UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

SISTEMA DE MONITORIZACIÓN DEL IDS SNORT

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: José Llopis Polvoreda

Tutor: Francisco Jose Abad Cerdá

2016/2017

Resumen

Durante los últimos años, los ataques cibernéticos han sufrido un importante incremento, llegando a generar pérdidas millonarias a las organizaciones afectadas. A su vez, las técnicas de ataque han aumentado en diversidad, además de ser cada vez más sofisticadas e inteligentes.

Una buena alternativa para mejorar la seguridad de nuestros datos, consiste en la instalación de IDSes o Sistemas de Detección de Intrusos, cuya principal función es la de detectar accesos no deseados a nuestra red o comportamientos anómalos en el interior de la misma.

En este proyecto se va estudiar el funcionamiento del IDS Snort, un sistema de detección de intrusos basado en red el cual detecta posibles ataques analizando cada paquete del tráfico de la red con un motor de reglas de ataques conocidos.

El principal problema de los IDS, y motivación de este trabajo, es que cuando se encuentran con una cantidad de tráfico tal que no pueden mantener el ritmo de análisis en tiempo real de los paquetes, empiezan a dejar pasar una parte de ellos sin analizar. Por tanto, un pico de tráfico inusual puede usarse para enmascarar un ataque. Mantener un rendimiento óptimo del sistema IDS, es por tanto, una tarea fundamental si se quiere detectar la totalidad de los ataques y que ninguno pase desapercibido.

Así pues, el objetivo de este trabajo será el de realizar un servidor centralizado que reciba datos estadísticos del IDS Snort y que estos se muestren en una interfaz gráfica para simplificar el estudio del rendimiento, con el fin de optimizar el funcionamiento del sistema IDS.

Palabras clave: ataques cibernéticos, seguridad, IDS, Snort, rendimiento.

Abstract

In recent years, cyber-attacks have experienced a significant increase, generating millions in losses to the organizations concerned. In turn, attack techniques have increased in diversity, being more sophisticated and intelligent.

A good alternative to improve the security of our data would be the installation of IDS or Intrusion Detection Systems, whose main function is to detect unwanted access or anomalous behavior in our network.

This project will study the operation of IDS Snort, an intrusion detection system based on network which detects possible attacks by analyzing each packet of network traffic with a rules engine of known attacks.

The main problem of IDS systems, and motivation of this work, is that when they work with a quantity of traffic that they can not maintain the rhythm of the real-time analysis of the packages, they begin to let some of them go unanalyzed. Therefore, an unusual traffic peak can be used to mask a cyber-attack. Maintaining optimum performance of the IDS system is therefore a fundamental task if you want to detect all attacks and not go unnoticed.

Thus, the aim of this work is to create a centralized server which receives statistical data from the Snort IDS and this data is shown in a graphical interface to simplify the study of the performance, in order to optimize the operation of the IDS system.

Keywords: cyber-attack, security, IDS, Snort, performance.

Tabla de contenidos

Lista de figuras

1. Introducción

1.1. Marco de estudio y justificación

Las propiedades de gran valor necesitan ser protegidas, tanto de posibles robos como de destrucción de las mismas. Es por esto que se instalan alarmas en hogares y en vehículos.

En el momento en que la alarma detecta alguna acción sospechosa, puede avisar en tiempo real a los propietarios y a la policía para que los daños sean los menores posibles.

La misma protección debería de ser aplicada en los sistemas informáticos, y con ello, a los datos, ya que hoy en día, es una información muy valiosa, sobre todo para las empresas.

Por esta razón, se deberían de instalar unas "alarmas" en las redes y en el momento en que se detecte alguna acción sospechosa, avisar en tiempo real para poder detenerla, con el fin de mantener los sistemas seguros y libres de intrusiones. Esto sería la función de los IDS: Intrusion Detection System, o Sistemas de Detección de Intrusos.

Más concretamente, un IDS es un programa de detección de accesos no autorizados a un host o a una red. El funcionamiento de esta herramienta se basa en el análisis pormenorizado del tráfico de red, el cual se compara con una base de firmas de ataques conocidos o comportamientos sospechosos, como puede ser el escaneo de puertos, paquetes malformados, etc. Además, también revisa el contenido de los paquetes y su comportamiento.

En la actualidad, por lo que se está observando, cada vez son más comunes los ataques a toda organización de los cuales se pueda sacar beneficio. El caso más reciente y que afectó a grandes compañías como Telefónica, la red de hospitales británica o el ministerio del Interior ruso, es el famoso *ransomware* llamado "Wanna Cry", traducido al español "Quiero Llorar".

El vector de entrada de este virus era mediante correos de entrada y archivos adjuntos maliciosos, los cuales engañaban al usuario para que se descargara el fichero anexado y lo ejecutara. Una vez infectado, Wanna Cry cifraba todos los datos del equipo afectado y se expandía de manera lateral por toda la red, aprovechando una vulnerabilidad de los sistemas operativos Windows, infectando así otros equipos a su alcance y una vez infectados, repetían el mismo procedimiento pidiendo un rescate de 300 dólares por cada equipo afectado si se querían recuperar los datos cifrados.

Con un IDS, con las firmas actualizadas de las últimas vulnerabilidades, analizando el tráfico de la red, se hubiera podido detectar los intentos de explotación de dicha vulnerabilidad y avisar a tiempo para que el impacto del ataque sobre la organización fuese el menor posible.

Este proyecto se ha pensado con la finalidad de crear una aplicación web para la monitorización de un sistema IDS Snort. A continuación, se puede observar un esquema de la infraestructura en la que se apoyaría el proyecto:

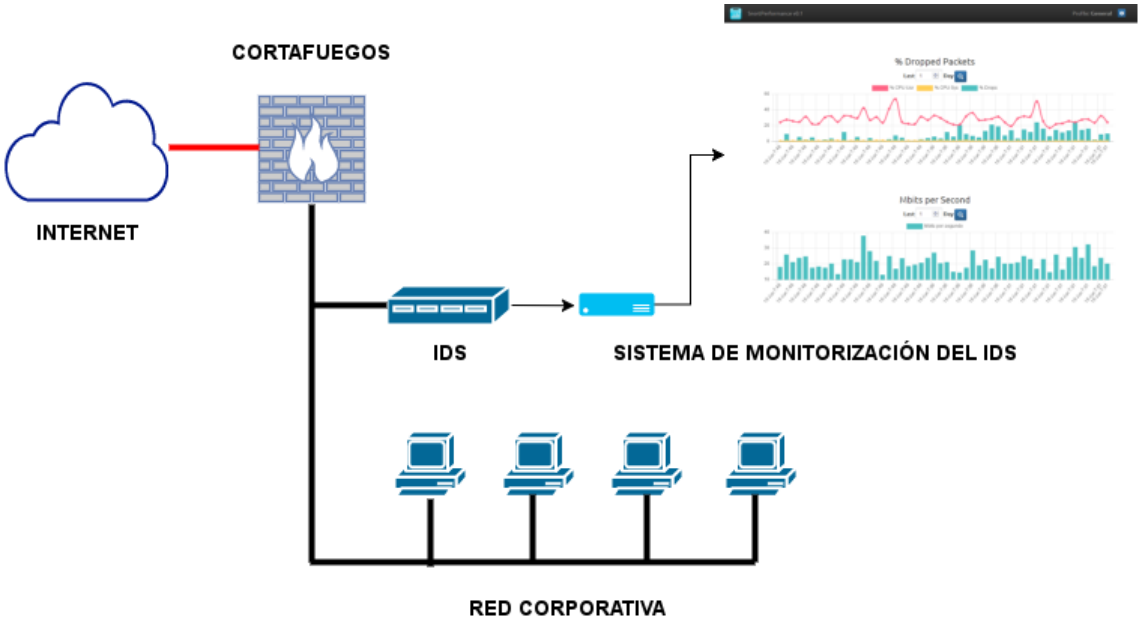


Ilustración 1. Esquema proyecto

Como se muestra en la imagen anterior, el objetivo del proyecto es recoger datos estadísticos del IDS y enviarlos a la herramienta diseñada, los cuales serán tratados y mostrados en una aplicación web para monitorizar el rendimiento del sistema de detección de intrusos Snort instalado en la organización.

El presente proyecto se ha realizado en la empresa valenciana S2 Grupo, entidad con una larga experiencia y conocimientos en la seguridad informática. Este Trabajo de Final de Grado pretende resolver un problema que surgió cuando se comprueba, en un determinado servidor, que el IDS Snort no genera todas las alertas que debería debido a que está dejando pasar paquetes sin analizar debido a un problema de rendimiento. Otra motivación de este trabajo es un gran interés por adquirir la mayor cantidad de conocimientos posibles sobre ciber seguridad.

A lo largo de la memoria, se harán uso de palabras técnicas y anglicismos, el significado de las cuales se podrá buscar en un breve glosario al final de la misma.

1.2. Objetivos del trabajo

A continuación se enumeran los objetivos principales del presente Trabajo Final de Grado:

1. Entender el concepto de un IDS.
2. Comprender el funcionamiento del IDS Snort.
3. Configurar Snort para que cree archivos con datos estadísticos generales del IDS, de las reglas y de los preprocesadores.

4. Crear agentes (programas) que *parseen* los datos estadísticos generados por Snort y envíen éstos a la herramienta diseñada, con el formato adecuado.
5. Diseñar una aplicación web que muestre de manera clara e intuitiva los datos sobre el rendimiento del sistema de detección de intrusos, haciendo uso de diferentes tipos de gráficas.

El objetivo final del trabajo es disponer de un servidor que recibirá datos estadísticos del IDS, con el fin de analizar su rendimiento de una manera rápida y cómoda.

1.3. Enfoque y método seguido

- Identificación del problema

Actualmente, la mayoría de los sistemas IDS instalados en las organizaciones, no disponen de sistemas que monitoricen el rendimiento de los mismos, a pesar de que estos sí ofrezcan la posibilidad de generar datos estadísticos sobre su funcionamiento.

En ocasiones, puede ocurrir que el IDS no genere alertas de posibles ataques y se crea que la razón de esto, es que nadie lo esté intentando y que los sistemas de la organización estén a salvo, pero que la verdadera razón de este comportamiento sea que el sistema de detección de intrusos no esté analizando todos los paquetes, llegando a analizar sólo una minoría debido a la gran cantidad de tráfico que pueda haber en una organización.

- Determinar los requerimientos de las organizaciones

Dicho lo anterior, si las organizaciones dispusieran de sistemas de monitorización de sus IDSes, en este caso Snort, ganarían un mayor control sobre sus sistemas de detección de intrusos y podrían optimizar estos con la finalidad de mejorar su funcionamiento y aumentar la seguridad de sus redes.

- Diseño de la herramienta

Así pues, se procederá a configurar Snort para que genere datos estadísticos sobre su rendimiento, y a enviar éstos a la herramienta a diseñar para que muestre estos valores al usuario, mediante una aplicación web que ayude a los técnicos a estudiar el funcionamiento de los sistemas IDSes.

- Documentación de la herramienta

Para la realización del servidor, se hará uso de varios lenguajes de programación: para la parte del *backend*, se utilizará GoLang, y para la parte del frontend, se hará uso de HTML, CSS y JavaScript. Por otra parte, los programas o agentes encargados de analizar y enviar los datos estadísticos a la aplicación, se desarrollarán en Python.

Finalmente, para almacenar todos los datos, se utilizará una base de datos MongoDB.

- Pruebas de la herramienta

Una vez terminada la herramienta, se realizarán una serie de pruebas para comprobar que la aplicación desarrollada cumple con sus objetivos.

1.4. Breve resumen de los siguientes capítulos de la memoria

A continuación se describe el contenido del resto de capítulos de la memoria:

- Estado del Arte: En este capítulo se estudiarán los sistemas de detección de intrusos y su funcionamiento, así como también las herramientas existentes con una función similar a la de este proyecto.
- Análisis del Problema: En esta parte, se plantea el problema principal del rendimiento de los IDSes y la solución propuesta para solventarlo.
- Diseño de la Solución: En este módulo se detallan las tareas de diseño de la herramienta propuesta, explicando los lenguajes de programación utilizados y las librerías más relevantes.
- Implementación: En esta sección se presenta el prototipo desarrollado que actuará de sistema de monitorización del rendimiento del IDS Snort.
- Pruebas con Datos Reales: Para comprobar que la herramienta diseñada sería aplicable en las organizaciones, se realiza una prueba con datos reales.
- Trabajos futuros: En esta parte final, se plantearán unas posibles mejoras para la aplicación desarrollada.
- Conclusión.

2. Estado del Arte

Antes de analizar las aplicaciones existentes con funcionalidad similar a la que se propone en el proyecto, se explicarán los sistemas principales a partir de los cuales se han realizado las aplicaciones anteriores.

2.1. Sistema de Detección de Intrusos

Hoy en día, "la Información" es considerada el activo más importante de las empresas. Si la información de una organización llegara a desaparecer o cayera en manos de la competencia, tendría un impacto crítico para la organización. Por esta razón, mantener a salvo "la Información" es un objetivo fundamental en cualquier empresa.

La seguridad de la información se define como el conjunto de medidas preventivas y reactivas de las organizaciones con el objetivo de resguardar y proteger la información, buscando la preservación de su confidencialidad, disponibilidad e integridad.

Alguna de estas medidas preventivas sería la instalación de sistemas de detección de intrusiones, considerando una intrusión, el conjunto de acciones que intentan comprometer la integridad, confidencialidad o disponibilidad de un recurso.

Los dos grandes sistemas de detección de intrusiones son los IDSes (Intrusion Detection System) y los IPSes (Intrusion Prevention System). El presente proyecto se centrará en los sistemas IDS.

Un sistema de detección de intrusos o IDS es un mecanismo de seguridad que lleva a cabo el análisis automático de parámetros que modelan la actividad de un entorno con el propósito de detectar intrusiones. Cuando se identifica una condición anómala, el IDS emite una alerta avisando de la posible intrusión con datos relacionados de la misma. Por ejemplo, en el IDS Snort, que se explicará más adelante, se puede configurar para que muestre información ampliada de las alertas que detecte, mostrando para cada alerta los siguientes datos: tiempo, mensaje de la alerta, clasificación, prioridad de la alerta, IP y puerto de origen/destino e información completa de las cabeceras de los paquetes registrados:

```
$ snort -A full -dev -l ./log -h 192.168.4.0/24 -c ../etc/snort.conf

[**] [1:620:2] SCAN Proxy (8080) attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
23/07/17-14:53:38.481065 192.168.4.3:3159 -> 192.168.4.15:8080
TCP TTL:128 TOS:0x0 ID:39918 IpLen:20 DgmLen:48 DF
******S* Seq: 0xE87CBBAD Ack: 0x0 Win: 0x4000 TcpLen: 28
TCP Options (4) => MSS: 1456 NOP NOP SackOK
```

Como se puede observar, la alerta anterior se ha generado porqué el día 23 de Julio de 2017, a las 14:53 horas, la dirección IP 192.168.4.3, a través del puerto 3159, realizó una

conexión hacia el puerto 8080 del proxy configurado en la red, en este caso, con dirección IP 192.168.4.15. Se tendría que analizar si dicha conexión es legítima y está permitida.

Por otro lado, el IPS, o sistema de prevención de intrusiones, tiene como misión realizar respuestas de contención contra posibles intrusiones, bloqueando los ataques detectados. Como ya se ha dicho anteriormente, este tipo de software no entra en el alcance del trabajo.

2.1.1. Funcionamiento y características

El funcionamiento de estas herramientas de detección de intrusos se basa en el análisis pormenorizado del tráfico de red, el cual al entrar al analizador se compara con firmas de ataques ya conocidos, o comportamientos sospechosos, como por ejemplo el escaneo de puertos, paquetes malformados o anómalos, incumplimiento de políticas de seguridad, etc. Los IDSes no sólo se encargan de analizar qué tipo de tráfico es, sino también del contenido de los paquetes y su comportamiento. Esta última característica permitiría detectar ataques nuevos contra los sistemas.

Suponiendo que en las organizaciones se tiene un cortafuegos o *firewall* en la red corporativa, los IDSes se pueden colocar en diferentes sitios:

1. Antes del firewall: se analizaría todo el tráfico, tanto entrante como saliente de la red, generando un número muy elevado de falsos positivos.
2. Después del firewall: se analizaría todo el tráfico, una vez pasado el cortafuegos, disminuyendo la cantidad de falsos positivos.
3. Instalando un IDS previo al firewall y otro después de éste: en este caso se unen las ventajas de las dos alternativas anteriores, pero también existe una gran desventaja, ya que sería una opción muy costosa de controlar.
4. En el propio Firewall: obteniendo un sistema, el cual, aparte de detectar el tráfico de red, también lo analiza.

NOTA: No hace falta disponer de un cortafuegos para la utilización de un IDS. Se han comentado las alternativas anteriores ya que el uso de los IDS es más habitual en las organizaciones cuya información es más valiosa para los intrusos, y el uso de una firewall se hace indispensable para evitar las conexiones no autorizadas a un sistema interno desde el exterior.

En la siguiente ilustración se muestra la alternativa de colocar un IDS previo al firewall y otro posterior:

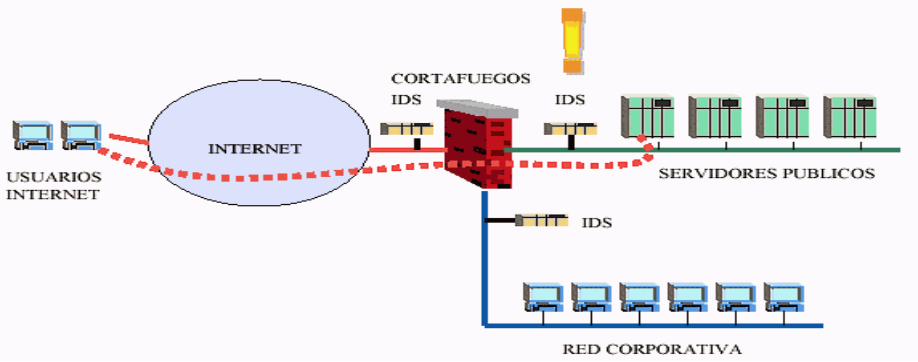


Ilustración 2. Ubicación de IDS antes y después de un Firewall y en varios segmentos de red

En la figura anterior se puede observar la ubicación del IDS antes del cortafuegos y después de él en dos segmentos de red diferentes. En el primer IDS, se analizaría todo el tráfico, lo cual generaría una gran cantidad de falsos positivos. En los IDS de después del *firewall*, se configurarían para que cada uno analizara los segmentos de red deseados.

Esta composición de IDSes es la más interesante ya que se dispondría de todos los intentos de intrusión, tanto desde el exterior como del interior de nuestra red, pudiendo comprobar, comparando las alertas del IDS anterior al *firewall* con los otros, si los ataques han sido detenidos o no por el *firewall*.

Por otro lado, como se ha dicho anteriormente, también sería la composición más costosa debido al despliegue de los IDSes, su configuración en cada segmento y por último y la más importante, el análisis de todas las alertas.

2.1.2. Tipos de IDS

Existen diferentes tipos de IDSes, clasificados según su situación física, forma en la que detectan las intrusiones y según su reacción al detectar un posible ataque:

- Clasificación por situación

  - HIDS (Host IDS): es un IDS particular para un servidor, sólo procesa los datos asociados a un recurso. El IDS intentará detectar los rastros dejados por los intrusos en el equipo atacado.
  - NIDS (Network IDS): es un IDS basado en red, procesa datos asociados a varios recursos. No tienen por qué analizar el tráfico de toda la red, en él se pueden indicar las redes o subredes a analizar. Casi ningún NIDS se configura para que analice toda la red. Este tipo de IDS es, en la actualidad, el más utilizado.

- Clasificación según la técnica de análisis

  - Detección de usos anómalos: este tipo de detección se apoya en comprender cuál es el tráfico "normal" de la red, para generar alertas cuando detecte tráfico fuera de lo "normal". Un ejemplo claro de este tipo sería, observar tráfico muy elevado dentro de una red fuera del horario laboral, por ejemplo a las 3:00 AM. La modelización del comportamiento es muy compleja, para ello se utilizan sistemas expertos o aprendizaje automático, como por ejemplo, redes neuronales, reconocimiento de patrones geométricos, etc. No muy utilizado en sistemas reales.
  - Detección de usos indebidos: por otro lado, en este tipo de detección, no se conoce lo que es "normal" en un sistema, sino que se conoce la actividad "anormal" o ataques hasta la fecha conocidos, de tal manera que cuando se detecte un ataque conocido se creará una alerta. La aproximación más habitual es el uso de *pattern matching* o búsqueda de patrones, cada intrusión tiene un patrón o firma asociado a ella, que serán interpretados por el IDS.

- Clasificación según su naturaleza: los IDS también se pueden clasificar según su reacción ante un posible ataque. Las respuestas del IDS pueden ser:

  - Respuestas Pasivas: se detecta un posible ataque o violación de la seguridad, se registra la información detectada del ataque y se genera una alerta de la

posible intrusión.

- Respuestas Reactivas: el IDS es capaz de iniciar una respuesta automática ante una actividad ilegal, por ejemplo, si se detecta un acceso de un usuario no autorizado, es capaz de sacar a dicho usuario del sistema, o si se detectase un ataque de criticidad alta desde una IP hostil, sería capaz de configurar el cortafuegos filtrando dicha IP origen.

2.2. SNORT

Snort[1] un *sniffer* de paquetes y un sistema de detección de intrusos basado en red o NIDS. Implementa un motor de detección de ataques el cual permite registrar, alertar y responder ante cualquier anomalía previamente definida. Estos registros quedan almacenados en formato binario el cual se puede convertir a formato PCAP[2] u otros formatos más legibles. Además también se puede guardar en bases de datos como es MySQL.

Así mismo, existen herramientas complementarias a Snort que hacen que este IDS sea un sistema muy completo y fácil de administrar. Como por ejemplo, herramientas que almacenan las alertas detectadas por Snort en una base de datos (Barnyard2) y otras que recogen de esta base de datos las alertas y las muestran en una interfaz gráfica de fácil manejo (BASE). A su vez, también existen otros programas complementarios para mostrar informes en tiempo real (ACID) o para convertir a Snort, además de IDS, en IPS.

Snort implementa un lenguaje de creación de reglas flexible, potente y sencillo, pudiendo crear todas las alertas que se requiera. Un usuario puede crear una regla y compartirla a través de Internet para que todos los demás usuarios de Snort se puedan beneficiar de esta firma. Existen grandes comunidades[3] las cuales nos ofrecen gran cantidad de reglas de ataques y conexiones sospechosas que podemos incluir en nuestro IDS de manera gratuita, aunque también hay conjuntos de reglas de pago. Este sistema de compartición de conocimiento contra ataques hace que Snort sea un sistema para detectar cualquier tipo de ataque.

Snort, es un software gratuito, bajo licencia GPL y puede ser instalado tanto en sistemas operativos Windows como en sistemas UNIX/Linux. Además, es un sistema probado y fiable y que cuenta con un gran soporte y actualizaciones conforme se van descubriendo nuevas vulnerabilidades a través de los distintos boletines de seguridad.

2.2.1. Funcionamiento

Snort puede funcionar en los siguientes tres modos:

- Modo Sniffer: se captura el tráfico en tiempo real de la red configurada en el archivo de configuración de Snort y se imprime por pantalla.
- Modo registro de paquetes: se guardan los paquetes de la red configurada, en el archivo de configuración de Snort, en ficheros con un determinado formato para su posterior análisis. Se puede volver a reproducir el tráfico almacenado en los ficheros.

---

[1] Página oficial del proyecto de Snort: https://www.snort.org/

[2] El formato PCAP se usa para archivos que contienen la información de los paquetes del tráfico capturado.

[3] Por ejemplo: Emerging Threats (https://rules.emergingthreats.net/) o las propias de Snort (https://www.snort.org/downloads)

- Modo NIDS: se comparan las tramas de todos los paquetes con el conjunto de reglas o patrones que se tengan configurados, mostrando por pantalla las coincidencias o almacenándose estas en un sistema basado en registros.

Una vez instalado, se deberán agregar los conjuntos de reglas que se quieran utilizar para detectar las actividades sospechosas que afecten en la red. Se puede pensar que agregando todos los conjuntos de reglas que existan, se detectarán más ataques, pero la realidad es que no. Cuantas más reglas se añadan, más se sobrecarga el programa, pudiendo llegar a tasas de pérdida de paquetes no analizados muy elevadas, con lo que habrá ataques que el IDS no detecte.

El siguiente cuadro muestra un ejemplo de una regla Snort:

```
alert tcp $EXTERNAL_NET any → $HOME_NET 80 (msg:"Alguien ha intentado
acceder a /etc/passwd"; content:"/etc/passwd";)
```

Estas reglas se componen de dos partes diferenciadas:

- Cabecera: en esta parte se indica quién activa la regla:

```
alert tcp $EXTERNAL_NET any → $HOME_NET 80
```

En este ejemplo, se generará una alerta cada vez que alguien desde una IP externa, desde cualquier puerto, realice una petición al puerto 80 de una IP interna.

- Opciones de la regla: en esta segunda parte se indica qué activa la regla:

```
(msg:"Alguien ha intentado acceder a /etc/passwd";
content:"/etc/passwd";)
```

La segunda parte de la regla indica que para que se active, en la petición debe aparecer "/etc/passwd", lo cual sería algo sospechoso ya que el recurso anterior es un recurso privado.

Para comprender mejor este software, en el ANEXO se puede encontrar una guía de instalación de Snort paso por paso. En ella se detallan las configuraciones y librerías necesarias para el correcto funcionamiento del IDS Snort.

2.2.2. Módulos y componentes

Snort es un sistema modular compuesto por diferentes partes, cada una encargada de funciones independientes que dotan a este IDS de mayor dinamismo e interoperabilidad.

El motor de Snort se divide en los siguientes módulos:

1. Interfaz de red
2. Método de captura
3. Filtro BPF
4. Preprocesadores
5. Motor de reglas
6. Filtrado de eventos
7. Salida

En primera instancia, se le debe indicar a Snort en qué interfaz, o interfaces debe analizar el tráfico y con qué métodos (DAQ, AF-PACKET, PF_RING, NFQ, otros). Posteriormente se tendrá que aplicar un filtro BPF (Berkeley Packet Filter) o filtrado de paquetes para cada interfaz configurada, ya que cada proceso separado tiene una configuración diferente.

Llegado a este punto, los preprocesadores arreglan, rearman o modifican los datos para dejarlos preparados y que posteriormente serán analizados en busca de intrusiones por el motor de reglas. Algunos preprocesadores también pueden realizar la función de detección de amenazas buscando anomalías en las cabeceras de los paquetes y generando alertas en caso de encontrar algo sospechoso.

A continuación se detallan algunos de los preprocesadores más importantes:

- frag3: encargado de ensamblar paquetes fragmentados. Necesario para un correcto análisis posterior.
- stream5: encargado de ensamblar paquetes. Gestiona sesiones TCP, UDP e ICMP. Además es capaz de detectar diferentes anomalías en los paquetes.
- http_inspect: encargado de la extracción de datos HTTP como por ejemplo: header, URI, host, etc.

Una vez preprocesados los paquetes, estos pasan a ser analizados por el motor de reglas predefinido, cuya función es detectar posibles intrusiones. A cada paquete analizado se le aplican todas las reglas cargadas. Las reglas que coincidan en los paquetes ejecutarán su acción asociada.

Al acabar este módulo, entrará en acción el filtrado de eventos, el cual, según la configuración indicada, generará alertas dependiendo de las veces que una regla haya coincidido con el contenido de un paquete.

Por último, dependiendo de qué detecte el motor dentro de un paquete, el sistema de alertas generará o no una alerta. Los registros de las alertas generadas se pueden almacenar en archivos de texto o en formato binario, muy rápido y ligero.

Dependiendo de los complementos de salida que se tengan configurados, una alerta tendrá una salida u otra, por ejemplo vía email, alarmas (o traps) SNMP, syslog, inserción en una base de datos, etc.

2.3. Estado del Arte

Una vez explicado el sistema IDS Snort, sobre el que se ha desarrollado la herramienta de monitorización del presente proyecto, se analizarán las aplicaciones de igual o parecidas funcionalidades que existen.

Existen múltiples aplicaciones complementarias al IDS Snort, como por ejemplo:

- SAM[4] (Snort Alert Monitor): herramienta para la monitorización de las alertas en tiempo real del IDS Snort. Posee varias alternativas para mostrar las intrusiones detectadas por Snort, además de mostrar información de estas con la ayuda de gráficos muy representativos.

[4] https://sourceforge.net/projects/snortalertmon/

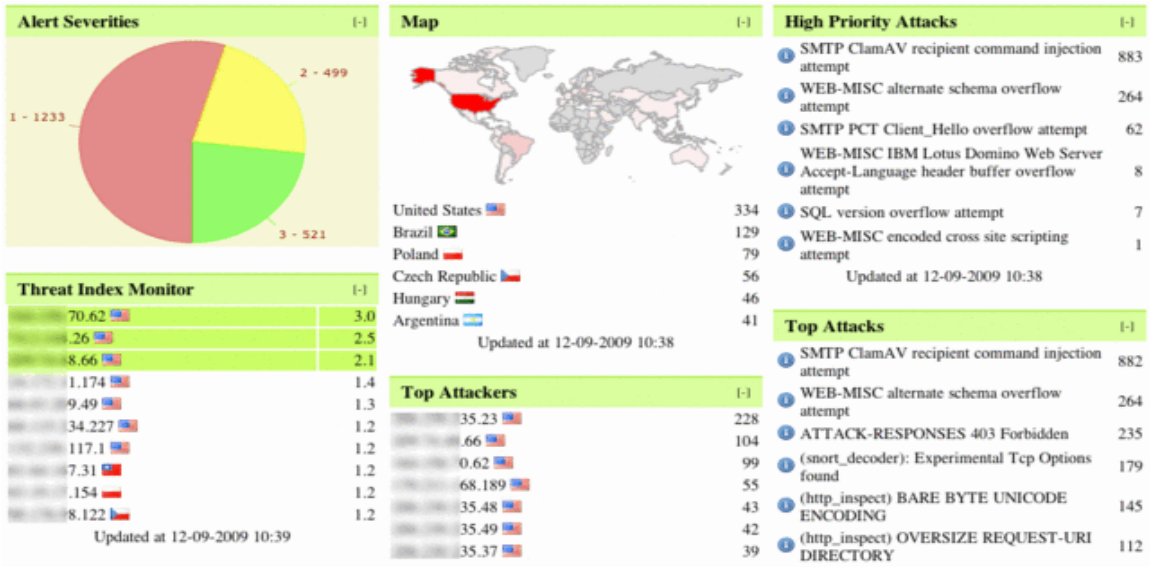


Ilustración 3. Interfaz de SAM (Snort Alert Monitor)

- Scanmap3d[5]: esta aplicación permite, a partir de la información generada por Snort y guardada en una base de datos MySQL, producir mapas en 3D del tráfico de la red para visualizar la actividad de las posibles intrusiones y facilitar el análisis de estos incidentes.

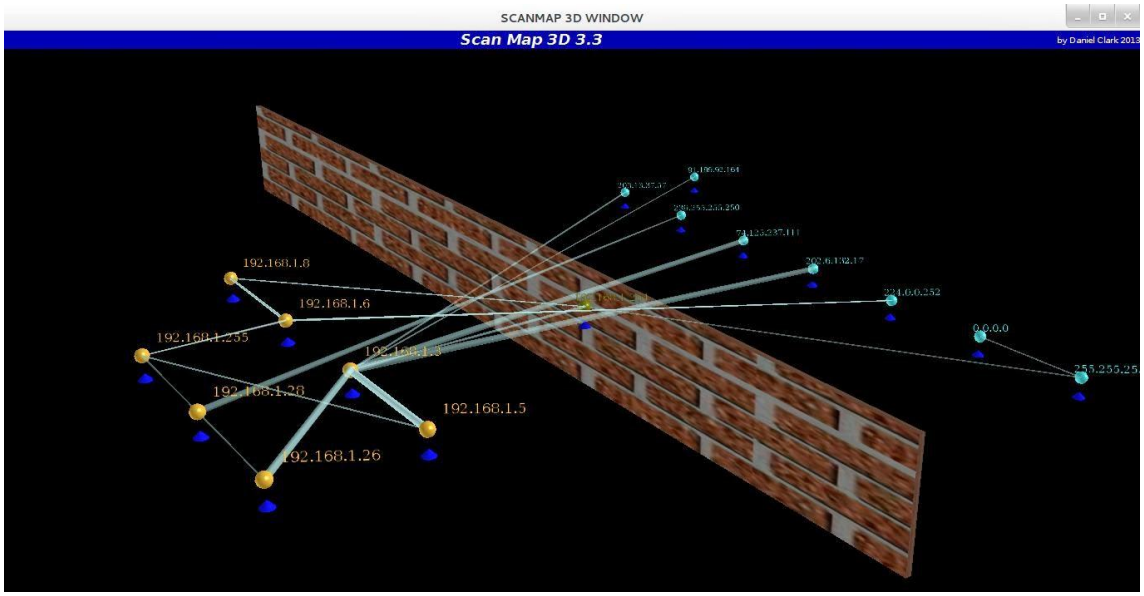


Ilustración 4. Interfaz de Scanmap3d

- Cyberprobe[6]: el objetivo de este programa es la monitorización de redes contra ciberataques. Esta aplicación, se puede integrar con el IDS Snort y contrastar, de manera gráfica y clara, los datos generados por las alertas de Snort con el tráfico de red.

[5] http://scanmap3d.sourceforge.net/
[6] https://sourceforge.net/projects/cyberprobe/?source=directory

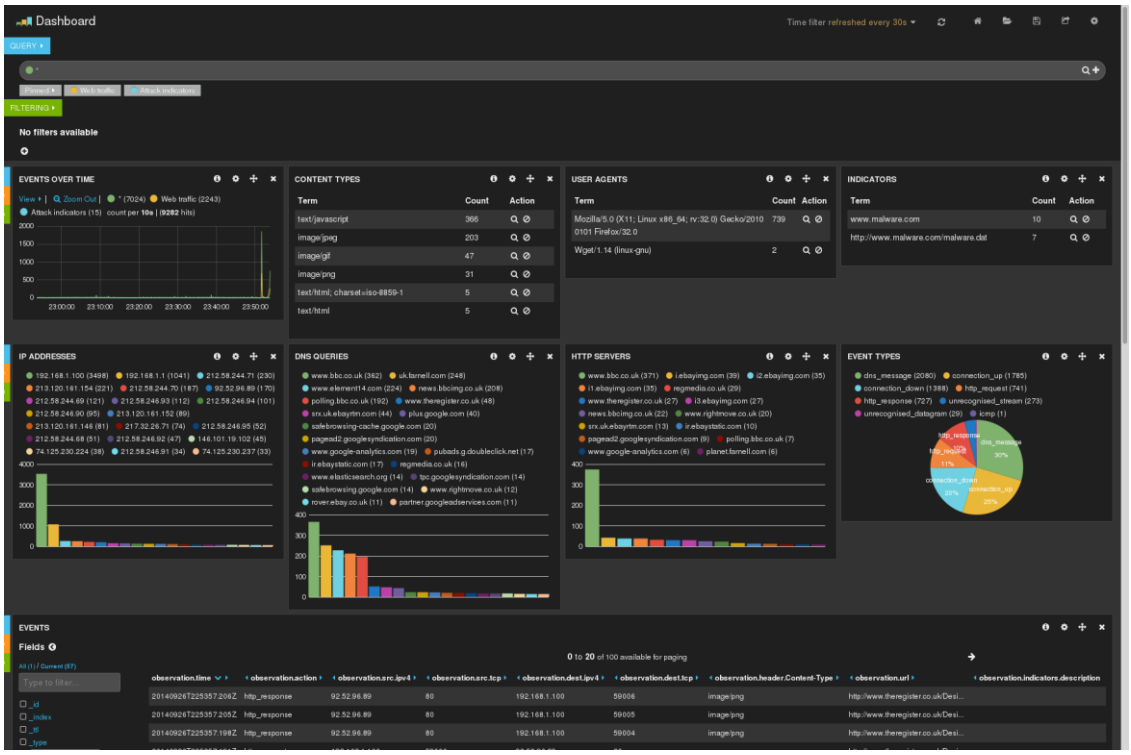


Ilustración 5. Interfaz Cyberprobe

- Nebula[7]: es una herramienta que, a través de ataques realizados a un honeypot, genera de manera automática firmas para Snort de ese mismo ataque.

Pero sin duda, los dos complementos más utilizados de Snort, que hacen que este IDS sea un sistema muy completo y fácil de administrar son:

- Barnyard2[8]: un intérprete de código abierto para los archivos binarios unified2 de salida de Snort de las alertas detectadas.

  Su principal objetivo es permitir que Snort almacene en disco las intrusiones detectadas de manera eficiente y dejar la tarea de convertir cada binario en diferentes formatos, en un proceso separado para no cargar el servicio de Snort.

- BASE[9] (Basic Analysis and Security Engine): se basa en el código del proyecto Analysis Console for Intrusion Databases (ACID). Esta aplicación es complementaria a la anterior y recoge las alertas que se almacenan en la base de datos proporcionando una interfaz web para poder consultar y analizar las alertas generadas por Snort de manera gráfica.

[7] https://www.honeynet.org/project/Nebula
[8] https://github.com/firnsy/barnyard2
[9] https://sourceforge.net/projects/secureideas/

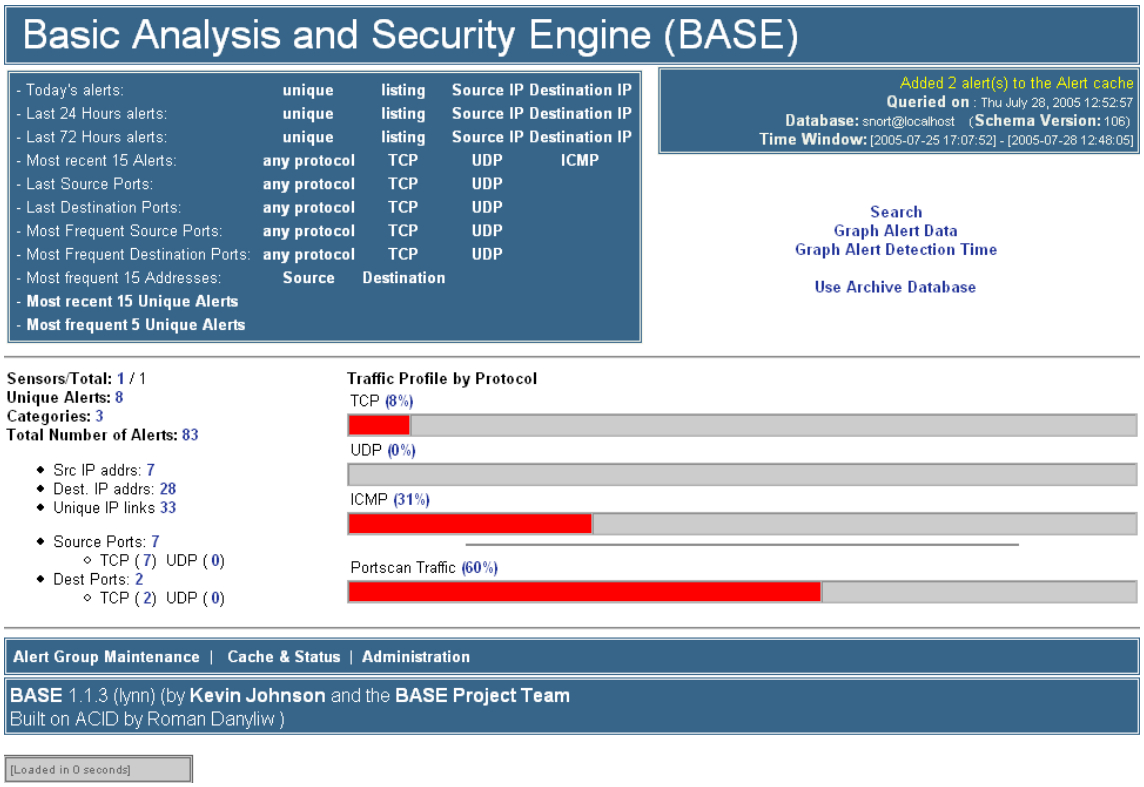


Ilustración 6. Interfaz BASE

Sin embargo, herramientas públicas que se centren en el rendimiento del IDS Snort, que es el objetivo principal del presente proyecto, sólo se tiene constancia del programa PMGraph.

PMGraph: es un script escrito en el lenguaje de programación Perl y desarrollado por Andreas Ostling, el cual genera gráficas del rendimiento del IDS Snort. La información que se ofrece en estas gráficas es: paquetes descartados, alertas por segundo, Mbit por segundo, Kpaquetes por segundo, paquetes SYN + SYN/ACK por segundo, sesiones por segundo, sesiones abiertas, flujo de eventos por segundo, eventos de tipo frag por segundo, promedio de bytes por paquete y estadísticas de uso de la CPU.

En la página siguiente se pueden observar las gráficas generadas por la herramienta PMGraph:

Graphs generated: 2017-05-23 18:55:03 (CEST)
First entry in input file: 2017-05-16 18:27:06 (CEST)
Last entry in input file: 2017-05-23 18:51:29 (CEST)
Showing last 24 hours

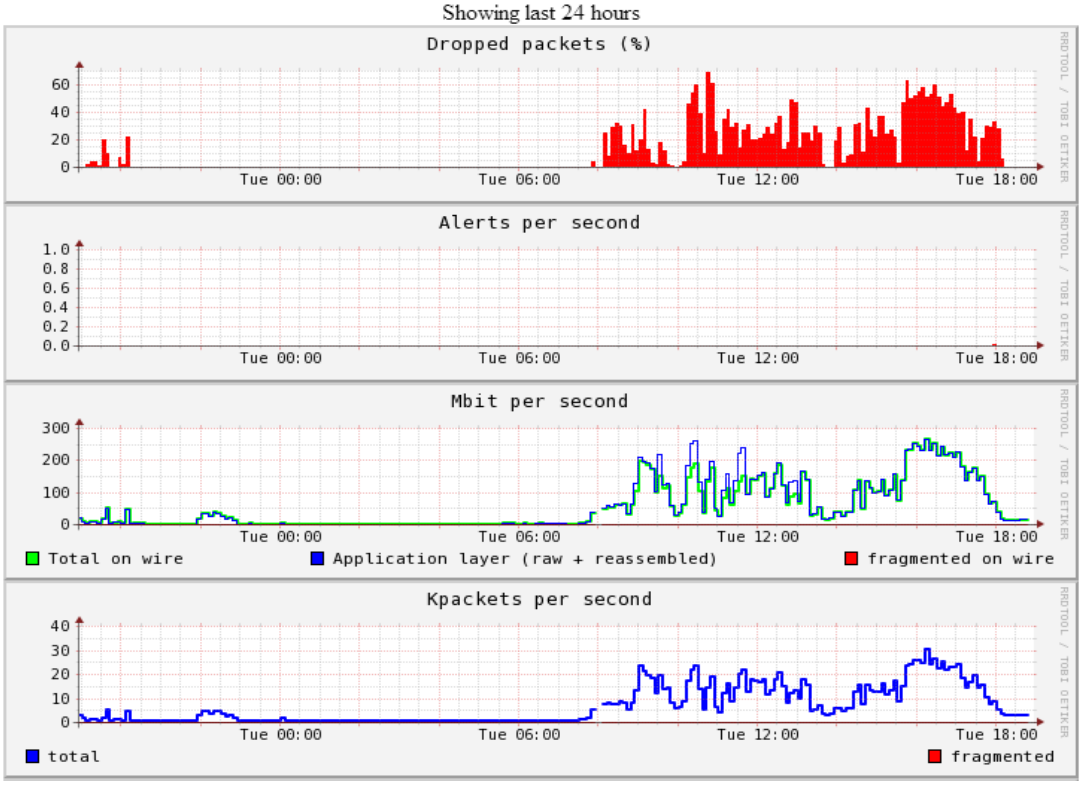


Ilustración 7. Gráficas 1 PMGraph

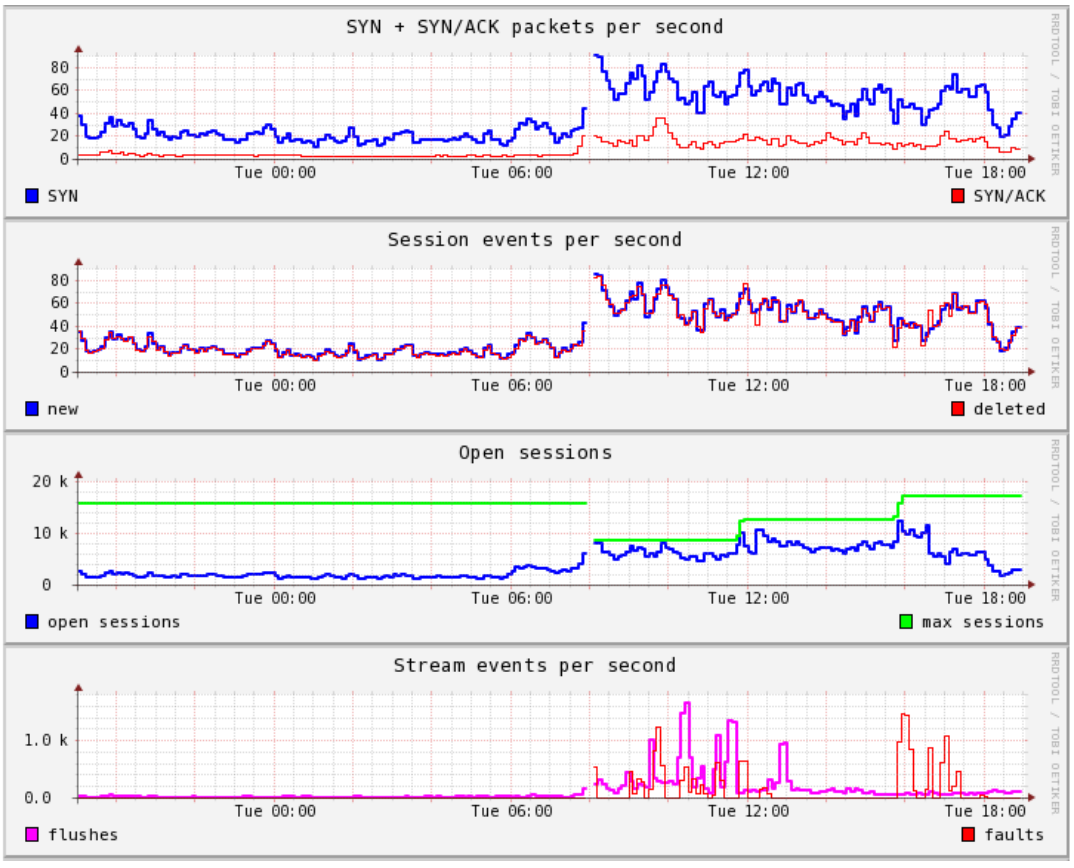


Ilustración 8. Gráficas 2 PMGraph

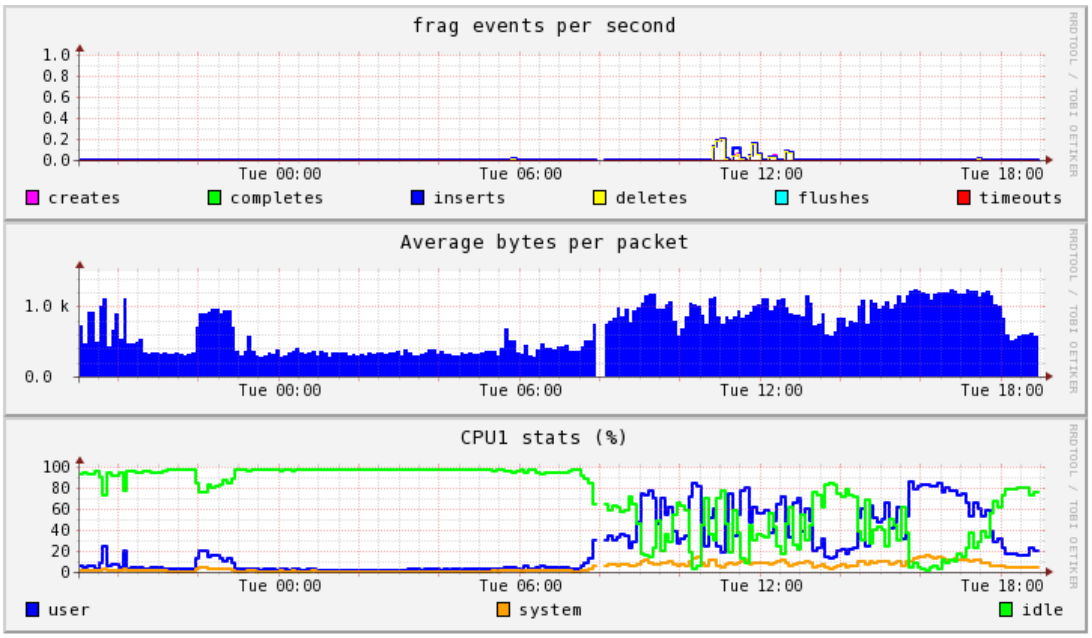


Ilustración 9. Gráficas 3 PMGraph

Nota: Se ha dividido la interfaz de PMGraph en 3 ilustraciones para una mejor visibilidad del contenido. En la aplicación web, todas las gráficas se muestran en la misma página como se puede ver a continuación:

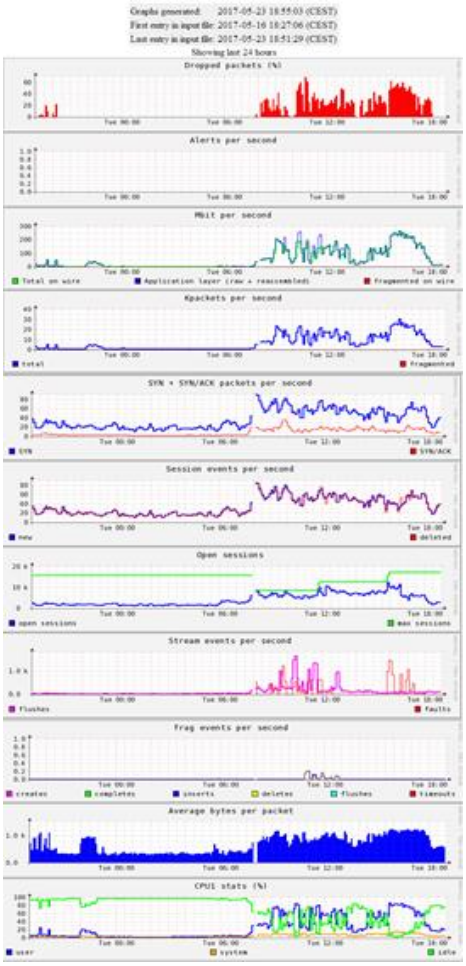


Ilustración 10. PMGraph

Además, si se hace click sobre cualquier gráfica, se puede hacer zoom sobre los datos y mostrar más valores históricos:

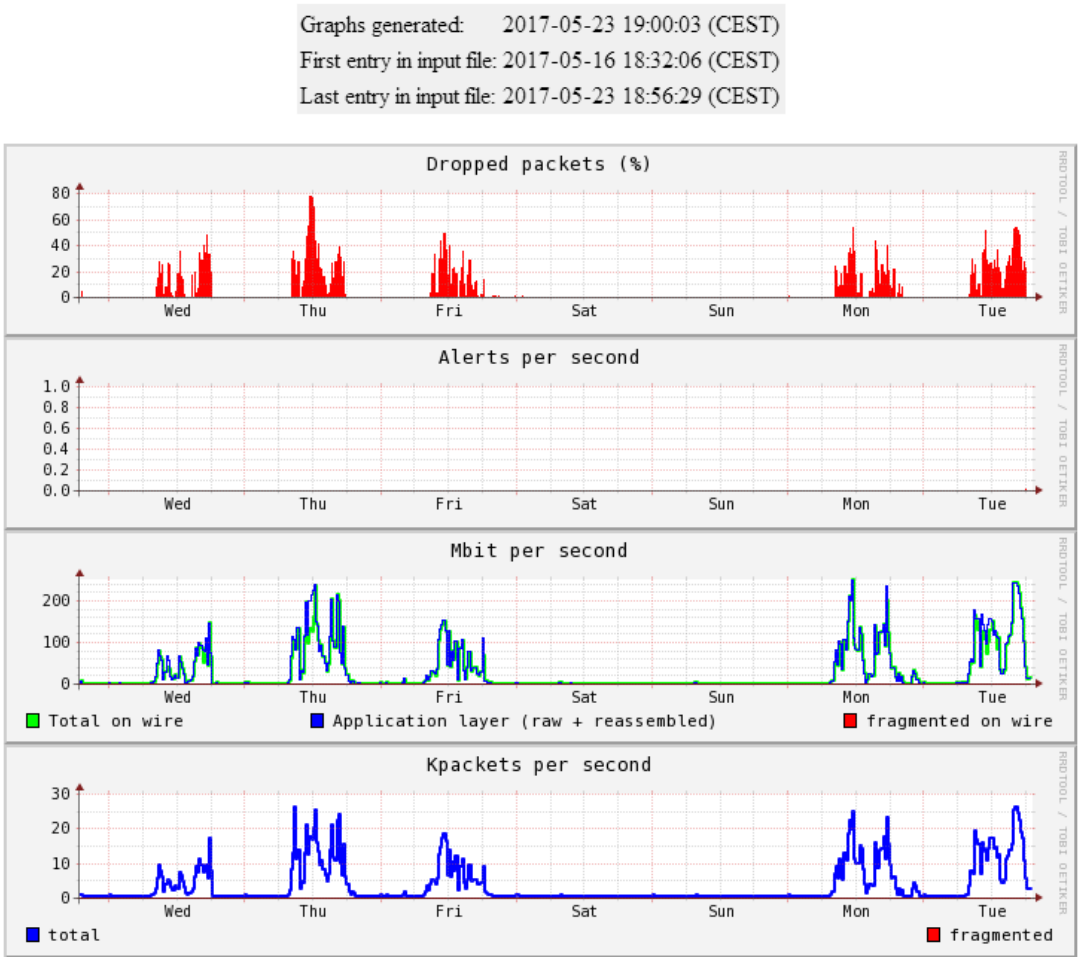


Ilustración 11. Zoom out PMGraph

Si se hace click de nuevo, vuelve al aspecto anterior. Esto es configurable desde el script de "pmgraph.pl".

Para utilizar este script hay que seguir los siguientes pasos:

1. Editar el archivo de configuración de Snort para que genere un archivo con datos sobre el rendimiento del IDS. Para esto se modificará la siguiente línea:

```
preprocesor perfmonitor: time 300 file /var/log/snort/stats.log
pktcnt 10000
```

   Donde *time 300* indica que el fichero se actualice cada 300 segundos, *file /var/log/snort/stats.log* es el fichero generado y *pktcnt 10000* es la cantidad de paquetes a procesar antes del tiempo indicado.
2. Instalar los paquetes necesarios en la máquina para poder ejecutar scripts desarrollados en perl.
3. Crear la carpeta donde se alojarán los archivos html generados por PMGraph con las gráficas del rendimiento de Snort:

```
mkdir /var/www/html/perfstats
```

4. Ejecutar el script pmgraph.pl con el siguiente comando:

```
$ pmgraph.pl /var/www/html/perfstats/ /var/log/snort/stats.log
```

La ejecución del script recopilará los últimos datos del registro generado por Snort, el cual almacena información del rendimiento del servicio, y con estos datos creará las gráficas en formato html en la carpeta indicada.

Una vez creados estos archivos html, podrán visualizarse en cualquier navegador, ya sea desplegándolos en un servidor web o abriéndolos directamente. El resultado se puede observar en las gráficas de arriba.

El uso habitual, es configurar dicho programa en el gestor de tareas o *crontab* para que se ejecute cada cierto tiempo (un tiempo razonable serían 5 minutos) y las gráficas se actualicen con los nuevos valores. Para esto, se ejecuta el comando *crontab –e* y se añade la siguiente línea en el gestor de tareas para que se actualicen las gráficas cada 5 minutos:

```
*/5 * * * * pmgraph.pl /var/www/html/perfstats/ /var/log/snort/stats.log
```

Por otro lado, también existen herramientas gráficas genéricas que se les pueden entregar los datos del rendimiento generados por Snort y para que generen gráficas con estos datos, como por ejemplo Zabbix, Zenoss o Nagios.

A continuación, y a modo de esquema resumen, se presenta una tabla para comprobar las funciones principales de las herramientas existentes y de la aplicación a desarrollar, que se le llamará de manera provisional SnortPerformance:

Programa	**Zabbix, Zenoss, Nagios**	**PMGraph**	**SnortPerformance**
Específico para el propósito		SI	SI
Estadísticas rendimiento Snort	SI	SI	SI
Estadisticas rendimiento reglas	SI		SI
Estadísticas rendimiento preprocesadores	SI		SI
Histórico de datos			SI
Dificultad puesta en marcha	Alta	Media	Media

3. Análisis del Problema

En el presente proyecto se propone integrar la información más relevante del rendimiento del IDS Snort en una aplicación gráfica, tanto los datos estadísticos del rendimiento del servicio como de las reglas y preprocesadores usados por el sistema.

La idea de este proyecto surge de un incidente en la empresa S2 Grupo, en el que al lanzar un conjunto de pruebas sobre un servidor IDS Snort no se generaban las alertas correspondientes. Tras analizar el caso, se concluye que debido a la gran cantidad de tráfico que pasaba por la interfaz de red configurada en Snort, junto con una gran cantidad de reglas genéricas, las cuales coincidían con la mayoría de paquetes, Snort descartaba la mayoría de los paquetes a analizar, llegando a alcanzar tasas de paquetes no analizados del 80-90%. Por esta razón, Snort no generaba todas las alertas que se le ponían a prueba.

Después de esta conclusión, se propuso la idea de realizar este trabajo. El objetivo del mismo es la visualización de los valores estadísticos en una interfaz gráfica para facilitar a los analistas y administradores de sistemas el estudio del rendimiento del sistema Snort y poder optimizarlo para que funcione al 100% de sus capacidades.

La decisión de crear esta herramienta de cero y no elegir ninguna de las ya existentes, se debe por lo desactualizadas que se encuentran estas aplicaciones y por qu ninguna de las herramientas publicadas permiten la posibilidad generar gráficas relacionadas con los datos estadísticos de las reglas y preprocesadores. Además, en la única aplicación diseñada para mostrar el rendimiento de Snort, PMGraph, estudiada anteriormente, solamente generaba gráficas con los datos de Snort, pero no se podía saber exactamente el valor de estos datos. En la herramienta propuesta se podrá observar el valor del dato, tanto en las gráficas como en una lista al final de la página junto a los demás valores.

Otro de los problemas de la necesidad de esta herramienta, es que analizar el rendimiento de Snort directamente a partir de los logs generados por Snort, puede resultar una tarea muy difícil y tediosa para los administradores. A continuación se puede ver un ejemplo de los datos estadísticos generados por Snort:

```
################################## Perfmon start: pid=17136 at=Wed Jun 15 09:45:55 2016 (1465976755) ##################################
#time,pkt_drop_percent,wire_mbits_per_sec.realtime,alerts_per_second,kpackets_wire_per_sec.realtime,avg_bytes_per_wire_packet,patmatch_percent,syns_per_second,synacks_per_second,new_sessions_per_second,deleted_sessions_per_second,total_sessions,max_sessions,stream_flushes_per_second,stream_faults,stream_timeouts,frag_creates_per_second,frag_completes_per_second,frag_inserts_per_second,frag_deletes_per_second,frag_autofrees_per_second,frag_flushes_per_second,current_frags,max_frags,frag_timeouts,frag_faults,iCPUs,usr[0],sys[0],idle[0],wire_mbits_per_sec.realtime,ipfrag_mbits_per_sec.realtime,ipreass_mbits_per_sec.realtime,rebuilt_mbits_per_sec.realtime,mbits_per_sec.realtime,avg_bytes_per_wire_packet,avg_bytes_per_ipfrag_packet,avg_bytes_per_ipreass_packet,avg_bytes_per_rebuilt_packet,avg_bytes_per_packet,kpackets_wire_per_sec.realtime,kpackets_ipfrag_per_sec.realtime,kpackets_ipreass_per_sec.realtime,kpackets_rebuilt_per_sec.realtime,kpackets_per_sec.realtime,pkt_stats.pkts_recv,pkt_stats.pkts_drop,total_blocked_packets,new_udp_sessions_per_second,deleted_udp_sessions_per_second,total_udp_sessions,max_udp_sessions,max_tcp_sessions_interval,curr_tcp_sessions_initializing,curr_tcp_sessions_established,curr_tcp_sessions_closing,tcp_sessions_midstream_per_second,tcp_sessions_closed_per_second,tcp_sessions_timedout_per_second,tcp_sessions_p
```

```
runed_per_second,tcp_sessions_dropped_async_per_second,current_attribute_host
s,attribute_table_reloads,mpls_mbits_per_sec.realtime,avg_bytes_per_mpls_packet,
kpackets_per_sec_mpls.realtime,total_tcp_filtered_packets,total_udp_filtered_packe
ts,ip4::trim,ip4::tos,ip4::df,ip4::rf,ip4::ttl,ip4::opts,icmp4::echo,ip6::ttl,ip6::opts,icmp
6::echo,tcp::syn_opt,tcp::opt,tcp::pad,tcp::rsv,tcp::ns,tcp::urg,tcp::urp,tcp::trim,tcp::ec
n_pkt,tcp::ecn_ssn,tcp::ts_ecr,tcp::ts_nop,tcp::ips_data,tcp::block,total_injected_pac
kets,frag3_mem_in_use,stream5_mem_in_use
1465976758,69.897,28.518,0.040,4.538,785,164.554,42.529,37.960,40.690,10.117,299
78,29978,17.696,0,2168,0.000,0.000,0.000,0.000,0.000,0.000,0,1,0,0,1,49.941,0.00
8,50.051,28.518,0.000,0.000,0.260,28.777,785,0,0,1839,789,4.538,0.000,0.000,0.01
8,4.554,453873,1053871,0,0.000,0.000,0,0,29978,17230,7733,5086,0.000,0.850,21.6
75,0.000,0.000,0,0,0.000,0,0.000,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,26462573,
1467930600,3.114,33.651,0.060,4.717,891,161.686,46.078,31.024,35.498,33.886,6655
0,174238,147.584,0,3093,0.010,0.010,0.010,0.010,0.000,0.010,3,5,0,0,1,32.542,0.400
,67.058,33.651,0.000,0.000,7.374,41.023,891,843,1649,6277,1054,4.717,0.000,0.000,
0.147,4.862,468144,15048,0,0.000,0.000,0,0,66684,34381,22955,14894,0.000,11.326
,31.165,0.000,0.000,0,0,0.000,0,0.000,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,2774,62121509,
1467930700,3.114,33.651,0.060,4.717,891,161.686,46.078,31.024,35.498,33.886,6655
0,174238,147.584,0,3093,0.010,0.010,0.010,0.010,0.000,0.010,3,5,0,0,1,32.542,0.400
,67.058,33.651,0.000,0.000,7.374,41.023,891,843,1649,6277,1054,4.717,0.000,0.000,
0.147,4.862,468144,15048,0,0.000,0.000,0,0,66684,34381,22955,14894,0.000,11.326
,31.165,0.000,0.000,0,0,0.000,0,0.000,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,2774,62121509,
(...)
```

Como se puede observar, estudiar el rendimiento de un sistema IDS a partir de estos registros sería una labor muy costosa y complicada.

Por otro lado, los datos sobre el uso de las reglas y de los preprocesadores, que están directamente relacionados con el rendimiento del IDS, y que genera el servicio de Snort, son más fáciles de estudiar y entender, como podemos ver a continuación:

- Rulestats

```
Rule Profile Statistics (all rules)
==========================================================
=
  Num     SID GID Rev     Checks   Matches    Alerts        Microsecs  Avg/Check
Avg/Match Avg/Nonmatch   Disabled
  ===     === === ===    ======  =======   ======       =========
=========  =========  ============  ========
    1 3000015   1   1 100181887   663422      21         17459957      0.2       1.8
0.2         0
    2 3000083   1   1 100181887    44091       8         27727318      0.3       2.2
0.3         0
    3 2001219   1  19     69305    25927     284           107840      1.6       3.8
0.2         0
    4 2020573   1   1     64392    22368       0           234107      3.6       9.0
0.8         0
    5 2021381   1   8    381922     9659       0           498183      1.3       7.9
1.1         0
    6 2002992   1   6    455789     8871       0           140028      0.3       5.0
0.2         0
```

```
    7  2402000   14106  281798775     7676       43          38884373       0.1       3.1
0.1         0
    8  2001972   1  18     16137     7486        8           19339        1.2       2.4        0.2
0
    9  2002994   1   6    103572     2625        4           34354        0.3       2.6        0.3
0
   10  2002993   1   6   1194681     2506        0          261074        0.2       3.3
0.2         0
   11  2002995   1   9   1194681     2450        7          257690        0.2       2.5        0.2
0
   12  2500052   13999  281798775     2109        4          40394906       0.1       8.4
0.1         0
   13  2019876   1   2      2227     2037      471           6842        3.1       3.3        0.4
0
   14  3000049   1   1     47959     1790      163         115819        2.4      43.4
0.8         0
   15  2403318   12779  281798775     1420        2          36655662       0.1       4.2
0.1         0
   16  2016539   1   4      3770     1373        0          230694       61.2     121.6       26.6
0
   17  3000050   1   1     47959     1367       37           16767        0.3       1.5        0.3
0
   18  2500070   13999  281798775     1360       14          39338237       0.1      12.0
0.1         0
   19  2403310   12779  281798775     1289       45          36621997       0.1       5.1
0.1         0
(... y así hasta la última regla que haya interactuado con el sistema)
```

- Preprocstats

```
Preprocessor Profile Statistics (all)
==========================================================
=
 Num          Preprocessor Layer     Checks      Exits         Microsecs  Avg/Check Pct
of Caller Pct of Total
 ===        ============ =====    ======     =====        =========
========= ============ ============
  1              frag3     0       927       927          6796570    7331.79        0.01        0.01
  1         frag3rebuild     1       453       453           243016     536.46        3.58
0.00
  2          frag3insert     1       474       474           89260     188.31        1.31        0.00
  2               detect     0 324385294 324385294       28753377314      88.64
53.13       53.13
  1                 mpse     1 430501586 430501586       13044242217      30.30
45.37       24.10
  2            rule eval     1 1023369377 1023369377       15415640435      15.06
53.61       28.49
  1       rule tree eval     2 1653134133 1653134133       15217354274       9.21
98.71       28.12
(... y así hasta el último preprocesador configurado en Snort)
```

Sin embargo, igual sólo interesaría analizar las 30 reglas que más microsegundos tarden en analizar un paquete o comparar las veces que éstas han coincidido con el contenido de los paquetes con las veces que han generado una alerta por dicha firma, opciones que estarán disponibles desde la herramienta a diseñar.

A continuación se detallan en una tabla los requisitos para comenzar a diseñar la herramienta propuesta en el proyecto:

Requisito	**Descripción**
R1	Recopilar todos los datos estadísticos
R2	Mostrar los datos recopilados en gráficas útiles
R3	Mantener en la interfaz gráfica un histórico de los datos estadísticos
R4	Actualización de gráficas en tiempo real sin recargar la web

3.1. Análisis de las soluciones

Ante los problemas planteados anteriormente, se propone implementar una solución que facilite el análisis de estos datos.

Como se ha analizado en el estado del arte, existen aplicaciones, las cuales se le pueden enviar los datos elegidos para la visualización de éstos en gráficas, pero sólo existe una herramienta específica para la visualización de estos datos estadísticos generados por Snort.

Ante la falta de soluciones existentes a este problema, se plantea crear una aplicación web específica para recoger estos datos estadísticos del IDS Snort y crear gráficas para estudiar el rendimiento, tanto del propio sistema IDS, como de las reglas y preprocesadores cargados.

3.2. Solución propuesta

La solución propuesta en este proyecto para solventar este problema, se dividirá en dos partes:

1. En primer lugar, se configurará Snort para que guarde los registros estadísticos en una carpeta compartida con el servidor donde se alojará la aplicación diseñada. A continuación, se desarrollarán unos agentes/programas que se programarán en el gestor de tareas donde se ubique la herramienta, para que se ejecuten cuando Snort genere los registros con los datos estadísticos del rendimiento de la sonda.

   La principal y única función de estos programas será la de *parsear* los registros del rendimiento generados por Snort y enviarlos a la aplicación.

   Se ha elegido la opción de guardar estos datos en una carpeta compartida y desde otra máquina *parsearlos*, para no afectar al rendimiento del servidor IDS.

2. En esta segunda parte, como se puede intuir, se recogerán estos datos enviados y se almacenarán en una base de datos. A partir de esta información almacenada, se crearán gráficas mostradas en una interfaz web para facilitar las labores de estudio y optimización del rendimiento del IDS Snort.

Existe mucha variedad de datos que almacena Snort en sus registros sobre el rendimiento del sistema. A continuación, se indican los distintos tipos de datos que Snort almacena en cada uno de los tres ficheros de datos que nos interesan:

- **Snort.stats**: time, pkt_drop_percent, wire_mbits_per_sec.realtime, alerts_per_second, kpackets_wire_per_sec.realtime, avg_bytes_per_wire_packet, patmatch_percent, syns_per_second,

synacks_per_second, new_sessions_per_second, deleted_sessions_per_second, total_sessions, max_sessions, stream_flushes_per_second, stream_faults, stream_timeouts, frag_creates_per_second, frag_completes_per_second, frag_inserts_per_second, frag_deletes_per_second, frag_autofrees_per_second, frag_flushes_per_second, current_frags, max_frags, frag_timeouts, frag_faults, iCPUs, usr[0], sys[0], idle[0], wire_mbits_per_sec.realtime, ipfrag_mbits_per_sec.realtime, ipreass_mbits_per_sec.realtime, rebuilt_mbits_per_sec.realtime, mbits_per_sec.realtime, avg_bytes_per_wire_packet, avg_bytes_per_ipfrag_packet, avg_bytes_per_ipreass_packet, avg_bytes_per_rebuilt_packet, avg_bytes_per_packet, kpackets_wire_per_sec.realtime, kpackets_ipfrag_per_sec.realtime, kpackets_ipreass_per_sec.realtime, kpackets_rebuilt_per_sec.realtime, kpackets_per_sec.realtime, pkt_stats.pkts_recv, pkt_stats.pkts_drop, total_blocked_packets, new_udp_sessions_per_second, deleted_udp_sessions_per_second, total_udp_sessions, max_udp_sessions, max_tcp_sessions_interval, curr_tcp_sessions_initializing, curr_tcp_sessions_established, curr_tcp_sessions_closing, tcp_sessions_midstream_per_second, tcp_sessions_closed_per_second, tcp_sessions_timedout_per_second, tcp_sessions_pruned_per_second, tcp_sessions_dropped_async_per_second, current_attribute_hosts, attribute_table_reloads, mpls_mbits_per_sec.realtime, avg_bytes_per_mpls_packet, kpackets_per_sec_mpls.realtime, total_tcp_filtered_packets, total_udp_filtered_packets, ip4::trim, ip4::tos, ip4::df, ip4::rf, ip4::ttl, ip4::opts, icmp4::echo, ip6::ttl, ip6::opts, icmp6::echo, tcp::syn_opt, tcp::opt, tcp::pad, tcp::rsv, tcp::ns, tcp::urg, tcp::urp, tcp::trim, tcp::ecn_pkt, tcp::ecn_ssn, tcp::ts_ecr, tcp::ts_nop, tcp::ips_data, tcp::block, total_injected_packets, frag3_mem_in_use, stream5_mem_in_use

- **Rules.stats**: Num, SID, GID, Rev, Checks, Matches, Alerts, Microsecs, Avg/Check, Avg/Match, Avg/Nonmatch, Disabled.

- **Preprocessor.stats**: Num, Preprocessor, Layer, Checks, Exits, Microsecs, Avg/Check, Pct of Caller, Pct of Total.

Para más información sobre estas variables se puede visitar la documentación oficial en los siguientes enlaces:

- http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node17.html#SECTION00326000000000000000 (para las variables de Snort.stats)
- http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node20.html#SECTION00353300000000000000 (para las variables de Rules.stats y Preprocessor.stats)

Ante la cantidad de información registrada, no tiene sentido realizar gráficas de todos los datos almacenados. El objetivo, como ya se ha dicho, es facilitar el análisis a los administradores, por lo que se van a elegir los datos más importantes y las combinaciones entre ellos más útiles. A continuación se detallan los datos que se van a enviar y almacenar en la base de datos:

- **Snort.stats**: time, pkt_drop_percent, wire_mbits_per_sec.realtime, alerts_per_second, kpackets_wire_per_sec.realtime, avg_bytes_per_wire_packet, syns_per_second, synacks_per_second, new_udp_sessions_per_second, deleted_udp_sessions_per_second,

curr_tcp_sessions_initializing, curr_tcp_sessions_established, curr_tcp_sessions_closing, iCPUs, usr[0], sys[0], idle[0], frag3_mem_in_use, stream5_mem_in_use

- **Rules.stats**: Num, SID, GID, Rev, Checks, Matches, Alerts, Microsecs, Avg/Check, Avg/Match, Avg/Nonmatch, Disabled, InclusionDate(*).

- **Preprocessor.stats**: Num, Preprocessor, Layer, Checks, Exits, Microsecs, Avg/Check, Pct of Caller, Pct of Total, InclusionDate(*).

*Como se puede observar, se ha añadido el siguiente dato a almacenar en las base de datos para tener registro de la fecha en la que se almacenó este dato. Este valor no se añade en los datos de "Snort.stats" ya que aquí ya se tiene el valor de "time".

Con los valores anteriores recopilados, se diseñarán las siguientes gráficas para el estudio del rendimiento del IDS Snort:

- "**% Dropped Packets**": se mostrará el porcentaje de paquetes no analizados del día X a las XX:XX horas. A su vez, se mostrará el porcentaje de CPU usado por parte del usuario y del sistema.

- "**Mbits per Second**": se mostrará la cantidad de Mbits por segundo utilizados por Snort en tiempo real del día X a las XX:XX horas.

- "**SYN + SYN/ACK Packets per Second**": se muestra la cantidad de paquetes SYN y SYN/ACK del día X a las XX:XX horas.

- "**Kpackets per Second**": se mostrará la cantidad de miles de paquetes por segundo consumidos por Snort en tiempo real del día X a las XX:XX horas.

- "**UDP Sessions**": se mostrará la cantidad de nuevas sesiones UDP y sesiones eliminadas del día X a las XX:XX horas.

- "**TCP Sessions**": se mostrará la cantidad de sesiones TCP cerradas, establecidas e inicializadas del día X a las XX:XX horas.

- "**Top Microsecs Analyzing a Rule**": se mostrarán las reglas que más microsegundos han tardado en analizar todos los paquetes. Además, se muestran la cantidad de veces que ha comprobado una regla en todos los paquetes y el promedio en que no ha coincidido.

- "**Top Checks**": se mostrarán las reglas que más veces han comprobado en todo el tráfico analizado. Además se complementa esta gráfica con el número de veces que la regla ha coincidido y ha generado una alerta.

- "**Top Preprocessor Executions**": se mostrará, de mayor a menor, los preprocesadores que más veces se han ejecutado con éxito. A su vez, se muestran las veces que han verificado contra un paquete.

- "**Top Avg/checks**": se mostrará, de mayor a menor, el promedio de veces que han comprobado los preprocesadores. A su vez, se muestran las veces que se han llamado a los preprocesadores.

4. Diseño de la Solución

En este punto de la memoria, una vez explicado el problema y planteada una solución, se inician las tareas de diseño de la misma, en las cuales se detallarán las herramientas que se van a utilizar para diseñar la aplicación planteada.

Al igual que se ha hecho en el análisis de la solución, se dividirá el diseño de la herramienta en dos partes:

1. Para los programas encargados de enviar los datos a la herramienta, se utilizará el lenguaje de programación Python en su versión 2.7. Se ha elegido este lenguaje por la tarea que van a desarrollar estos agentes, la cual va a ser analizar los registros estadísticos de Snort y enviarlos a la aplicación creada. Para esta tarea tan "sencilla", este lenguaje es de gran utilidad ya que cuenta con la ayuda de múltiples librerías que facilitan la programación de estos *scripts*, haciendo posible su desarrollo de manera rápida y asequible.

2. Para el diseño de la herramienta, se utilizará una base de datos MongoDB[10] y para su desarrollo se hará uso de 3 lenguajes de programación:

    a. Como ya se ha mencionado, para el almacenamiento de los datos se ha elegido una base de datos MongoDB.

    MongoDB es un sistema de base de datos noSQL multiplataforma orientado a documentos donde cada entrada puede tener un esquema de datos diferente.

    Lo más destacable de este sistema es su velocidad de búsqueda de datos y su rico sistema de consulta. Pero, la principal elección de esta base de datos y no otra, ha sido su gran escalabilidad, ya que si en un futuro se deseara almacenar más tipos de datos, se podrían añadir estos cambios en la base de datos de manera rápida y sencilla y sin afectar al funcionamiento de la aplicación.

    b. Para la parte del *Backend*, se utilizará Go[11] en su versión 1.7.6: Go, o Golang, es un lenguaje de programación concurrente y compilado inspirado en la sintaxis de C. Es un lenguaje relativamente "nuevo", cuyo primer lanzamiento fue en 2009, desarrollado por Google como un proyecto de código abierto.

    Se ha elegido este lenguaje para la construcción de la herramienta ya que está diseñado para la programación de sistemas, aunque este lenguaje ofrece un gran abanico de posibilidades para programar cualquier proyecto. Además, existen una gran cantidad de librerías que facilitarán su desarrollo.

    Para este proyecto, además de las múltiples librerías que ofrece Go, también se hará uso de las siguientes:

[10] https://www.mongodb.com/es
[11] https://golang.org/

- Mgo[12]: Es un controlador de MongoDB para el lenguaje de Go que implementa una gran lista de funciones bajo una sencilla API que sigue los estándares del lenguaje de Go. Dicha API facilitará la interacción entre la base de datos y la aplicación a desarrollar.

- Gorilla/mux[13]: Gorilla es un conjunto de herramientas web para el lenguaje de programación Go. En particular, se hará uso de la API de mux, un recurso de Gorilla que proporciona funciones de enrutador y distribuidor de URL, el cual se usará para administrar los diferentes recursos de la aplicación web a implementar.

c. Por último, para la parte del *Frontend*, se va a utilizar el lenguaje de etiquetado HTML en su versión 5. HTML5, es un lenguaje estándar para diseñar el contenido de las páginas web. A su vez, se ha combinado con el lenguaje CSS (Cascading Style Sheets) para describir la presentación del documento HTML.

   A su vez, se utilizará el lenguaje de programación interpretado JavaScript para mejorar la interfaz web de la aplicación y proporcionar dinamismo a esta, como por ejemplo, con las gráficas mostradas en la página web. Además, para facilitar la interacción entre la parte del frontend (interfaz) y el backend (servidor), se gastará una biblioteca multiplataforma de JavaScript, llamada jQuery[14].

En un primer boceto, el aspecto de la aplicación web sería como muestra la siguiente figura:

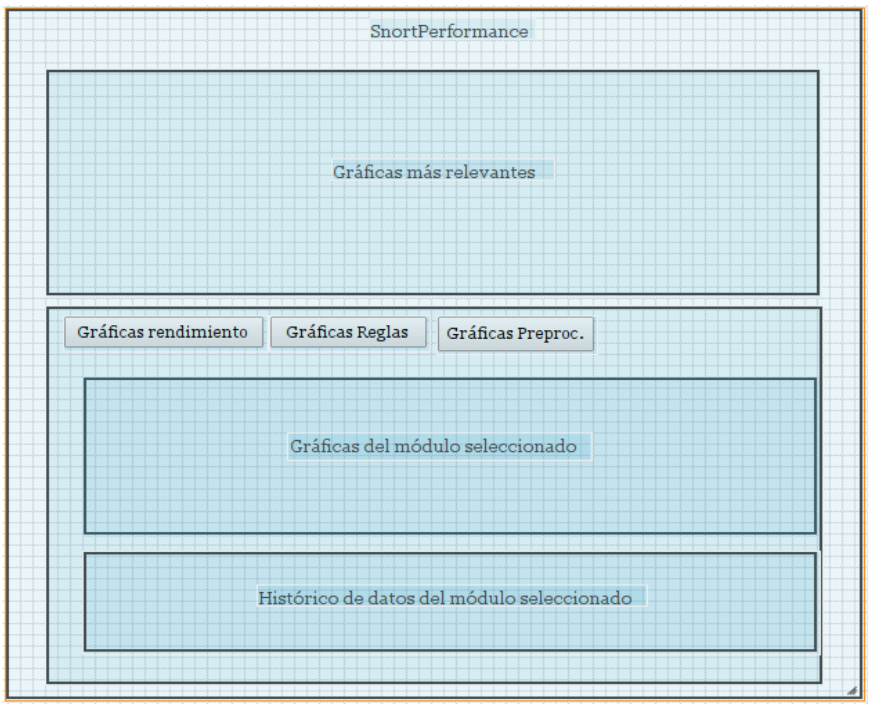


Ilustración 12. Boceto aplicación web

[12] gopkg.in/mgo.v2
[13] www.gorillatoolkit.org/pkg/mux
[14] https://jquery.com/

Por otro lado, para la parte del servidor IDS, se ha instalado Snort en su versión 2.9.6.0 sobre un sistema operativo Ubuntu Server 14.04 LTS.

A modo de esquema resumen, en la siguiente imagen se puede observar cómo quedaría diseñada la herramienta propuesta:

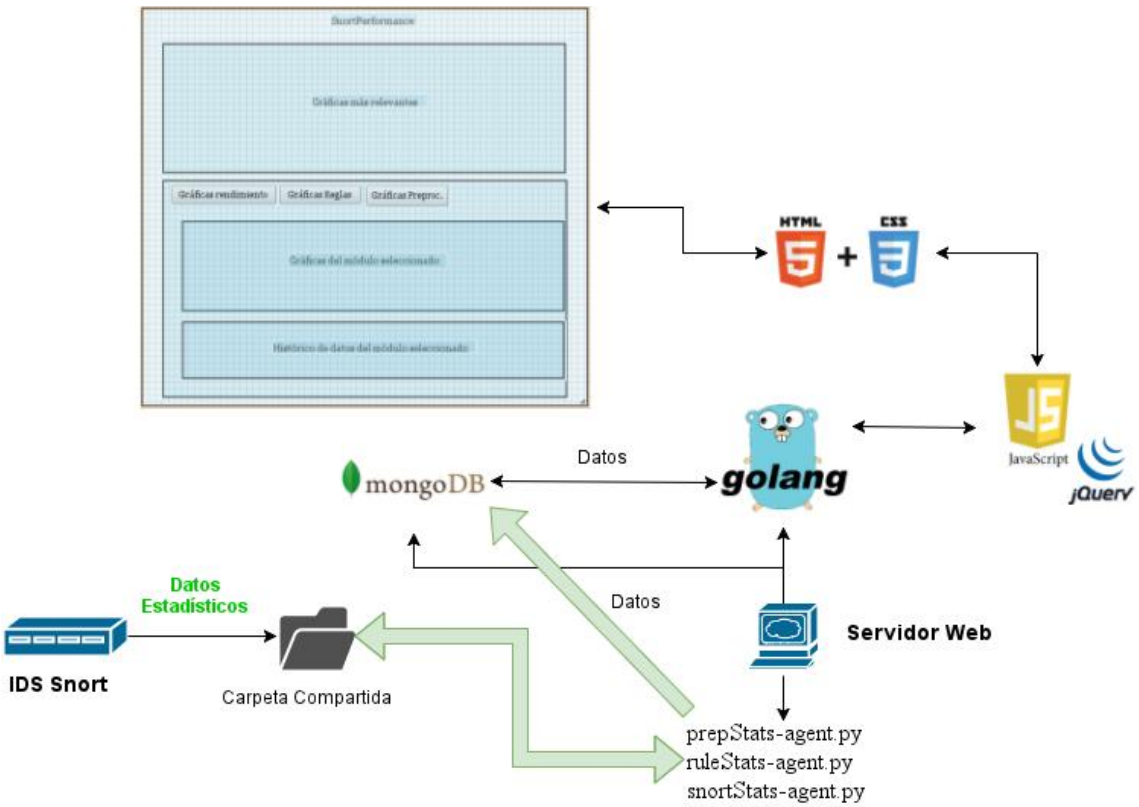


Ilustración 13. Esquema diseño de la herramienta

5. Implementación

En este capítulo se detallarán los requisitos principales para poner en funcionamiento la herramienta diseñada y que el cliente pueda visualizar y estudiar el rendimiento de su sistema IDS Snort de una manera sencilla y cómoda.

5.1. Primera parte: envío de datos estadísticos de Snort

Se ha instalado en un servidor Ubuntu 14.04 un IDS Snort funcionando y analizando todo el tráfico de la red configurada. La instalación de Snort se puede ver en el ANEXO final como se ha detallado previamente.

Por otro lado, se modifican las siguientes configuraciones de Snort en su archivo de configuración /etc/snort/snort.conf para que el sistema IDS genere registros de estadísticas:

1. **Perfmonitor**: Preprocesador el cual genera un archivo de estadísticas globales sobre el funcionamiento general de Snort. Este archivo se va actualizando con nuevos datos estadísticos cada cierto tiempo, indicado en la configuración del preprocesador. Algunos ejemplos de estos datos, como se ha visto en el apartado anterior, serían; tasa de paquetes sin analizar, microsegundos que tarda una regla en analizar un paquete, etc:

```
# performance statistics.  For more information, see the Snort Manual, Configuri
ng Snort - Preprocessors - Performance Monitor
preprocessor perfmonitor: time 300 file /home/snort/SnortStatsCompartida/snort.s
tats pktcnt 10000
```

Ilustración 14. Configuración preprocesador perfmonitor

   Dicho archivo se genera cuando se ejecuta Snort por primera vez con esta configuración, y a partir de ese momento, se va actualizando con los nuevos datos estadísticos calculados por Snort cada X segundos, indicados en las opciones del propio preprocesador.

2. **Profile_rules**: configuración la cual, en cada reinicio o apagado de Snort, genera un archivo de estadísticas de las reglas utilizadas en el periodo anterior, desde que se encendió Snort hasta que se apagó. Algunos ejemplos de estos datos serían; nº de alertas por firma, promedio de comprobaciones por firma, etc.

3. **Profile_preprocs**: igual que profile_rules, pero en este caso, los datos estadísticos son de los preprocesadores. Algunos ejemplos de estos datos serían; nº de ejecuciones de cada preprocesador, promedio de comprobaciones por preprocesador, etc.

```
##################################################
# Configure Perf Profiling for debugging
# For more information see README.PerfProfiling
##################################################
config profile_rules: print all, sort avg_ticks, filename /home/snort/SnortStats
Compartida/rulestats.stats
config profile_preprocs: print all, sort avg_ticks, filename /home/snort/SnortSt
atsCompartida/preprocstats.stats
```

Ilustración 15. Configuración rendimiento reglas y preprocesadores

Estos dos últimos archivos, se generan cuando la sesión del servicio de Snort finaliza, bien sea porqué se haya parado (stop) o se haya reiniciado (restart), ya que el proceso de reiniciar conlleva parar el servicio para volverlo a iniciar.

Como se puede observar en las ilustraciones 14 y 15, se configuran estos módulos para que guarden los ficheros con datos estadísticos en una carpeta compartida llamada "SnortStatsCompartida".

Para compartir esta carpeta se ha utilizado el servicio de Samba [15], instalado directamente desde los repositorios de Ubuntu:

```
sudo apt-get install samba
```

Una vez instalado, se añade la carpeta a compartir, con los requisitos que se requieran, en el archivo de configuración de Samba */etc/samba/smb.conf*:

```
[SnortStatsCompartida]
   path = /home/snort/SnortStatsCompartida
   read only = yes
   guest ok = yes
```

Ilustración 16. Samba: configuración carpeta compartida

- *path*: ruta del directorio que se va a compartir.
- *read only*: determina si el recurso compartido es sólo de lectura. En este caso, no se requiere escribir en los archivos, por lo que configuraremos este parámetro con un "yes".
- *guest ok*: permite que los clientes se puedan conectar al recurso compartido sin necesidad de autenticación. Como la información almacenada en esta carpeta no es crítica, se ha decidido poder acceder a la carpeta compartida sin contraseña.

Una vez realizadas estas configuraciones, se programa en el *crontab* del servidor donde se alojará la herramienta, la ejecución de los agentes/programas, hechos en Python y mencionados anteriormente, para que lean los archivos de estadísticas creados por Snort y que envíen estos datos en formato JSON[16] a la aplicación. A continuación se enumeran los agentes a programar:

1. **globalStats-agent.py**: se programa para que cuando el archivo de estadísticas globales se modifique, éste envíe los nuevos datos actualizados a la herramienta.

[15] Programa libre para la compartición de archivos e impresoras a través del protocolo SMB/CIFS. (https://help.ubuntu.com/lts/serverguide/samba.html)
[16] JavaScript Object Notation: formato de texto ligero para el intercambio de datos)

2. **ruleStats-agent.py**: se añade una tarea en el crontab, por ejemplo justo después de que se reinicie Snort, para *parsear* y enviar los datos estadísticos de las reglas a la herramienta.
3. **preprocStats-agent.py**: se añade una tarea en el crontab, por ejemplo justo después de que se reinicie Snort, para *parsear* y enviar los datos estadísticos de los preprocesadores a la herramienta.

El primer agente, bastaría con ejecutarlo una vez al inicio del sistema, no hace falta añadirlo al gestor de tareas, ya que está programado para que envíe los nuevos datos cuando se modifique el propio archivo de estadísticas generado por Snort. Para que se lance al inicio del sistema, se puede programar su ejecución en el archivo */etc/rc.local*, el contenido del fichero quedaría como se muestra en la siguiente figura:

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

python /home/snort/agentes/globalStats-agent.py
exit 0
```

Ilustración 17. Ejecución globalStats-agent.py en el inicio del sistema con rc.local

Sin embargo, los otros dos agentes, se tendrían que ejecutar cuando Snort genere los registros de las reglas y preprocesadores, que cómo se ha dicho, son generados al finalizar la sesión del IDS. Por lo cual, se podría añadir en el gestor de tareas o *crontab* del servidor IDS, que se reinicie el servicio de Snort a las 3:00 a.m. todos los días:

```
0 3 * * * service snort restart
```

Ilustración 18. Programación en el crontab para el reinicio de Snort

Y, por otro lado, programar en el gestor de tareas del servidor donde se aloja la herramienta, la ejecución de los agentes 5 minutos más tarde, tiempo de sobra para que Snort se reinicie y cree los ficheros con los valores estadísticos. A continuación se puede observar cómo quedaría el gestor de tareas después de los cambios realizados:

```
5 3 * * * python /home/snort/agentes/ruleStats-agent.py
5 3 * * * python /home/snort/agentes/preprocStats-agent.py
```

Ilustración 19. Programación en el crontab para la ejecución de los agentes

5.2. Segunda parte: Implementación de la herramienta

El código de la herramienta se divide en 7 módulos diferenciados. A continuación se detalla un breve resumen de cada módulo:

1. **snortperformance.go**: módulo principal encargado de lanzar la herramienta. Sus funciones principales son: llamar al módulo spDB.go, explicado a continuación, para conectar la aplicación con la base de datos y llamar al módulo router.go, también explicado a continuación, para desplegar el servidor web en la dirección IP y puerto configurados:

```
func main() {
    flag.Parse()

    log.SetPrefix("[●] ")
    log.SetFlags(log.Ldate | log.Ltime | log.Lmicroseconds)
    log.Printf("Starting SnortPerformance server %s", version)

    // Web db stuff configuration and initialization
    db.SFDB = db.SFDBStore{
        DBAddr: dbAddrFlag,
        DBPort: dbPortFlag,
        DBName: dbNameFlag,
    }
    db.SFDB.Connect()

    // Sets the global version for web handler
    web.Version = version

    // Start Web router goroutine
    web.Router(webAddrFlag, webPortFlag, db.SFDB)
}
```

Ilustración 20. Muestra de código snortperformance.go

2. **spDB.go**: módulo encargado de crear y conectarse a las colecciones de la base de datos correspondientes. Además, incluye todos los métodos necesarios que tengan que interactuar con la colección de MongoDB, como por ejemplo, la función de almacenar los datos estadísticos de las reglas en la base de datos:

```
// InsertRuleStats: Inserts Rule stats of rules from snort stats file called rulestats.log
into db in rulestatsCollection collection
func (s *SFDBStore) InsertRuleStats(st RuleStatsLog) error {
    col := s.db.C(rulestatsCollection)

    now := time.Now()
    today := time.Date(now.Year(), now.Month(), now.Day(), now.Hour(), now.Minute(), 00, 00,
        now.Location())

    st.InclusionDate = today
    if err := col.Insert(st); err != nil {
        return err
    }

    return nil
}
```

Ilustración 21. Muestra de código spDB.go

3. **router.go**: módulo encargado de desplegar la aplicación web en la dirección IP y puerto configurado en el módulo principal. Además, en él también se configuran todos los manejadores de contenido de la web, los cuales se utilizarán, por ejemplo, para saber a qué recurso enviar los datos estadísticos y saber qué hacer con ellos:

```
// Statistics Handlers
r.HandleFunc("/uploadRuleStats", uploadRuleStatsHandler).Methods("POST")
r.HandleFunc("/uploadPrepStats", uploadPrepStatsHandler).Methods("POST")
r.HandleFunc("/uploadTotalStats", uploadTotalStats).Methods("POST")

// Serves static content
r.PathPrefix("/").Handler(http.FileServer(http.Dir("./static/")))

// Sets r to middleware
http.Handle("/", headerMiddleware(r))

// Starts serving
log.Printf("Listening on: http://%s:%s", addr, port)
http.ListenAndServe(addr+":"+port, nil)
```

Ilustración 22. Muestra de código router.go.

4. **webController.go**: módulo donde se ubican todos los métodos encargados de realizar la función correspondiente a cada manejador web. A continuación se puede ver un ejemplo del método llamado cuando se accede al manejador subrayado, en color naranja, en la ilustración 22 (http://dir-ip-aplicación:puerto-aplicación/uploadTotalStats):

```
// uploadTotalStats: take total stats from stats.log
func uploadTotalStats(rw http.ResponseWriter, r *http.Request) {
    // Receive JSON data an store it in struct
    decoder := json.NewDecoder(r.Body)
    var stat db.TotalStatsLog
    err := decoder.Decode(&stat)
    if err != nil {
        log.Println(err)
    }

    // Insert stats into DB
    if err := SFDatabase.InsertTotalStats(stat); err != nil {
        log.Println(err)
    }

    http.Redirect(rw, r, "/statistics", http.StatusFound)

    return
}
```

Ilustración 23. Muestra de código webController.go

5. **index.html**: el módulo anterior llama a este módulo cuando se accede a la ruta principal de la web (http://dir-ip-aplicación:puerto-aplicación/). La función de esta parte, como ya se ha explicado en puntos anteriores, es representar visualmente la página web de la aplicación creada. A continuación se muestra un ejemplo del código html de la gráfica “% Dropped Packets”:

```
<div class="container">
    <div class="row">
        <div class="col-md-12">

                <div class="text-center right-spacing">
                    <h3>% Dropped Packets</h3>
                    <label for="contain" type="text">Last</label>
                    <input class="chartDay" id="dayDrop" type="number" min="1" max="7"
                    placeholder="1" value="1"/>
                    <label for="contain">Day</label>
                    <button type="button" class="btn btn-primary chartStat" id="searchDrop">
                    <span class="glyphicon glyphicon-search" aria-hidden="true"></span></
                    button>
                </div>

            <canvas id="chartTotalStatsDrop" height="75"></canvas>
        </div>
    </div>
</div>
```

Ilustración 24. Muestra de código index.html

6. **main.css**: módulo encargado para definir el diseño de las diferentes partes de la interfaz web. Además de este archivo CSS, también se han utilizado otros ya predefinidos: bootstrap.min.css, bootstrap-theme.min.css y font-awesome.min.css. A continuación se puede ver un ejemplo del código del archivo main.css donde se configuran las medidas y posición de las gráficas de la web:

```
/* Charts */
.chartStat {
    padding: 4px 8px;
}
.chartDay {
    width: 70px;
    height: 30px;
    padding: 4px 8px;
}

.chartLog {
    width: 100%;
}
```

Ilustración 25. Muestra de código main.css

7. **app.js**: módulo que realiza la función de intermediario entre el módulo 5 y el módulo 3, encargado de dar funcionalidad a las gráficas y a los botones para proporcionar dinamismo a la interfaz web. A continuación se muestra el método para crear la gráfica "% Dropped Paclets" según la cantidad de días atrás indicada:

```
// TotalStatsLog
function populateTotalStatsLogChartsDrop() {
    var day = $("#dayDrop").val()

    $.ajax({
        type: "GET",
        url: "/api/v1/stats/totalstats/drops/"+day,
    })

    .done(function(d) {
        var stats = []
        stats = d.stats

        var datalabels = []
        var datadrops = []
        var datacpuusr = []
        var datacpusys = []

        $.each(d.stats, function(idx, val){
            date = timeConverter(val.time)
            datalabels.push(date)
            datadrops.push(val.pkt_drop_percent)
            datacpuusr.push(val['usr[0]'])
            datacpusys.push(val['sys[0]'])
        });

        var data = {
            labels: datalabels,
            datasets: [
                {
                    type:'line',
                    label: "% CPU Usr",
                    data: datacpuusr,
                    fill: false,
                    backgroundColor: colorPallete[1],
                    borderColor: colorPallete[1],
                    hoverBackgroundColor: colorPallete[1],
                    hoverBorderColor: colorPallete[1]

                }, {
                    type:'line',
                    label: "% CPU Sys",
                    data: datacpusys,
                    fill: false,
                    backgroundColor: colorPallete[2],
                    borderColor: colorPallete[2],
                    hoverBackgroundColor: colorPallete[2],
                    hoverBorderColor: colorPallete[2]

                },{
                    type:'bar',
                    label: "% Drops",
                    data: datadrops,
                    fill: false,
                    hoverBorderWidth: 5000,
                    backgroundColor: colorPallete[0],
                    borderColor: colorPallete[0],
                    hoverBackgroundColor: colorPallete[0],
                    hoverBorderColor: colorPallete[0]
                }
            ]
        };

        var myLineChart = new Chart($(chartTotalStatsDrop), {
            type: 'bar',
            data: data,
                options: {
                responsive: true,
                tooltips: {
                  mode: 'label'
              },
              elements: {
                line: {
                    fill: false
                }
            },
            }
        });
    })

    .fail(function(d) {
        console.log(d)
        console.log("Err")
    });
}
```

Ilustración 26. Muestra de código app.js

Una vez programada la herramienta, y dado que Go es un lenguaje de programación compilado, se compila la aplicación en un solo ejecutable. Ya con la herramienta compilada, se ha colocado sobre un servidor Debian 8.7 Jessie, en el cual se ha instalado el gestor de base de datos MongoDB.

Con todo esto, se ha ejecutado la herramienta en el servidor, sin necesidad de instalar Go en el sistema ni importar sus librerías. Lo que sí que habrá que importar a este

servidor, será la información estática de la interfaz web del programa: fuentes, imágenes y el código HTML, CSS y JavaScript.

En este punto, se lanza la aplicación y se deja funcionando a la espera de recibir datos del sistema IDS. Su ejecución mostrará la siguiente información en pantalla:

```
/tfg-SnortPerformance# ./snortPerformanceServer
[●] 2017/07/01 08:33:38.747571 Starting SnortPerformance server v0.1β
[●] 2017/07/01 08:33:38.749155 Mongo: SnortPerformance database conected
[●] 2017/07/01 08:33:38.749815 Listening on: http://192.168.1.24:8080
```

Ilustración 27. Ejecución de la herramienta SnortPerformance

Ahora, ya es posible acceder a la aplicación dirigiéndose a la dirección "**http://192.168.1.24:8080**".

La primera impresión que se tendría al acceder a la aplicación es la siguiente:

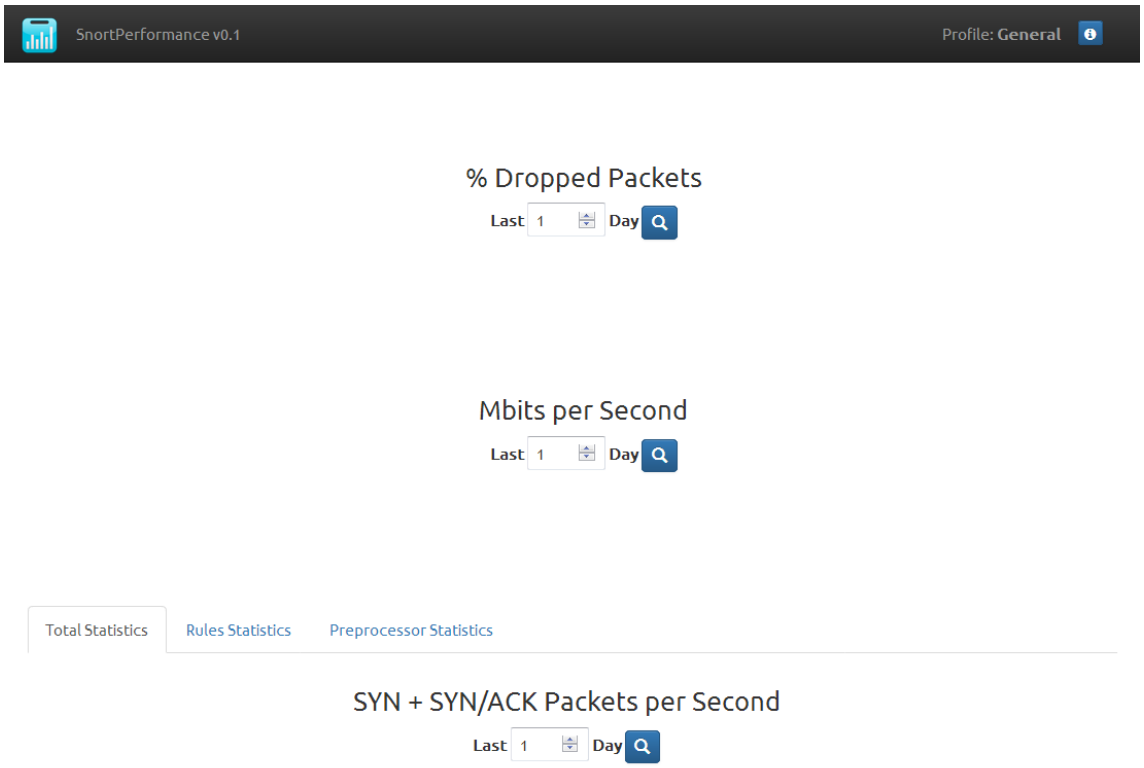


Ilustración 28. Interfaz SnortPerformance sin datos

Por el momento, aún no se han recibido datos estadísticos, por tal motivo, la aplicación aparece tan triste.

En la parte de arriba a la derecha se puede observar un cuadro de ayuda en el cual se detalla un breve resumen de la herramienta diseñada y su funcionamiento. Si se pulsa sobre él aparece la siguiente información:

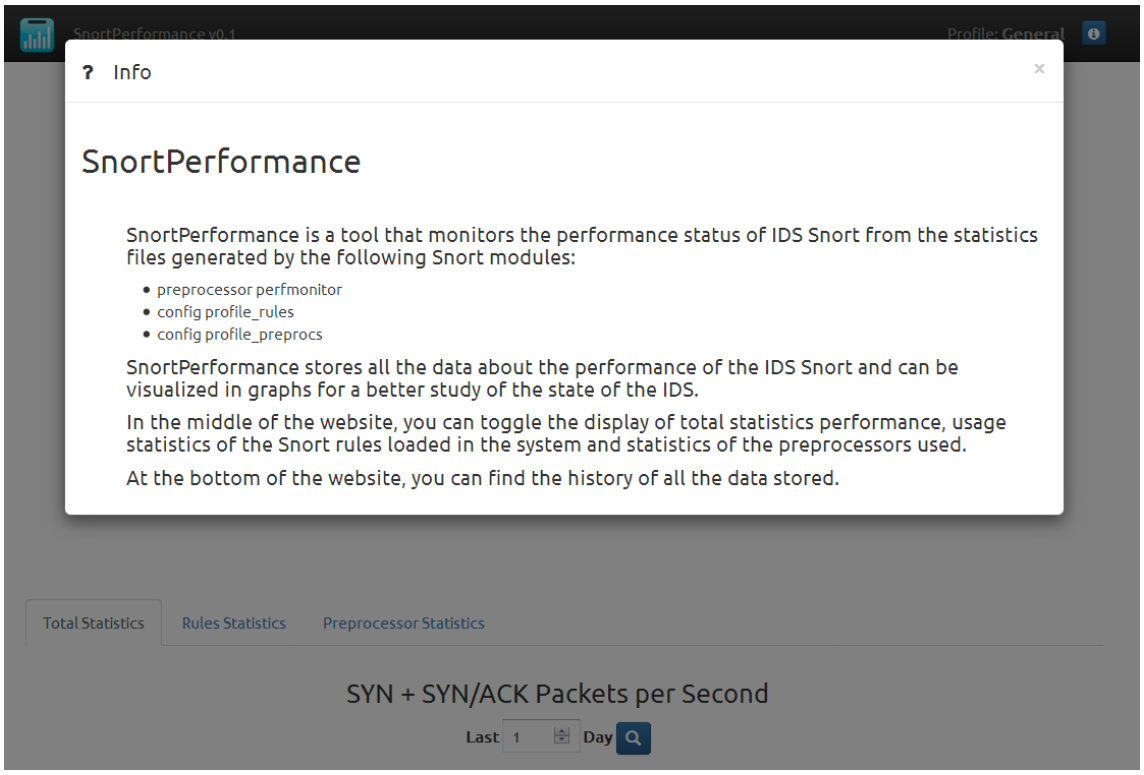


Ilustración 29. Cuadro de información de la herramienta

Traducido al español:

"SnortPerformance es una herramienta que monitoriza el estado de rendimiento del IDS Snort a partir de los archivos de estadísticas generados por los siguientes módulos de Snort:

- preprocesador perfmonitor
- config profile_rules
- config profile_preprocs

SnortPerformance almacena todos los datos sobre el rendimiento del IDS Snort y pueden ser visualizados en gráficas para un mejor estudio del estado del IDS.

En la parte central de la página web, se puede alternar la visualización del rendimiento de las estadísticas globales, las estadísticas de las reglas de Snort cargadas en el sistema y las estadísticas de los preprocesadores utilizados.

En la parte inferior de la página web, puede encontrar el historial de todos los datos almacenados."

Pasados 5 minutos, tal y como se había configurado Snort, éste actualiza su registro de datos sobre el rendimiento del sistema IDS, y el agente detecta esta actualización y envía los datos a la aplicación, la cual genera las gráficas correspondientes a los valores registrados:

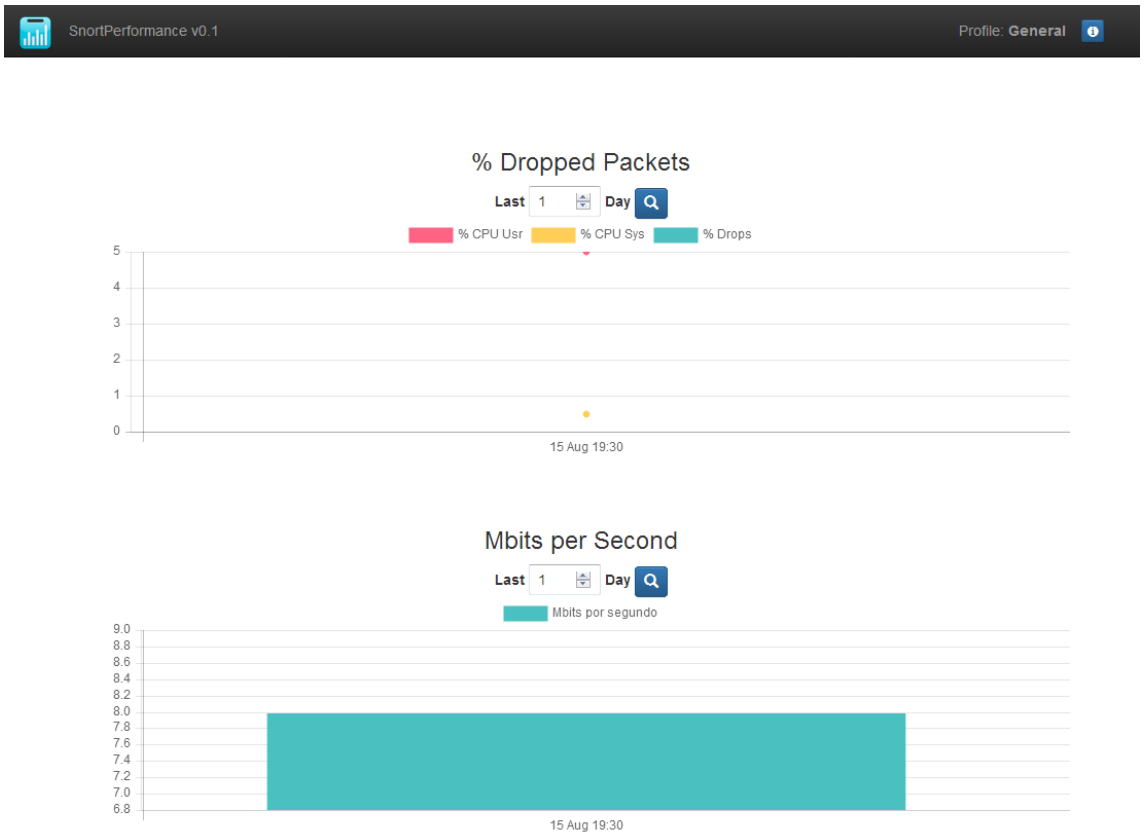


Ilustración 30. Gráficas 1 SnortPerformance

Si se coloca el cursor sobre los datos de la gráfica se muestra un recuadro con los valores exactos de las variables del dato señalado, como se puede ver a continuación:

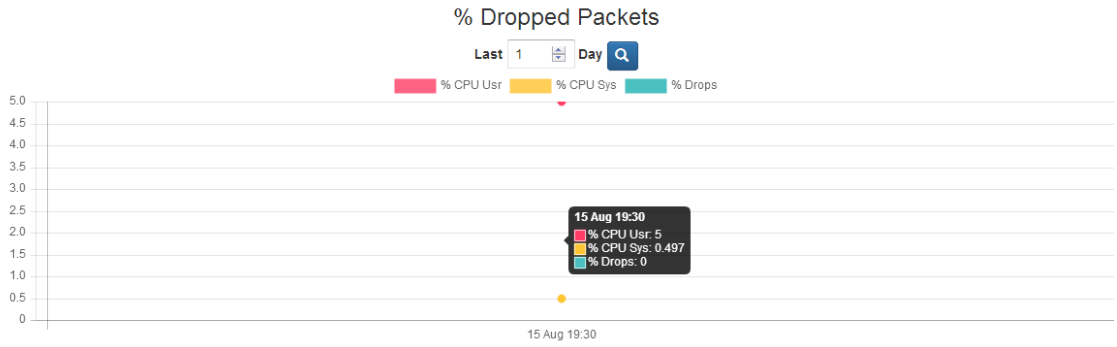


Ilustración 31. Valores de los datos sobre la gráfica

Siguiendo por la interfaz web, se encuentran las siguientes gráficas:

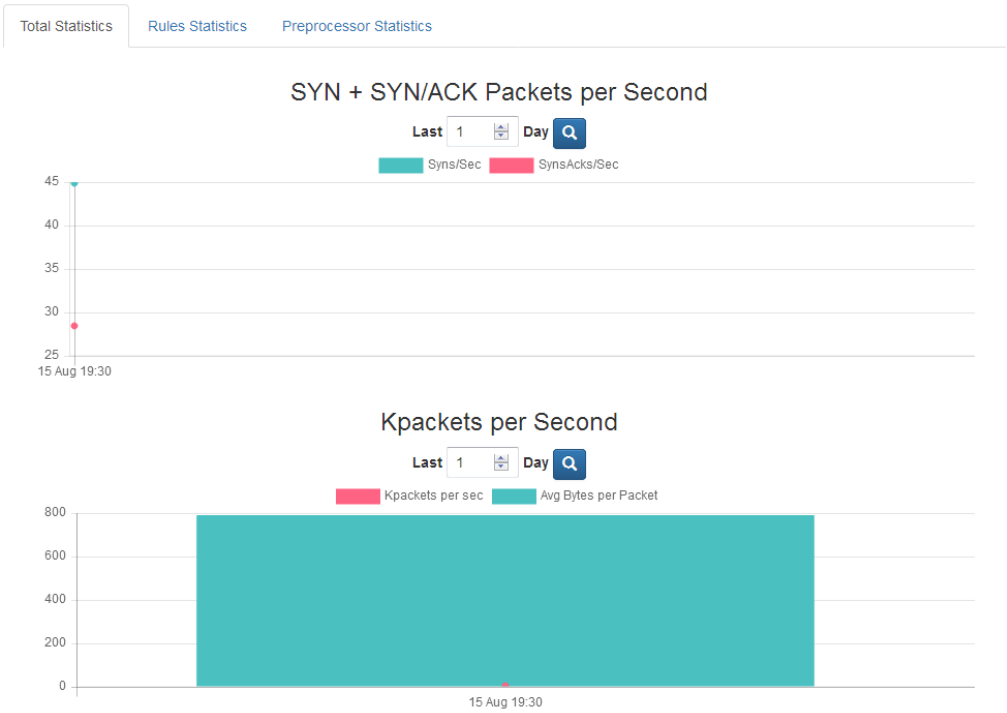


Ilustración 32. Gráficas 2 SnortPerformance

Otra funcionalidad de las gráficas y que puede ser de gran utilidad para el analista, es la opción de deshabilitar parámetros de las mismas. Así pues, en la gráfica anterior parece que el día 15 de Agosto a las 19:30 horas se registraron cero "Kpackets per sec", pero, como se puede ver en la figura siguiente, si deshabilitamos el parámetro "Avg Bytes per Packet", el cual tiene un valor muy alto, se observa que se registraron aproximadamente tres "Kpackets per sec":

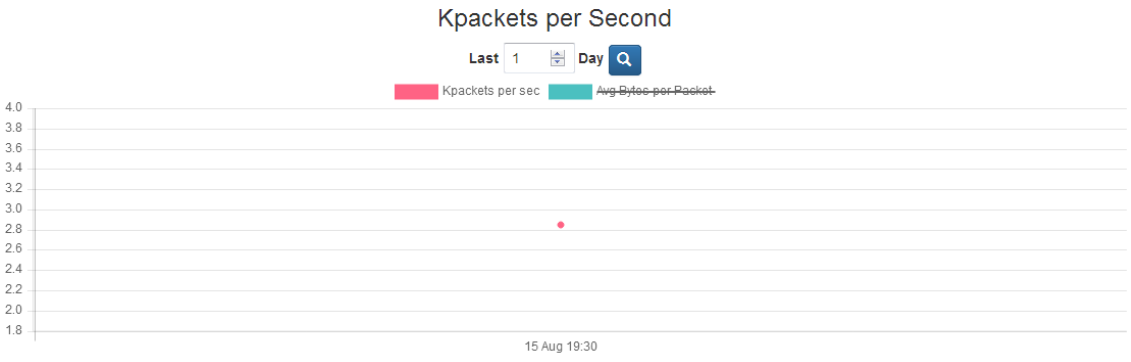


Ilustración 33. Deshabilitación de parámetros sobre las gráficas.

Si se sigue avanzando por la interfaz, se llega al final de la misma, dónde se observa, abajo del todo, un listado de los datos registrados, aunque por el momento sólo se ha recibido 1 dato:

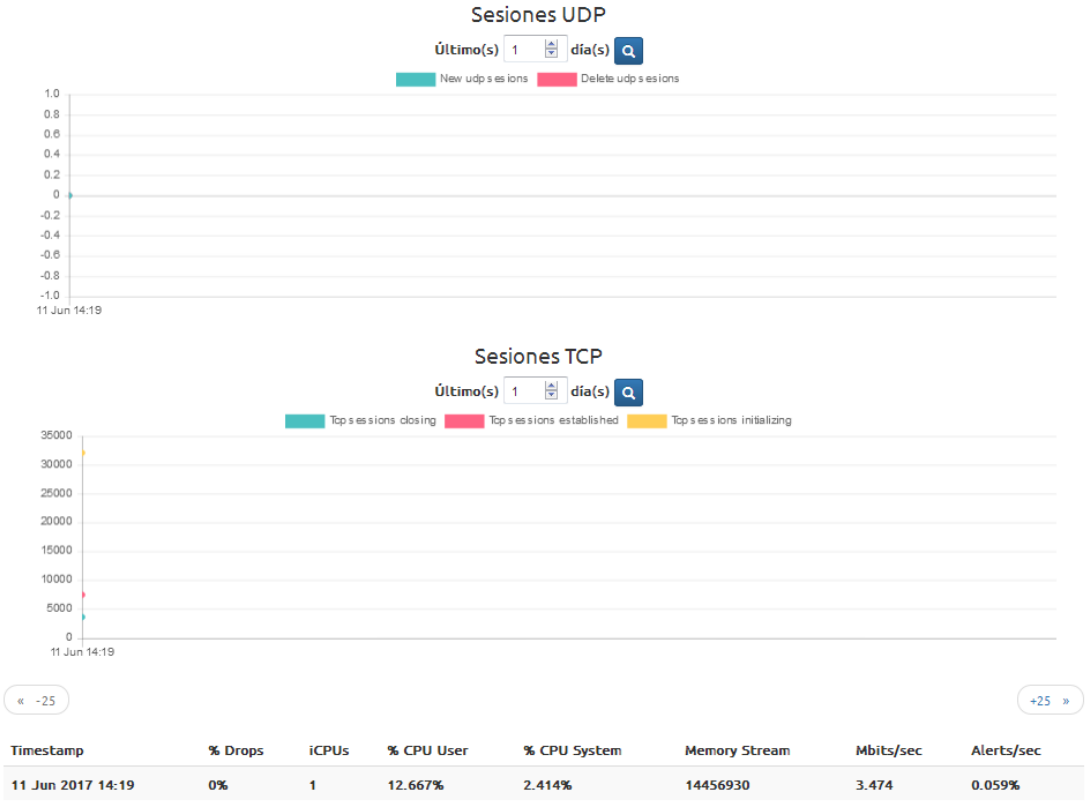


Ilustración 34. Gráficas 3 SnortPerformance

Sin embargo, las pestañas “Rules Statistics” y “Preprocessor Statistics” siguen vacías, ya que la sesión de Snort aún no ha terminado y no se han generado los registros correspondientes al rendimiento de las reglas y preprocesadores:

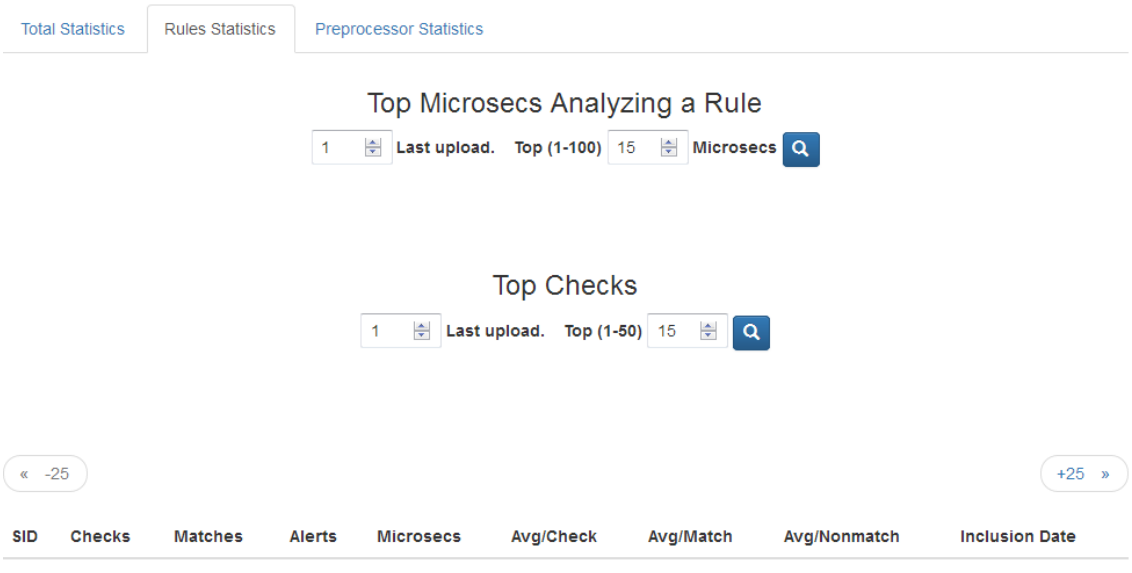


Ilustración 35. Interfaz estadísticas de reglas sin datos

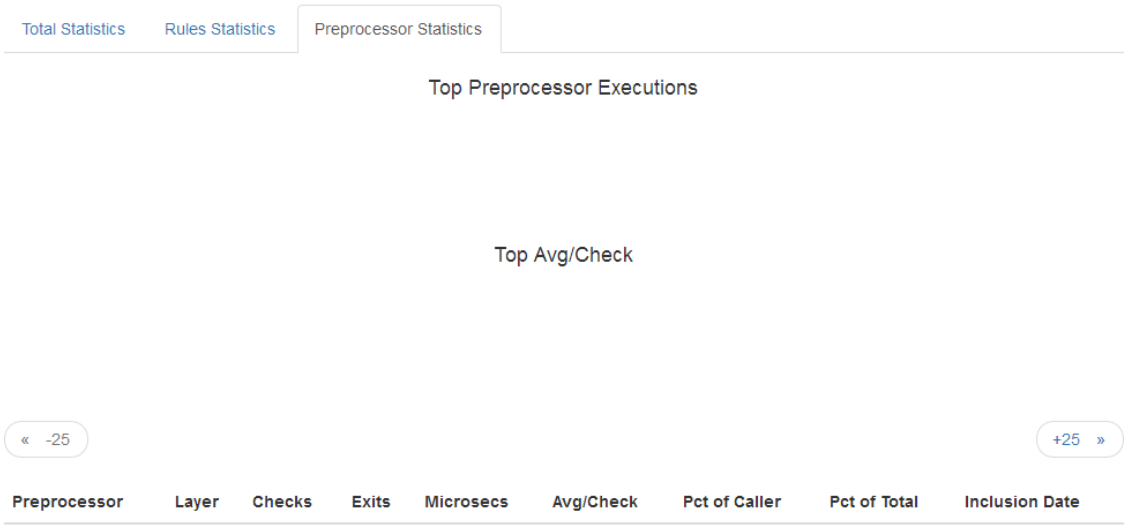


Ilustración 36. Interfaz estadísticas de preprocesadores sin datos

Pasado un tiempo, cuando llegue la hora programada en el *crontab*, se reiniciará la sesión de Snort, generando los 2 archivos de estadísticas, tanto de reglas, como de los preprocesadores (en este caso, se ha ejecutado a mano el reinicio de Snort para no tener que esperar a la hora programada). Cuando llegue el momento indicado en el gestor de tareas, se ejecutarán los programas creados para enviar esta información a la herramienta. Así se mostraría la aplicación web una vez recibidos los datos:

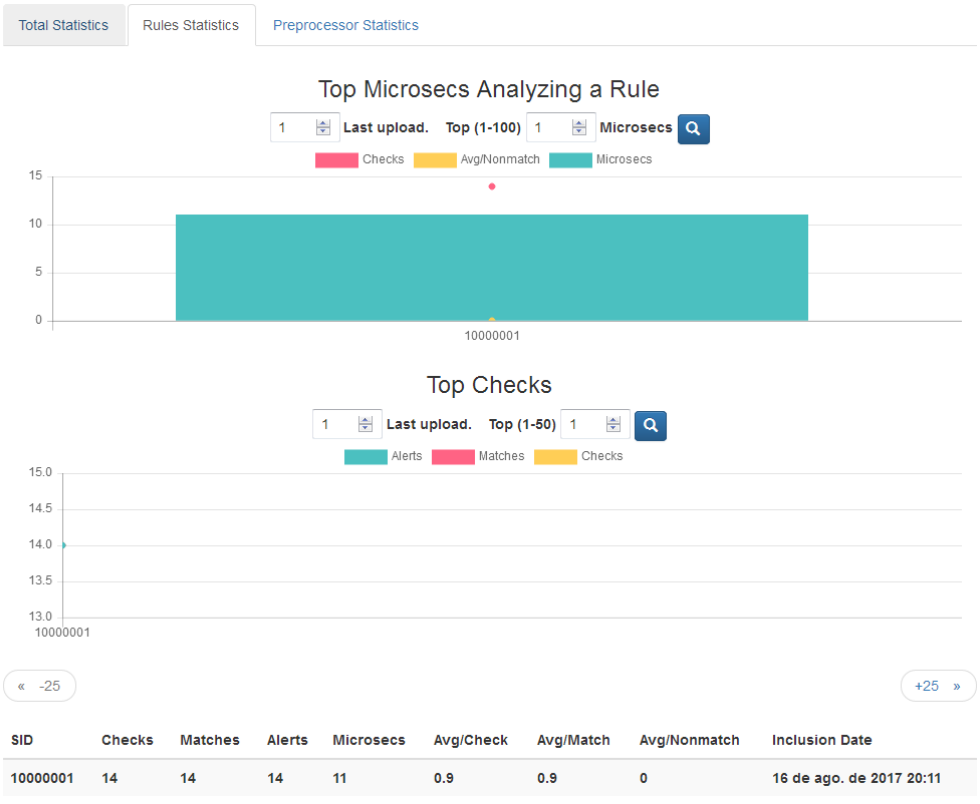


Ilustración 37. Gráficas estadísticas e histórico de datos de reglas

Para esta prueba se ha creado una regla, con identificador (*sid*) 10000001, para que Snort genere alertas cuando alguien realiza un *ping* al servidor. Como se ha podido observar en la figura de arriba, se ha hecho que se generen 14 alertas.

A continuación se muestran las gráficas relacionadas con el uso de los preprocesadores en la última sesión de Snort y debajo de las gráficas, como se puede ver la ilustración 39, el histórico de todos los valores estadísticos, de más reciente a más antiguo, de los preprocesadores utilizados:

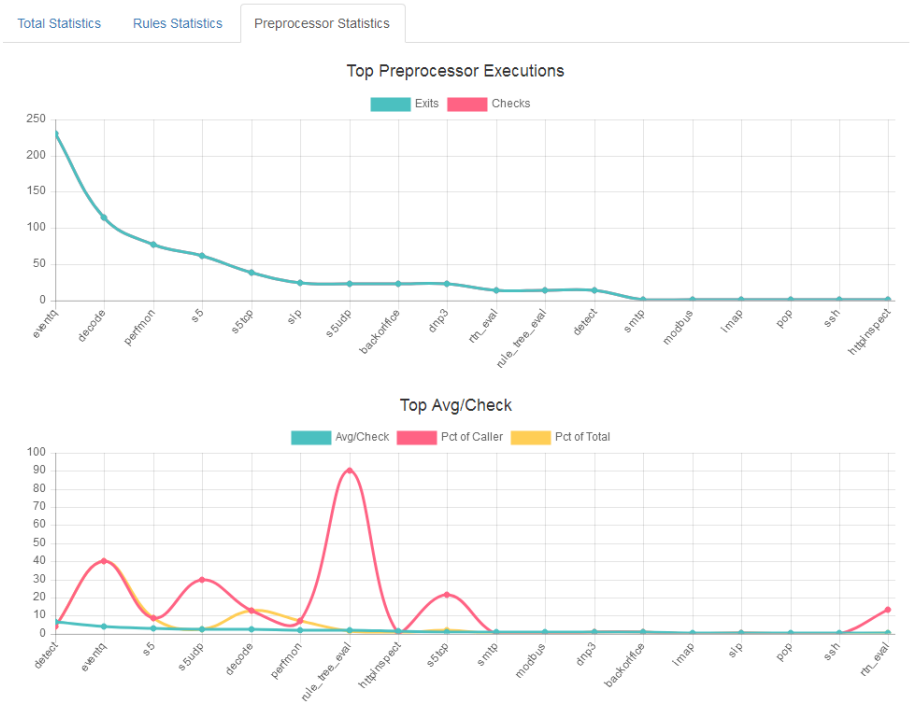


Ilustración 38. Gráficas de estadísticas de preprocesadores

« -25 +25 »

Preprocessor	Layer	Checks	Exits	Microsecs	Avg/Check	Pct of Caller	Pct of Total	Inclusion Date
rtn_eval	2	14	14	3	0.28	13.05	0.18	16 de ago. de 2017 20:13
ssh	0	1	1	0	0.42	0.02	0.02	16 de ago. de 2017 20:13
pop	0	1	1	0	0.49	0.02	0.02	16 de ago. de 2017 20:13
sip	0	24	24	11	0.49	0.55	0.55	16 de ago. de 2017 20:13
imap	0	1	1	0	0.5	0.02	0.02	16 de ago. de 2017 20:13
backorifice	0	23	23	13	0.61	0.64	0.64	16 de ago. de 2017 20:13
dnp3	0	23	23	16	0.7	0.74	0.74	16 de ago. de 2017 20:13
modbus	0	1	1	0	0.85	0.04	0.04	16 de ago. de 2017 20:13
smtp	0	1	1	0	0.93	0.04	0.04	16 de ago. de 2017 20:13
httpinspect	0	1	1	1	1.45	0.07	0.07	16 de ago. de 2017 20:13
rule_tree_eval	2	14	14	26	1.92	90.29	1.24	16 de ago. de 2017 20:13
perfmon	0	77	77	148	1.93	6.87	6.87	16 de ago. de 2017 20:13
decode	0	115	115	270	2.35	12.5	12.5	16 de ago. de 2017 20:13
s5tcp	1	38	38	39	1.05	21.37	1.84	16 de ago. de 2017 20:13
s5udp	1	23	23	55	2.43	29.97	2.59	16 de ago. de 2017 20:13
s5	0	61	61	186	3.06	8.63	8.63	16 de ago. de 2017 20:13
eventq	0	230	230	862	3.75	39.87	39.87	16 de ago. de 2017 20:13
detect	0	14	14	88	6.3	4.07	4.07	16 de ago. de 2017 20:13

Ilustración 39. Histórico de datos estadísticos de preprocesadores

Para entender mejor como está diseñada la web, a continuación se ha creado un mapa con todas las capturas anteriores ubicadas en su posición correspondiente de la página web:

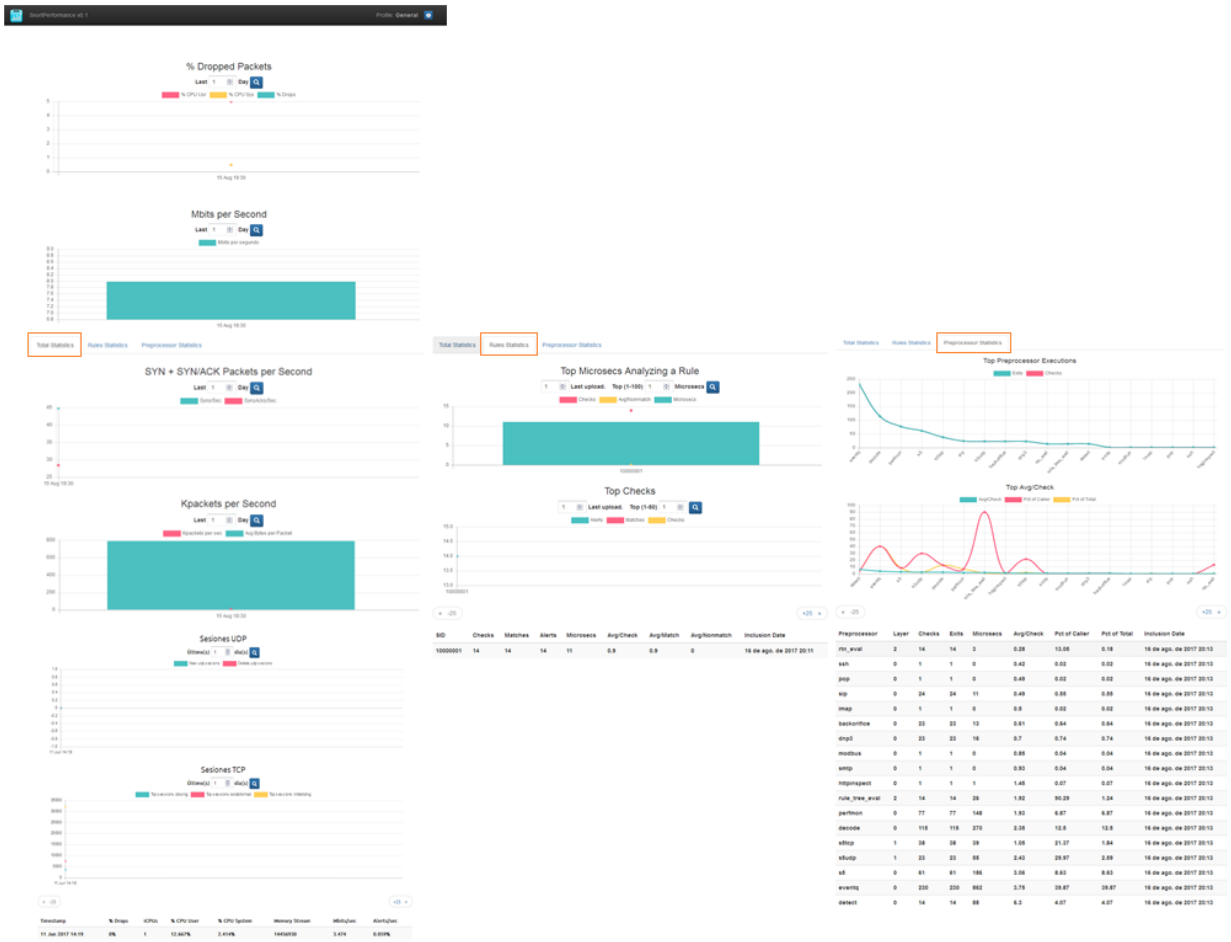


Ilustración 40. Diseño de la aplicación web

Remarcadas con un recuadro de color naranja se encuentran las 3 pestañas por las que se puede ir alternando: *Total Statistics*, *Rules Statistics* y *Preprocessor Statistics*.

Con los resultados obtenidos, se habrían cumplido 3 de los 4 requisitos propuestos en el apartado 3 de Análisis:

Requisito	**Descripción**
R1	Recopilar todos los datos estadísticos
R2	Mostrar los datos recopilados en gráficas útiles
R3	Mantener en la interfaz gráfica un histórico de los datos estadísticos
R4	Actualización de gráficas en tiempo real sin recargar la web

Se propone este requisito para trabajos futuros.

6. Pruebas con Datos Reales

Se ha conseguido una muestra de datos estadísticos reales de un sistema IDS Snort en producción dentro de una organización. A continuación, se muestra cómo quedaría la aplicación web diseñada con esta muestra real:

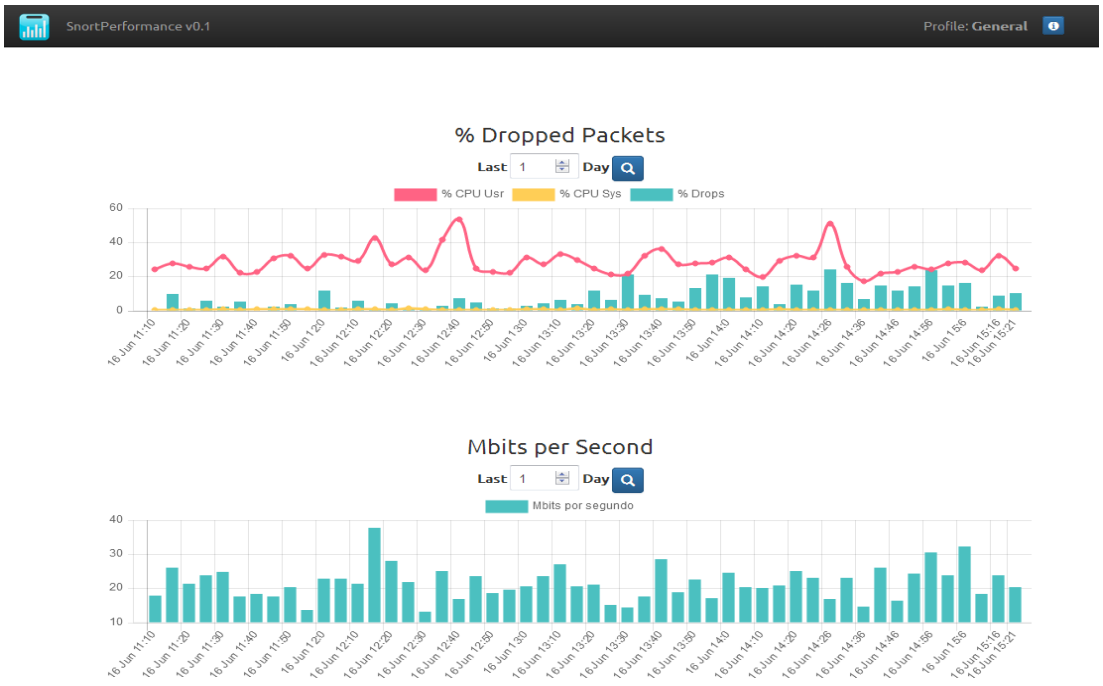


Ilustración 41. Gráfica 1 SnortPerformance con datos reales

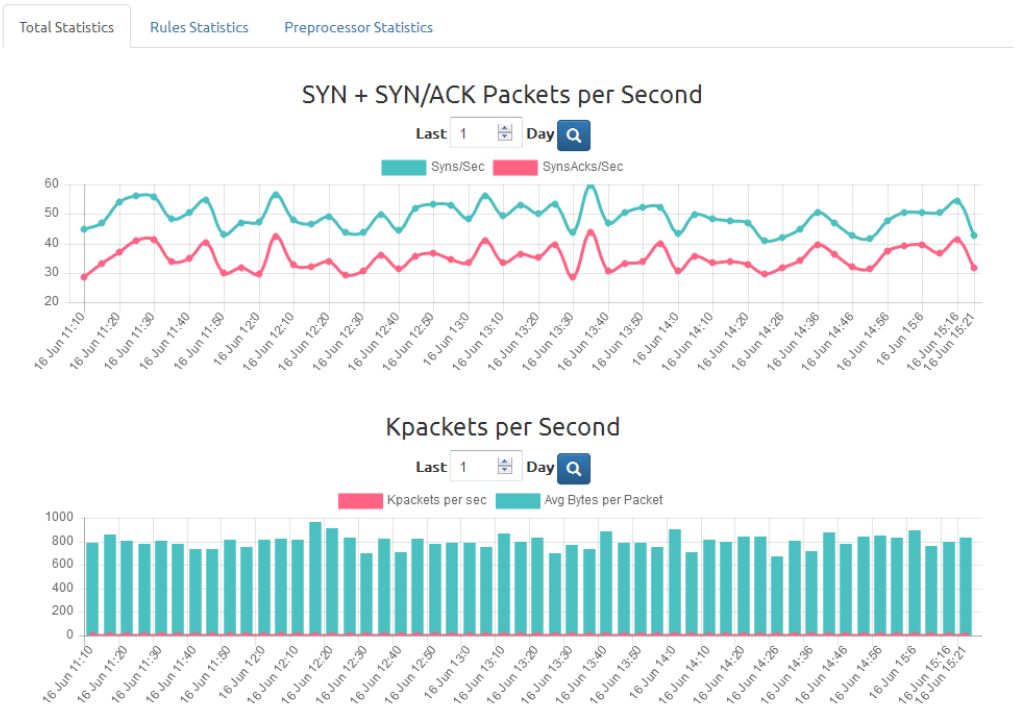


Ilustración 42. Gráfica 2 SnortPerformance con datos reales

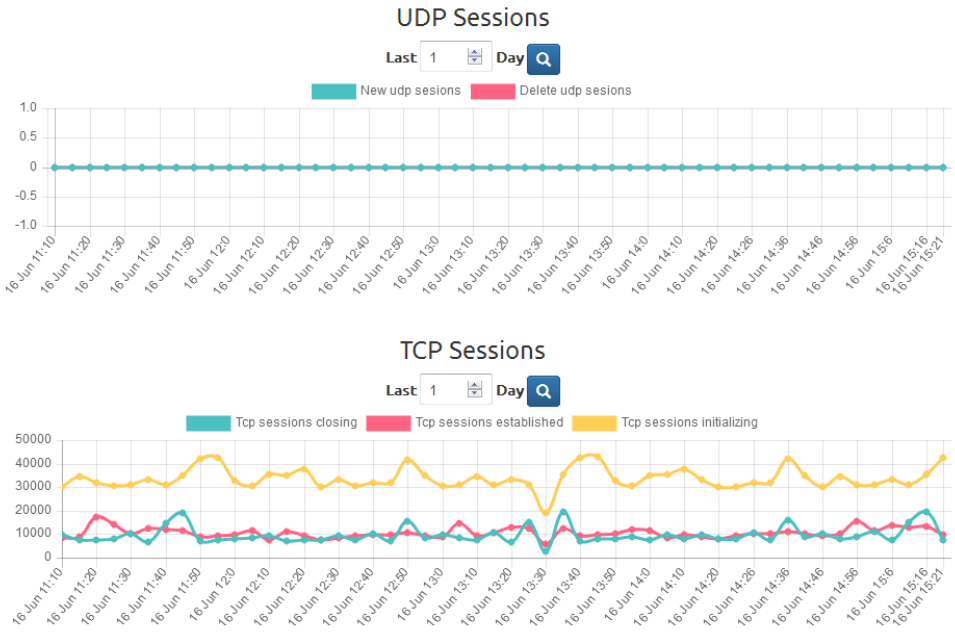


Ilustración 43. Gráfica 3 SnortPerformance con datos reales

« -25 +25 »

Timestamp	% Drops	iCPUs	% CPU User	% CPU System	Memory Stream	Mbits/sec	Alerts/sec
16 Jun 2016 15:21	10.191%	1	24.433%	0.435%	23781564	20.247	0.09%
16 Jun 2016 15:16	8.936%	1	32.203%	0.605%	27362965	23.874	0.06%
16 Jun 2016 15:11	2.5%	1	23.904%	0.348%	24456326	18.453	0.07%
16 Jun 2016 15:6	16.149%	1	27.945%	0.171%	18397431	32.317	0.131%
16 Jun 2016 15:1	14.809%	1	27.5%	0.681%	30401598	23.724	0.08%
16 Jun 2016 14:56	23.793%	1	24.403%	0.26%	21797082	30.6	0.089%
16 Jun 2016 14:51	13.989%	1	25.651%	0.147%	22628156	24.361	0.08%
16 Jun 2016 14:46	11.598%	1	22.91%	0.369%	19107621	16.4	0.129%
16 Jun 2016 14:41	14.803%	1	21.88%	0.234%	16431963	26.036	0.05%
16 Jun 2016 14:36	6.705%	1	17.267%	0.139%	19239850	14.744	0.069%
16 Jun 2016 14:31	16.108%	1	25.478%	0.174%	21549291	23.114	0.119%
16 Jun 2016 14:26	24.309%	1	50.899%	0.093%	30540006	16.888	0.181%
16 Jun 2016 14:25	11.931%	1	31.389%	0.355%	42010471	23.087	0.06%
16 Jun 2016 14:20	15.029%	1	32.115%	0.389%	28381326	25.084	0.076%
16 Jun 2016 14:15	3.629%	1	29.125%	0.569%	29310304	20.91	0.088%
16 Jun 2016 14:10	14.36%	1	19.569%	0.217%	17249573	20.216	0.04%
16 Jun 2016 14:5	7.781%	1	24.222%	0.453%	20543069	20.232	0.14%
16 Jun 2016 14:0	18.976%	1	31.3%	0.545%	28496883	24.625	0.05%
16 Jun 2016 13:55	21.235%	1	28.35%	0.327%	24888684	17.222	0.05%
16 Jun 2016 13:50	13.485%	1	27.621%	0.241%	37177713	22.635	0.11%
16 Jun 2016 13:45	5.391%	1	27.14%	0.562%	26039640	18.957	0.08%
16 Jun 2016 13:40	7.064%	1	36.03%	0.637%	24570513	28.512	0.13%
16 Jun 2016 13:35	9.378%	1	31.902%	0.871%	25004087	17.689	0.03%
16 Jun 2016 13:30	21.082%	1	21.486%	0.349%	8373427	14.463	0.045%
16 Jun 2016 13:25	6.185%	1	21.414%	0.602%	24208401	15.107	0.04%

Ilustración 44. Histórico de datos estadísticos reales

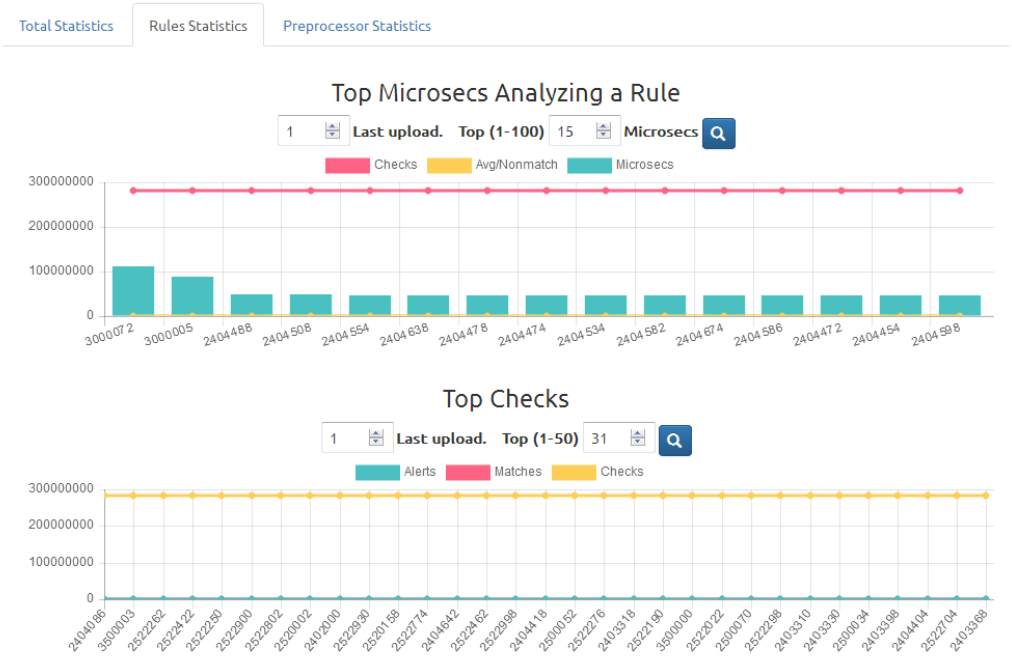


Ilustración 45. Gráficas estadísticas de reglas con datos reales

« -25 +25 »

SID	Checks	Matches	Alerts	Microsecs	Avg/Check	Avg/Match	Avg/Nonmatch	Inclusion Date
2004461	2	0	0	154180	77090.1	0	77090.1	11 de jun. de 2017 22:04
2808577	71416195	0	0	14696502	0.2	0	0.2	11 de jun. de 2017 22:04
2004378	58	0	0	387367	6678.8	0	6678.8	11 de jun. de 2017 22:04
2003790	3659	0	0	189510	51.8	0	51.8	11 de jun. de 2017 22:04
2006127	1	0	0	0	0.5	0	0.5	11 de jun. de 2017 22:04
2522900	281798775	0	0	35734878	0.1	0	0.1	11 de jun. de 2017 22:04
2815802	3212	0	0	1664	0.5	0	0.5	11 de jun. de 2017 22:04
2019745	39	0	0	16191	415.2	0	415.2	11 de jun. de 2017 22:04
2813060	75	0	0	564779	7530.4	0	7530.4	11 de jun. de 2017 22:04
2523015	6897880	0	0	6601727	1	0	1	11 de jun. de 2017 22:04
2017246	17	0	0	5	0.3	0	0.3	11 de jun. de 2017 22:04
2811901	53930	0	0	34097	0.6	0	0.6	11 de jun. de 2017 22:04
2002976	3	0	0	80346	26782	0	26782	11 de jun. de 2017 22:04
2008488	2	0	0	41587	20793.6	0	20793.6	11 de jun. de 2017 22:04
2500023	6897880	0	0	8736696	1.3	0	1.3	11 de jun. de 2017 22:04
2016553	1983	0	0	154634	78	0	78	11 de jun. de 2017 22:04
2021950	8	0	0	1156976	144622.1	0	144622.1	11 de jun. de 2017 22:04
2014099	1734	0	0	6077	3.5	0	3.5	11 de jun. de 2017 22:04
2808804	4	0	0	57956	14489	0	14489	11 de jun. de 2017 22:04
2021044	6598	0	0	10515	1.6	0	1.6	11 de jun. de 2017 22:04
3500000	281798775	0	0	35507270	0.1	0	0.1	11 de jun. de 2017 22:04
2522022	281798775	0	0	36097184	0.1	0	0.1	11 de jun. de 2017 22:04
2809928	172	0	0	184407	1072.1	0	1072.1	11 de jun. de 2017 22:04
3000022	100181887	0	0	25475892	0.3	0	0.3	11 de jun. de 2017 22:04
2807762	5167	0	0	247188	47.8	0	47.8	11 de jun. de 2017 22:04

Ilustración 46. Histórico de datos estadísticos reales de reglas

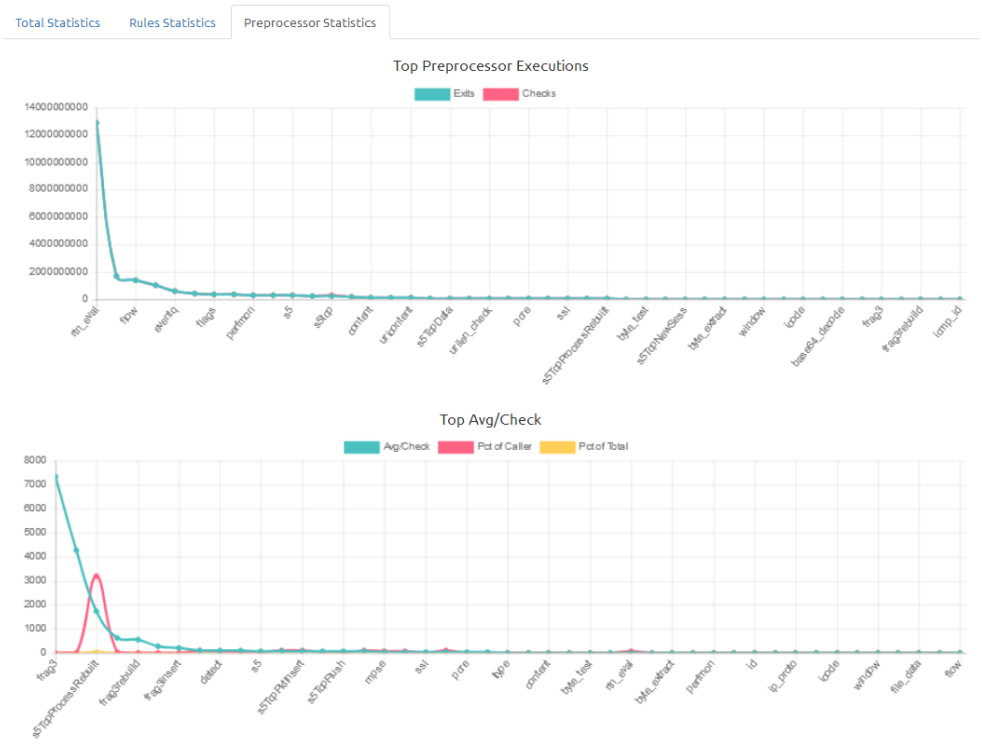


Ilustración 47. Gráficas estadísticas de preprocesadores con datos reales

« -25 +25 »

Preprocessor	Layer	Checks	Exits	Microsecs	Avg/Check	Pct of Caller	Pct of Total	Inclusion Date
dns	0	3188126	3188126	258598	0.08	0	0	11 de jun. de 2017 22:04
ssh	0	60947216	60937223	7350237	0.12	0.01	0.01	11 de jun. de 2017 22:04
perfmon	0	293163440	293163440	71947950	0.25	0.13	0.13	11 de jun. de 2017 22:04
eventq	0	578551596	578551596	211220603	0.37	0.39	0.39	11 de jun. de 2017 22:04
decode	0	285771719	285771719	679251882	2.38	1.26	1.26	11 de jun. de 2017 22:04
ssl	0	10300147	10300147	140170878	13.61	0.26	0.26	11 de jun. de 2017 22:04
s5TcpBuildPacket	4	9769260	9769260	379647259	38.86	87.17	0.7	11 de jun. de 2017 22:04
s5TcpProcessRebuilt	4	8056227	8056227	13849768542	1719.14	3179.96	25.59	11 de jun. de 2017 22:04
s5TcpFlush	3	10396628	10396628	435532810	41.89	2.44	0.8	11 de jun. de 2017 22:04
s5TcpPktInsert	4	42050634	37943452	2081919842	49.51	99.43	3.85	11 de jun. de 2017 22:04
s5TcpData	3	44270724	44270724	2093793739	47.3	11.75	3.87	11 de jun. de 2017 22:04
s5TcpState	2	194273823	194273823	17818535298	91.72	96.07	32.93	11 de jun. de 2017 22:04
s5TcpNewSess	2	3017826	3017826	1902111608	630.29	10.26	3.51	11 de jun. de 2017 22:04
s5tcp	1	273726299	194273823	18547647690	67.76	93.17	34.27	11 de jun. de 2017 22:04
s5	0	273726299	273726299	19906267433	72.72	36.79	36.79	11 de jun. de 2017 22:04
httpinspect	0	131635216	131635216	10998549734	83.55	20.32	20.32	11 de jun. de 2017 22:04
rtn_eval	2	12908648220	12908648220	11115434868	0.86	72.1	20.54	11 de jun. de 2017 22:04
flags	3	351183208	351183208	6432931	0.02	0.04	0.01	11 de jun. de 2017 22:04
flow	3	1379782143	1379782143	32893423	0.02	0.22	0.06	11 de jun. de 2017 22:04
file_data	3	14533891	14533891	720750	0.05	0	0	11 de jun. de 2017 22:04
base64_data	3	8074	8074	479	0.06	0	0	11 de jun. de 2017 22:04
window	3	44114	44114	3539	0.08	0	0	11 de jun. de 2017 22:04
icode	3	23309	23309	2052	0.09	0	0	11 de jun. de 2017 22:04
ip_proto	3	4526285	4526285	530729	0.12	0	0	11 de jun. de 2017 22:04
urilen_check	3	28231394	28231394	3726399	0.13	0.02	0.01	11 de jun. de 2017 22:04

Ilustración 48. Histórico de datos estadísticos reales de preprocesadores

6.1. Detectando ataques e intrusiones con la herramienta

Observando las gráficas del rendimiento del IDS Snort proporcionadas por la herramienta diseñada, es posible detectar ataques que el propio IDS no alertaría. En este punto, se van a presentar dos posibles ataques que podrían ser detectados con un simple vistazo de la aplicación:

1. Ataque DoS o Denegación de Servicio contra la organización: analizando los resultados de la gráfica, se puede detectar un posible ataque de denegación de servicio contra un recurso de la organización.

   Un ataque DoS, o de Denegación de Servicio, trata de sobrecargar los recursos computacionales del sistema afectado con el objetivo de dejarlo inaccesible para los usuarios legítimos.

   A continuación, se propone una situación en la que se podría dar este tipo de ataque:

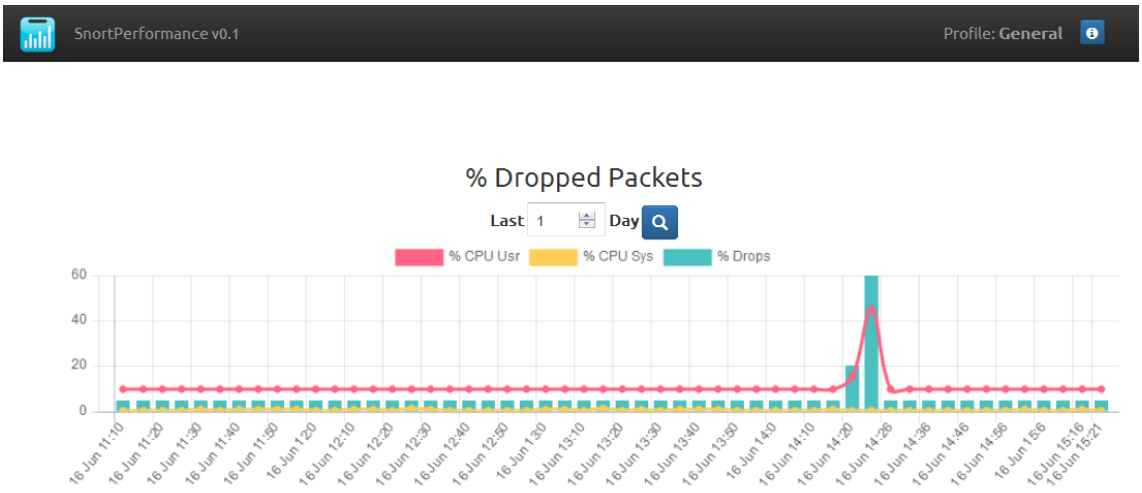


Ilustración 49. Detectando un posible DoS con SnortPerformance

Como se puede observar en la figura anterior, entre las 11:10 horas y las 14:20 horas, el sistema IDS tiene aproximadamente una tasa de paquetes sin poder analizar del 5%, sin embargo, algo ha pasado de las 14:20 horas hasta las 14:26 horas, que ha aumentado esta tasa del 5% al 60%. Es muy probable que algún servicio expuesto a internet de la organización haya sido víctima de un ataque DoS, y que el aumento de paquetes descartados por el IDS se haya producido por las múltiples peticiones originadas por los atacantes con el objetivo de vulnerar este recurso.

Se tendría que analizar este incidente y determinar si se trata de un ataque recibido, lo cual habría que tratar y llevar a cabo las acciones correspondientes, o simplemente, es un falso positivo debido a una auditoría interna con alguna herramienta automática, la cual generaría mucho tráfico, por ejemplo.

2. Equipo infectado realizando un DDoS, o DoS Distribuido, hacia el exterior: otro caso que se podría detectar con la herramienta, sería la infección de un equipo interno con algún tipo de virus informático para realizar ataques DoS distribuidos contra algún servicio ajeno a la organización.

   Este tipo de ataque de denegación de servicio es el más habitual por su eficacia y sencillez tecnológica, el cual divide el ataque entre todos los equipos infectados, haciendo éste más potente y difícil de detectar por la cantidad de direcciones IP implicadas.

   En la siguiente figura se puede ver un comportamiento sospechoso que podría indicar este tipo de infección en algún equipo:

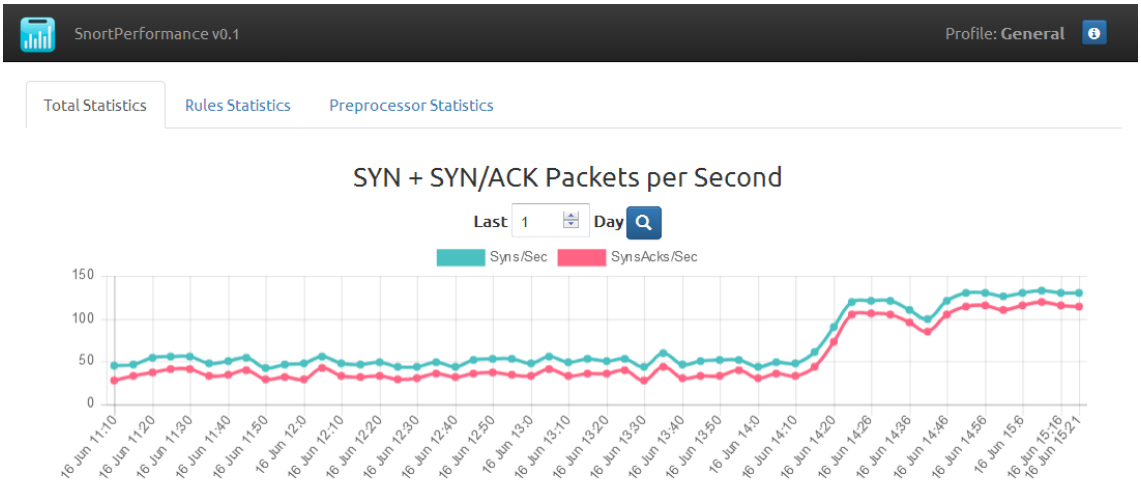


Ilustración 50. Detectando una posible infección con SnortPerformance

Como se puede observar, entre las 11:10 horas y las 14:20 horas, el sistema IDS tiene aproximadamente una cantidad entre 30 y 50 paquetes SYN y SYN/ACK por segundo, sin embargo, algo ha pasado a las 14:20 horas que ha aumentado este valor de 30-50 paquetes a 100-140, y no tiende a bajar.

Los paquetes implicados, SYN y SYN/ACK, son los encargados del establecimiento de una conexión. Un equipo cliente que quiere realizar una conexión con un servidor, le enviará un paquete SYN a éste, y si la conexión es aceptada, el servidor responderá al cliente con un paquete SYN/ACK.

En este caso, se debería de estudiar este incidente y comprobar qué está originando esa cantidad tan elevada de peticiones respecto al comportamiento normal de la organización. Al igual que en el caso anterior, se tendría que analizar y determinar si se trata de una infección en un equipo, el cual está participando en un ataque DDoS, o si se trata de un falso positivo.

7. Conclusión

Cada vez son más comunes los ciber ataques hacia las organizaciones con el fin de robar información de las mismas y sacar un beneficio de ello, vendiéndola a la competencia o pidiendo un rescate por los datos robados.

Para estar alerta de estas intrusiones, se recomienda la instalación de sistemas de detección de intrusos o, más comúnmente llamados, IDS. Un sistema IDS se encargaría de alertar en tiempo real de los ataques recibidos o de comportamientos anómalos detectados.

Con el objetivo de mantener estos sistemas optimizados y funcionando correctamente se ha diseñado una herramienta para facilitar la monitorización, el almacenamiento y estudio de los datos estadísticos generados por el IDS Snort.

El objetivo de la aplicación desarrollada es monitorizar el rendimiento de Snort mediante una interfaz web para estudiar el estado del mismo con el fin de optimizar este sistema para que sea capaz de mantener una alta tasa de detección de los posibles ciber ataques que se reciban en una organización. Además, analizando las gráficas generadas por la herramienta, también es posible detectar ataques o infecciones en equipos de la empresa donde esté implantada.

En definitiva, se ha desarrollado una aplicación web base para ayudar a monitorizar el rendimiento de los sistemas IDS Snort, pero con varias mejoras y trabajos futuros para convertir la herramienta diseñada en una aplicación completa e indispensable para todas las organizaciones que utilicen Snort.

8. Trabajos futuros

A continuación se proponen una serie de mejoras para la aplicación desarrollada y ofrecer un servicio más completo y útil para el analista:

- Autenticación necesaria para acceder a las gráficas del rendimiento, para que dicha información no sea accesible por usuarios no autorizados.
- Actualización dinámica de las gráficas, mostrando datos estadísticos en tiempo real del IDS Snort sin recargar la página web.
- Poder monitorizar el rendimiento de diferentes interfaces de Snort.
- Acceso a datos históricos y hacerlos visibles también en las gráficas.
- Botones desplegables con consejos para mejorar el rendimiento de Snort y sacar el máximo partido a este sistema IDS.
- Generación de informes semanales con los datos más relevantes de la semana.
- Añadir una opción para subir un fichero con datos estadísticos de Snort y que sean mostrados en otra pestaña.
- Notificar, vía correo electrónico, cuando se detecte algún valor anómalo, superior o inferior al valor normal del dato.

9. Glosario/Diccionario

- **Ransomware**: virus informático que cifra la información del equipo infectado a cambio de un rescate.
- **Script**: archivo de órdenes.
- **Back-end**: motor, tecnologías en el lado del servidor.
- **Front-end**: interfaz, tecnologías en el lado del cliente.
- **Firewall**: sistema informático colocado entre una red privada e Internet, capaz de mejorar la seguridad de las comunicaciones, evitando conexiones no autorizadas.
- **Sniffer**: analizador de paquetes.
- **Crontab**: programa en Unix/Linux que permite la ejecución de comandos o scripts automáticamente.
- **JSON (JavaScript Object Notation)**: formato de texto ligero para el intercambio de datos.
- **Parsear**: análisis sintáctico de un documento o sentencia.

10. Referencias

- "The Practice of Network Security Monitoring", Richard Bejtlich, San Francisco, 2013.
- Blog de seguridad "Security Art Work" http://www.securityartwork.es/:
    - "IDS en el cortafuegos", Antonio Villalón, 2010: http://www.securityartwork.es/2010/06/29/ids-en-el-cortafuegos/
    - "Cuando un cerdo no es suficiente, monta una granja", Nelo Belda, 2010: http://www.securityartwork.es/2010/10/20/cuando-un-cerdo-no-es-suficiente-monta-una-granja/
    - "Cuando un cerdo no es suficiente, monta una granja (II)", Nelo Belda, 2010: http://www.securityartwork.es/2010/10/29/cuando-un-cerdo-no-es-suficiente-monta-una-granja-ii/
    - "Cuando un cerdo no es suficiente, monta una granja (III)", Nelo Belda, 2010: http://www.securityartwork.es/2010/11/19/cuando-un-cerdo-no-es-suficiente-monta-una-granja-iii/
    - "Evasión en IDS", Joaquin Moreno, 2010: http://www.securityartwork.es/2010/06/21/evasion-en-ids-i/
- Presentación "Sistemas de detección de intrusos", Antonio Villalón, Universidad Politécnica de Madrid, Mayo 2005.
- "Snort Manual", The Snort Project: http://manual-snort-org.s3-website-us-east-1.amazonaws.com/
- Blog de seguridad "Seguridad y Redes", Septiembre 2010: https://seguridadyredes.wordpress.com/2010/09/23/snort-formato-tipos-e-interpretacion-de-las-alertas-actualizacion/
- Blog de seguridad "LIONSEC", 2016: http://lionsec.net/blog/wp-content/uploads/2016/01/ids.gif
- Blog de seguridad "Sublime Robots", artículo "Configure Snort to Run as a NIDS", Noah Dietrich, Octubre 2015: http://sublimerobots.com/2015/12/snort-2-9-8-x-on-ubuntu-part-2/
- Blog de desarrolladores "GenbetaDev", artículo "MongoDB: qué es, cómo funciona y cuándo podemos usarlo (o no)", RUBENFA, Febrero 2014: https://www.genbetadev.com/bases-de-datos/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no
- Blog de seguridad "Guru de la Informática", artículo "Herramientas para afinar y complementar IDS Snort.", Álvaro Paz, Julio 2015: http://www.gurudelainformatica.es/2015/07/herramientas-para-afinar-y-complementar.html

11. ANEXO

11.1. Instalación Snort en el cliente

11.1.1. Sobre esta guía

En la siguiente guía instalaremos Snort 2.9.8.3 como un NIDS (Network Intrusion Detection System) en un servidor con un sistema operativo Ubuntu 14.04 Server LTS x64.

Aunque existen proyectos que instalan Snort automáticamente en la máquina, como por ejemplo Autosnort (https://github.com/da667/Autosnort), en esta guía se instalará Snort paso a paso para entender mejor su configuración y funcionamiento.

Además, se asume que estamos registrados en el sistema como usuario normal, y tendremos que correr algunos comandos con permisos de administrador con el comando sudo. Esto ayudará a identificar qué comandos requieren permisos de administrador y cuáles no. También se creará un usuario sin privilegios llamado *snort* que lo usaremos para lanzar la aplicación cuando estén creados los servicios, siguiendo las mejores prácticas de seguridad.

11.1.2. Instalando Ubuntu

Una vez instalado Ubuntu en el servidor e iniciado sesión, comprobaremos que tenemos acceso a internet y actualizaremos el sistema. Una vez actualizado, reiniciaremos el equipo para asegurarnos que todos los parches se han aplicado correctamente:

```
# Actualizamos el sistema
sudo apt-get update
sudo apt-get dist-upgrade –y
# Y reiniciamos para aplicar los cambios
sudo reboot
```

11.1.3. Configuración de la tarjeta de red

Como se indica en el manual de Snort: http://manual.snort.org/node7.html:
“Algunas tarjetas de red tienen características que pueden afectar Snort. Dos de estas características se denominan “Large Receive Offload” (lro) o “Descarga de Recepción Grande” y “Generic Receive Offload” (gro) o “Descarga de Recepción Genérica”. Con estas características habilitadas, la tarjeta de red realiza el reensamblaje de paquetes antes de que sean procesados por el núcleo.”

Por lo anterior, desactivaremos LRO y GRO de nuestra tarjeta de red. Para ello, usaremos el comando *ethtool* en el archivo de configuración de la interfaz de red /etc/network/interfaces.

Editaremos el archivo anterior con el editor de texto vi, por ejemplo:

```
sudo vi /etc/network/interfaces
```

Y añadiremos las siguientes dos líneas al final:

```
post-up ethtool -K eth0 gro off
post-up ethtool -K eth0 lro off
```

Guardamos el archivo, reiniciamos el servidor y comprobamos que LRO y GRO están deshabilitados:

```
proyectosnort@proyectoSnort:~$ ethtool -k eth0 | grep receive-offload
generic-receive-offload: off
large-receive-offload: off [fixed]
proyectosnort@proyectoSnort:~$
```

11.1.4. Instalando los Pre-Requisitos de Snort

Snort trabaja con cuatro pre-requisitos principales:

- PCAP (libpcap-dev) disponible en el repositorio de Ubuntu: librería que proporciona funciones para la captura de paquetes en la capa de red.
- PCRE (libpcre3-dev) disponible en el repositorio de Ubuntu: librería que proporciona funciones para soportar expresiones regulares cuya sintaxis y semántica sean tan parecidas como sea posible a las del lenguaje de Perl 5.
- Libdnet (libdumbnet-dev) disponible en el repositorio de Ubuntu: librería que proporciona funciones de red de bajo nivel.
- DAQ (https://www.snort.org/downloads) disponible en la página oficial de snort: librería que proporciona funciones para la adquisición de datos de paquetes de entrada o salida.

Primero instalaremos el paquete *build-essentials* el cual instalará las herramientas necesarias para construir nuestro software.

```
sudo apt-get install -y build-essential
```

Una vez instalado, instalaremos los pre-requisitos de Snort disponibles en el repositorio de Ubuntu:

```
sudo apt-get install -y libpcap-dev libpcre3-dev libdumbnet-dev
```

Y para acabar, instaleremos el pre-requisito DAQ, el cual también necesita de otros pre-requesitos, disponibles en el repositorio de Ubuntu:

```
sudo apt-get install -y bison flex
wget https://www.snort.org/downloads/snort/daq-2.0.6.tar.gz
tar -xvzf daq-2.0.6.tar.gz
cd daq-2.0.6
./configure
make
sudo make install
```

Adicionalmente, también se tendrá que instalar el paquete *zlibg* el cual es una biblioteca de compresión que nos permitirá configurar Snort correctamente.

```
sudo apt-get install -y zlib1g-dev
```

11.1.5. Instalando Snort

En este punto, ya tenemos la máquina preparada para instalar Snort. La versión de Snort que instalaremos será la 2.9.8.3, lanzada el 25 de Abril del 2016:

```
wget https://snort.org/downloads/snort/snort-2.9.8.0.tar.gz
tar -xvzf snort-2.9.8.0.tar.gz
cd snort-2.9.8.0
./configure --enable-sourcefire
make
sudo make install
```

La opción *--enable-sourcefire* nos ofrece la posibilidad de configurar Snort como lo está en Sourcefire [17]. También se pueden ver las diferentes opciones de compilado con ./configure --help .

A continuación, y para acabar con la instalación de Snort, ejecutaremos los siguientes dos comandos, para actualizar las librerías compartidas y crear un enlace simbólico al binario de Snort en /usr/sbin:

```
sudo ldconfig
sudo ln -s /usr/local/bin/snort /usr/sbin/snort
```

Por último, para comprobar que Snort funciona correctamente, podemos ejecutar Snort con la etiqueta -V. Si se ha instalado bien, la ejecución del binario con la opción -V debería mostrarnos algo similar a lo siguiente:

```
proyectosnort@proyectoSnort:~$ snort -V

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.8.3 GRE (Build 383)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2015 Cisco and/or its affiliates. All rights reser
ved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.5.3
           Using PCRE version: 8.31 2012-07-06
           Using ZLIB version: 1.2.8
```

11.1.6. Configurando Snort en Modo NIDS

Para no lanzar Snort como administrador, crearemos una cuenta sin privilegios y un grupo desde el que se lanzará el proceso de Snort (snort:snort). Además crearemos las carpetas y ficheros requeridos por Snort, con los permisos necesarios para que funcione correctamente:

```
# Crear el grupo y usuario para lanzar Snort:
sudo groupadd snort
sudo useradd snort -r -s /sbin/nologin -c SNORT_IDS -g snort

# Crear los directorios de Snort necesarios:
sudo mkdir /etc/snort
sudo mkdir /etc/snort/rules
```

[17] http://blog.snort.org/2011/09/snort-291-installation-guide-for-centos.html

```
sudo mkdir /etc/snort/rules/iplists
sudo mkdir /etc/snort/preproc_rules
sudo mkdir /usr/local/lib/snort_dynamicrules
sudo mkdir /etc/snort/so_rules

# Crear archivos que contendrán reglas y listas de IPs :
sudo touch /etc/snort/rules/iplists/black_list.rules
sudo touch /etc/snort/rules/iplists/white_list.rules
sudo touch /etc/snort/rules/local.rules
sudo touch /etc/snort/sid-msg.map

# Crear los directorios donde se almacenarán nuestros registros:
sudo mkdir /var/log/snort
sudo mkdir /var/log/snort/archived_logs

# Modificar los permisos de las carpetas:
sudo chmod -R 5775 /etc/snort
sudo chmod -R 5775 /var/log/snort
sudo chmod -R 5775 /var/log/snort/archived_logs
sudo chmod -R 5775 /etc/snort/so_rules
sudo chmod -R 5775 /usr/local/lib/snort_dynamicrules
```

A continuación, cambiaremos la propiedad de las carpetas creadas anteriormente para asegurarnos que Snort pueda acceder a los archivos:

```
sudo chown -R snort:snort /etc/snort
sudo chown -R snort:snort /var/log/snort
sudo chown -R snort:snort /usr/local/lib/snort_dynamicrules
```

Snort necesita sus archivos de configuración y sus preprocesadores dinámicos, así que los copiaremos dentro de las carpetas creadas anteriormente:

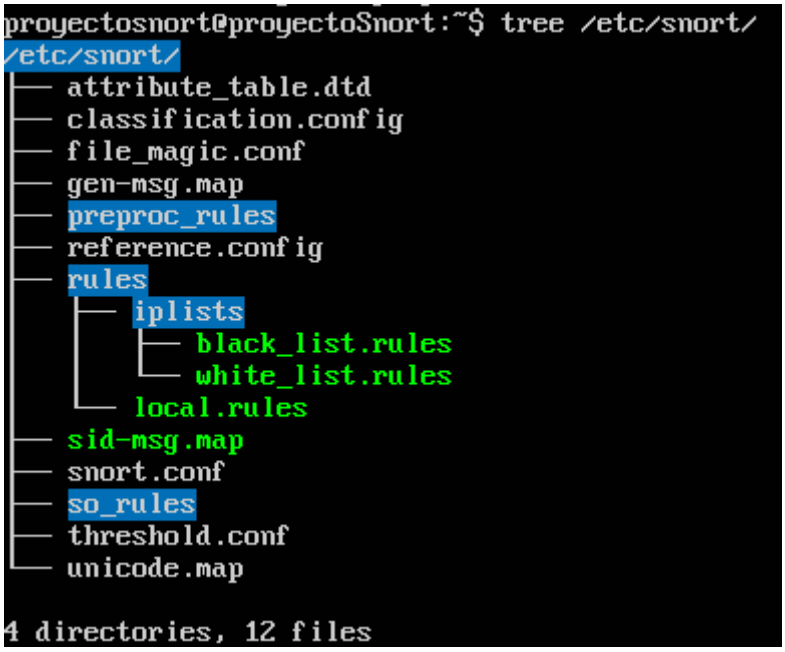
```
cd snort-2.9.8.3/etc/
sudo cp *.conf* /etc/snort
sudo cp *.map /etc/snort
sudo cp *.dtd /etc/snort

cd snort-2.9.8.3/src/dynamic-
preprocessors/build/usr/local/lib/snort_dynamicpreprocessor/
sudo cp * /usr/local/lib/snort_dynamicpreprocessor/
```

Tendremos la siguiente disposición de carpetas:

- Archivo binario Snort: */usr/local/bin/snort*
- Archivo de configuración Snort: */etc/snort/snort.conf*
- Directorio de registros Snort: */var/log/snort*
- Directorios de reglas Snort: */etc/snort/rules*
  /etc/snort/so rules
  /etc/snort/preproc rules
  /usr/local/lib/snort dynamicrules
- Directorio de listas IP Snort: */etc/snort/rules/iplists*
- Preprocesadores dinámicos Snort: */usr/local/lib/snort dynamicpreprocessor/*

Y nuestro directorio de Snort tendría la siguiente forma utilizando la herramienta *tree* (disponible desde el repositorio de Ubuntu *sudo apt-get install -y tree*):



Ahora necesitaremos editar el archivo de configuración principal de Snort, */etc/snort/snort.conf*. Cuando lancemos Snort con este archivo, estaremos lanzando Snort en modo NIDS.

Por el momento, y hasta que no se ponga en producción el servicio de Snort, comentaremos todas las reglas individuales que coge el archivo de configuración de Snort. Para ello, utilizaremos el siguiente comando que comentará todas las líneas de *snort.conf* que incluyan conjuntos de reglas:

```
sudo sed -i "s/include \$RULE\_PATH/#include \$RULE\_PATH/" /etc/snort/snort.conf
```

A continuación, modificaremos manualmente algunas configuraciones de *snort.conf*. Abrimos el archivo de configuración:

```
sudo vi /etc/snort/snort.conf
```

Y modificaremos algunas líneas para que Snort funcione a nuestro gusto:

- En la línea 45, modificaremos el valor de HOME_NET por el de la red que queremos proteger, en nuestro caso 192.168.56.0 con una máscara de 24 bits (255.255.255.0):

  NOTA: Podemos observar en qué red estamos conectados con el comando *ifconfig etho | grep "inet"*.

- A partir de la línea 104, modificaremos los valores con las carpetas que hemos creado previamente:

```
101 # Path to your rules files (this can be a relative path)
102 # Note for Windows users:  You are advised to make this an absolute path,
103 # such as:  c:\snort\rules
104 var RULE_PATH /etc/snort/rules
105 var SO_RULE_PATH /etc/snort/so_rules
106 var PREPROC_RULE_PATH /etc/snort/preproc_rules
107
108 # If you are using reputation preprocessor set these
109 # Currently there is a bug with relative paths, they are relative to where s
    nort is
110 # not relative to snort.conf like the above variables
111 # This is completely inconsistent with how other vars work, BUG 89986
112 # Set the absolute path appropriately
113 var WHITE_LIST_PATH /etc/snort/rules/iplists
114 var BLACK_LIST_PATH /etc/snort/rules/iplists_
```

- Por último, descomentaremos la línea 546 para habilitar nuestro archivo de reglas locales *local.rules*:

```
545 # site specific rules
546 include $RULE_PATH/local.rules
```

Una vez realizadas las configuraciones anteriores, tenemos que verificar que todos los archivos creados y modificados anteriormente son válidos y funcionan correctamente. Esto se puede probar con el siguiente comando:

```
sudo snort -T -c /etc/snort/snort.conf -i eth0
```

Usaremos la etiqueta -T para testear la configuración del archivo, -c para indicar el archivo de configuración de Snort y -i para indicar la interfaz de red por la que está escuchando Snort. Después de la ejecución del comando, debería salir algo como esto:

(...)

```
Snort successfully validated the configuration!
Snort exiting
proyectosnort@proyectoSnort:~$ _
```

11.1.7. Probando Snort

Ya tenemos Snort instalado y bien configurado. Ahora, para comprobar que funciona correctamente, crearemos una regla simple para que Snort cree una alerta cuando reciba un mensaje ICMP, el cual podemos generar haciéndonos un simple *ping* a nuestra máquina.

Dicho lo anterior, añadiremos la siguiente regla en nuestro archivo, por el momento vacío, de reglas locales *local.rules*:

```
alert icmp any any -> $HOME_NET any (msg:"Test para detectar si
alguien nos hace ping"; GID:1; sid:10000001; rev:001; classtype:icmp-
event;)
```

Ahora, cuando lancemos Snort, este cargará el archivo *local.rules* con la regla añadida y la usará en todo el tráfico que pase por la interfaz. Si el tráfico es muy elevado, algunos paquetes pueden no ser analizados. En este caso, Snort generará una alerta cuando vea un mensaje ICMP.

Podemos comprobar que la configuración de Snort es correcta tras los cambios realizados, con el comando anterior:

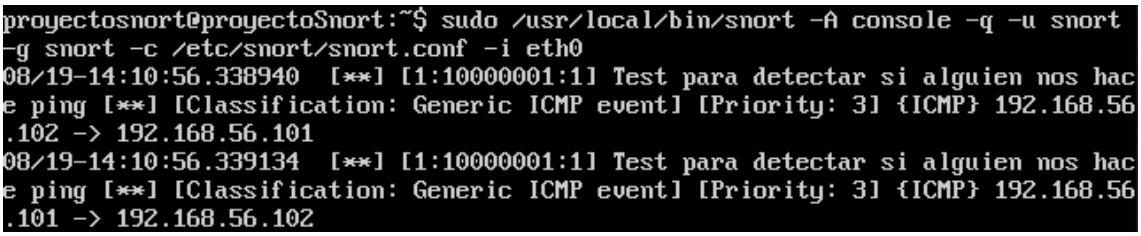
```
sudo snort -T -c /etc/snort/snort.conf -i eth0
```

Una vez comprobado que Snort carga correctamente la regla añadida, podemos lanzar Snort en modo NIDS y que las alertas generadas se muestren por consola. Ejecutaremos el siguiente comando:

```
sudo /usr/local/bin/snort -A console -q -u snort -g snort -c
/etc/snort/snort.conf -i eth0
```

-A console	Imprime las alertas generadas en la consola
-q	Modo silencioso
-u snort	Ejecuta Snort con el usuario snort
-g snort	Ejecuta Snort con el grupo snort
-c /etc/snort/snort.conf	Indica el archivo de configuración
-i eth0	Indica la interfaz de red por la que Snort analizará el tráfico

Una vez ejecutado el comando, lanzaremos un ping a nuestra máquina. Debería mostrarse algo similar a esto en la consola:



Donde podemos observar que la dirección IP 192.168.56.102 ha hecho un ping a nuestra máquina. Ya podemos decir que tenemos Snort correctamente configurado y funciona bien.

Otro aspecto importante es que Snort ha guardado un registro de esta alerta en */var/log/snort*, con nombre snort.log.xxxxxxxxx.

Para complementar nuestro IDS Snort, existen otras herramientas gratuitas que harán de nuestro IDS una herramienta más completa y fácil de usar, simplificando la labor del analista. Algunas de estas herramientas son:

- **Barnyard2**: para almacenar las alertas generadas por Snort en una base de datos.
- **BASE**: interfaz web la cual muestra las alertas de Snort almacenadas en la base de datos anterior.