

UNIVERSIDAD DE ALMERÍA

COMPLEMENTO TRABAJO FIN DE GRADO

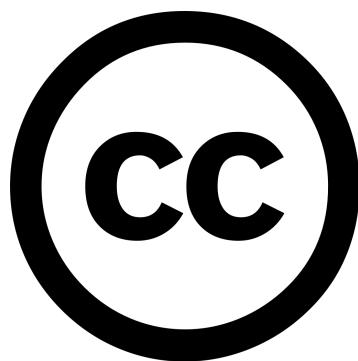
Utilización de un NIDS para redes SOHO (R-Snort)

Autor: Petrovics, Deian Orlando

Tutor: Gómez López, Julio

Cotutor: Padilla Soriano, Nicolás

1 de abril de 2025



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

Índice general

| | |
|---|----|
| Abreviaturas | 4 |
| Introducción | 6 |
| 1 Motivación | 7 |
| 2 Objetivos | 8 |
| 3 Fases de la realización y cronograma | 9 |
| 4 Estructura y metodología | 10 |
| 5 Sistemas de Detección de Intrusos | 11 |
| 5.1 Los sistemas de detección de intrusos | 11 |
| 5.1.1 Definición y propósito de los IDS | 11 |
| 5.1.2 Diferencia entre detección de intrusos y prevención de intrusos | 11 |
| 5.1.3 Tipos de IDS | 11 |
| 5.1.4 Basados en análisis de comportamiento vs. basados en firma | 12 |
| 5.1.5 Arquitectura y componentes de un IDS | 12 |
| 5.1.6 Funcionalidades | 12 |
| 5.1.7 Ventajas y limitaciones de los IDS | 12 |
| 5.1.8 Contexto actual | 13 |
| 5.1.9 Ejemplos y casos prácticos | 13 |
| 5.2 Snort | 13 |
| 5.2.1 Introducción a Snort | 13 |
| 5.2.2 Características principales | 13 |
| 5.2.3 Arquitectura de Snort | 14 |
| 5.2.4 Reglas y actualizaciones | 14 |
| 5.2.5 Personalización y extensibilidad | 14 |
| 5.2.6 Casos de uso y ejemplos | 15 |
| 5.2.7 Limitaciones y consideraciones | 15 |
| 5.3 Necesidades de seguridad en pequeñas redes | 15 |
| 5.3.1 Definición de pequeñas redes (SOHO) | 15 |
| 5.3.2 Amenazas y vulnerabilidades comunes | 16 |
| 5.3.3 Retos específicos | 16 |
| 5.3.4 Necesidades de seguridad | 16 |
| 5.3.5 Soluciones adecuadas para SOHO | 17 |
| 5.3.6 Beneficios de implementar un NIDS en pequeñas redes | 17 |

| | | |
|---|--|-----------|
| 5.3.7 | Buenas prácticas de seguridad para SOHO | 17 |
| 5.4 | Complementos y plugins para Snort | 18 |
| 6 | Utilización de un NIDS para pequeñas redes | 20 |
| 6.1 | Introducción | 20 |
| 6.2 | Especificaciones, características y requisitos | 20 |
| 6.2.1 | Especificaciones de Snort 3 | 20 |
| 6.2.2 | Requisitos del sistema | 21 |
| 6.3 | Entorno de trabajo | 21 |
| 6.3.1 | Esquema de red | 22 |
| 6.3.2 | Hardware (Raspberry Pi 5) | 22 |
| 6.3.3 | Software (Snort y sus complementos) | 22 |
| 6.4 | Instalación y personalización de complementos | 24 |
| 6.4.1 | Instalación de Snort V3 | 24 |
| 6.4.2 | Instalación de reglas y plugins | 39 |
| 6.4.3 | Configuración preprocesador HTTP Inspect | 41 |
| 6.4.4 | SSL Inspector | 43 |
| 6.4.5 | Stream IP | 43 |
| 6.4.6 | Stream TCP | 44 |
| 6.4.7 | Reputation | 44 |
| 6.4.8 | Datos sensibles | 45 |
| 6.4.9 | Antivirus ClamAV | 46 |
| 6.5 | Generación de un script para la instalación automática | 47 |
| 6.5.1 | Esquema de red de monitorización con R-SNORT | 47 |
| 6.5.2 | Primera fase del script automático | 48 |
| 6.5.3 | Transición a paquete .deb | 52 |
| 7 | Casos prácticos: utilización de R-Snort | 55 |
| 7.1 | Entorno de trabajo | 55 |
| 7.2 | Instalación | 55 |
| 7.3 | Utilización y pruebas | 62 |
| 7.3.1 | Benchmark de rendimiento | 62 |
| 7.3.2 | Pruebas | 68 |
| 7.4 | Resumen | 71 |
| Resultados y discusión | 72 | |
| Conclusiones | 73 | |
| A Anexo A | 75 | |
| B Anexo B | 76 | |

Abreviaturas

(Todavía en proceso)

- **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.
- **RAM:** Random Access Memory. Memoria principal de un sistema, necesaria para cargar y ejecutar aplicaciones.
- **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 creciente dependencia de la tecnología en el día a día ha resaltado la importancia de garantizar la seguridad en las redes informáticas, independientemente de su tamaño. Sin embargo, las pequeñas redes, como las utilizadas en oficinas y hogares, suelen estar desprotegidas debido a la falta de recursos técnicos y económicos, lo que las convierte en objetivos vulnerables para posibles ataques. Este trabajo propone una solución accesible y práctica que permita a estos entornos mejorar significativamente su nivel de seguridad mediante el desarrollo de un sistema de detección de intrusos (IDS) basado en herramientas de bajo coste y fácil implementación.

1. Motivación

Desde el inicio de mi formación, sentí una especial inclinación hacia la seguridad informática y la administración de sistemas. Asignaturas como Administración de Redes, Sistemas Operativos y Seguridad Informática me permitieron profundizar en los retos a los que se enfrentan redes pequeñas y medianas, evidenciando que, a diferencia de las grandes empresas, estos entornos carecen de medidas de protección adecuadas. Esta realidad despertó en mí el interés de buscar soluciones prácticas y asequibles para mitigar esta problemática, con el objetivo de hacer la seguridad informática más accesible para estos entornos.

2. Objetivos

El principal objetivo de este trabajo es diseñar y desarrollar un sistema de detección de intrusos (IDS) adaptado a pequeñas redes, empleando Snort y una Raspberry Pi. Los objetivos específicos serán:

- **Integración de herramientas complementarias:** incorporar plugins, antivirus y filtros de contenido para cubrir las necesidades de seguridad específicas de pequeñas redes.
- **Automatización y portabilidad:** crear un script automatizado que facilite la instalación y configuración del sistema en distintos entornos.
- **Evaluación de efectividad y rendimiento:** realizar pruebas para garantizar que el sistema sea eficiente y adecuado para su propósito en condiciones reales.

Se trata proporcionar una solución sencilla, económica y efectiva que impulse la seguridad en redes de menor escala.

3. Fases de la realización y cronograma

(EN DESARROLLO)

4. Estructura y metodología

(EN DESARROLLO)

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 seguridad que monitorea redes y sistemas en busca de actividades maliciosas o violaciones de políticas de seguridad. Su función principal es identificar accesos no autorizados o comportamientos anómalos que puedan comprometer la integridad, confidencialidad o disponibilidad de los sistemas informáticos.

5.1.2. Diferencia entre detección de intrusos y prevención de intrusos

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.

5.1.3. Tipos de IDS

Basados en host (HIDS)

Monitorizan y analizan la actividad interna de un sistema individual, revisando logs, integridad de archivos y procesos en ejecución.

Basados en red (NIDS)

Supervisan el tráfico de red en busca de actividades sospechosas, analizando paquetes que circulan por la red para detectar patrones de ataque.

5.1.4. Basados en análisis de comportamiento vs. basados en firma

Anomaly-based

Detectan desviaciones del comportamiento normal establecido, identificando actividades inusuales que podrían indicar una intrusión.

Signature-based

Buscan patrones conocidos de ataques comparando el tráfico con una base de datos de firmas de amenazas previamente identificadas.

5.1.5. Arquitectura y componentes de un IDS

Un IDS típico consta de:

- **Sensores:** capturan datos de la red o del host.
- **Analizadores:** procesan y examinan los datos recopilados para identificar posibles intrusiones.
- **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.

5.1.6. Funcionalidades

- **Monitoreo en tiempo real:** supervisa continuamente la actividad para detectar intrusiones de manera inmediata.
- **Generación de alertas:** notifica a los administradores sobre actividades sospechosas o confirmadas.
- **Registro de eventos:** documenta incidentes para análisis posterior y cumplimiento normativo.
- **Integración con otros sistemas de seguridad:** trabaja conjuntamente con firewalls, sistemas de gestión de eventos (SIEM) y otras herramientas para una defensa más robusta.

5.1.7. Ventajas y limitaciones de los IDS

Ventajas

- Detección temprana de amenazas.
- Mejora en la respuesta a incidentes.
- Cumplimiento de normativas de seguridad.

Limitaciones

- Posibilidad de falsos positivos y negativos.
- Necesidad de actualización constante para reconocer nuevas amenazas.
- Consumo de recursos y posible impacto en el rendimiento del sistema.

5.1.8. Contexto actual

Con el incremento de amenazas avanzadas y persistentes (APT), 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 eficaz.

5.1.9. Ejemplos y casos prácticos

Ataques como el gusano *SQL Slammer* en 2003 fueron detectados por IDS basados en firmas, mientras que amenazas más sofisticadas, como *Stuxnet* en 2010, 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 como uno de los NIDS más populares y versátiles, destacando sus características, funcionamiento y relevancia en entornos de seguridad.

5.2.1. Introducción a Snort

Desarrollado por Martin Roesch en 1998, Snort ha evolucionado hasta convertirse en un estándar en sistemas de detección de intrusiones de código abierto. En 2013, Cisco Systems adquirió Sourcefire, la empresa detrás de Snort, integrándolo en su portafolio de soluciones de seguridad.

5.2.2. Características principales

- **Motor de detección basado en firmas:** utiliza reglas para identificar patrones de ataque conocidos.
- **Modos de operación:**
 - **Sniffer:** captura y muestra paquetes en tiempo real.
 - **Packet Logger:** guarda paquetes en disco para análisis posterior.
 - **Network Intrusion Detection:** analiza tráfico y detecta intrusiones.

5.2.3. Arquitectura de Snort

- **Preprocesadores:** preparan el tráfico para su análisis, manejando tareas como la normalización y la desfragmentación.
- **Motor de detección:** aplica reglas al tráfico para identificar posibles amenazas.
- **Plugins de salida:** definen cómo se reportan las alertas, ya sea a través de archivos, bases de datos o syslog.

5.2.4. 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. 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.

Tipos de reglas

- **Community Rules:** reglas creadas y mantenidas por la comunidad de usuarios de Snort, disponibles de forma gratuita bajo la licencia GPLv2.
- **Registered Rules:** reglas proporcionadas por Cisco Talos, disponibles para usuarios registrados con un retraso de 30 días respecto a su lanzamiento para suscriptores.
- **Subscriber Rules:** reglas actualizadas en tiempo real, disponibles para suscriptores de pago, ofreciendo protección contra amenazas emergentes sin retrasos.

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.

5.2.5. 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.

5.2.6. Casos de uso y ejemplos

Implementación en diferentes entornos

Snort es utilizado en una amplia variedad de escenarios, desde pequeñas oficinas hasta grandes corporaciones y entornos educativos, gracias a su capacidad de adaptación.

Detección de ataques comunes

Snort 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, se tiene la capacidad de detectar casi cualquier tipo de actividad sospechosa gracias a las reglas creadas por la comunidad o por las del propio administrador de sistemas.

5.2.7. Limitaciones y consideraciones

Rendimiento en redes de alta velocidad

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

Gestión de falsos positivos

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

5.3. Necesidades de seguridad en pequeñas redes

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.

5.3.1. Definición de pequeñas redes (SOHO)

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.

5.3.2. Amenazas y vulnerabilidades comunes

Amenazas externas

Incluyen malware, phishing y ataques de fuerza bruta dirigidos a servicios expuestos a Internet.

Amenazas internas

Pueden surgir del uso inapropiado de recursos, dispositivos no seguros conectados a la red y falta de políticas de seguridad claras.

Dispositivos IoT

La proliferación de dispositivos IoT introduce nuevas vulnerabilidades debido a configuraciones inseguras y falta de actualizaciones.

5.3.3. Retos específicos

Limitaciones de recursos

Presupuestos reducidos y ausencia de personal especializado dificultan la implementación de soluciones de seguridad robustas.

Conectividad constante

La dependencia de servicios en la nube y el teletrabajo aumentan la superficie de ataque y requieren medidas de seguridad adicionales.

5.3.4. Necesidades de seguridad

Protección básica

Implementación de firewalls, antivirus y mantenimiento de software actualizado.

Monitoreo y detección

Es esencial conocer la actividad en la red; herramientas como Snort pueden proporcionar visibilidad y detección de amenazas.

Facilidad de uso y gestión

Las soluciones deben ser intuitivas y, preferiblemente, automatizar tareas como actualizaciones y generación de informes.

5.3.5. Soluciones adecuadas para SOHO

Herramientas de código abierto

Ofrecen ventajas en costo y flexibilidad, además de contar con comunidades de soporte activas.

Dispositivos de bajo costo

Hardware asequible, como *Raspberry Pi*, puede utilizarse para implementar soluciones de seguridad efectivas.

Servicios gestionados

Externalizar ciertas funciones de seguridad puede ser una opción viable para pequeñas empresas sin personal especializado.

5.3.6. Beneficios de implementar un NIDS en pequeñas redes

- **Detección temprana de amenazas:** permite identificar y responder rápidamente a actividades maliciosas.
- **Visibilidad del tráfico de red:** proporciona información sobre el comportamiento de la red y posibles vulnerabilidades.
- **Cumplimiento de normativas básicas de seguridad:** ayuda a cumplir con estándares y regulaciones aplicables.

5.3.7. Buenas prácticas de seguridad para SOHO

- **Políticas de contraseñas fuertes:** usar contraseñas robustas y cambiarlas periódicamente.
- **Segmentación de red:** separar segmentos de red para limitar el alcance de posibles intrusiones.
- **Educación y concienciación de usuarios:** capacitar al personal sobre prácticas seguras y reconocimiento de amenazas comunes.
- **Copias de seguridad regulares:** realizar backups periódicos para asegurar la recuperación ante incidentes.

5.4. 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. 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 un NIDS para pequeñas redes

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 ataques. En el caso de pequeñas redes, un NIDS puede proporcionar una primera línea de defensa sin la necesidad de costosas soluciones empresariales.

En este documento, se describe la instalación y configuración de Snort 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.

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.
- **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*, 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+)
- **Arquitectura:** x86_64 o ARMv8 (Raspberry Pi 4/5 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

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 8GB 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 32GB
 - 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.

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:

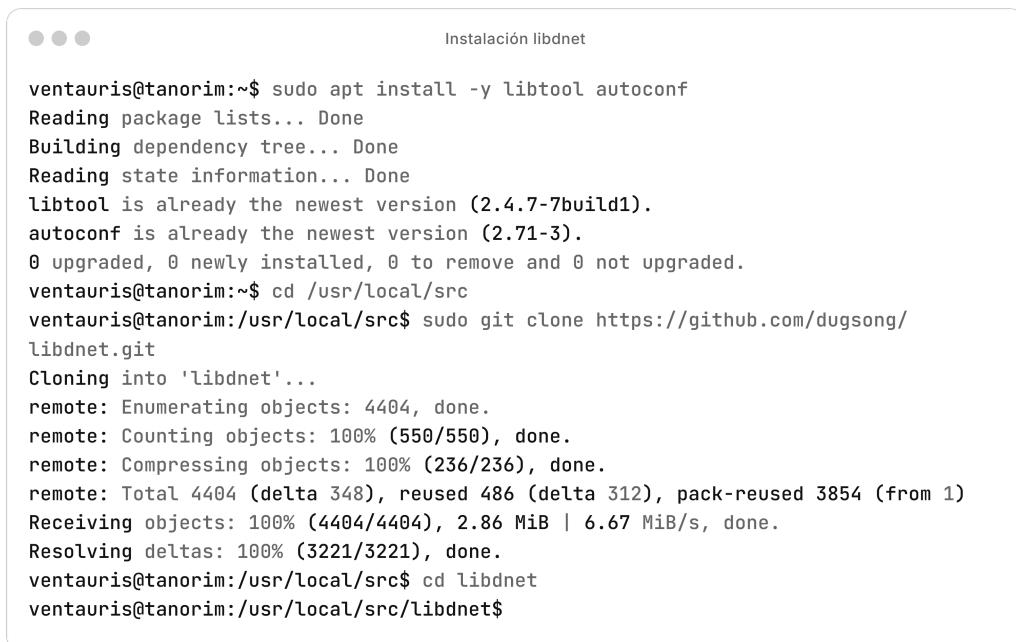
- **HTTP Inspect:** Analiza tráfico HTTP/HTTPS para detectar ataques SQL/XSS e irregularidades en los encabezados.
- **SSL/TLS:** Inspecciona metadatos del tráfico cifrado y detecta conexiones sospechosas.
- **Stream IP:** Ensambla paquetes IP fragmentados para detectar intentos de evasión.
- **Stream 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

El desarrollo de lo que será R-Snort empezará con la instalación de Snort mismo en su versión actual. Para ello, es buena práctica actualizar 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.

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.



```
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 6.1: Instalando dependencias necesarias.

Ahora instalamos el paquete check, que es una herramienta para ejecutar pruebas en C. Es un requisito para compilar libdnet correctamente. Con esto, seguimos preparando el entorno antes de compilar Snort 3.



```
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 6.2: Instalando dependencia check.

Aquí ejecutamos `./configure` para preparar la compilación de libdnet. Este script revisa el entorno del sistema, verifica dependencias y configura los archivos necesarios para compilar el código correctamente.



```
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 6.3: Configurando libdnet antes de la compilación.

Posteriormente ejecutamos sudo make para compilar libdnet. Este comando convierte el código fuente en ejecutables y bibliotecas listas para su instalación. Vemos que 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 Linux system named "ventauris" with the command "sudo make". The output of the command is displayed, showing the recursive nature of the build process as it enters and exits various source code directories like "include", "dnet", and "test". It also shows that there was nothing to be done for certain targets like "all" and "all-am". The terminal prompt "ventauris@tanorim:/usr/local/src/libdnet\$" is visible at the end.

```
ventauris@tanorim:/usr/local/src/libdnet$ sudo make
Making all in include
make[1]: Entering directory '/usr/local/src/libdnet/include'
make  all-recurse
make[2]: Entering directory '/usr/local/src/libdnet/include'
Making all in dnet
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 6.4: Compilando libdnet con make.

Ahora usamos 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.

```
● ● ● 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 6.5: Instalando libdnet en el sistema.

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.

```
● ● ● 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 6.6: Verificación de la instalación de libdnet.

Hemos editado 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 6.7: Configurando libdnet para que sea reconocido por el sistema.

Ahora configuraremos el entorno para que pkg-config pueda encontrar libdnet. Primeiro, 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 6.8: Añadiendo libdnet al PATH y verificando su reconocimiento.

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 6.9: 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 6.10: Preparando libdaq para su compilación.

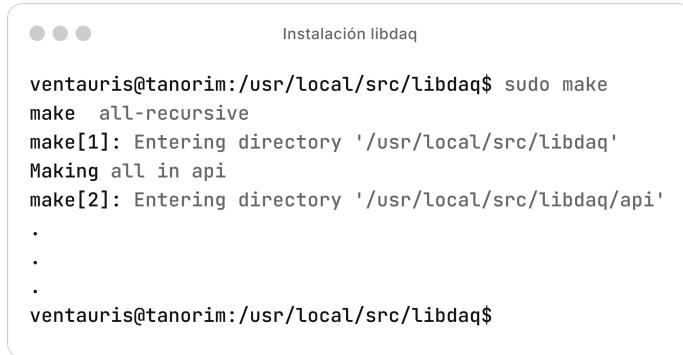
Ejecutamos ./configure para preparar el entorno de compilación de libdaq. Este script revisa que el sistema tenga todas las dependencias necesarias y genera los archivos de configuración adecuados para compilar el software sin problemas.



```
••• 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 6.11: Configurando libdaq antes de la compilación.

Ejecutamos make para compilar libdaq. Este proceso traduce el código fuente a binarios ejecutables, asegurándose de que todas las dependencias y archivos necesarios se generen correctamente.



```
••• 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 6.12: Compilando libdaq.

Ejecutamos sudo make install para instalar libdaq en el sistema. Esto copia los archivos compilados a sus ubicaciones correspondientes para que puedan ser utilizados por Snort y otros programas que 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 6.13: 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_pcapy.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 6.14: Verificando la instalación de libdaq.

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.

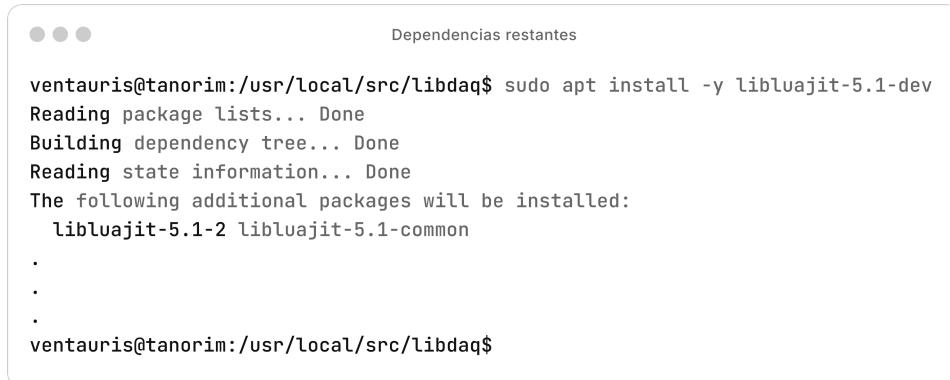


```
Dependencias restantes

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 6.15: Instalando dependencias necesarias.

Aquí instalamos libluajit-5.1-dev, más dependencias para Snort 3, ya que Snort 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.

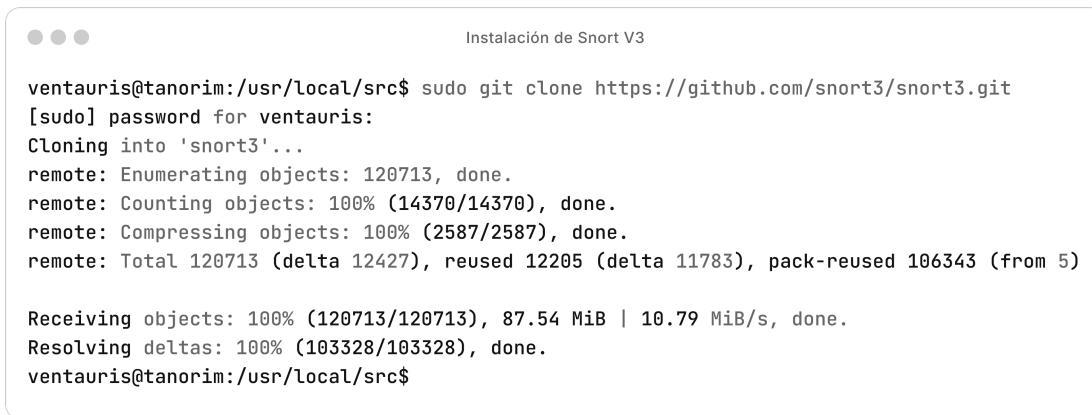


```
Dependencias restantes

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 6.16: Instalando LuaJIT para 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 6.17: 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 6.18: Definiendo la ruta de instalación de Snort.

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 6.19: 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 6.20: Instalando la biblioteca PCRE2 para el manejo de expresiones regulares.

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 6.21: Configurando Snort 3 con CMake.

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

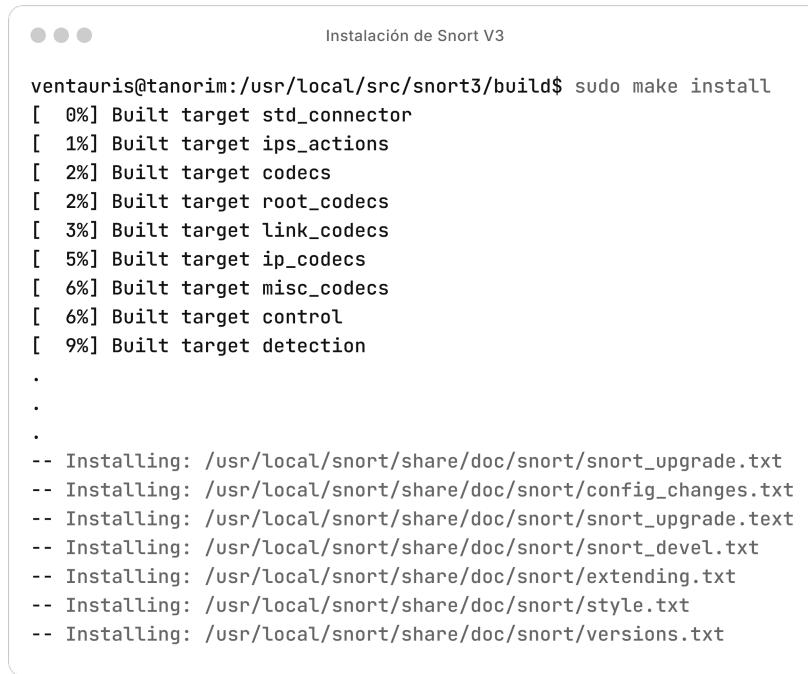


The image shows a terminal window titled "Instalación de Snort V3". The window has three gray dots at the top left. The terminal content displays the compilation process of Snort version 3. It shows numerous build logs indicating the compilation of various CXX objects and the linking of executables. The logs include paths like "tools/snort2lua/preprocessor_states/CMakeFiles/preprocessor_states.dir/pps_imap.cc.o" and "tools/snort2lua/init_state.cc.o". The compilation is nearly complete, with most objects reaching 100% completion. The final lines show the building of targets "snort2lua", "snort", and 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 6.22: 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.



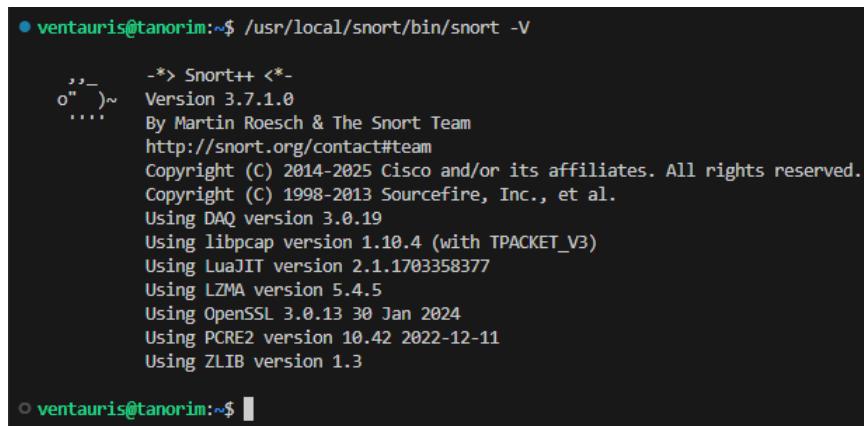
The terminal window shows the command "sudo make install" being run in the directory "/usr/local/src/snort3/build". The output displays the progress of building various targets (std_connector, ips_actions, codecs, root_codecs, link_codecs, ip_codecs, misc_codecs, control, detection) at 0% to 9% completion. It then lists the files being installed, including configuration files like snort_upgrade.txt, config_changes.txt, and several documentation files such as snort_devel.txt, extending.txt, style.txt, and versions.txt.

```
Instalación de Snort V3

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 6.23: Instalación de Snort.

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 terminal window shows the command "/usr/local/snort/bin/snort -V" being run. The output displays the Snort++ logo, version 3.7.1.0, copyright information from 1998-2013 Sourcefire, Inc., et al., and a list of dependencies and library versions used during compilation, including DAQ 3.0.19, libpcap 1.10.4, LuaJIT 2.1.1703358377, LZMA 5.4.5, OpenSSL 3.0.13, PCRE2 10.42, and ZLIB 1.3.

```
• 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 6.24: 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 6.25: Configuración de Snort validada con éxito.

6.4.2. Instalación de reglas y plugins

Tras la correcta instalación de Snort, nos aprovecharemos de la versión modular que nos trae la nueva versión 3, que maneja archivos con extensión .lua para manejar su configuración. A continuación se mostrará una guía paso a paso de como instalar las reglas básicas de la comunidad de Snort.

Descargamos las Community Rules de Snort con wget, descomprimiéndolas con tar.

```
● ● ● 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 6.26: Añadiendo reglas preconfiguradas a Snort.

Posteriormente editamos el archivo snort.lua, ubicado en /usr/local/snort/etc/snort/, para incluir las reglas de community.rules. Esto permite que Snort cargue estas reglas al iniciar y pueda detectar amenazas basadas en ellas. La edición se puede hacer con nano, vim o cualquier otro editor de texto.

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

Una vez agregadas las reglas a snort.lua, hacemos la prueba para comprobar que no ha habido ningún error.

```
● ● ● 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 6.28: Validación configuración de Snort.

6.4.3. Configuración preprocesador HTTP Inspect

Accedemos a snort.lua y buscamos http_inspect = y escribimos las reglas adecuadas para la gestión de una PYME. Las reglas incluyen:

| Parámetro | Valor | Uso en una PYME |
|---------------------|-------|--|
| request_depth | -1 | Inspeccionar todo el contenido de la petición (detección de inyecciones o malware en el cuerpo). |
| response_depth | -1 | Revisar la totalidad de la respuesta enviada por el servidor. |
| unzip | true | Descomprimir gzip/deflate (evita que contenido malicioso comprimido pase desapercibido). |
| oversize_dir_length | 500 | Alertar si una ruta de la URI supera longitud excesiva (potenciales intentos de ataque). |
| maximum_headers | 200 | Detectar exceso de cabeceras (protege ante ataques por cabeceras anómalas). |

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 6.29: Configuración http_inspect.

Además de la configuración se necesita agregar al final de la sección de binders en snort.lua el siguiente bloque de código para el correcto funcionamiento.

```

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 = 'dnsp3' }, use = { type = 'dns' } },
    { 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' } },

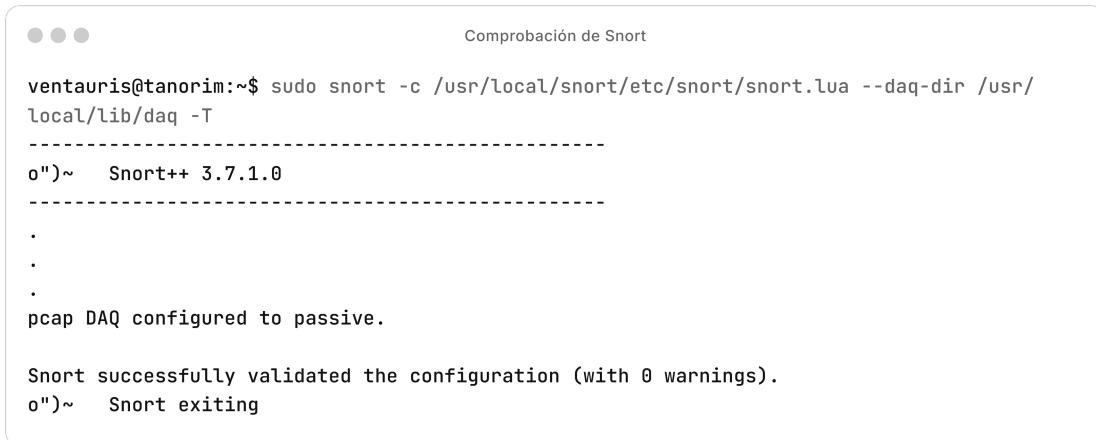
    { use = { type = 'wizard' } },
}

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

```

Figura 6.30: Configuración de binders.

Por último validamos la configuración para asegurarnos de que no hay errores.



```

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 6.31: Validación de la configuración.

6.4.4. SSL Inspector

El siguiente módulo que habilitaremos será el SSL Inspector, el procedimiento es similar al anterior. Comenzaremos por modificar el archivo snort.lua, buscamos el apartado de ssl y agregaremos lo siguiente.

```
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 6.32: Configuración de SSL.

Posteriormente vincularemos la configuración ssl en la sección binder en caso de no tenerlo.

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

Figura 6.33: Víncular con binders.

A continuación, podríamos verificar la configuración de snort, tras asegurarnos que la sintaxis es válida, reiniciamos snort mediante systemctl.

6.4.5. Stream IP

Antiguamente conocido como Frag3, ha sido actualizado a una nueva versión de nombre Stream IP. Su configuración será la siguiente 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 6.34: Configuración de Stream IP.

Validamos la sintaxis y reiniciamos snort de nuevo.

6.4.6. Stream TCP

Otro procesador que se conocía por el nombre de Stream5, su nombre ha cambiado a Stream TCP. Se configura de manera similar a Stream IP. Una vez hechos los cambios en snort.lua, validamos la sintaxis y reiniciamos Snort.

```
stream_tcp = {
    policy = 'linux',
    max_window = 1048576,
    overlap_limit = 10,
    max_pdu = 16384,
    reassemble_async = true,
    queue_limit = {
        max_bytes = 4194304,
        max_segments = 2048,
        asymmetric_ids_flush_threshold = 2097152, -- Umbral de descarga para flujos asimétricos (2 MB, protege memoria)
    },
    small_segments = {
        count = 5,
        maximum_size = 64,
    },
    session_timeout = 180,
    embryonic_timeout = 30,
    idle_timeout = 1800,
}
```

Figura 6.35: Configuración de Stream TCP.

6.4.7. Reputation

Este procesador trabaja bloqueando IPs sospechosas o que han sido documentadas como maliciosas usando listas. Para su configuración se ha creado una carpeta /reputation donde se guardará una lista de IPs sospechosas, las IPs se han obtenido de emergingthreats.net. Como ya es costumbre, tras realizar la configuración se valida la sintaxis de snort y se reinicia el servicio.

```
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 6.36: Configuración de Reputation.

6.4.8. Datos sensibles

En versiones anteriores de Snort existe el preprocesador Sensitive Data, sin embargo en las versiones más nuevas este se ha desechado, sin embargo vamos a adaptar algunas expresiones regulares para alertar si se comparten datos sensibles básicos de una PYME.

Creamos un nuevo archivo de reglas de nombre custom.rules y agregamos las siguientes reglas.

```
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 6.37: Expresiones regulares para protección de datos sensibles.

Agregamos la ruta donde se encuentran las expresiones regulares en snort.lua, comprobamos de nuevo la sintaxis de snort mediante libdaq y reiniciamos el servicio.

```
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 6.38: Agregación de custom.rules.

6.4.9. Antivirus ClamAV

Finalmente, la instalación del antivirus Clam AV.

```
● 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 6.39: Instalación ClamAV.

Detenemos el servicio de ClamAV actualizamos las firmas y volvemos a iniciar el servicio.

```
● 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 6.40: Instalación ClamAV.

6.5. Generación de un script para la instalación automática

6.5.1. Esquema de red de monitorización con R-SNORT

Para permitir la inspección total del tráfico en una red local, se diseñó un esquema de red específico que garantiza que todo el tráfico entre dispositivos —así como el que entra y sale hacia Internet— pueda ser observado por el sistema R-SNORT.

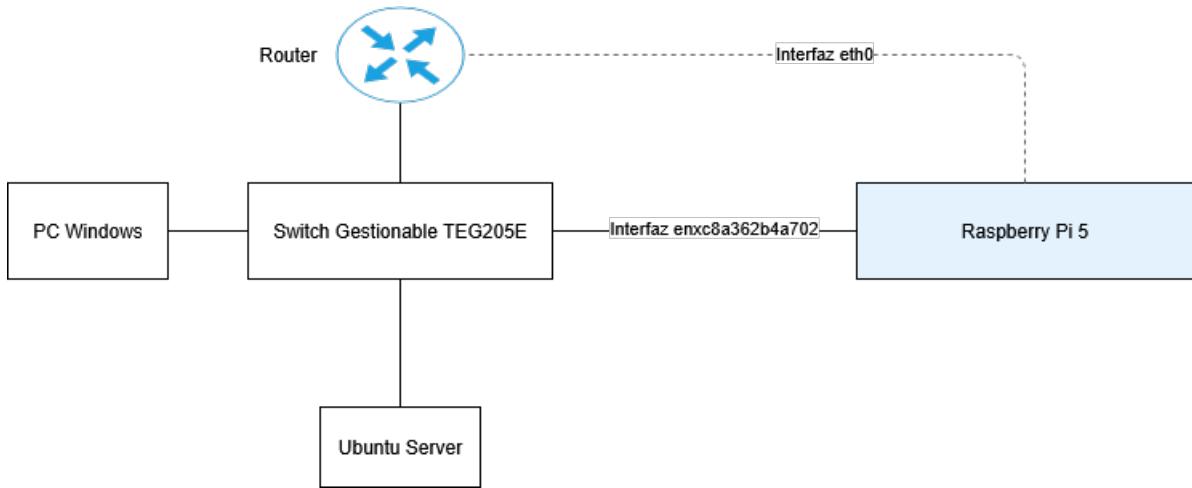


Figura 6.41: Esquema de red real del proyecto.

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** con dos interfaces de red Ethernet:
 - **eth0**: conectada al router para conectividad del propio sistema R-SNORT.
 - **enxc8a362b4a702**: conectada al puerto 4 del switch configurado como puerto espejo.

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 (Raspberry Pi), permitiendo que la interfaz **enxc8a362b4a702** de la Pi capture absolutamente todo el tráfico local e Internet.

Objetivo de este diseño

El objetivo principal es garantizar que R-SNORT actúe como un sistema de detección de intrusiones pasivo (*Network-based IDS*) sin interferir en el flujo real de los datos. Para ello se necesitan dos condiciones técnicas:

1. La interfaz utilizada para la captura debe estar en **modo promiscuo**, sin dirección IP asignada, para actuar como una sonda pasiva y evitar interferencias en la red.
2. El switch debe soportar **duplicación de tráfico** (mirroring) para que todo el tráfico entre dispositivos y hacia/desde Internet sea reenviado a la interfaz de análisis.

Justificación técnica

Este tipo de arquitectura es habitual en entornos profesionales donde se busca realizar inspección profunda del tráfico (*Deep Packet Inspection*) sin introducir latencia ni puntos únicos de fallo en la red. Además, permite mantener a R-SNORT completamente aislado de los sistemas a proteger, reforzando así la seguridad de la propia solución IDS.

El uso de una Raspberry Pi 5 se justifica por su bajo consumo, arquitectura ARM eficiente, y capacidad de operar con interfaces Ethernet Gigabit, siempre que se utilice un adaptador USB 3.0 como segundo puerto.

Alternativas descartadas

Se valoró brevemente implementar una solución enrutada —donde el tráfico pase directamente a través de la Raspberry Pi—, sin embargo, esto:

- Aumentaría la latencia para todos los dispositivos de la red.
- Requiere configuración NAT/bridge compleja en la Raspberry.
- Introduce un punto único de fallo.

Por tanto, la arquitectura basada en **switch con mirroring** y sonda pasiva resulta más robusta, profesional y realista para el entorno de una 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 rutinario 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 en la Figura B.5.

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-dev
3      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 1.5GB 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_';
2          ↪ then
3              apt-get install --reinstall -y xz-utils liblzma5
4                  ↪ liblzma-dev
5      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/
3             ↪ snort/etc/snort/snort.lua -i eth0 -A alert_fast
4 Restart=always
5 User=root
6 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 recomendaciones del ecosistema Debian.

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

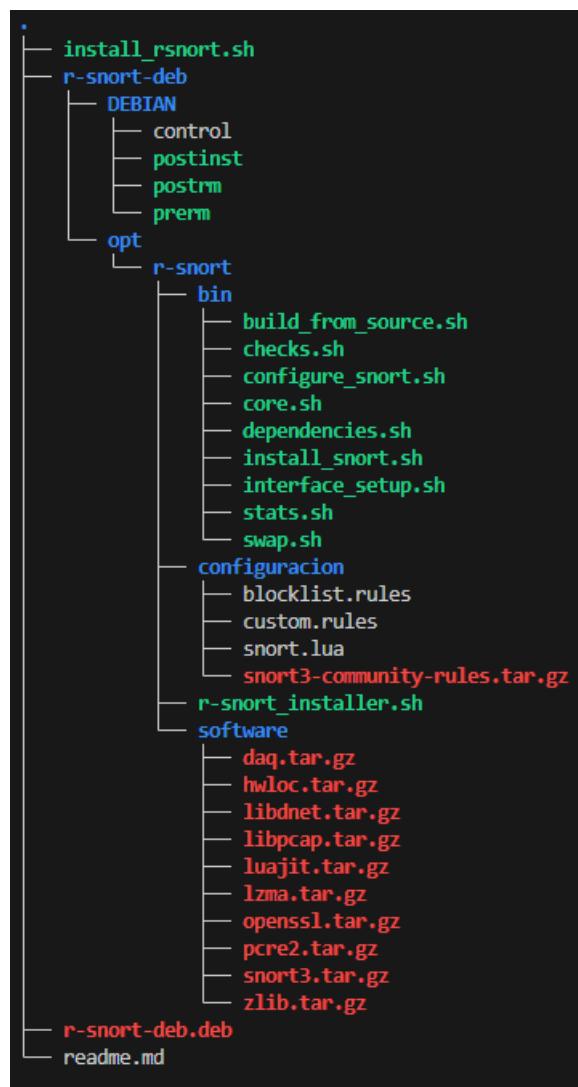


Figura 6.42: Estructura del proyecto reempaquetado como archivo .deb.

Mejoras introducidas

La transición permitió integrar mejoras que antes solo eran posibles manualmente:

- **Selección dinámica de interfaz:** se desarrolló un script de instalación previo, `install_rsnort.sh` (ver Figura B.13), que detecta las interfaces Ethernet disponibles y permite al usuario seleccionar la que se conectará al puerto espejo del switch. Este valor se guarda en `/etc/rsnort_iface` para su uso posterior.
- **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 es un paso de especial importancia para que Snort pueda inspeccionar correctamente todo el tráfico que recibe del puerto espejo del switch.

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`, permitiendo un arranque automático y controlado del servicio cada vez que el sistema se reinicia.
- **Fortalecimiento con ClamAV:** Para añadir una capa extra de protección, se incorporó la instalación y activación automática de ClamAV, útil en entornos donde Snort podría capturar tráfico de archivos contaminados.

Justificación del rediseño

Este rediseño responde no solo a criterios técnicos, sino también de usabilidad. En entornos reales como pequeñas y medianas empresas que desean adoptar soluciones NIDS es importante que la herramienta pueda instalarse, configurarse y mantenerse sin conocimientos avanzados de 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.
3. El paquete se instala y configura sin intervención adicional.

Esta arquitectura modular empacada sigue las mejores prácticas de ingeniería de software y permite escalar, actualizar o personalizar R-SNORT en el futuro.

Resultado final

Gracias a la evolución a paquete .deb, R-SNORT se convierte en una herramienta más robusta y amigable con el usuario final, hay que recordar que este proyecto trata de traer una solución eficaz y sencilla a redes SOHO. Con un enfoque modular interno y una experiencia de instalación unificada para el usuario final, el sistema mantiene su potencia de inspección sin perder simplicidad.

7. Casos prácticos: 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 provee conectividad a Internet a toda la red.
- Dos equipos clientes: un PC con Windows y otro con Ubuntu Server, ambos conectados al switch.

El puerto 4 del switch se ha configurado como destino del *mirroring*, recibiendo una copia del tráfico de los puertos 1 a 3 (que contienen al router y los dos PCs) de esta manera R-Snort será capaz de recibir una copia de tráfico local como paquetes provenientes de internet. La Raspberry escucha dicho tráfico a través del adaptador UGREEN, el cual se pone automáticamente en modo promiscuo, sin dirección IP asignada, para evitar interferencias con 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 pregunta al usuario por la interfaz de red conectada al switch.

2. Instalación del paquete `.deb`, que realiza la compilación de 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 pasando por la instalación hardware como software:

La Raspberry Pi 5 cuenta con dos interfaces Ethernet: una conectada directamente al router para obtener acceso a internet 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 gracias al port mirroring.

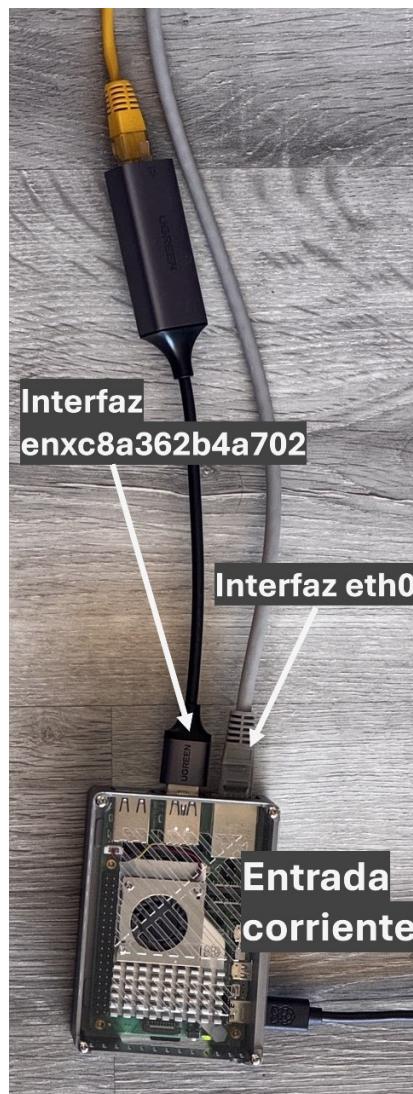


Figura 7.1: Raspberry Pi conectada con dos interfaces Ethernet.

Asignación de dispositivos al switch

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

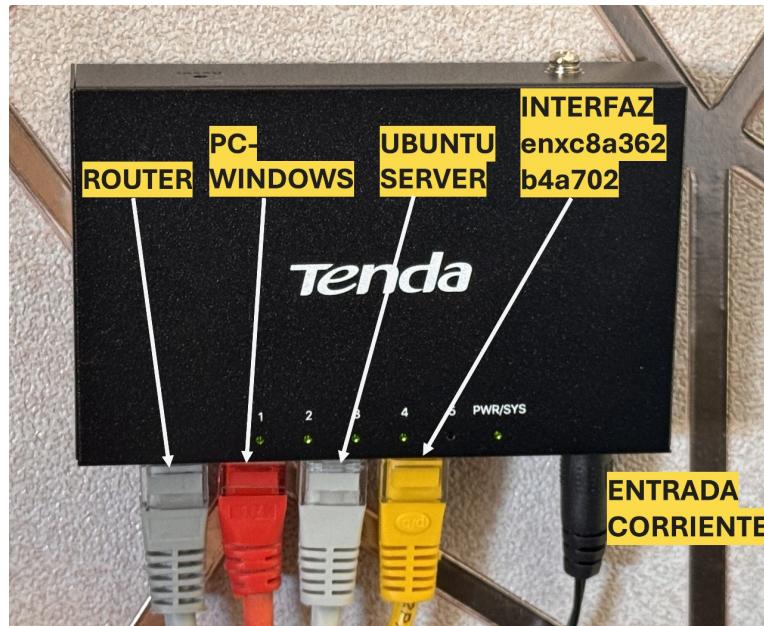


Figura 7.2: Conexiones físicas al switch gestionable.

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 respondiendo a su dirección IP.

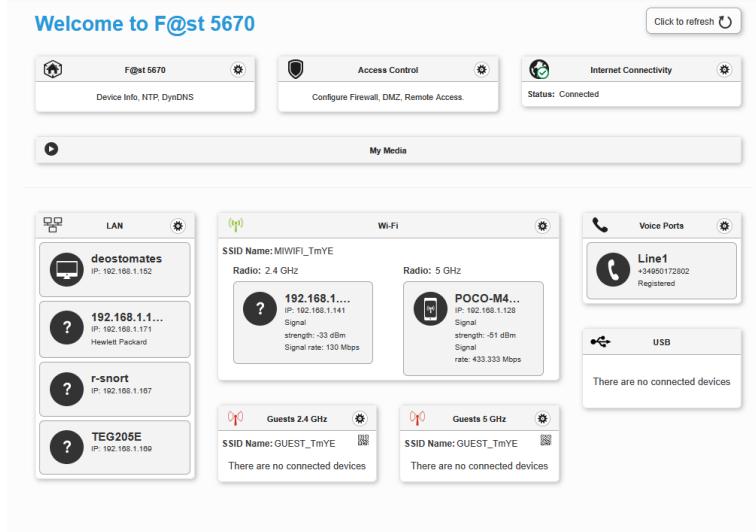


Figura 7.3: Dispositivos visibles en la red doméstica.

Acceso al switch Tenda

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

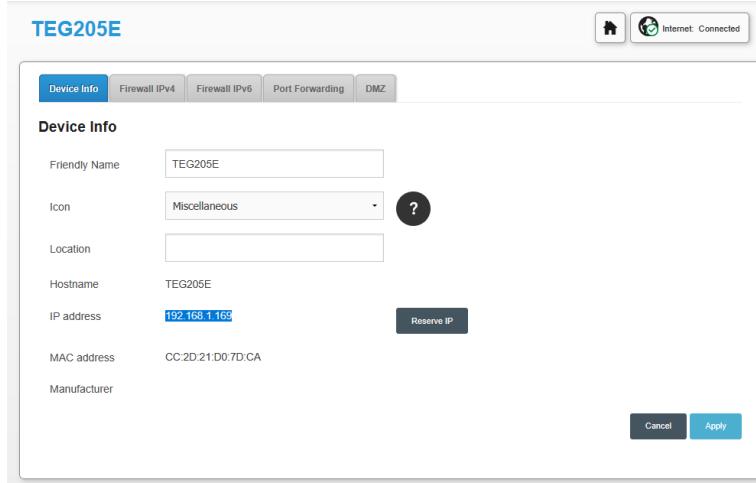


Figura 7.4: IP local del switch detectada desde el router.

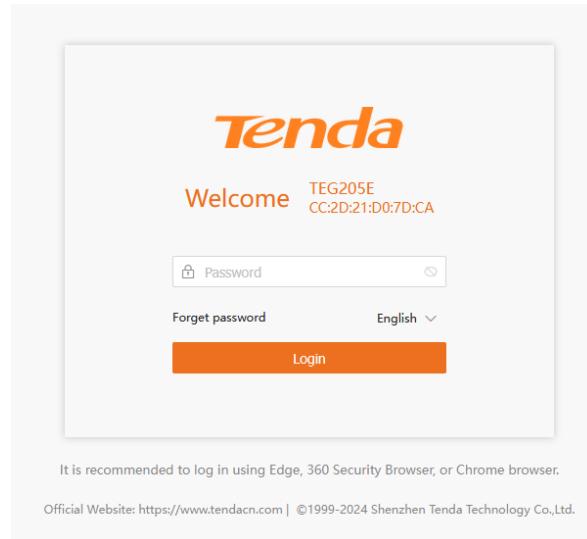


Figura 7.5: Pantalla de acceso al panel de administración.

Seguridad inicial del dispositivo

En el primer acceso al switch, el sistema requiere cambiar la contraseña predeterminada para mejorar la seguridad.

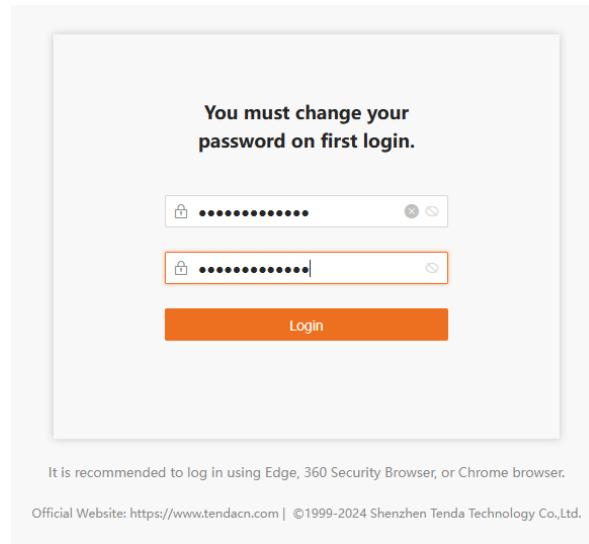


Figura 7.6: Cambio obligatorio de contraseña en el primer inicio.

Configuración avanzada del switch

Una vez autenticados, accedemos a la interfaz de administración donde se pueden configurar los diversos parámetros del switch. Desde esta sección se define el port mirroring, se activan o desactivan puertos y se observa el estado de la red.

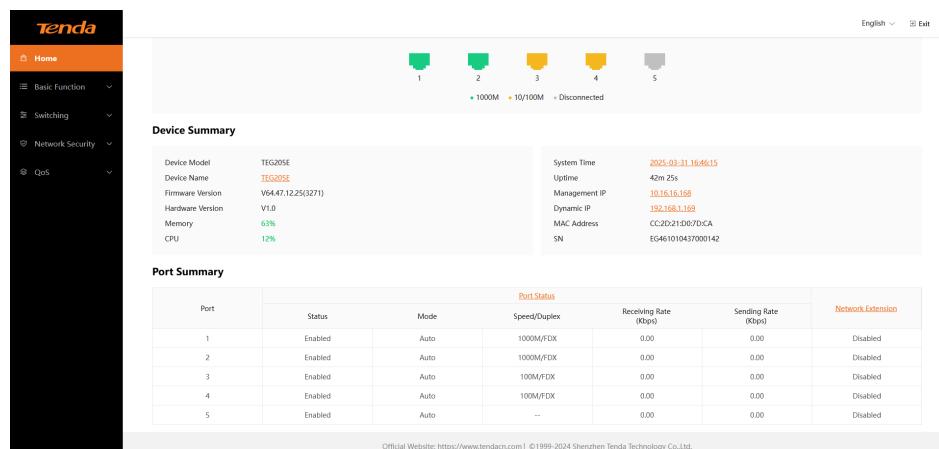


Figura 7.7: Vista general del panel de gestión del switch.

Activación del Port Mirroring

Accedemos a switching>port mirroring. El port mirroring es una funcionalidad que nos permitirá acceder a todo el tráfico de red tanto local como externo. Se configura replicando el tráfico de los puertos donde están conectados el router y los PCs (puertos 1, 2 y 3) hacia el puerto 4, que está conectado a la interfaz de análisis de la Raspberry Pi. Esto replicará todo el tráfico local y de internet al puerto de la Raspberry Pi permitiendo a Snort realizar la inspección.

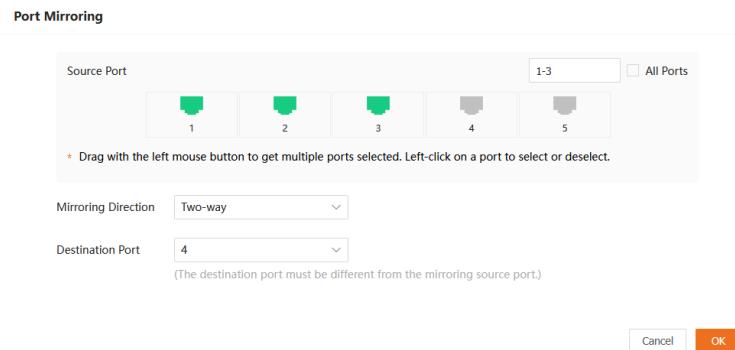


Figura 7.8: Verificación del estado de los puertos tras la configuración.

Por último ejecutamos como administrador `installer_snort.sh` para que dé comienzo a la instalación de R-Snort, seleccionamos la interfaz de red correspondiente al puerto 4 del switch que es el que recibirá el tráfico de los demás puertos y esperemos a que acabe la instalación. Para obtener el código fuente del instalador, puede consultarse el Anexo A.

Figura 7.9: *

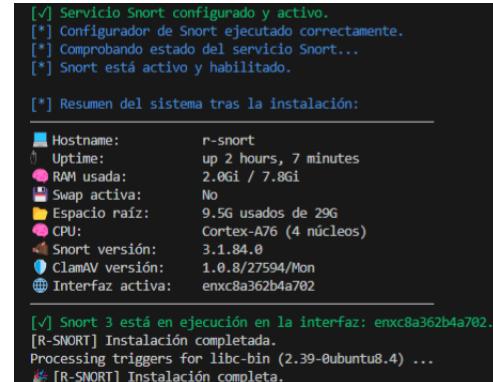


Figura 7.10: *

Figura 7.11: Proceso completo de instalación automática de R-Snort.

7.3. Utilización y pruebas

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

7.3.1. Benchmark de rendimiento

Metodología

Para medir el rendimiento de R-Snort, se utilizó la herramienta `dstat`, la cual permite monitorizar 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, permitiendo establecer una línea base del consumo del sistema sin el IDS.

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

Resultados e interpretación de gráficas

A continuación, se presentan las gráficas obtenidas:

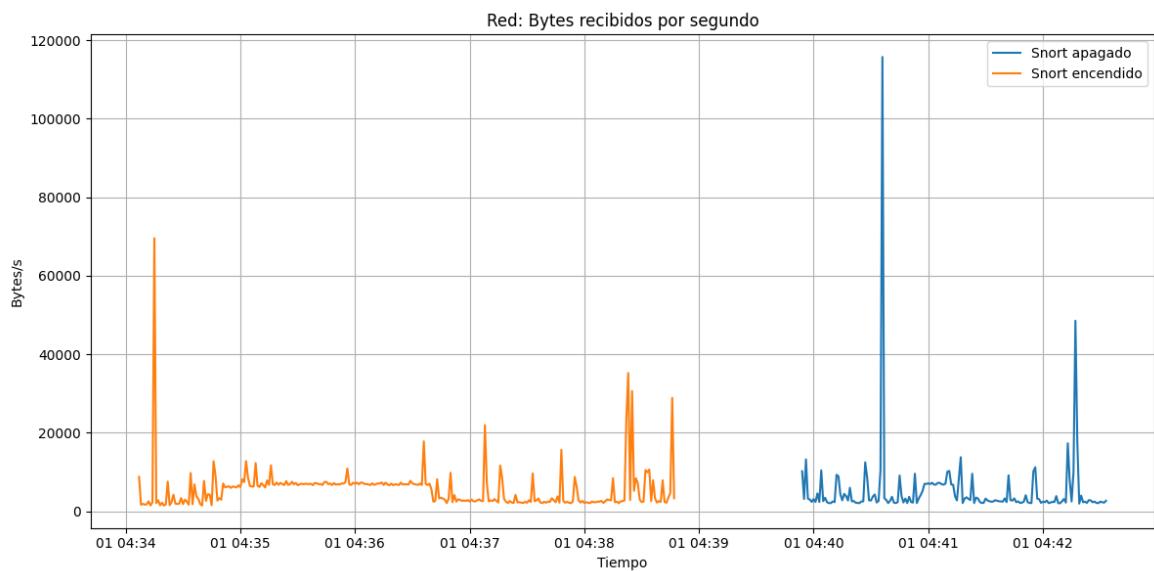


Figura 7.12: Red: Bytes recibidos por segundo

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

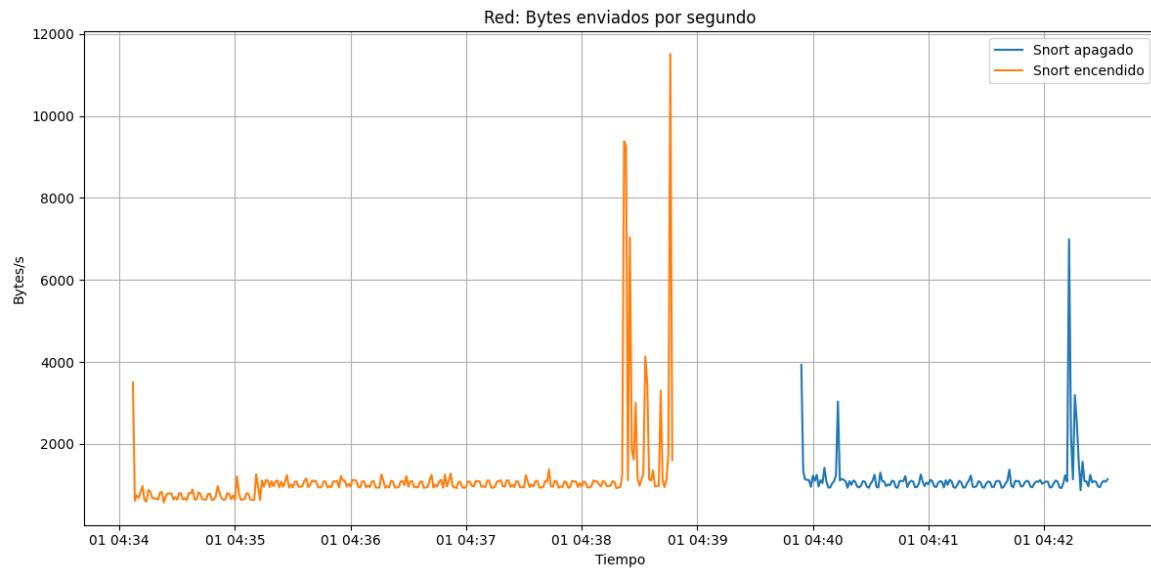


Figura 7.13: Red: Bytes enviados por segundo

Análisis: El patrón de tráfico enviado es mayoritariamente bajo en ambas sesiones, aunque con Snort encendido se observa un ligero aumento en los picos. Esto podría atribuirse a respuestas generadas por Snort al detectar ciertos paquetes como por ejemplo las pruebas que involucran más actividad como copiado de datos y escaneo de puertos.

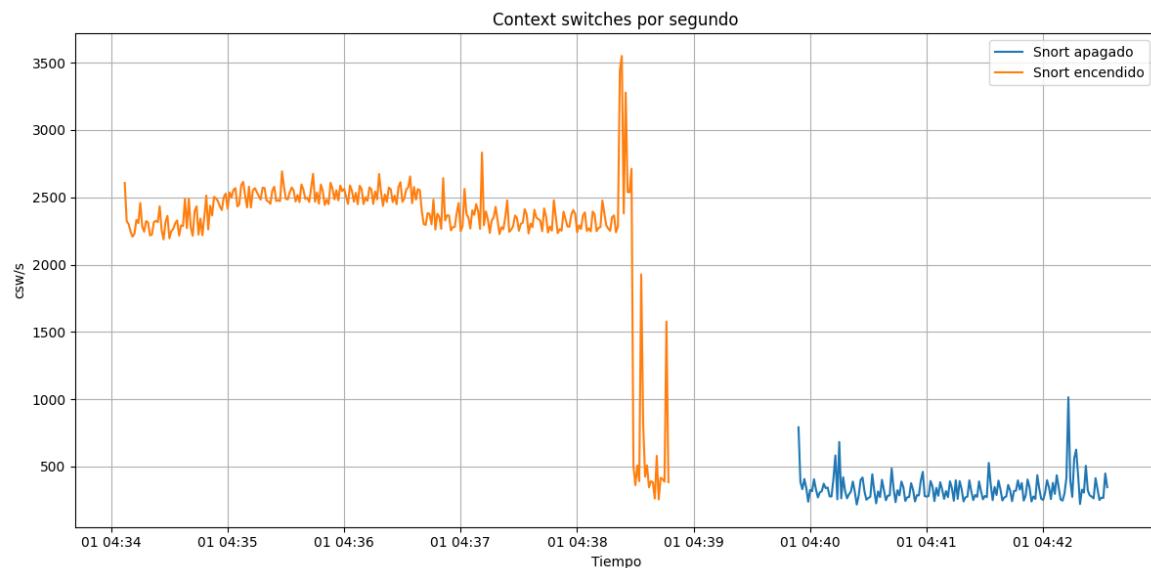


Figura 7.14: Context switches por segundo

Análisis: El número de cambios de contexto se incrementa notablemente cuando Snort está activo. Esto es esperable ya que analiza paquetes en tiempo real, forzando al sistema a alternar con mayor frecuencia entre procesos.

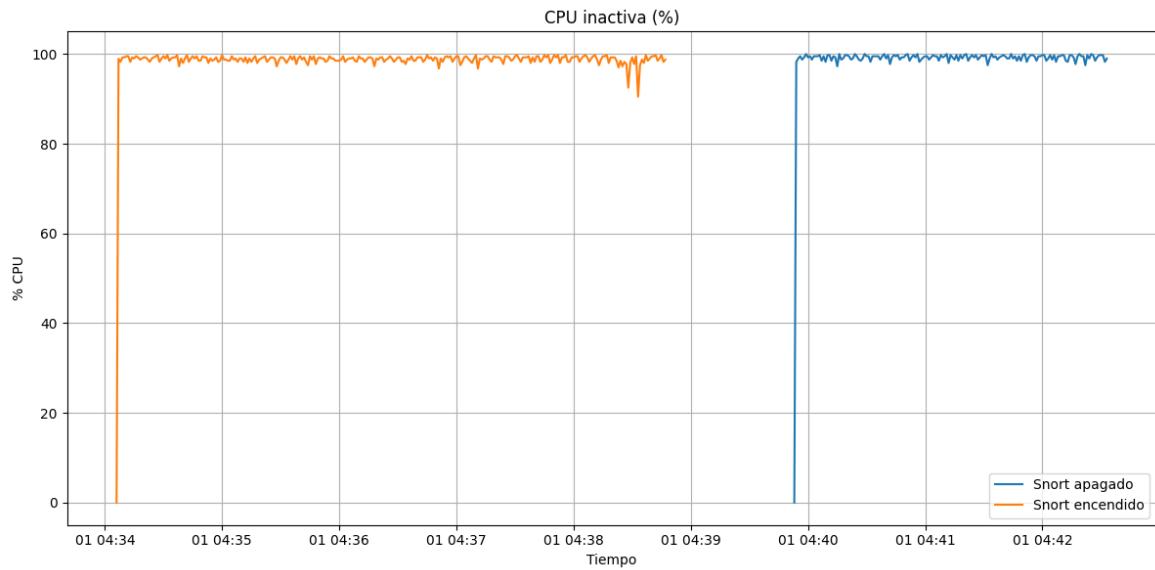


Figura 7.15: CPU inactiva (%)

Análisis: La CPU permanece prácticamente inactiva cuando Snort está apagado. Con Snort encendido, aunque la carga sigue siendo ligera, se reduce levemente el porcentaje de inactividad, lo que indica que Snort tiene un impacto ligero pero medible en el uso de la CPU.

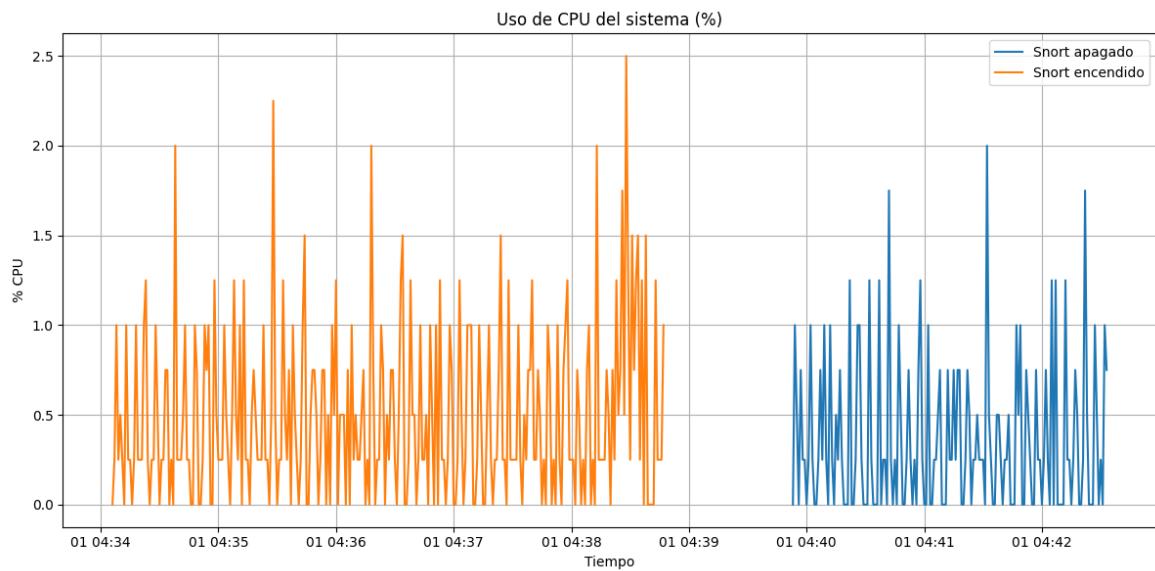


Figura 7.16: Uso de CPU del sistema (%)

Análisis: Se aprecia un ligero incremento en el uso del CPU en modo sistema cuando Snort está encendido. Esto es coherente con el hecho de que Snort realiza muchas operaciones a nivel de kernel (por ejemplo, captura de paquetes con libpcap).

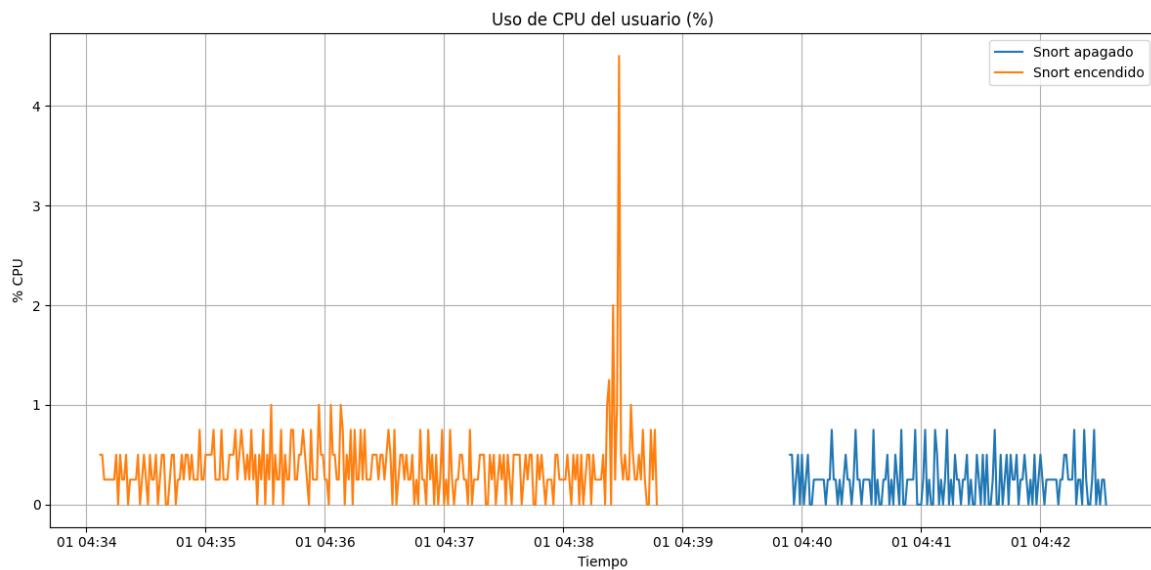


Figura 7.17: Uso de CPU del usuario (%)

Análisis: El impacto en el espacio de usuario es moderado. Aunque no es extremadamente alto, hay una clara diferencia entre el sistema con Snort encendido y apagado, lo que indica actividad computacional asociada al motor de detección de Snort.

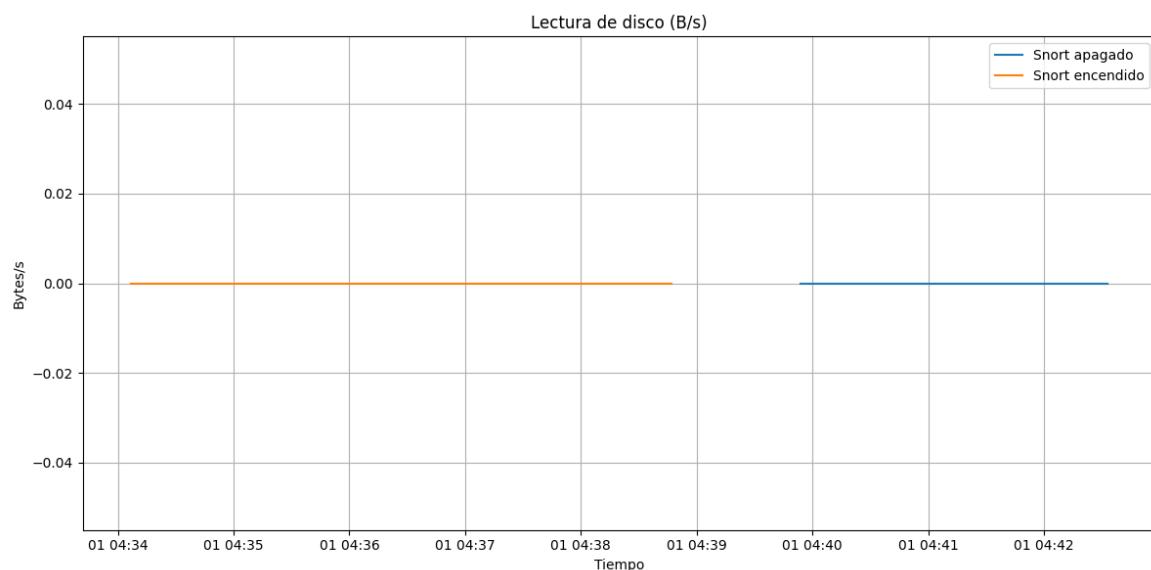


Figura 7.18: Lectura de disco (B/s)

Análisis: No se registran lecturas de disco en ninguna de las dos condiciones, lo cual es esperado si Snort está funcionando en memoria y no realizando operaciones de lectura intensiva desde disco.

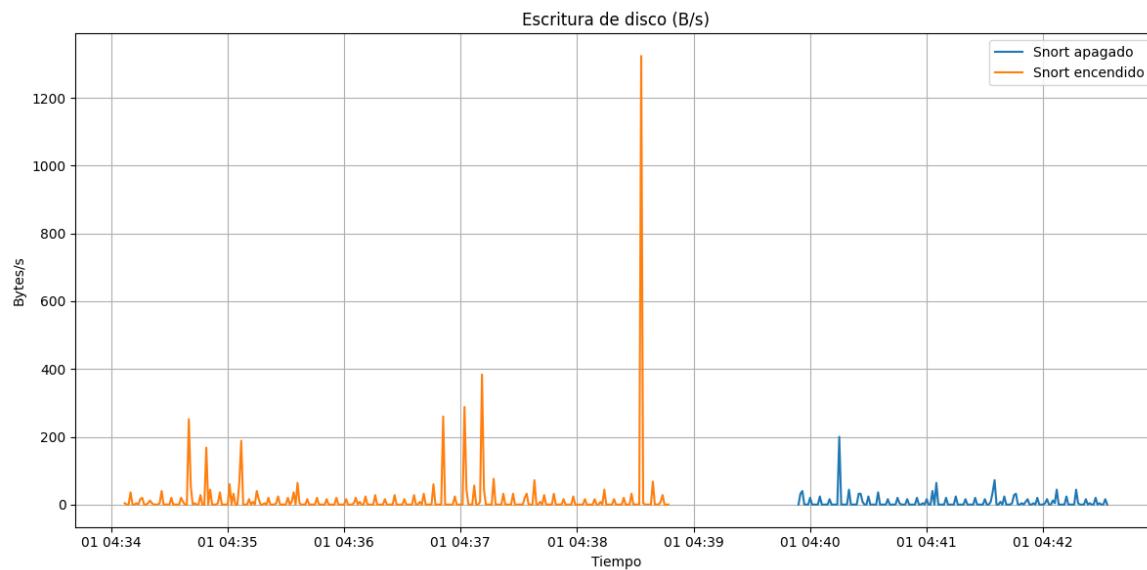


Figura 7.19: Escritura de disco (B/s)

Análisis: Se observa una mayor actividad de escritura en disco cuando Snort está activo. Esto corresponde al registro de alertas y logs generados por el sistema.

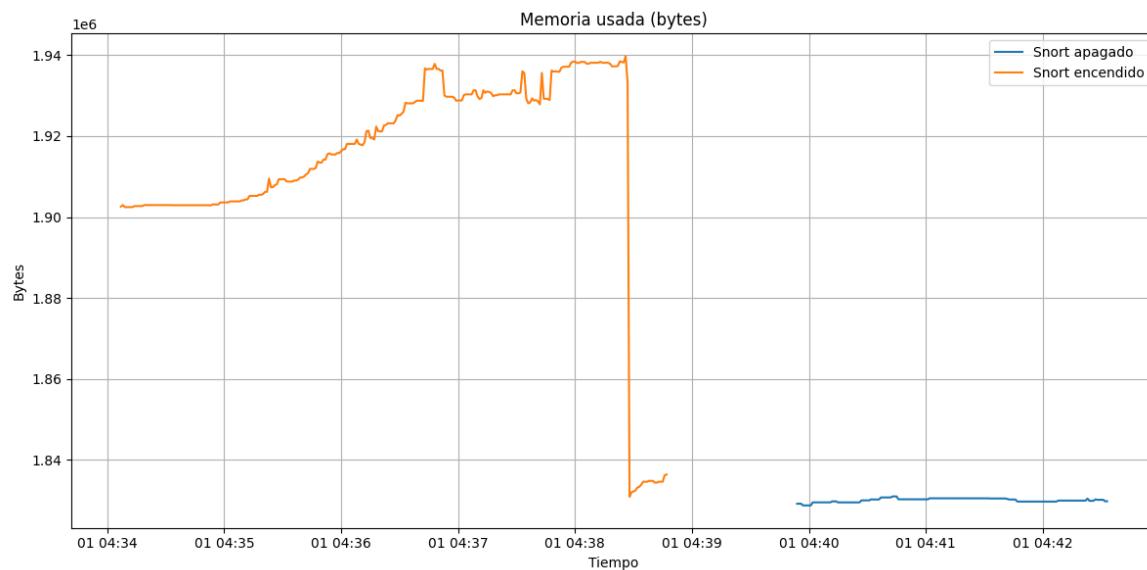


Figura 7.20: Memoria usada (bytes)

Análisis: Hay un incremento progresivo en el uso de memoria cuando Snort está encendido, esto se debe a la carga de reglas, buffers de captura y estructuras de datos internas. Al desactivar Snort, el consumo baja visiblemente.

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 presencia es claramente detectable, lo cual es esperable para un sistema de detección de intrusos activo en tiempo real.

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

- **Carga del sistema:** se ha monitorizado el uso de CPU y RAM durante escenarios con carga de red baja, media y alta. Esto se ha hecho utilizando herramientas como `htop`, `free -h` y el script de estadísticas internas de R-Snort.
- **Capacidad de análisis:** se ha evaluado cuántos paquetes por segundo puede procesar Snort sin pérdidas ni errores, utilizando herramientas de generación de tráfico como `iperf3` y `ping flood` desde los dos PCs.

Para poner estos datos en perspectiva, se han comparado con estudios previos en la literatura sobre el rendimiento de sistemas IDS en dispositivos embebidos. Estos resultados validan 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 comprobar el correcto funcionamiento del sistema:

- **Detección de escaneos:** se ha lanzado un escaneo Nmap desde el PC con Ubuntu Server al PC con Windows, comprobando que se generan alertas en el frontend.
- **Generación de tráfico legítimo:** navegación web, transferencia de archivos y conexión SSH para asegurar que Snort distingue correctamente entre tráfico benigno y potencialmente malicioso.
- **Visualización en el panel web:** las alertas generadas por Snort son visualizadas en tiempo real en el dashboard desarrollado con Angular y Spring Boot.

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

En esta sección se muestran pruebas reales llevadas a cabo en el entorno propuesto (ver topología en la sección anterior), donde R-Snort ha demostrado su capacidad de detección mediante reglas tanto personalizadas como comunitarias. Cada caso práctico se acompaña de la alerta generada y una breve descripción técnica del contexto.

1. PINGs malformados y tráfico ICMP inusual

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

```
ping 8.8.8.8
```

Snort generó varias alertas relacionadas con tráfico ICMP, incluyendo paquetes de tipo Unix ping, Echo reply y detecciones de PINGs anómalos. Estas alertas confirman que los preprocesadores stream, icmp y la inspección de TTLs (activada en stream_ip) están funcionando correctamente.

Configuración personalizada validada: uso de valores no por defecto para 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 7.21: Tráfico ICMP detectado por los preprocesadores y reglas community.

2. Fugas de datos sensibles (email y tarjetas)

Para comprobar el correcto funcionamiento de las reglas personalizadas definidas en `custom.rules`, se simularon envíos 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 y generó alertas para ambos casos, reconociendo tanto direcciones de correo como números de tarjeta de crédito mediante expresiones regulares.

Configuración personalizada validada: reglas propias usando pcre y flow en `custom.rules`, integradas en `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 7.22: Alertas generadas por reglas personalizadas ante filtraciones de datos.

3. Tráfico DNS sospechoso detectado automáticamente

Sin necesidad de ejecutar comandos manuales, el sistema detectó respuestas DNS con un TTL muy bajo y sin autoridad, lo que Snort considera un posible intento de spoofing.

Estas alertas surgieron del tráfico generado al navegar desde el PC con Windows, mostrando la efectividad de las reglas community aplicadas sobre el preprocesador DNS.

Configuración personalizada validada: activación del módulo `dns` mediante el bloque `binder` y `default_mod`.

```
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 -> 192.168.1.152:51844
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 -> 192.168.1.152:68306
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 -> 192.168.1.152:51844
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 -> 192.168.1.152:68306
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
04/01-03:13:52.108057 [**] [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 -> 192.168.1.152:68306
```

Figura 7.23: Respuestas DNS con TTL anómalo detectadas como tráfico malicioso.

4. Escaneo SNMP detectado con Nmap

Para probar el módulo de SNMP, se lanzó un escaneo de puertos UDP desde Ubuntu al equipo Windows utilizando:

```
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, registrándolos como intentos de fuga de información.

Configuración personalizada validada: activación explícita de `rpc_decode` y `snmp` en el archivo `snort.lua`, junto con las reglas `community`.

```
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
04/01-03:42:11.982680 [**] [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 7.24: Detección de actividad SNMP sospechosa tras escaneo dirigido.

5. Detección de NUSS (Número de Seguridad Social)

Finalmente, se verificó la última de las reglas personalizadas, diseñada para detectar números de 12 cifras que simulan NUSS españoles. El comando utilizado fue:

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

Snort detectó correctamente este patrón como una posible violación de privacidad, asignándole la clasificación y prioridad adecuada.

Configuración personalizada validada: expresión regular personalizada para el patrón NUSS, classtype propia, y prioridad 1 asignada manualmente.

```
04/01-03:47:25.600085 [**] [1:384:8] "PROTOCOL_IOMP PING" [**] [Classification: Misc activity] [Priority: 3] {IOMP} 192.168.1.169 -> 142.250.184.164  
04/01-03:47:25.615235 [**] [1:408:8] "PROTOCOL_IOMP Echo Reply" [**] [Classification: Misc activity] [Priority: 3] {IOMP} 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 7.25: Alerta generada por la regla personalizada de detección de NUSS.

7.4. Resumen

Tras completar la instalación de R-Snort, se revisa su comportamiento al analizar el tráfico de una red real y se mide su efecto sobre los recursos de la Raspberry Pi. Se usa la herramienta dstat para comparar el sistema con Snort encendido y con Snort apagado, y se confirma que, aunque se aprecia algo más de uso de CPU, memoria y escritura en disco, el impacto sigue siendo razonable para un dispositivo con limitaciones de hardware. El tráfico de red se ve levemente alterado cuando Snort está en marcha, lo que sugiere que el procesamiento en tiempo real produce una ligera contención de paquetes.

También se realizan ensayos prácticos para probar la capacidad de detección. Desde escaneos de puertos con Nmap hasta envío de datos sensibles (como correos electrónicos o números de tarjeta), R-Snort reacciona y emite alertas apropiadas. Se ve que maneja bien la diferencia entre tráfico común (transferencias de archivos o simples pings) y acciones sospechosas, lo que se refleja en el panel web que muestra las alertas en tiempo real. En conjunto, estos experimentos confirman que el sistema puede proteger una red pequeña sin saturar el hardware y, sobre todo, sin complicar en exceso la experiencia de uso.

Resultados y discusión

Conclusiones

Después de probar R-Snort y analizar tráfico real, lo que salta a la vista es que su consumo de recursos es bastante moderado. Aunque se nota algo más de actividad en la CPU y en la memoria cuando Snort está analizando paquetes, el sistema no se viene abajo ni genera molestos retrasos. Es sorprendente lo bien que se comporta la Raspberry pese a sus modestos recursos.

Durante las pruebas, la herramienta capturó todo tipo de sucesos, desde simples pings a direcciones como 8.8.8.8 hasta ataques simulados con Nmap. También reaccionó ante envíos de datos delicados, como números de tarjeta, y alertó puntualmente cuando detectó cadenas de texto que podrían indicar fugas de información. Para quien está vigilando la red, es reconfortante ver que, en la práctica, el sistema lanza avisos cuando descubre algo fuera de lo normal.

No menos interesante es comprobar cómo, pese a estar pendiente de cada paquete, R-Snort no impide navegar ni transferir archivos con fluidez. Mientras descargaba un fichero grande o abría varias páginas web, todo continuaba con normalidad, solo que con un vigilante silencioso analizando en segundo plano. Aunque si hay un pico de tráfico elevado, la Raspberry trabaja un poco más, pero aun así sigue respondiendo.

En síntesis, lo que se concluye es que instalar R-Snort en un dispositivo tan manejable como una Raspberry Pi deja un entorno bien protegido sin sacrificar la experiencia habitual de uso. Se aprecian alertas útiles, un impacto en los recursos bastante discreto y la certeza de que se cubren muchos de los ataques más comunes. Es una forma realista y nada abrumadora de añadir una capa de seguridad a redes pequeñas.

Bibliografía

- [1] Cisco Systems. *Snort 3.1 Installation Guide (Linux)*. Disponible en: <https://docs.snort.org/>.
- [2] Cisco Talos. *Snort 3.1.84.0 Release Notes*, 2023. Disponible en: <https://github.com/snort3/snort3/releases>.
- [3] Lippmann, R., Fried, D., Ingols, K. *Analysis of Snort 3 Preprocessors and Performance*. MITRE, 2020.
- [4] Raspberry Pi Foundation. *Raspberry Pi 5 Technical Specifications*. Disponible en: <https://www.raspberrypi.com/products/raspberry-pi-5/>.
- [5] Usama, M. *Performance Evaluation of Intrusion Detection Systems on Low-End Hardware*. International Journal of Computer Science and Information Security, vol. 19, nº 3, 2021.
- [6] J. Humble, D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, 2010.
- [7] M. Roesch, “Snort: Lightweight intrusion detection for networks,” in *Proceedings of the 13th USENIX Conference on System Administration*, 1999.
- [8] S. Axelsson, “Intrusion detection systems: A survey and taxonomy,” Technical report, Department of Computer Engineering, Chalmers University of Technology, 2000.
- [9] Cisco Talos. *ClamAV Documentation*. Disponible en: <https://docs.clamav.net/>.

A. Anexo A

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

```

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

```

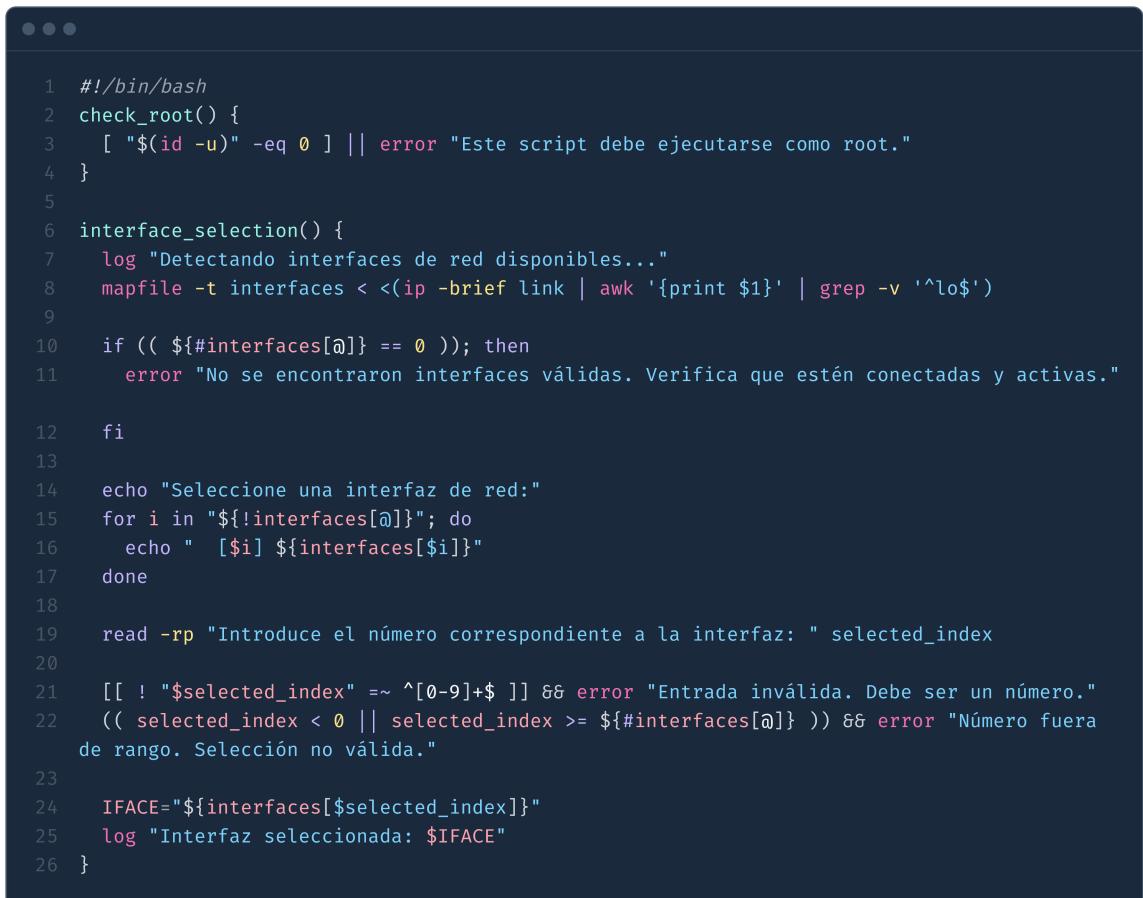
Figura B.1: Script principal r-snort_installer.sh



```

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 (( ${#interfaces[@]} == 0 )); then
11         error "No se encontraron interfaces válidas. Verifica que estén conectadas y activas."
12     fi
13
14     echo "Seleccione una interfaz de red:"
15     for i in "${!interfaces[@]}"; do
16         echo "  [$i] ${interfaces[$i]}"
17     done
18
19     read -rp "Introduce el número correspondiente a la interfaz: " selected_index
20
21     [[ ! "$selected_index" =~ ^[0-9]+$ ]] && error "Entrada inválida. Debe ser un número."
22     (( selected_index < 0 || selected_index >= ${#interfaces[@]} )) && error "Número fuera
de rango. Selección no válida."
23
24     IFACE="${interfaces[$selected_index]}"
25     log "Interfaz seleccionada: $IFACE"
26 }

```

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 -q '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         log "Instalando: ${basename "$archivo")"
25         tar -xf "$archivo"
26         dir=$(find . -mindepth 1 -maxdepth 1 -type d | grep -v '^./.' | head -n 1)
27
28         if [[ -z "$dir" || ! -d "$dir" ]]; then
29             error "No se encontró un directorio válido tras descomprimir $archivo"
30         fi
31
32         log "Entrando en directorio: $dir"
33         cd "$dir"
34
35         case "$archivo" in
36             *luajit*)
37                 make -j${(nproc)}
38                 make install PREFIX=/usr
39                 ;;
40
41             *openssl*)
42                 local target
43                 if uname -m | grep -q aarch64; then
44                     target="linux-aarch64"
45                 else
46                     target="linux-generic32"
47                 fi
48                 ./Configure --prefix=/usr --openssldir=/etc/ssl "$target"
49                 make -j${(nproc)}
50                 make install
51                 ;;
52
53             *daq*)
54                 log "Instalando DAQ con precauciones (desactivando 'set -e' temporalmente)"
55                 set +e
56                 if [[ -f "bootstrap" ]]; then
57                     chmod +x bootstrap
58                     ./bootstrap
59                     bootstrap_status=$?
60                 else
61                     bootstrap_status=1
62                 fi
63                 if [[ $bootstrap_status -ne 0 && -f "configure.ac" && ! -f "configure" ]]; then
64                     autoreconf -f
65                 fi
66                 set -e
67                 ./configure --prefix=/usr --enable-shared
68                 make -j${(nproc)}
69                 make install || error "Fallo al instalar DAQ"
70                 ;;
71
72             *)
73                 if [[ -f "configure.ac" && ! -f "configure" ]]; then
74                     [[ -f "bootstrap" ]] && chmod +x bootstrap && ./bootstrap || autoreconf -f
75                 fi
76                 if [[ -f "configure" ]]; then
77                     ./configure --prefix=/usr --enable-shared
78                 else
79                     cmake . -DCMAKE_INSTALL_PREFIX=/usr
80                 fi
81                 make -j${(nproc)}
82                 make install || error "Fallo al instalar ${basename "$archivo)"
83                 ;;
84             esac
85         esac
86
87         cd ..
88         rm -rf "$dir"
89         success "${(basename "$archivo") instalado}"
90     }
91
92     software_package_install() {
93         cd "$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" "$INSTALL_DIR/etc/snort/snort3-community-
rules"
12    touch "$INSTALL_DIR/etc/snort/reputation/interface.info"
13    tar -xzf "$CONFIG_DIR/snort3-community-rules.tar.gz" -C "$INSTALL_DIR/etc/snort/snort3-
community-rules" --strip-components=1
14
15    if [ -f /etc/systemd/system/snort.service ]; then
16        cp /etc/systemd/system/snort.service /etc/systemd/system/snort.service.bak
17        log "Backup del servicio original guardado."
18    fi
19
20    cat > /etc/systemd/system/snort.service <<EOF
21 [Unit]
22 Description=Snort NIDS Daemon
23 After=network.target
24
25 [Service]
26 ExecStart=$INSTALL_DIR/bin/snort -c $INSTALL_DIR/etc/snort/snort.lua -i $IFACE -A
alert_fast
27 ExecReload=/bin/kill -HUP \${MAINPID}
28 Restart=always
29 User=root
30 Group=root
31 LimitCORE=infinity
32 LimitNOFILE=65536
33 LimitNPROC=65536
34 PIDFile=/var/run/snort.pid
35
36 [Install]
37 WantedBy=multi-user.target
38 EOF
39
40    systemctl daemon-reexec
41    systemctl daemon-reload
42    systemctl enable snort.service
43    systemctl restart snort.service || error "No se pudo iniciar Snort."
44    sleep 2
45    systemctl status snort.service --no-pager
46    success "Servicio Snort configurado y activo."
47 }

```

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
21    echo "_____"
22    echo -e "█ Hostname:      $(hostname)"
23    echo -e "⌚ Uptime:        $uptime_str"
24    echo -e "MemoryWarning:   $used_ram / $total_ram"
25    echo -e "██ Swap activa:  $([ "$swap_enabled" -eq 0 ] && echo "No" || echo "Sí
($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 "_____"
31
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
 8      corrupto."
 9      exit 1
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-
9    daemon
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': ' '/^([0-9]+: 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 -y
47 }
48 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.