

UNIVERSIDAD DE ALMERÍA

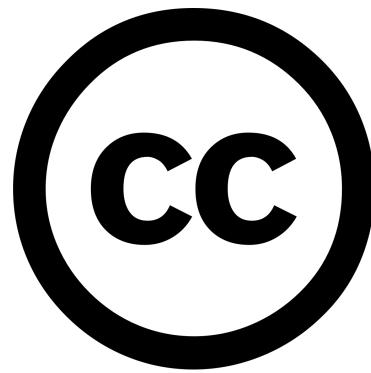
COMPLEMENTO TRABAJO FIN DE GRADO

Utilización de un NIDS para redes SOHO (R-Snort)

Autor: Petrovics, Deian Orlando

Tutor: Gómez López, Julio

Cotutor: Padilla Soriano, Nicolás



Este trabajo está bajo una licencia Creative Commons
Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Índice general

Abreviaturas	5
Introducción	7
1 Motivación	8
2 Objetivos	9
3 Fases de la realización y cronograma	10
3.1 Fase 1: Inspiración inicial y propuesta del proyecto (junio-agosto 2024)	10
3.2 Fase 2: Prototipado inicial y pruebas preliminares (septiembre 2024)	10
3.3 Fase 3: Estudio y selección de componentes (enero-febrero 2025)	11
3.4 Fase 4: Desarrollo del sistema R-Snort (enero 2025)	11
3.5 Fase 5: Pruebas de validación y rendimiento (marzo-abril 2025)	11
3.6 Cronograma general	12
4 Estructura y metodología	13
5 Sistemas de Detección de Intrusos	15
5.1 Los sistemas de detección de intrusos	15
5.1.1 Definición y propósito de los IDS	15
5.1.2 Clasificación de los IDS	15
5.1.3 Arquitectura y componentes de un IDS	16
5.1.4 Funcionalidades	16
5.1.5 Ventajas y limitaciones de los IDS	16
5.1.6 Contexto actual	17
5.2 Snort	17
5.2.1 Características principales	17
5.2.2 Arquitectura de Snort	18
5.2.3 Reglas y actualizaciones	18
5.2.4 Personalización y extensibilidad	19
5.2.5 Limitaciones y consideraciones	19
5.3 Necesidades de seguridad en pequeñas redes (SOHO)	20
5.3.1 Amenazas y vulnerabilidades comunes	20
5.3.2 Retos específicos	20
5.3.3 Soluciones adecuadas para redes SOHO	21
5.4 Beneficios y recomendaciones de los sistemas de detección en redes reducidas	22
5.5 Complementos y plugins para Snort	22

6 Utilización de Snort en redes SOHO	24
6.1 Introducción	24
6.2 Especificaciones, características y requisitos	24
6.2.1 Especificaciones de Snort 3	24
6.2.2 Requisitos del sistema	24
6.3 Entorno de trabajo	25
6.3.1 Esquema de red	25
6.3.2 Hardware (Raspberry Pi 5)	26
6.3.3 Software (Snort y sus complementos)	27
6.4 Instalación y personalización de complementos	29
6.4.1 Instalación de Snort V3	29
6.4.2 Instalación de reglas y plugins	44
6.5 Generación de un script para la instalación automática	53
6.5.1 Esquema de red de monitorización con R-SNORT	53
6.5.2 Primera fase del script automático	54
6.5.3 Transición a paquete .deb	57
7 Caso práctico: utilización de R-Snort	60
7.1 Entorno de trabajo	60
7.2 Instalación	61
7.3 Utilización y pruebas	66
7.3.1 Pruebas de rendimiento	66
7.3.2 Pruebas de comportamiento	73
7.4 Resultados	76
7.5 Resumen	76
8 Resultados y discusión	77
8.1 Discusión crítica	77
8.2 Limitaciones del proyecto	78
8.3 Conclusión de los resultados	78
Conclusiones	79
Trabajo futuro	80
A Anexo A: Repositorio de R-SNORT	86
B Anexo B: Script de instalación	87

Abreviaturas

- **APT:** (Advanced Persistent Threat). Amenaza Avanzada Persistente. Suele involucrar ataques dirigidos y prolongados contra objetivos concretos.
- **ClamAV:** Antivirus de código abierto que analiza y detecta archivos maliciosos o infecciones. Se integra como complemento en el proyecto.
- **CPU:** (Central Processing Unit). Unidad central de procesamiento, comúnmente conocida como procesador, encargada de ejecutar instrucciones en un sistema.
- **CSV:** (Comma-Separated Values). Formato de archivo de texto plano que representa datos tabulares separados por comas o puntos y coma.
- **Debian Package (*.deb):** Formato estándar de paquetes en sistemas GNU/Linux basados en Debian/Ubuntu. Facilita la instalación y gestión de software.
- **DoS:** (Denial of Service). Ataque que busca interrumpir el funcionamiento normal de un sistema o red.
- **FTP:** (File Transfer Protocol). Protocolo para la transferencia de archivos en redes IP.
- **GNU/Linux:** Sistema operativo de software libre en el que se basan distribuciones como Ubuntu, Debian, CentOS, etc.
- **HIDS:** (Host Intrusion Detection System). Sistema de Detección de Intrusiones basado en Host, centrado en vigilar el comportamiento interno de un equipo específico.
- **HTTP:** (Hypertext Transfer Protocol). Protocolo de comunicación utilizado en la web para transmitir información entre cliente y servidor.
- **HTTP2:** Segunda versión del protocolo HTTP, optimizada para mayor velocidad y eficiencia.
- **ICMP:** (Internet Control Message Protocol). Protocolo usado para enviar mensajes de error y diagnóstico en redes IP.
- **IDS:** (Intrusion Detection System). Sistema de Detección de Intrusiones (término general). Comprende tanto la detección en host (HIDS) como en red (NIDS).
- **IEC 104:** Protocolo de comunicación utilizado en sistemas de automatización industrial y redes eléctricas.
- **IMAP:** (Internet Message Access Protocol). Protocolo para acceder y gestionar correos electrónicos en servidores remotos.

- **IPS:** (Intrusion Prevention System). Sistema de Prevención de Intrusiones. Además de detectar acciones maliciosas, reacciona automáticamente para bloquearlas.
- **LuaJIT:** Implementación just-in-time (JIT) del lenguaje de scripting Lua, que Snort 3 utiliza para reglas y configuraciones más flexibles.
- **MIME:** (Multipurpose Internet Mail Extensions). Estándar para enviar contenido diverso (como archivos) a través de correos electrónicos.
- **NIDS:** (Network Intrusion Detection System). Sistema de Detección de Intrusiones en Red. Se encarga de monitorear el tráfico que circula por la red en busca de acciones sospechosas o maliciosas.
- **OT:** (Operational Technology). Tecnología usada para controlar procesos físicos en industrias, como sistemas SCADA o PLCs.
- **POP3:** (Post Office Protocol version 3). Protocolo usado para recuperar correos electrónicos desde un servidor.
- **Raspberry Pi (R-Pi):** Máquina de bajo coste y tamaño reducido. Muy popular para proyectos de electrónica, servidores ligeros.
- **R-SNORT:** Adaptación o paquete de Snort diseñado para ejecutarse de forma optimizada en una Raspberry Pi, con funciones específicas para redes SOHO.
- **SIEM:** (Security Information and Event Management). Plataforma que recopila y correlaciona datos de seguridad (logs, alertas, eventos) para proporcionar una visión global y centralizada.
- **SIP:** (Session Initiation Protocol). Protocolo usado principalmente para establecer y controlar sesiones multimedia, como llamadas VoIP.
- **SMB:** (Server Message Block). Protocolo de red para compartir archivos, impresoras y puertos serie entre nodos.
- **Snort:** Herramienta de código abierto usada para la detección de intrusiones en red, muy extendida en el ámbito de la ciberseguridad.
- **SOHO:** (Small Office/Home Office). Redes pequeñas o domésticas, típicas de oficinas y hogares con recursos más limitados que una gran empresa.
- **SSL/TLS:** (Secure Sockets Layer / Transport Layer Security). Protocolos de cifrado que permiten la comunicación segura entre sistemas a través de redes.

Introducción

La dependencia actual del uso de tecnología difiere en gran medida del uso que se le daba hace años. El crecimiento del globalismo, que impulsa la interconexión entre todos nosotros, hace que el conocimiento sobre informática y sus fallos de seguridad se dé a conocer mundialmente, permitiendo a individuos malintencionados aprovechar dichas vulnerabilidades por diversos motivos: económicos, sociales o incluso como competición entre ellos.

Organizaciones de renombre cuentan con los mayores expertos en seguridad informática; sin embargo, las redes domésticas o de pequeñas o medianas empresas no gozan de tal presupuesto, quedando normalmente vulnerables a todo tipo de ataques [1]. Este trabajo propone una solución accesible que permita a entornos más modestos estar protegidos mediante un sistema de detección de intrusos (IDS) basado en herramientas de bajo impacto económico y fácil implementación.

1. Motivación

Desde antes de empezar el grado de Ingeniería Informática me sentía atraído por los entornos sin interfaz gráfica: terminales, comandos... Esto despertó en mí una inclinación hacia la ciberseguridad. Al llegar a la carrera, este gusto por la seguridad informática se ha visto impulsado por asignaturas como Sistemas Operativos, Seguridad Informática, Tecnologías de Acceso a Red, Administración de Redes y Sistemas Operativos, entre otras que también han puesto su grano de arena.

Esto me ha llevado a implementar algunas prácticas en mi propia red doméstica, haciéndome ver lo evidente que es la falta de seguridad y protección de cualquier tipo en los hogares y también en algunas PYMES. Esta realidad me ha hecho tratar de buscar alternativas prácticas y asequibles para menguar esta problemática.

El objetivo principal de este trabajo es llevar la seguridad informática al alcance de quien lo necesite, de forma automatizada y sin la necesidad de un profundo conocimiento técnico para su instalación, haciendo posible que cualquiera pueda defenderse del vasto mundo de Internet. De esta manera nace R-Snort.

2. Objetivos

El objetivo principal de este trabajo es el desarrollo de un sistema automático de instalación de un detector de intrusos personalizado, con los requerimientos generales que puedan tener la amplia mayoría de redes SOHO o PYMES. El nombre del proyecto R-Snort proviene de los dos protagonistas de este sistema: una Raspberry Pi 5 [2] por su bajo coste y eficiencia energética, que será el equipo embebido utilizado —aunque se puede extraer a otras máquinas similares— y Snort [3], el IDS elegido para la tarea de detectar intrusos. Este ha sido seleccionado por ser de código abierto y, en su nueva versión Snort 3, por ser altamente configurable gracias al uso de archivos en formato LUA [4], lo cual lo hace modular y facilita su personalización.

Los objetivos específicos serán:

- **Integración de herramientas complementarias:** Se van a incorporar y configurar plugins, preprocesadores, filtrado de contenido y sistemas antivirus para cubrir las necesidades de seguridad específicas de pequeñas redes.
- **Automatización y portabilidad:** Con el objetivo de hacer un software accesible a todo el mundo, no solo en lo económico, sino también en lo práctico, R-Snort estará disponible para su despliegue automático, teniendo en cuenta las especificaciones del equipo en el que se llevará a cabo la instalación. De esta forma, el usuario no necesitará conocimientos profundos de GNU/Linux ni de redes.
- **Evaluación de efectividad y rendimiento:** Tras el correcto desarrollo e instalación de R-Snort, se realizarán pruebas tanto de rendimiento como de ataques simulados, para comprobar su funcionamiento en condiciones reales, como es una red doméstica.

Este trabajo se enfoca en proporcionar una solución económica, sencilla y eficiente que refuerce la seguridad de redes a menor escala.

3. Fases de la realización y cronograma

Para el desarrollo de R-Snort se ha seguido una estrategia iterativa. Se pretende adaptar todos los recursos disponibles junto con los conocimientos adquiridos a lo largo del grado, entremezclándose con los requerimientos universales que pueda tener una red SOHO. Comenzando con ideas sencillas, R-Snort fue madurando hacia el sistema robusto de detección de intrusos que es ahora.

A continuación, se van a describir las distintas fases del proyecto, desde el concepto inicial hasta el producto actual.

3.1. Fase 1: Inspiración inicial y propuesta del proyecto (junio-agosto 2024)

Durante el verano de 2024 comencé un proyecto personal ajeno a lo académico. Este proyecto plantó, de manera informal, la semilla de lo que sería el proyecto final. Para concretar, se puso en marcha un servidor personal a mi servicio y al de algunos conocidos y amigos, sin ningún objetivo más allá del ocio y entretenimiento. A lo largo del tiempo, iba agregando prácticas de seguridad informática para proteger dicho servidor de ataques indeseables, ya que, al estar conectado a la red de manera pública, no fueron pocos los que trataron de vulnerarlo y acceder a nuestra información privada.

Estas prácticas puestas en marcha, aunque inicialmente rudimentarias, despertaron aún más el interés en explorar mecanismos y formas más eficaces, aplicables a todo tipo de redes.

En septiembre de ese mismo año, tras disipar la abstracción acerca del proyecto final, la idea era clara: perfilar y diseñar un sistema de detección de intrusos basado en Snort, para una red PYME o SOHO, empleando una Raspberry Pi.

3.2. Fase 2: Prototipado inicial y pruebas preliminares (septiembre 2024)

El primer esbozo de R-Snort consistió en un equipo de segunda mano con Ubuntu Server [5], donde se instaló la versión por defecto de Snort que viene en los repositorios oficiales de Ubuntu. Apenas se activaron algunas reglas para comprobar que funcionaba correctamente. Posteriormente, se agregó una base de datos PostgreSQL relacional [6] desplegada en Docker. Esta versión preliminar, si bien contaba con falta de optimización y escalabilidad, ya era capaz

de recoger algunos eventos generados por el IDS.

3.3. Fase 3: Estudio y selección de componentes (enero–febrero 2025)

Teniendo en cuenta los requisitos principales, que eran la **eficiencia energética** y de consumo, se analizaron los componentes necesarios para un *entorno de producción realista*. Un gran número de *plugins*, preprocesadores y distintas configuraciones disponibles han sido tenidas en cuenta para posteriormente evaluar las más adecuadas para entornos **SOHO**.

Durante esta fase, y antes de poner en marcha las configuraciones seleccionadas, se descartaron algunas por **incompatibilidades** con la versión de *Snort*. Herramientas como *Barnyard2* [7] y el formato *Unified2* ya no eran compatibles. También se decidió abandonar la versión **3.7 de Snort** debido a errores de *NUMA (Non-Uniform Memory Access)* que dificultaban su instalación automática. Tras ello, se propuso el uso de la versión **Snort 3.1.84.0** [8], más estable para entornos embebidos.

3.4. Fase 4: Desarrollo del sistema R-Snort (enero 2025)

Esta fase supuso la consolidación del proyecto como producto técnico funcional. Se diseñó un sistema modular que permitía instalar Snort 3 de forma completamente automática sobre una *Raspberry Pi*, configurando tanto sus dependencias como los módulos necesarios para su operación además de tener en cuenta la memoria RAM del sistema para la instalación.

3.5. Fase 5: Pruebas de validación y rendimiento (marzo-abril 2025)

Con **R-Snort** ya en funcionamiento, había que probarlo bajo distintos tipos de ataques simulados en una *red doméstica* y ver cómo responde a nivel de **eficiencia**. Se llevaron a cabo varias pruebas y el rendimiento fue evaluado en distintos escenarios, recopilando información sobre el uso de la **CPU, red y memoria**, comparándola con *Snort* activado y desactivado, para posteriormente generar **gráficas ilustrativas** que ayuden al lector a comprender de mejor manera el impacto sobre el equipo que supone *Snort* [9].

3.6. Cronograma general

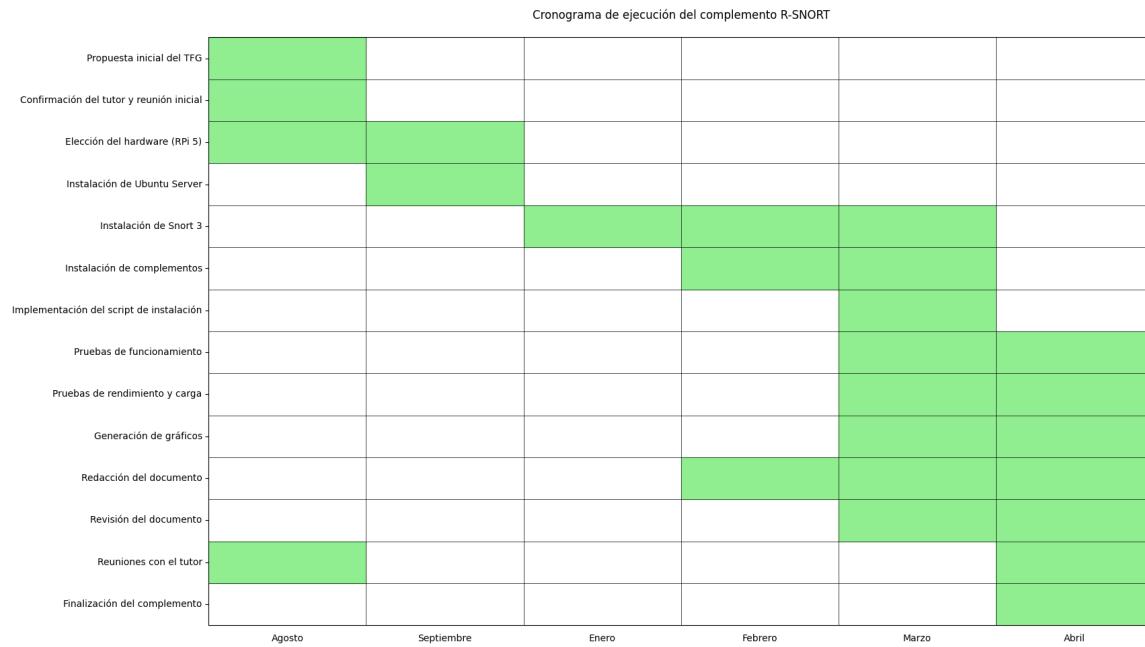


Figura 3.1: Cronograma de las tareas.

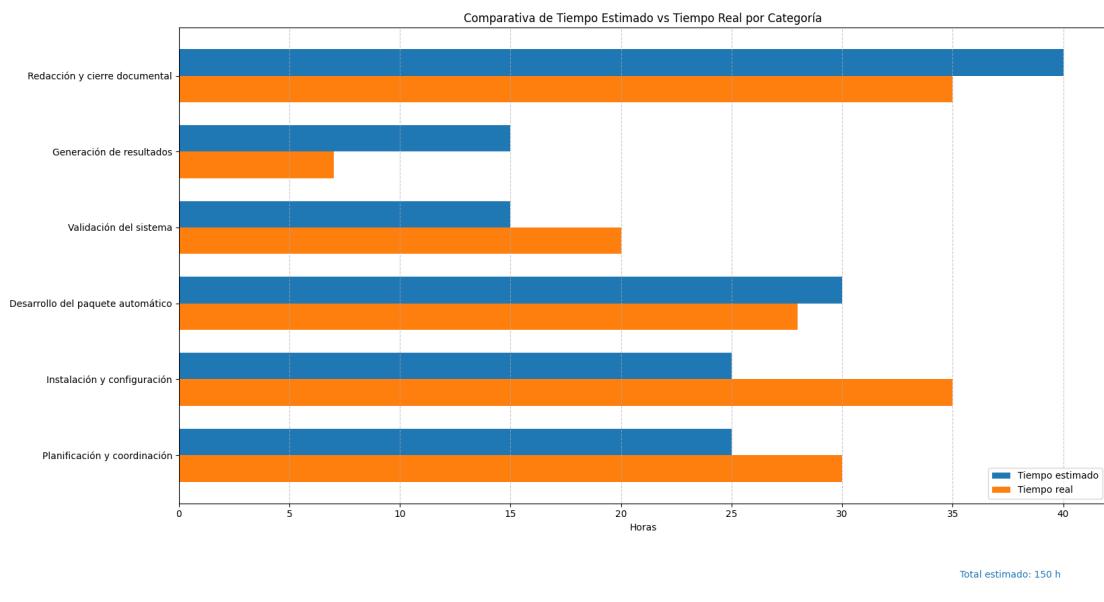


Figura 3.2: Comparación entre el tiempo estimado y el real.

4. Estructura y metodología

El enfoque seguido ha sido principalmente **empírico**, centrado en la construcción de un sistema funcional y su posterior optimización. La metodología puede resumirse en lo siguiente:

- **Iteración y mejora continua:** En lugar de definir de antemano todos los componentes del sistema, se optó por una construcción *modular* e *incremental*. Esto ha permitido corregir errores tempranos, redefinir objetivos parciales y mejorar la **robustez** del sistema progresivamente.
- **Análisis de compatibilidad y adecuación:** Dado que el entorno de ejecución es una *Raspberry Pi* con arquitectura *ARM* y los sistemas en los que se planea el despliegue de este trabajo serán plataformas similares, fue necesario evaluar cuidadosamente la compatibilidad de las versiones de *Snort* y sus dependencias. Este análisis condujo, por ejemplo, a descartar **Snort 3.7** por incompatibilidades con *NUMA* y optar por **Snort 3.1.84** como versión estable.
- **Selección de componentes según el escenario:** Se priorizaron configuraciones, módulos y *plugins* relevantes para proteger redes pequeñas, como las de una oficina doméstica (**SOHO**), descartando funciones innecesarias o incompatibles con *Snort 3*. La selección se basó tanto en documentación oficial como en pruebas reales de rendimiento.
- **Automatización y accesibilidad:** Una parte importante del trabajo se centró en la creación de un sistema de instalación automática, lo más transparente posible para el usuario final. La *modularización* de scripts y la documentación clara fueron el objetivo, pues se espera que el usuario no tenga un conocimiento demasiado técnico. Por ello, se ha tratado de automatizar cada proceso.
- **Evaluación empírica del rendimiento:** Finalmente, el sistema fue sometido a pruebas prácticas, midiendo el consumo de recursos con y sin *Snort* activo. Los resultados fueron procesados mediante herramientas estadísticas y visualizados en forma de **gráficas generadas automáticamente**.

Estructura del documento

El documento se ha dividido en distintos puntos que pretenden ayudar al lector a seguir sin complicación el flujo de trabajo:

- **Capítulos 1 al 4:** Introducen la motivación personal y los objetivos generales, junto con las fases de realización y la estructura del presente documento.
- **Capítulo 5:** Se exponen las principales funcionalidades, ventajas y explicación del comportamiento de los *IDS*; posteriormente, se hace una introducción didáctica a *Snort* como *IDS*.
- **Capítulo 6:** Este punto describe la utilización de un *NIDS* en las redes objetivo. Se muestran las especificaciones de **Snort 3**, los requisitos de su instalación con sus *preprocesadores* y configuración personalizada para **SOHO** [10], junto con la descripción de su transformación a *script* automática para, finalmente, convertirse en un paquete automático *.deb*, que en esencia es el corazón de instalación de **R-Snort**.
- **Capítulo 7:** Casos prácticos de **R-Snort** funcionando en un entorno real como es una *red doméstica*, puesta a prueba y evaluación del rendimiento.
- **Capítulo 8:** Finalizando el proyecto, se discuten los resultados obtenidos durante este trabajo.
- **Conclusiones:** Recoge las pruebas prácticas realizadas sobre el sistema, así como las gráficas de rendimiento obtenidas.
- Además se incluyen un resumen de resultados, conclusiones y anexos con información complementaria.

Esta estructura ha sido cuidadosamente diseñada para poder replicarse y adaptarse a otros entornos, contribuyendo así a la difusión de herramientas de seguridad accesibles.

5. Sistemas de Detección de Intrusos

5.1. Los sistemas de detección de intrusos

El objetivo de esta sección es mostrar una visión general de los Sistemas de Detección de Intrusos (IDS), su importancia en la seguridad informática y cómo encajan en el panorama actual de ciberseguridad.

5.1.1. Definición y propósito de los IDS

Un sistema de **Detección de Intrusos (IDS)** es una herramienta de *ciberseguridad* que monitorea una red en busca de actividades anómalas o algún tipo de violación de políticas de seguridad, tanto del interior de la red como provenientes del exterior [11].

Su principal función es **detectar accesos no autorizados y comportamientos fuera de lo común** que, de alguna manera, puedan comprometer la **confidencialidad, integridad o disponibilidad** de los equipos.

Mientras que un **IDS** se encarga de detectar y alertar sobre posibles intrusiones, un **Sistema de Prevención de Intrusos (IPS)** no solo detecta, sino que también toma medidas para **bloquear o prevenir dichas intrusiones en tiempo real** [12].

5.1.2. Clasificación de los IDS

- **Según su ubicación:**

- **IDS basados en host (HIDS):** Monitorizan y analizan la actividad interna de un sistema individual, revisando *logs*, integridad de archivos y procesos en ejecución.
- **IDS basados en red (NIDS):** Supervisan el tráfico de red en busca de actividades sospechosas, analizando los *paquetes* que circulan por la red para detectar patrones de ataque.

- **Según el tipo de datos que analizan:**

- **Basado en anomalías:** Buscan comportamientos fuera de lo común dentro de un sistema o red que puedan resultar sospechosos o malintencionados [13].
- **Basado en firmas:** Reconocen patrones previamente identificados, mediante bases de datos u otros métodos, para detectar comportamientos que coinciden con ciberataques conocidos o fugas de datos [14].

5.1.3. Arquitectura y componentes de un IDS

Un **IDS** generalmente cuenta con las siguientes partes:

- **Sensores:** Estos son los encargados de capturar los datos provenientes de la red o del propio anfitrión.
- **Analizadores:** Se encargan de analizar y examinar los patrones en los paquetes recibidos en busca de coincidencias con comportamientos anómalos o firmas conocidas.
- **Bases de datos de firmas:** Almacenan patrones de ataques conocidos para la detección basada en firmas.
- **Interfaz de gestión:** Permite a los administradores configurar el sistema, revisar alertas y generar informes.

5.1.4. Funcionalidades

El sistema incorpora una serie de funcionalidades principalmente orientadas a la supervisión continua, la respuesta ante incidentes y la integración con otras herramientas de ciberseguridad.

- **Monitorización en tiempo real:** Está siempre activo con el objetivo de captar cualquier tipo de intrusión de manera inmediata y poder actuar con rapidez.
- **Generación de alertas:** Por cada tipo de intrusión o sospecha, se genera un determinado tipo de alerta con descripciones relevantes para que de un rápido vistazo el administrador encargado pueda tomar acción rápidamente.
- **Registro de eventos:** Se documentan los incidentes ocurridos para posterior análisis forense y cumplimiento normativo, facilitando el trabajo del especialista que se encargue de un determinado ataque.
- **Integración con otros sistemas de seguridad:** Trabaja en conjunto con *cortafuegos*, sistemas de gestión de eventos (**SIEM**), entre otras herramientas, con el objetivo de mejorar la defensa continuamente y estar preparados para amenazas posteriores.

5.1.5. Ventajas y limitaciones de los IDS

Como todo sistema de seguridad, los IDS tienen puntos fuertes y aspectos mejorables. A continuación se resumen sus principales ventajas y limitaciones:

Ventajas

- Permiten detectar amenazas en etapas tempranas, lo que ayuda a responder antes de que causen daños.
- Mejoran la capacidad de reacción ante incidentes de seguridad.
- Facilitan el cumplimiento de normativas y ayudan en el análisis posterior de eventos.

Limitaciones

- Pueden generar falsos positivos o no detectar ciertas amenazas.
- Requieren actualizaciones constantes para seguir siendo efectivos, y algunas opciones automáticas son de pago.
- Suponen un cierto consumo de recursos, aunque esto suele ser aceptable si se prioriza la seguridad.

5.1.6. Contexto actual

Con el incremento de amenazas avanzadas y persistentes (**APT**) [16], los **IDS** deben evolucionar incorporando *inteligencia artificial* y *aprendizaje automático* para mejorar la detección. Además, la integración con sistemas de gestión de información y eventos de seguridad (**SIEM**) es de especial interés para una respuesta coordinada y más eficaz.

Ataques como el gusano *SQL Slammer* en 2003 [17] fueron detectados por *IDS basados en firmas*, mientras que amenazas más sofisticadas, como *Stuxnet* en 2010 [18] [19], requirieron análisis más avanzados para su identificación. Comparaciones entre **HIDS** y **NIDS** en entornos específicos muestran que, aunque los HIDS ofrecen una visión detallada del host, los NIDS proporcionan una perspectiva más amplia del tráfico de red.

5.2. Snort

En esta sección se pretende profundizar en **Snort**[20] como uno de los **NIDS** más populares y versátiles, destacando sus características, funcionamiento y relevancia en entornos de seguridad.

Desarrollado por Martin Roesch en 1998, Snort ha evolucionado hasta convertirse en un estándar en sistemas de detección de intrusiones de código abierto junto con otros detectores de intrusos como Suricata [22]. En 2013, Cisco Systems adquirió Sourcefire, la empresa detrás de Snort, integrándolo en su portafolio de soluciones de seguridad. *Snort* es utilizado en una amplia variedad de escenarios, desde pequeñas oficinas hasta grandes corporaciones y entornos educativos, gracias a su capacidad de adaptación. Puede identificar actividades maliciosas como inyecciones *SQL*, escaneos de puertos y ataques de denegación de servicio (**DoS**), proporcionando alertas en tiempo real para una respuesta rápida. Tiene la capacidad de detectar casi cualquier tipo de actividad sospechosa gracias a las reglas creadas por la comunidad o por el propio administrador del sistema.

5.2.1. Características principales

Snort se distingue por algunas funciones que le permiten adaptarse a distintos escenarios y necesidades. Entre las más destacadas se encuentran:

- **Motor de detección basado en firmas:** utiliza reglas predefinidas para detectar patrones de ataque conocidos.
- **Modos de operación:**

- **Sniffer:** Captura y muestra los paquetes de red en tiempo real.
- **Packet Logger:** Guarda los paquetes para analizarlos más adelante.
- **Network Intrusion Detection (NIDS):** Examina el tráfico de red y lanza alertas si detecta actividad sospechosa.
- **Network Intrusion Prevention (IPS):** Además de detectar tráfico malicioso, es capaz de bloquear o rechazar automáticamente paquetes peligrosos en tiempo real para prevenir intrusiones.

5.2.2. Arquitectura de Snort

El funcionamiento de Snort se basa en una arquitectura modular, donde cada componente cumple un papel específico dentro del proceso de detección. Sus principales elementos son:

- **Preprocesadores:** Adaptan el tráfico para su análisis, resolviendo fragmentaciones y otros detalles técnicos.
- **Motor de detección:** Compara el tráfico con las reglas definidas para identificar amenazas.
- **Plugins de salida:** Se encargan de registrar o enviar las alertas generadas, ya sea a archivos, bases de datos o servicios como syslog.

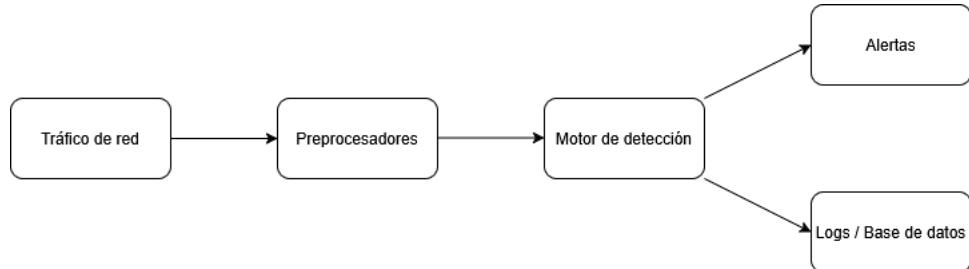


Figura 5.1: Arquitectura del funcionamiento de Snort.

5.2.3. Reglas y actualizaciones

Estructura de una regla de Snort

Una regla típica de **Snort** se compone de dos partes principales: el **encabezado** y las **opciones** [21]. Por un lado, el encabezado define la acción a tomar (por ejemplo, alertar), el protocolo, las direcciones IP de origen y destino, y los puertos. Las opciones especifican condiciones adicionales, como patrones de contenido a buscar, mensajes de alerta y otros parámetros.

```
alert tcp any any -> 192.168.1.0/24 80 (msg:"Possible acceso HTTP";
content:"GET"; sid:1000001; rev:1;)
```

Esta regla genera una alerta si se detecta una conexión TCP desde cualquier origen hacia la red 192.168.1.0/24 en el puerto 80, y si el contenido del paquete contiene la cadena "GET", típica de una solicitud HTTP. El campo `msg` define el mensaje de alerta, mientras que `sid` y `rev` son identificadores únicos para gestión y versiones.

Tipos de reglas

Snort utiliza distintos conjuntos de reglas que varían en disponibilidad, frecuencia de actualización y nivel de protección. Los tipos principales son:

- **Community Rules:** Reglas creadas y mantenidas por la comunidad de usuarios de *Snort*, disponibles de forma gratuita bajo la licencia *GPLv2*.
- **Registered Rules:** Proporcionadas por *Cisco Talos*, disponibles para usuarios registrados, aunque con un retraso de 30 días respecto al lanzamiento para suscriptores.
- **Subscriber Rules:** Reglas actualizadas en tiempo real, pensadas para usuarios de pago. Ofrecen protección frente a amenazas emergentes sin retrasos [23] (*no usadas en este proyecto*).

Gestión de reglas

Herramientas como *PulledPork* y *Oinkmaster* facilitan la descarga, actualización y gestión de las reglas de *Snort*, automatizando procesos y asegurando que el sistema esté protegido contra las últimas amenazas.

5.2.4. Personalización y extensibilidad

Creación de reglas personalizadas

Los administradores pueden desarrollar reglas específicas adaptadas a las necesidades particulares de su entorno, permitiendo una detección más precisa de amenazas relevantes para su organización.

Uso de variables y listas

Snort permite definir variables para representar direcciones IP, rangos de puertos y otras configuraciones, facilitando la gestión y actualización de las reglas.

Integración con otros sistemas

Snort puede integrarse con herramientas como *Barnyard2*, *Snorby* y *Sguil* para mejorar la gestión de alertas, análisis de eventos y generación de informes.

5.2.5. Limitaciones y consideraciones

Rendimiento en redes de alta velocidad

En entornos con tráfico de red intenso, es una necesidad prioritaria contar con **hardware adecuado** y una **configuración óptima** para asegurar que *Snort* funcione correctamente sin afectar el rendimiento del sistema.

Gestión de falsos positivos

La tarea de ajustar y afinar las reglas para minimizar las alertas falsas es prioritaria, evitando así una sobrecarga de información y permitiendo a los administradores centrarse en amenazas reales.

5.3. Necesidades de seguridad en pequeñas redes (SOHO)

Se pretende analizar las necesidades específicas de seguridad en redes pequeñas, como oficinas domésticas o pequeñas empresas (**SOHO**), y cómo soluciones como *Snort* pueden ser implementadas eficientemente.

Las redes **SOHO** suelen estar compuestas por un número reducido de dispositivos y recursos limitados, y a menudo carecen de personal especializado en *TI*. A pesar de su tamaño, estas redes juegan un papel importante para la economía y requieren medidas de seguridad adecuadas [24].

5.3.1. Amenazas y vulnerabilidades comunes

Toda red, por pequeña que sea, está expuesta a distintos tipos de amenazas. Algunas vienen del exterior, otras desde dentro, y muchas están relacionadas con dispositivos mal configurados. A continuación se describen las más relevantes:

Amenazas externas

Incluyen el uso de *malware*, ataques de *phishing* y fuerza bruta contra servicios accesibles desde Internet.

Amenazas internas

Pueden deberse a un mal uso de los recursos, conexiones inseguras desde dispositivos personales o la falta de normas claras sobre seguridad en la red.

Dispositivos IoT

El crecimiento del número de dispositivos conectados (como cámaras, sensores o dispositivos *wearables*) supone un riesgo añadido. Entre los problemas más frecuentes se encuentran:

- Uso de credenciales por defecto que no pueden cambiarse.
- Comunicación sin cifrado, como en versiones inseguras de MQTT.
- Falta de actualizaciones o mantenimiento del firmware [25].

Una solución práctica es implementar segmentación de red mediante VLANs para aislar estos dispositivos, combinado con reglas específicas en *Snort 3* para monitorizar tráfico en puertos comunes IoT (ej: 1883 para MQTT, 5683 para CoAP).

5.3.2. Retos específicos

Limitaciones de recursos

Al tratarse de una solución orientada a entornos pequeños, optimizar el consumo de recursos pertenece a la lista de objetivos principales. Para ello, se aplican medidas como:

- Uso de hardware eficiente como la *Raspberry Pi 5*, con un consumo inferior a 10W.
- Empleo de reglas gratuitas, como las de *Emerging Threats Open*.

- Configuraciones predefinidas en *Snort 3* enfocadas a detectar amenazas críticas.

Además, la versión 3 de Snort introduce un motor multi-hilo que mejora el rendimiento en sistemas con pocos recursos, gracias a un procesamiento más selectivo del tráfico [26].

Conectividad constante

Estrategias de mitigación:

- Inspección de tráfico VPN (*WireGuard/OpenVPN*) mediante preprocesadores de *Snort*.
- Reglas específicas para detección de ataques a servicios RDP/SSH.
- Integración con sistemas de autenticación multifactor para acceso remoto.

5.3.3. Soluciones adecuadas para redes SOHO

En entornos pequeños o domésticos, la seguridad debe abordarse por capas. A continuación se describe lo que debería incluir una posible configuración mínima:

- Cortafuegos basado en `iptables` con políticas restrictivas por defecto.
- Integración de un *antivirus de red*, como *ClamAV*, para inspeccionar tráfico sospechoso.
- Mecanismos de actualización automática mediante repositorios firmados digitalmente.

Las pequeñas oficinas o entornos domésticos SOHO suelen contar con presupuestos limitados y pocos recursos técnicos. En estos casos, el uso de soluciones de código abierto es una alternativa muy viable por su bajo coste y flexibilidad.

Herramientas de código abierto

Optar por software libre tiene múltiples beneficios:

- **Comunidades activas** que mantienen más de 500 reglas actualizadas mensualmente en herramientas como *Snort*.
- **Compatibilidad multiplataforma**, lo que facilita su uso en distintos dispositivos y sistemas operativos.
- **Auditoría del código fuente**, que permite identificar vulnerabilidades o errores de seguridad antes de que sean explotados.

Dispositivos de bajo coste

Una solución económica y funcional puede construirse sobre una *Raspberry Pi 5* con:

- Un sistema operativo seguro, como *Ubuntu Server LTS*, *Raspbian OS* o *Debian*.
- Supervisión de parámetros físicos: temperatura, voltaje y consumo energético.

5.4. Beneficios y recomendaciones de los sistemas de detección en redes reducidas

Implementar un sistema de detección en redes pequeñas ofrece múltiples ventajas:

- **Identificación preventiva de riesgos:** La monitorización continuo permite detectar comportamientos sospechosos a tiempo, mejorando la capacidad de respuesta.
- **Análisis del tráfico de red:** Facilita auditorías internas al observar cómo se comunican los dispositivos entre sí, ayudando a detectar puntos débiles.
- **Cumplimiento de normativas de seguridad:** Aporta trazabilidad y evidencia para cumplir con protocolos básicos de protección de datos.

Por otro lado para mejorar la seguridad general en redes pequeñas, se recomienda adoptar las siguientes medidas:

- **Autenticación reforzada:** Emplear contraseñas robustas, que incluyan símbolos, números, mayúsculas y minúsculas, y actualizarlas periódicamente. Evitar datos personales o corporativos.
- **Segmentación de red:** Dividir la red en zonas lógicas para contener posibles incidentes y limitar la propagación en caso de intrusión.
- **Copias de seguridad:** Automatizar el respaldo de datos críticos para garantizar su recuperación ante fallos o ataques.
- **Monitorización y detección de intrusos:** Implementar sistemas que permitan vigilar el tráfico de red y detectar actividades anómalas en tiempo real. El uso de un IDS (Sistema de Detección de Intrusos) constituye una defensa activa en redes SOHO. Como ejemplo, *Snort 3* ofrece funciones de inspección profunda de paquetes, entre las que se incluyen la detección de patrones en protocolos de aplicación (HTTP, DNS, FTP), y el soporte nativo para formatos modernos como *JSON* y *HTTP/2*.

5.5. Complementos y plugins para Snort

Snort 3 adopta una arquitectura modular que permite activar o desactivar complementos según las necesidades de cada entorno. Esto resulta especialmente útil en entornos de recursos limitados como una Raspberry Pi (o cualquier otro sistema similar), donde es necesario encontrar un equilibrio entre capacidad de inspección y rendimiento. En el caso de R-SNORT, se han seleccionado e integrado los módulos más relevantes para proteger la red de una PYME sin comprometer la estabilidad del sistema.

A continuación, se detallan los complementos activados y configurados en el archivo, agrupados por su funcionalidad:

- **Inspección de protocolos a nivel de aplicación:** R-SNORT incorpora múltiples plugins para analizar protocolos de red en profundidad. Entre ellos destacan:

- **HTTP y HTTP2 (http_inspect):** Permiten la inspección completa de peticiones y respuestas, incluyendo cabeceras, URIs y cuerpos comprimidos, así como la detección de estructuras anómalas como cabeceras sobredimensionadas o URIs malformadas.
- **DNS, SMTP, FTP, SSH, SIP, Telnet, POP3, IMAP, SSL/TLS:** Cada uno gestionado por su módulo correspondiente, para detectar comportamientos sospechosos y ataques comunes como *exfiltración de datos*, *tunneling* o abuso de protocolos.
- **Inspección de flujos y reensamblado:** La configuración activa los módulos modernos de **Snort 3** para gestionar flujos de red:
 - **Stream IP y Stream TCP:** Sustituyen a los antiguos `frag3` y `stream5`. Se encargan del reensamblado de fragmentos IP y sesiones TCP, previniendo técnicas de evasión mediante solapamiento de paquetes o sesiones incompletas.
 - **Configuraciones de timeout, solapamientos, fragmentos pequeños y límites de sesión:** Estos parámetros han sido ajustados específicamente para mantener un buen rendimiento en entornos embebidos, sin comprometer la seguridad.
- **Reputación IP y listas negras (reputation):** **R-SNORT** integra un sistema de reputación básico que permite bloquear IPs listadas en un archivo de tipo `blocklist.rules`. Esta funcionalidad es útil para prevenir conexiones hacia o desde dominios maliciosos previamente identificados.
- **Soporte para protocolos industriales:** Aunque no todos están activos simultáneamente, el sistema tiene soporte habilitado para protocolos comunes en entornos industriales como Modbus, DNP3, S7CommPlus, CIP o IEC 104. Esto permitiría adaptar **R-SNORT** a una red *OT (Operational Technology)* con muy pocos cambios.
- **Integración con antivirus (ClamAV):** Aunque no se encuentra directamente gestionado desde el archivo de configuración de Snort, el sistema está preparado para trabajar conjuntamente con **ClamAV** mediante scripts de análisis de tráfico o detección de amenazas basada en archivos. Esta funcionalidad complementa a Snort para detectar *malware* en transmisiones de red.
- **Inspección de tráfico cifrado y evasión SSL (ssl):** El módulo de SSL permite detectar y registrar ciertos ataques conocidos como *Heartbleed*, estableciendo límites de tamaño en los mensajes `heartbeat` y controlando el comportamiento anómalo de certificados.
- **Inspección de archivos y tipos MIME:** Aunque se ha desactivado la descompresión de formatos complejos como PDF o ZIP por motivos de rendimiento, el sistema conserva capacidades básicas de inspección de archivos mediante los módulos `file_id` y `file_policy`, útiles para reglas que dependan del tipo de archivo transmitido.

Cabe destacar que **Snort 3** ya no utiliza el formato de salida `Unified2`, por lo que herramientas como **Barnyard2** no son compatibles [27]. En su lugar, **R-SNORT** se apoya en registros en texto plano y en el análisis posterior mediante un sistema de visualización web personalizado.

6. Utilización de Snort en redes SOHO

6.1. Introducción

Un Sistema de Detección de Intrusos en la Red (NIDS, por sus siglas en inglés) es una herramienta de seguridad que permite la monitorización del tráfico en busca de actividades sospechosas o cualquier tipo de ataque que coincida con patrones ya conocidos y documentados con anterioridad. En el caso de pequeñas redes, un NIDS puede proporcionar una primera línea de defensa sin la necesidad de costosas soluciones empresariales favoreciendo el crecimiento libre de peligro cibernético de las PYMES.

En este documento, se describe la instalación y configuración de Snort Versión 3 y complementos como NIDS en una Raspberry Pi 5 con Ubuntu Server, aprovechando su eficiencia en el consumo de recursos y su capacidad de detección de amenazas en redes pequeñas o domésticas.

6.2. Especificaciones, características y requisitos

6.2.1. Especificaciones de Snort 3

Snort 3 es una versión mejorada de su predecesor, con una arquitectura modular y mejoras significativas en rendimiento y configuración. Algunas de sus características incluyen:

- **Arquitectura modular:** Permite una configuración flexible y personalizable apoyada por una gran cantidad de documentación.
- **Mejoras en rendimiento:** Optimizaciones en el motor de detección para un procesamiento más eficiente.
- **Soporte para multihilo:** Permite el uso de múltiples procesadores para mejorar el rendimiento en sistemas modernos.
- **Reglas en Lua:** Facilita una configuración más avanzada y personalizable.
- **Integración con DAQ (Data Acquisition Library):** Ofrece opciones flexibles para la captura de paquetes en la red.
- **Compatibilidad con reglas de Snort 2:** Permite utilizar reglas preexistentes.

6.2.2. Requisitos del sistema

Para garantizar el correcto funcionamiento de Snort en su versión 3, especialmente en entornos donde se emplean múltiples preprocesadores como *daq*, *appid*, *file_inspect*, *http_inspect* o

el preprocesador *SSL/TLS* entre otros, es importante que el sistema anfitrión cumpla con una serie de requisitos mínimos. No obstante, se recomienda ir más allá de estas especificaciones si se desea un rendimiento fluido y una capacidad de respuesta adecuada ante flujos de tráfico moderados o elevados.

A continuación, se detallan los requisitos mínimos y recomendados para la instalación y operación estable de Snort 3.1.84, especialmente en entornos de tipo laboratorio o pequeñas redes empresariales.

Requisitos mínimos

- **Sistema operativo:** Linux (preferiblemente Ubuntu Server 20.04 LTS o superior, o CentOS 7+, Debian)
- **Arquitectura:** x86_64 o ARMv8 (Raspberry Pi compatible, con limitaciones)
- **CPU:** Procesador de 2 núcleos a 1.5 GHz
- **Memoria RAM:** 2 GB
- **Almacenamiento libre:** 4 GB
- **Conectividad de red:** Interfaz Ethernet o Wi-Fi dedicada
- **Dependencias básicas:** CMake (3.5+), GCC (6.0+), libpcap-dev, libpcre, zlib, OpenSSL, LuaJIT, libdnet, entre otros.

Requisitos recomendados

- **Sistema operativo:** Ubuntu Server 22.04 LTS
- **CPU:** 4 núcleos a 2.5 GHz o superior
- **Memoria RAM:** 8 GB
- **Almacenamiento libre:** 20 GB
- **Interfaz de red dedicada:** Tarjeta de red exclusiva en modo promiscuo
- **Extra:** Desactivar NUMA en arquitecturas embebidas como Raspberry Pi.

6.3. Entorno de trabajo

El entorno de trabajo define los componentes físicos y lógicos involucrados en la implementación del NIDS con Snort 3. Para estructurar este apartado, se consideran los siguientes elementos:

6.3.1. Esquema de red

El esquema de red presente en el desarrollo del trabajo está compuesto por un **router doméstico**, un **switch gestionable** que permite *port mirroring* hacia el puerto de **R-SNORT**. De esta manera, no se produce latencia ya que no es necesario interponer el proyecto entre la red

local de máquinas y el router, desarrollándose así las comunicaciones con la fluidez habitual del sistema, junto a otros dos equipos que más adelante servirán para las pruebas de **detección de intrusos** y de **rendimiento**. La **Raspberry Pi** cuenta con dos interfaces *ethernet*: la interfaz `enxc8a362b4a702`, proporcionada por un adaptador externo, será la encargada de recoger el **tráfico total de la red**, y la interfaz predeterminada `eth0`, que será la encargada de suministrar **acceso a internet** a la Raspberry Pi.

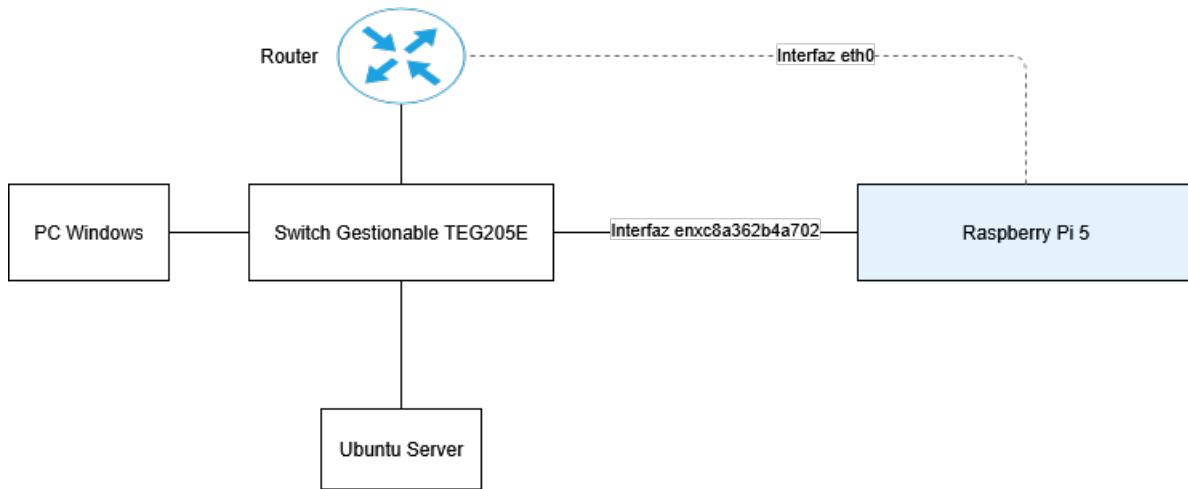


Figura 6.1: Esquema de red real del proyecto.

Descripción del esquema

La red está compuesta por:

- Un **switch gestionable** (modelo Tenda TEG205E) con capacidad de *Port Mirroring*.
- Dos dispositivos cliente conectados a los puertos 2 y 3 del switch: un ordenador con **Windows** y una máquina con **Ubuntu Server**.
- Un **router doméstico** conectado al puerto 1 del switch, encargado de proporcionar acceso a Internet.
- Una **Raspberry Pi 5** equipada con dos interfaces de red Ethernet:
 - `eth0`: Conectada directamente al router, proporciona conectividad a Internet para la propia Raspberry Pi.
 - `enxc8a362b4a702`: Conectada al puerto 4 del switch, el cual está configurado como puerto espejo.

6.3.2. Hardware (Raspberry Pi 5)

La **Raspberry Pi 5** se utiliza como base para la implementación del *NIDS* debido a su bajo consumo energético y su potencia suficiente para analizar tráfico de red en pequeños entornos. Sus características principales incluyen:

- **CPU**: ARM Cortex-A76 (4 núcleos a 2.4 GHz)
- **GPU**: VideoCore VII

- **RAM:** Modelos de 8 GB LPDDR4X

- **Conectividad:**

- 2 puertos USB 3.0
- 2 puertos USB 2.0
- 1 puerto Ethernet Gigabit
- WiFi 802.11ac y Bluetooth 5.0 (*deshabilitados por seguridad*)

- **Almacenamiento:**

- MicroSD de 32 GB
- Soporte para SSD a través de USB 3.0

6.3.3. Software (Snort y sus complementos)

Snort 3 es un sistema de prevención y detección de intrusos en la red (*NIDS/NIPS*) que introduce una serie de mejoras significativas sobre sus versiones anteriores, incluyendo mayor eficiencia, modularidad y una arquitectura basada en plugins. Estas mejoras hacen que **Snort 3** sea más *adaptable, eficiente y personalizable* [28].

Comparación con Snort 2

Snort 3 introduce una serie de mejoras sobre Snort 2:

- Configuración más flexible y simplificada.
- Mayor rendimiento gracias al uso de múltiples hilos.
- Soporte para más de 200 plugins que permiten ampliar su funcionalidad.
- Sistema de reglas más eficiente y simplificado.
- Mejora en la detección de amenazas emergentes y reducción de falsos positivos.

Plugins y complementos utilizados

Para potenciar las capacidades de detección de **Snort 3** en este proyecto, se han seleccionado los siguientes *preprocesadores* y herramientas adicionales:

- **HTTP Inspect:** Analiza tráfico HTTP/HTTPS para detectar ataques *SQL/XSS* e irregularidades en los encabezados.
- **SSL/TLS:** Inspecciona metadatos del tráfico cifrado y detecta conexiones sospechosas o anómalas.
- **Stream IP:** Ensambla paquetes IP fragmentados para detectar intentos de evasión.
- **Stream TCP:** Reensambla flujos TCP/UDP para un análisis más preciso.
- **Reputation Preprocessor:** Bloquea tráfico de fuentes maliciosas basado en listas de reputación.

- **Conjunto de reglas de datos sensibles:** Detecta información sensible como números de tarjetas de crédito o credenciales expuestas.
- **ClamAV:** Sistema antivirus de código abierto que complementa la detección de amenazas de Snort con análisis basado en firmas.

6.4. Instalación y personalización de complementos

6.4.1. Instalación de Snort V3

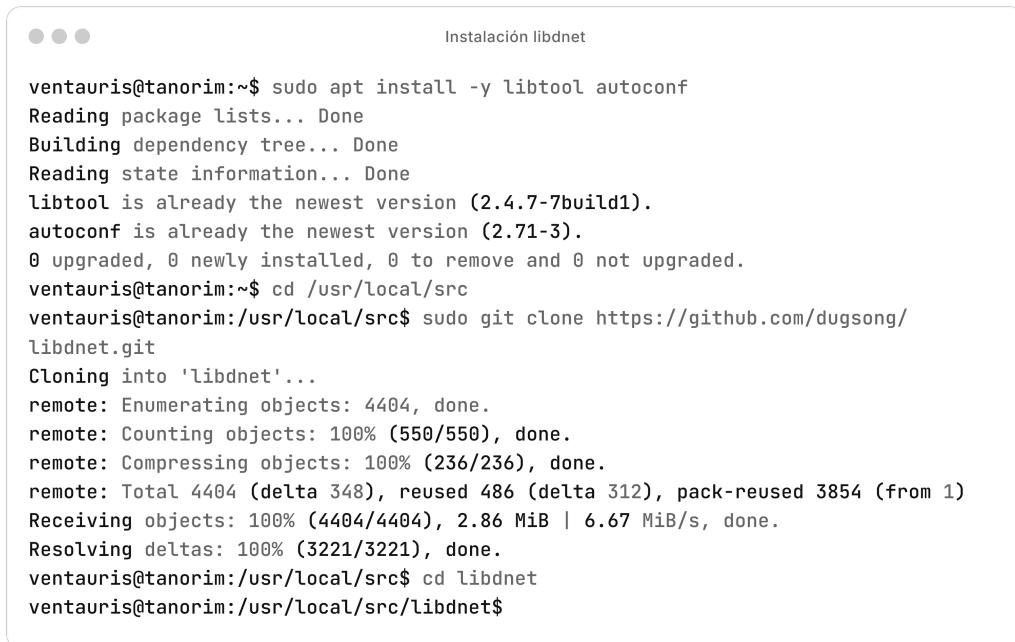
Preparación del entorno

El desarrollo de lo que será **R-SNORT** empezará con la instalación de **Snort** [29] mismo en su versión actual. Para ello, es buena práctica actualizar los repositorios y el equipo mediante un **update** y un **upgrade** antes de comenzar la instalación. Una vez hecho esto, empezaremos con la instalación de las distintas librerías y dependencias de **Snort V3**.

```
1 $ sudo apt-get update && sudo apt-get upgrade -y
```

Instalación y configuración de libdnet

Primero, instalamos un par de herramientas necesarias (**libtool** y **autoconf**) para compilar Snort 3. Luego, nos movemos al directorio **/usr/local/src** y clonamos el repositorio de **libdnet** desde GitHub, que es una librería importante para que Snort funcione correctamente. Finalmente, entramos en el directorio **libdnet** para seguir con la instalación.

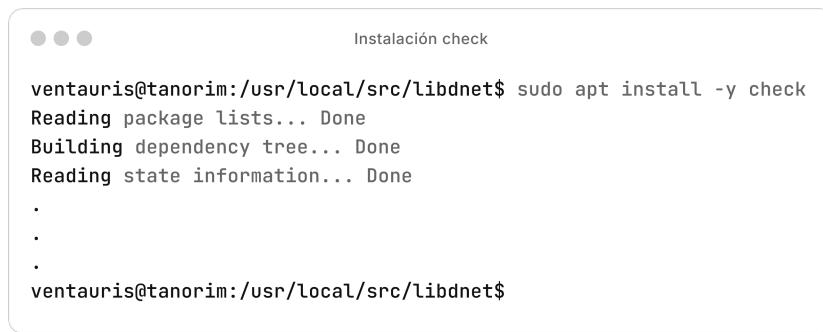


The terminal window shows the following command and its output:

```
Instalación libdnet
ventauris@tanorim:~$ sudo apt install -y libtool autoconf
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libtool is already the newest version (2.4.7-7build1).
autoconf is already the newest version (2.71-3).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
ventauris@tanorim:~$ cd /usr/local/src
ventauris@tanorim:/usr/local/src$ sudo git clone https://github.com/dugsong/libdnet.git
Cloning into 'libdnet'...
remote: Enumerating objects: 4404, done.
remote: Counting objects: 100% (550/550), done.
remote: Compressing objects: 100% (236/236), done.
remote: Total 4404 (delta 348), reused 486 (delta 312), pack-reused 3854 (from 1)
Receiving objects: 100% (4404/4404), 2.86 MiB | 6.67 MiB/s, done.
Resolving deltas: 100% (3221/3221), done.
ventauris@tanorim:/usr/local/src$ cd libdnet
ventauris@tanorim:/usr/local/src/libdnet$
```

Figura 6.2: Instalando dependencias necesarias.

A continuación, el paquete `check` es instalado. Esta herramienta servirá principalmente para ejecutar algunas pruebas en C. Es un requisito para poder compilar `libdnet` correctamente. Gracias a estas preparaciones, estamos configurando el entorno para la correcta compilación de Snort.



```
● ● ●           Instalación check

ventauris@tanorim:/usr/local/src/libdnet$ sudo apt install -y check
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
.
.
.

ventauris@tanorim:/usr/local/src/libdnet$
```

Figura 6.3: Instalando dependencia `check`.

La ejecución de `./configure` prepara el entorno para la compilación de `libdnet`, dependencia fundamental para el correcto funcionamiento del desarrollo posterior y de Snort. Su funcionamiento consiste en la revisión del sistema, verifica dependencias y configura los archivos necesarios para compilar código correctamente.



```
● ● ●           Instalación check

ventauris@tanorim:/usr/local/src/libdnet$ sudo ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a race-free mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
.
.

ventauris@tanorim:/usr/local/src/libdnet$
```

Figura 6.4: Configurando `libdnet` antes de la compilación.

Llevamos a cabo la compilación de **libdnet** mediante **sudo make**; este comando convierte el código fuente de la dependencia en un ejecutable con bibliotecas a disposición para la instalación. Recorre los directorios del proyecto asegurándose de que todos los archivos necesarios sean procesados.



The image shows a terminal window titled "Instalación check". The terminal is running on a Mac OS X system, as indicated by the window title bar. The command entered is "sudo make". The output of the command shows the build process for libdnet. It starts with "Making all in include", then enters the "include" directory, and then the "dnet" directory. It then leaves the "dnet" directory and enters the "test" directory. Finally, it leaves the "test" directory and exits the "libdnet" directory. The process ends with "ventauris@tanorim:/usr/local/src/libdnet\$".

```
Instalación check

ventauris@tanorim:/usr/local/src/libdnet$ sudo make
Making all in include
make[1]: Entering directory '/usr/local/src/libdnet/include'
make  all-recursive
make[2]: Entering directory '/usr/local/src/libdnet/include'
make[3]: Entering directory '/usr/local/src/libdnet/include/dnet'
make[3]: Nothing to be done for 'all'.
make[3]: Leaving directory '/usr/local/src/libdnet/include/dnet'
make[3]: Entering directory '/usr/local/src/libdnet/include'
make[3]: Leaving directory '/usr/local/src/libdnet/include'
make[2]: Leaving directory '/usr/local/src/libdnet/include'
make[1]: Leaving directory '/usr/local/src/libdnet/include'
Making all in man
.
.
.
make[2]: Leaving directory '/usr/local/src/libdnet/test/dnet'
make[2]: Entering directory '/usr/local/src/libdnet/test'
make[2]: Nothing to be done for 'all-am'.
make[2]: Leaving directory '/usr/local/src/libdnet/test'
make[1]: Leaving directory '/usr/local/src/libdnet/test'
make[1]: Entering directory '/usr/local/src/libdnet'
make[1]: Nothing to be done for 'all-am'.
make[1]: Leaving directory '/usr/local/src/libdnet'
ventauris@tanorim:/usr/local/src/libdnet$
```

Figura 6.5: Compilando **libdnet** con **make**.

Ahora ejecutamos `sudo make install` para instalar `libdnet` en el sistema. Este comando copia los archivos compilados a sus directorios correspondientes, asegurando que puedan ser utilizados por otras aplicaciones y librerías.



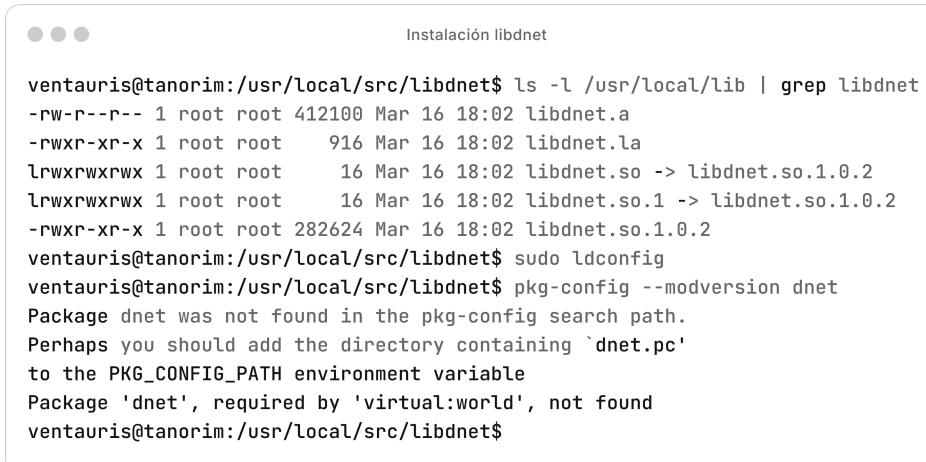
The terminal window shows the command `sudo make install` being run in the directory `/usr/local/src/libdnet`. The output indicates that the installation is successful, with files being copied to their respective include and dnet directories. The process ends with the message "make[1]: Leaving directory '/usr/local/src/libdnet'".

```
Instalación libdnet
ventauris@tanorim:/usr/local/src/libdnet$ sudo make install
Making install in include
make[1]: Entering directory '/usr/local/src/libdnet/include'
Making install in dnet
make[2]: Entering directory '/usr/local/src/libdnet/include/dnet'
make[3]: Entering directory '/usr/local/src/libdnet/include/dnet'
make[3]: Nothing to be done for 'install-exec-am'.
/usr/bin/mkdir -p '/usr/local/include/dnet'
/usr/bin/install -c -m 644 addr.h arp.h blob.h eth.h fw.h icmp.h intf.h ip.h ip6.h
.
.
.
make[1]: Leaving directory '/usr/local/src/libdnet'
ventauris@tanorim:/usr/local/src/libdnet$
```

Figura 6.6: Instalando `libdnet` en el sistema.

Solución al problema de `PKG_CONFIG_PATH`

Listamos los archivos de `libdnet` en `/usr/local/lib` para asegurarnos de que se instalaron correctamente. Luego, intentamos verificar su versión con `pkg-config --modversion dnet`, pero aparece un error indicando que no se encuentra en el `PKG_CONFIG_PATH`. Esto indica que es necesario añadir la ruta correcta para que el sistema reconozca la librería.



The terminal window shows the user listing files in `/usr/local/lib` and then running `ldconfig` to update the library cache. After that, they attempt to use `pkg-config --modversion dnet` but receive an error message stating that the package was not found in the search path. It suggests adding the directory containing `dnet.pc` to the `PKG_CONFIG_PATH`.

```
Instalación libdnet
ventauris@tanorim:/usr/local/src/libdnet$ ls -l /usr/local/lib | grep libdnet
-rw-r--r-- 1 root root 412100 Mar 16 18:02 libdnet.a
-rwxr-xr-x 1 root root    916 Mar 16 18:02 libdnet.la
lrwxrwxrwx 1 root root      16 Mar 16 18:02 libdnet.so -> libdnet.so.1.0.2
lrwxrwxrwx 1 root root      16 Mar 16 18:02 libdnet.so.1 -> libdnet.so.1.0.2
-rwxr-xr-x 1 root root 282624 Mar 16 18:02 libdnet.so.1.0.2
ventauris@tanorim:/usr/local/src/libdnet$ sudo ldconfig
ventauris@tanorim:/usr/local/src/libdnet$ pkg-config --modversion dnet
Package dnet was not found in the pkg-config search path.
Perhaps you should add the directory containing `dnet.pc'
to the PKG_CONFIG_PATH environment variable
Package 'dnet', required by 'virtual:world', not found
ventauris@tanorim:/usr/local/src/libdnet$
```

Figura 6.7: Verificación de la instalación de `libdnet`.

Editamos el archivo `dnet.pc` usando `nano`, añadiendo las rutas correctas para que `pkg-config` pueda encontrar `libdnet`. Definimos las variables `libdir` e `includedir`, asegurándonos de que apunten a `/usr/local/lib` y `/usr/local/include`, respectivamente. Esto soluciona el problema anterior donde `pkg-config` no encontraba la librería.

```

GNU nano 7.2
prefix=/usr/local
exec_prefix=${prefix}
libdir=${exec_prefix}/lib
includedir=${prefix}/include

Name: dnet
Description: libdnet
Version: 1.0
Libs: -L${libdir} -ldnet
Cflags: -I${includedir}

```

Figura 6.8: Configurando `libdnet` para que sea reconocido por el sistema.

Ahora configuraremos el entorno para que `pkg-config` pueda encontrar `libdnet`. Primero, exportamos la variable `PKG_CONFIG_PATH` con la ruta correcta y la agregamos permanentemente al archivo `~/.bashrc`. Luego, recargamos la configuración con `source ~/.bashrc`, actualizamos las librerías con `ldconfig` y verificamos que `libdnet` es reconocido correctamente ejecutando `pkg-config --modversion dnet`, confirmando la versión instalada.

• • • Instalación libdnet

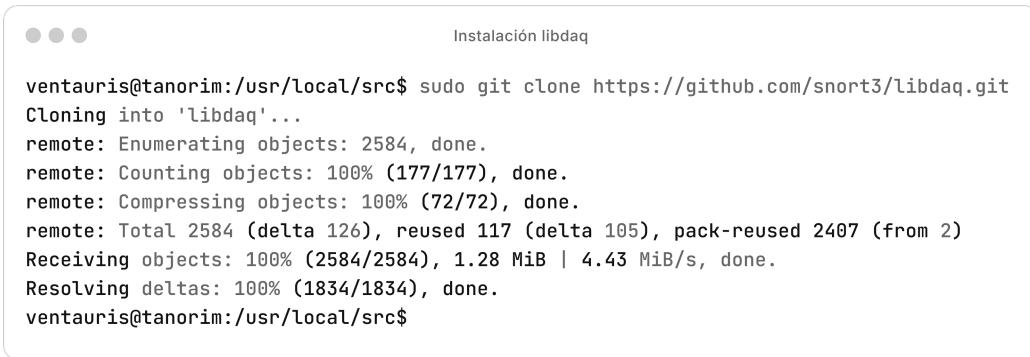
```

ventauris@tanorim:/usr/local/src/libdnet$ export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:$PKG_CONFIG_PATH
ventauris@tanorim:/usr/local/src/libdnet$ echo 'export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:$PKG_CONFIG_PATH' >> ~/.bashrc
ventauris@tanorim:/usr/local/src/libdnet$ source ~/.bashrc
ventauris@tanorim:/usr/local/src/libdnet$ sudo ldconfig
ventauris@tanorim:/usr/local/src/libdnet$ pkg-config --modversion dnet
1.0
ventauris@tanorim:/usr/local/src/libdnet$
```

Figura 6.9: Añadiendo `libdnet` al PATH y verificando su reconocimiento.

Compilación e instalación de libdaq

Clonamos el repositorio de **libdaq**, un componente necesario para **Snort 3**, desde su fuente oficial en GitHub. Este paso descarga el código fuente necesario para su compilación e instalación en la **Raspberry Pi**.



Instalación libdaq

```
ventauris@tanorim:/usr/local/src$ sudo git clone https://github.com/snort3/libdaq.git
Cloning into 'libdaq'...
remote: Enumerating objects: 2584, done.
remote: Counting objects: 100% (177/177), done.
remote: Compressing objects: 100% (72/72), done.
remote: Total 2584 (delta 126), reused 117 (delta 105), pack-reused 2407 (from 2)
Receiving objects: 100% (2584/2584), 1.28 MiB | 4.43 MiB/s, done.
Resolving deltas: 100% (1834/1834), done.
ventauris@tanorim:/usr/local/src$
```

Figura 6.10: Descargando **libdaq** desde el repositorio oficial.

Entramos en el directorio de **libdaq** y ejecutamos el script **bootstrap**, que se encarga de generar los archivos de configuración necesarios para compilar el software correctamente. Se usa **autoreconf** para asegurarse de que todos los scripts de compilación estén en orden.



Instalación libdaq

```
ventauris@tanorim:/usr/local/src$ cd libdaq
ventauris@tanorim:/usr/local/src/libdaq$ sudo ./bootstrap
+ autoreconf -ivf --warnings=all
.
.
.
autoreconf: Leaving directory '.'
ventauris@tanorim:/usr/local/src/libdaq$
```

Figura 6.11: Preparando **libdaq** para su compilación.

Ejecutamos `./configure` para preparar el entorno de compilación de `libdaq`. Como se ha comentado anteriormente, este script revisa que el sistema tenga todas las dependencias necesarias y genera los archivos de configuración adecuados para compilar el software sin complicaciones.



Instalación libdaq

```
ventauris@tanorim:/usr/local/src/libdaq$ sudo ./configure
checking for a BSD-compatible install... /usr/bin/install -c
.
.
.
ventauris@tanorim:/usr/local/src/libdaq$
```

Figura 6.12: Configurando `libdaq` antes de la compilación.

La ejecución de `make` llevará a cabo la compilación de `libdaq`. Este proceso traduce el código fuente a binarios ejecutables, asegurándose de que todas las dependencias y archivos necesarios se generen correctamente para su posterior instalación.



Instalación libdaq

```
ventauris@tanorim:/usr/local/src/libdaq$ sudo make
make all-recursive
make[1]: Entering directory '/usr/local/src/libdaq'
Making all in api
make[2]: Entering directory '/usr/local/src/libdaq/api'
.
.
.
ventauris@tanorim:/usr/local/src/libdaq$
```

Figura 6.13: Compilando `libdaq`.

La instalación final la llevaremos a cabo mediante `sudo make install`. Esto copia los archivos compilados a sus ubicaciones correspondientes dentro del sistema para que puedan ser utilizados por **Snort** y otros programas que así lo requieran.

```
● ● ●           Instalación libdaq

ventauris@tanorim:/usr/local/src/libdaq$ sudo make install
Making install in api
make[1]: Entering directory '/usr/local/src/libdaq/api'
make[2]: Entering directory '/usr/local/src/libdaq/api'
.
.
.
make[1]: Leaving directory '/usr/local/src/libdaq'
ventauris@tanorim:/usr/local/src/libdaq$
```

Figura 6.14: Instalando `libdaq`.

Listamos los archivos de configuración de `libdaq` en `/usr/local/lib/pkgconfig` para asegurarnos de que la instalación se haya completado correctamente. Después, exportamos la variable `PKG_CONFIG_PATH` para que el sistema reconozca la librería. Finalmente, usamos `pkg-config --modversion libdaq` para confirmar que la versión instalada es la **3.0.19**.

```
● ● ●           Instalación libdaq

ventauris@tanorim:/usr/local/src/libdaq$ ls -l /usr/local/lib/pkgconfig | grep libdaq
-rw-r--r-- 1 root root 291 Mar 16 19:39 libdaq.pc
-rw-r--r-- 1 root root 342 Mar 16 19:39 libdaq_static_afpacket.pc
-rw-r--r-- 1 root root 322 Mar 16 19:39 libdaq_static_bpf.pc
-rw-r--r-- 1 root root 316 Mar 16 19:39 libdaq_static_dump.pc
-rw-r--r-- 1 root root 314 Mar 16 19:39 libdaq_static_fst.pc
-rw-r--r-- 1 root root 309 Mar 16 19:39 libdaq_static_gwlb.pc
-rw-r--r-- 1 root root 326 Mar 16 19:39 libdaq_static_pcap.pc
-rw-r--r-- 1 root root 325 Mar 16 19:39 libdaq_static_savefile.pc
-rw-r--r-- 1 root root 313 Mar 16 19:39 libdaq_static_trace.pc
ventauris@tanorim:/usr/local/src/libdaq$ export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:
$PKG_CONFIG_PATH
ventauris@tanorim:/usr/local/src/libdaq$ pkg-config --modversion libdaq
3.0.19
ventauris@tanorim:/usr/local/src/libdaq$
```

Figura 6.15: Verificando la instalación de `libdaq`.

Instalación de la dependencia libhwloc-dev

Instalamos **libhwloc-dev**, una librería necesaria para la ejecución de **Snort 3** y sus dependencias. Se incluyen automáticamente otros paquetes adicionales como **libhwloc-plugins**, **libnuma-dev** y **libpciaccess0**, que ayudan en la gestión de hardware y optimización del rendimiento.



```
Dependencias restantes

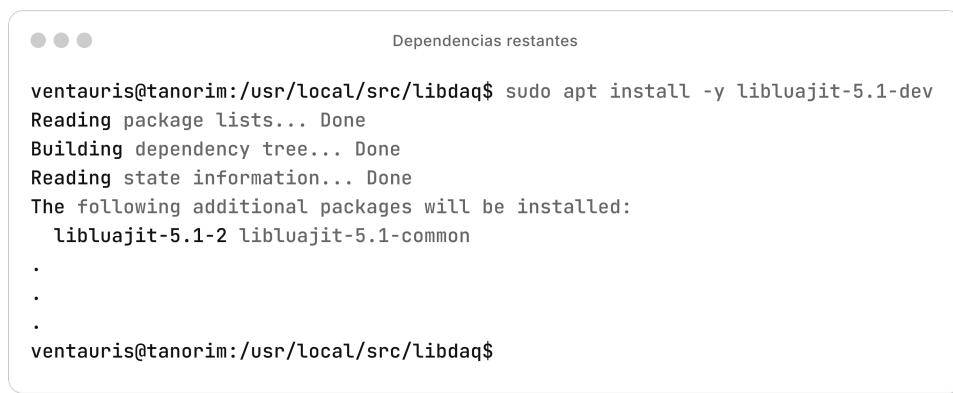
ventauris@tanorim:/usr/local/src/libdaq$ sudo apt install -y libhwloc-dev
[sudo] password for ventauris:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libhwloc-plugins libhwloc15 libnuma-dev libpciaccess0 libxnvctrl0 ocl-icd-libopencl1
.
.
.

ventauris@tanorim:/usr/local/src/libdaq$
```

Figura 6.16: Instalando dependencias necesarias.

Instalación de la dependencia libluajit

Aquí instalamos **libluajit-5.1-dev**, más dependencias para **Snort 3**, ya que utiliza **LuaJIT** para la configuración y personalización de reglas. También se instalan automáticamente **libluajit-5.1-2** y **libluajit-5.1-common**, que contienen las bibliotecas compartidas necesarias para funcionar correctamente.



```
Dependencias restantes

ventauris@tanorim:/usr/local/src/libdaq$ sudo apt install -y libluajit-5.1-dev
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libluajit-5.1-2 libluajit-5.1-common
.
.
.

ventauris@tanorim:/usr/local/src/libdaq$
```

Figura 6.17: Instalando **LuaJIT** para Snort.

Instalación de Flex y Bison

Instalamos **Flex** y **Bison**, dos herramientas para el análisis léxico y sintáctico en la compilación de Snort. Después de la instalación, verificamos la versión de **Flex** con **flex --version** para asegurarnos de que se instaló correctamente.

```
Dependencias restantes

ventauris@tanorim:/usr/local/src/snort3$ sudo apt install flex bison -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libfl-dev libfl2
.
.
.
ventauris@tanorim:/usr/local/src/snort3$ flex --version
flex 2.6.4
```

Figura 6.18: Instalando **Flex** y **Bison**, herramientas necesarias para la compilación.

En este paso, instalamos **libpcre2-dev**, una biblioteca para el manejo de expresiones regulares en **Snort 3**. Nos es útil para la detección de patrones en el tráfico de red.

```
Dependencias restantes

ventauris@tanorim:/usr/local/src/snort3$ sudo apt install libpcre2-dev -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libpcre2-16-0 libpcre2-32-0 libpcre2-posix3
.
.
.
ventauris@tanorim:/usr/local/src/snort3$
```

Figura 6.19: Instalando la biblioteca PCRE2 para el manejo de expresiones regulares.

Clonación y compilación del código fuente de Snort

Descargamos el código fuente de **Snort 3** directamente desde su repositorio oficial en GitHub mediante `git clone`.

```
● ● ●           Instalación de Snort V3

ventauris@tanorim:/usr/local/src$ sudo git clone https://github.com/snort3/snort3.git
[sudo] password for ventauris:
Cloning into 'snort3'...
remote: Enumerating objects: 120713, done.
remote: Counting objects: 100% (14370/14370), done.
remote: Compressing objects: 100% (2587/2587), done.
remote: Total 120713 (delta 12427), reused 12205 (delta 11783), pack-reused 106343 (from 5)

Receiving objects: 100% (120713/120713), 87.54 MiB | 10.79 MiB/s, done.
Resolving deltas: 100% (103328/103328), done.
ventauris@tanorim:/usr/local/src$
```

Figura 6.20: Clonando el repositorio de Snort 3.

Aquí establecemos la variable `my_path` para definir el directorio de instalación de Snort. Luego, agregamos esta configuración al archivo `/.bashrc` para que se cargue automáticamente en futuras sesiones. Finalmente, usamos `source` `/.bashrc` para aplicar los cambios de inmediato.

```
● ● ●           Instalación de Snort V3

ventauris@tanorim:/usr/local/src/snort3$ export my_path=/usr/local/snort
ventauris@tanorim:/usr/local/src/snort3$ echo 'export my_path=/usr/local/snort' >>
~/.bashrc
ventauris@tanorim:/usr/local/src/snort3$ source ~/.bashrc
ventauris@tanorim:/usr/local/src/snort3$
```

Figura 6.21: Definiendo la ruta de instalación de Snort.

Ejecutamos el script `configure_cmake.sh` para configurar la compilación de **Snort 3**. Se especifica el prefijo de instalación con `$my_path`. Se generan los archivos de configuración y se definen las opciones de características, incluyendo los módulos DAQ que se activarán.

```
Instalación de Snort V3

ventauris@tanorim:/usr/local/src/snort3$ sudo ./configure_cmake.sh --prefix=$my_path
./configure_cmake.sh: 523: [: Illegal number:
Build Directory : build
Source Directory: /usr/local/src/snort3
CMake Warning:
  Ignoring empty string ("") provided on the command line.

CMake Deprecation Warning at CMakeLists.txt:1 (cmake_minimum_required):
  Compatibility with CMake < 3.5 will be removed from a future version of
  CMake.

  Update the VERSION argument <min> value or use a ...<max> suffix to tell
  CMake that the project does not need compatibility with older versions.

.

.

.

Feature options:
  DAQ Modules:      Static (afpacket;bpf;dump;fst;gwlb;pcap;savefile;trace)
  libatomic:         System-provided
  Hyperscan:        OFF
  ICONV:            ON
  Libunwind:        OFF
  LZMA:             ON
  RPC DB:           Built-in
  SafeC:            OFF
  TCMalloc:          OFF
  JEMalloc:          OFF
  UUID:              OFF
  NUMA:              ON
  LibML:             OFF
-----
-- Configuring done (5.3s)
-- Generating done (0.4s)
-- Build files have been written to: /usr/local/src/snort3/build
ventauris@tanorim:/usr/local/src/snort3$
```

Figura 6.22: Configurando Snort 3 con CMake.

Compilación e instalación de Snort

Compilamos **Snort** con `make -j $(nproc)`, lo que nos permite aprovechar todos los núcleos del procesador para acelerar el proceso.

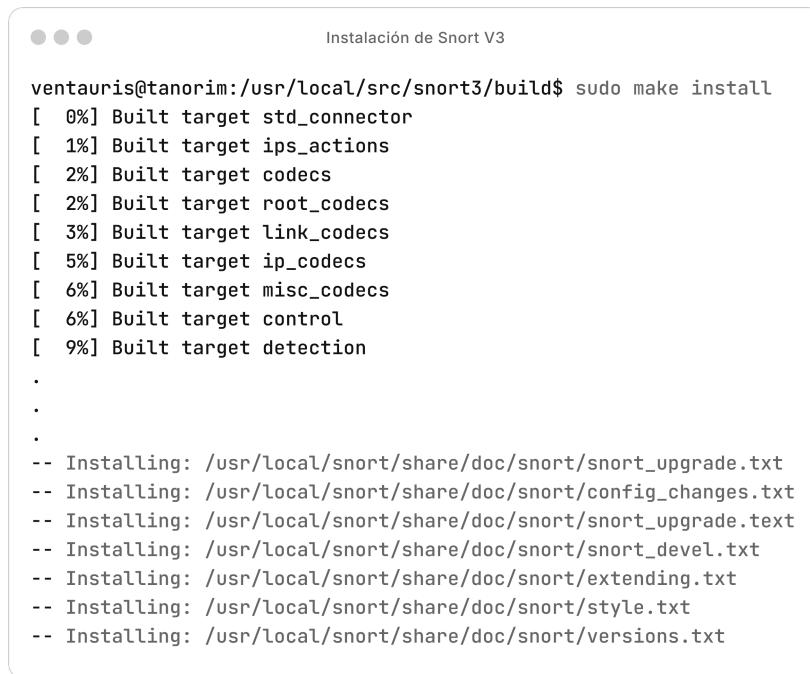


The image shows a terminal window titled "Instalación de Snort V3". The window contains a command-line log of the Snort compilation process. The log shows the compilation of various CXX objects and executables, with progress percentages from 98% to 100%. The compilation is performed in parallel using multiple cores, as indicated by the frequent use of the word "[100%]" followed by core numbers like "1", "2", "3", "4", "5", "6", "7", "8", "9", and "10". The log ends with the command "ventauris@tanorim:/usr/local/src/snort3/build\$".

```
.  
. [ 98%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/  
preprocessor_states.dir/pps_imap.cc.o  
[ 98%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/  
preprocessor_states.dir/pps_modbus.cc.o  
[ 98%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/  
preprocessor_states.dir/pps_rna.cc.o  
[100%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/  
preprocessor_states.dir/pps_smtp.cc.o  
[100%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/  
preprocessor_states.dir/pps_sfportscan.cc.o  
[100%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/  
preprocessor_states.dir/pps_stream5_ip.cc.o  
[100%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/  
preprocessor_states.dir/pps_stream5_global.cc.o  
[100%] Linking CXX executable snort  
[100%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/  
preprocessor_states.dir/pps_stream5_tcp.cc.o  
[100%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/  
preprocessor_states.dir/pps_stream5_udp.cc.o  
[100%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/  
preprocessor_states.dir/pps_stream5_ha.cc.o  
[100%] Building CXX object tools/snort2lua/preprocessor_states/CMakeFiles/  
preprocessor_states.dir/preprocessor_api.cc.o  
[100%] Built target preprocessor_states  
[100%] Building CXX object tools/snort2lua/CMakeFiles/snort2lua.dir/snort2lua.cc.o  
[100%] Building CXX object tools/snort2lua/CMakeFiles/snort2lua.dir/init_state.cc.o  
[100%] Linking CXX executable snort2lua  
[100%] Built target snort2lua  
[100%] Built target snort  
ventauris@tanorim:/usr/local/src/snort3/build$
```

Figura 6.23: Compilación de Snort.

Instalamos Snort tras haberlo compilado con `sudo make install`. Este comando copia los archivos generados en sus respectivas ubicaciones dentro del sistema.



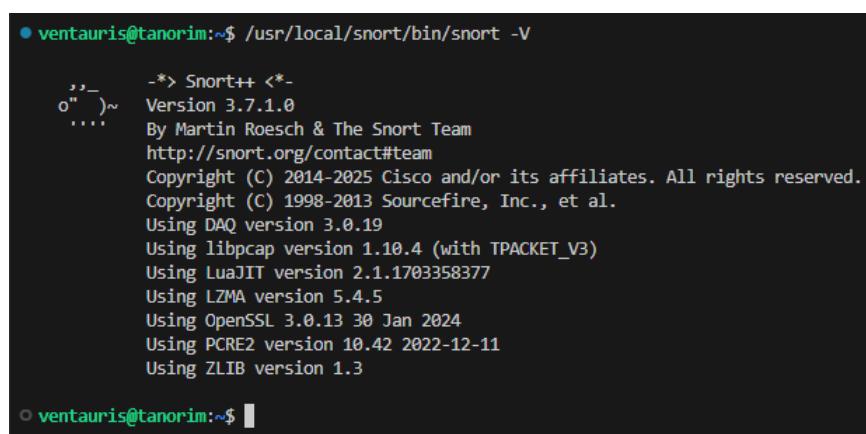
The screenshot shows a terminal window titled "Instalación de Snort V3". It displays the output of the command `sudo make install`. The progress bar at the top shows the build percentage from 0% to 9%. The main output lists various targets being built and files being installed. The targets include std_connector, ips_actions, codecs, root_codecs, link_codecs, ip_codecs, misc_codecs, control, and detection. The installed files are snort_upgrade.txt, config_changes.txt, snort_upgrade.text, snort-devel.txt, extending.txt, style.txt, and versions.txt.

```
ventauris@tanorim:/usr/local/src/snort3/build$ sudo make install
[  0%] Built target std_connector
[  1%] Built target ips_actions
[  2%] Built target codecs
[  2%] Built target root_codecs
[  3%] Built target link_codecs
[  5%] Built target ip_codecs
[  6%] Built target misc_codecs
[  6%] Built target control
[  9%] Built target detection
.
.
.
-- Installing: /usr/local/snort/share/doc/snort/snort_upgrade.txt
-- Installing: /usr/local/snort/share/doc/snort/config_changes.txt
-- Installing: /usr/local/snort/share/doc/snort/snort_upgrade.text
-- Installing: /usr/local/snort/share/doc/snort/snort-devel.txt
-- Installing: /usr/local/snort/share/doc/snort/extending.txt
-- Installing: /usr/local/snort/share/doc/snort/style.txt
-- Installing: /usr/local/snort/share/doc/snort/versions.txt
```

Figura 6.24: Instalación de Snort.

Configuración inicial y validación

Finalmente, verificamos que **Snort** se ha instalado correctamente con el comando: `/usr/local/snort/bin/snort -V`. Esto nos muestra la versión instalada (**3.7.1.0**) junto con las bibliotecas y dependencias utilizadas, como **DAQ**, **libpcap**, **LuaJIT**, **OpenSSL**, entre otras.



The screenshot shows a terminal window displaying the output of the command `/usr/local/snort/bin/snort -V`. The output shows the Snort++ version 3.7.1.0, copyright information from Cisco and Sourcefire, and a list of dependencies and libraries used during compilation, including DAQ, libpcap, LuaJIT, LZMA, OpenSSL, PCRE2, and ZLIB.

```
ventauris@tanorim:~$ /usr/local/snort/bin/snort -V
,,-  -*> Snort++ <*-_
o" )~ Version 3.7.1.0
... By Martin Roesch & The Snort Team
http://snort.org/contact#team
Copyright (C) 2014-2025 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using DAQ version 3.0.19
Using libpcap version 1.10.4 (with TPACKET_V3)
Using LuaJIT version 2.1.1703358377
Using LZMA version 5.4.5
Using OpenSSL 3.0.13 30 Jan 2024
Using PCRE2 version 10.42 2022-12-11
Using ZLIB version 1.3

ventauris@tanorim:~$
```

Figura 6.25: Snort instalado con éxito.

Creamos el directorio de configuración de Snort (`/usr/local/snort/etc/snort`) y copiamos los archivos de configuración en formato Lua desde el directorio fuente de **Snort 3**. Luego, ejecutamos Snort con la configuración especificada para validar que todo esté correctamente configurado. La salida muestra que Snort ha cargado las reglas y módulos sin errores ni advertencias.

```
● ● ●           Instalación de Snort V3

ventauris@tanorim:~$ sudo mkdir -p /usr/local/snort/etc/snort
ventauris@tanorim:~$ sudo cp /usr/local/src/snort3/lua/*.lua /usr/local/snort/etc/snort/
ventauris@tanorim:~$ /usr/local/snort/bin/snort -c /usr/local/snort/etc/snort/snort.lua
-----
o")~  Snort++ 3.7.1.0
-----
Loading /usr/local/snort/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
    stream_ip
    stream_icmp
    .
    .

search engine (ac_bnfa)
    instances: 2
    patterns: 438
    pattern chars: 2602
    num states: 1832
    num match states: 392
    memory scale: KB
    total memory: 71.2812
    pattern memory: 19.6484
    match list memory: 28.4375
    transition memory: 22.9453
appid: MaxRss diff: 2944
appid: patterns loaded: 300
-----
pcap DAQ configured to passive.

Snort successfully validated the configuration (with 0 warnings).
o")~  Snort exiting
ventauris@tanorim:~$
```

Figura 6.26: Configuración de Snort validada con éxito.

6.4.2. Instalación de reglas y plugins

Tras la correcta instalación de Snort, aprovecharemos la arquitectura modular introducida en la versión 3, la cual utiliza archivos con extensión .lua para gestionar su configuración de forma más flexible y estructurada. A continuación, se presenta una guía paso a paso sobre cómo instalar las reglas básicas proporcionadas por la comunidad de Snort, conocidas como *Community Rules*.

```
● ● ● Descarga de las community rules
ventauris@tanorim:/usr/local/snort/etc/snort$ sudo wget https://www.snort.org/downloads/
community/snort3-community-rules.tar.gz
.
.
.
snort3-community-rules.tar.gz
100%[=====] 323.67K 998KB/s in 0.3s
2025-03-21 22:58:21 (998 KB/s) - 'snort3-community-rules.tar.gz' saved [331442/331442]

ventauris@tanorim:/usr/local/snort/etc/snort$ sudo tar -xvzf snort3-community-rules.tar.gz
snort3-community-rules/
snort3-community-rules/snort3-community.rules
snort3-community-rules/VRT-License.txt
snort3-community-rules/LICENSE
snort3-community-rules/AUTHORS
snort3-community-rules/sid-msg.map
ventauris@tanorim:/usr/local/snort/etc/snort$ ls
AUTHORS balanced.lua custom.rules inline.lua security.lua
snort.conf snort3-community-rules snort_defaults.lua
LICENSE connectivity.lua file_magic.rules max_detect.lua sensitive_data.rules
snort.lua snort3-community-rules.tar.gz talos.lua
ventauris@tanorim:/usr/local/snort/etc/snort$ cd snort3-community-rules/
ventauris@tanorim:/usr/local/snort/etc/snort/snort3-community-rules$ ls
AUTHORS LICENSE VRT-License.txt sid-msg.map snort3-community.rules
ventauris@tanorim:/usr/local/snort/etc/snort/snort3-community-rules$
```

Figura 6.27: Añadiendo reglas preconfiguradas a Snort.

Posteriormente, editamos el archivo `snort.lua` [30], ubicado en `/usr/local/snort/etc/snort/`, para incluir la ruta del archivo `community.rules`. Esto permite que Snort cargue automáticamente estas reglas al iniciar y pueda utilizarlas para detectar patrones maliciosos en el tráfico de red. La edición del archivo puede realizarse con herramientas como `nano`, `vim` o cualquier otro editor de texto de preferencia.

```
ips =
{
    -- use this to enable decoder and inspector alerts
    --enable_builtin_rules = true,

    -- use include for rules files; be sure to set your path
    -- note that rules files can include other rules files
    -- (see also related path vars at the top of snort_defaults.lua)
    rules = [[
        include /usr/local/snort/etc/snort/snort3-community-rules/snort3-community.rules
    ]],
    variables = default_variables
}
```

Figura 6.28: Modificación de `snort.lua` para agregar las reglas preconfiguradas.

Una vez agregadas las reglas a `snort.lua`, realizamos una validación de la configuración ejecutando Snort en modo prueba. Esto permite comprobar que no existan errores sintácticos o de carga, garantizando así que el sistema pueda funcionar correctamente.



The terminal window shows the validation process:

```
Comprobación de las reglas
ventauris@tanorim:/usr/local/snort/etc/snort$ sudo snort -c /usr/local/snort/etc/snort/
snort.lua
-----
o")~  Snort++ 3.7.1.0
-----
Loading /usr/local/snort/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
.
.
.
pcap DAQ configured to passive.

Snort successfully validated the configuration (with 0 warnings).
o")~  Snort exiting
ventauris@tanorim:/usr/local/snort/etc/snort$
```

Figura 6.29: Validación configuración de Snort.

Configuración preprocesador HTTP Inspect

Accedemos al archivo `snort.lua` y localizamos la sección correspondiente al preprocesador `http_inspect = {}`. En ella escribimos los parámetros adecuados para la gestión y protección de una red corporativa de pequeña o mediana empresa (PYME). Esta configuración permite a Snort analizar a fondo el tráfico HTTP y detectar comportamientos anómalos asociados a amenazas comunes como ataques por inyección, carga de malware o manipulación de cabeceras.

Las reglas aplicadas son las siguientes:

Parámetro	Valor	Uso en una PYME
<code>request_depth</code>	-1	Inspeccionar todo el contenido de la petición HTTP. Permite detectar inyecciones SQL, código malicioso y malware embbebido.
<code>response_depth</code>	-1	Revisar la totalidad de la respuesta enviada por el servidor, útil para detectar descargas de archivos maliciosos.
<code>unzip</code>	<code>true</code>	Habilita la descompresión de contenidos codificados como gzip o deflate. Evita que el contenido malicioso comprimido pase desapercibido.
<code>oversize_dir_length</code>	500	Genera alertas si una URI presenta una ruta anormalmente larga, característica común en ataques de desbordamiento o evasión.
<code>maximum_headers</code>	200	Detecta un número excesivo de cabeceras HTTP, que puede ser indicio de ataques por manipulación de protocolo.

Cuadro 6.1: Parámetros de `http_inspect`.

```
-- http_inspect para inspección HTTP
http_inspect =
{
    -- Escanear todo el cuerpo de la petición/respuesta (ojo a la carga en una Pi)
    request_depth = -1,
    response_depth = -1,

    -- Activa descompresión de gzip/deflate para inspeccionar payload
    unzip = true,

    -- Longitud máxima de directorio en URI, pasado este valor se dispara alerta 119:15
    oversize_dir_length = 500,

    -- Número máximo de cabeceras permitidas (ej. 200), si se superan -> alerta 119:20
    maximum_headers = 200,

    -- Tamaño máximo (en bytes) de una cabecera individual antes de alertar 119:19
    maximum_header_length = 4096,

    -- ¿Normalizar caracteres UTF en las respuestas?
    normalize_utf = true,

    -- Descomprimir PDF, SWF, ZIP, etc. (cuidado con rendimiento)
    decompress_pdf = false,
    decompress_swf = false,
    decompress_zip = false,
    decompress_vba = false,

    -- Profundidad de escaneo en adjuntos MIME
    max_mime_attach = 5,

    -- Ejemplo: bloquear (o alertar) si el cliente usa ciertos métodos
    --allowed_methods = 'GET,POST,HEAD,OPTIONS',
    --disallowed_methods = 'DELETE,TRACE,TRACK'
    -- (Solo activalo si estás seguro de que tu app no requiere esos métodos)

    -- Manejo de + como espacio en URIs
    plus_to_space = true
}
```

Figura 6.30: Configuración http_inspect.

```
binder =
{
    -- port bindings required for protocols without wizard support
    { when = { proto = 'udp', ports = '53', role='server' }, use = { type = 'dns' } },
    { when = { proto = 'tcp', ports = '53', role='server' }, use = { type = 'dns' } },
    { when = { proto = 'tcp', ports = '111', role='server' }, use = { type = 'rpc_decode' } },
    { when = { proto = 'tcp', ports = '502', role='server' }, use = { type = 'modbus' } },
    { when = { proto = 'tcp', ports = '2123 2152 3386', role='server' }, use = { type = 'gtp_inspect' } },
    { when = { proto = 'tcp', ports = '2404', role='server' }, use = { type = 'iec104' } },
    { when = { proto = 'udp', ports = '2222', role = 'server' }, use = { type = 'cip' } },
    { when = { proto = 'tcp', ports = '44818', role = 'server' }, use = { type = 'cip' } },

    { when = { proto = 'tcp', service = 'dcerpc' }, use = { type = 'dce_tcp' } },
    { when = { proto = 'udp', service = 'dcerpc' }, use = { type = 'dce_udp' } },
    { when = { proto = 'udp', service = 'netflow' }, use = { type = 'netflow' } },

    { when = { service = 'netbios-ssn' }, use = { type = 'dce_smb' } },
    { when = { service = 'dce_http_server' }, use = { type = 'dce_http_server' } },
    { when = { service = 'dce_http_proxy' }, use = { type = 'dce_http_proxy' } },

    { when = { service = 'cip' }, use = { type = 'cip' } },
    { when = { service = 'dnp3' }, use = { type = 'dnp3' } },
    { when = { service = 'dns' }, use = { type = 'dns' } },
    { when = { service = 'ftp' }, use = { type = 'ftp_server' } },
    { when = { service = 'ftp-data' }, use = { type = 'ftp_data' } },
    { when = { service = 'gtp' }, use = { type = 'gtp_inspect' } },
    { when = { service = 'imap' }, use = { type = 'imap' } },
    { when = { service = 'http' }, use = { type = 'http_inspect' } },
    { when = { service = 'http2' }, use = { type = 'http2_inspect' } },
    { when = { service = 'iec104' }, use = { type = 'iec104' } },
    { when = { service = 'mms' }, use = { type = 'mms' } },
    { when = { service = 'modbus' }, use = { type = 'modbus' } },
    { when = { service = 'pop3' }, use = { type = 'pop' } },
    { when = { service = 'ssh' }, use = { type = 'ssh' } },
    { when = { service = 'sip' }, use = { type = 'sip' } },
    { when = { service = 'smtp' }, use = { type = 'smtp' } },
    { when = { service = 'ssl' }, use = { type = 'ssl' } },
    { when = { service = 'sunrpc' }, use = { type = 'rpc_decode' } },
    { when = { service = 's7complus' }, use = { type = 's7complus' } },
    { when = { service = 'telnet' }, use = { type = 'telnet' } },

    { use = { type = 'wizard' } },

    {
        when = { proto = 'tcp', ports = '80 443', role = 'server' },
        use = { type = 'http_inspect' }
    }
}
```

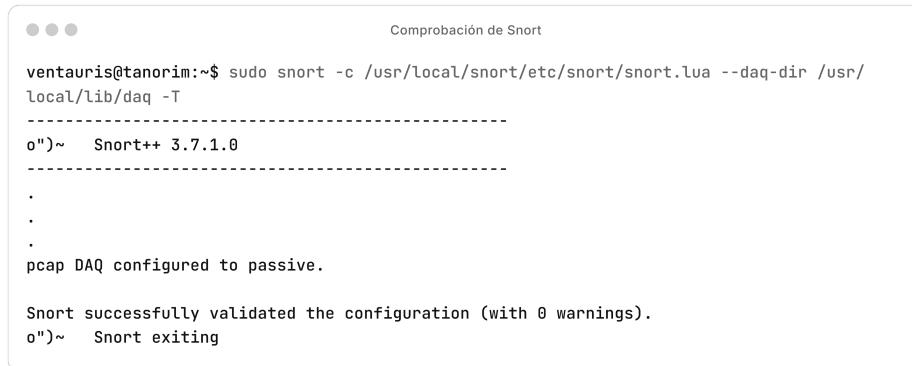
Figura 6.31: Configuración de binders.

Además de la configuración general de `http_inspect` mostrada en la Figura 6.30, es necesario agregar, al final de la sección de `binders` en el archivo `snort.lua`, el bloque de código representado en la Figura 6.31 para asegurar su correcto funcionamiento.

Esta configuración adicional es esencial para que **Snort** pueda vincular correctamente el

tráfico HTTP con su correspondiente módulo de inspección durante el análisis en tiempo real. Sin este paso, es posible que el motor de detección no aplique correctamente las reglas diseñadas para HTTP, reduciendo así la eficacia del sistema de detección de intrusiones.

Por último, validamos la configuración ejecutando Snort en modo prueba para asegurarnos de que no se han introducido errores y que todas las configuraciones han sido cargadas exitosamente.



The terminal window shows the command: sudo snort -c /usr/local/snort/etc/snort/snort.lua --daq-dir /usr/local/lib/daq -T. The output indicates Snort++ 3.7.1.0, and it successfully validates the configuration with 0 warnings before exiting.

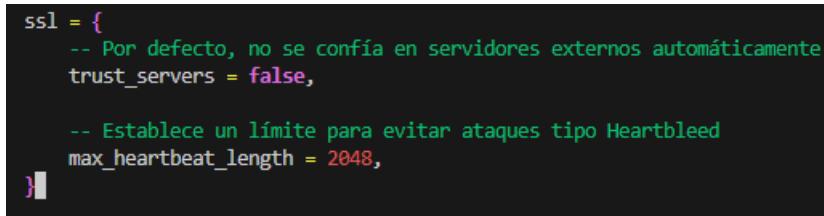
```
Comprobación de Snort
ventauris@tanorim:~$ sudo snort -c /usr/local/snort/etc/snort/snort.lua --daq-dir /usr/local/lib/daq -T
-----
o")~  Snort++ 3.7.1.0
-----
.
.
.
pcap DAQ configured to passive.

Snort successfully validated the configuration (with 0 warnings).
o")~  Snort exiting
```

Figura 6.32: Validación de la configuración.

SSL Inspector

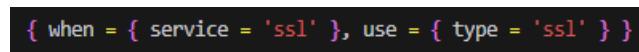
El siguiente módulo que habilitaremos será el **SSL Inspector**. El procedimiento de activación es similar al seguido en módulos anteriores. Comenzaremos por modificar el archivo **snort.lua**, donde localizaremos la sección correspondiente a **ssl** y añadiremos el bloque de configuración mostrado a continuación.



```
ssl = {
    -- Por defecto, no se confía en servidores externos automáticamente
    trust_servers = false,
    -- Establece un límite para evitar ataques tipo Heartbleed
    max_heartbeat_length = 2048,
}
```

Figura 6.33: Configuración de SSL.

Posteriormente, es necesario vincular la configuración de **ssl** en la sección **binders**, en caso de que no se haya hecho previamente. Esta vinculación garantiza que el tráfico cifrado sea correctamente dirigido al módulo de inspección SSL.



```
{ when = { service = 'ssl' }, use = { type = 'ssl' } }
```

Figura 6.34: Vincular con binders.

A continuación, verificamos la configuración de Snort para asegurarnos de que la sintaxis sea válida. Si todo es correcto, reiniciamos el servicio mediante **systemctl** para aplicar los cambios.

```
1 # Validar la configuracion de Snort
2 sudo snort -c /usr/local/snort/etc/snort/snort.lua -T
3
4 # Reiniciar el servicio de Snort
5 sudo systemctl restart snort
```

Stream IP

Antiguamente conocido como **Frag3**, este preprocesador ha sido reemplazado por una versión más moderna bajo el nombre de **Stream IP**. Este módulo es responsable de reensamblar fragmentos de paquetes IP para prevenir técnicas de evasión. Su configuración se realiza en el archivo **snort.lua**.

```
stream_ip = {
    max_frags = 8192,          -- máximo número de fragmentos simultáneos
    max_overlaps = 5,           -- máximo número permitido de solapamientos (0 para ilimitado)
    min_frag_length = 128,      -- alerta si la longitud del fragmento es menor que 128 bytes
    min_ttl = 5,                -- ignora fragmentos con TTL menor a 5
    policy = 'linux',           -- política de reensamblado (por defecto recomendado)
    session_timeout = 60,        -- tiempo en segundos antes de eliminar una sesión de reensamblado IP
}
```

Figura 6.35: Configuración de Stream IP.

Una vez completada la configuración, validamos la sintaxis del archivo y reiniciamos Snort para aplicar los cambios.

Stream TCP

Otro preprocesador que puede marcar la diferencia en protección de redes SOHO es **Stream TCP**, sucesor de **Stream5**. Este módulo permite el reensamblado de flujos TCP, lo cual es esencial para una inspección precisa del tráfico de red a nivel de sesión. Su configuración se realiza de forma similar a **Stream IP**. Tras editar **snort.lua**, comprobamos la validez de la sintaxis y reiniciamos el servicio.

```
stream_tcp = {
    policy = 'linux',                                -- Política de reensamblado optimizada para Linux
    max_window = 1048576,                            -- Ventana TCP máxima permitida (1 MB, segura y suficiente para la mayoría de las PYMES)
    overlap_limit = 10,                             -- Limita la cantidad máxima de segmentos solapados (protege contra ataques de evasión)
    max_pdu = 16384,                                -- Tamaño máximo permitido para PDU reensambladas (16 KB, valor seguro por defecto)
    reassemble_async = true,                         -- Reensamblar aunque el tráfico aún no se haya visto en ambas direcciones
    queue_limit = {
        max_bytes = 4194304,                          -- Máximo de bytes en cola por sesión/dirección (4 MB, valor seguro estándar)
        max_segments = 2048,                           -- Máximo número de segmentos en cola por sesión/dirección
        asymmetric_ids_flush_threshold = 2097152, -- Umbral de descarga para flujos asimétricos (2 MB, protege memoria)
    },
    small_segments = {
        count = 5,                                 -- Genera alerta si se reciben 5 segmentos TCP pequeños consecutivos
        maximum_size = 64,                           -- Considera pequeño cualquier segmento menor a 64 bytes
    },
    session_timeout = 180,                           -- Tiempo de espera para cerrar sesiones inactivas (3 minutos)
    embryonic_timeout = 30,                         -- Tiempo de espera para conexiones no establecidas (30 segundos)
    idle_timeout = 1800,                            -- Cerrar sesiones tras 30 min sin actividad (libera recursos)
}
```

Figura 6.36: Configuración de Stream TCP.

Reputation

Este preprocesador permite bloquear IPs clasificadas como maliciosas utilizando listas negras. Para su funcionamiento se ha creado una carpeta **/reputation**, donde se almacena un archivo con IPs sospechosas, obtenidas del repositorio de *emergingthreats.net* [31]. Esta funcionalidad es útil para evitar conexiones no deseadas hacia o desde nodos peligrosos.

Como en los casos anteriores, tras configurar el módulo, se valida la sintaxis y se reinicia Snort para aplicar los cambios.

```
reputation = {
    blocklist = 'blocklist.rules',
    -- allowlist no es obligatoria ahora, pero se pueden hacer excepciones
    list_dir = '/usr/local/snort/etc/snort/reputation',
    memcap = 500,
    nested_ip = 'inner',
    priority = 'allowlist',
    scan_local = false,
    allow = 'do_not_block',
}
```

Figura 6.37: Configuración de Reputation.

Datos sensibles

En versiones anteriores de Snort existía el preprocesador *Sensitive Data*, pero ha sido descontinuado en las versiones más recientes. Sin embargo, es posible emular parte de su funcionalidad adaptando expresiones regulares personalizadas que alerten sobre la presencia de datos sensibles típicos en el entorno de una PYME, como credenciales o números de tarjetas.

Para ello, se crea un nuevo archivo de reglas llamado `custom.rules`, donde se definen patrones específicos a detectar.

```
GNU nano 7.2                                         /usr/local/snort/etc/snort/custom.rules *
# Detección de emails en texto plano
alert tcp $HOME_NET any -> $EXTERNAL_NET any (
    msg:"Sensitive Data - Email detected";
    flow:established,to_server;
    pcre:"/[a-zA-Z0-9._%-+@[a-zA-Z0-9.-]+.[a-zA-Z]{2,}"/";
    classtype:sdf; sid:1000001; rev:1;
)

# Detección de número de tarjeta de crédito Visa, Mastercard, Amex, Discover (16 dígitos básico)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (
    msg:"Sensitive Data - Credit Card detected";
    flow:established,to_server;
    pcre:"/^\b(?:4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14}|3[47][0-9]{13}|6(?:011|5[0-9]{2})[0-9]{12})\b/";
    classtype:sdf; sid:1000002; rev:1;
)

#Detección de Número de Seguridad Social (NUSS) de España (formato: 12 dígitos, sin espacios)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (
    msg:"Sensitive Data - NUSS España detectado";
    flow:established,to_server;
    pcre:"/\b\d{12}\b/";
    classtype:policy-violation; sid:1000003; rev:1;
)
```

Figura 6.38: Expresiones regulares para protección de datos sensibles.

Posteriormente, se añade la ruta de este archivo en `snort.lua`, se comprueba la validez de la configuración utilizando `libdaq` y se reinicia el servicio para aplicar los cambios.

```
ips =
{
    -- use this to enable decoder and inspector alerts
    --enable_builtin_rules = true,

    -- use include for rules files; be sure to set your path
    -- note that rules files can include other rules files
    -- (see also related path vars at the top of snort_defaults.lua)
    rules = [[
        include /usr/local/snort/etc/snort/snort3-community-rules/snort3-community.rules
        include /usr/local/snort/etc/snort/custom.rules
    ]],
    variables = default_variables
}
```

Figura 6.39: Agregación de `custom.rules`.

Antivirus ClamAV

Finalmente, se procede con la instalación del antivirus **ClamAV**, una solución de código abierto utilizada para la detección de malware. Su integración complementa las capacidades de Snort mediante el análisis basado en firmas.

```
● ventauris@tanorim:~$ sudo apt install clamav clamav-daemon -y
[sudo] password for ventauris:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  clamav-base clamav-freshclam clamdscan libclamav11t64
Suggested packages:
  libclamunrar clamav-docs daemon libclamunrar11
The following NEW packages will be installed:
  clamav clamav-base clamav-daemon clamav-freshclam clamdscan libclamav11t64
0 upgraded, 6 newly installed, 0 to remove and 16 not upgraded.
Need to get 9495 kB of archives.
After this operation, 38.3 MB of additional disk space will be used.
Get:1 http://ports.ubuntu.com/ubuntu-ports noble-updates/main arm64 clamav-base all 1.0.8+dfsg-0ubuntu0.24.04.1 [93.5 kB]
Get:2 http://ports.ubuntu.com/ubuntu-ports noble-updates/main arm64 libclamav11t64 arm64 1.0.8+dfsg-0ubuntu0.24.04.1 [4613 kB]
Get:3 http://ports.ubuntu.com/ubuntu-ports noble-updates/main arm64 clamav-freshclam arm64 1.0.8+dfsg-0ubuntu0.24.04.1 [96.7 kB]
Get:4 http://ports.ubuntu.com/ubuntu-ports noble-updates/main arm64 clamav-daemon arm64 1.0.8+dfsg-0ubuntu0.24.04.1 [211 kB]
Get:5 http://ports.ubuntu.com/ubuntu-ports noble-updates/main arm64 clamav arm64 1.0.8+dfsg-0ubuntu0.24.04.1 [4429 kB]
Get:6 http://ports.ubuntu.com/ubuntu-ports noble-updates/main arm64 clamdscan arm64 1.0.8+dfsg-0ubuntu0.24.04.1 [51.1 kB]
Fetched 9495 kB in 1s (7522 kB/s)
Preconfiguring packages ...
Selecting previously unselected package clamav-base.
(Reading database ... 88716 files and directories currently installed.)
Preparing to unpack .../0-clamav-base_1.0.8+dfsg-0ubuntu0.24.04.1_all.deb ...
Unpacking clamav-base (1.0.8+dfsg-0ubuntu0.24.04.1) ...
Selecting previously unselected package libclamav11t64:arm64.
Preparing to unpack .../1-libclamav11t64_1.0.8+dfsg-0ubuntu0.24.04.1_arm64.deb ...
Unpacking libclamav11t64:arm64 (1.0.8+dfsg-0ubuntu0.24.04.1) ...
Selecting previously unselected package clamav-freshclam.
Preparing to unpack .../2-clamav-freshclam_1.0.8+dfsg-0ubuntu0.24.04.1_arm64.deb ...
Unpacking clamav-freshclam (1.0.8+dfsg-0ubuntu0.24.04.1) ...
Selecting previously unselected package clamav-daemon.
Preparing to unpack .../3-clamav-daemon_1.0.8+dfsg-0ubuntu0.24.04.1_arm64.deb ...
Unpacking clamav-daemon (1.0.8+dfsg-0ubuntu0.24.04.1) ...
Selecting previously unselected package clamav.
Preparing to unpack .../4-clamav_1.0.8+dfsg-0ubuntu0.24.04.1_arm64.deb ...
Unpacking clamav (1.0.8+dfsg-0ubuntu0.24.04.1) ...
Selecting previously unselected package clamdscan.
Preparing to unpack .../5-clamdscan_1.0.8+dfsg-0ubuntu0.24.04.1_arm64.deb ...
Unpacking clamdscan (1.0.8+dfsg-0ubuntu0.24.04.1) ...
Setting up libclamav11t64:arm64 (1.0.8+dfsg-0ubuntu0.24.04.1) ...
Setting up clamav-base (1.0.8+dfsg-0ubuntu0.24.04.1) ...
Setting up clamav-freshclam (1.0.8+dfsg-0ubuntu0.24.04.1) ...
Setting up clamdscan (1.0.8+dfsg-0ubuntu0.24.04.1) ...
Setting up clamav-daemon (1.0.8+dfsg-0ubuntu0.24.04.1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/clamav-daemon.service → /usr/lib/systemd/system/clamav-daemon.service.
Created symlink /etc/systemd/system/sockets.target.wants/clamav-daemon.socket → /usr/lib/systemd/system/clamav-daemon.socket.
```

Figura 6.40: Instalación de ClamAV.

Tras la instalación, se detiene el servicio, se actualizan las firmas de virus y se reinicia el demonio para activar la protección.

```
● ventauris@tanorim:~$ sudo systemctl stop clamav-freshclam
● ventauris@tanorim:~$ sudo freshclam
ClamAV update process started at Sun Mar 23 21:05:49 2025
Sun Mar 23 21:05:49 2025 -> daily.cvd database is up-to-date (version: 27586, sigs: 2074246, f-level: 90, builder: raynman)
Sun Mar 23 21:05:49 2025 -> main.cvd database is up-to-date (version: 62, sigs: 6647427, f-level: 90, builder: sigmgr)
Sun Mar 23 21:05:49 2025 -> bytecode.cvd database is up-to-date (version: 335, sigs: 86, f-level: 90, builder: raynman)
● ventauris@tanorim:~$ sudo systemctl start clamav-freshclam
```

Figura 6.41: Actualización de firmas y reinicio del servicio ClamAV.

6.5. Generación de un script para la instalación automática

6.5.1. Esquema de red de monitorización con R-SNORT

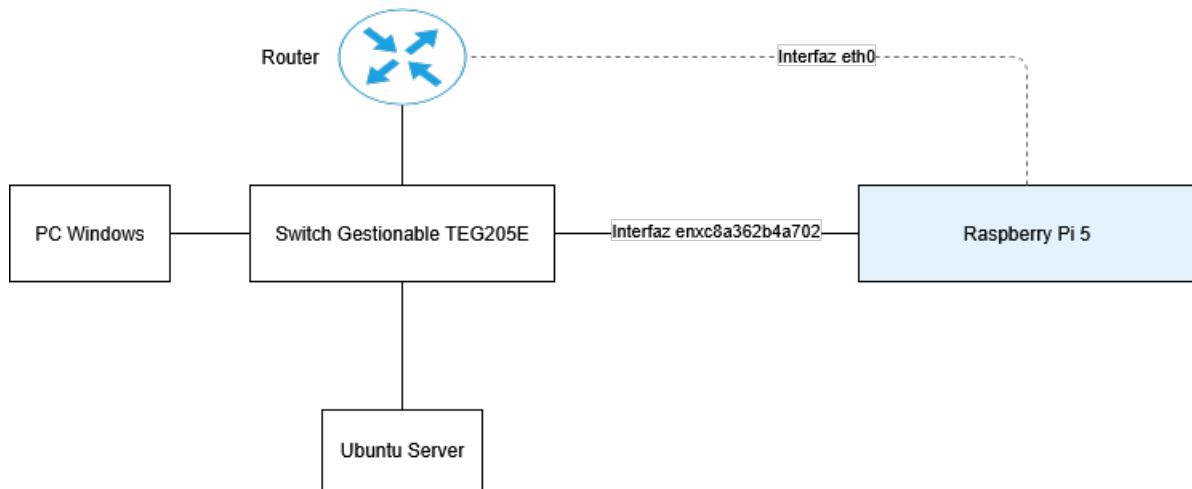


Figura 6.42: Esquema de red real del proyecto.

El *Port Mirroring* se configura para duplicar el tráfico de los puertos 1, 2 y 3 (los dispositivos productivos y el router) hacia el puerto 4, permitiendo que la interfaz `enxc8a362b4a702` de la Raspberry Pi capture de forma pasiva y en tiempo real todo el tráfico interno y externo de la red.

Objetivo de este diseño

El objetivo principal de esta arquitectura es que **R-SNORT** funcione como un sistema de detección de intrusiones pasivo (*Network-based IDS*) sin interferir en el flujo de los datos. Para ello, deben cumplirse dos condiciones técnicas fundamentales:

1. La interfaz utilizada para la captura debe operar en **modo promiscuo**, sin dirección IP asignada, actuando como una sonda pasiva totalmente invisible para el resto de la red.
2. El switch debe soportar **duplicación de tráfico (mirroring)** para garantizar que todo el tráfico relevante sea reenviado correctamente al puerto de análisis.

Justificación técnica

Este tipo de arquitectura es común en entornos profesionales donde se desea realizar *inspección profunda de paquetes (Deep Packet Inspection)* sin introducir latencia ni crear puntos únicos de fallo en la red [32]. Además, permite mantener completamente aislado el sistema IDS del resto de dispositivos, lo cual incrementa la seguridad del propio entorno de monitorización.

La elección de una **Raspberry Pi 5** se justifica por su bajo consumo energético, su arquitectura ARM eficiente, y su capacidad para operar con interfaces Ethernet Gigabit, siempre que se utilice un adaptador USB 3.0 para incorporar una segunda interfaz de red.

Alternativas descartadas

Se valoró la posibilidad de implementar una arquitectura enrutada —donde todo el tráfico pasara a través de la Raspberry Pi—, sin embargo, esta solución presentaba varios inconvenientes:

- Aumentaría la latencia y la carga de red para todos los dispositivos conectados.
- Requiere una configuración compleja de **NAT** o **bridging** en la Raspberry Pi.
- Introduce un punto único de fallo que compromete la disponibilidad de la red en caso de errores.

Por tanto, la arquitectura basada en un **switch con mirroring** y una sonda pasiva resulta significativamente más robusta, profesional y realista para el entorno de una pequeña o mediana empresa (**PYME**).

6.5.2. Primera fase del script automático

La instalación manual de *Snort 3*, especialmente en dispositivos embebidos como la Raspberry Pi, puede convertirse en un proceso complejo y propenso a errores debido a la cantidad de dependencias, compilaciones desde código fuente y configuraciones específicas del sistema. Por este motivo, se ha desarrollado un script de instalación automática denominado **R-SNORT INSTALLER**, que encapsula todo el proceso de despliegue de manera modular, robusta y reproducible.

Motivación y diseño modular

Automatizar la instalación de herramientas en ciberseguridad como Snort es un paso habitual para garantizar la repetibilidad de los entornos de prueba, minimizar el error humano y facilitar la portabilidad entre dispositivos. Esta filosofía está alineada con los principios de *Infrastructure as Code* (IaC), promovida en múltiples estudios recientes sobre automatización segura.

El script propuesto sigue una arquitectura modular, dividiendo su lógica en funciones independientes organizadas en ficheros temáticos:

- **r-snort_installer.sh**: Núcleo del sistema, orquesta las fases del proceso de instalación desde una perspectiva modular, gestionando logs, permisos, selección de interfaz y llamadas a los demás módulos. Véase la Figura B.1 para una muestra del flujo de ejecución principal.
- **core.sh**: Módulo responsable de los mensajes de log, errores y el banner de bienvenida, garantizando una salida legible y profesional para el usuario. Véase la Figura B.2.
- **checks.sh**: Este módulo verifica que el script se ejecute como root y solicita al usuario que seleccione la interfaz de red sobre la que actuará Snort. Representación visual en la Figura B.3.
- **dependencies.sh**: Instala los paquetes de compilación y bibliotecas necesarios en entornos Debian/Ubuntu. La Figura B.4 muestra un fragmento de esta lógica.

- **build_from_source.sh**: Gestiona y compila desde fuentes los componentes adicionales (luajit, daq, openssl, etc.). También maneja errores comunes de compatibilidad, como se muestra a continuación:

Listing 6.1: Corrección de versiones incompatibles de xz/liblzma

```

1      if ! xz -t "$archivo" 2>&1 | grep -qv 'version<='`XZ_'; then
2          apt-get install --reinstall -y xz-utils liblzma5 liblzma-
3              ↪ dev
        fi

```

- **install_snort.sh**: Compila Snort 3.1.84.0 con correcciones específicas para evitar fallos con versiones recientes de OpenSSL y desactiva el soporte NUMA. Véase la Figura B.6.
- **configure_snort.sh**: Prepara la configuración final de Snort, cargando reglas, scripts .lua, y desplegando el servicio con **systemd**. Representado en la Figura B.7.
- **swap.sh**: Activa dinámicamente un archivo de swap temporal si se detecta menos de 1.5 GB de RAM, evitando fallos de compilación en la Raspberry Pi. Ver la Figura B.8.
- **stats.sh**: Recopila y presenta estadísticas del sistema tras la instalación, incluyendo versiones de Snort y ClamAV, recursos utilizados y configuración de red. Ilustrado en la Figura B.9.

Características de robustez y control de errores

El script implementa buenas prácticas de scripting en Bash, incluyendo:

- Uso de **set -euo pipefail** para evitar ejecuciones silenciosas tras errores.
- Captura de errores con **trap** para identificar fallos que desestabilicen o anulen la instalación.
- Registro detallado en **/var/log/snort_install.log** mediante redirección directa del flujo de salida.

Además, cada función de especial importancia o con condiciones de carrera se acompaña de validaciones explícitas. Por ejemplo, antes de descomprimir archivos **.tar.gz** o **.tar.xz**, se comprueba su integridad con **gzip -t** o **xz -t**, y se mitigan fallos comunes de **liblzma** con reinstalación forzada si es necesario.

Listing 6.2: Validación e instalación segura de paquetes .xz

```

1      if ! xz -t "$archivo" 2>&1 | grep -qv 'version<='`XZ_'; then
2          apt-get install --reinstall -y xz-utils liblzma5 liblzma-dev
3      fi

```

Automatización total del servicio Snort

Una vez instalado, **Snort** es configurado automáticamente como un servicio **systemd**. El script genera dinámicamente el archivo **snort.service** y lo enlaza al directorio correcto, con los parámetros adecuados de reinicio automático, límites de recursos y uso de la interfaz de red seleccionada interactivamente por el usuario.

Listing 6.3: Sección relevante del systemd generado

```
1 [Service]
2 ExecStart=/usr/local/snort/bin/snort -c /usr/local/snort/etc/snort/snort
   ↪ .lua -i eth0 -A alert_fast
3 Restart=always
4 User=root
5 Group=root
```

Mitigación de limitaciones de hardware

Para abordar las limitaciones de memoria RAM propias de sistemas como la ARM (no presente en Raspberry Pi 5 de 8GB de RAM), el script evalúa si el sistema posee menos de 1.5GB de RAM y en ese caso crea un archivo de swap temporal de 2GB:

```
1 if [ "$mem_kb" -lt 1500000 ]; then
2   fallocate -l 2G /swapfile_snort
```

Esta característica permite compilar Snort de forma estable incluso en configuraciones reducidas, evitando fallos durante la fase de `make -j$(nproc)`.

Instalación adicional de ClamAV y control antivirus

Dado que el proyecto R-Snort está pensado como un sistema IDS autónomo para pequeñas y medianas empresas, se incluyó también la instalación automática de ClamAV, permitiendo disponer de un escáner antivirus activo desde el arranque. La instalación es completamente silenciosa y actualiza su base de firmas mediante `freshclam`.

Resultados e impacto del diseño

El instalador automático consigue reducir a menos de 15 minutos la instalación completa de Snort, desde dependencias hasta configuración funcional. Esta reducción de complejidad aumenta la fiabilidad y la reproducibilidad del entorno, dos factores muy demandados para sistemas de detección de intrusos.

Además, se ha verificado que, tras una ejecución limpia del instalador, Snort queda operativo como servicio persistente, escuchando tráfico en tiempo real y generando alertas sobre tráfico malicioso, todo ello sin intervención posterior del usuario.

6.5.3. Transición a paquete .deb

Con el objetivo de profesionalizar el despliegue de **R-SNORT** y facilitar su uso en entornos reales, se reestructuró el proyecto como un paquete Debian estándar. Esta decisión permite que el propio sistema operativo gestione la instalación, dependencias, configuración y servicio a través de scripts como `postinst` (ver Figura B.10), `prerm` (ver Figura B.11) y `postrm` (ver Figura B.12), alineándose con las buenas prácticas del ecosistema Debian [33].

El resultado es una arquitectura empaquetada bajo el nombre `r-snort-deb.deb`, que se instala fácilmente mediante un único comando y permite revertir el proceso de instalación sin dejar rastros en el sistema.

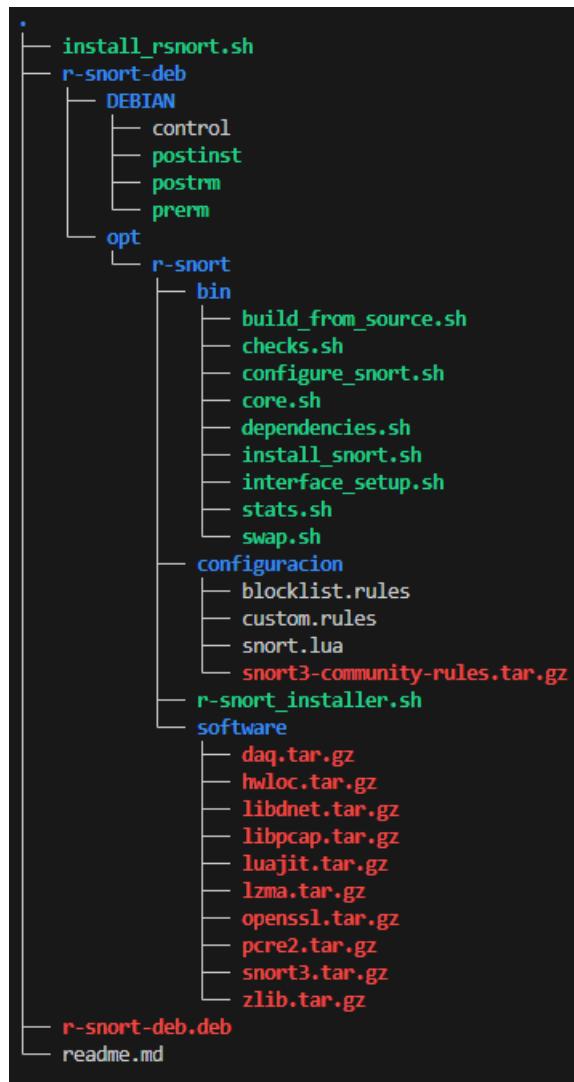


Figura 6.43: Estructura del proyecto reempaquetado como archivo .deb.

Mejoras introducidas

La transición al formato `.deb` permitió incorporar una serie de mejoras que anteriormente solo podían aplicarse de forma manual:

- **Selección dinámica de interfaz:** Se desarrolló un script de instalación previa, `install_rsnort.sh`

(ver Figura B.13), que detecta automáticamente las interfaces Ethernet disponibles y permite al usuario seleccionar la que se conectará al puerto espejo del switch. La elección se guarda en `/etc/rsnort_iface` para que pueda ser utilizada posteriormente por otros componentes del sistema.

- **Modo promiscuo y sin IP:** Se creó el script `interface_setup.sh` (ver Figura B.14) que activa automáticamente la interfaz seleccionada, elimina cualquier dirección IP existente para evitar conflictos, y la configura en modo *promiscuo*. Este paso es esencial para que Snort pueda inspeccionar todo el tráfico redirigido desde el switch sin interferir en el funcionamiento de la red.

Listing 6.4: Activación automática del modo promiscuo en `interface_setup.sh`

```

1      if [[ "$state" != "UP" ]]; then
2          ip link set dev "$iface" up
3      fi
4
5      ip addr flush dev "$iface"
6      ip link set "$iface" promisc on

```

- **Despliegue como servicio systemd:** Se redefinió el proceso de inicio de Snort mediante una unidad personalizada de `systemd`, lo que permite un arranque automático, supervisión y control del servicio tras cada reinicio del sistema.
- **Fortalecimiento con ClamAV:** Para añadir una capa extra de protección, se integró la instalación y activación automática del antivirus ClamAV. Esto es especialmente útil en entornos donde Snort pueda capturar tráfico de archivos potencialmente contaminados o vectores de malware.

Justificación del rediseño

Este rediseño responde no solo a criterios técnicos, sino también a aspectos de usabilidad. En entornos reales, como pequeñas y medianas empresas (PYMEs), que desean adoptar soluciones *NIDS*, es esencial que la herramienta pueda instalarse, configurarse y mantenerse sin requerir conocimientos avanzados en administración de sistemas. Gracias a esta transición, **R-SNORT** puede considerarse una solución *plug-and-play*:

1. El usuario ejecuta `install_rsnort.sh` (ver Figura B.13).
2. Selecciona la interfaz conectada al switch mediante un menú interactivo.
3. El paquete se instala y configura automáticamente, sin intervención adicional.

Esta arquitectura modular empaquetada sigue las mejores prácticas de la ingeniería de software y permite escalar, actualizar o personalizar R-SNORT en futuras versiones con mínima fricción.

Resultado final

Gracias a la evolución hacia un paquete .deb, **R-SNORT** se consolida como una herramienta más robusta, mantenible y amigable con el usuario final. Es importante recordar que este proyecto busca ofrecer una solución eficaz, ligera y accesible para redes SOHO (*Small Office / Home Office*). Con un enfoque modular interno y una experiencia de instalación unificada, el sistema mantiene toda su capacidad de inspección avanzada sin renunciar a la simplicidad en su despliegue.

7. Caso práctico: utilización de R-Snort

7.1. Entorno de trabajo

El sistema R-Snort ha sido desplegado en un entorno de red realista a pequeña escala, compuesto por los siguientes elementos:

- Una Raspberry Pi 5 con sistema operativo Ubuntu Server, encargada de ejecutar el paquete automatizado con Snort.
- Un switch gestionable Tenda TEG205E, configurado para replicar todo el tráfico de red mediante la funcionalidad de *port mirroring*.
- Un adaptador UGREEN USB 3.0 a Ethernet Gigabit conectado a la Raspberry, que actúa como interfaz de captura para Snort.
- Un router doméstico (F@st 5670) que proporciona conectividad a Internet a toda la red.
- Dos equipos cliente: un PC con Windows y otro con Ubuntu Server, ambos conectados al switch.

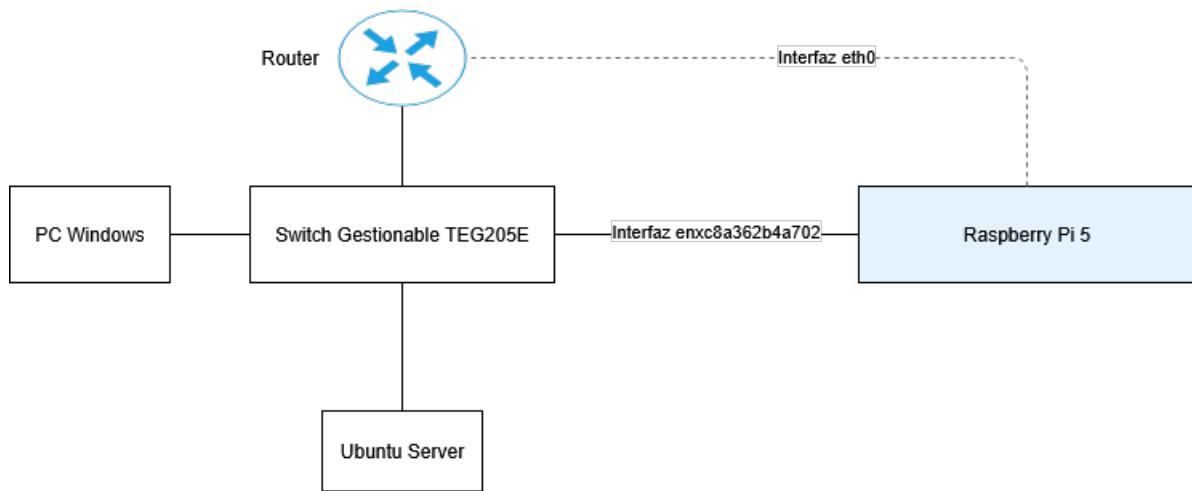


Figura 7.1: Esquema de red real del proyecto.

El puerto 4 del switch se ha configurado como destino del *mirroring*, recibiendo una copia del tráfico de los puertos 1 a 3 (correspondientes al router y los dos PCs). De esta manera, R-Snort es capaz de recibir una copia tanto del tráfico interno como del proveniente de Internet. La Raspberry escucha dicho tráfico a través del adaptador UGREEN, el cual se configura automáticamente en modo promiscuo, sin dirección IP asignada, para evitar interferencias en la red.

7.2. Instalación

La instalación se ha simplificado al máximo para que cualquier usuario pueda desplegar R-Snort sin conocimientos técnicos avanzados. El proceso consta de dos fases principales:

1. Ejecución del script `install_rsnort.sh`, que actualiza paquetes, instala dependencias y solicita al usuario que seleccione la interfaz de red conectada al switch.
2. Instalación del paquete `.deb`, que compila Snort y sus dependencias, configura la interfaz en modo promiscuo, crea el servicio en `systemd`, e inicia automáticamente el demonio de Snort.

A continuación, se presentan algunas capturas del proceso, incluyendo tanto la preparación del hardware como de la instalación software:

La Raspberry Pi 5 cuenta con dos interfaces Ethernet: una conectada directamente al router para acceder a Internet desde fuera de la red local (`eth0`), y otra a través de un adaptador USB 3.0 (UGREEN) destinada exclusivamente a capturar tráfico de red (`enxc8a362b4a702`). Esta segunda interfaz es la que recibe todo el tráfico replicado desde el switch mediante *port mirroring*.



Figura 7.2: Raspberry Pi conectada con dos interfaces Ethernet.

Asignación de dispositivos al switch

El switch gestionable Tenda permite configurar el tráfico de cada puerto. En este caso, se conectaron el router, los dos PCs (Windows y Ubuntu Server) y la Raspberry Pi. Esta última se conecta a un puerto configurado como espejo de los demás, lo que le permite capturar el tráfico replicado.

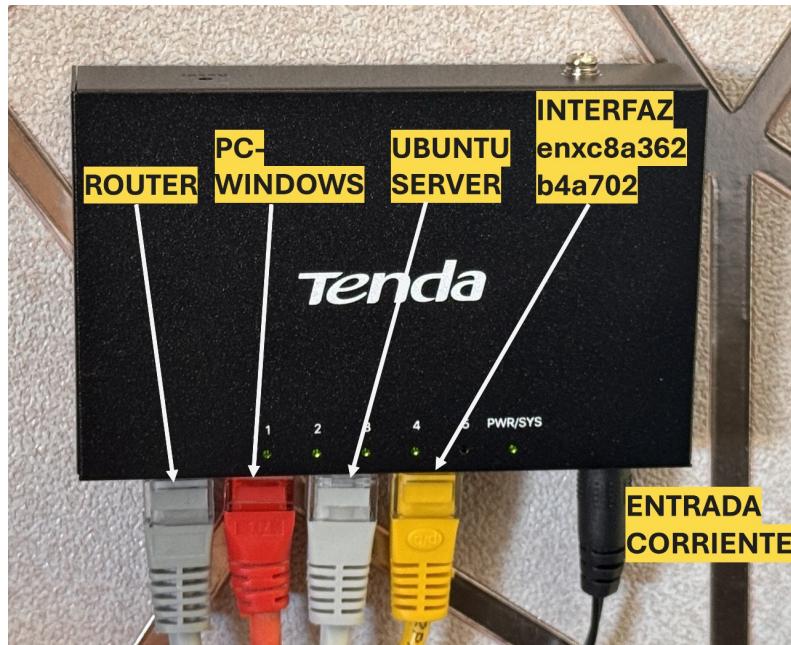


Figura 7.3: Conexiones físicas al switch gestionable.

Identificación del entorno LAN

Desde la interfaz web del router se pueden visualizar todos los dispositivos conectados. Esto permite verificar que tanto el switch como la Raspberry Pi están correctamente integrados en la red y responden a su dirección IP.

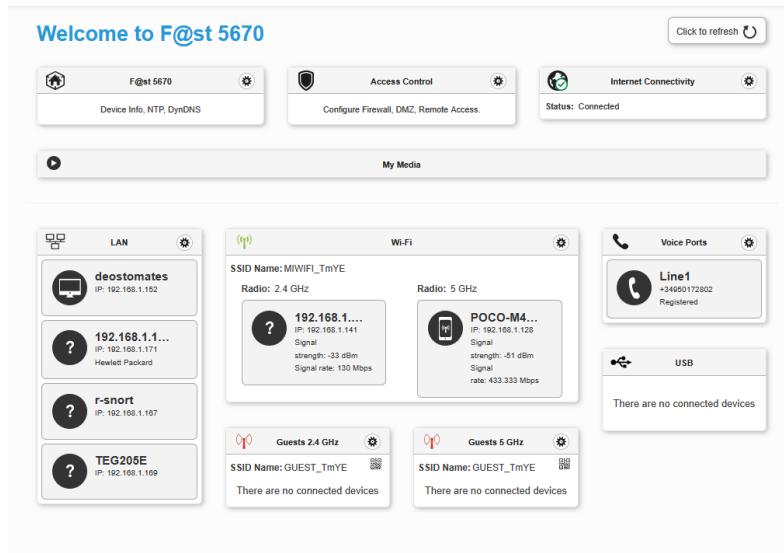


Figura 7.4: Dispositivos visibles en la red doméstica.

Acceso al switch Tenda

Para configurar el port mirroring es necesario acceder al panel web del switch. Para ello, se localiza su dirección IP desde el router (192.168.1.169) y se accede a ella mediante un navegador web.

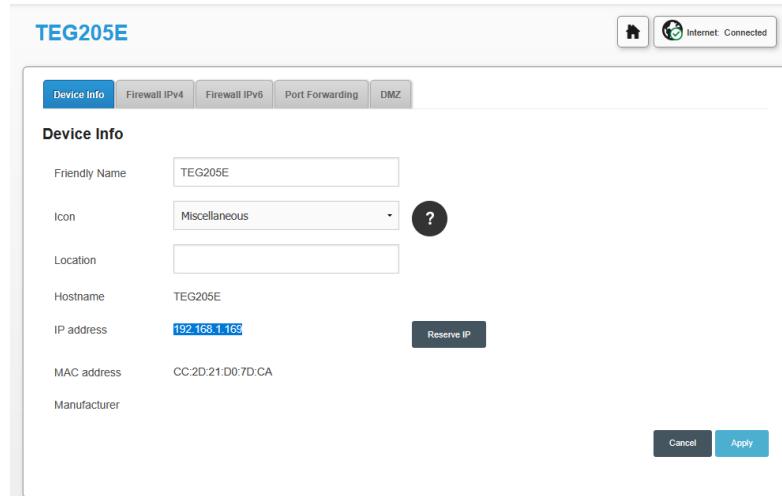


Figura 7.5: IP local del switch detectada desde el router.

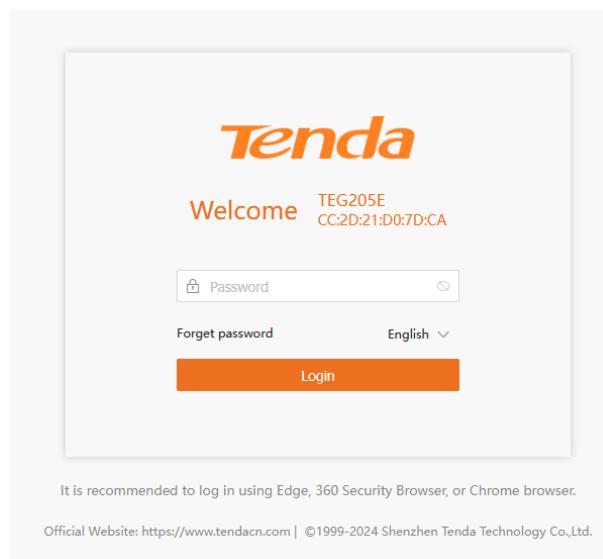


Figura 7.6: Pantalla de acceso al panel de administración.

Seguridad inicial del dispositivo

En el primer acceso al switch, el sistema solicita el cambio obligatorio de la contraseña predeterminada para mejorar la seguridad del entorno.

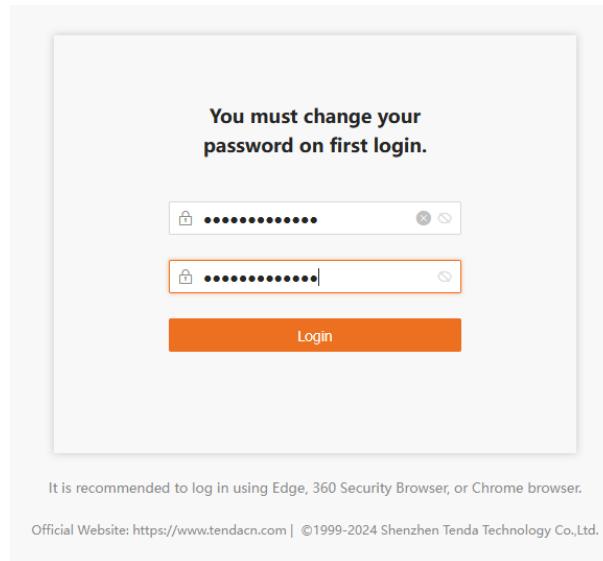


Figura 7.7: Cambio obligatorio de contraseña en el primer inicio.

Configuración avanzada del switch

Una vez autenticados, accedemos a la interfaz de administración, donde es posible configurar diversos parámetros del dispositivo. Desde esta sección se define el *port mirroring*, se habilitan o deshabilitan puertos y se monitoriza el estado de la red.

Port	Status	Mode	Speed/Duplex	Receiving Rate (Kbps)	Sending Rate (Kbps)	Network Extension
1	Enabled	Auto	1000M/FDX	0.00	0.00	Disabled
2	Enabled	Auto	1000M/FDX	0.00	0.00	Disabled
3	Enabled	Auto	1000M/FDX	0.00	0.00	Disabled
4	Enabled	Auto	1000M/FDX	0.00	0.00	Disabled
5	Enabled	Auto	--	0.00	0.00	Disabled

Figura 7.8: Vista general del panel de gestión del switch.

Activación del Port Mirroring

Desde el apartado **switching >port mirroring**, activamos esta funcionalidad para duplicar el tráfico de red. Se configura el switch para replicar el tráfico de los puertos donde están conectados el router y los PCs (puertos 1, 2 y 3) hacia el puerto 4, que está vinculado a la interfaz de análisis de la Raspberry Pi. Esto permite que Snort reciba una copia completa del tráfico local y externo.

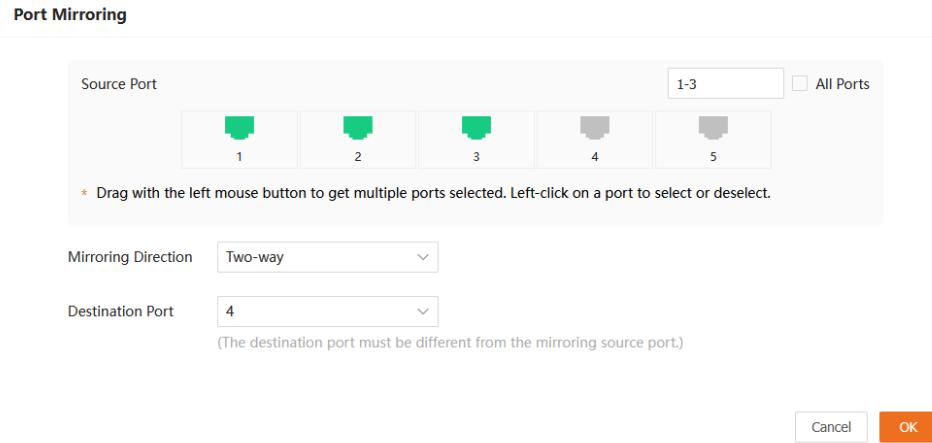


Figura 7.9: Verificación del estado de los puertos tras la configuración.

Una vez la Raspberry Pi 5 sea capaz de recibir todo el tráfico en modo espejo por la interfaz correspondiente de toda la red local al mismo tiempo que tiene conexión a internet externo por la interfaz predeterminada del sistema se pone en marcha el paquete automatizado para instalar la Snort junto con la configuración completa.

Finalmente, ejecutamos como administrador el script `installer_snort.sh` para iniciar la instalación de R-Snort. Se selecciona la interfaz de red correspondiente al puerto 4 del switch, que es el que recibe el tráfico duplicado [34]. Una vez finalizado el proceso, el sistema queda completamente operativo. El código fuente del instalador puede consultarse en el Anexo A.

```

snort-lab:~$ snort-/-r-snort-automatico$ sudo ./install_rsnort.sh
[!] [R-SNORT] Actualizando lista de paquetes...
Hit:1 http://ports.ubuntu.com/ubuntu-ports noble InRelease
Hit:2 http://ports.ubuntu.com/ubuntu-ports noble-updates InRelease
Hit:3 http://ports.ubuntu.com/ubuntu-ports noble-security InRelease
Get:4 http://ports.ubuntu.com/ubuntu-ports noble-security InRelease [126 kB]
Get:5 http://ports.ubuntu.com/ubuntu-ports noble-security/universe arm64 Packages [800 kB]
Get:6 http://ports.ubuntu.com/ubuntu-ports noble-security/universe Translation-en [179 kB]
Fetched 118 kB in 1s (1200 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading information... Done
79 packages can be upgraded. Run 'apt list --upgradable' to see them.
[!] [R-SNORT] Instalando dependencias...
Reading package lists... Done
Building dependency tree... Done
Reading information... Done
batch is already the newest version (5.2.21-2ubuntu4).
build-essential is already the newest version (12.10ubuntu1).
libpcap-dev is already the newest version (1.10.4-4.1ubuntu3).
xz-utils is already the newest version (5.6.1+really5.4.5-1build0.1).
liblzo2-dev is already the newest version (5.6.1+really5.4.5-1build0.1).
clamav is already the newest version (1.0.84fsg-ubuntu0.24.04.1).
ClamAV is already the newest version (1.0.84fsg-ubuntu0.24.04.1).
0 upgraded, 0 newly installed, 0 to remove and 79 not upgraded.
[!] [R-SNORT] Dependencias instaladas...
[!] [R-SNORT] Buscando interfaces Ethernet disponibles...
Interfaces disponibles:
[0] eth0
[1] enx8a362b4a702
[?] Elige la interfaz para analizar tráfico (la de switch): 1
[!] Interfaz seleccionada: enx8a362b4a702

[!] [R-SNORT] Instalando paquete .deb...
(Reading database ... 86623 files and directories currently installed.)
Preparing to unpack r-snort-deb.deb ...
[!] [R-SNORT] Deteniendo servicio r-snort antes de eliminar el paquete...
[!] [R-SNORT] Snort detenido (1.0) ... (si estaba en ejecución).
Unpacking r-snort (1.0) over (1.0) ... (sin limpiar).
[!] [R-SNORT] postrm ejecutado con modo 'upgrade' (sin limpiar).
Setting up r-snort (1.0) ...
[!] [R-SNORT] Iniciando instalación automática...
R-SNORT
Snort 3 Installer
[*] Instalación R-SNORT iniciado
[*] Interfaz: elegida desde configuración previa: enx8a362b4a702
[*] Verificando estado de la interfaz enx8a362b4a702...
[*] La interfaz enx8a362b4a702 ya está UP.
[*] enx8a362b4a702 ya está en modo promiscuo.
[!] Interfaz enx8a362b4a702 preparada para análisis de red.

[!] Servicio Snort configurado y activo.
[*] Configurador de Snort ejecutado correctamente.
[*] Comprobando estado del servicio Snort...
[*] Snort está activo y habilitado.

[*] Resumen del sistema tras la instalación:
[!] Hostname: r-snort
[!] Uptime: up 2 hours, 7 minutes
[!] RAM usada: 2.0Gi / 7.8Gi
[!] Swap activa: No
[!] Espacio raíz: 9.5G usados de 29G
[!] CPU: Cortex-A76 (4 núcleos)
[!] Snort versión: 3.1.84.0
[!] ClamAV versión: 1.0.8/27594/Mon
[!] Interfaz activa: enx8a362b4a702

[!] Snort 3 está en ejecución en la interfaz: enx8a362b4a702.
[!] [R-SNORT] Instalación completada.
[*] Processing triggers for liblc-bin (2.39-0ubuntu8.4) ...
[!] [R-SNORT] Instalación completa.

```

Figura 7.10: *

(a) Ejecución de R-Snort.

Figura 7.11: *

(b) Finalización del instalador.

Figura 7.12: Proceso completo de instalación automática de R-Snort.

7.3. Utilización y pruebas

Una vez completada la instalación, se procedió a la validación funcional del sistema y a su evaluación de rendimiento. El objetivo principal es demostrar que R-Snort es capaz de monitorear eficazmente el tráfico de red y detectar amenazas sin sobresaturar los recursos del sistema.

7.3.1. Pruebas de rendimiento

Metodología

Para medir el rendimiento de R-Snort, se utilizó la herramienta `dstat`, que permite monitorear en tiempo real diversos parámetros del sistema, como el uso de CPU, memoria, red y disco. Se realizaron dos sesiones de captura de datos:

- Una con Snort encendido y funcionando en modo *inline*, procesando el tráfico en tiempo real.
- Otra con Snort completamente apagado, lo que permitió establecer una línea base del consumo del sistema sin el IDS.

Cada sesión de monitorización generó un archivo CSV con un intervalo de muestreo de 1 segundo, que posteriormente fue procesado para generar gráficas comparativas. Se utilizó Python, junto con las bibliotecas `pandas` y `matplotlib`, para el análisis y visualización de datos.

Resultados e interpretación de gráficas

A continuación, se presentan las gráficas obtenidas:

Prueba de red: Bytes recibidos por segundo

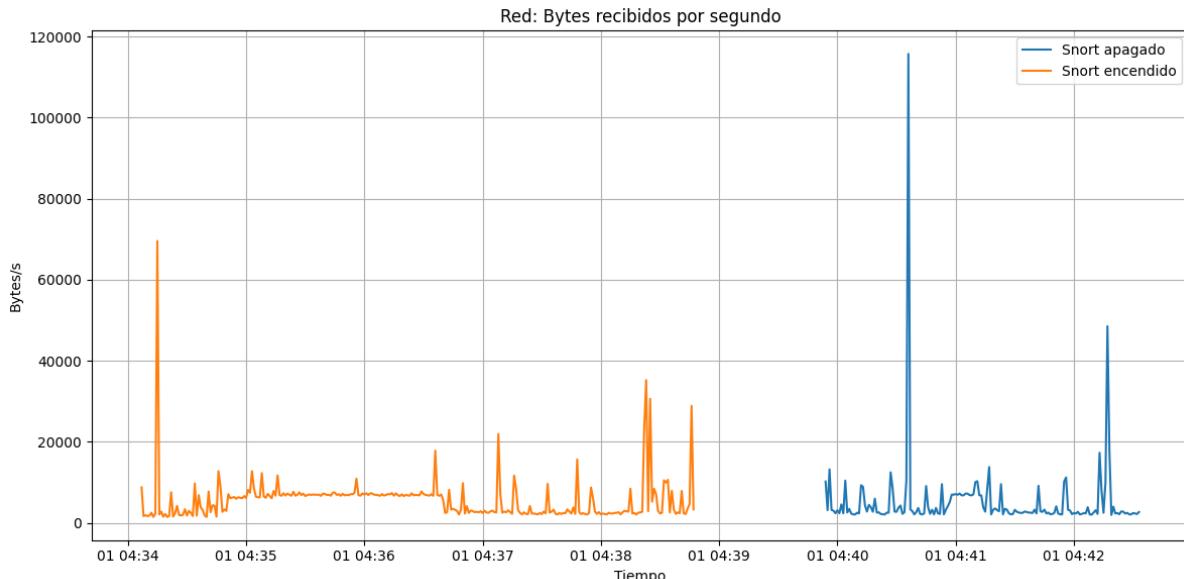


Figura 7.13: Red: Bytes recibidos por segundo

Análisis: Se observa un patrón de tráfico más irregular y, en ocasiones, más intenso cuando Snort está apagado. Con Snort activo, el tráfico parece más contenido, probablemente porque parte de los paquetes son descartados o procesados de forma más lenta. Esto sugiere una influencia directa del IDS sobre el tráfico recibido.

Prueba de red: Bytes enviados por segundo

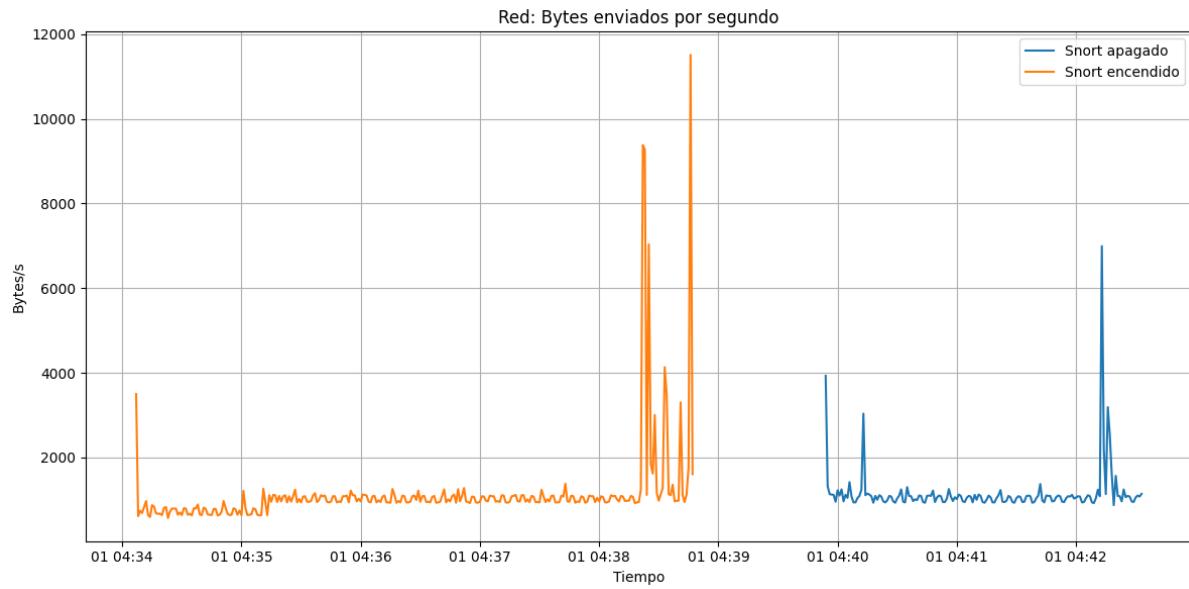


Figura 7.14: Red: Bytes enviados por segundo

Análisis: El patrón de tráfico enviado es bajo en ambas sesiones, aunque con Snort encendido se detecta un ligero aumento en los picos. Esto puede deberse a respuestas generadas por Snort tras detectar actividad anómala, como en pruebas de copia de datos o escaneo de puertos.

Prueba de rendimiento: Context switches por segundo

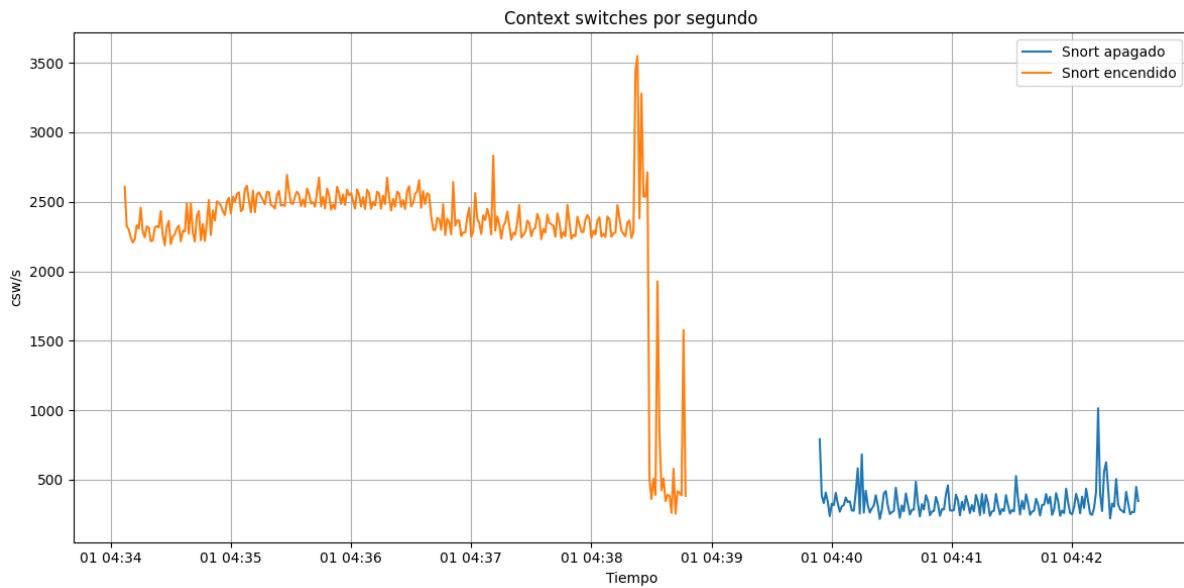


Figura 7.15: Context switches por segundo

Análisis: El número de cambios de contexto se incrementa notablemente con Snort activo. Este comportamiento es esperable, ya que el análisis en tiempo real de paquetes obliga al sistema a alternar con mayor frecuencia entre procesos.

Prueba de uso de CPU: Tiempo de inactividad

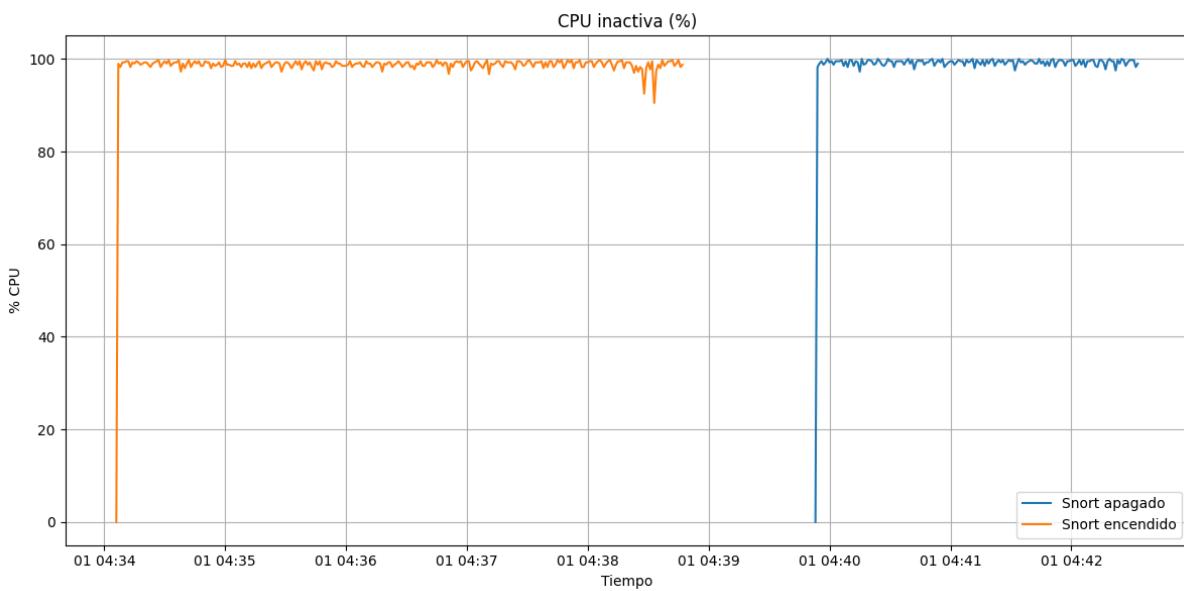


Figura 7.16: CPU inactiva (%)

Análisis: La CPU permanece prácticamente inactiva cuando Snort está apagado. Al activarlo, se reduce levemente el porcentaje de inactividad, indicando un impacto ligero pero constante sobre el procesador.

Prueba de uso de CPU: Espacio de sistema

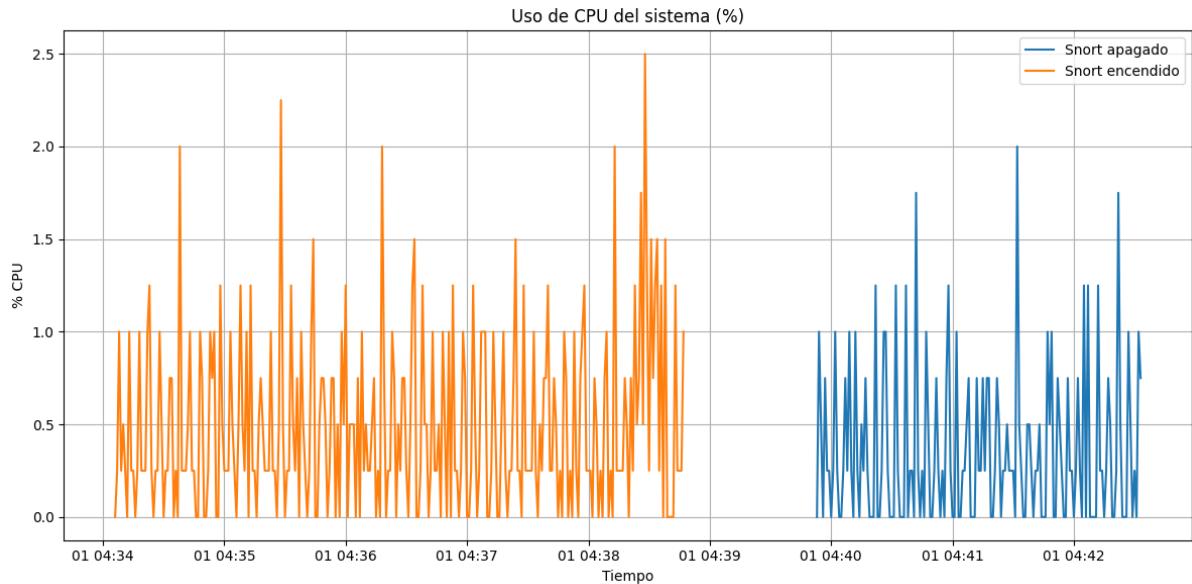


Figura 7.17: Uso de CPU del sistema (%)

Análisis: Se observa un aumento en el uso de CPU en espacio de sistema con Snort activo. Este comportamiento es coherente con el hecho de que el IDS opera en niveles bajos, haciendo uso intensivo de bibliotecas como `libpcap` para la captura de paquetes.

Prueba de uso de CPU: Espacio de usuario

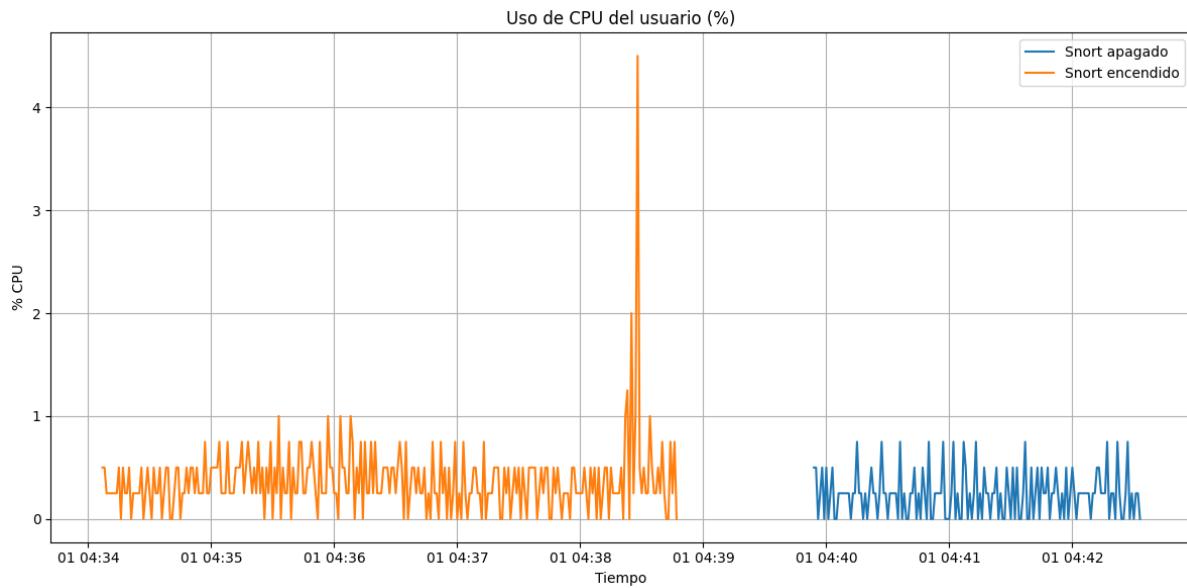


Figura 7.18: Uso de CPU del usuario (%)

Análisis: El impacto en el espacio de usuario es moderado. Aunque no se alcanzan valores elevados, la diferencia entre tener Snort encendido o apagado es clara, reflejando la carga asociada al motor de detección.

Prueba de rendimiento en disco: Escritura/Lectura en B/s

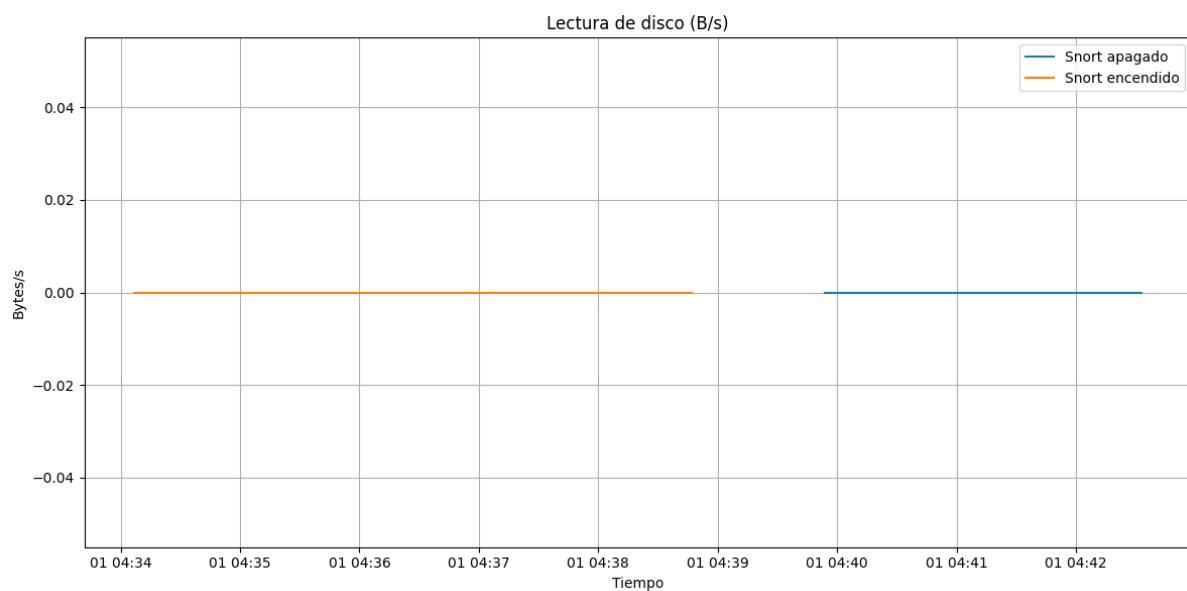


Figura 7.19: Lectura de disco (B/s)

Análisis: No se detectan lecturas de disco relevantes en ninguna de las dos sesiones. Este resultado es esperado, dado que Snort opera principalmente en memoria RAM y no realiza accesos intensivos a disco en su modo de funcionamiento habitual.

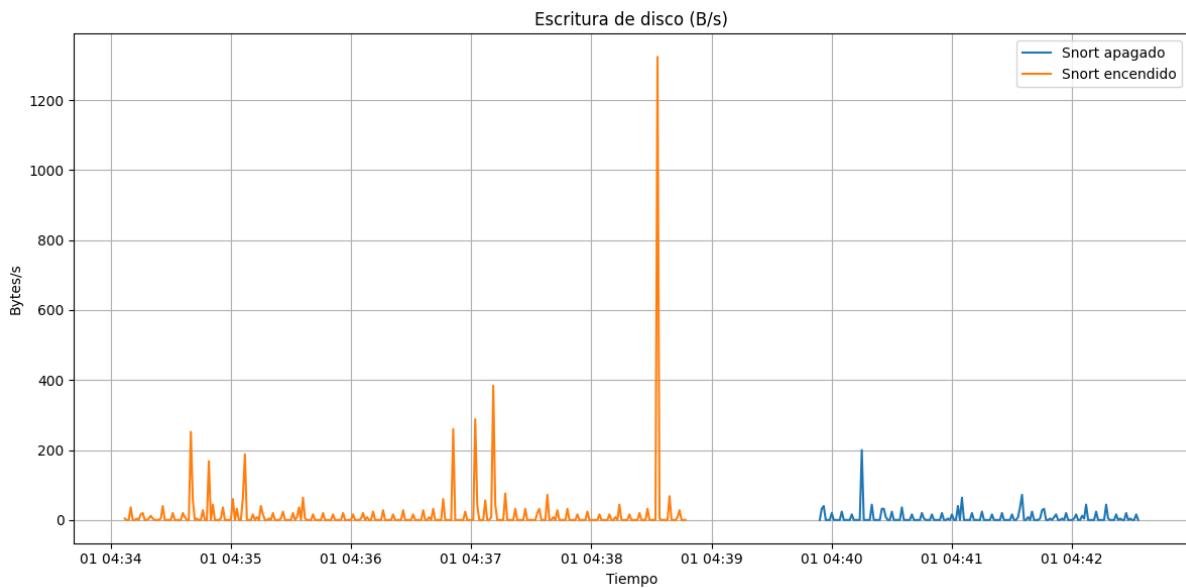


Figura 7.20: Escritura de disco (B/s)

Análisis: Se registra un aumento significativo en la escritura en disco cuando Snort está activo, correspondiente a la generación de logs y alertas de seguridad.

Prueba de uso de memoria RAM

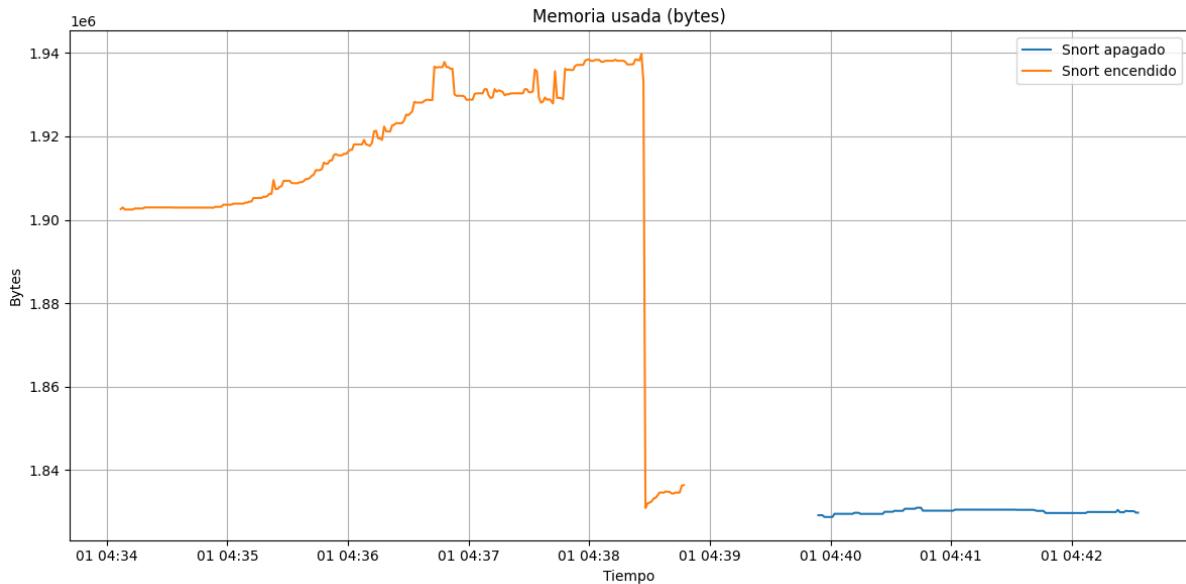


Figura 7.21: Memoria usada (bytes)

Análisis: El uso de memoria crece de forma progresiva cuando Snort está encendido. Esto se debe a la carga de reglas, buffers de captura y estructuras internas del IDS. Al desactivarlo, el consumo de RAM disminuye considerablemente.

En conjunto, estos resultados permiten concluir que R-Snort es eficiente y tiene un impacto reducido sobre el sistema en términos de consumo de recursos. No obstante, su actividad es claramente detectable, lo cual es esperable para un sistema de detección de intrusos activo y en tiempo real.

Las pruebas de rendimiento se han enfocado en dos métricas principales:

- **Carga del sistema:** Se monitorizó el uso de CPU y RAM bajo escenarios de carga de red baja, media y alta. Esto se realizó utilizando herramientas como `htop`, `free -h` y el script de estadísticas internas de R-Snort.
- **Capacidad de análisis:** Se evaluó la cantidad de paquetes por segundo que Snort puede procesar sin pérdidas ni errores, mediante herramientas de generación de tráfico como `iperf3` y `ping flood` desde los dos PCs.

Para poner estos datos en contexto, se compararon con estudios previos presentes en la literatura sobre el rendimiento de sistemas IDS en dispositivos embebidos [35]. Los resultados obtenidos confirman que R-Snort ofrece un rendimiento competitivo incluso en hardware de bajo consumo.

7.3.2. Pruebas

Se han llevado a cabo distintas pruebas funcionales para verificar el correcto comportamiento del sistema:

- **Detección de escaneos:** Se lanzó un escaneo Nmap desde el PC con Ubuntu Server al equipo con Windows, generándose correctamente alertas visibles en el panel web.
- **Generación de tráfico legítimo:** Navegación web, transferencia de archivos y conexión SSH, asegurando que Snort discrimina adecuadamente entre tráfico benigno y potencialmente malicioso.
- **Visualización en el panel web:** Las alertas generadas por Snort se muestran en tiempo real en el dashboard desarrollado con Angular y Spring Boot.

Estas pruebas confirman que el sistema funciona de forma autónoma, detecta tráfico relevante sin incurrir en falsos positivos masivos y expone la información de manera clara y accesible para el usuario final.

A continuación, se presentan pruebas reales ejecutadas en el entorno definido anteriormente (ver sección de topología de red), en las que R-Snort demuestra su capacidad de detección a través de reglas personalizadas y comunitarias. Cada caso incluye la alerta generada y una breve explicación técnica.

1. PINGs malformados y tráfico ICMP inusual

Desde el equipo Ubuntu Server (192.168.1.171) se ejecutó un ping continuo hacia Google DNS (8.8.8.8):

```
ping 8.8.8.8
```

Snort generó múltiples alertas relacionadas con tráfico ICMP, incluyendo paquetes tipo Unix ping, Echo reply y PINGs anómalos. Esto confirma el correcto funcionamiento de los preprocesadores **stream**, **icmp** y la inspección de TTLs habilitada en **stream_ip**.

Configuración personalizada validada: ajuste de parámetros como **min_ttl**, **max_overlaps** y **policy** en el bloque **stream_ip**.

```
04/01/03:16:37.235460 [**] [1:408:8] "PROTOCOL-ICMP Echo Reply" [**] [Classification: Misc activity] [Priority: 3] {ICMP} 8.8.8.8 -> 192.168.1.171
04/01/03:16:37.249477 [**] [1:366:11] "PROTOCOL-ICMP PING Unix" [**] [Classification: Misc activity] [Priority: 3] {ICMP} 192.168.1.171 -> 8.8.8.8
04/01/03:16:37.249477 [**] [1:29456:3] "PROTOCOL-ICMP Unusual PING detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 192.168.1.171 -> 8.8.8.8
04/01/03:16:37.249477 [**] [1:384:8] "PROTOCOL-ICMP PING" [**] [Classification: Misc activity] [Priority: 3] {ICMP} 192.168.1.171 -> 8.8.8.8
04/01/03:16:37.251465 [**] [1:408:8] "PROTOCOL-ICMP Echo Reply" [**] [Classification: Misc activity] [Priority: 3] {ICMP} 8.8.8.8 -> 192.168.1.171
[]
```

Figura 7.22: Tráfico ICMP detectado por preprocesadores y reglas comunitarias.

2. Fugas de datos sensibles (emails y tarjetas)

Se evaluó el funcionamiento de las reglas definidas en **custom.rules**, simulando la filtración de datos confidenciales desde Ubuntu hacia el router:

```
curl -d "Mi email es prueba@ejemplo.com" http://192.168.1.1
curl -d "Mi tarjeta es 4111111111111111" http://192.168.1.1
```

Snort detectó ambas cadenas mediante expresiones regulares, generando alertas en tiempo real por detección de datos sensibles.

Configuración personalizada validada: uso de reglas propias con **pcre** y **flow**, integradas correctamente desde **custom.rules** en el archivo **snort.lua**.

```
04/01/03:21:35.194948 [**] [1:1000002:1] "Sensitive Data - Credit Card detected" [**] [Classification: Sensitive Data] [Priority: 2] {TCP} 192.168.1.171:55398 -> 192.168.1.1:80
04/01/03:21:35.692227 [**] [1:29456:3] "PROTOCOL-ICMP Unusual PING detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 192.168.1.169 -> 142.250.184.164
04/01/03:21:35.692227 [**] [1:384:8] "PROTOCOL-ICMP PING" [**] [Classification: Misc activity] [Priority: 3] {ICMP} 192.168.1.169 -> 142.250.184.164
04/01/03:21:35.710848 [**] [1:408:8] "PROTOCOL-ICMP Echo Reply" [**] [Classification: Misc activity] [Priority: 3] {ICMP} 142.250.184.164 -> 192.168.1.169
04/01/03:21:52.811907 [**] [1:1000001:1] "Sensitive Data - Email detected" [**] [Classification: Sensitive Data] [Priority: 2] {TCP} 192.168.1.171:40916 -> 192.168.1.1:80
```

Figura 7.23: Alertas generadas por reglas personalizadas ante filtraciones de datos.

3. Tráfico DNS sospechoso detectado automáticamente

Sin necesidad de ejecutar comandos específicos, Snort detectó respuestas DNS con TTL muy bajo y sin autoridad, consideradas como posibles intentos de *spoofing*.

Estas alertas se originaron durante la navegación web desde el PC con Windows, demostrando la efectividad de las reglas comunitarias aplicadas sobre el módulo DNS.

Configuración personalizada validada: activación del módulo **dns** mediante los bloques **binder** y **default_mod** en **snort.lua**.

```
04/01-03:13:51.617574 [**] [1:254:17] "PROTOCOL-DNS SPOOF query response with TTL of 1 min, and no authority" [**] [Classification: Potentially Bad Traffic] [Priority: 2] [UDP] 46.6.113.34:53 -> 192.168.1.152:51944
04/01-03:13:51.645621 [**] [1:254:17] "PROTOCOL-DNS SPOOF query response with TTL of 1 min, and no authority" [**] [Classification: Potentially Bad Traffic] [Priority: 2] [UDP] 46.6.113.34:53 -> 192.168.1.152:60306
04/01-03:13:51.689783 [**] [1:254:17] "PROTOCOL-DNS SPOOF query response with TTL of 1 min, and no authority" [**] [Classification: Potentially Bad Traffic] [Priority: 2] [UDP] 212.230.135.1:53 -> 192.168.1.152:51944
04/01-03:13:52.067471 [**] [1:254:17] "PROTOCOL-DNS SPOOF query response with TTL of 1 min, and no authority" [**] [Classification: Potentially Bad Traffic] [Priority: 2] [UDP] 46.6.113.34:53 -> 192.168.1.152:60306
04/01-03:13:52.078495 [**] [1:254:17] "PROTOCOL-DNS SPOOF query response with TTL of 1 min, and no authority" [**] [Classification: Potentially Bad Traffic] [Priority: 2] [UDP] 46.6.113.34:53 -> 192.168.1.152:63459
04/01-03:13:52.108857 [**] [1:254:17] "PROTOCOL-DNS SPOOF query response with TTL of 1 min, and no authority" [**] [Classification: Potentially Bad Traffic] [Priority: 2] [UDP] 212.230.135.1:53 -> 192.168.1.152:60306
```

Figura 7.24: Respuestas DNS con TTL anómalo detectadas como tráfico malicioso.

4. Escaneo SNMP detectado con Nmap

Se realizó un escaneo UDP desde el Ubuntu Server al equipo Windows para evaluar la detección de tráfico SNMP:

```
sudo nmap -sU -p 161,705 192.168.1.152
```

Snort identificó intentos de conexión tanto al puerto SNMP clásico como al puerto de AgentX, clasificándolos como intentos de reconocimiento o fuga de información.

Configuración personalizada validada: activación explícita de `rpc_decode` y `snmp` en `snort.lua`, además de reglas comunitarias específicas para tráfico SNMP.

```
04/01-03:42:10.882373 [**] [1:1418:19] "PROTOCOL-SNMP request tcp" [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.1.171:43868 -> 192.168.1.152:161
04/01-03:42:10.982627 [**] [1:1418:19] "PROTOCOL-SNMP request tcp" [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.1.171:43869 -> 192.168.1.152:161
04/01-03:42:11.882454 [**] [1:1421:19] "PROTOCOL-SNMP AgentX/tcp request" [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.1.171:43868 -> 192.168.1.152:795
04/01-03:42:11.982688 [**] [1:1421:19] "PROTOCOL-SNMP AgentX/tcp request" [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.1.171:43869 -> 192.168.1.152:795
```

Figura 7.25: Detección de actividad SNMP sospechosa tras escaneo dirigido.

5. Detección de NUSS (Número de Seguridad Social)

Se evaluó una regla personalizada para detectar cadenas numéricas de 12 dígitos que simulan un NUSS español. Desde el equipo Ubuntu se ejecutó:

```
curl -d "Mi número de SSN es 123456789012" http://192.168.1.1
```

Snort detectó la cadena como una posible violación de privacidad, generando una alerta con la clasificación y prioridad definidas manualmente.

Configuración personalizada validada: expresión regular para patrones tipo NUSS, con una `classtype` propia y prioridad 1.

```
04/01-03:47:25.600057 [**] [1:384:8] "PROTOCOL-ICMP PING" [**] [Classification: Misc activity] [Priority: 3] {ICMP} 192.168.1.169 -> 142.250.184.164
04/01-03:47:25.615235 [**] [1:408:8] "PROTOCOL-ICMP Echo Reply" [**] [Classification: Misc activity] [Priority: 3] {ICMP} 142.250.184.164 -> 192.168.1.169
04/01-03:47:27.195867 [**] [1:1000003:1] "Sensitive Data - NUSS España detectado" [**] [Classification: Potential Corporate Privacy Violation] [Priority: 1] {TCP} 192.168.1.171:53758 -> 192.168.1.1:80
```

Figura 7.26: Alerta generada por la regla personalizada de detección de NUSS.

7.4. Resumen

Una vez completada la instalación de R-Snort, se ha evaluado su comportamiento en un entorno de red real, midiendo tanto su impacto en los recursos de la Raspberry Pi como su capacidad de detección. Para ello, se utilizó la herramienta `dstat` con el fin de comparar el rendimiento del sistema con Snort activado y desactivado. Los resultados indican que, aunque se observa un incremento en el uso de CPU, memoria y escritura en disco, dicho impacto es moderado y esperable para un dispositivo de recursos limitados. Asimismo, el tráfico de red presenta una leve alteración con Snort activo, lo que sugiere una ligera contención de paquetes provocada por el procesamiento en tiempo real.

Además, se llevaron a cabo pruebas prácticas orientadas a validar las capacidades de detección del sistema. Acciones como escaneos de puertos con `Nmap` o el envío de datos sensibles (correos electrónicos, números de tarjeta o NUSS) fueron correctamente identificadas y notificadas mediante alertas. El sistema demostró una buena capacidad de diferenciación entre tráfico legítimo (por ejemplo, navegación web, transferencia de archivos o pings) y actividades potencialmente maliciosas. Estas alertas fueron reflejadas de forma inmediata en el panel web desarrollado, validando la integración completa entre el motor de detección y la interfaz de visualización.

En conjunto, los resultados confirman que R-Snort es una solución eficaz para proteger redes pequeñas, ofreciendo capacidades avanzadas de detección sin comprometer el rendimiento del sistema ni la experiencia de usuario.

8. Resultados y discusión

8.1. Síntesis de resultados

A lo largo de las pruebas realizadas, R-Snort ha demostrado ser una herramienta capaz de cara a la detección de intrusos en redes pequeñas. Los resultados del benchmark muestran un consumo de recursos moderado incluso en condiciones de tráfico elevado. Las gráficas obtenidas mediante `dstat` y procesadas con `Python` indican que la carga adicional generada por el sistema es asumible en términos de CPU, RAM y E/S en disco. Este comportamiento encaja con las expectativas de un sistema NIDS pasivo, diseñado para operar en segundo plano sin degradar la experiencia de los usuarios legítimos de la red.

Además, las pruebas funcionales con tráfico real han validado el correcto funcionamiento de los preprocesadores de Snort, así como la eficacia tanto de las reglas comunitarias como de las reglas personalizadas definidas para este proyecto. Casos como la detección de fugas de datos (emails, tarjetas, NUSS), escaneos activos o respuestas DNS anómalas fueron correctamente identificados y registrados.

8.2. Discusión crítica

Robustez del sistema

Uno de los principales hallazgos es la robustez operativa de R-Snort en un entorno embedido. Aunque Snort es tradicionalmente utilizado en servidores más potentes, su compilación y despliegue en una Raspberry Pi 5, junto con su correcta ejecución, demuestran que es viable trasladar capacidades avanzadas de inspección a plataformas de bajo consumo. La inclusión de mecanismos como *port mirroring*, configuración en modo promiscuo y despliegue como servicio `systemd`, junto a la modularidad del paquete `.deb`, refuerzan este hecho.

Impacto en el rendimiento

Los resultados del benchmark reflejan que el sistema introduce una carga ligera pero estable indicando que es perfectamente viable su uso en sistemas de menor potencia. El incremento de context switches, uso de CPU en espacio de sistema y la escritura en disco (asociada al logueo de eventos) es coherente con lo esperado en sistemas de inspección profunda. No obstante, el rendimiento general de la Raspberry Pi se mantuvo estable, sin caídas ni congestión apreciable, lo que sugiere que Snort está bien optimizado para su función en este contexto.

Eficacia en la detección

La capacidad de detección ha sido prometedora. Snort detectó escaneos de puertos, tráfico DNS sospechoso, anomalías en ICMP y transmisiones de datos sensibles. La reacción del sistema fue inmediata y coherente con los patrones definidos. Sin embargo, hay que matizar que todas las pruebas fueron realizadas en un entorno controlado. La efectividad en redes más complejas y dinámicas con mayor volumen de tráfico y ataques más sofisticados quizás requiera ajustes adicionales en las reglas, thresholds de alertas y quizás el uso de motores de correlación adicionales.

Usabilidad y mantenimiento

La experiencia de usuario ha sido positiva, en todo momento se ha tratado de ser user-friendly, tanto en la instalación como en la supervisión del sistema. El instalador modular automatizado y la arquitectura empaquetada facilitan el mantenimiento y reducen la necesidad de intervención del usuario. Este aspecto fue el objetivo desde el principio para que soluciones como R-Snort puedan ser adoptadas en entornos SOHO o por usuarios no expertos sin un alto impacto económico.

8.3. Limitaciones del proyecto

A pesar de los buenos resultados, deben reconocerse varias limitaciones:

- Las pruebas se realizaron en un entorno de red pequeño, con tráfico moderado y escasa variabilidad. No se evaluó el rendimiento bajo ataques simultáneos o flujos de datos cifrados intensivos.
- Snort no realizó inspección activa sobre tráfico TLS/HTTPS más allá de los metadatos, lo cual limita su alcance en un contexto donde el cifrado es predominante.
- La recolección de datos para el benchmark se centró en métricas de sistema, sin incluir análisis del ratio de falsos positivos o falsos negativos.

8.4. Conclusión de los resultados

En conjunto, los resultados respaldan la viabilidad del proyecto: R-Snort es capaz de proporcionar vigilancia activa sobre una red pequeña sin requerir hardware costoso ni administración compleja. Su desempeño, adaptabilidad y facilidad de despliegue lo convierten en una alternativa prometedora para quienes buscan una solución IDS ligera, efectiva y de código abierto.

No obstante, su uso debe entenderse como una primera capa de defensa. Para entornos más exigentes o críticos, sería recomendable complementarlo con herramientas de análisis forense, correlación de eventos y sistemas de respuesta automática.

Conclusiones

Tras las pruebas realizadas con R-Snort y el análisis del tráfico real, se concluye que el sistema presenta un consumo de recursos moderado y perfectamente asumible. Aunque se observa un incremento en la actividad de la CPU y la memoria cuando el motor de Snort está activo, el sistema permanece estable y responde adecuadamente en todo momento. Resulta destacable el rendimiento de la Raspberry Pi, que, a pesar de sus limitaciones de hardware, soporta con solvencia la carga asociada a la monitorización en tiempo real.

Durante la fase de pruebas, R-Snort logró detectar una amplia variedad de eventos, desde peticiones ICMP simples (como ping hacia 8.8.8.8) hasta escaneos de puertos mediante Nmap, pasando por intentos simulados de fuga de información con datos sensibles como correos electrónicos o números de tarjeta de crédito. En todos los casos, el sistema generó alertas precisas y en tiempo real.

Un aspecto especialmente relevante es que la inspección continua del tráfico no interfiere de manera perceptible con el uso cotidiano de la red. Durante tareas como navegación web, descarga de archivos o conexiones SSH, el rendimiento se mantuvo fluido, lo que demuestra que R-Snort puede operar como una sonda pasiva sin afectar negativamente la experiencia del usuario. Incluso ante picos de tráfico, el sistema continúa funcionando de forma aceptable, aunque con una carga adicional esperada en el procesador.

En resumen, desplegar R-Snort en un dispositivo tan compacto como una Raspberry Pi proporciona una solución de monitorización efectiva, de bajo impacto y con resultados fiables. Su capacidad para detectar comportamientos anómalos, generar alertas útiles y operar de forma transparente en la red lo convierten en una herramienta idónea para entornos domésticos o pequeñas oficinas, ofreciendo una capa adicional de seguridad sin requerir una infraestructura compleja ni costosa.

Trabajo futuro

Si bien R-Snort demuestra ser una solución funcional y eficiente en entornos SOHO, existen múltiples líneas de mejora que podrían explorarse para ampliar sus capacidades. A continuación, se detallan algunas propuestas que podrían aportar valor al sistema en versiones futuras:

1. Soporte mejorado para tráfico cifrado

Snort puede detectar metadatos de sesiones TLS, pero no realiza inspección profunda del contenido cifrado. Para mejorar la cobertura:

- Se podría integrar un proxy TLS para inspección SSL mediante técnicas de *SSL interception*, aunque esto plantea implicaciones legales y éticas.
- Alternativamente, se podrían usar heurísticas sobre patrones de tráfico cifrado (frecuencia, duración, tamaño) para inferir comportamientos anómalos.

2. Correlación de eventos y análisis de comportamiento

El uso de reglas aisladas tiene limitaciones frente a ataques distribuidos o técnicas evasivas. Por ello, se propone:

- Integrar un motor de correlación simple que agrupe alertas relacionadas por IP, hora o tipo.
- Desarrollar reglas basadas en comportamiento (por ejemplo, múltiples conexiones en puertos no estándar en un corto intervalo de tiempo).
- Implementar funciones de *rate limiting* y alertas por umbral para detectar escaneos o ataques de fuerza bruta progresivos.

3. Integración con herramientas de respuesta activa

R-Snort actúa como sistema de detección pasiva. En un futuro, podría complementarse con capacidades de respuesta automatizada:

- Bloqueo temporal de IPs detectadas como maliciosas mediante reglas en `iptables` o listas negras.
- Integración con fail2ban o sistemas similares para el bloqueo parcial o definitivo de los atacantes.
- Notificaciones en tiempo real por correo o servicios como Telegram o Discord Webhooks.

4. Paquete multi-plataforma y documentación extendida

El actual instalador está optimizado para Raspberry Pi 5 y Ubuntu Server. Sería útil ampliar la compatibilidad para:

- Otras arquitecturas ARM (como Orange Pi, Odroid o Rock Pi).
- Versiones de Debian y derivados (como Raspberry Pi OS o Ubuntu Desktop).
- Profesionalizar más el repositorio con documentación técnica detallada, guía de usuario y plantilla de personalización de reglas. Actualmente el repositorio existe pero se podría mejorar.

5. Métricas de calidad y validación formal

Para futuras versiones se podrían incluir:

- Pruebas automáticas de regresión tras cada cambio en las reglas.
- Métricas de rendimiento bajo ataques reales generados con herramientas como Metasploit o Scapy.
- Evaluación sistemática del ratio de falsos positivos y negativos.

6. Incorporación de aprendizaje automático

Aunque Snort no está diseñado para detección basada en IA, podría integrarse con un módulo externo que analice el tráfico histórico y proponga reglas o patrones de interés. Se podría investigar:

- Detección de anomalías mediante clustering como k-means o DBSCAN.
- Clasificación binaria de tráfico legítimo vs malicioso con árboles de decisión o redes neuronales simples.

Bibliografía

- [1] European Union Agency for Cybersecurity (ENISA). *Cybersecurity for SMEs: Challenges and Recommendations*, 2021. Disponible en: <https://www.enisa.europa.eu/sites/default/files/publications/ENISA%20Report%20-%20Cybersecurity%20for%20SMEs%20Challenges%20and%20Recommendations.pdf>. Último acceso: 9 abril 2025.
- [2] Rodríguez Eguren, P. S., Chichizola, F., y Rucci, E. (2018). *Análisis del uso de un cluster de Raspberry Pi para cómputo de alto rendimiento*. En XXIV Congreso Argentino de Ciencias de la Computación (CACIC 2018), La Plata, Argentina, pp. 134-144. Disponible en: <http://sedici.unlp.edu.ar/handle/10915/73047>. Último acceso: 9 abril 2025.
- [3] M. Roesch, “Snort: Lightweight Intrusion Detection for Networks,” en *Proceedings of the 13th USENIX Conference on System Administration (LISA '99)*, Seattle, Washington, USA, 1999. Disponible en: https://www.usenix.org/legacy/event/lisa99/full_papers/roesch/roesch.pdf.
- [4] Cisco Systems. *Snort 3 Official Website*. Disponible en: <https://www.snort.org/snort3>. Último acceso: 9 abril 2025.
- [5] Insani, P. P., Kanedi, I., y Al Akbar, A. (2023). *Implementation of Snort as a Wireless Network Security Detection Tool Using Linux Ubuntu*. *Jurnal Komputer, Informasi dan Teknologi*, 3(2), 443–458.
- [6] Gkamas, T., Karaiskos, V., y Kontogiannis, S. (2022). *Performance evaluation of distributed database strategies using Docker as a Service for industrial IoT data: Application to Industry 4.0*. *Information*, 13(4), 190. MDPI.
- [7] O’Leary, M. (2015). *Snort*. En *Cyber Operations: Building, Defending, and Attacking Modern Computer Networks* (pp. 605–641). Springer.
- [8] Snort Team. *Snort 3 version 3.1.84.0 Release Notes*. GitHub, 2023. Disponible en: <https://github.com/snort3/snort3/releases/tag/3.1.84.0>. Último acceso: 9 abril 2025.
- [9] Kuruviла, A. P., Meng, X., Kundu, S., Pandey, G., y Basu, K. (2022). *Explainable machine learning for intrusion detection via hardware performance counters*. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11), 4952–4964. IEEE.
- [10] Coşar, M., y Karasartova, S. (2017). *A firewall application on SOHO networks with Raspberry Pi and Snort*. En *2017 International Conference on Computer Science and Engineering (UBMK)* (pp. 1000–1003). IEEE.

- [11] P. Cichonski, T. Millar, T. Grance, K. Scarfone. *Guide to Intrusion Detection and Prevention Systems (IDPS)*. NIST Special Publication 800-94, 2012. Disponible en: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-94.pdf>.
- [12] Abbas, S., Naser, W., y Kadhim, A. *Subject review: Intrusion detection system (IDS) and intrusion prevention system (IPS)*. Global Journal of Engineering and Technology Advances, vol. 2, nº 14, pp. 155–158, 2023.
- [13] Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., y Vázquez, E. (2009). *Anomaly-based network intrusion detection: Techniques, systems and challenges*. *Computers & Security*, 28(1–2), 18–28. Elsevier.
- [14] Signature-based Detection. (2005). *Signature-based Intrusion Detection*.
- [15] Younus, Z. S., y Alanezi, M. (2023). *Detect and Mitigate Cyberattacks Using SIEM*. En *2023 16th International Conference on Developments in eSystems Engineering (DeSE)* (pp. 510–515). IEEE.
- [16] Cortés Novoa, A. F. (2017). *Amenazas persistentes avanzadas (APT): modelo de funcionamiento y análisis al caso de estudio ProjectSauron* (Tesis de grado). Universidad Piloto de Colombia.
- [17] Travis, G., Balas, E., Ripley, D., y Wallace, S. (2004). *Analysis of the “SQL Slammer” worm and its effects on Indiana University and related institutions*. Indiana University. Disponible en: <https://www.megasecurity.org/papers/Back-Doored%20by%20the%20Slammer.pdf>. Último acceso: 9 abril 2025.
- [18] Kerr, P. K., Rollins, J., y Theohary, C. A. (2010). *The Stuxnet computer worm: Harbinger of an emerging warfare capability*. Congressional Research Service, Washington, DC. Disponible en: <https://cyberwar.nl/d/R41524.pdf>. Último acceso: 9 abril 2025.
- [19] Al-Rabiahah, S. (2018). *The “Stuxnet” virus of 2010 as an example of a “APT” and its “Recent” variances*. En *2018 21st Saudi Computer Society National Computer Conference (NCC)* (pp. 1–5). IEEE.
- [20] Snort. *Página oficial de Snort*. Disponible en: <https://www.snort.org/>
- [21] Cisco Systems. *Rule Writer’s Guide to Snort 3 Rules*. Disponible en: https://snort-org-site.s3.amazonaws.com/production/document_files/files/000/000/596/original/Rules_Writers_Guide_to_Snort_3_Rules.pdf. Último acceso: 9 abril 2025.
- [22] Open Information Security Foundation (OISF). *Suricata Official Website*. Disponible en: <https://suricata.io/>. Último acceso: 9 abril 2025.
- [23] Cisco Talos. *Registered vs. Subscriber Rules*. Snort.org. Disponible en: <https://snort.org/documents/registered-vs-subscriber>.
- [24] Rueda, Á. F., Rodríguez García, J. A., y Sanz de Castro, I. (s.f.). *Revisiting SOHO Router Attacks*. En *In Depth Security Vol. II* (p. 135). Disponible en: <https://d-nb.info/115018552X/34#page=143>. Último acceso: 9 abril 2025.

- [25] Bakhshi, T., Ghita, B., & Kuzminykh, I. (2024). *A review of IoT firmware vulnerabilities and auditing techniques*. Sensors, 24(2), 708. MDPI. Recuperado de <https://www.mdpi.com/1424-8220/24/2/708/pdf>. Último acceso: 9 abril 2025.
- [26] Park, W., y Ahn, S. (2017). *Performance comparison and detection analysis in Snort and Suricata environment*. *Wireless Personal Communications*, 94, 241–252. Springer.
- [27] Netgate. *Snort 3.2.9.14 and Snort 4.1.1 Updates – Release Notes for pfSense 2.4.5_p1 and pfSense 2.5*, 2020. Disponible en: <https://forum.netgate.com/topic/155429/snort-3-2-9-14-and-snort-4-1-1-updates-release-notes-for-pfsense-2-4-5-p1-and-pfsense-2-5>. Último acceso: 9 abril 2025.
- [28] Combs, R., & Munshaw, J. (2020). *The major differences that set Snort 3 apart from Snort 2*. Recuperado de <https://blog.snort.org/2020/08/snort-3-2-differences.html>. Último acceso: 9 abril 2025.
- [29] Cisco Systems. *Snort 3.1.8.0 Installation Guide for Ubuntu 18 and 20*. Disponible en: https://snort-org-site.s3.amazonaws.com/production/document_files/files/000/012/147/original/Snort_3.1.8.0_on_Ubuntu_18_and_20.pdf. Último acceso: 9 abril 2025.
- [30] Cisco. (2023). *Snort 3.1.84.0 User Manual*. La documentación oficial proporciona una guía detallada sobre la configuración de snort.lua. Recuperado de https://snort-org-site.s3.amazonaws.com/production/release_files/files/000/038/976/original/snort_user.html. Último acceso: 10 abril 2025.
- [31] Emerging Threats. (2025). *Emerging Threats Firewall Block List*. Recuperado de <https://rules.emergingthreats.net/fwrules/emerging-Block-IPs.txt>. Último acceso: 10 abril 2025.
- [32] Song, W., Beshley, M., Przystupa, K., Beshley, H., Kochan, O., Pryslupskyi, A., & Su, J. (2020). *A software deep packet inspection system for network traffic analysis and anomaly detection*. Sensors, 20(6), 1637. Recuperado de <https://www.mdpi.com/1424-8220/20/6/1637/pdf>. Último acceso: 10 abril 2025.
- [33] Debian Project. (2024). *Debian New Maintainers' Guide – Chapter 6: Good packaging practices*. Recuperado de <https://www.debian.org/doc/manuals/debmake-doc/ch06.en.html>. Último acceso: 10 abril 2025.
- [34] Bartman, T., & Kraft, J. (2016). *An introduction to applying network intrusion detection for industrial control systems*. Iron & Steel Technology, 13(12), 70–77. En este trabajo se describe cómo emplear Snort en entornos industriales mediante un switch configurado con port mirroring, lo que permite la inspección pasiva del tráfico sin afectar la red productiva. Recuperado de https://assets.contentstack.io/v3/assets/bltea08f3d94a418a1b/bltfe74dab1ab1bc726/62e949c0e791ec0dc5db3668/6753_IntroductionApplying_TB_20160212_Web2.pdf. Último acceso: 10 abril 2025.
- [35] de la Cruz, J. E. C., Goyzueta, C. A. R., & Cahuana, C. D. (2020, septiembre). *Intrusion detection and prevention system for production supervision in small businesses based on*

Raspberry Pi and Snort. En 2020 IEEE XXVII International Conference on Electronics, Electrical Engineering and Computing (INTERCON) (pp. 1–4). IEEE.

A. Anexo A: Repositorio de R-SNORT

El código completo del instalador R-Snort, incluyendo los scripts modulares, archivos de configuración, reglas y paquetes necesarios para compilar Snort 3 en sistemas ARM como Raspberry Pi 5, se encuentra disponible en el siguiente repositorio:

- **Repositorio GitHub:** <https://github.com/deianp189/r-snort-installer.git>
- **Última versión estable:** v1.0.0
- **Tipo de licencia:** UAL

Este repositorio puede ser clonado con el siguiente comando:

```
1 git clone https://github.com/deianp189/r-snort-installer.git
```


B. Anexo B: Script de instalación

```

 1 #!/bin/bash
 2 ##### R-SNORT INSTALLER #####
 3 #                                     #
 4 ##### set -euo pipefail
 5 trap 'echo -e
 6   "\n\033[0;31m[X] Fallo en linea $LINENO del script principal\033[0m"' ERR
 7
 8 CONFIG_DIR="$(pwd)/configuracion"
 9 SOFTWARE_DIR="$(pwd)/software"
10 INSTALL_DIR="/usr/local/snort"
11 LOG_FILE="/var/log/snort_install.log"
12
13 # Redirigir salida a log inmediatamente
14 exec > >(tee -a "$LOG_FILE") 2>&1
15
16 # Importar funciones
17 source ./bin/core.sh
18 source ./bin/checks.sh
19 source ./bin/swap.sh
20 source ./bin/dependencies.sh
21 source ./bin/build_from_source.sh
22 source ./bin/install_snort.sh
23 source ./bin/configure_snort.sh
24 source ./bin/stats.sh
25
26 # Verificación mínima
27 type snort_config >/dev/null ||
28 { echo "La función snort_config no está disponible"; exit 1; }
29
30 # Comprobaciones iniciales
31 check_root
32 ascii_banner
33 log "Instalador R-SNORT iniciado"
34
35 # Selección de interfaz
36 interface_selection
37 export IFACE
38
39
40 # Crear estructura de directorios
41 mkdir -p "$INSTALL_DIR"/{bin,etc/snort,lib,include,share,logs,rules}
42
43 ##### Ejecución modular #####
44 #                                     #
45 ##### dependencies_install
46
47 dependencies_install
48 software_package_install
49 snort_install "$SOFTWARE_DIR" "$INSTALL_DIR"
50 temp_swap_clean
51
52 # Configuración final
53 if snort_config "$CONFIG_DIR" "$INSTALL_DIR" "$IFACE"; then
54   log "configurar_snort ejecutado correctamente."
55 else
56   error "configurar_snort falló."
57 fi
58
59 # Comprobación de estado del servicio
60 log "Comprobando estado del servicio Snort..."
61 if systemctl is-enabled --quiet snort && systemctl is-active --quiet snort; then
62   log "Snort está activo y habilitado."
63 else
64   error "Snort no se encuentra activo o habilitado.
65   Verifica manualmente con: systemctl status snort"
66 fi
67
68 # Estadísticas
69 log "Llamando a mostrar_estadisticas con: $IFACE $INSTALL_DIR"
70 if show_stats "$IFACE" "$INSTALL_DIR"; then
71   log "mostrar_estadisticas ejecutado correctamente."
72 else
73   error "mostrar_estadisticas falló."
74 fi

```

```

1  #!/bin/bash
2
3  RED='\033[0;31m'; GREEN='\033[0;32m'; BLUE='\033[0;34m'; NC='\033[0m'
4
5  ascii_banner() {
6      cat << 'EOF'
7      R-SNORT BANNER
8      Snort 3 Installer
9      EOF
10 }
11
12 log() { echo -e "${BLUE}[*] $1${NC}"; }
13 success() { echo -e "${GREEN}[✓] $1${NC}"; }
14 error() { echo -e "${RED}[X] $1${NC}" >&2; exit 1; }

```

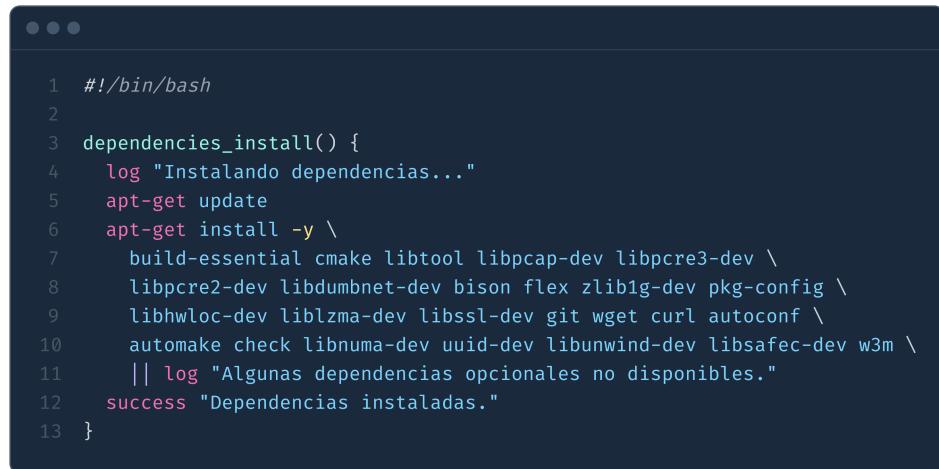
Figura B.2: Banner de bienvenida `core.sh`

```

1  #!/bin/bash
2  check_root() {
3      [ "$($id -u)" -eq 0 ] || error "Este script debe ejecutarse como root."
4  }
5
6  interface_selection() {
7      log "Detectando interfaces de red disponibles..."
8      mapfile -t interfaces < <(ip -brief link | awk '{print $1}' | grep -v '^lo$')
9
10     if (( ${#interfaces[@]} == 0 )); then
11         error "No se encontraron interfaces válidas. Verifica que estén conectadas y activas."
12
13     fi
14
15     echo "Seleccione una interfaz de red:"
16     for i in "${!interfaces[@]}"; do
17         echo "  [$i] ${interfaces[$i]}"
18     done
19
20     read -rp "Introduce el número correspondiente a la interfaz: " selected_index
21
22     [[ ! "$selected_index" =~ ^[0-9]+\$ ]] && error "Entrada inválida. Debe ser un número."
23     (( selected_index < 0 || selected_index >= ${#interfaces[@]} )) && error "Número fuera
de rango. Selección no válida."
24
25     IFACE="${interfaces[$selected_index]}"
26     log "Interfaz seleccionada: $IFACE"
27 }

```

Figura B.3: Comprobador de root e interfaces `checks.sh`



```
1  #!/bin/bash
2
3  dependencies_install() {
4      log "Instalando dependencias..."
5      apt-get update
6      apt-get install -y \
7          build-essential cmake libtool libpcap-dev libpcre3-dev \
8          libpcre2-dev libdumbnet-dev bison flex zlib1g-dev pkg-config \
9          libhwloc-dev liblzma-dev libssl-dev git wget curl autoconf \
10         automake check libnuma-dev uuid-dev libunwind-dev libsafec-dev w3m \
11         || log "Algunas dependencias opcionales no disponibles."
12     success "Dependencias instaladas."
13 }
```

Figura B.4: Instalador de dependencias `dependencies.sh`

```

1  #!/bin/bash
2
3  package_install() {
4      local archivo="$1"
5
6      # Validar integridad antes de intentar descomprimir
7      if [[ "$archivo" == *.tar.gz ]]; then
8          gzip -t "$archivo" || error "Archivo corrupto (gzip): $archivo"
9
10     elif [[ "$archivo" == *.tar.xz ]]; then
11         if ! command -v xz >/dev/null 2>&1; then
12             log "Instalando xz-utils..."
13             apt-get update && apt-get install -y xz-utils liblzma-dev || error "No se pudo
14             instalar xz-utils"
15         fi
16
17         # Detectar fallo por incompatibilidad de versión de liblzma
18         if ! xz -t "$archivo" 2>&1 | grep -qv 'versión XZ_'; then
19             log "Corrigiendo incompatibilidad de xz/liblzma..."
20             apt-get install --reinstall -y xz-utils liblzma-dev || error "No se pudo reinstalar
21             xz/liblzma"
22         fi
23
24         xz -t "$archivo" || error "Archivo corrupto (xz): $archivo"
25     fi
26
27     log "Instalando: $(basename \"$archivo\")"
28     tar -xf "$archivo"
29     dir=$(find . -mindepth 1 -maxdepth 1 -type d | grep -v '^./.' | head -n 1)
30
31     if [[ -z "$dir" || ! -d "$dir" ]]; then
32         error "No se encontró un directorio válido tras descomprimir $archivo"
33     fi
34
35     log "Entrando en directorio: $dir"
36     cd "$dir"
37
38     case "$archivo" in
39         *luajit*)
40             make -j$(nproc)
41             make install PREFIX=/usr
42             ;;
43
44         *openssl*)
45             local target
46             if uname -m | grep -q aarch64; then
47                 target="linux-aarch64"
48             else
49                 target="linux-generic32"
50             fi
51             ./Configure --prefix=/usr --openssldir=/etc/ssl "$target"
52             make -j$(nproc)
53             make install
54             ;;
55
56         *daq*)
57             log "Instalando DAQ con precauciones (desactivando 'set -e' temporalmente)..."
58             set +e
59             if [[ -f "bootstrap" ]]; then
60                 chmod +x bootstrap
61                 ./bootstrap
62                 bootstrap_status=$?
63             else
64                 bootstrap_status=1
65             fi
66             if [[ $bootstrap_status -ne 0 && -f "configure.ac" && ! -f "configure" ]]; then
67                 autoreconf -f
68             fi
69             set -e
70             ./configure --prefix=/usr --enable-shared
71             make -j$(nproc)
72             make install || error "Fallo al instalar DAQ"
73             ;;
74
75         *)
76             if [[ -f "configure.ac" && ! -f "configure" ]]; then
77                 [[ -f "bootstrap" ]] && chmod +x bootstrap && ./bootstrap || autoreconf -f
78             fi
79             if [[ -f "configure" ]]; then
80                 ./configure --prefix=/usr --enable-shared
81             else
82                 cmake . -DCMAKE_INSTALL_PREFIX=/usr
83             fi
84             make -j$(nproc)
85             make install || error "Fallo al instalar $(basename \"$archivo\")"
86             ;;
87         esac
88     cd ..
89     rm -rf "$dir"
90     success "$(basename \"$archivo\") instalado."
91 }
92 software_package_install() {
93     cd "$SOFTWARE_DIR"
94     log "Ordenando paquetes para instalación de dependencias primero..."
95     for f in $(ls *.tar.gz *.tar.xz 2>/dev/null | sort | grep -vi snort); do
96         package_install "$f"
97     done
98     log "Snort se instalará en una fase posterior. Omitido aquí."
99
100    log "Instalando ClamAV para protección antivirus..."
101    apt-get install -y clamav clamav-daemon || error "No se pudo instalar ClamAV"
102    freshclam || log "No se pudo actualizar la base de firmas en este momento"
103    systemctl enable clamav-freshclam
104    systemctl enable clamav-daemon
105    systemctl restart clamav-daemon
106    systemctl is-active --quiet clamav-daemon && success "ClamAV está activo." || log
107    "ClamAV instalado pero no activo."
108 }

```

Figura B.5: Instalador de paquetes `build_from_source.sh`

```
1 #!/bin/bash
2
3 snort_install() {
4     local SOFTWARE_DIR="$1"
5     local INSTALL_DIR="$2"
6
7     log "Preparando instalación de Snort 3 (versión estable sin soporte NUMA)..."
8
9     cd "$SOFTWARE_DIR"
10    tar -xzf snort3.tar.gz
11    cd "$(find . -maxdepth 1 -type d -name 'snort3*' | head -n 1)"
12
13    # Corrige bug de hilos si es necesario (solo versiones antiguas)
14    sed -i 's/[\ \\"\\$NUMTHREADS\\\" -lt \\"\\$MINTHREADS\\\" \]/[ \\"${NUMTHREADS:-0}\\\" -lt
15 \\"${MINTHREADS:-1}\\\" ]/' configure_cmake.sh
16
17    # Prevenir advertencias por OpenSSL 3+
18    unset LDFLAGS
19    unset CXXFLAGS
20    export LDFLAGS=""
21    export CXXFLAGS="-Wno-deprecated-declarations"
22
23    ./configure_cmake.sh --prefix="$INSTALL_DIR"
24
25    cd build
26    temp_swap_if_necessary
27
28    log "Compilando Snort 3. Puede tardar varios minutos..."
29    make -j$(nproc) || error "Fallo en make al compilar Snort 3."
30    make install
31    ldconfig
32    ln -sf "$INSTALL_DIR/bin/snort" /usr/local/bin/snort
33
34    success "Snort 3 instalado con éxito."
35 }
```

Figura B.6: Compilador de snort `install_snort.sh`

```

1  #!/bin/bash
2
3  snort_config() {
4      local CONFIG_DIR="$1"
5      local INSTALL_DIR="$2"
6      local IFACE="$3"
7
8      log "Copiando configuración..."
9      cp "$CONFIG_DIR/snort.lua" "$INSTALL_DIR/etc/snort/"
10     cp "$CONFIG_DIR/custom.rules" "$INSTALL_DIR/etc/snort/"
11     mkdir -p "$INSTALL_DIR/etc/snort/reputation" "$INSTALL_DIR/etc/snort/snort3-community-
12     rules"
13     touch "$INSTALL_DIR/etc/snort/reputation/interface.info"
14     tar -xzf "$CONFIG_DIR/snort3-community-rules.tar.gz" -C "$INSTALL_DIR/etc/snort/snort3-
15     community-rules" --strip-components=1
16
17     if [ -f /etc/systemd/system/snort.service ]; then
18         cp /etc/systemd/system/snort.service /etc/systemd/system/snort.service.bak
19         log "Backup del servicio original guardado."
20     fi
21
22     cat > /etc/systemd/system/snort.service <<EOF
23     [Unit]
24     Description=Snort NIDS Daemon
25     After=network.target
26
27     [Service]
28     ExecStart=$INSTALL_DIR/bin/snort -c $INSTALL_DIR/etc/snort/snort.lua -i $IFACE -A
29     alert_fast
30     ExecReload=/bin/kill -HUP \${MAINPID}
31     Restart=always
32     User=root
33     Group=root
34     LimitCORE=infinity
35     LimitNOFILE=65536
36     LimitNPROC=65536
37     PIDFile=/var/run/snort.pid
38
39     [Install]
40     WantedBy=multi-user.target
41     EOF
42
43     systemctl daemon-reexec
44     systemctl daemon-reload
45     systemctl enable snort.service
46     systemctl restart snort.service || error "No se pudo iniciar Snort."
47     sleep 2
48     systemctl status snort.service --no-pager
49     success "Servicio Snort configurado y activo."
50 }

```

Figura B.7: Configurador de snort `configure_snort.sh`

```
1 #!/bin/bash
2
3 temp_swap_if_necessary() {
4     local mem_kb
5     mem_kb=$(grep MemTotal /proc/meminfo | awk '{print $2}')
6
7     if [ "$mem_kb" -lt 1500000 ]; then
8         if ! free | awk '/^Swap:/ {exit !$2}'; then
9             log "Creando swap temporal..."
10
11         if ! fallocate -l 2G /swapfile_snort; then
12             log "fallocate falló, usando dd como alternativa..."
13             dd if=/dev/zero of=/swapfile_snort bs=1M count=2048 || {
14                 error "No se pudo crear swap con dd tampoco. Abortando."
15             return 1
16         }
17     fi
18
19     chmod 600 /swapfile_snort
20     mkswap /swapfile_snort
21     swapon /swapfile_snort
22     success "Swap temporal creada en /swapfile_snort"
23     else
24         log "Swap ya activa. No se necesita crear otra."
25     fi
26 else
27     log "RAM suficiente (>=1.5 GB). No se necesita swap temporal."
28 fi
29 }
30
31 temp_swap_clean() {
32     if swapon --show | grep -q "/swapfile_snort"; then
33         log "Desactivando swap temporal..."
34         if swapoff /swapfile_snort; then
35             rm -f /swapfile_snort
36             success "Swap temporal eliminada."
37         else
38             log "Swap ya estaba desactivada o no se pudo eliminar. Continuando."
39         fi
40     else
41         log "No hay swap temporal activa. Nada que limpiar."
42     fi
43 }
```

Figura B.8: Comprobador de swap `swap.sh`

```

1  #!/bin/bash
2
3  show_stats() {
4      local IFACE="$1"
5      local INSTALL_DIR="$2"
6
7      echo
8      log "Resumen del sistema tras la instalación:"
9      uptime_str=$(uptime -p)
10     total_ram=$(free -h | awk '/Mem:/ {print $2}')
11     used_ram=$(free -h | awk '/Mem:/ {print $3}')
12     swap_enabled=$(swapon --noheadings | wc -l)
13     swap_used=$(free -h | awk '/Swap:/ {print $3 "/" $2}')
14     disk_usage=$(df -h / | awk 'NR==2 {print $3 " usados de " $2}')
15     cpu_model=$(lscpu | grep "Model name" | sed 's/Model name:\s*//')
16     cpu_cores=$(nproc)
17     snort_version=$(("$INSTALL_DIR/bin/snort" -V 2>/dev/null | grep -i "Version" | awk
18 '{print $3}' || echo "No encontrado")
19     clamav_version=$(clamscan -V 2>/dev/null | awk '{print $2}' || echo "No encontrado")
20     echo "-----"
21     echo -e "Hostname: $(hostname)"
22     echo -e "Uptime: $uptime_str"
23     echo -e "RAM usada: $used_ram / $total_ram"
24     echo -e "Swap activa: $([ "$swap_enabled" -eq 0 ] && echo "No" || echo "Sí
25     ($swap_used)")"
26     echo -e "Espacio raíz: $disk_usage"
27     echo -e "CPU: $cpu_model ($cpu_cores núcleos)"
28     echo -e "Snort versión: ${snort_version:-No encontrado}"
29     echo -e "ClamAV versión: ${clamav_version:-No encontrado}"
30     echo -e "Interfaz activa: $IFACE"
31     echo "-----"
32     success "Snort 3 está en ejecución en la interfaz: $IFACE."
33 }

```

Figura B.9: Estadísticas finales stats.sh

```
● ● ●

1 #!/bin/bash
2 set -e
3
4 echo "[R-SNORT] Iniciando instalación automática..."
5
6 if [ ! -d /opt/r-snort ]; then
7   echo "[R-SNORT] ERROR: /opt/r-snort no existe. La instalación falló o el paquete está
     corrupto."
8   exit 1
9 fi
10
11 cd /opt/r-snort
12 chmod +x *.sh bin/*.sh
13
14 ./r-snort_installer.sh
15
16 echo "[R-SNORT] Instalación completada."
```

Figura B.10: Contenido del script `postinst` utilizado durante la instalación del paquete.

```
● ● ●

1 #!/bin/bash
2 echo "[R-SNORT] Deteniendo servicio Snort antes de eliminar el paquete..."
3 systemctl stop snort.service 2>/dev/null || true
4 systemctl disable snort.service 2>/dev/null || true
5 echo "[R-SNORT] Snort detenido correctamente (si estaba en ejecución)."
```

Figura B.11: Contenido del script `prerm` ejecutado antes de desinstalar el paquete.

```
1  #!/bin/bash
2  set -e
3
4  case "$1" in
5      remove|purge)
6          echo "[R-SNORT] Eliminando restos del paquete..."
7          rm -rf /opt/r-snort 2>/dev/null || true
8          rm -f /etc/systemd/system/snort.service 2>/dev/null || true
9          systemctl daemon-reexec 2>/dev/null || true
10         systemctl daemon-reload 2>/dev/null || true
11         echo "[R-SNORT] Limpieza completada."
12         ;;
13     *)
14         echo "[R-SNORT] postrm ejecutado con modo '$1' (sin limpiar)."
15         ;;
16 esac
```

Figura B.12: Contenido del script `postrm` ejecutado después de la desinstalación del paquete.

```
1  #!/bin/bash
2  set -e
3
4  echo "🌐 [R-SNORT] Actualizando lista de paquetes..."
5  sudo apt update
6
7  echo "📦 [R-SNORT] Instalando dependencias..."
8  sudo apt install -y bash build-essential libpcap-dev xz-utils liblzma-dev clamav clamav-
daemon
9
10 echo "✅ [R-SNORT] Dependencias instaladas."
11
12 # Buscar interfaces Ethernet conectadas
13 echo "🌐 Buscando interfaces Ethernet disponibles..."
14 interfaces=$(ip -o link show | awk -F': ' '/^[:digit:]+: e/ {print $2})"
15
16 if [[ ${#interfaces[@]} -eq 0 ]]; then
17   echo "❌ No se encontraron interfaces Ethernet. ¿Está el adaptador conectado?"
18   exit 1
19 fi
20
21 echo "🌐 Interfaces disponibles:"
22 for i in "${!interfaces[@]}"; do
23   echo "  [$i] ${interfaces[$i]}"
24 done
25
26 read -rp "➡️ Elige la interfaz para analizar tráfico (la del switch): " index
27 IFACE="${interfaces[$index]}"
28
29 # Guardar la interfaz en archivo para que el script dentro del .deb la use
30 echo "$IFACE" | sudo tee /etc/rsnort_iface > /dev/null
31
32 # Verificación
33 echo "✅ Interfaz seleccionada: $IFACE"
34 echo
35
36 # Instalación del paquete
37 if [ ! -f r-snort-deb.deb ]; then
38   echo "❌ [ERROR] No se encontró el archivo r-snort-deb.deb"
39   echo "➡️ Ejecuta: dpkg-deb --build r-snort-deb"
40   exit 1
41 fi
42
43 echo "📦 [R-SNORT] Instalando paquete .deb..."
44 sudo dpkg -i r-snort-deb.deb || {
45   echo "⚠️ dpkg reportó errores. Intentando solucionarlos..."
46   sudo apt --fix-broken install -
47 }
48
49 echo "🎉 [R-SNORT] Instalación completa."
```

Figura B.13: Script `install_rsnort.sh` encargado del proceso de instalación automatizada y selección de interfaz.

```
1 #!/bin/bash
2
3 interface_setup() {
4     local iface="$1"
5
6     log "Verificando estado de la interfaz $iface..."
7     state=$(ip link show "$iface" | grep -o 'state [A-Z]*' | awk '{print $2}')
8
9     if [[ "$state" != "UP" ]]; then
10        log "La interfaz $iface está DOWN. Activando..."
11        ip link set dev "$iface" up || error "No se pudo activar la interfaz $iface."
12    else
13        log "La interfaz $iface ya está UP."
14    fi
15
16    # Verificar si tiene IP y eliminarla (para sniffeo puro)
17    if ip addr show "$iface" | grep -q 'inet'; then
18        log "Eliminando IP de $iface para sniffeo sin interferencias..."
19        ip addr flush dev "$iface"
20    fi
21
22    # Establecer modo promiscuo si no lo está
23    if ! ip link show "$iface" | grep -q PROMISC; then
24        log "Activando modo promiscuo en $iface..."
25        ip link set "$iface" promisc on || error "No se pudo activar modo promiscuo en
$iface."
26    else
27        log "$iface ya está en modo promiscuo."
28    fi
29
30    success "Interfaz $iface preparada para análisis de red."
31 }
32
33 [[ -n "${IFACE:-}" ]] && interface_setup "$IFACE"
```

Figura B.14: Script `interface_setup.sh` encargado de configurar la interfaz en modo promiscuo y sin dirección IP.