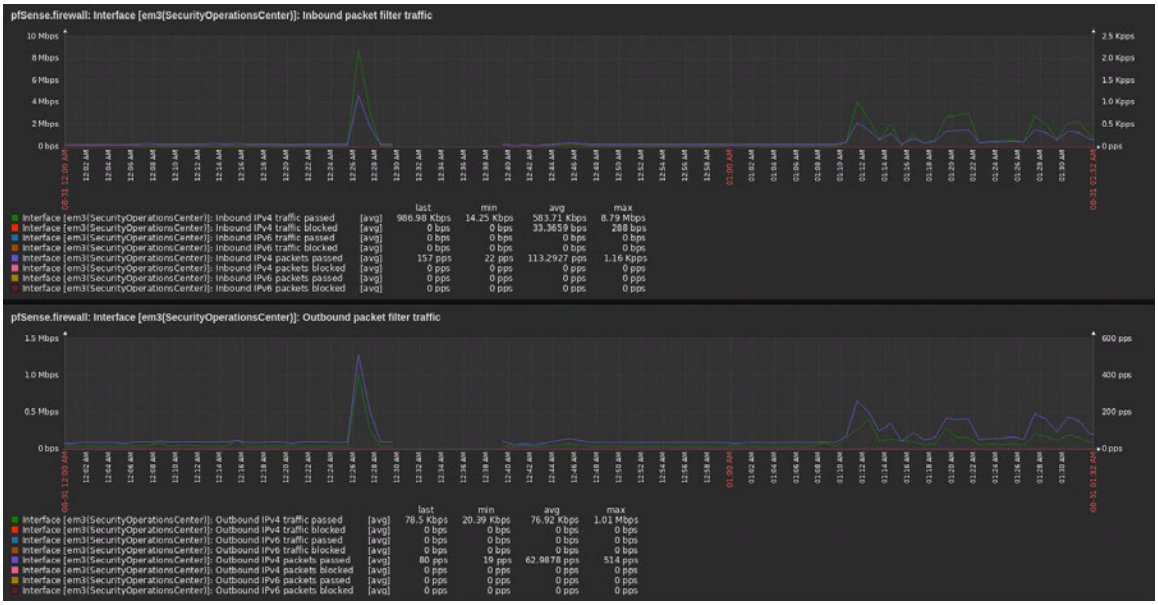
Nota. Vista del estado de la red SOC, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

Figura 267. *Dashboard* MOC en Zabbix.



Nota. Vista del estado de la red MOC, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

Figura 268. *Dashboard* MOC: análisis detallado de rendimiento de red.



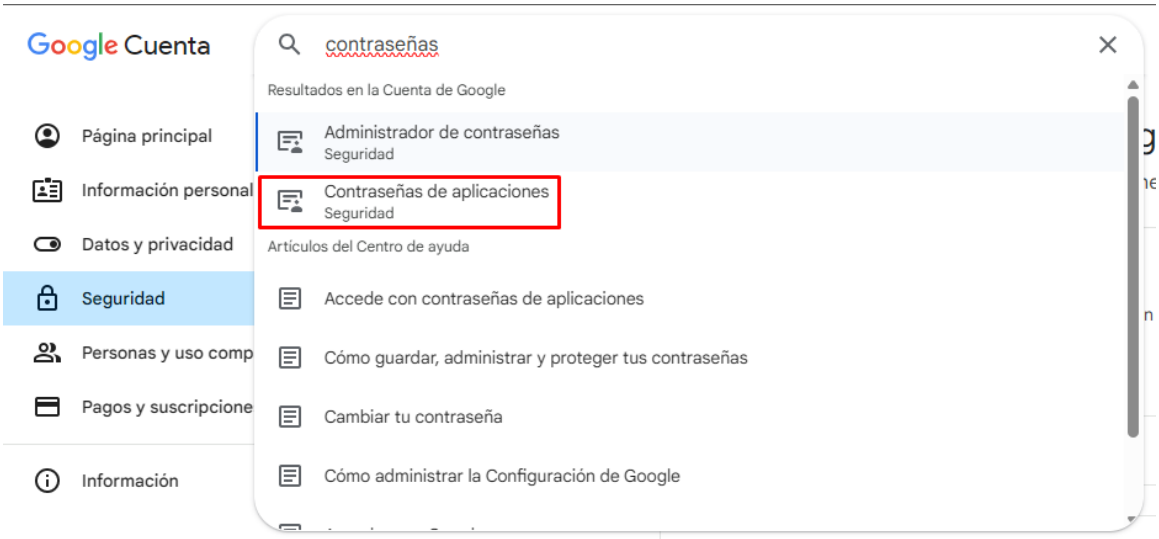
Nota. Vista del estado de la red MOC, mostrando información clave de los dispositivos monitoreados. Elaboración propia.

12.2. Habilitar el uso de contraseñas de aplicaciones en Gmail

Para habilitar el uso de contraseñas de aplicaciones en Gmail, se deben seguir los siguientes pasos:

1. Iniciar sesión en la cuenta de Gmail que se quiera utilizar para recibir notificaciones.
2. Acceder a la sección *Seguridad* en la configuración de la cuenta.
3. Comprobar que la verificación en dos pasos esté habilitada. Si no lo está, seguir las instrucciones para activarla.
4. Buscar la opción *Contraseñas de aplicaciones* y hacer clic en ella.

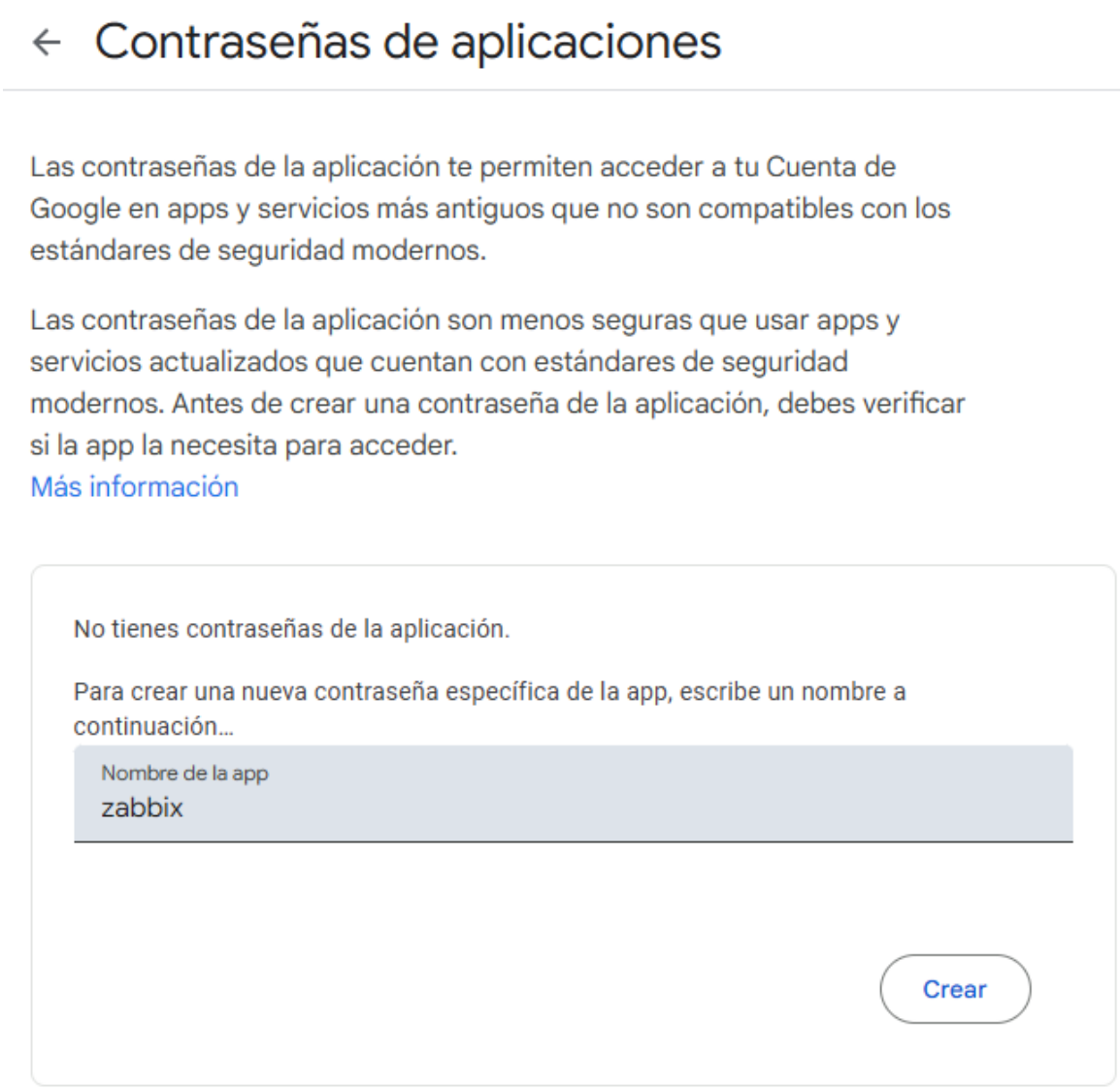
Figura 269. Acceso a la sección de contraseñas de aplicaciones en la cuenta de Google.



Nota. Vista del panel de Seguridad donde, tras habilitar la verificación en dos pasos, se habilita la opción de generar contraseñas de aplicaciones necesarias para que Zabbix pueda autenticar el envío de correos sin usar la clave principal. Elaboración propia.

5. Luego se pide seleccionar un nombre para la aplicación a la cual se le va a generar la contraseña, por ejemplo: Zabbix.

Figura 270. Selección del nombre de la aplicación antes de generar la contraseña.



Nota. Elección del identificador (por ejemplo "Zabbix") para asociar la contraseña de aplicación y facilitar su futura gestión o revocación sin afectar otras integraciones. Elaboración propia.

6. Finalmente, se genera la contraseña de aplicación que se utilizará en lugar de la contraseña habitual de la cuenta de Gmail para la aplicación seleccionada.

Figura 271. Contraseña de aplicación generada para integración con Zabbix.

Nota. Visualización única de la clave de 16 caracteres que debe copiarse y luego usarse en el medio de correo de Zabbix; no es recuperable posteriormente, solo revocable. Elaboración propia.

12.3. Sistema físico

Figura 272. Fotografía del sistema físico implementado.

Nota. Imagen que muestra la disposición física de las computadoras utilizadas en la implementación del sistema. Elaboración propia.

CAPÍTULO 13

Glosario

Alerta. Notificación generada cuando se cumple una condición de seguridad o una regla predefinida.

Apache HTTP *server*. Servidor HTTP de código abierto para publicar aplicaciones y sitios web, con soporte de módulos y virtual hosts.

Archivo *hosts*. Archivo local que mapea nombres de dominio a direcciones IP y tiene precedencia sobre consultas DNS.

***Blocklist*.** Conjunto de dominios o direcciones catalogados como no deseados para denegarlos automáticamente.

***Bogon networks*.** Prefijos IP no asignados o reservados que no deberían aparecer en Internet público.

***Bridged*.** Modo de red de máquina virtual que la conecta como un *host* más en la red física mediante puente.

***Dynamic host configuration protocol* (DHCP).** Servicio de configuración dinámica que asigna direcciones IP y otros parámetros de red a los clientes.

***Demilitarized zone* (DMZ).** Subred expuesta controladamente entre la red interna y el exterior para publicar servicios sin acceso directo a la red privada.

***Domain name system* (DNS).** Sistema de nombres de dominio que resuelve nombres simbólicos a direcciones IP.

DNS *sinkhole*. Resolutor que bloquea o responde con direcciones no enrutables a dominios catalogados como maliciosos o no deseados.

***Firewall*.** Sistema de control de tráfico de red que aplica políticas para permitir, denegar o registrar conexiones según criterios definidos.

FreeBSD Sistema operativo código abierto utilizado como base en diversas soluciones de red.

***Faster than light* (FTL).** Motor de Pi-hole que integra resolutor DNS, registro de consultas, análisis y caché.

Gateway. Dispositivo o dirección que enruta tráfico desde una red local hacia otras redes.

***GUID partition table* (GPT).** Esquema moderno de particionado de discos, común en sistemas con UEFI.

***Grand unified bootloader* (GRUB).** Gestor de arranque que carga sistemas operativos y kernels.

***Greenbone security assistant* (GSA).** Interfaz web para gestionar escaneos y resultados de OpenVAS/GVM.

***Greenbone vulnerability management* (GVM).** Conjunto de servicios para gestión de escaneos y pruebas de vulnerabilidad.

***Hypertext transfer protocol* (HTTP).** Protocolo de aplicación para la transferencia de documentos hipermedia.

Hypervisor. Software que crea y ejecuta máquinas virtuales abstrayendo el hardware físico.

***Internet control message protocol* (ICMP).** Protocolo de red para mensajería y diagnóstico.

***Intrusion detection system* (IDS).** Sistema que monitorea la actividad para identificar intrusiones y generar alertas.

Interfaz de red. Entidad lógica o física que permite a un equipo enviar y recibir tráfico en una red.

IP estática. Dirección IP configurada manualmente y que no cambia sin intervención.

***Intrusion prevention system* (IPS).** Sistema que, además de detectar, bloquea o mitiga automáticamente actividad maliciosa.

Iptables. Infraestructura de filtrado de paquetes en Linux para definir tablas y cadenas de reglas.

Kali Linux. Distribución Linux orientada a pruebas de penetración y seguridad ofensiva.

***Local area network* (LAN).** Red local de alcance limitado que comparte el mismo dominio de broadcast.

Loopback. Dirección de retorno local (IPv4 127.0.0.0/8; IPv6 ::1) usada para pruebas y servicios locales.

***Media access control* (MAC).** Identificador único de capa de enlace (IEEE 802) asignado a una interfaz de red.

MD5 *signature hash*. Huella criptográfica de 128 bits generada por el algoritmo MD5, usada para verificación de integridad.

Metasploitable 3. Máquina virtual intencionalmente vulnerable diseñada para pruebas y entrenamiento.

***Monitoring operations center* (MOC).** Función dedicada a observabilidad y visualización operacional.

Monitoreo Observación continua de métricas y eventos para evaluar estado y rendimiento.

***Network address translation* (NAT).** Traducción de direcciones y puertos entre dominios de red.

***Network time protocol* (NTP).** Protocolo para sincronización horaria entre sistemas.

Oinkcode. Token personal usado para descargar conjuntos de reglas oficiales de Snort.

OpenVAS. Escáner de vulnerabilidades de código abierto que ejecuta pruebas autenticadas y no autenticadas.

***Payload*.** Parte útil de un paquete o flujo que contiene los datos transmitidos, excluyendo cabeceras.

PfSense. Distribución basada en FreeBSD para funciones de *firewall* y *router* con administración web.

Pi-hole. DNS *sinkhole* para bloquear dominios no deseados a nivel de resolución.

***Port forwarding*.** Regla de NAT que redirige conexiones entrantes hacia un *host* y puerto internos.

PostgreSQL. Sistema de gestión de bases de datos relacional de código abierto.

***Query logging*.** Registro de consultas DNS y respuestas para auditoría y análisis.

RFC 1918. Especificación que define los rangos de direcciones IPv4 privadas (10/8, 172.16/12, 192.168/16).

Segmentación de red. División de una red en subredes lógicas o físicas para aislar, limitar alcance y aplicar políticas específicas.

***Signature*.** Patrón formalizado que describe una amenaza conocida para su detección automática.

Snort. Motor IDS/IPS de código abierto que inspecciona tráfico en tiempo real mediante reglas y decodificadores de protocolo.

***Security operations center* (SOC).** Equipo o función dedicada a vigilancia y respuesta ante incidentes.

***Transmission control protocol* (TCP).** Protocolo de transporte orientado a conexión y fiable.

Ubuntu *desktop/server*. Distribución Linux basada en Debian, disponible en ediciones para escritorio y servidor.

***User datagram protocol* (UDP).** Protocolo de transporte no orientado y sin garantía de entrega.

***Upstream* DNS.** Servidor resolutor al que se reenvían consultas cuando no pueden resolverse localmente.

***Virtual network editor*.** Herramienta de VMware para crear y configurar redes virtuales VMnet.

***Virtual local area network* (VLAN).** Red local virtual que separa dominios de broadcast sobre la misma infraestructura física.

VLAN *tagging*. Inserción del identificador 802.1Q en tramas Ethernet para asignarlas a una VLAN.

VMnet. Redes/Interfaces virtuales numeradas de VMware utilizadas para segmentación.

VMware *workstation pro* 17. Hypervisor de escritorio para creación y administración de máquinas virtuales.

***Wide area network* (WAN).** Red de alcance geográfico amplio que interconecta redes locales.

***Whitelist*.** Lista de elementos explícitamente permitidos por una política de seguridad.

Zabbix. Plataforma de monitoreo para recopilar métricas, generar alertas y visualizar datos en tiempo real.

Zabbix *agent*. Agente que recopila y envía métricas al servidor Zabbix, en modo pasivo o activo.

***Zero trust*.** Modelo que elimina confianza implícita y valida explícitamente cada acceso a recursos.

***Zettabyte file system* (ZFS).** Sistema de archivos con verificación de integridad, volúmenes lógicos y *snapshots*.