eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado en Ingeniería Informática

▪ Trabajo de Fin de Grado ▪

Ingeniería de Computadores

EVALUACIÓN DE SISTEMAS DE DETECCIÓN DE AMENAZAS: SNORT Y SURICATA

Ainhoa Veramendi

2021

Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado en Ingeniería Informática

▪ Trabajo de Fin de Grado ▪

Ingeniería de Computadores

EVALUACIÓN DE SISTEMAS DE DETECCIÓN DE AMENAZAS: SNORT Y SURICATA

Ainhoa Veramendi

2021

Dirección

José Miguel-Alonso

Resumen

En las últimas dos décadas Internet ha evolucionado a una gran velocidad y hoy en día está muy presente en nuestra sociedad. Es común el uso de Internet con el fin de realizar compras, transacciones, ocio, intercambio de información, entre otros. Por lo tanto, el alcance y posibilidades a la hora de realizar ciberataques se han incrementado significativamente.

Esta realidad ha hecho necesaria la implantación de diferentes medidas de seguridad, entre las que se encuentran los sistemas de detección de intrusiones. Estos sistemas se especializan en detectar posibles ataques o intrusiones en la red o equipo con el fin de poder tomar medidas que limiten o impidan los efectos maliciosos de dichos ataques.

El objetivo de este proyecto es analizar la eficacia de dos sistemas de intrusiones open-source bien conocidos: Snort y Suricata. Mediante una amplia variedad de ataques simulados se pondrá a prueba la habilidad de ambos programas para detectar diferentes ataques e intrusiones en la red.

Azken bi hamarkadetan Internet bilakaera handia izan du eta gaur egun gure gizartean eragin asko dauka. Ohikoa da Internet erabiltzea erosketak, transakzioak, informazio trukea egiteko, aisialdirako, besteak beste. Hori dela eta, zibererasoen garrantzia eta hauek burutzeko aukerak nabarmen handitu dira.

Ondorioz, segurtasun neurriak ezartzeko beharra sortu da. Horien artean, intrusio detekzio sistemak daude. Hauek sarea edo ordenagailua erasotzeko edota intrusio saiakerak detektatzeko gai dira. Horrez gain, hauek saihesteko neurriak hartu ditzakete.

Proiektu honen helburua bi intrusio detekzio sistema ezagunen eraginkortasuna ebaluatzea da: Snort eta Suricata. Simulatutako eraso ugarien bidez, bi programek sarean eraso eta intrusio desberdinak hautemateko duten gaitasuna probatuko da.

The last two decades the internet has evolved at great speed and it is very present in society. It is common to use the internet in order to make a purchase, transaction, leisure, exchange of information, among others. Therefore, the significance and possibilities when carrying out cyberattacks have increased significantly.

Different security measures have been developed in order to counteract these attacks, including Intrusion Detection Systems. These systems are able to detect possible attacks or intrusions in the network or device and take measures that limit or prevent the effects of the attacks.

The scope of this project is to analyze the effectiveness of two well-known open-source Intrusion Detection Systems: Snort and Suricata. Through a wide variety of simulated attacks, the ability of both programs to detect different attacks and intrusions on the network will be tested.

Índice de contenidos

Lista de Figuras y Tablas

FIGURAS

TABLAS

1 Introducción y objetivos

Los primeros intentos de fortalecer la seguridad de los procedimientos e inspeccionar datos se dieron sobre la década de los 60, cuando los sistemas financieros comenzaron a ser auditados con el fin de detectar fraudes y errores. El primer concepto de Sistema de Detección de Intrusiones fue creado en 1980 por James Anderson en la Agencia de Seguridad Nacional (NSA) de EE.UU., pero hasta 1986 no se publicaría el primer modelo llamado *Intrusion Detection Expert System* (IDES), desarrollado por Dorothy E. Denning y Peter G. Neumann.[17]

1.1 Introducción

1.1.1 Definición de IDS

Un Sistema de Detección de Intrusiones o IDS (siglas de *Intrusion Detection System*) es una aplicación que se encarga de descubrir intentos de ataques a un ordenador o a una red. Para ello, monitorizan diferentes procesos y archivos del sistema, así como el tráfico de la red. Esta información se coteja con una base de datos con firmas de ataques.

Se considera intrusión o ataque cualquier intento de (1) acceso no autorizado, (2) ganar privilegios, (3) comprometer la integridad o confidencialidad del sistema, (4) explotar vulnerabilidades o (5) interrupción de servicios.

Los IDS se limitan a detectar las intrusiones y alertar al usuario, por lo que no evitan que ocurran ataques. Sin embargo, los Sistemas de Prevención de Intrusiones o IPS (siglas de *Intrusion Prevention System*) no solo detectan intrusiones, sino que tienen la capacidad de tomar diferentes medidas como respuesta, entre ellas bloquear conexiones y reconfigurar el cortafuegos. Hoy en día son comunes los Sistemas de Detección y Prevención de Intrusiones o IDPS (siglas de *Intrusion Detection and Prevention System*), los cuales trabajan tanto de IDS como de IPS. Las dos aplicaciones a evaluar en este TFG, Snort [13] y Suricata [14], son ejemplos de IDPS distribuidos como software libre.

La mayoría de IDS basan su capacidad de detección en una base de datos de firmas con la que poder comparar el tráfico de la red con posibles ataques conocidos. Gracias a ello pueden distinguir el tráfico fraudulento o malicioso del tráfico benigno. Por ello es muy importante mantener actualizados tanto el IDS como la base de datos. Por otro lado, aparte de analizar el tipo de tráfico los IDS también se revisan el contenido y comportamiento del tráfico para una mayor efectividad a la hora de detectar ataques.

1.1.2 Tipos de IDS

Hay dos tipos de IDS, Sistemas de Intrusiones basados en Host o HIDS (siglas *de Host-based Intrusion Detection Systems*) y Sistemas de Intrusiones basados en Red o NIDS (siglas de *Network-based Intrusion Detection Systems*).

Sistemas de Detección de Intrusiones basados en Host

Los HIDS se instalan en un equipo con el objetivo de monitorizar eventos, procesos en ejecución, integridad de archivos y tráfico de la red para el dispositivo. A diferencia de los NIDS, los HIDS se encargan de monitorizar tan solo el equipo en el que se han instalado, por lo que hay que instalar un IDS en cada equipo monitorizado.

Los HIDS operan creando una base de datos de los ficheros del sistema, pudiendo así detectar cualquier modificación en los archivos protegidos que afecten a, por ejemplo, permisos, tamaño de archivo o fecha de modificación. Así mismo, tienen acceso a los eventos del sistema y a información sobre qué procesos o usuarios acceden a los recursos. De esta manera, son capaces de monitorizar en detalle los eventos del sistema y a su vez identificar al intruso. Por otro lado, monitorizan el tráfico entrante y saliente del equipo, no de la red, en busca de accesos no autorizados o ataques, así como, conexiones HTTP ilícitas o ataques de denegación de servicio o DDoS (siglas de *Distributed Denial of Service*).

Una ventaja adicional de los HIDS es que pueden trabajar en entornos de tráfico de red cifrado, ya que estos analizan los paquetes antes de cifrarse en origen o después de descifrarse en destino.

Los HIDS tienen algunas desventajas: (1) consumen recursos del propio sistema que están monitorizando; (2) un ataque con éxito a un equipo puede desactivar el HIDS, o evitar que genere alertas; (3) por definición, su ámbito de actuación se limita a un único equipo, no a toda la red.

Sistemas de Detección de Intrusiones basados en Red

Los NIDS detectan ataques y accesos no autorizados a la red. Para ello, monitorizan el tráfico gracias a sensores instalados en uno o varios segmentos de la red. Es de suma importancia situarlos bien para poder proteger los puntos críticos de la infraestructura de comunicación. El NIDS puede estar formado por una combinación de hardware y software, o ser únicamente software.

- Hardware especializado y software: se colocan uno o más sensores en uno o varios segmentos de la red, uno o más servidores de administración del IDS y consolas (ver figura 1.1).

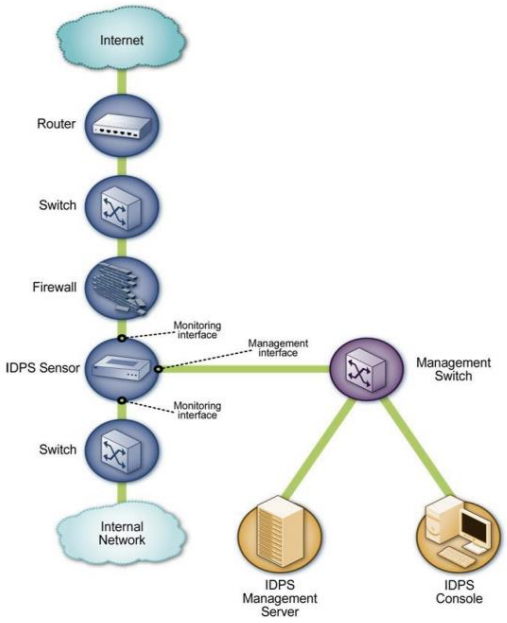

Figura 1.1. Esquema representativo de un IDPS inline basado en red. Figura tomada de [1].

- Solo software: se instala el NIDS en un host y se monitoriza el tráfico que pasa por sus interfaces. No es aconsejable en redes conmutadas, ya que no todo el tráfico de la red pasa por un mismo segmento. Por lo tanto, esta configuración se usa generalmente en entornos domésticos o pequeñas redes.

La posición del IDS se puede configurar de esta manera: Inline y Pasivo.

- Inline: el sensor se coloca para que el tráfico de la red pase a través de él. De esta manera, se podrá usar el IDPS para bloquear tráfico de la red. Esto requiere de un equipo con la

potencia necesaria para procesar el tráfico y no producir retardos significativos (ver figura 1.1).

- Pasivo: en este caso no pasa tráfico real sino una copia del mismo. Para ello se utilizan diferentes características de los switches de red, por ejemplo, el "SPAN" o "*Port Mirroring*". Con este método el switch envía todos los paquetes de un puerto o una VLAN entera a otro puerto, donde se ubicará el sensor.

Métodos de detección

Hay dos métodos básicos de detección de ataques: el basado en conocimiento (firmas) y el basado en comportamiento (anomalías). Hoy en día, los IDS utilizan principalmente el método basado en firmas, aunque también existen IDS que combinan ambos métodos.

- Los IDS basados en firmas analizan el tráfico en busca de patrones de ataques ya conocidos. Estos patrones o secuencias de eventos se comparan con la base de conocimiento en la cual se hallan las firmas de ataques conocidos. En caso de coincidencia, se envía al usuario una alerta que incluye el tipo de ataque detectado.

- Los IDS basados en detección de anomalías crean un modelo del uso benigno o normal de la red, contra el que se comparará el tráfico. Se considerará peligroso aquel tráfico que difiera del habitual, y se enviará la correspondiente alerta. De esta manera es posible detectar no solo ataques ya conocidos, sino también ataques de "día cero" (vistos por primera vez), ya que la detección no depende de una base de conocimiento de ataques.

Generalmente, el segundo método produce muchos falsos positivos debido a comportamientos impredecibles tanto de usuarios como de equipos y redes.

1.1.3 Ejemplos de IDS

Recopilamos en la siguiente tabla algunos de los IDS más utilizados (ver tabla 1.1):

Nombre del IDS	Tipo de IDS	¿Open source?	Comentarios
Suricata	NIDS/NIPS	Si	Compatible con el formato de archivos y reglas de Snort. Trabaja tanto con firmas como con detección de anomalías. Tiene "*multi-threading*", aceleración con GPU y varios modelos estadísticos de detección de anomalías.
Snort	NIDS/NIPS	Si	Es de los IDS más conocidos y actualmente es mantenido por Cisco. Tiene tres modos de uso: "*sniffer*" de paquetes, "*logger*" de paquetes y modo Detección de Intrusiones en la Red. Trabaja tanto con firmas como con detección de anomalías. Posee reglas mantenidas por la comunidad y también un sistema de pago por suscripción que actualiza constantemente las reglas.
Bro/Zeek	NIDS	Si	Funciona como "*logger*" y analizador de paquetes. Bro tiene soporte para analizar paquetes intercambiados mediante varios protocolos de capa de aplicación, por

			ejemplo, DNS, FTP, HTTP y SSL. Trabaja tanto con firmas como con detección de anomalías.
Fortinet	IPS	No	Incluye múltiples servicios incluyendo hardware, IPS en la nube y en máquinas virtuales.
Palo Alto	IPS	No	Desarrollado para mejorar la efectividad de los cortafuegos de la misma compañía. La base de conocimiento sobre ataques es actualizada a diario. Es capaz de prevenir ataques independientemente del puerto, protocolo o encriptación SSL. Dispone de sensores físicos.
Alien Vault OSSIM	NIDS	Si	Alien Vault dispone de Open Threat Exchange, una Plataforma en la que los usuarios contribuyen y reciben información en tiempo real sobre métodos de ataque, amenazas y host maliciosos. Está en continuo desarrollo por AT&T.
OSSEC	HIDS	Si	Detección de Intrusiones en hosts a base de logs y monitorización de la integridad de archivos. Almacena información sobre el sistema, software instalado, hardware, servicios de red, utilización, eventos y más.

Tabla 1.1. Tabla de los IDS/IPS más utilizados.

1.2 Objetivos del proyecto

El objetivo de este TFG es el estudio de los Sistemas de Detección de Intrusiones Snort y Suricata, incluyendo un análisis de su efectividad en la detección de ataques que sirve como criterio de comparación entre ambas herramientas.

Snort y Suricata forman parte del conjunto de IDS más utilizados, en buena medida porque se distribuyen como software libre. Estas son las razones por las que se han escogido para este TFG. Una característica importante de ambos IDS es que no requieren de ningún sensor o hardware específico, lo que facilita el acceso y despliegue. Además, son compatibles en lo que se refiere a los formatos de las firmas de ataques y reglas, lo que facilita la comparación entre ellos.

El desarrollo de este TFG requiere del conocimiento del funcionamiento de ambos IDS, así como entender la sintaxis de los ficheros de firmas (conocidas como reglas), mediante las que Suricata y Snort detectan los ataques. Además, con el fin de evaluar la efectividad de los dos IDS, es necesario disponer del material suficiente para poner a prueba sus capacidades de detección. Para ello, se investigarán diferentes opciones, entre ellas, el uso de repositorios de archivos PCAP (capturas de paquetes) que incluyan tráfico malicioso mezclado con tráfico normal.

Una vez obtenidos los resultados del análisis, será necesario transformarlos para su tabulación y representación, lo que facilitará su interpretación. Se obtendrán diferentes estadísticas para poder determinar la efectividad de los dos IDS ante diferentes amenazas. La efectividad se medirá con distintas configuraciones, para determinar cuál es la más efectiva en función del uso y del tipo de tráfico y ataque.

A continuación, se enumerarán más detalladamente las tareas que han sido necesarias para llevar a cabo este proyecto:

1. Planificación del proyecto.
2. Estudio del funcionamiento de los IDS Snort y Suricata.
3. Estudio de diferentes herramientas y recursos para simular ataques.
4. Implementación de diferentes scripts para el procesamiento de los resultados del análisis.
5. Análisis y comparación de resultados.
6. Conclusiones y propuestas para la mejora de efectividad de estas herramientas.

1.3 Planificación y gestión

A continuación se describen la estructura de descomposición del proyecto, los paquetes de trabajo que lo componen, la duración estimada y el análisis de riesgos del proyecto.

1.3.1 Estructura de descomposición del trabajo

Una estructura de descomposición del trabajo es una herramienta que consiste en la descomposición jerárquica del trabajo. El propósito es organizar y definir el alcance total del proyecto. Los elementos finales que componen su forma jerárquica son los paquetes de trabajo. La estructura del TFG se ha planteado de esta manera (ver figura 1.2).

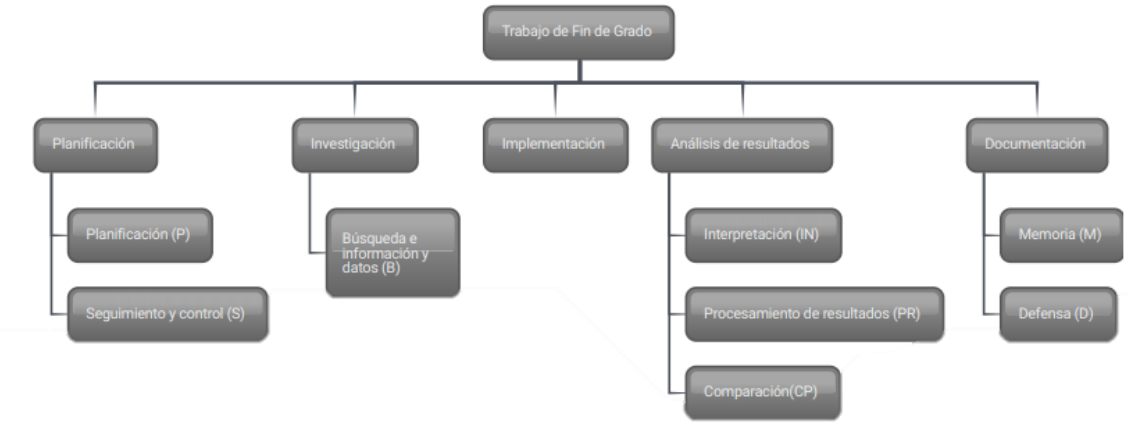

Figura 1.2. Estructura de descomposición del trabajo

1.3.2 Paquetes de trabajo

Un paquete de trabajo es una descripción de una operación que va a llevarse a cabo en un proyecto, por ejemplo, el trabajo que se ha de realizar y el resultado que se desea obtener en una tarea definida dentro del proyecto.

- Paquete de planificación y gestión:
    - **Planificación (P):** Se realizará la planificación del Trabajo de Fin de Grado. Este paquete incluye la planificación de la distribución de los paquetes del proyecto,

así como el tiempo estimado de duración de cada paquete, análisis de riesgos y preparación del entorno de experimentación.

  - **Seguimiento y control (S):** Reuniones con el profesor, posibles actualizaciones de los objetivos del proyecto y planificación.

- **Paquete de Investigación:**

  - **Búsqueda de información y datos (B)**

    - **B1. Sistemas de detección de intrusiones:** Se buscará información sobre los diferentes tipos de IDS y su funcionamiento.

    - **B2. Suricata:** Se adquirirán los conocimientos del uso y funcionamiento de Suricata.

    - **B3. Material para simulación de ataques:** Obtención y comprensión de diferentes trazas y datasets de tráfico malicioso.

    - **B4. Snort:** Se adquirirán los conocimientos del uso y funcionamiento de Snort.

    - **B5. Reglas:** Obtención de reglas e información sobre su funcionamiento en Suricata y Snort.

- **Paquete de Implementación:**

  - **Implementación (I):**

    - **I1. Instalación y puesta en marcha de Suricata:** Instalación en una máquina virtual y configuración de su funcionamiento.

    - **I1. Instalación y puesta en marcha de Snort:** Instalación en una máquina virtual y configuración de su funcionamiento.

    - **I3. Creación de código para procesar los resultados:** Escribir varios "scripts" que procesen los archivos producidos por los IDS con el fin de poder interpretar los resultados.

    - **I4. Simulación de ataques:** Procesar el tráfico malicioso en Snort y Suricata.

- **Paquete de Análisis de resultados:**

  - **Interpretación (IN):**

    - **IN1. Resultados de Suricata:** Comprensión e interpretación de los resultados obtenidos con Suricata.

    - **IN2. Resultados de Snort:** Comprensión e interpretación de los resultados obtenidos con Snort.

- **Procesamiento de resultados (PR):** Obtención de diferentes estadísticas y gráficos para el análisis de los resultados obtenidos.

- **Comparación (CP):** Comparación entre los resultados de Suricata y Snort, y conclusiones.

- **Paquete de Documentación:**

  - **Memoria (M):** Realización de la memoria final del proyecto.

  - **Defensa (D):** Preparación de la defensa del proyecto.

1.3.3 Tiempo de desarrollo de las tareas

Nombre del paquete	PT	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24
Planificación	P																								
Seguimiento y control	SC																								
Sis. detección intrusiones	B1																								
Suricata	B2																								
Mat. simulación de ataques	B3																								
Snort	B4																								
Reglas	B5																								
Instalación de Suricata	I1																								
Instalación de Snort	I2																								
Creación de código	I3																								
Simulación de ataques	I4																								
Resultados Suricata	IN1																								
Resultados Snort	IN2																								
Procesamiento de datos	PR																								
Comparación	CP																								
Memoria	M																								
Defensa	D																								

Figura 1.3. Tabla de tiempo de desarrollo de tareas

1.3.4 Análisis de riesgos

Se ha realizado un análisis de los posibles riesgos a la hora de llevar a cabo este proyecto. También se han buscado medidas de prevención y soluciones en caso de necesidad.

Riesgos y soluciones

- La simulación de ataques es de los componentes más importantes y necesarios a la hora de evaluar los sistemas de detección de intrusiones. Por lo tanto, en el caso de no encontrar un dataset válido para testear, habría que crear una red de máquinas virtuales en la que, mediante scripts se lanzarían ataques a un host.

- Nos encontramos con una constante aparición de nuevos ataques, por lo que las reglas de los sistemas de detección de intrusiones quedan obsoletas enseguida, y tienen por ello que ser actualizadas frecuentemente. Por este motivo, al comienzo del proyecto se escogerá la última versión de software y conjunto de reglas con las que se llevará acabo la evaluación.

- La probabilidad de pérdida de datos es baja, pero para mayor seguridad se hará una copia de seguridad de las máquinas virtuales y datos obtenidos en Google Drive.

- Es posible que no se haya estimado adecuadamente la dificultad del proyecto. De acuerdo con el profesor/tutor, se podrán añadir o eliminar tareas para ajustar el trabajo a realizar con el tiempo asignado al TFG.

1.3.5 Desviaciones

En este apartado se describen las desviaciones que ha habido respecto al plan inicial.

Trazas de ataques (datasets)

Al comienzo del proyecto, con el fin de evaluar los sistemas de detección de intrusiones, se iba a hacer uso del dataset de DARPA. "DARPA dataset" [4][15] creado en 1999, son trazas muy conocidas y utilizadas. Estas trazas solo contienen ataques, por lo que no es posible analizar posibles falsos positivos. Debido a su antigüedad, no representa al tráfico actual de manera adecuada, por lo que se decidió realizar una comparación de diferentes datasets disponibles con el fin de utilizar las trazas que mejor puedan evaluar la capacidad de los IDS. Tras la valoración de diferentes opciones, se ha escogido el dataset CIC-IDS2017 [2][3] del Canadian Institute of Cybersecurity. Estas trazas publicadas en 2017, cuentan nuevos tipos de ataques, que DARPA y otros antiguos datasets no contienen, además incluye tráfico benigno de manera que representa mejor el tráfico real de una red.

Interpretación de resultados

El dataset de CIC-IDS2017 dispone de documentación en la que se detalla el flujo de paquetes, a qué ataque pertenecen y la hora a la que ha ocurrido el ataque. Inicialmente, la evaluación se iba a realizar comparando la hora de los ataques del documento con la hora de las alertas creadas por los IDS, pero las horas que aparecen en los resultados no concuerdan con las horas que figuran en la documentación, por lo que por problemas de viabilidad se realizó un cambió en la forma de comparación. Este consiste en comprobar si los ataques que contienen las trazas aparecen en alguna de las alertas generadas por el IDS siempre y cuando las direcciones IP coincidan con las que figuran en la documentación del ataque.

Sistemas de detección de intrusiones

La idea original del proyecto consistía en evaluar en detalle únicamente el IDS Suricata. Durante el desarrollo del proyecto se añadió Snort a la evaluación, y se decidió realizar una comparación entre ambos IDS.

2 Sistemas de detección de intrusiones

A continuación se explica de forma más detallada el funcionamiento de los IDS analizados en este TFG. Además, se describe en profundidad cómo trabajan tanto Snort como Suricata.

2.1 Snort

Snort es un IDPS de código abierto creado en 1998 por Martin Roesch, fundador de Sourcefire. En 2013, Cisco compró Sourcefire y se hizo cargo del desarrollo de Snort.

2.1.1 Funcionamiento

Snort cuenta con tres modos de uso, (1) “sniffer”, (2) “packet logger” y (3) modo NIDS.

Sniffer

Con este modo, Snort se limita a imprimir en pantalla el flujo de paquetes (ver figura 2.1). Existen diferentes configuraciones mediante las que se modifican el número de datos a mostrar en pantalla. Un ejemplo de una ejecución de Snort en modo sniffer sería la siguiente (ver figura 2.1):

```
./snort -vd
```

```
snortultimate@ubuntu:~$ sudo snort -vd
[sudo] password for snortultimate:
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
pcap DAQ configured to passive.
Acquiring network traffic from "ens33".
Decoding Ethernet

        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.17 GRE (Build 199)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2020 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.8.1
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.2.11

Commencing packet processing (pid=5998)
WARNING: No preprocessors configured for policy 0.
05/05-09:30:13.262680 192.168.32.134:51124 -> 192.168.32.2:53
UDP TTL:64 TOS:0x0 ID:57432 IpLen:20 DgmLen:86 DF
Len: 58
E3 AF 01 00 00 01 00 00 00 00 00 01 12 63 6F 6E  .............con
6E 65 63 74 69 76 69 74 79 2D 63 68 65 63 6B 06  nectivity-check.
75 62 75 6E 74 75 03 63 6F 6D 00 00 01 00 01 00  ubuntu.com......
00 29 02 00 00 00 00 00 00 00                    .)........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

05/05-09:30:13.262816 192.168.32.134:40038 -> 192.168.32.2:53
UDP TTL:64 TOS:0x0 ID:57433 IpLen:20 DgmLen:86 DF
Len: 58
23 D3 01 00 00 01 00 00 00 00 00 01 12 63 6F 6E  #............con
6E 65 63 74 69 76 69 74 79 2D 63 68 65 63 6B 06  nectivity-check.
75 62 75 6E 74 75 03 63 6F 6D 00 00 1C 00 01 00  ubuntu.com......
00 29 02 00 00 00 00 00 00 00                    .)........
```

Figura 2.1. Consola con una ejecución de Snort en modo sniffer.

Con este comando se muestran los datos del paquete, la dirección IP y el encabezado TCP/UDP/ICMP.

Packet logger

En este modo, Snort guarda el flujo de paquetes entrante en el directorio que se le indique. Un ejemplo de puesta en marcha de Snort en modo logger sería el siguiente:

```
./snort -dev -l ./log -h 192.168.1.0/24
```

Con este comando, Snort guarda los paquetes relativos a la red seleccionada (en este caso *192.168.1.0/24*). Dentro del directorio */log* crea subdirectorios con el nombre de la dirección IP de origen, en los cuales se guardan los datos y encabezados TCP/IP de los paquetes.

NIDS

Este modo, Snort realiza una comparación de cada paquete con el conjunto de reglas. En el caso de que se encuentre una coincidencia entre un paquete y una regla, toma la acción establecida para dicha regla (por ejemplo, alertar, ignorar o descartar). Un ejemplo de ejecución en este modo sería:

```
./snort -d -h 192.168.1.0/24 -l ./log -c snort.conf
```

En el archivo *snort.conf* se configura qué reglas se usan, así como las acciones a tomar en el caso de que se activen. Si no se especifica un directorio de salida, los archivos de resultados se guardan en */var/log/snort*.

Funcionamiento de Snort como NIDS

Los componentes más importantes de Snort como NIDS son los preprocesadores (*preprocessors*) y el motor de detección (*detection engine*). Para una mayor flexibilidad, los preprocesadores se añaden a Snort de forma modular a modo de "*plug-in*", de manera que se pueden añadir o quitar en función del análisis que se desee realizar.

También, se pueden añadir "plug-ins" de detección, los cuales comprueban datos (definidos por una regla) en cada paquete en busca de coincidencias. De esta manera, se va analizando el paquete haciendo comprobaciones con diferentes "plug-ins".

Preprocesadores

Un preprocesador procesa los datos de los paquetes antes de que lleguen al motor de detección. De esta manera, Snort puede detectar posibles ataques que no se ajusten al modelo de patrones. Por ejemplo, ataques fragmentados en los que el ataque se divide en diferentes paquetes de un flujo TCP. Los diferentes preprocesadores disponibles pueden añadirse de forma modular.

Estos son algunos de los preprocesadores disponibles:

1. Reensamblado de flujo de datos (en inglés *stream reassembly).* Ejemplos: stream4, frag3.

2. Seguimiento de flujo de paquetes (en inglés *flow tracking*). Ejemplo: *flow*.

3. Detección de anomalías. Aunque hay más, podemos citar el escaneo de puertos. Ejemplo: *sfPortscan*.

Motor de detección

El motor de detección tiene dos funciones principales, el análisis de reglas y detección en base a firmas. El motor de detección crea firmas de ataques mediante el análisis de las reglas de Snort.

Al iniciar Snort en modo IDS, se cargan las reglas del conjunto seleccionado, una a una, en una estructura de datos interna. Además, es posible especificar la prioridad de cada regla. Nótese que, al cargar las reglas al inicio, un cambio en las reglas a aplicar solo será efectivo cuando se reinicie Snort.

Resultados

Las opciones de guardar los resultados de los datos procesados por Snort son las siguientes: (1) archivo *log* en texto o binario, (2) bases de datos SQL y (3) enviar los datos por la red mediante SNMP o WinPopup.

2.1.2 Otras funciones

Procesamiento de eventos (*event processing*)

Snort permite modificar la manera en la que se procesan los eventos. Estos son unos ejemplos:

- Filtrado de eventos (*event filtering*). Permite configurar el número de veces que un evento puede ser registrado en un intervalo de tiempo. De esta manera se evita que uno o varios eventos, activados repetidas veces en un periodo de tiempo concreto, inunden el registro de eventos.

- Supresión de eventos (*event suppression*). Permite suprimir los registros generados por un evento sin tener que modificar o eliminar la regla. Es posible suprimir eventos para direcciones IP o SIDs específicos (*Signature ID o id de la firma*), y de esta manera, evitar por ejemplo que se active la regla si el tráfico va o proviene de cierta dirección IP o grupo de direcciones.

- Filtrado de tasas (*rate filtering*). Permite configurar una nueva acción a tomar si, en un intervalo de tiempo especificado, se excede cierto número de eventos. Por ejemplo, se puede hacer una configuración para que, si hay más de 100 intentos de conexión en menos de un segundo procedentes de una misma dirección IP, se bloqueen futuros intentos de conexión desde esa dirección.

Procesamiento de archivos PCAP

En Snort es posible procesar archivos PCAP (capturas de paquetes) como si fuese tráfico capturado desde una interfaz de red. Esta es una manera útil y cómoda de testar y analizar el funcionamiento de Snort sin necesidad de ponerlo en marcha en una red real. Después de procesar el archivo PCAP se crean dos archivos: “alert” contiene las alertas que han sido activadas, y “snort.log” contiene los paquetes que han activado las reglas.

2.2 Suricata

Suricata es un IDPS de código abierto desarrollado por OISF (siglas de Open Information Security Fundation), lanzado en 2010. Una de las características clave de Suricata es que es multihilo, por lo que, puede ejecutarse en paralelo y así aprovechar la capacidad de multiprocesamiento que disponen los procesadores actuales.

2.2.1 Funcionamiento

Suricata dispone de diferentes opciones de línea de comandos, estas son algunas de ellas:

- -c <ruta>

Se especifica la ruta del archivo de configuración.

- -r <ruta del archivo PCAP>

Suricata se ejecuta en modo offline. Procesa el archivo PCAP y registra las alertas de las reglas activadas.

- -i <interfaz>

Analiza el tráfico proveniente de la interfaz especificada.

- -runmode <modo de ejecución>

Se especifica el modo de ejecución deseado. Suricata se compone de varios hilos, módulos de hilos y colas. Un módulo ejecuta la descodificación de los paquetes, otro se encarga de la detección y el último de los archivos de resultado. Un paquete se procesa y se pasa al siguiente hilo. En cada momento solo puede haber un hilo procesando un paquete, pero puede haber varios hilos simultáneos trabajando con diferentes paquetes.

Modos de ejecución de Suricata

Suricata puede operar en tres modos de ejecución diferentes: (1) *single*, (2) *workers* y (3) *autofp*.

- Modo "*workers*". La carga se balancea entre todos los hilos de proceso de Suricata. Cada hilo de proceso contiene el flujo entero del paquete (ver figura 2.2).

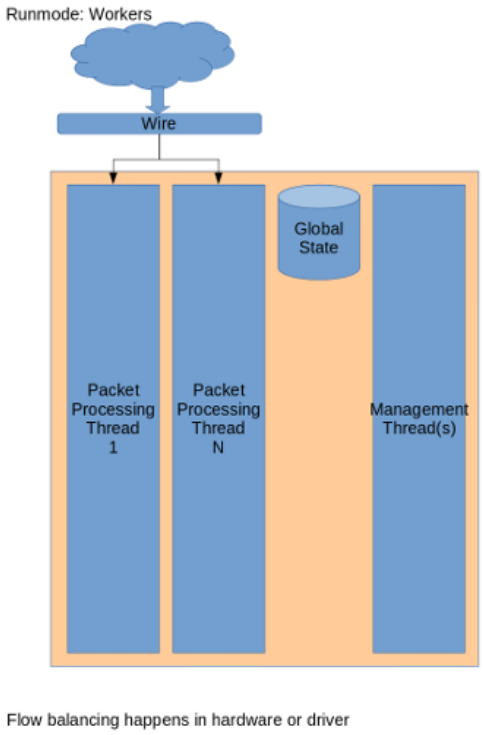

Figura 2.2. Modo de ejecución *workers* de Suricata

- Modo "*autofp*". Se usa para procesar los archivos PCAP o para ciertas configuraciones en modo IPS. A diferencia del modo "*workers*", hay uno o varios hilos de captura de paquetes que se encargan de capturar y decodificar los paquetes. Después, los paquetes pasan a los hilos de proceso de "*workers*" (ver figura 2.3).

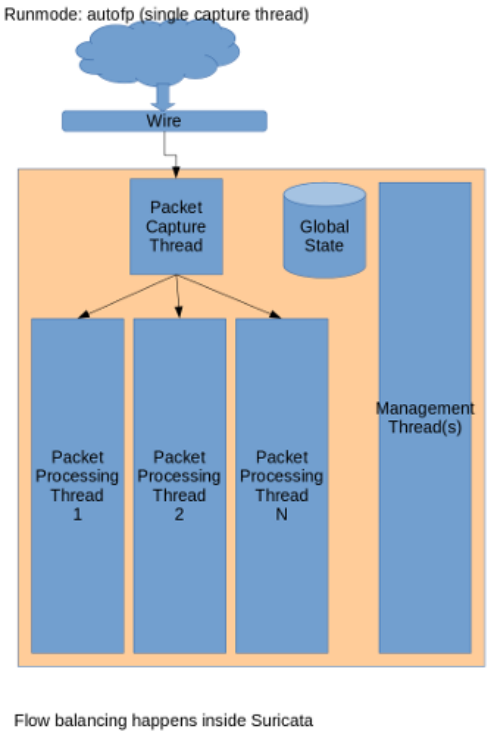

Figura 2.3. Modo de ejecución *autofp* de Suricata

- Modo **"*single*".** Funciona igual que el modo "*workers*" con la diferencia de que dispone de un solo hilo de proceso (ver figura 2.4).

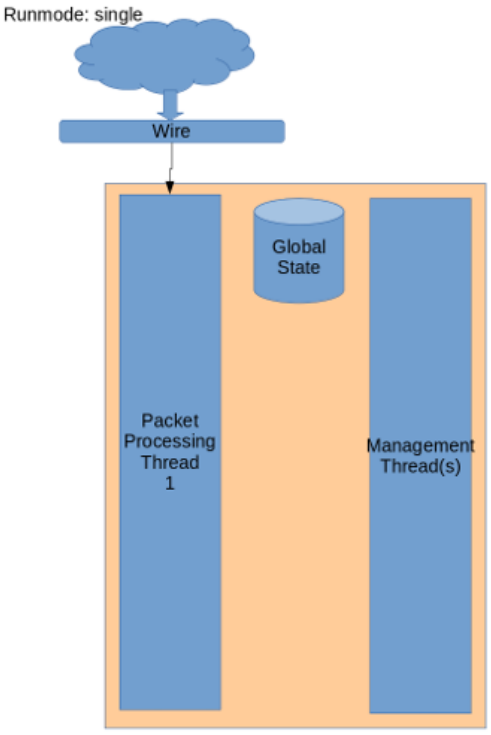

Figura 2.4. Modo de ejecución *single* de Suricata

2.2.2 Configuración de Suricata

La configuración de Suricata se realiza mediante el archivo *suricata.yaml*. Estas son algunas de las configuraciones disponibles:

- **Max-pending-packets.** Indica el máximo de paquetes que puede procesar Suricata al mismo tiempo. El rango de paquetes empieza desde uno hasta 100.000 o más. Cuanto mayor es el número, mejor uso se hace de los núcleos del procesador (el multiprocesamiento), por lo que mejorará el rendimiento. No obstante, el consumo de memoria aumentará.

```
max-pending-packets: 1024
```

- **Action-order.** Indica la prioridad de las reglas. La prioridad determina el orden en el que se procesaran las reglas.

```
action-order:
- pass
- drop
- reject
- alert
```

  De este modo, las reglas de *pass* se comprobarán antes que el resto, por lo que, en el caso hipotético de que un paquete coincida con una regla *pass* y también con otra regla *reject*, la segunda nunca llegaría a activarse porque la regla *pass* se comprobaría primero y se daría por finalizado el análisis de ese paquete.

- **Configuraciones del archivo EVE.** EVE (Extensible Event Format) es un archivo con formato JSON que contiene los resultados del procesamiento de paquetes. También incluye registros de anomalías, metadatos y alertas. Guardar resultados en archivos EVE facilita el uso de herramientas de terceros para su procesamiento y visualización, como por ejemplo ELK Stack [7].

- **Packet log (*pcap-log*).** Guarda en un archivo PCAP todos los paquetes registrados por Suricata. Los archivos PCAP pueden ser comprimidos antes de guardarse en el disco.

```
- pcap-log:
enabled:  yes
filename: log.pcap

# Limit in MB.
limit: 32
```

- **Syslog.** Se envían las alertas y eventos a *syslog*.

```
- syslog:
      enabled: no
      facility: local5
      level: Info #Indica el nivel de las
           alertas a enviar. Estos
      son los niveles posibles:
```

```
Emergency, Alert, Critical,
Error, Warning, Notice, Info
and Debug.
```

- **Drop.log.** Los paquetes que han sido bloqueados por acciónes *drop* se guardan en el archivo *drop.log*.

```
- drop:
      enabled: yes
      filename: drop.log
      append: yes
      filetype: regular
```

- **File-store (*File Extraction*).** Extrae los archivos intercambiados mediante protocolos HTTP. Para usar esta opción hay que hacer uso de las reglas:

```
alert http any any -> any any (msg:"FILE PDF file claimed";
fileext:"pdf"; filestore; sid:2; rev:1;)
```

A la hora de escribir la regla se puede activar el guardado del archivo mediante el uso de la palabra *filestore*. Si se desea guardar un tipo de archivo en concreto, se especifica mediante la palabra *fileext*. Esta función se puede activar y desactivar en el archivo *suricata.yaml* modificando el valor de *enabled*.

```
- file-store:
version: 2
enabled: no
```

2.3 Reglas

Tanto en Snort como en Suricata, la generación de alertas depende del conjunto de reglas utilizado para detectar intrusiones. Como los ataques evolucionan, apareciendo versiones de ataques anteriores, o ataques novedosos, es muy importante mantener actualizado y bien configurado este conjunto. Suricata y Snort usan la misma sintaxis para definir reglas. No obstante, a pesar de que comparten la sintaxis no tienen una compatibilidad total, puesto que hay ciertas expresiones y funciones exclusivas de cada uno. Por este motivo, Suricata y Snort disponen de reglas optimizadas para su uso en el correspondiente IDS.

2.3.1 Sintaxis de las reglas

Las reglas se componen de dos secciones funcionales: encabezado y opciones de regla.

Encabezado

El encabezado contiene los requerimientos para que se aplique la firma. Así como, (1) acción (acción a tomar en el caso de que se active la regla), (2) protocolo, (3) dirección IP destino/origen y (4) puerto destino/origen. El encabezado tiene la siguiente sintaxis:

```
Acción Protocolo (Dirección IP Origen) (Puerto origen) ->
(Dirección IP Destino) (Puerto destino)
```

Este sería un ejemplo de un encabezado:

```
alert udp $HOME_NET 80 -> 213.100.1.20 any
```

Acción: Estas son las acciones disponibles: (1) *alert* (genera una alerta), (2) *log* (guarda el paquete), (3) *pass* (ignora el paquete), (4) *drop* (bloquea el paquete), (5) *reject* (bloquea el paquete, lo guarda y envía de vuelta un mensaje *"TCP reset"* o "*ICMP port unreachable"* si es UDP) y (6) *sdrop* (bloquea el paquete, pero no lo guarda). Suricata no dispone de opción de guardado del paquete mediante la acción de las reglas, es necesario indicarlo en el modo de ejecución si se desea hacer uso de esta función.

Protocolo: Estos son los protocolos disponibles: TCP, UDP, ICMP e IP. En Suricata, además de los mencionados, están disponibles protocolos de la capa de aplicación (nivel 7 del modelo OSI), tales como, *http*, *ftp*, *tls*, *smb*, *dns*, *ssh* y más.

Direcciones IP: La flecha indica la dirección del tráfico. En el caso de que sea bidireccional, se indicaría con "<>". Existe la posibilidad de añadir una máscara de red añadiendo el símbolo "/" después de la dirección IP y a continuación el valor de máscara deseado. Es posible añadir una exclamación ("!") antes de una dirección para aplicar el efecto de negación. Así, esa o esas direcciones IP se excluyen. Si se desea indicar cualquier dirección IP posible se puede usar el valor *any*. Es posible definir variables uno o varios rangos de direcciones IP. Este es un ejemplo:

- En snort.conf: ipvar HOME_NET any

- En suricata.yaml: vars:
  address-groups:
  HOME_NET:"[192.168.0.0/16,10.0.0.0/8,172.16.0.0/12]"

Puerto: puerto TCP/UDP. Se puede especificar un rango de puertos mediante ":". También es posible utilizar "!" en forma de negación.

Opciones

Las opciones que componen la regla son estas: (1) Mensaje, (2) Flujo, (3) Contenido, (4) Referencia, (5) Clasificación e (6) Identificador de la regla.

```
(msg:"APP-DETECT Apple Messages push.apple.com DNS TXT request attempt"; flow:to_server; byte_test:1,!&,0xF8,2; content:"|04|push|05|apple|03|com|00 00 10 00 01|"; fast_pattern:only; metadata:policy max-detect-ips drop, service dns; reference:url,www.apple.com/osx/apps/all.html#messages; classtype:policy-violation; sid:25080; rev:2;)
```

Mensaje: Información sobre el ataque al que pertenece la alerta.

Flujo: Permite indicar si la regla se aplica a servidores o clientes.

Contenido: Es de las partes más importantes, ya que permite configurar la búsqueda de caracteres, valores en binario o patrones específicos en los datos de los paquetes. Cuando un paquete coincide con una regla, se procede a comprobar el resto de opciones de regla. En esta sección se puede especificar qué datos binarios, números hexadecimales o texto debe contener el paquete. Además, para describir en más detalle el patrón o contenido que active la regla,

existen diferentes parámetros, por ejemplo, (1) length (especifica la longitud del contenido), (2) distance (indica dónde empieza la parte del paquete a analizar), (3) http_header (las opciones de regla se restringen al encabezado de una solicitud HTTP o respuesta de servidor HTTP), entre otros.

Referencia: Fuentes externas de información sobre el ataque.

Clasificación: Indica en qué grupo se clasifica el ataque que intenta detectar la regla. Cada clasificación tiene una prioridad asignada, dependiendo de la gravedad. Las alertas que advierten de ataques de alto riesgo son asignadas una prioridad de 1, la más alta, mientras que a las alertas que simplemente proporcionan información son asignadas una prioridad de 4, la más baja.

Id de la regla: el *sid* es un número único que se utiliza para identificar la regla; *rev* indica el número de versión de la regla.

2.4 Diferencias entre Snort y Suricata

2.4.1 Reglas

Snort dispone de un grupo de expertos en ciberseguridad denominado Talos, que se especializa en descubrir nuevas amenazas y en dar respuesta a las vulnerabilidades y malware más actuales. De esta manera, Talos se encarga de mantener el conjunto de reglas de Snort actualizado ante nuevos ataques. Talos cuenta con una suscripción de pago por la que proporcionan las últimas actualizaciones en el conjunto de reglas. No obstante, existe una versión gratuita, para usuarios registrados, que contiene la mayoría de reglas, pero con 30 días de retraso respecto a la de pago. Además, en la propia página de Snort se puede descargar un conjunto de reglas de la comunidad, sin suscripción.

Las reglas de Suricata las actualiza Emerging Threats de Proofpoint, una compañía de ciberseguridad. La misma empresa también genera una versión del conjunto de reglas específico para Snort. También es posible usar el conjunto de reglas de Talos. No obstante, estas son creadas específicamente para Snort y es necesario adaptar algunos parámetros por no ser del todo compatibles con Suricata.

Ejemplos de diferencias en las reglas

- En Suricata la detección de protocolo es independiente del puerto, por lo que, es posible indicar que una regla se active con un protocolo en concreto, sin especificar en qué puertos. Sin embargo, en Snort es necesario especificar los puertos.

- En Snort se puede limitar el número de alertas por cada paquete analizado, mientras que en Suricata el límite es 15 y no puede ser modificado.

2.4.2 Otras diferencias

- Snort dispone un amplio soporte y documentación en la red, lo que facilita su uso y despliegue.

- Suricata permite el multiprocesamiento, por lo que puede aprovechar los procesadores multi-núcleo y multi-hilo. Sin embargo, Snort se ejecuta en un solo hilo. Debido a esto,

Suricata tiene ventaja respecto a Snort en el rendimiento. Además, Suricata dispone de aceleración por hardware (permite utilizar la tarjeta gráfica).

- Suricata trabaja también con la capa de aplicación (nivel 7 del modelo OSI), por lo que, puede inspeccionar los certificados TSL/SSL, solicitudes HTTP y más.

- Suricata permite el uso de LuaJIT. Lua es un lenguaje imperativo, multiparadigma y muy ligero. Gracias a su sencillez y ligereza ha sido utilizado para configurar aplicaciones, controlar hardware, procesar datos de entrada a sistemas complejos o en videojuegos. En el caso de Suricata permite el uso de "scripts" con los que poder combinar reglas y mejorar la eficacia de detección.

En conclusión, Suricata dispone de más funcionalidades, pero su documentación no es extensa. Por su parte, Snort dispone de mucha información respecto a su uso y una amplia comunidad de usuarios.

3 Entorno de experimentación

En este capítulo se detallan los experimentos a realizar, el entorno de trabajo, las trazas de ataques, la instalación de los IDS, los conjuntos de reglas utilizados, y los criterios y estructuras de evaluación.

3.1 Experimentos a realizar

Los experimentos principales son el análisis de Suricata, Snort 2.7.0 y Snort 2.9.17 usando las trazas de ataque CIC-IDS2017 del Canadian Institute of Cybersecurity. Después de procesar las trazas de ataque, se filtrarán y procesarán los archivos de resultados obtenidos, con los que se obtendrán diferentes estadísticas, como por ejemplo el número de verdaderos positivos y falsos positivos, con el fin de concluir la efectividad de cada IDS y realizar una comparación entre ellos. Por otro lado, con el fin de determinar la importancia del conjunto de reglas se realizará el mismo procedimiento de análisis con Snort 2.9.17 utilizando el conjunto de reglas utilizado en Snort 2.7.0 (snortrules-snapshot-2700). El conjunto de reglas utilizado en los experimentos principales se detallará más adelante.

3.2 Entorno de trabajo

El entorno de trabajo lo forman dos máquinas virtuales instaladas en el mismo ordenador. En una se ha instalado Snort y, en la otra Suricata.

Para el uso y creación de las máquinas virtuales se ha utilizado VMWare Workstation 15 y se ha instalado Ubuntu 18.04.3 como sistema operativo. Las características de las máquinas virtuales son las siguientes: 6 núcleos de CPU, 8GB de RAM y 50GB de capacidad de almacenamiento.

3.3 Trazas de ataques

Como ya se ha mencionado anteriormente, las trazas de ataques a usar como dataset en los experimentos son muy importantes. Por este motivo se ha decidido realizar una comparación entre diferentes datasets (ver tabla 3.1)[3][5].

Dataset	Desarrollado por	Tipos de ataques	Descripción
DARPA	MIT, Lincoln Laboratory	DoS, R2L, U2R y Probe [5]	No representa tráfico de red real, ausencia de instancias de tráfico benigno e irregularidades en instancias de datos de ataques.
KDD CUP 99	University of California	DoS, R2L, U2R y Probe [5]	Es una versión más refinada de DARPA. Contiene muestras redundantes y duplicadas.
NSL-KDD	University of California	DoS, R2L, U2R y Probe [5]	Es una versión más refinada de KDD CUP 99. Contiene de un número limitado de ataques.
CAIDA	Center of Applied Internet Data Analysis	DDoS	Las instancias que contienen son muy específicas de un tipo de ataque.
Kyoto	Kyoto University	Sesiones de ataques y tráfico normal	Se desarrolló colocando “honeypots” también conocidos como señuelos. No se dispone de detalles sobre los ataques que contiene.
ISCX2012	Canadian Institute of Cybersecurity	DoS, DDoS, Fuerza bruta e Infiltración	Dispone de diferentes escenarios con actividades intrusivas, y las instancias de datos están etiquetadas.
CIC-IDS2017	Canadian Institute of Cybersecurity	Fuerza bruta, escaneo de puertos, Botnet,	Se usan perfiles de red para generar el dataset de manera específica. Dispone de varios

		Dos, DDoS, Ataques web e Infiltración	tipos de ataques y tipos de infiltración.
CSE-CIC-IDS2018	Canadian Institute of Cybersecurity	Fuerza bruta, escaneo de puertos, Botnet, Dos, DDoS, Ataques web e Infiltración	Se usan perfiles de red para generar el dataset de manera específica. Dispone de varios tipos de ataques y tipos de infiltración.

Tabla 3.1. Tabla con diferentes dataset, sus tipos de ataques y descripción

Los dataset DARPA, KDD CUP 99 y NSL-KDD se han descartado ya que, al ser antiguos y disponer de pocos ataques, no representan de manera adecuada el tráfico actual [3][5]. CAIDA solo dispone de ataques DDoS por lo que no es útil a la hora de evaluar la efectividad de un IDS en su totalidad. Por otro lado, el dataset Kyoto no dispone de información detallada sobre las instancias de ataques que contiene, por lo que no es adecuado a la hora de realizar una evaluación.

Por último, entre los tres dataset desarrollados por el Canadian Institute of Cybersecurity (University of New Brunswick), se ha descartado ISCX2012 ya que dispone de menos tipos de ataques respecto a los otros dos y es menos actual. Además, hoy en día el protocolo HTTPS supone un 70% del tráfico y este no dispone de HTTPS [3]. Debido a esto, el Canadian Institute of Cybersecurity desarrolló CIC-IDS2017, el cual contiene 5 días de tráfico con 2.830.742 instancias [16]. Dispone de nuevos tipos de ataques actuales y, este sí, tráfico HTTPS. En 2018 Communications Security Establishment (CSE) se unió al proyecto y desarrollaron una mejora del dataset CIC-IDS2017, CSE-CIC-IDS2018. Este dataset contiene los mismos tipos de ataques y es similar a CIC-IDS2017, no obstante este dispone de una mayor red de clientes y atacantes, conteniendo 16.233.022 instancias distribuidas a lo largo de 10 días [5][16]. Teniendo esto en cuenta, se ha concluido que no es necesaria la gran cantidad de tráfico que proporciona el dataset CSE-CIC-IDS2018 y se considera que con el tráfico más reducido del dataset CIC-IDS2017 es posible evaluar de manera efectiva los IDS.

El dataset CIC-IDS2017 [3] contiene trazas de ataque que simulan un tráfico real, en el que figuran diferentes tipos de intentos de ataque a lo largo de 5 días. Además, abarcan los tipos de ataques más comunes [5][6], como son: ataques de fuerza bruta FTP, Ataques Web, DoS, DDoS, Heartbleed, infiltraciones, Botnet, escaneos de puerto y ataques de fuerza bruta SSH. El dataset incluye un fichero Excel en el que se aporta información sobre los flujos de paquetes, como por ejemplo el día de la semana, tipo de ataque o si es benigno, hora de inicio y finalización del ataque, dirección IP destino y origen, el número de paquetes o tamaño.

Las trazas de ataque se dividen en 5 archivos PCAP, cada uno de ellos correspondiente a un día: lunes, martes, miércoles, jueves y viernes (capturadas entre el 3 y el 7 de julio de 2017). A la hora de realizar la evaluación no se ha utilizado el archivo PCAP del lunes, puesto que este solo contiene tráfico benigno. Cada archivo contiene lo siguiente [3]:

Lunes (Monday-WorkingHours.pcap)

Tráfico benigno (Actividad humana normal)

Martes (Tuesday-WorkingHours.pcap)

- Ataques de fuerza bruta [8]
    - ftp-login [8]
    - ssh-login [8]

Miércoles (Wednesday -WorkingHours.pcap)

- Ataques DoS y DDoS
    - DoS Slowloris
    - DoS Slowhttptest [11]
    - DoS Hulk
    - DoS GoldenEye [12]
- Heartbleed (Puerto 444)

Jueves (Thursday-WorkingHours.pcap)

- Ataques Web
    - Fuerza bruta
    - XSS
    - Inyección SQL
- Infiltración
    - Meta exploit
    - Cooldisk
    - Descarga por Dropbox
    - Escaneo de puertos
    - Nmap

Viernes (Friday-WorkingHours.pcap)

- Botnet ARES
- Escaneo de puertos
- DDoS LOIC

3.4 Instalación de Snort y Suricata

La instalación de los IDS se ha realizado de la siguiente manera:

- Snort. Se ha descargado el archivo de instalación, version 2.9.17 de Snort. Snort 2.7.0 se ha descargado mediante un repositorio de paquetes de instalación para Ubuntu.
- Suricata. Se ha descargado mediante los repositorios la versión 5.0.3.

Las diferentes versiones se han obtenido de la siguiente manera:

- Snort 2.9.17: se ha descargado en la página web de Snort. [10]
- Snort 2.7.0: uso del comando *"apt-get install snort"*
- Suricata: uso del commando *"apt-get install suricata"*

Los repositorios utilizados son los actualizados en Ubuntu a fecha de octubre del 2020.

Durante la realización de este proyecto, tanto Snort como Suricata han sido actualizados varias veces. No obstante, durante el proyecto se decidió continuar con las versiones arriba indicadas, las cuales no presentan una diferencia significativa respecto a sus versiones más actualizadas.

3.5 Conjunto de reglas

Snort 2.9.17 ha sido utilizado con el conjunto de reglas para usuarios registrados (*snortrules-snapshot-29170*), el más actualizado a fecha de octubre del 2020 [10], mientras que Snort 2.7.0 ha sido utilizado con las reglas de la versión de Snort 2.7.0 (snortrules-snapshot-2700) que contenía en la descarga desde el repositorio y las reglas de la comunidad.

Para Suricata se ha utilizado el conjunto de reglas de Emerging Threats a fecha de octubre del 2020 [9].

3.6 Estructuras y criterios de evaluación

3.6.1 Estructura de evaluación

El primer paso es poner en marcha los dos IDS en las máquinas virtuales y descargar el dataset de CIC-IDS2017 y su respectiva documentación. Una vez hecho esto, se procesan los archivos PCAP tanto en Snort como en Suricata.

Snort

Para procesar un archivo PCAP en Snort, en este caso el archivo del viernes, se ha utilizado el siguiente comando:

```
sudo snort -l /home/ubuntu/Material/Snort -c /etc/snort/snort.conf -
A fast -r "/home/ubuntu/shares/Material/Friday-WorkingHours.pcap"
```

Una vez Snort termine creará dos archivos con los resultados: alert.txt y snort.log.XXXX.

- snort.log.XXXX (X siendo el número de identificación): contiene los paquetes en formato *tcpdump*.
- alert.txt: contiene todas las alertas generadas.

A la hora de realizar el análisis se va hacer uso del archivo alert.txt. El contenido de un archivo alert.txt es el siguiente (ver figura 3.1):

Figura 3.1. Extracto de alert.txt

Por un lado tenemos la fecha y hora extraída de los archivos PCAP y los números de identificación de la regla en ese orden ([**] se utiliza de separador):

```
07/07-08:00:28.268892  [**] [1:1201:7]
```

Mensaje, clasificación y prioridad asociados a las reglas:

ATTACK-RESPONSES 403 Forbidden [**] [Classification: Attempted Information Leak] [Priority: 2]

Y, por último, protocolo, direcciones IP y puertos, tanto origen como destino:

{TCP} 195.50.176.88:80 -> 192.168.10.5:50798

Con el fin de procesar con mayor facilidad los datos obtenidos, se ha escrito un script Python (ver figura 3.2) que extrae los datos del archivo "alert.txt" y forma un archivo con el formato *csv* (ver figura 3.3):

```
import pandas as pd
import re

header = ['Day','Timestamp','Alertmsg','Classification','Protocol','Priority','Source','Dest']
df = pd.DataFrame(columns = header)

with open('/home/ubuntu/Material/Snort/Thursday/thursday.log') as dat:
    lines = dat.readlines()
for line in lines:

    time = re.split('\.[0-9][0-9][0-9][0-9][0-9][0-9]',line)
    x = re.split('\[[0-9]:[0-9][0-9][0-9][0-9]+:[[0-9]+] ',time[1])
    alertmsg = re.split('\[\*\*]\s\[',x[1])
    classi = re.split('] \[Priority:',alertmsg[1])
    prio = re.split('] \{',classi[1])
    proto = re.split('}',prio[1])
    source = re.split('->\s',proto[1])
    dest = source[1].split()
    df2 = pd.DataFrame([['Thursday',time[0],alertmsg[0],classi[0],proto[0],prio[0],source[0],dest[0]]], columns = header)
    df = df.append(df2)

df.to_csv('/home/ubuntu/Material/Snort/Thursday.csv')
```

Figura 3.2. Código del script procesar.py

Al disponer de los datos en formato csv se puede hacer uso de filtros en programas de hojas de cálculo con los que buscar la información de manera más fácil. Si se aplica un filtro en la columna escogida se muestran todos los contenidos únicos disponibles en dicha columna. Los contenidos pueden desmarcarse y desaparecen, facilitando así la búsqueda y análisis. De esta manera, escogiendo diferentes columnas, es posible filtrar las alertas por direcciones IP, mensaje de alerta, protocolo o prioridad entre otros.

Uno de los usos sería el filtrado por direcciones IP; se filtran las alertas para que contengan las direcciones IP del atacante y víctima y se comparan las alertas que coincidan con los ataques.

Friday	07/07-11:21:27	ICMP PING NMAP	Classification: Attempted Information Leak	ICMP	2	172.16.0.1	192.168.10.50
Friday	07/07-11:21:27	ICMP PING	Classification: Misc activity	ICMP	3	172.16.0.1	192.168.10.50
Friday	07/07-11:21:27	ICMP Echo Reply	Classification: Misc activity	ICMP	3	192.168.10.50	172.16.0.1
Friday	07/07-11:21:27	ICMP Destination Unreachable Port Unreachable	Classification: Misc activity	ICMP	3	192.168.10.50	172.16.0.1
Friday	07/07-11:21:35	WEB-MISC PCT Client_Hello overflow attempt	Classification: Attempted Administrator Privilege Gain	TCP	1	192.168.10.12:33010	50.16.207.248:443
Friday	07/07-11:21:35	WEB-MISC weblogic/tomcat .jsp view source attempt	Classification: Web Application Attack	TCP	1	192.168.10.17:44441	199.59.88.242:80
Friday	07/07-11:21:35	INFO web bug 0x0 gif attempt	Classification: Misc activity	TCP	3	199.59.88.242:80	192.168.10.17:44441
Friday	07/07-11:21:35	WEB-MISC weblogic/tomcat .jsp view source attempt	Classification: Web Application Attack	TCP	1	192.168.10.17:44442	199.59.88.242:80
Friday	07/07-11:21:36	INFO web bug 0x0 gif attempt	Classification: Misc activity	TCP	3	199.59.88.242:80	192.168.10.17:44442
Friday	07/07-11:21:35	WEB-MISC weblogic/tomcat .jsp view source attempt	Classification: Web Application Attack	TCP	1	192.168.10.17:44444	199.59.88.242:80
Friday	07/07-11:21:35	WEB-MISC weblogic/tomcat .jsp view source attempt	Classification: Web Application Attack	TCP	1	192.168.10.17:44441	199.59.88.242:80
Friday	07/07-11:21:36	INFO web bug 0x0 gif attempt	Classification: Misc activity	TCP	3	199.59.88.242:80	192.168.10.17:44441
Friday	07/07-11:21:35	WEB-MISC weblogic/tomcat .jsp view source attempt	Classification: Web Application Attack	TCP	1	192.168.10.17:44445	199.59.88.242:80
Friday	07/07-11:21:36	INFO web bug 0x0 gif attempt	Classification: Misc activity	TCP	3	199.59.88.242:80	192.168.10.17:44444
Friday	07/07-11:21:35	WEB-MISC weblogic/tomcat .jsp view source attempt	Classification: Web Application Attack	TCP	1	192.168.10.17:44447	199.59.88.242:80
Friday	07/07-11:21:36	INFO web bug 0x0 gif attempt	Classification: Misc activity	TCP	3	199.59.88.242:80	192.168.10.17:44445
Friday	07/07-11:21:36	INFO web bug 0x0 gif attempt	Classification: Misc activity	TCP	3	199.59.88.242:80	192.168.10.17:44447
Friday	07/07-11:21:36	WEB-MISC weblogic/tomcat .jsp view source attempt	Classification: Web Application Attack	TCP	1	192.168.10.17:44449	199.59.88.242:80
Friday	07/07-11:21:36	INFO web bug 0x0 gif attempt	Classification: Misc activity	TCP	3	199.59.88.242:80	192.168.10.17:44449
Friday	07/07-11:22:01	INFO web bug 0x0 gif attempt	Classification: Misc activity	TCP	3	138.108.7.20:80	192.168.10.5:54664
Friday	07/07-11:22:01	INFO web bug 0x0 gif attempt	Classification: Misc activity	TCP	3	64.30.224.172:80	192.168.10.5:54615
Friday	07/07-11:22:01	INFO web bug 0x0 gif attempt	Classification: Misc activity	TCP	3	64.30.224.172:80	192.168.10.5:54615
Friday	07/07-11:22:01	INFO web bug 0x0 gif attempt	Classification: Misc activity	TCP	3	173.241.242.220:80	192.168.10.5:54670
Friday	07/07-11:22:01	INFO web bug 0x0 gif attempt	Classification: Misc activity	TCP	3	173.241.242.220:80	192.168.10.5:54668
Friday	07/07-11:22:02	SNMP request tcp	Classification: Attempted Information Leak	TCP	2	172.16.0.1:35609	192.168.10.50:161
Friday	07/07-11:22:02	SNMP AgentX/tcp request	Classification: Attempted Information Leak	TCP	2	172.16.0.1:35609	192.168.10.50:705
Friday	07/07-11:22:02	COMMUNITY WEB-MISC mod_jrun overflow attempt	Classification: Web Application Attack	TCP	1	192.168.10.5:54705	64.30.224.172:80
Friday	07/07-11:22:02	INFO web bug 0x0 gif attempt	Classification: Misc activity	TCP	3	64.30.224.172:80	192.168.10.5:54705

Figura 3.3. Muestra del archivo csv cargado en un programa de hojas de cálculo

Por otro lado, es posible detectar falsos positivos comprobando si las direcciones IP mostradas en la alerta generada corresponden con las que figuran en el fichero Excel que acompaña al PCAP de entrada. En el caso de que no correspondan la alerta generada es un falso positivo.

Por este motivo, con el fin de comprobar si Snort ha sido capaz de detectar los ataques que figuran en el archivo PCAP procesado, se filtra el *log* en busca de una alerta de los ataques y/o direcciones IP del atacante o víctima. Si se encuentra una alerta o alertas que correspondan a un ataque, el ataque se marca como "detectado" y la alerta se considera como verdadero positivo. En el caso de no haber ninguna alerta que corresponda, entonces el ataque se marca como "no detectado" (falso negativo). Si se genera una alerta que no corresponde a ningún ataque entonces, se clasificará como falso positivo. Hay alertas que se repiten las cuales pueden ser verdaderos positivos, ya que son generadas por los ataques del archivo PCAP, y que otras veces también son generadas con tráfico no perteneciente al ataque, por lo que también se clasifican como falsos positivos. Debido a esto, con el fin de simplificar, aquellas alertas que se hayan clasificado tanto como verdaderos positivos como falsos positivos, se considerarán como alertas de verdaderos positivos a la hora de analizar los resultados. Se considera un verdadero positivo cuando una alerta generada corresponde a un ataque que figura en el fichero Excel, para ello además del tipo de ataque se comprueba que coincidan las direcciones IP y puertos. Se considera un falso positivo cuando una alerta generada no corresponde a ningún ataque que figure en el fichero Excel. Se considera falso negativo cuando un ataque no ha sido detectado, es decir, no se ha generado ninguna alerta correspondiente a dicho ataque.

Una vez hecho esto con cada archivo PCAP, se obtiene el número total de ataques detectados. Para poder conseguir unos resultados más detallados es necesario procesar el archivo alert.txt teniendo en cuenta cada alerta que figura. Se ha escrito otro script que, basándose en qué ataques han sido clasificados como "detectados" y proporcionando la identificación de las alertas de verdaderos positivos, procesa el archivo alert.txt y produce un archivo de resultados con la cantidad de verdaderos positivos y falsos positivos (ver tabla 3.2).

Configuración	Verdaderos positivos	Falsos positivos
1	10	35
2	1044	443
3	1044	451

Tabla 3.2. Verdaderos positivos y falsos positivos

Se han obtenido resultados con diferentes configuraciones relacionadas con la prioridad de las alertas, con el fin de determinar si la precisión de detección aumenta y los falsos positivos disminuyen dependiendo de dicha prioridad. En concreto, se han estudiado tres configuraciones:

- Configuración 1: solo se tienen en cuenta las alertas de prioridad 1.
- Configuración 2: solo se tienen en cuenta las alertas de prioridad 1 y 2.
- Configuración 3: se tienen en cuenta las alertas de prioridad 1, 2 y 3.

Suricata

Este es el comando utilizado en Suricata para procesar archivos PCAP:

```
sudo suricata -r /home/ubuntu/shares/Material/Friday-WorkingHours.pcap
```

Una vez procesado el archivo, Suricata habrá generado cuatro archivos con resultados:

- **stats.log:** contiene estadísticas de la ejecución.
- **suricata.log:** archivo que indica el número de núcleos usados, número total de alertas, archivos procesados y demás.
- **even.json:** contiene las alertas, metadatos, anomalías y demás en formato JSON, lo que facilita el procesado de los resultados en programas de terceros.
- **fast.log:** contiene todas las alertas.

A la hora de procesar los resultados se va a utilizar el archivo fast.log (ver figura 3.4). Tiene el mismo formato que alert.txt de Snort.

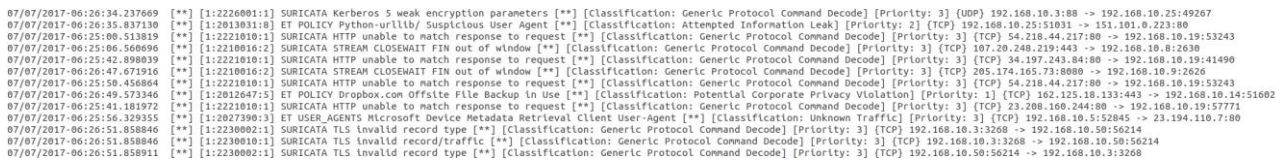

Figura 3.4. Un extracto de fast.log

Al igual que en Snort, la alerta se divide en varias partes:

- Fecha y hora y número de identificación de la regla:

```
07/07/2017-06:26:35.837130  [**] [1:2013031:8]
```

- El mensaje de la alerta y la clasificación:

```
ET POLICY Python-urllib/ Suspicious User Agent [**]
[Classification: Attempted Information Leak]
```

- Prioridad, protocolo y direcciones IP / puertos origen y destino:

```
[Priority: 2] {TCP} 192.168.10.25:51031 -> 151.101.0.223:80
```

Gracias a que Snort y Suricata comparten la sintaxis de las alertas, se han utilizado los mismos scripts para procesar los datos. Se han usado las mismas tres configuraciones de prioridad.

Una vez obtenidos los datos con ambos IDS el siguiente paso es analizar, comparar y sacar conclusiones.

3.6.2 Criterios de evaluación

Durante el proyecto han surgido diferentes problemas a la hora de realizar las comprobaciones entre los resultados y los ataques, por lo que ha sido necesario cambiar el método de evaluación. A continuación, se detallan la propuesta inicial, los problemas encontrados y la solución adoptada.

Propuesta inicial

Cuando Suricata y Snort procesan un archivo PCAP, crean un *log* que contiene un listado de las alertas que se han generado. Ambos comparten la misma sintaxis de alertas, en las que se detalla el tipo de ataque, dirección IP de origen y destino del paquete y más. Por otra parte, se dispone del fichero Excel asociado a cada PCAP con la información de los flujos de paquetes.

Gracias a esta información, el planteamiento inicial de evaluación consistía en escribir un script cuya función sería recoger los datos del *log* de alertas, examinarlo conjuntamente con el fichero Excel asociado al dataset, y realizar una comparación entre cada flujo de paquetes maligno y las alertas generadas. De esta manera, las alertas generadas que no coincidan con ningún flujo de paquetes se considerarían falsos positivos, y los flujos de paquetes malignos que no coincidan con ninguna alerta serían falsos negativos. Al realizarse los ataques de manera reiterada, con el fin de asociar cada alerta al flujo de paquetes correspondiente, se haría uso de las marcas de tiempo incluidas tanto en el *log* de alertas como en el fichero Excel.

Nos encontramos con varios problemas a la hora de llevar a cabo la evaluación siguiendo este criterio. El primero es que las horas incluidas en las alertas generadas tanto por Snort o Suricata no concuerdan con las mostradas en el Excel del dataset, por lo que no pueden usarse las horas para identificar el flujo de paquetes correspondiente. Se han realizado diferentes pruebas con el fin de identificar la razón, sin éxito, y por lo tanto, no ha sido posible encontrar una solución.

El segundo es que los IDS disponen de un límite de alertas que pueden generarse debido a un tipo de ataque en un cierto espacio de tiempo, por lo que el número de alertas generadas es posible que no coincida con el número de flujos de paquetes que figuran en el fichero Excel. Por este motivo, no es posible realizar una comparación entre cada flujo de paquetes maligno y las alertas generadas.

Solución adoptada

El procedimiento de la evaluación adoptado finalmente es el siguiente. Se realiza un análisis del *log* de alertas en busca de aquellas correspondientes a los ataques que figuran en la documentación del dataset. Todas las alertas, para las que se encuentre una correspondencia, se clasifican como *verdaderos positivos*. Aquellas para las que no se encuentre correspondencia se clasifican como *falsos positivos.* Por último, los ataques documentados en el dataset que no tengan alerta correspondiente se clasifican como *falsos negativos*.

4 Evaluación

En este apartado se explican en detalle los resultados de la evaluación de Snort y Suricata, y se hace una comparación entre ellos.

4.1 Criterios de comparación

El análisis se ha realizado con las siguientes versiones de los IDS y reglas:

- Snort 2.7.0 con snapshot 2700
- Snort 2.9.17 con snapshot 29170 y snapshot 2700
- Suricata 5.0.3 con el conjunto de reglas de Emerging Threats a fecha de octubre del 2020

Términos utilizados:

- Alertas totales generadas, de las cuales se especifica el número de alertas de verdaderos positivos y falsos positivos.
- Alertas únicas totales, a la hora de obtener estos datos solo se tiene en cuenta cada alerta una única vez en cada archivo PCAP. Así, es posible evitar posibles equivocaciones a la hora de interpretar los datos. Un ataque puede generar miles de alertas, que si se consideran por separado (cada una como un ataque) pueden desvirtuar los resultados.
- Ataques detectados, aquí se especifica cuántos de los 17 ataques documentados en el dataset han sido detectados.
- Verdaderos positivos: alertas que corresponden a ataques que figuran en los archivos PCAP.
- Falsos positivos: alertas que no corresponden a ningún ataque que figure en los archivos PCAP.
- Falsos negativos: ataques que no han sido detectados, es decir, no se ha generado ninguna alerta correspondiente a dicho ataque.
- Configuración 1: solo se tienen en cuenta las alertas de prioridad 1 a la hora de contabilizar los resultados, ignorando las menos importantes (valor mayor de prioridad).
- Configuración 2: se tienen en cuenta las alertas de prioridad 1 y las de prioridad 2.
- Configuración 3: se tienen en cuenta las alertas con cualquier nivel de prioridad (1, 2 y 3).

4.2 Evaluación de Snort 2.7.0

Estos son los resultados obtenidos al procesar todos los archivos PCAP en Snort 2.7.0:

Alertas totales generadas:

Configuración	Verdaderos positivos	Falsos positivos	Porcentaje de verdaderos positivos, sobre el total
1	935	3764	19.9%
2	13736	6542	67.7%
3	15461	24525	38.6%

Tabla 4.1. Alertas totales generadas en Snort 2.7.0

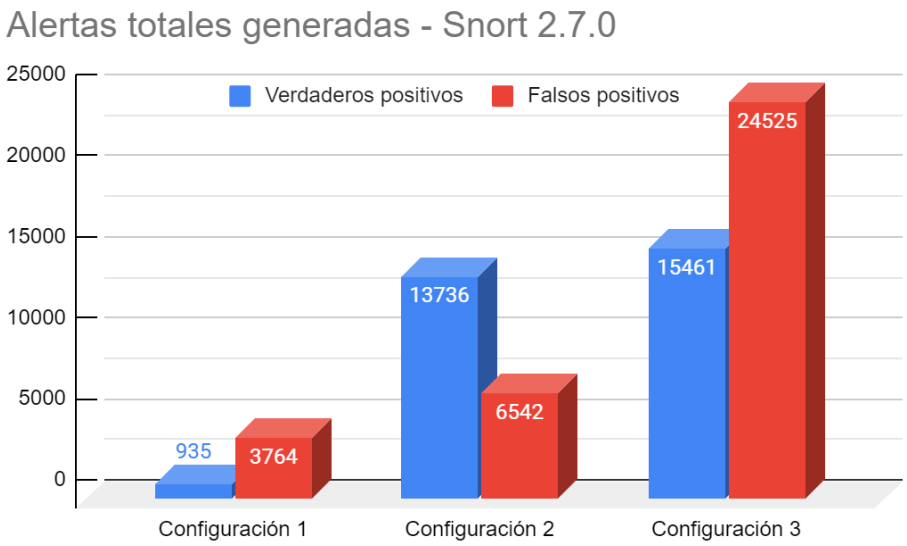

Figura 4.1. Alertas totales generadas en Snort 2.7.0

Alertas únicas totales:

Configuración	Verdaderos positivos	Falsos positivos	Porcentaje de verdaderos positivos, sobre el total
1	1	13	7.6%
2	9	30	30%
3	14	45	31.1%

Tabla 4.2. Alertas únicas totales en Snort 2.7.0

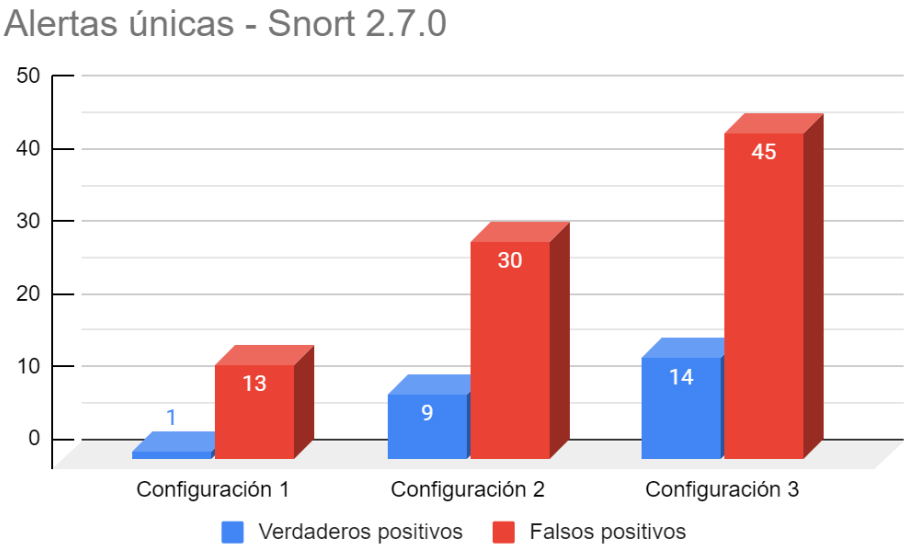

Figura 4.2. Alertas únicas en las 3 configuraciones en Snort 2.7.0

Si observamos los resultados obtenidos con la **configuración 1** (ver tabla 4.1 y figura 4.1), **935** verdaderos positivos y **3764** falsos positivos, tan solo el **19.9%** de las alertas generadas son verdaderos positivos. Por lo que ni un cuarto de las alertas generadas pertenecen a ataques reales. Sin embargo, con la **configuración 2**, la efectividad sube hasta un **67.7%**. Por último, con la **configuración 3**, las alertas de verdaderos positivos bajan a un **38.6%.** Teniendo en cuenta estos

datos, la opción que mejor porcentaje obtiene es la **configuración 2**. Con respecto a la **configuración 1,** se incrementa el número de alertas positivas en **12801,** mientras que las alertas que son falsos positivos solo aumentan en **2778.**

Si la comparamos con la **configuración 3**, esta última no solo obtiene un porcentaje de verdaderos positivos bastante menor (**38.6%**), sino que se generan una gran cantidad de falsos positivos (**24525**), mientras que la cantidad de alertas de verdaderos positivos solo aumentan en **1725**. Además, el porcentaje de alertas únicas es **31.1%**, mientras que la configuración obtiene prácticamente el mismo porcentaje, **30%** (ver tabla 4.2 y figura 4.2).

No obstante, aunque el **67.7%** de las alertas sean verdaderos positivos, si solo se tienen en cuenta las alertas únicas, tan solo el **30%** son verdaderos positivos. Por lo tanto, una o varias alertas se han generado una gran cantidad de veces.

Detección de ataques: cuántos y cuáles de los ataques en el dataset han sido detectados correctamente.

Número de ataques totales 17, de los cuales **6** han sido detectados por **Snort 2.7.0** en las configuraciones de prioridad 2 y 3.

Martes

- Ataques de fuerza bruta
  - SSH-Patator (No detectado)
  - FTP-Patator (Detectado)

Miércoles

- Ataques DoS y DDoS
  - DoS Slowloris (No detectado)
  - DoS Slowhttptest (No detectado)
  - DoS Hulk (No detectado)
  - DoS GoldenEye (No detectado)
- Heartbleed (Puerto 444) (No detectado)

Jueves

- Ataques Web
  - Fuerza bruta (No detectado)
  - XSS (No detectado)
  - Inyección SQL (No detectado)
- Infiltración
  - Meta exploit (No detectado)
  - Cooldisk (No detectado)
  - Descarga por Dropbox (Detectado)
  - Escaneo de puertos (Detectado)
  - Nmap (Detectado)

Viernes

- Botnet ARES (Detectado)
- Escaneo de puertos (Detectado)
- DDoS LOIC (No detectado)

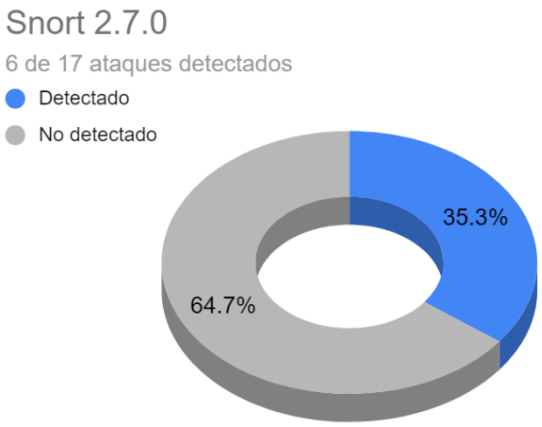

Figura 4.3. Ataques detectados en Snort 2.7.0

Ataques detectados

	Detectados	Porcentaje
Configuración 1	1	5.88%
Configuración 2	6	35.2%
Configuración 3	6	35.2%

Tabla 4.3. Ataques detectados en Snort 2.7.0

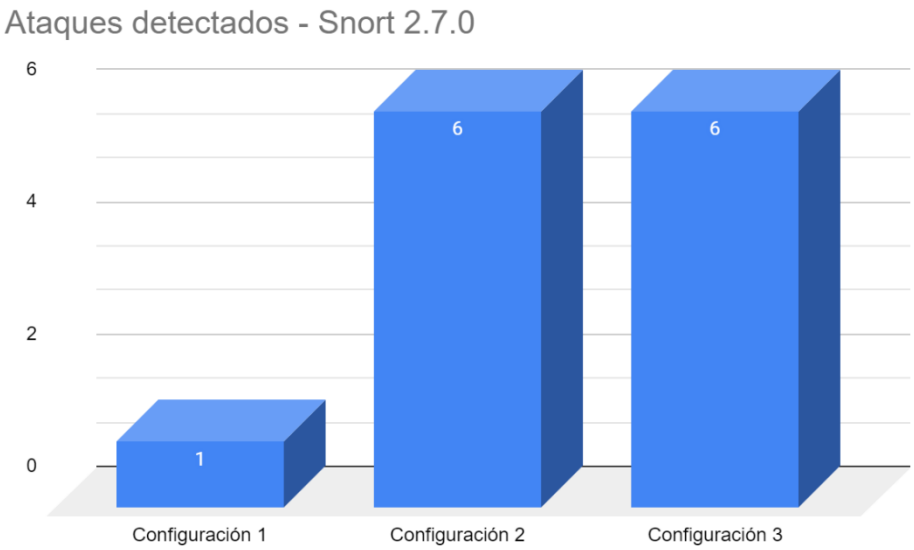

Figura 4.4. Ataques detectados en las 3 configuraciones Snort 2.7.0

Análisis

Ninguno de los ataques **DDoS** ha sido detectado. Mientras que los escaneos de puertos del jueves y viernes, **botnet Ares**, **Nmap**, descarga por **Dropbox** y el ataque de fuerza bruta **FTP-patator** han sido identificados con éxito.

El máximo de ataques detectados han sido **6** de **17** ataques, por lo que ha habido **11** falsos negativos (ver figura 4.3). Con la **configuración 2** (ver tabla 4.3 y ver figura 4.4)**,** también se detectan **6** ataques, lo que indica que ninguna de las alertas de verdaderos positivos es de **prioridad 3**. En la **configuración 1** tan solo se detecta un ataque, por lo que claramente es la menos efectiva a la hora de detectar ataques.

Conclusiones

Después de analizar los datos obtenidos, la **configuración 2** es claramente la más efectiva. No solo tiene el mayor porcentaje de alertas de verdaderos positivos (**67,7%**), sino que detecta la misma cantidad de ataques que si no se filtrase por prioridades. Por otro lado, solo **6** (**35,3%**) de los **17** ataques que figuran en el fichero Excel son detectados, mientras que con la **configuración 1** se detecta tan solo un ataque y con la **configuración 3**, obteniendo prácticamente el mismo porcentaje de alertas únicas positivas y una mucho mayor cantidad de alertas generadas, se detectan esos mismos **6** ataques.

4.3 Evaluación de Snort 2.9.17

Estos son los resultados obtenidos al procesar todos los archivos **PCAP** en **Snort 2.9.17**:

Alertas totales generadas:

Configuración	Verdaderos positivos	Falsos positivos	Porcentaje de verdaderos positivos, sobre el total
1	10	35	22.2%
2	1044	443	70.2%
3	1044	451	69.8%

Tabla 4.4. Alertas totales generadas en Snort 2.9.17

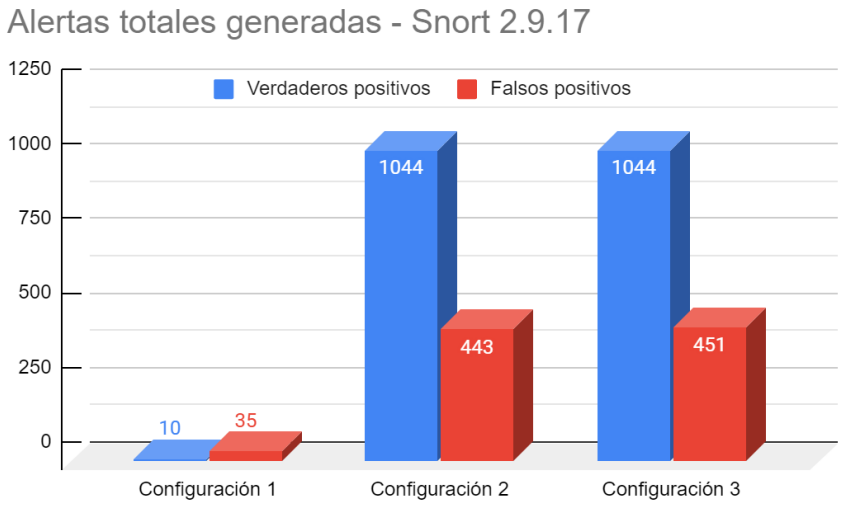

Figura 4.5. Alertas totales generadas en Snort 2.9.17

Alertas únicas totales:

Configuración	Verdaderos positivos	Falsos positivos	Porcentaje de verdaderos positivos, sobre el total
1	1	1	50%
2	2	2	50%
3	2	3	40%

Tabla 4.5. Resultados de Snort 2.9.17

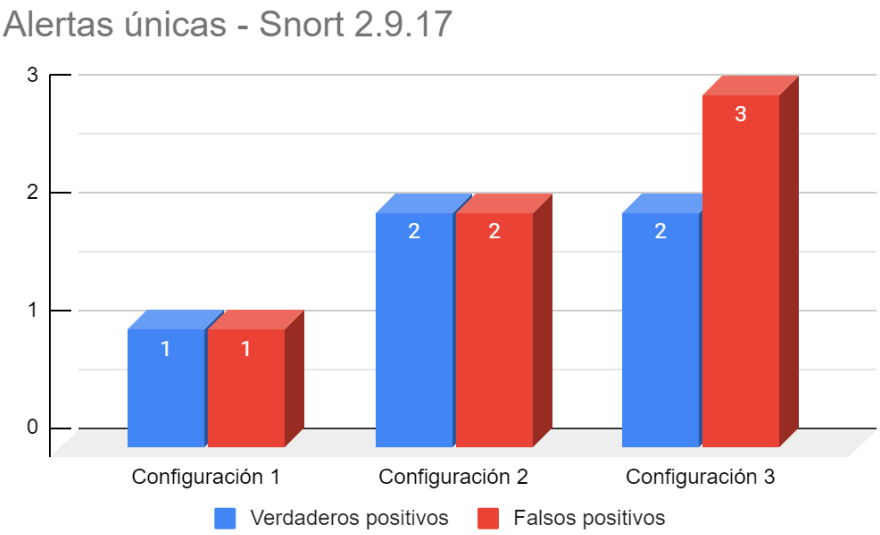

Figura 4.6. Alertas únicas en las 3 configuraciones en Snort 2.9.17

En la **configuración 3** de las **1495** alertas generadas tan solo hay **5** alertas únicas, de las cuales **2** son verdaderos positivos y **3** son falsos positivos (ver tablas 4.4 y 4.5, y ver figuras 4.5 y 4.6). Por lo tanto, una o varias alertas se han generado repetidas veces en gran cantidad.

El número de alertas generadas con la **configuración 1** es muy reducido, tan solo se generan **45** alertas de las cuales solo el **22.2%** son verdaderos positivos.

Con la **configuración 2**, el número de alertas generadas aumenta a **1487**. Además, el **70.2%** de esas alertas generadas son verdaderos positivos.

La **configuración 3** obtiene prácticamente los mismos resultados que la **configuración 2**: misma cantidad de alertas positivas con **8** falsos positivos más.

Respecto a las alertas únicas, la **configuración 1** genera **1** alerta de verdadero positivo y una de falso positivo, mientras que tanto la **configuración 2** como la **configuración 3** generan **2** alertas de verdaderos positivos. No obstante, la **configuración 3** genera **3** alertas de falsos positivos frente a las **2** de la **configuración 2**.

Detección de ataques

Número de ataques totales 17, de los cuales 2 han sido detectados por Snort 2.9.17

Martes

- Ataques de fuerza bruta
  - SSH-Patator (No detectado)
  - FTP-Patator (No detectado)

Miércoles

- Ataques DoS y DDoS
  - DoS Slowloris (No detectado)
  - DoS Slowhttptest (Detectado)
  - DoS Hulk (No detectado)
  - DoS GoldenEye (No detectado)
- Heartbleed (Puerto 444) (No detectado)

Jueves

- Ataques Web
  - Fuerza bruta (No detectado)
  - XSS (No detectado)
  - Inyección SQL (Detectado)
- Infiltración
  - Meta exploit (No detectado)
  - Cooldisk (No detectado)
  - Descarga por Dropbox (No detectado)
  - Escaneo de puertos (No detectado)
  - Nmap (No detectado)

Viernes

- Botnet ARES (No detectado)
- Escaneo de puertos (No detectado)
- DDoS LOIC (No detectado)

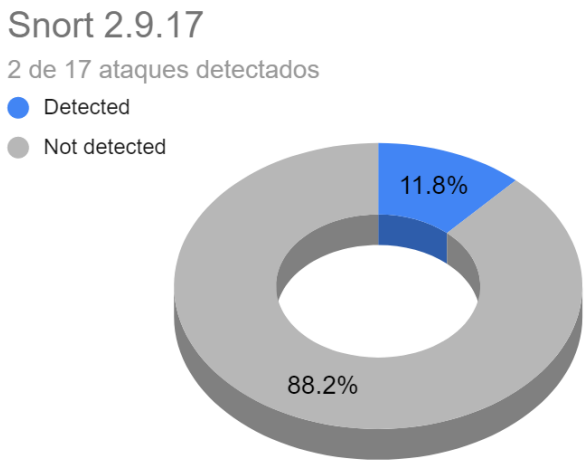

Figura 4.7. Gráficos de los resultados en Snort 2.9.17

Ataques detectados

	Detectados	Porcentaje
Configuración 1	1	5.88%
Configuración 2	2	11.7%
Configuración 3	2	11.7%

Tabla 4.6. Ataques detectados en Snort 2.9.17

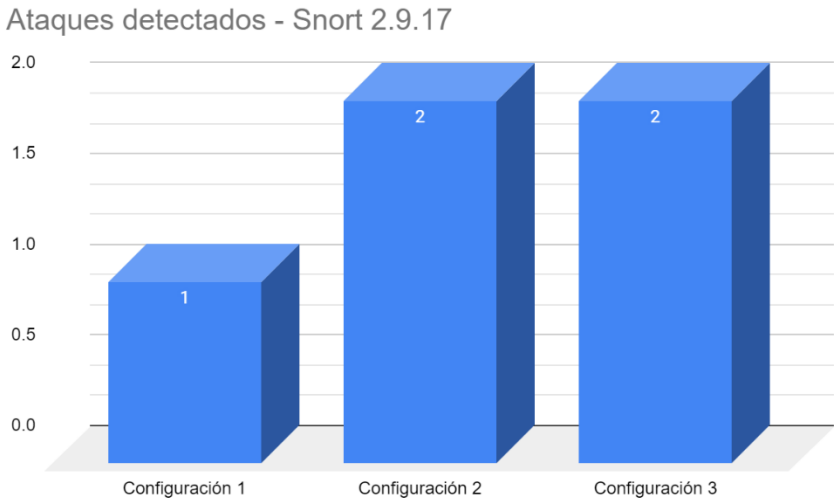

Figura 4.8. Ataques detectados en las 3 configuraciones en Snort 2.9.17

Análisis

En la mejor de las configuraciones testadas se han detectado tan solo **2** ataques de los **17** que figuran en el archivo **PCAP,** por lo que ha habido **15** falsos negativos (ver figura 4.7). Con la **configuración 1** se detecta un ataque y con la **configuración 2** se detectan dos ataques (ver tabla 4.6 y figura 4.8), que son **DDoS Slowhttptest** e **inyección SQL** (los mismos detectados con la **configuración 3**). Con la **configuración 1** solo se detecta el ataque de **inyección de SQL**.

Conclusiones

La cantidad de alertas generadas es muy reducida en todas las configuraciones, por lo que no es un factor a tener en cuenta a la hora de determinar qué configuración es la mejor. Por lo tanto, la configuración que más ataques detecte es la más efectiva.

La **configuración 2** y la **configuración 3** detectan **2** de los **17** ataques, mientras que la **configuración 1** tan solo detecta **uno**. No obstante, aunque la **configuración 3** detecte **2** ataques, también genera **3** alertas únicas de falsos positivos respecto a las **2** generadas por la **configuración 2**.

Por lo tanto, se ha concluido que **Snort 2.9.17** es muy poco efectivo a la hora de detectar los ataques que figuran en el archivo PCAP, independientemente de la configuración de prioridad elegida.

4.4 Evaluación de Suricata 5.0.3

Estos son los resultados obtenidos al procesar todos los archivos **PCAP** en **Suricata 5.0.3**:

Alertas totales generadas:

Configuración	Verdaderos positivos	Falsos positivos	Porcentaje de verdaderos positivos, sobre el total
1	23860	752	96.9%
2	23862	3208	88.1%
3	23870	132607	15.2%

Tabla 4.7. Alertas totales generadas en Suricata 5.0.3

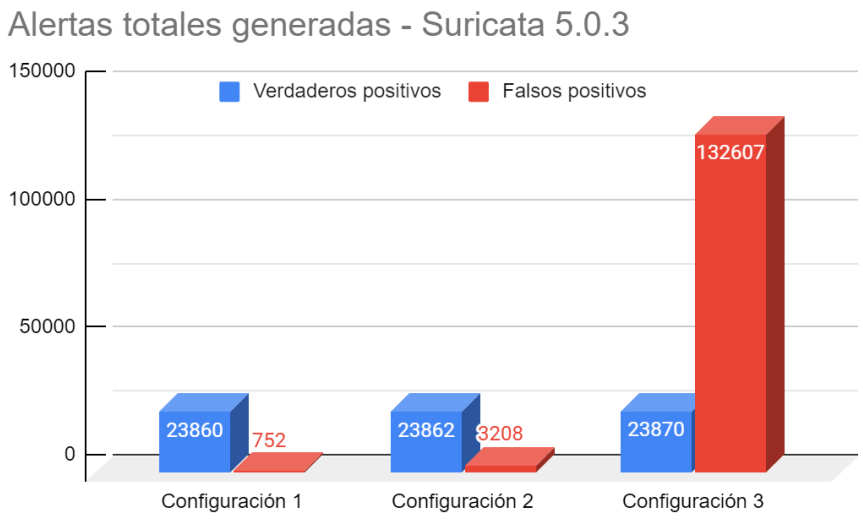

Figura 4.9. Alertas totales generadas en Suricata 5.0.3

Alertas únicas totales

Configuración	Verdaderos positivos	Falsos positivos	Porcentaje de verdaderos positivos, sobre el total
1	4	11	26.6%
2	5	23	17.8%
3	8	70	10.2%

Tabla 4.8. Alertas únicas totales en Suricata 5.0.3

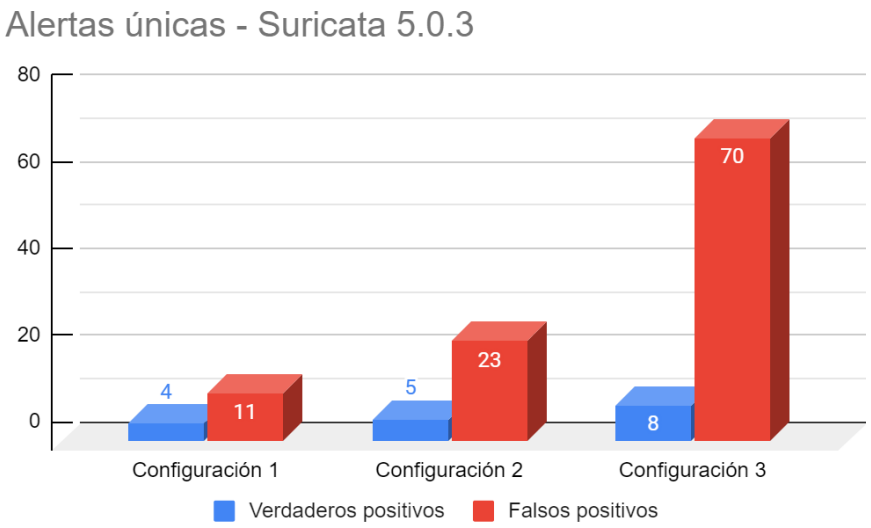

Figura 4.10. Alertas únicas en las 3 configuraciones en Suricata 5.0.3

En las **configuraciones 1** y **2**, los porcentajes de alertas de verdaderos positivos es muy alto, **96.9%** y **88.1%** respectivamente (ver tabla 4.7 y figura 4.9)**.** Con la **configuración 2** solo se detecta un verdadero positivo más que con la **configuración 1** (**5** frente a **4**) mientras que el número de alertas únicas totales correspondientes a falsos positivos asciende de **11** a **23**.

En cuanto a la **configuración 3**, podemos ver que la cantidad de falsos positivos se incrementa en **129399** respecto a la **configuración 2**. Lo que supone un **4133%** más de falsos positivos. En cuanto al número de alertas únicas, se generan **47** más (**70**) que con la **configuración 2**, lo que supone un gran aumento de falsos positivos, mientras que el número de alertas únicas de verdaderos positivos solo aumenta en **3** (**8**) respecto a las generadas con la **configuración 2**.

Pese a que las **configuraciones 1** y **2** generan muchas más alertas de verdaderos positivos que de falsos positivos, se generan más alertas únicas de falsos positivos que de verdaderos positivos (ver tabla 4.8 y figura 4.10). Por lo que una o más alertas de verdaderos positivos se generan en gran cantidad de manera repetida. La **configuración 1** genera la mitad de alertas únicas de verdaderos positivos (**4**) que la **configuración 3.** No obstante, la **configuración 3** genera **59** alertas únicas más de falsos positivos (**70**).

Detección de ataques

Número de ataques totales 17 de los cuales 6 han sido detectados por Suricata 5.0.3.

Martes

- Ataques de fuerza bruta
  - SSH-Patator (No detectado)
  - FTP-Patator (No detectado)

Miércoles

- Ataques DoS y DDoS
  - DoS Slowloris (No detectado)
  - DoS Slowhttptest (No detectado)
  - DoS Hulk (No detectado)
  - DoS GoldenEye (No detectado)
- Heartbleed (Puerto 444) (Detectado)

Jueves

- Ataques Web
  - Fuerza bruta (No detectado)
  - XSS (No detectado)
  - Inyección SQL (No detectado)
- Infiltración
  - Meta exploit (Detectado)
  - Cooldisk (No detectado)
  - Descarga por Dropbox (Detectado)
  - Escaneo de puertos (Detectado)
  - Nmap (Detectado)

Viernes

- Botnet ARES (No detectado)
- Escaneo de puertos (Detectado)
- DDoSLOIC (No detectado)

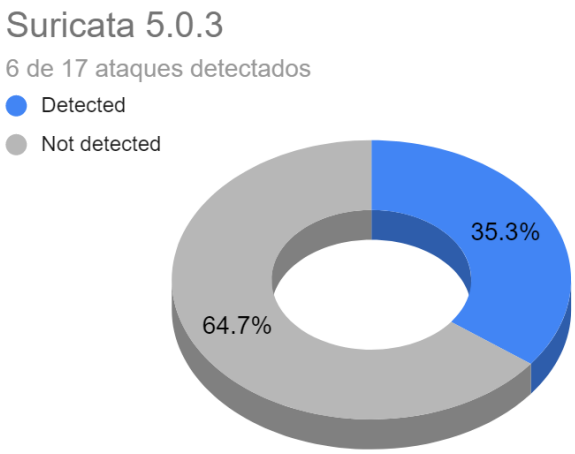

Figura 4.11. Ataques detectados en Suricata 5.0.3

Ataques detectados

	Detectados	Porcentaje
Configuración 1	3	17.6%
Configuración 2	4	23.5%
Configuración 3	6	35.2%

Tabla 4.9. Ataques detectados en Suricata 5.0.3

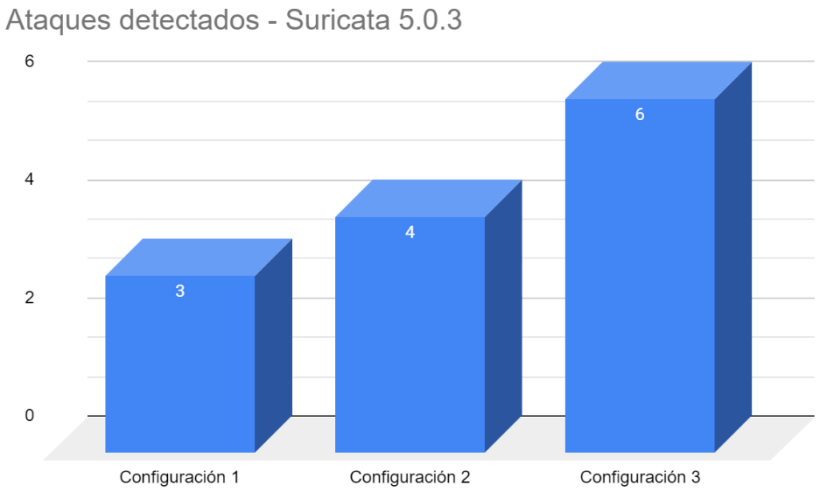

Figura 4.12. Ataques detectados en las 3 configuraciones en Suricata 5.0.3

Análisis

Se han detectado hasta **6** de **17** ataques en la **configuración 3** (ver figura 4.11), por lo que ha habido **11** falsos negativos. Estos son los ataques detectados: el ataque de **Heartbleed**, **Meta Exploit**, descarga por **Dropbox**, **Nmap** y escaneo de puertos (jueves y viernes).

La **configuración 2** ha detectado **2** ataques menos (**4**) que la **configuración 3** (**6**) y la configuración **1**, **3** ataques menos (**3**). Por lo que con la **configuración 1**, se han detectado la mitad de los ataques que se hubieran detectado con la **configuración 3** (ver tabla 4.9 y figura 4.12).

Conclusiones

Después de haber analizado los resultados, es evidente que la **configuración 3** de **Suricata** no es del todo viable. Aunque, es cierto que es la que sirve para detectar más ataques reales, introduce una enorme cantidad de falsos positivos (incrementan en un **4133%** respecto a la **configuración 2**), lo que dificulta de manera significativa la gestión de alertas, por lo tanto, la efectividad. Además, el número de alertas únicas de falsos positivos que genera (**70**) aumenta de manera significativa respecto a la **configuración 2** (**23**), mientras que tan solo genera **3** alertas únicas más (**8**) respecto la **configuración 2** (**5**).

En cuanto a la **configuración 1**, genera una cantidad similar de alertas totales y alertas de verdaderos positivos respecto a la **configuración 2**, sin embargo, detecta solo **3** ataques. Por otro lado, la **configuración 2** detecta **4** ataques.

Por lo tanto, la **configuración 2** es la más efectiva, ya que genera una cantidad similar de alertas de verdaderos positivos y falsos positivos a la **configuración 1**, pero detecta **2** ataques más.

4.5 Evaluación con el conjunto de reglas snapshot 2700

Por último, se va a realizar una última prueba con el fin de determinar si la diferencia de resultados entre **Snort 2.7.0** y **Snort 2.9.17** es debida a las diferencias entre versiones o al conjunto de reglas. Esta prueba consiste en utilizar el conjunto de reglas de **Snort 2.7.0** (**snapshot 2700**) en **Snort 2.9.17**, comparado con el utilizado en la experimentación anterior (**snapshot 29170**).

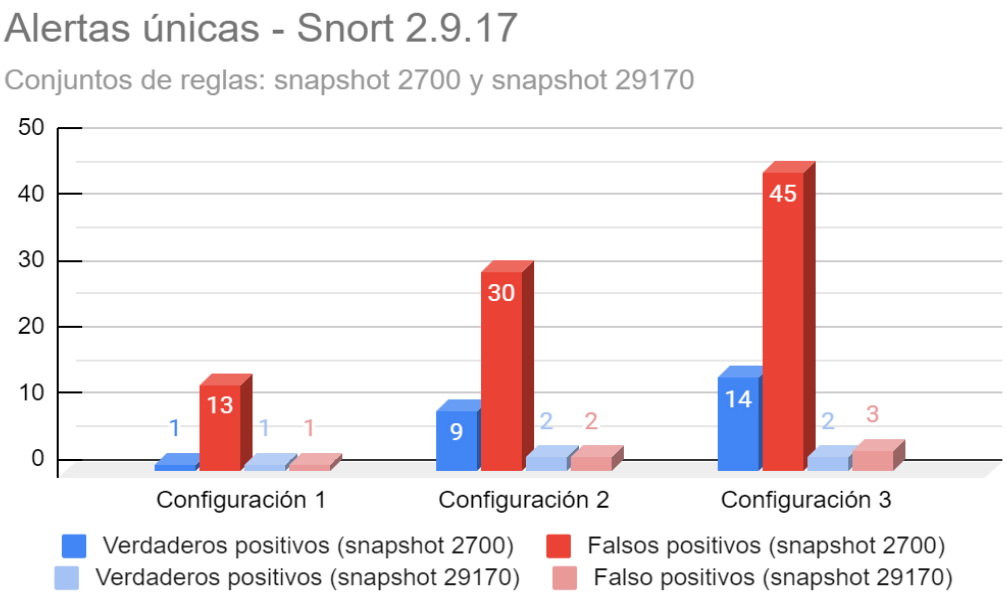

Figura 4.13. Comparación de alertas únicas generadas entre el conjunto de reglas snapshot 2700 y 29170 en Snort 2.9.17

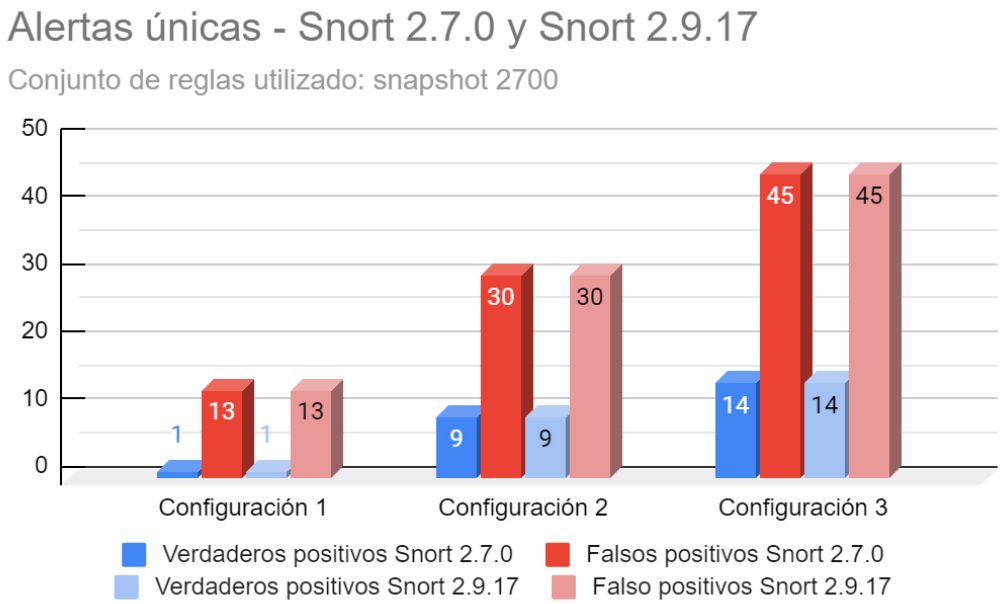

Figura 4.14. Comparación de alertas únicas generadas entre el Snort 2.7.0 (snapshot 2700) y Snort 2.9.17 con el conjunto de reglas snapshot 2700

Por un lado se ha realizado una comparación entre los conjuntos de reglas **snapshot 2700** y **snapshot 29170** de las alertas únicas generadas en diferentes configuraciones (ver figura 4.13).

Y por último se ha realizado una comparación entre **Snort 2.7.0** (**snapshot 2700**) y **Snort 2.9.17** de las alertas únicas generadas con el conjunto de reglas **snapshot 2700** (ver figura 4.14).

Tras obtener los resultados y analizarlos (ver figuras 4.13 y 4.14), se ha observado que

1. Con el conjunto de reglas **snapshot 2700** (comparado con el original, **snapshot 29170**), **Snort 2.9.17** genera, para cualquier configuración, muchas más alertas, tanto de verdaderos positivos como de verdaderos negativos.
2. **Snort 2.9.17** y **Snort 2.7.0** generan exactamente las mismas alertas cuando utilizan el mismo conjunto de reglas.

Por lo tanto, no podemos decir que **Snort 2.7.0** sea más efectivo que **Snort 2.9.17**. Al contrario: su efectividad es **idéntica**, y depende únicamente del conjunto de reglas. Concretamente, el conjunto **snapshot 2700** detecta muchos más ataques, a costa de un gran número de falsos positivos. El conjunto **snapshot 29170** es muy poco efectivo porque, aunque genera pocos falsos positivos, la capacidad de detección de ataques reales es escasa.

4.6 Comparación de resultados

4.6.1 Comparación

En este apartado, se va a realizar una comparación entre los resultados de los diferentes IDS. Los resultados que se van a utilizar para realizar la comparación son los obtenidos con la **configuración 2**, ya que, esta ha sido la más efectiva en todos los casos.

Alertas totales generadas

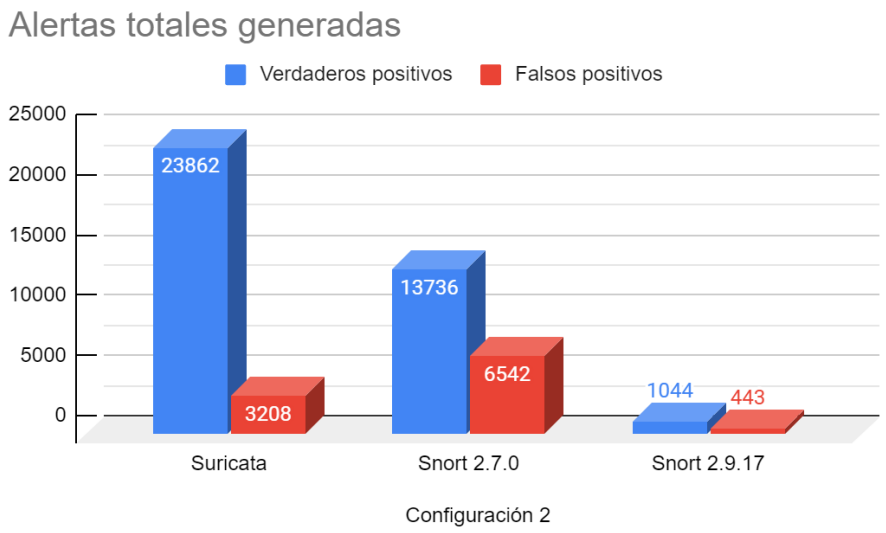

Figura 4.15. Comparación de alertas totales generadas entre Suricata, Snort 2.7.0 y Snort 2.9.17

Los tres generan más alertas de verdaderos positivos que de falsos positivos, **Suricata** un **88.1%**, **Snort 2.7.0** un **67.7%** y **Snort 2.9.17** un **70.2%**. En cuanto a la cantidad de alertas generadas, **Suricata** genera una gran cantidad de alertas (**27070**), lo que puede llegar a dificultar la gestión. Por otro lado, **Snort 2.7.0** genera una cantidad moderada de alertas (**20278**). Por último**, Snort 2.9.17**, genera una cantidad muy reducida de alertas (**1487**) (ver figura 4.15).

Alertas únicas totales

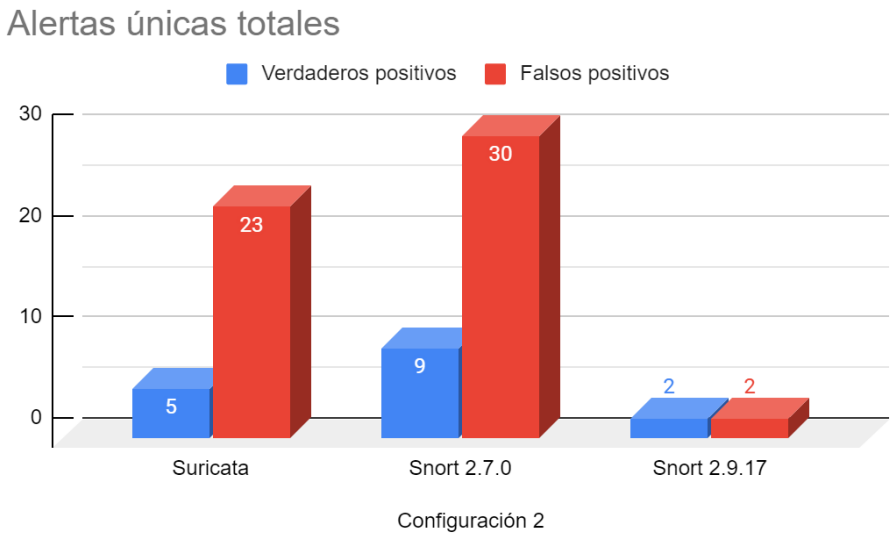

Figura 4.16. Comparación de alertas únicas totales entre Suricata, Snort 2.7.0 y Snort 2.9.17

En los tres casos, pese a generar una mayor cantidad de alertas de verdaderos positivos que de negativos, si solo se tienen en cuenta las alertas únicas, se generan más alertas de falsos positivos que de verdaderos positivos. Por lo tanto, en todos los casos una o más alertas de verdaderos positivos se generan reiteradamente en gran cantidad (ver figura 4.16).

Ataques detectados

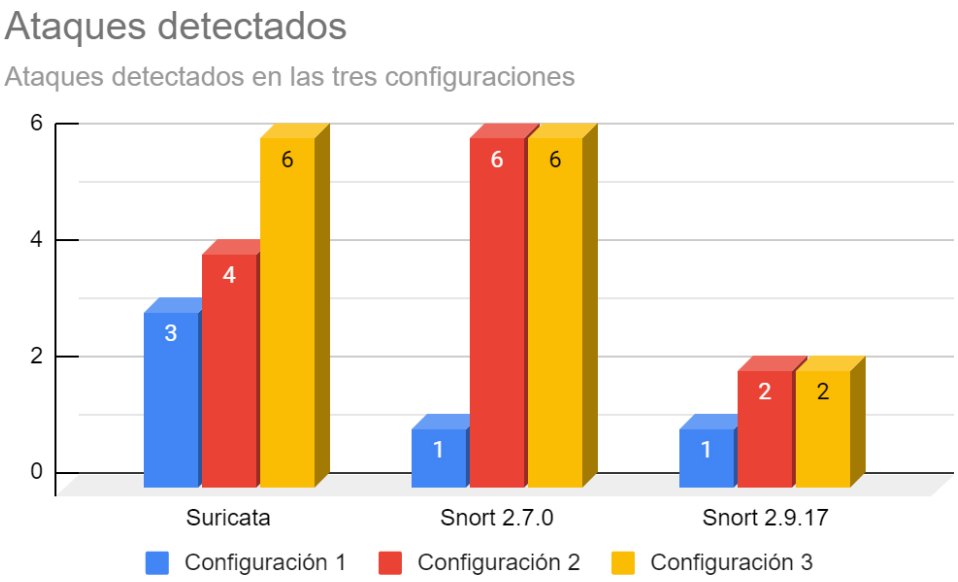

Figura 4.17. Comparación de ataques detectados entre Suricata, Snort 2.7.0 y Snort 2.9.17

A pesar de que se ha concluido que las **configuraciones 1** y **3** no son las más efectivas, cabe destacar que con la **configuración 1, Suricata 5.0.3** es el que más ataques detecta. Por otro lado, **Snort 2.7.0** con la **configuración 2** ya detecta el máximo de ataques que llega a detectar teniendo en cuenta todas las alertas (ver figura 4.17).

4.6.2 Conclusiones

La **configuración 2** ha sido la más efectiva en los tres casos. Con los tres IDS testados sido la más equilibrada, puesto que ofrece el mayor número de ataques detectados sin generar una cantidad excesiva de alertas de falsos positivos.

En cuanto a qué IDS ha sido el más efectivo, **Snort 2.9.17** queda descartado ya que ha sido el que menos ataques ha detectado (**2**), mientras que, **Suricata 5.0.3** ha detectado **4** ataques y **Snort 2.7.0.** ha detectado **6** ataques.

En cuanto al número de alertas generadas, **Snort 2.7.0** es el más efectivo: aunque **Suricata 5.0.3** obtenga un mayor porcentaje de alertas de verdaderos positivos, el número de alertas totales es mucho mayor al de **Snort**. Por lo tanto, **Snort** no solo detecta más ataques (**6**), sino que genera una cantidad de alertas más reducida.

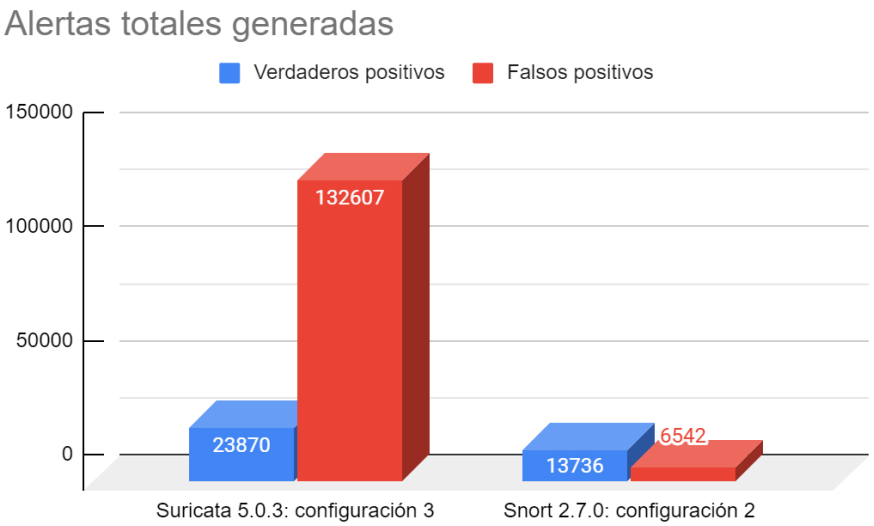

Figura 4.18. Comparación alertas totales generadas entre Suricata configuración 3 y Snort 2.7.0 configuración 2

Si realizamos la comparación con la **configuración 3** de **Suricata**, la cual detecta **6** ataques al igual que **Snort 2.7.0**, podemos ver que la cantidad de alertas generadas por **Suricata 5.0.3** es excesiva (ver figura 4.18), **156.477** alertas frente a las **20278** alertas de **Snort 2.7.0**.

Por lo tanto, **Snort 2.7.0** es el IDS más efectivo a la hora de detectar los ataques que figuran en las trazas de ataque utilizadas, tanto en la capacidad de detección como facilidad de gestión de alertas.

No obstante, aunque los mejores resultados los haya obtenido **Snort 2.7.0**, con tan solo un **35.3%** de los ataques detectados no podemos decir que es un IDS efectivo. Además, como se ha comprobado, en el análisis realizado con el conjunto de reglas **snapshot 2700** en **Snort 2.9.17**, que la efectividad depende completamente del conjunto de reglas utilizado, podemos concluir que el conjunto de reglas **snapshot 2700**, es el más efectivo de los evaluados independientemente de la versión de **Snort** utilizada.

5 Conclusiones y líneas de trabajo abiertas

En este apartado se van a detallar las conclusiones del proyecto y las posible futuras líneas de trabajo abiertas.

5.1 Conclusiones

Una vez finalizado el proyecto, podemos concluir que se han cumplido los objetivos establecidos. Se han conseguido y estudiado unas trazas de ataque que incluyen un variado número de ataques comunes, y nos han servido para poder simular un tráfico malicioso real en las pruebas. Se ha logrado un conocimiento sobre el funcionamiento de los sistemas de intrusiones, sobre todo de Snort y Suricata, los cuales han sido estudiados en detalle.

Otro componente vital de los sistemas de detección de intrusiones lo constituyen los conjuntos de reglas. Ha sido necesaria la comprensión de la sintaxis de las reglas para un efectivo análisis de los resultados. Además, con el fin de poder gestionar y procesar los resultados obtenidos, dentro de este proyecto ha sido necesario el desarrollo y uso de varios *scripts*.

Por último, la parte más importante de este trabajo ha sido el diseño y ejecución de una colección de experimentos, así como la comprensión de los resultados obtenidos para poder analizarlos. Durante el análisis se han extraído diferentes datos y gráficas de los resultados, los cuales han ayudado a determinar la efectividad de los sistemas de intrusión analizados, y de esta manera se ha concluido que Snort 2.7.0 con la configuración 2 es el IDS más efectivo de entre los comprobados. Por otro lado, en el análisis realizado con el conjunto de reglas snapshot 2700 se ha concluido que la efectividad de Snort 2.7.0 no es debida a la versión si no al conjunto de reglas utilizado y la efectividad de los IDS evaluados depende totalmente en el conjunto de reglas utilizado. Por lo tanto, el conjunto de reglas de Snort 2.7.0 (snortrules-snapshot-2700) ha sido el más efectivo a la hora de detectar los ataques incluidos en el dataset CIC-IDS-2017.

No obstante, la efectividad del conjunto de reglas de Snort 2.7.0 deja mucho que desear, detectando tan solo 6 de los 17 ataques incluidos en el dataset CIC-IDS-2017, por lo que se ha concluido que para lograr una mejor efectividad es necesario adaptar y optimizar el conjunto de reglas a las necesidades que presente la red en la que se desee instalar el IDS.

5.2 Líneas de trabajo abiertas

Debido a que, como se ha concluido en este TFG, la capacidad de detección de un IDS se debe en completamente al conjunto de reglas utilizado, se considera interesante estudiar posibles mejoras para los conjuntos de reglas como futuras líneas de trabajo abiertas.

También sería interesante realizar un estudio en la red en la que se vaya a instalar el sistema de detección de intrusiones. De esta manera, sería posible determinar a qué tipo de ataques es vulnerable la red y así adaptar el conjunto de reglas a las necesidades de la red. Una vez realizado el estudio se pueden crear, añadir o editar las reglas existentes con el fin de aumentar la posibilidad de detección de los ataques.

Por otro lado, los conjuntos de reglas utilizados tienen un número muy elevados de reglas, muchas de las cuales son las responsables de generar la mayoría de los falsos positivos. Estas reglas pueden ser borradas o comentadas con el fin de reducir la cantidad de falsos positivos y así mejorar y facilitar la gestión de alertas. Por este motivo, una posible línea abierta podría ser el realizar un estudio del conjunto de reglas y determinar cuáles son problemáticas o propensas a generar falsos positivos.

Bibliografía

[1] Lawal B.O., Longe O.Babatope y Ibitola A.: Strategic Sensor Placement for Intrusion Detection in Network-based IDS. *International Journal of Intelligent Systems and Applications*, 1, 2014.
https://www.researchgate.net/publication/272853714_Strategic_Sensor_Placement_for_Intrusion_Detection_in_Network-Based_IDS

[2] University of New Brunswick. CIC- IDS2017 Intrusion Detection Evaluation Dataset (CIC-IDS2017). *https://www.unb.ca/cic/datasets/ids-2017.html.* Visitada el 10/2020

[3] Sharafaldin I., Lashkari A. Habibi y Ghorbani Ali A. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. Canadian Institute for Cybersecurity (CIC), University of New Brunswick (UNB), Canada.
https://www.scitepress.org/Papers/2018/66398/66398.pdf

[4] Lincoln Laboratory Massachusetts Institute of Technology. 1999 DARPA Intrusion Detection Evaluation Dataset. *https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset*

[5] Thakkar A. y Lohiya R.: A review if the Advancement in Intrusion Detection Datasets. *International Conference on Computational Intelligence and Data Science (ICCIDS 2019), 2019.*
https://www.sciencedirect.com/science/article/pii/S1877050920307961

[6] Panigrahi R. y Borah S.: A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems. *International Journal of Engineering & Technology, 24, 2, 2018.*
https://www.researchgate.net/publication/329045441_A_detailed_analysis_of_CICIDS2017_dataset_for_designing_Intrusion_Detection_Systems

[7] Elasticsearch NV. ELK Stack. *https://www.elastic.co/what-is/elk-stack*.

[8] Lanjelot. Patator repository. *https://github.com/lanjelot/patator*.

[9] Emerging Threats. Conjunto de reglas, Suricata. Visitada el 10/2020.
http://rules.emergingthreats.net/open/suricata-5.0/

[10] Snort. Paquetes de instalación de Snort y conjuntos de reglas de Snort.
https://www.snort.org/downloads/#rule-downloads. Visitada el 11/2020

[11] Shekyan. Slowhttptest repository. *https://github.com/shekyan/slowhttptest*

[12] Jseidl. GoldenEye repository. *https://github.com/jseidl/GoldenEye*

[13] Snort home page. *https://www.snort.org/*. Visitada el 10/2020.

[14] Suricata home page. https://suricata-ids.org/. Visitada el 10/2020.

[15] Cunningham RK, Lippmann RP, Fried DJ, Garfinkel SL, Graf I, Kendall KR, et al. Evaluating intrusion detection systems without attacking your friends. The 1998 DARPA intrusion detection evaluation. Massachusetts Institute of Technology Lexington Lincoln Lab, 1999. *https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.36.6039&rep=rep1&type=pdf*

[16] Taghi M. Khoshgoftaar y Joffrey L. Leevy. A survey and analysis of intrusion detection models based on CSE-CIC-IDS2018 Big Data. *Journal of Big Data, 23, 11, 2020. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00382-x*

[17] Guy Bruneau. The History and Evolution of Intrusion Detection. *SANS Institute, Information Security Reading Room, 2021. https://www.sans.org/reading-room/whitepapers/detection/history-evolution-intrusion-detection-344*

Anexo A: Script usado

alldaymerge.py:

```
import pandas as pd
import re
import matplotlib.pyplot as plt
import numpy
import pylab as pss

#header = ['TotalPriority','Success','FalsePositive']
header = ['Success','FalsePositive']
ind=['Prio1', 'Prio2', 'Prio3']
df = pd.DataFrame(columns = header)

nalert1 = 0
nfalso1 = 0

nalert2 = 0
nfalso2 = 0

nalert3 = 0
nfalso3 = 0

totalpo = 0
totalfa = 0
total = 0
found = False
#Tuesday
with  open('/home/ubuntu/Material/Suricata/Tuesday/tuesday.log')
as dat:
    lines = dat.readlines() #f.read().splitlines()
for line in lines:
    found = False
    time = re.split('\.[0-9][0-9][0-9][0-9][0-9][0-9]',line)
    x    =    re.split('\[[0-9]:[0-9][0-9][0-9][0-9][0-9][0-9][0-
9]:[[0-9]+] ',time[1])
    alertmsg = re.split('\[\*\*]\s\[',x[1])
    classi = re.split('] \[Priority:',alertmsg[1])
    prio = re.split('] \{',classi[1])

    if "1" in prio[0]:
        nfalso1 = nfalso1 + 1
        nfalso2 = nfalso2 + 1
        nfalso3 = nfalso3 + 1
    elif "2" in prio[0]:
        nfalso2 = nfalso2 + 1
        nfalso3 = nfalso3 + 1
```

```
    else :
        nfalso3 = nfalso3 + 1
    totalfa = totalfa + 1
total = total + 1


#WEDNESDAY
with
open('/home/ubuntu/Material/Suricata/Wednesday/wednesday.log')
as dat:
    lines = dat.readlines() #f.read().splitlines()
for line in lines:
    found = False
    time = re.split('\.[0-9][0-9][0-9][0-9][0-9][0-9]',line)
    x    =    re.split('\[[0-9]:[0-9][0-9][0-9][0-9][0-9][0-9][0-
9]:[[0-9]+] ',time[1])
    alertmsg = re.split('\[\*\*]\s\[',x[1])
    classi = re.split('] \[Priority:',alertmsg[1])
    prio = re.split('] \{',classi[1])

    if "ET EXPLOIT Possible OpenSSL HeartBleed Large HeartBeat
Response" in alertmsg[0]:
        found = True

    if found:
        if "1" in prio[0]:
            nalert1 = nalert1 + 1
            nalert2 = nalert2 + 1
            nalert3 = nalert3 + 1
        elif "2" in prio[0]:
            nalert2 = nalert2 + 1
            nalert3 = nalert3 + 1
        else :
            nalert3 = nalert3 + 1
        totalpo = totalpo + 1
    else :
        if "1" in prio[0]:
            nfalso1 = nfalso1 + 1
            nfalso2 = nfalso2 + 1
            nfalso3 = nfalso3 + 1
        elif "2" in prio[0]:
            nfalso2 = nfalso2 + 1
            nfalso3 = nfalso3 + 1
        else :
            nfalso3 = nfalso3 + 1
        totalfa = totalfa + 1
    total = total + 1
#THURSDAY
```

```
with
open('/home/ubuntu/Material/Suricata/Thursday/thursday.log')  as
dat:
    lines = dat.readlines() #f.read().splitlines()
for line in lines:
    found = False
    time = re.split('\.[0-9][0-9][0-9][0-9][0-9][0-9]',line)
    x    =    re.split('\[[0-9]:[0-9][0-9][0-9][0-9][0-9][0-9][0-
9]:[[0-9]+] ',time[1])
    alertmsg = re.split('\[\*\*]\s\[',x[1])
    classi = re.split('] \[Priority:',alertmsg[1])
    prio = re.split('] \{',classi[1])

    if "GPL EXPLOIT Microsoft cmd.exe banner" in alertmsg[0]:
        found = True
    elif "ET SCAN" in alertmsg[0]:
        found = True
    elif "ET MALWARE Windows" in alertmsg[0]:
        found = True
    if found:
        if "1" in prio[0]:
            nalert1 = nalert1 + 1
            nalert2 = nalert2 + 1
            nalert3 = nalert3 + 1
        elif "2" in prio[0]:
            nalert2 = nalert2 + 1
            nalert3 = nalert3 + 1
        else :
            nalert3 = nalert3 + 1
        totalpo = totalpo + 1
    else :
        if "1" in prio[0]:
            nfalso1 = nfalso1 + 1
            nfalso2 = nfalso2 + 1
            nfalso3 = nfalso3 + 1
        elif "2" in prio[0]:
            nfalso2 = nfalso2 + 1
            nfalso3 = nfalso3 + 1
        else :
            nfalso3 = nfalso3 + 1
        totalfa = totalfa + 1
    total = total + 1
#Friday
with open('/home/ubuntu/Material/Suricata/Friday/friday.log') as
dat:
    lines = dat.readlines() #f.read().splitlines()
for line in lines:
    found = False
```

```
    time = re.split('\.[0-9][0-9][0-9][0-9][0-9][0-9]',line)
    x    =    re.split('\[[0-9]:[0-9][0-9][0-9][0-9][0-9][0-9][0-
9]:[[0-9]+] ',time[1])
    alertmsg = re.split('\[\*\*]\s\[',x[1])
    classi = re.split('] \[Priority:',alertmsg[1])
    prio = re.split('] \{',classi[1])

    if "ET SCAN Behavioral Unusual Port" in alertmsg[0]:
        found = True

    if found:
        if "1" in prio[0]:
            nalert1 = nalert1 + 1
            nalert2 = nalert2 + 1
            nalert3 = nalert3 + 1
        elif "2" in prio[0]:
            nalert2 = nalert2 + 1
            nalert3 = nalert3 + 1
        else :
            nalert3 = nalert3 + 1
        totalpo = totalpo + 1
    else :
        if "1" in prio[0]:
            nfalso1 = nfalso1 + 1
            nfalso2 = nfalso2 + 1
            nfalso3 = nfalso3 + 1
        elif "2" in prio[0]:
            nfalso2 = nfalso2 + 1
            nfalso3 = nfalso3 + 1
        else :
            nfalso3 = nfalso3 + 1
        totalfa = totalfa + 1
    total = total + 1

dfp = pd.DataFrame(columns = header, index = ['Priority'])
df1 = pd.DataFrame([[nalert1,nfalso1]], columns = header, index =
["1"])
df2 = pd.DataFrame([[nalert2,nfalso2]], columns = header, index =
["2"])
df3 = pd.DataFrame([[nalert3,nfalso3]], columns = header, index =
["3"])
df = df.append(dfp)
df = df.append(df1)
df = df.append(df2)
df = df.append(df3)
df.to_csv('/home/ubuntu/Material/Suricata/totaltestduricata.csv'
)
```